VNIVERSITAT DE VALÈNCIA

Máster en Ingeniería Electrónica

VNIVERSITAT DE VALÈNCIA

Trabajo Fin de Máster

Desarrollo de un sistema de captación de imágenes y datos para dispositivos IoT

Autor: Rubén Garrido Alonso

Tutores: Vicent Girbés Juan y Valero Laparra Pérez-Muelas

Julio 2023

VNIVERSITAT DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria ETSE-UV

Trabajo Fin de Máster

Desarrollo de un sistema de captación de imágenes y datos para dispositivos IoT

Autor: Rubén Garrido Alonso

Tutores: Vicent Girbés Juan y Valero Laparra Pérez-Muelas

Declaración de autoría:

Yo, Rubén Garrido Alonso, declaro la autoría del Trabajo Fin de Máster titulado "Desarrollo de un sistema de captación de imágenes y datos para dispositivos IoT" y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual. El material no original que figura en este trabajo ha sido atribuido a sus legítimos autores.

Valencia, 7 de julio de 2023

Fdo: Rubén Garrido Alonso

Resumen:

El proyecto que se describe a lo largo de esta memoria se centra en la investigación, desarrollo e implementación de un sistema de captura de imágenes utilizando el microcontrolador ESP32. El objetivo principal es diseñar una solución que permita capturar imágenes utilizando una cámara controlada por un microcontrolador que tenga un consumo bajo de recursos y enviarlas a un servidor remoto para su posterior procesado con un modelo de inteligencia artificial capaz de detectar objetos a partir de una imagen.
Para lograr este objetivo, se ha utilizado el microcontrolador ESP32, configurado para conectarse a una red Wi-Fi y establecer una comunicación con un servidor, el cual se ha diseñado para poder ser compatible con diferentes plataformas.
El sistema se ha probado y validado, obteniendo resultados satisfactorios en cuanto a la captura y envío de imágenes. Se ha logrado establecer una conexión estable con el servidor remoto y transmitir las imágenes capturadas de manera eficiente.

Abstract:

The project described in this thesis focuses on the research, development and implementation of an image capture system using the ESP32 microcontroller. The main goal is to design a solution that enables capturing images using a camera controlled by a resource-efficient microcontroller and sending them to a remote server for further processing with an artificial intelligence model which gives the object detection capability to the system.
To achieve this objective, the ESP32 microcontroller has been utilized, configured to connect to a Wi-Fi network and establish communication with a server designed to be cross-platform compatible.
The system has been tested and validated, obtaining satisfactory results in terms of image capture and transmission. A stable connection with the remote server has been established, and the captured images are efficiently transmitted.

Resum:

El projecte que es descriu en aquesta memòria es centra en la investigació, desenvolupament i implementació d'un sistema de captura d'imatges utilitzant el microcontrolador ESP32. d'objectiu principal és dissenyar una solució que done a l'usuari la capacitat de capturar imatges mitjançant l'ús dúna càmera controlada per un microcontrolador de baix consum per a posteriorment enviar-les a un servidor remot per al seu processament amb un model d'intel·ligència artificial capaç de detectar objectes.
Per a poder aconseguir aquest objectiu, s'ha utilitzat el microcontrolador ESP32, configurat per connectar-se a una xarxa Wi-Fi i establir comunicació amb un servidor, el qual s'ha dissenyat per a poder funcionar en diferents plataformes.
El sistema s'ha provat i validat, obtenint resultats satisfactoris en quant a la captura i enviament d'imatges. S'ha aconseguit establir una connexió estable amb el servidor remot i transmetre les imatges capturades de manera eficient.

Agradecimientos:

Quiero agradecer a mi familia así como a mis amigos que me han apoyado a lo largo del desarrollo de este proyecto. También a mis tutores, quienes me han ayudado con gran afán durante todo el proceso de desarrollo del mismo.

Índice general

Capítulo 1

Introducción

1.1. Introducción

Cada vez es mayor el peso que la tecnología está teniendo en el día a día de la sociedad. La interacción con dispositivos tecnológicos está convirtiéndose poco a poco en algo cotidiano. Son cada vez más las tareas que pueden ser suplidas por dispositivos electrónicos que ayudan a que tareas del día a día, incluso labores más peligrosas puedan desarrollarse de una forma segura y eficiente.

Uno de los sectores en auge en esta última década es el de los sistemas embarcados y la capacidad de procesamiento de estos mismos. Es cada vez mayor la capacidad de dotar de inteligencia a dispositivos con dimensiones mas reducidas pero con una presente rendimiento que crece de forma exponencial.

El proyecto se encarga de dotar a determinados dispositivos de la capacidad de comunicación con un servidor central, utilizado para procesar una determinada información que cada uno de los dispositivo decida enviar, generalmente, imágenes capturadas mediante dispositivos de visión. En dicho servidor central se cuenta con un modelo de inteligencia artificial capaz de detectar objetos a partir de una imagen dada. Para poder conseguir esto, se ha tenido que poner a punto microcontrolador capaz de poder capturar imágenes a través de una cámara. Posteriormente, en un servidor central, dicha información será atendida y procesada para extraer la información necesaria.

1.2. Objetivos

El objetivo principal de este proyecto es establecer una comunicación bidireccional efectiva entre el microcontrolador ESP32 y el servidor remoto, permitiendo la captura y transferencia de imágenes de manera confiable. Además, se busca demostrar la viabilidad de utilizar el ESP32 como una solución integral para aplicaciones de captura de imágenes y comunicación en tiempo real.

A lo largo de la implementación del proyecto, se tendrán en cuenta aspectos como la eficiencia energética, la estabilidad y la escalabilidad del sistema. También se explorarán posibles mejoras y consideraciones adicionales, como el procesamiento de imágenes en el servidor y la integración con técnicas de inteligencia artificial para análisis de imágenes.

En resumen, este proyecto tiene como objetivo desarrollar un sistema basado en el

microcontrolador ESP32 para la captura y transferencia de imágenes a través de HTTP. Se espera que este sistema pueda aplicarse en diferentes escenarios, como vigilancia, automatización del hogar, control de accesos, entre otros, brindando una solución confiable y versátil para la captura y transmisión de imágenes en tiempo real.

Es importante destacar que las tecnologías utilizadas para el desarrollo del proyecto pueden variar en función de las necesidades y limitaciones identificadas a lo largo del proceso.

Capítulo 2

Estado del arte

Es cada vez mayor el peso que la tecnología está teniendo en el día a día de la sociedad. La interacción con dispositivos tecnológicos está convirtiéndose poco a poco en algo cotidiano. Son cada vez más las tareas que pueden ser suplidas por dispositivos electrónicos que ayudan a que tareas del día a día, incluso labores más peligrosas puedan desarrollarse de una forma segura y eficiente.

Uno de los sectores en auge en esta última década es el sector de los sistemas embarcados. La capacidad de procesamiento de estos y permite dotar de inteligencia a dispositivos cada vez con dimensiones mas reducidas pero con una capacidad de rendimiento que crece de forma exponencial.

El proyecto que se describirá a lo largo de la memoria se encarga de dotar a un microcontrolador de la capacidad de capturar imágenes bajo la demanda de un determinado usuario. Para poder conseguir esto, se ha tenido que poner a punto un microcontrolador junto a una cámara para poder realizar dicha captura.

Además, se ha tenido que dotar al dispositivo de la capacidad de conectarse a la red de forma inalámbrica, para que de esta forma sea operativo desde cualquier lugar.

Por otro lado, se ha levantado un servidor capaz de albergar todas las funcionalidades y peticiones que realiza el microcontrolador, cerrando de esta forma el círculo de funcionalidades.

2.1. Entorno Linux

Existen diferentes recursos que se pueden utilizar a la hora de poder levantar un servidor HTTP, así como diferente hardware en el que poder hacerlo. En este proyecto, la intención es dotar al sistema de la mayor flexibilidad posible y por ese motivo la plataforma elegida ha sido un entorno Linux.

Linux es conocido como un sistema operativo basado en código abierto y gratuito, dicho OS se sostiene sobre un kernel, donde reside la verdadera funcionalidad de Linux, por tanto, una mejor descripción sería decir que, Linux es un kernel sobre el cual se basa un sistema operativo. Fue creado por Linus Torvalds en 1991 como un proyecto personal y se ha convertido en uno de los sistemas más populares y ampliamente utilizados en el mundo.

Gran parte de la popularidad y del auge en el uso de Linux, es su naturaleza de código

abierto, más comúnmente conocido como open source, lo que significa que el código fuente del kernel está disponible para que cualquier persona lo pueda ver, modificar y distribuir. Esto fomenta que la comunidad de Linux sea completamente abierta, por lo que favorece la mejora de su funcionamiento así como el añadido continuo de nuevas funcionalidades que poco a poco abarcan más campos de desarrollo.

Otra de las razones por las cuales Linux es tan utilizado en la industria, reside en, la capacidad del adaptación del kernel de Linux a cualquier arquitectura hardware del dispositivo que se quiera poner en marcha para un proyecto. De hecho, es muy común utilizar hardware ligeramente diferente en las etapas de prototipado y en la puesta en marcha en producción. Por lo cual, se puede utilizar una misma versión de kernel para diferentes arquitecturas y el código que se escriba no difiere de un hardware a otro, o si lo hace, son cambios superficiales. Esto ocasiona que podamos utilizar la misma versión del Kernel de Linux tanto en el PC en el que estemos desarrollando como en el hardware que pondremos en funcionamiento.

2.2. Estudio previo

El proyecto se apoya en el microcontrolador ESP32, el protocolo HTTP/HTTPS, el servidor en Python con Flask y una variedad de bibliotecas adicionales para ofrecer una solución completa y eficiente. Estas tecnologías permiten la captura de imágenes, la comunicación segura con el servidor y el procesamiento de los datos recibidos. La elección de estas tecnologías se basa en su adecuación al propósito del proyecto, su capacidad de cumplir con los requisitos de rendimiento y seguridad, así como su popularidad y soporte en la comunidad de desarrollo.

2.2.1. Microcontrolador ESP32

El microcontrolador ESP32, desarrollado por Espressif Systems, es el componente central de este proyecto. Este dispositivo destaca por su alto rendimiento, bajo consumo de energía y amplias capacidades de conectividad. Cuenta con un procesador de doble núcleo, conectividad Wi-Fi y Bluetooth, así como interfaces GPIO para interactuar con diversos periféricos. El ESP32 ofrece una plataforma sólida y versátil para implementar aplicaciones IoT y sistemas embebidos.

Espressif Systems tiene una amplia gama de microcontroladores tal y como se muestra en la figura 2.1, dentro los microcontroladores disponibles, hay diferentes criterios a tener en cuenta que pueden influir en la elección. Aquí se muestra una pequeña descripción de los criterios que se deben considerar.

Potencia de procesamiento

Dentro de la gama de microcontroladores de Espressif, concretamente ESP32, existen diferentes velocidades de reloj y configuraciones de núcleos. Es relativamente complejo evaluar la potencia que va a consumir nuestro programa antes siquiera de escribir el código que va a ejecutar dicha aplicación. Aun así, existen diferentes recursos en la red donde podemos consultar proyectos similares, para tener en cuenta el hardware que se ha utilizado en dicho proyecto.

Figura 2.1: Diferentes modelos de microcontrolador ESP32.

Memoria

Otro recurso que se debe de tener en cuenta cuando se está evaluando la tecnología a utilizar, es la capacidad de almacenamiento del dispositivo así como el tamaño de memoria RAM que se dispone en el dispositivo. Es importante plantear que se debe de almacenar en el microcontrolador, por ejemplo en el caso que se describe se debe de almacenar el binario generado a partir de código escrito además de credenciales Wifi, imágenes capturadas de la cámara de forma temporal, certificados de servidores, etc.

A parte de lo descrito en el párrafo anterior, se debe asegurar que haya suficiente memoria RAM para las operaciones y cálculos requeridos en el proyecto.

Periféricos y conectividad

No se ha de olvidar el nivel de conectividad que se le quiera dar al microcontrolador utilizado en el proyecto. En el caso de este proyecto en particular, se deben de considerar diferentes periféricos o diferentes canales de conectividad. A continuación se describen los canales periféricos que se han tenido en cuenta.

WiFi Una de las principales funcionalidades a nivel de conectividad debe de ser la conectividad WiFi, ya que el dispositivo va a funcionar el 100 % del tiempo de forma remota, por este motivo se necesita no perder la comunicación con el dispositivo en ningún momento, además de que se debe de poder establecer un canal de comunicación bidireccional.

Bluetooth Para poder realizar un proceso de vinculación a la red a través de Bluetooth y dotar al dispositivo de conectividad, se debe de prestar atención a las diferentes funcionalidades que el microcontrolador debe de proporcionar respecto al Bluetooth, así como el consumo que este periférico debe de tener durante el uso del mismo.

Sensórica Tal y como se describe en el proyecto, el sensor principal alrededor del cual gira el proyecto es la cámara. Por tanto, busca un microcontrolador que cuente con conectividad SSBC, que son las siglas de la interfaz de bus de cámara serial. Además de esta interfaz, hay que comprobar que el microcontrolador escogido, cuente con interfaces comunes para poder dotar de escalabilidad al proyecto, tales como UART, I2C, SPI, ADC, GPIO... De esta forma se garantiza que en un futuro el proyecto se pueda ampliar y se le puedan añadir los sensores adaptados a nuevas necesidades del proyecto.

Consumo de energía La eficiencia energética es otro de los factores a tener en cuenta a la hora de escoger un dispositivo. En este caso en concreto, es cierto que no es condición sine qua non que el dispositivo funcione conectado a batería, ya que puede ser que la aplicación de dicho proyecto, cuente con una fuente de energía que no se agote y no haya que recargar.

Compatibilidad y soporte Espressif cuenta con un SDK (software development kit) de código abierto o open source. Es por ello que, al poder compartir todo este código abiertamente, se forma una comunidad bastante amplia colaborando y manteniendo código, facilitando el desarrollo de aplicaciones con aportaciones de ejemplos, librerías... En el siguiente capítulo se explica la estructura de este SDK, todas las herramientas que ofrece y por qué se ha escogido esta plataforma de desarrollo.

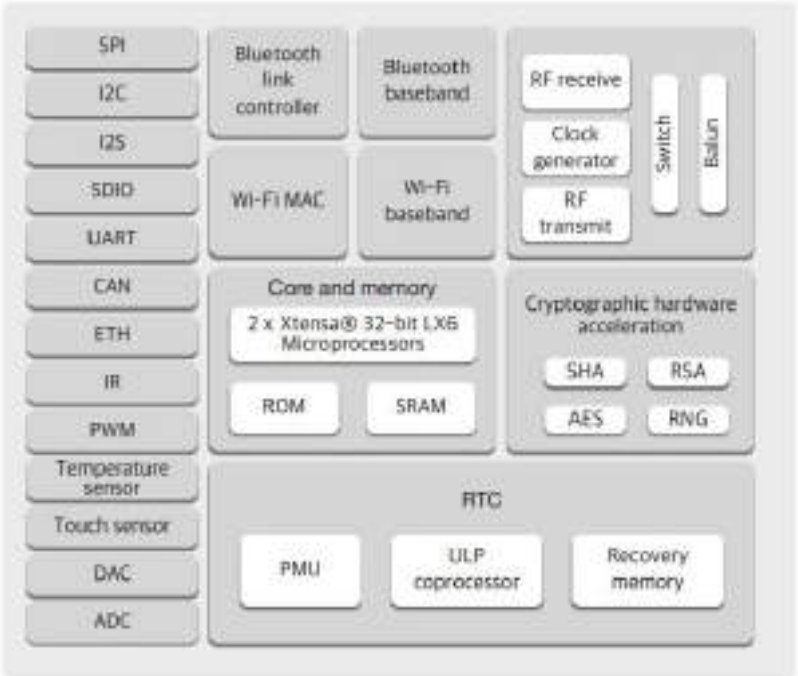


Figura 2.2: Diferentes funcionalidades del microcontrolador ESP32.

En la figura 2.2 se puede observar una muestra de las diferentes funcionalidades ofrecidas por el ESP32. Dicho esto, en el proyecto mencionado, no se ha estudiado en pro-

fundidad el consumo del microcontrolador, aunque sí se han tomado estimaciones que se describirán a lo largo de la memoria. A continuación se muestra la Tabla 2.1 con una comparativa entre los diferentes modelos de microcontrolador de ESP32 que podemos elegir.

Tabla 2.1: Comparación de características de la serie ESP32

Característica	**ESP32 Series**	**ESP32-S2 Series**
Launch year	2016	2020
Core	Xtensa dual-core 32-bit	Xtensa single-core 32-bit
Wi-Fi protocols	802.11 b/g/n, 2.4 GHz	802.11 b/g/n, 2.4 GHz
Bluetooth	Bluetooth v4.2 and BLE	-
Typical frequency	240 MHz	240 MHz
SRAM	520 KB	320 KB
ROM	448 KB	128 KB
Embedded flash	2 MB, 4 MB	2 MB, 4 MB
External flash	Up to 16 MB	Up to 1 GB
ADC	Two 12-bit, 18 channels	Two 13-bit, 20 channels
DAC	Two 8-bit channels	Two 8-bit channels
Temperature sensor	Not present	Present

2.2.2. Servidor HTTP Python

Para la comunicación entre el microcontrolador y el servidor, se ha utilizado el protocolo HTTP (Hypertext Transfer Protocol) y su versión segura HTTPS (HTTP Secure). Estos protocolos permiten la transferencia de datos entre el cliente (microcontrolador) y el servidor de forma segura y eficiente. El uso de HTTPS garantiza la encriptación de los datos transmitidos, protegiendo la confidencialidad y la integridad de la información.

Tabla 2.2: Comparación de servidor en Flask con otros lenguajes

Lenguaje	**Complejidad de programación**	**Consumo de recursos**
Python (Flask)	Baja	Moderado
Python (Django)	Baja a moderada	Moderado
Node.js	Baja a moderada	Moderado a alto
Java (Spring)	Moderada	Alto
Ruby (Ruby on Rails)	Moderada	Moderado a alto
C/C++ (mongoose-cpp)	Moderada a alta	Bajo

La Tabla 2.2 presenta una comparación de diferentes lenguajes y frameworks utilizados para implementar servidores. En base a esta tabla, se puede observar que el servidor en Flask, desarrollado en Python, tiene una baja complejidad de programación y un consumo de recursos moderado.

A continuación se analizan con mayor profundidad los diferentes frameworks que se han planteado en la tabla 2.2

Python (Flask) Flask destaca por su baja complejidad de programación y un consumo de recursos moderado. Es una excelente opción para proyectos que requieren simplicidad en el desarrollo y compatibilidad multiplataforma.

Python (Django) Aunque Python con Django tiene una complejidad de programación baja a moderada, el consumo de recursos es moderado. Es una buena opción si se necesita una base de datos NoSQL como MongoDB junto con Python.

Node.js Node.js tiene una complejidad de programación baja a moderada y un consumo de recursos moderado a alto. Es conocido por su enfoque en eventos y su alta escalabilidad. Es una buena opción para aplicaciones en tiempo real y de alta concurrencia.

Java (Spring) Java con el framework Spring tiene una complejidad de programación moderada y un consumo de recursos alto. Spring es ampliamente utilizado en aplicaciones empresariales debido a su robustez y capacidad de gestión de transacciones.

Ruby (Ruby on Rails) Ruby on Rails tiene una complejidad de programación moderada y un consumo de recursos moderado a alto. Es conocido por su enfoque en la convención sobre la configuración y la rapidez en el desarrollo.

C/C++ (mongoose-cpp) El uso de C/C++ con el framework mongoose-cpp tiene una complejidad de programación moderada a alta y un consumo de recursos bajo. Es una opción adecuada para aplicaciones que requieren un alto rendimiento y una gestión directa de los recursos del sistema.

La elección de Flask como servidor se basó en varios factores. En primer lugar, la baja complejidad de programación de Flask lo hace muy accesible para desarrolladores, incluso aquellos que no tienen una gran experiencia en el desarrollo de servidores. Flask utiliza una sintaxis sencilla y legible, lo que facilita la implementación de funcionalidades y la gestión de endpoints.

Además, Flask es compatible con múltiples plataformas, lo que permite que el servidor pueda ejecutarse en diversos sistemas operativos, como Linux, Windows y macOS. Esta compatibilidad multiplataforma es una ventaja importante, ya que proporciona flexibilidad y permite que el servidor se ejecute en diferentes entornos sin requerir cambios significativos en el código.

En términos de consumo de recursos, Flask tiene un nivel moderado, lo que significa que puede manejar adecuadamente las solicitudes y no requiere una gran cantidad de memoria o capacidad de procesamiento. Esto es beneficioso tanto para entornos de desarrollo con recursos limitados como para aplicaciones desplegadas en servidores de producción. En la figura 2.3 se puede observar una comparativa cerca del uso de ese Web framework.

Teniendo en cuenta estos factores, la elección de Flask como servidor fue respaldada por su simplicidad en programación y su compatibilidad multiplataforma. Esto permitirá un desarrollo eficiente y una fácil adaptación del servidor a diferentes sistemas operativos, lo que resulta en una solución flexible y de bajo consumo de recursos para el proyecto.

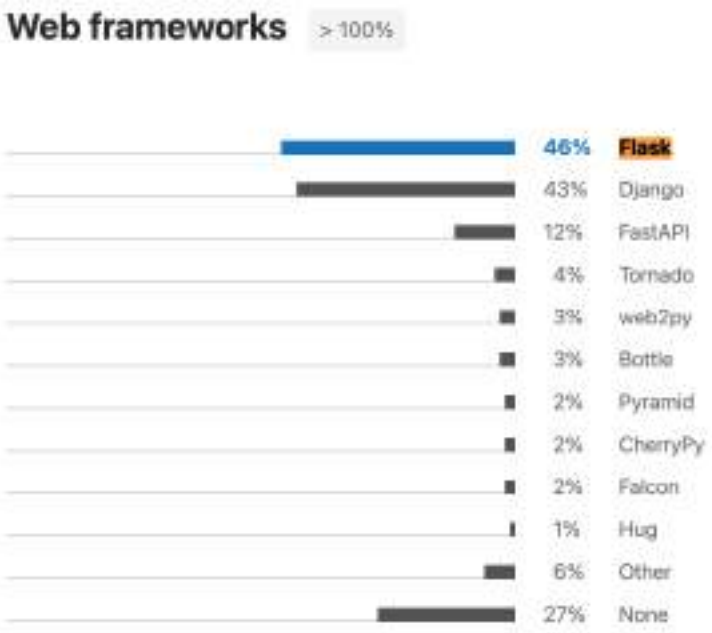


Figura 2.3: Comparativa Frameworks.

Capítulo 3

Arquitectura del microcontrolador

3.1. Estructura software

En este apartado, se abordará el proceso de desarrollo, así como el entorno software del microcontrolador ESP32, para ello, a lo largo del capítulo se describirá y mostrará cómo ha sido el proceso de desarrollo sobre un equipo Linux, mas concretamente basado en el sistema operativo Ubuntu y utilizando el SDK proporcionado por Espressif, conocido como ESP-IDF (Espressif IoT Development Framework). El cual, tal y como se ha mencionado con aterioridad, es de uso público y de código abierto, lo cual lo hace ideal para un entorno de desarrollo en materia de investigación y para un uso, no necesariamente comercial.

El desarrollo de aplicaciones para el ESP32 requiere de una serie de herramientas y recursos específicos que nos permiten aprovechar al máximo las capacidades y funcionalidades del microcontrolador. En este sentido, se describirán detalladamente los pasos necesarios para configurar el entorno de desarrollo en Ubuntu, incluyendo la instalación de las herramientas y la configuración de las variables de entorno correspondientes.

El SDK ESP-IDF es una potente y completa plataforma para el desarrollo de firmware para el ESP32. Proporciona una amplia variedad de bibliotecas llamadas componentes que facilitan la interacción con los periféricos del microcontrolador, así como la implementación de protocolos de comunicación como Wi-Fi, Bluetooth y muchos más protocolos y abstraccines de bajo nivel, como puede ser el sistema operativo FreeRTOS o bien librerías para el manejo de periféricos.

Durante el desarrollo, se mostrarán las principales características y funcionalidades del ESP-IDF, como la configuración de las diferentes librerías a través de su estructura de precompilación, el manejo de interrupciones, el acceso a la red Wi-Fi y la comunicación con otros dispositivos mediante protocolos como MQTT o HTTP el cual resultará especialmente útil a lo largo del desarrollo del proyecto. Además, se abordarán aspectos relacionados con la depuración y el monitoreo del sistema en tiempo real, siempre utilizando las herramientas proporcionadas por el vendedor.

La elección de Ubuntu como sistema operativo para el desarrollo se basa en su amplia compatibilidad con las herramientas y bibliotecas necesarias, así como en su facilidad de uso y su comunidad activa de desarrolladores, tal y como se menciona con anterioridad, linux ofrece una plataforma de código abierto mantenido principalmente por la comunidad, es por esto, que mas concretamente Ubuntu ofrece un entorno estable y confiable para llevar a cabo el desarrollo de software para el ESP32, permitiendo aprovechar al máximo

las capacidades de este potente microcontrolador.

En resumen, este apartado del trabajo se enfocará en describir el proceso de desarrollo de software para el ESP32 utilizando el sistema operativo Ubuntu y el SDK ESP-IDF. Se explorarán las herramientas, configuraciones y funcionalidades necesarias para aprovechar al máximo las capacidades del microcontrolador en el contexto de la implementación de un sistema completo.

3.1.1. Entorno de desarrollo ESP-IDF

El SDK de Espressif ESP-IDF (Espressif IoT Development Framework) es una plataforma de desarrollo para los microcontroladores ESP32 y ESP32-S2 entre otros. Proporciona un conjunto de herramientas, bibliotecas y ejemplos que facilitan la programación y la creación de aplicaciones para estos dispositivos, así como su mantenimiento y generación de estructura. En esta sección, exploraremos la estructura interna del SDK y las diferentes herramientas que ofrece.

Directorio principal del proyecto

Al crear un proyecto con ESP-IDF, se crea un directorio principal que contiene todos los archivos y carpetas relacionados con el proyecto, a este directorio principal, se le llama carpeta o directorio de la aplicación. Aquí se encuentran los archivos de configuración, el código fuente, las bibliotecas y otros recursos necesarios para construir la aplicación.

Carpeta components

Esta carpeta alberga los componentes reutilizables que se pueden utilizar en diferentes proyectos. Los componentes son módulos independientes que contienen código y archivos asociados, y se pueden agregar a un proyecto simplemente copiando la carpeta del componente en la carpeta çomponents"del proyecto.

Cada uno de los componentes, tiene unas directivas de compilación, así como unos ficheros fuente los cuales se estructuran en base a la configuración que se escoja de cada uno de los componentes.

Makefile y CMakeLists.txt

Estos archivos son utilizados por las herramientas de construcción para compilar y enlazar el código fuente del proyecto. El archivo Makefile es utilizado por el sistema de compilación basado en Make, mientras que el archivo CMakeLists.txt es utilizado por el sistema de compilación basado en CMake. Ambos archivos definen las dependencias, las opciones de compilación y otros parámetros necesarios para construir la aplicación.

Como se menciona con anterioridad, cada uno de los componentes contiene su CMakeLists.txt generando de esta forma un sistema anidado de directivas de compilación, facilitanto la estructuración modular del proyecto.

Herramientas de construcción

ESP-IDF proporciona un conjunto de herramientas de construcción que simplifican el proceso de compilación y carga de firmware en los microcontroladores ESP32, en la figura 3.1 se muestra gráficamente su arquitectura. Algunas de las herramientas más utilizadas son:

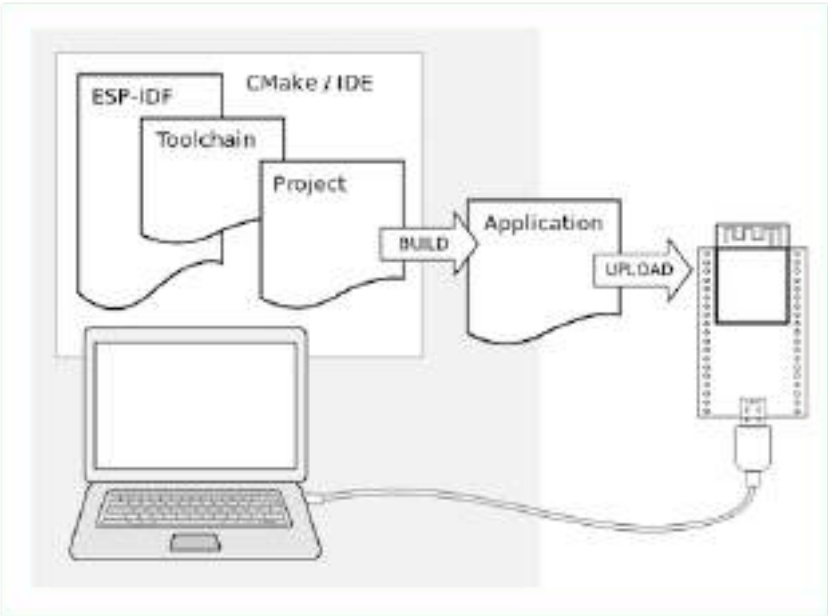


Figura 3.1: ESP-IDF arquitectura.

- **make**: Es una herramienta de construcción ampliamente utilizada en proyectos de software. Permite automatizar el proceso de compilación y enlace, utilizando reglas definidas en el archivo Makefile.
- **idf.py**: Es una interfaz de línea de comandos que proporciona comandos para compilar, cargar y depurar aplicaciones ESP-IDF. Permite configurar y gestionar el entorno de desarrollo de forma sencilla.
- **menuconfig**: Es una herramienta de configuración basada en texto que permite personalizar las opciones de configuración de la aplicación. Permite seleccionar características, habilitar o deshabilitar componentes y establecer parámetros específicos del hardware.

Toolchain y Proceso de Compilación

La toolchain utilizada en el proyecto de ESP-IDF es una colección de herramientas de compilación y enlace necesarias para compilar el código fuente y generar el firmware que se cargará en los microcontroladores ESP32. Esta toolchain incluye el compilador de C/C++, el enlazador y otras utilidades necesarias para el proceso de compilación.

El proceso de compilación en ESP-IDF se basa en el uso de scripts y herramientas proporcionadas por el SDK algunas de ellas gráficas, tal y como muestra la figura 3.2. A continuación, se describe brevemente cómo funciona el proceso de compilación:

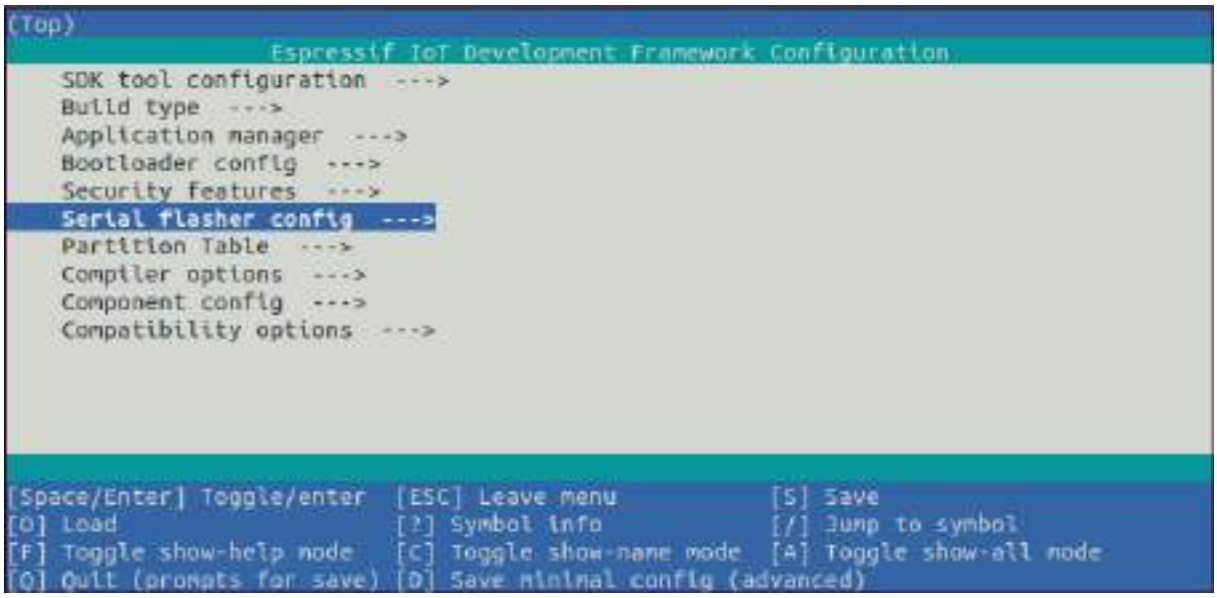


Figura 3.2: Configuración de proyecto.

1. **Configuración**: Antes de compilar el proyecto, es necesario configurar las opciones y las características específicas del hardware. Esto se realiza utilizando la herramienta `menuconfig` que proporciona ESP-IDF. Mediante esta herramienta, se pueden seleccionar las opciones de configuración, habilitar o deshabilitar componentes y establecer parámetros específicos del hardware.

2. **Makefiles y CMake**: ESP-IDF admite tanto el sistema de compilación basado en Make como el basado en CMake. El proyecto incluye un archivo Makefile o CMakeLists.txt que define las reglas de compilación y enlace, así como las dependencias del proyecto. Estos archivos especifican qué archivos y componentes se deben compilar y cómo se deben enlazar.

3. **Compilación**: Una vez configurado el proyecto y definidos los archivos y las dependencias, se puede iniciar el proceso de compilación. Utilizando la toolchain adecuada, el código fuente se compila en código binario para el microcontrolador ESP32. Durante la compilación, se aplican las opciones de optimización y se generan los archivos objeto correspondientes.

4. **Enlace o linkado**: Después de la compilación, los archivos objeto generados se enlazan para formar el firmware final. Durante el proceso de enlace, se resuelven las referencias a funciones y variables, se agregan las bibliotecas necesarias y se genera el archivo binario del firmware listo para ser cargado en el microcontrolador.

5. **Generación de firmware**: Una vez completados los pasos de compilación y enlace, se genera el archivo de firmware (generalmente con extensión .bin) que contiene el código ejecutable para el microcontrolador ESP32. Este archivo se puede cargar en el microcontrolador utilizando herramientas como `idf.py`.

   Dicho archivo binario, a parte de contener todo el código compilado, también debe de contener las instrucciónes de en que dirección de memoria se debe de situar cada uno de los fragmentos de dicho binario.

La toolchain del proyecto se encarga de administrar todo el proceso de compilación y enlace, asegurando que el código fuente se convierta en un firmware válido y listo para su

ejecución en los microcontroladores ESP32. A través de la configuración, los Makefiles o CMakeLists y las herramientas proporcionadas por ESP-IDF, los desarrolladores pueden compilar y generar el firmware de manera eficiente y confiable.

3.1.2. Control de versiones

Para trabajar con el control de versiones se debe de seguir una determinada metodología con el objetivo de llevar una desarrollo del proyecto de forma correcta. Generalmente a la hora de desarrollar un proyecto en GIT, se suele hacer de forma colaborativa, donde todo un equipo de desarrollo se verá involucrado en un determinado código. En este caso, solo uno o dos usuarios serán aquellos que hagan modificaciones en el código.

La metodología mas adecuada que se ha escogido para llevar a cabo el proyecto ha sido "Git Flow". Git Flow es una metodología de trabajo donde el repositorio que aloja el proyecto se estructura en diversas ramas o espacios donde se diferencian diversas partes del código y donde se estructura el código de forma que según la versión o funcionalidad se puede escoger de una forma sencilla. Generalmente la estructura de las diferentes ramas utilizadas es la que se muestra en la FIGURAX.

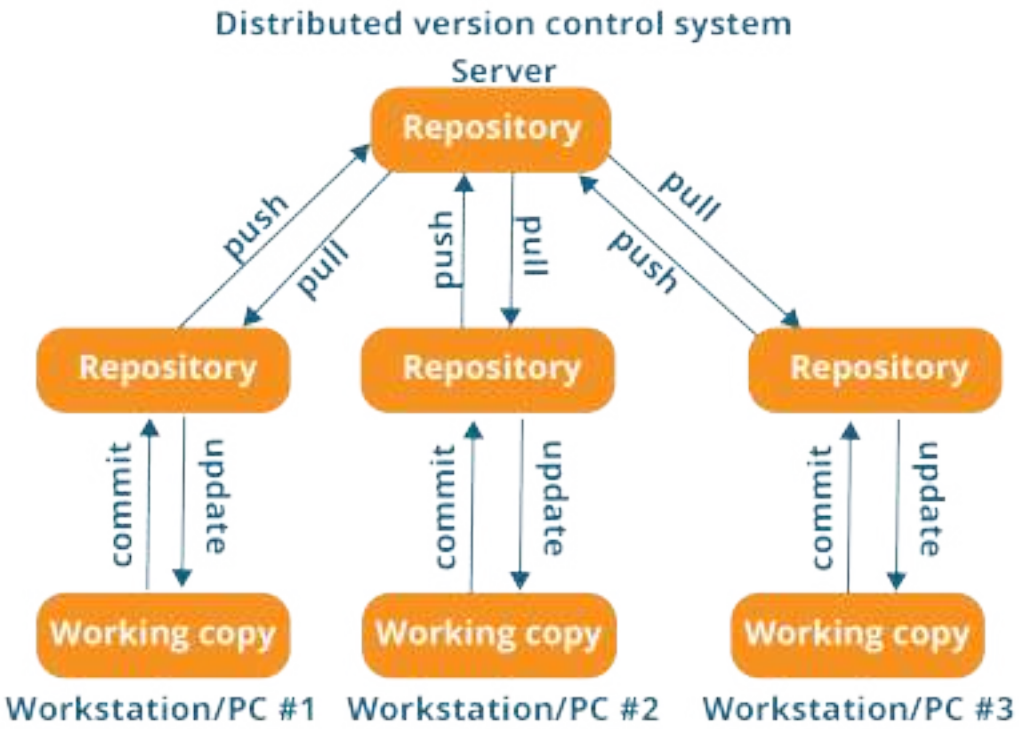


Figura 3.3: Metodología de trabajo GIT.

Cada una de las ramas alberga el código con las siguientes funcionalidades:

- Master : alberga el código que reside actualmente en producción, en este caso alberga el código que se ha utilizado en su versión completamente funcional mas reciente.
- Release : es el último código funcional que se ha enviado para evaluar su calidad y poder enviarlo a producción, también contiene una versión definitiva del código o proyecto a utilizar pero no necesariamente debe de ser el código utilizado en producción.

- Develop : alberga el código que está en actual desarrollo, no necesariamente debe de funcionar a la perfección, aunque es recomendable que todo el código que se pueda descargar de la rama de develop al menos debe de compilar y albergar funcionalidades básicas desarrolladas, aunque en estas se puedan encontrar diferentes bugs. Es la rama donde se encuentra el código antes de mandarlo a la rama de release.

- Feature : en la rama de feature se desarrollan las nuevas funcionalidades que se van añadir al código.

En el caso de este proyecto se ha desglosado en tres ramas, ya que el proyecto no se ha llevado a producción masiva, ha sido suficiente con tener tres ramas diferentes para estructurar de forma correcta el proyecto. En este caso se ha utilizado las ramas "develop"donde se ha ido guardando el código funcional en desarrollo, la rama "feature/"donde se iba desarrollando cada una de las funcionalidades antes de combinarlas con la rama "developz por último la rama master, donde se ha ido metiendo el código que era completamente funcional y definitivo. En la figura 3.3 s epuede observar el comportamiento de un control de versiones cuando diferentes desarrolladores hacen cambios sobre un mismo repositorio.

3.1.3. Contenedor Docker

En el contexto del proyecto descrito en la memoria, se ha decidido utilizar la tecnología de contenedores Docker para facilitar la compilación del proyecto de ESP32.

Docker es una plataforma de virtualización a nivel de sistema operativo que permite empaquetar una aplicación y todas sus dependencias en una unidad llamada c̈ontenedor.̈ Funciona especialmente bien sobre Linux aunque oficialmente tiene soporte multiplataforma, el funcionamiento en otros sistemas operativos esta algo mas limitado.

El uso de contenedores Docker para compilar proyectos de ESP32 ofrece varias ventajas significativas. En primer lugar, proporciona un entorno de compilación aislado y reproducible, lo cual es muy beneficioso para proyectos open source en los cuales se quiera compartir un entorno de desarrollo entre diferentes desarrolladores.

Esto significa que todas las dependencias y configuraciones necesarias para compilar el proyecto se encuentran dentro del contenedor, lo que garantiza que las compilaciones sean consistentes y evita problemas de compatibilidad con el entorno de desarrollo local.

Figura 3.4: Logotipo Docker.

Además, Docker simplifica la gestión de herramientas y dependencias necesarias para compilar proyectos de ESP32. Se puede crear una imagen Docker personalizada que contenga todas las herramientas requeridas, como el SDK de ESP-IDF y las bibliotecas necesarias.

A continuación, se puede ejecutar un contenedor basado en esa imagen para compilar el proyecto sin tener que instalar y configurar manualmente todas las herramientas en el entorno de desarrollo local.

Para utilizar Docker en el proyecto de ESP32, se deben seguir los siguientes pasos:

1. Instalar Docker en el sistema operativo siguiendo las instrucciones específicas para la distribución utilizada.
2. Crear un archivo llamado `Dockerfile` en el directorio raíz del proyecto de ESP32. Este archivo contendrá las instrucciones para construir la imagen Docker.
3. Dentro del `Dockerfile`, especificar una imagen base que contenga las herramientas y dependencias necesarias para compilar el proyecto de ESP32. Se pueden buscar imágenes base predefinidas en el Docker Hub o crear una imagen personalizada.
4. Definir los comandos necesarios dentro del `Dockerfile` para copiar el proyecto de ESP32 dentro del contenedor, instalar dependencias adicionales si las hay y compilar el proyecto utilizando el SDK de ESP-IDF.
5. Construir la imagen Docker ejecutando el comando `docker build` en el directorio donde se encuentra el `Dockerfile`.
6. Una vez que se haya construido correctamente la imagen, se puede ejecutar un contenedor basado en esa imagen utilizando el comando `docker run`. Esto permitirá compilar el proyecto de ESP32 dentro del contenedor, manteniendo un entorno de compilación aislado y reproducible.

El uso de contenedores Docker para compilar proyectos de ESP32 ofrece una forma conveniente y segura de gestionar las dependencias y los entornos de compilación, lo que facilita la colaboración y la reproducción de resultados en diferentes sistemas. Además, se puede integrar Docker en los flujos de trabajo de CI/CD para automatizar la compilación y el despliegue del proyecto.

3.2. Descripción arquitectura ESP32

En la era de la conectividad y el IoT, la capacidad de capturar y transferir información en tiempo real se ha convertido en un requisito fundamental para numerosos dispositivos así como aplicaciones, la necesidad de dispositivos que puedan capturar imágenes y enviarlas de manera eficiente se ha vuelto cada vez más importante.

En este contexto, el microcontrolador ESP32 se posiciona como una solución a dicho planteamiento bastante potente. Con su combinación de potencia de procesamiento, conectividad Wi-Fi y Bluetooth, y capacidad para interactuar con periféricos externos, el ESP32 es una de las opciones de referencia a la hora de elegir un dispositivo para desarrollar, tanto por su potencia de cómputo, su versatilidad y su fácil acceso a desarrolladores al tener una plataforma open source desarrollada por estos mismos.

En este trabajo, se hará énfasis en la utilización del microcontrolador ESP32 en conjunto con una cámara para capturar imágenes y enviarlas a través del protocolo HTTP. Esta funcionalidad permitirá explorar las capacidades del ESP32 en términos de captura de imágenes en tiempo real y comunicación con servidores remotos.

Tal y como se ha comantado con anterioridad, para lograr esto, se han aprovechador todas las herramientas y todas las capacidades del SDK open source que proporciona Espressif. Además, se han explorado las opciones disponibles en términos de periféricos y recursos del ESP32, así como las ventajas que ofrece en comparación con otros lenguajes y plataformas.

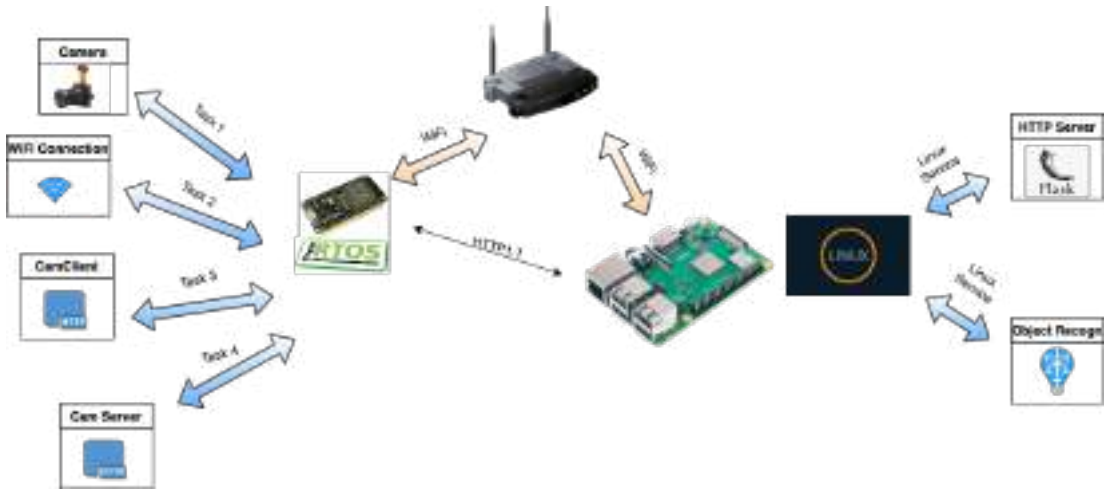


Figura 3.5: Diagrama funcionamiento principal.

Tal y como se observa en la figura 3.5, la estructura del proyecto a nivel de software se ha administrado en dos componentes principales. Por un lado el funcionamiento del servidor, el cual se ha escrito en Python, tal y como se describirá en el siguiente apartado.

Dicho servidor, es el encargado de gestionar las peticiones HTTP que realiza la cámara para poder posteriormente procesarlas con la inteligencia artificial, tal y como muestran los dos servicios a los que se hace referencia el el diagrama.

En cuanto al microcontrolador, se ha diseñado la aplicación software en diferentes tareas o hilos, los cuales son independientes entre si y se ejecutan en los diferentes núcleos del microcontrolador, gestionados por el sistema operativo. En la figura 3.6 se puede observar un esquema gráfico del funcionamiento.

Cada una de las tareas, utiliza un tamaño máximo de stack o de pila que se reserva en memoria, por lo cual, para poder optimizar al máximo la memoria y el funcionamiento del programa se ha ajustar para que cubriendo toda la casuística, se pueda ejecutar el programa sin rebasar el tamaño ni de stack ni de heap.

Las diferentes tareas que se han diseñado corresponden a los módulos de **Cámara, Network Connection, Camera Client, Camera Server**.

Para poder comunicar los diferentes hilos o tareas, se ha utilizado las colas de FreeRTOS que son capaces de poder enviar información de una hilo a otro.

Las colas en FreeRTOS permiten sincronización de manera eficiente así como de forma segura, ya que pretenden evitar condiciones de carrera. Siguen el principio FIFO (First-In-First-Out), lo que significa que el primer elemento en entrar en la cola será el primero en salir. En la figura 3.7 se observa un diagrama de funcionamiento.

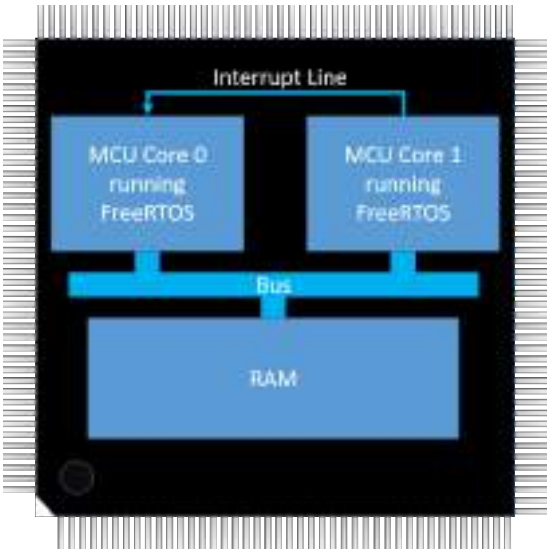


Figura 3.6: Gestión multicore en microcontrolador con RTOS.

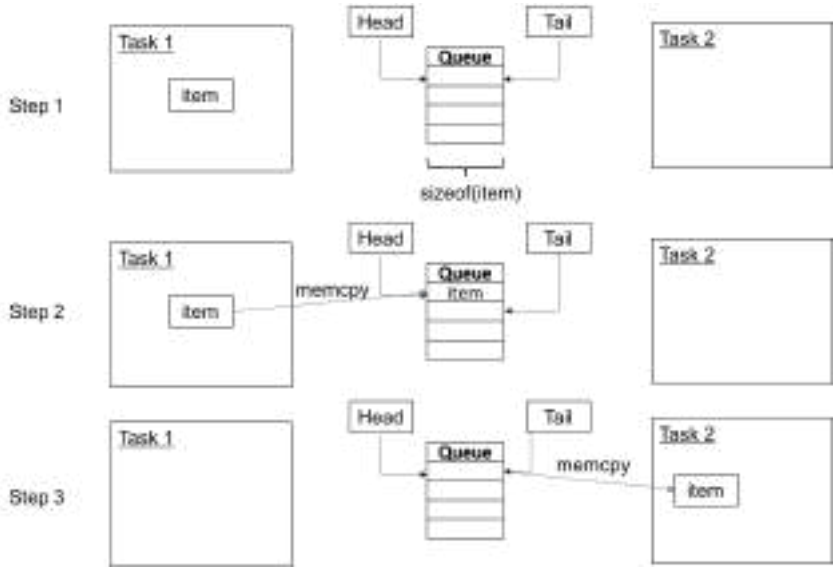


Figura 3.7: Comunicación interproceso a través de colas en FreeRTOS.

Las tareas pueden enviar elementos a la cola utilizando la función "xQueueSend()", la cual coloca el elemento al final de la cola, se le configura un tamaño máximo de mensaje así como un número máximo de mensajes que se pueden encolar. Por otro lado, las tareas pueden recibir elementos de la cola utilizando la función "xQueueReceive()", la cual extrae el primer elemento de la cola. Las colas de FreeRTOS son capaces de bloquear un hilo.

3.2.1. Sistema operativo FreeRTOS

Como la gran mayoría de las aplicaciones que obtienen una entidad relativamente grande y necesitan de una organización para gestionar diferentes hilos, se hace uso de un sistema operativo con el objetivo de administrar el sistema y poder adaptar el funcionamiento de los diferentes módulo o componentes.

Figura 3.8: Logo FreeRTOS.

En el desarrollo de aplicaciones para el microcontrolador ESP32, se utiliza FreeRTOS como sistema operativo en tiempo real. FreeRTOS es un kernel de programación en tiempo real de código abierto ampliamente utilizado en sistemas embebidos. Proporciona una planificación de tareas eficiente y recursos para el manejo de hilos, semáforos, colas y otros mecanismos de sincronización.

En ESP-IDF, el kit de desarrollo de software para ESP32, FreeRTOS está integrado y proporciona una capa de abstracción de hardware y funciones adicionales específicas de ESP32. Esto permite a los desarrolladores aprovechar las características y beneficios de FreeRTOS mientras trabajan con el microcontrolador ESP32.

El kernel FreeRTOS se ejecuta los diferentes núcleos del microcontrolador, aunque se puede configurar para que sea un único núcleo el que se encargue de correr el sistema operativo. Se encarga de administrar la planificación de tareas y los recursos del sistema. Permite la ejecución simultánea de múltiples tareas, cada una con su propio contexto y prioridad. El planificador también conocido como scheduler, decide qué tarea se ejecutará en un determinado momento y garantiza que se realice un cambio de contexto adecuado cuando sea necesario.

Figura 3.9: FreeRTOS ejemplo aplicación.

ESP-IDF, al tener un integración y soporte oficial de FreeRTOS, proporcionado por la empresa Amazon, ofrece una serie de componentes y bibliotecas que ya de por si mismos utilizan el sistema operativo FreeRTOS. Estos componentes incluyen la gestión del Wi-Fi, Bluetooth, TCP/IP, la gestión de memoria, el sistema de archivos, entre otros. Cada

componente se implementa como una tarea o un conjunto de tareas que se ejecutan en paralelo dentro del sistema FreeRTOS. Esto permite una programación estructurada y modular, así como una gestión eficiente de los recursos. En la figura 3.9 se muestra un ejemplo gráfico.

Además, ESP-IDF proporciona API y funciones específicas para trabajar con FreeRTOS en el contexto de ESP32. Esto incluye funciones para la creación y gestión de tareas, semáforos, colas y otros objetos de sincronización. También ofrece herramientas de depuración y monitoreo para ayudar a los desarrolladores a comprender el comportamiento del sistema en tiempo real. Todas estas herramientas resultan de bastante utilidad a la hora de desarrollar, ya que facilitan mucho las tareas de depuración. También resultan útiles en un entorno de producción donde se quiere monitorear la carga del sistema en tiempo real. En la figura 3.10 se muestra un ejemplo de funcionamiento.

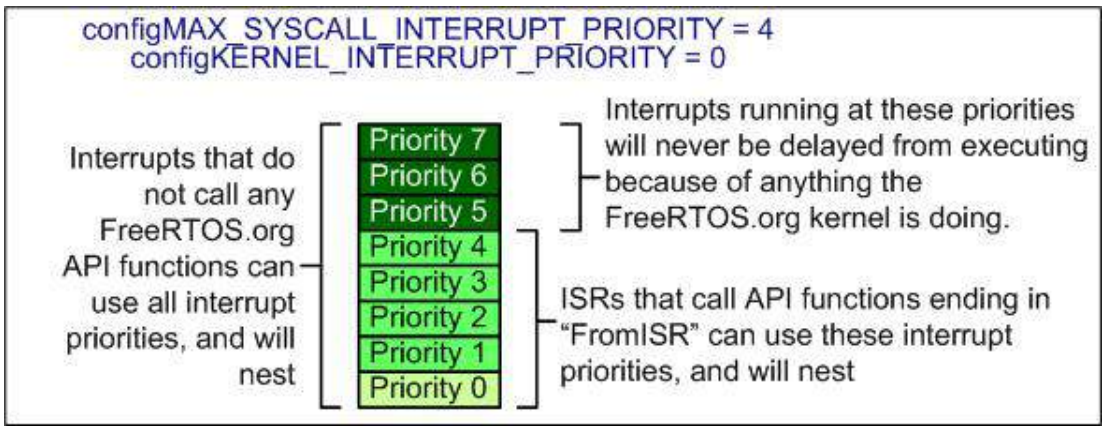


Figura 3.10: FreeRTOS ejemplo aplicación.

3.3. Protocolos utilizados

3.3.1. WiFi

En la actualidad, el Wi-Fi se ha convertido en una parte integral de nuestras vidas, permitiéndonos conectarnos a internet de forma inalámbrica y disfrutar de la comodidad de la conectividad en cualquier lugar. Ya sea en casa, en la oficina o en espacios públicos, el Wi-Fi nos brinda acceso a una amplia gama de servicios y aplicaciones.

El término "Wi-Fi"se refiere a una tecnología que utiliza ondas de radio para transmitir y recibir datos de manera inalámbrica. A diferencia de las conexiones por cable, el Wi-Fi elimina la necesidad de cables físicos, lo que nos brinda libertad de movimiento y flexibilidad para conectarnos a redes locales y a internet. En la figura 3.11 se muestra un esquema de funcionalidades wifi

El funcionamiento del Wi-Fi se basa en el estándar IEEE 802.11, que define las especificaciones técnicas para redes inalámbricas. Estas redes se componen de dos elementos principales: el punto de acceso (Access Point, AP) y los dispositivos clientes.

El punto de acceso actúa como un "hubçentral al que se conectan los dispositivos cliente. Es responsable de recibir los datos de los dispositivos y transmitirlos a través de la red cableada o inalámbrica. Los puntos de acceso suelen estar conectados a un

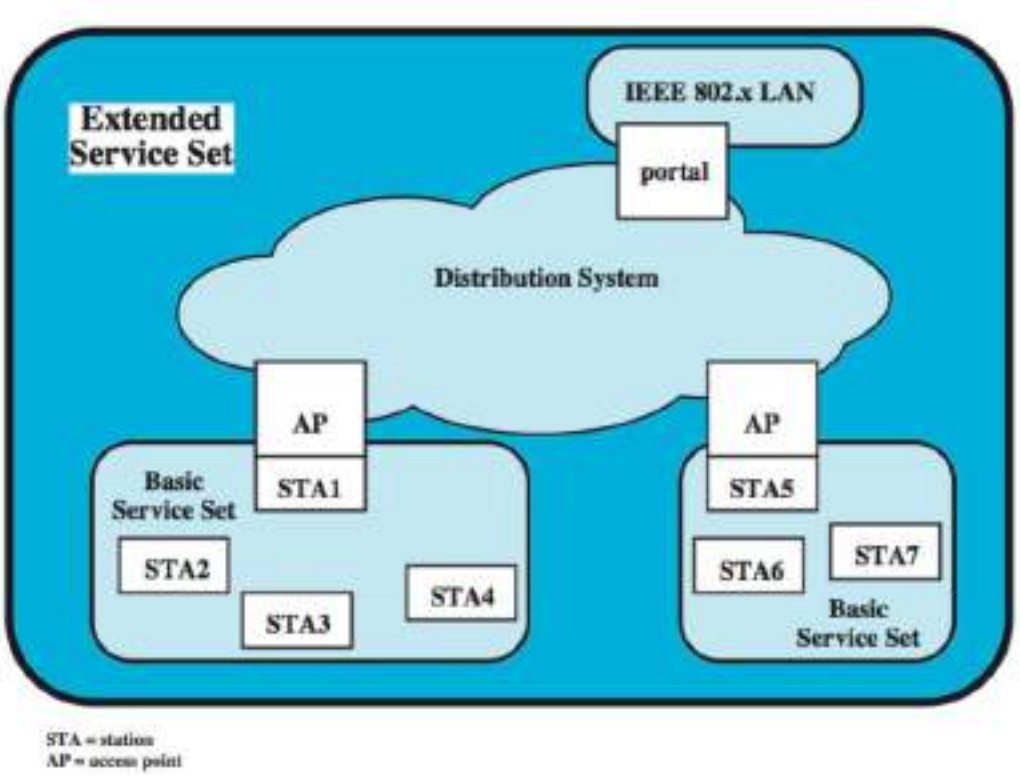


Figura 3.11: Arquitectura del estándar IEEE-802-11.

enrutador, que a su vez está conectado a internet, lo que permite a los dispositivos cliente acceder a la red global.

Por otro lado, los dispositivos cliente son aquellos que se conectan al punto de acceso para acceder a la red. Estos pueden ser ordenadores portátiles, teléfonos, tablets, cámaras, electrodomésticos inteligentes y una amplia variedad de dispositivos habilitados para Wi-Fi. Cada dispositivo que actúe como cliente debe estar equipado con una tarjeta de red inalámbrica compatible con Wi-Fi para poder establecer la conexión tal y como se muestra en la imagen 3.12.

El proceso de conexión Wi-Fi implica una serie de pasos. En primer lugar, el dispositivo cliente escanea las redes disponibles y muestra una lista de puntos de acceso detectados. Luego, el usuario selecciona la red a la que desea conectarse e ingresa una contraseña si es necesario. A continuación, el dispositivo cliente envía una solicitud de conexión al punto de acceso seleccionado, y una vez que se establece la conexión, se crea un enlace de comunicación bidireccional para transmitir y recibir datos.

El Wi-Fi utiliza diferentes frecuencias de radio para la transmisión de datos, principalmente en las bandas de 2.4 GHz y 5 GHz. La banda de 2.4 GHz ofrece una mayor cobertura, pero puede estar más congestionada debido a la presencia de otros dispositivos inalámbricos, como teléfonos inalámbricos y microondas. Por otro lado, la banda de 5 GHz proporciona velocidades más rápidas y menos interferencias, pero tiene un alcance más limitado. En la figura 3.13 se muestra una breve comparativa.

Además, el Wi-Fi utiliza técnicas de modulación y codificación para transmitir datos de manera eficiente. Estas técnicas permiten la transmisión de datos en forma de señales digitales a través de las ondas de radio, que son recibidas y decodificadas por los dispositivos cliente.

Figura 3.12: Estructura conexiones.

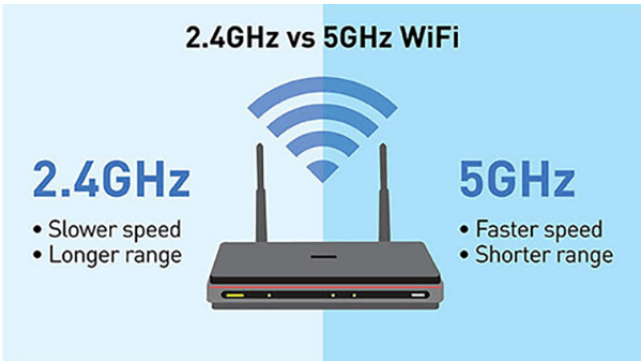


Figura 3.13: WiFi 5GHz y 5GHz.

El funcionamiento del Wi-Fi a nivel de hardware en un microcontrolador implica una serie de componentes y procesos esenciales que trabajan en conjunto para habilitar la comunicación inalámbrica. A continuación, se detallan los aspectos técnicos clave involucrados en este proceso:

El transceptor Wi-Fi es el componente central del hardware Wi-Fi en un microcontrolador. Este transceptor se encarga de la transmisión y recepción de las señales inalámbricas y opera en una de las bandas de frecuencia de 2.4 GHz o 5 GHz, según la tecnología Wi-Fi utilizada.

La antena es un elemento fundamental para la transmisión y recepción de las señales Wi-Fi. Convierte las señales eléctricas en ondas electromagnéticas y viceversa. Dependiendo del diseño del microcontrolador, la antena puede ser interna o externa.

El oscilador de cristal es otro componente crucial que genera la frecuencia necesaria para el funcionamiento del transceptor Wi-Fi. Este oscilador produce una señal de frecuencia precisa que se utiliza para sincronizar la transmisión y recepción de datos.

El amplificador de potencia es utilizado para aumentar la potencia de la señal Wi-Fi antes de ser transmitida por la antena. Este componente garantiza que la señal tenga suficiente potencia para alcanzar distancias adecuadas y superar posibles obstáculos.

Para filtrar y separar las señales de transmisión y recepción, se utilizan un filtro y un

duplexor. El filtro ayuda a eliminar el ruido o interferencias no deseadas en las señales, mientras que el duplexor permite la transmisión y recepción simultáneas a través de una misma antena.

El controlador de Wi-Fi es un componente encargado de gestionar y controlar las operaciones del hardware Wi-Fi. Este controlador se encarga de la configuración, el establecimiento de conexiones, la autenticación y el cifrado de datos para garantizar la seguridad y la integridad de la comunicación.

Existen dos principales modos de funcionamiento WiFi, dependiendo del propósito que se necesite en cada parte del proyecto, se utiliza un modo STA (modo estación) o modo AP (punto de acceso), cada uno de los modos tiene características y funcionalidades únicas. A modo de resumen, a continuación se enumeran las principales características.

WiFi STA (Station): El modo STA o modo estación, se utiliza cuando queremos que la interfaz de red wifi actúe como un cliente. En este modo, el dispositivo se conecta a un punto de acceso (AP) existente la cual porporcionará de conectividad y compartirá el trafico con la iterfaz STA. El dispositivo STA establece una conexión con el AP utilizando el protocolo de asociación y autenticación. Una vez conectado, el dispositivo STA puede enviar y recibir datos a través del AP para acceder a los recursos de red. En la figura 3.14 se muestra una representación gráfica de como sería el funcionamiento.

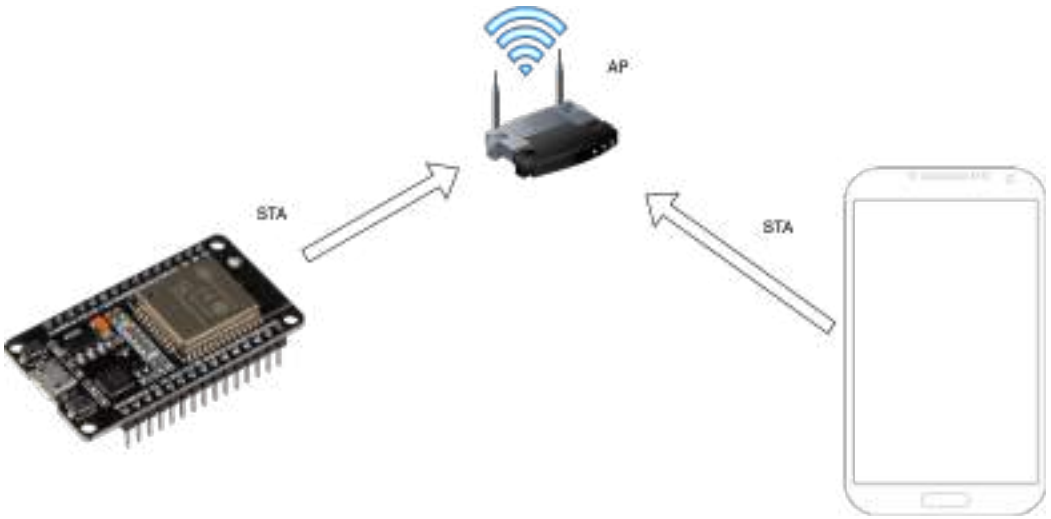


Figura 3.14: Funcionamiento STA.

En términos técnicos, el dispositivo STA utiliza una dirección MAC única para identificarse en la red.

WiFi AP (Access Point): El modo AP se utiliza cuando el dispositivo WiFi se configura como un punto de acceso. En este modo, el dispositivo es el encargado de proporcionar la información a los dispositivos que esten funcionando en modo estación y quieran realizar una conexión hacia el punto de acceso. Actúa como un enrutador a una red WiFi.

El dispositivo AP emite una señal WiFi y también tiene una dirección MAC única. El punto de acceso se identifica, entre otros parámetros, con un SSID (Service Set Identifier) que identifica a la red WiFi y puede estar protegido por una contraseña o clave de seguridad. El dispositivo AP asigna direcciones IP a los dispositivos conectados mediante el uso de un servidor DHCP y enruta los paquetes de datos entre los dispositivos conectados y otros dispositivos en la red. En la figura 3.15 se muestra una representación gráfica.

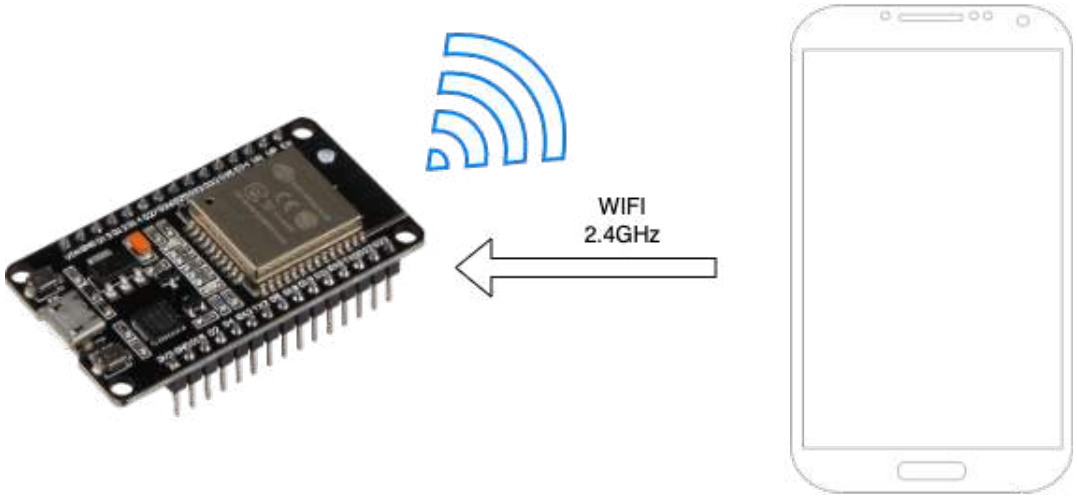


Figura 3.15: Funcionamiento AP.

Generalmente, una interfaz de red a nivel hardware puede funcionar en cualquiera de los dos modos, y es por tanto el software, quien se encarga de poner en funcionamiento dicha interfaz en el modo de funcionamiento adecuado para la aplicación en cada momento.

El uso y comunicación a nivel de WiFi en un microcontrolador, puede parecer una tarea compleja en caso de tener que comunicar directamente con el hardware para poder convertir el código programado en señales WiFI, afortunadamente, el entorno de desarrollo ESP-IDF proporciona una librería HAL (Hardware Abstractión Layer) la cual se ha diseñado para poder acceder a todas las funcionalidades WiFi sin necesidad des escribir código a muy bajo nivel, simplificando de esta forma, todas las tareas relacionadas con el WiFi.

```
esp_err_t esp_wifi_init(const wifi_init_config_t *config)
{
    if ((config->feature_caps & CONFIG_FEATURE_CACHE_TX_BUF_BIT) && (
        WIFI_CACHE_TX_BUFFER_NUM == 0))
    {
        ESP_LOGE(TAG, "Number of WiFi cache TX buffers should not equal 0
            when enable SPIRAM");
        return ESP_ERR_NOT_SUPPORTED;
    }
    esp_wifi_power_domain_on();
#ifdef CONFIG_PM_ENABLE
    if (s_wifi_modem_sleep_lock == NULL) {
        esp_err_t err = esp_pm_lock_create(ESP_PM_APB_FREQ_MAX, 0, "wifi",
                &s_wifi_modem_sleep_lock);
        if (err != ESP_OK) {
            return err;
        }
    }
#endif

#if CONFIG_ESP_WIFI_SLP_IRAM_OPT
    esp_pm_register_light_sleep_default_params_config_callback(
```

```
        esp_wifi_internal_update_light_sleep_default_params);

    int min_freq_mhz = esp_pm_impl_get_cpu_freq(PM_MODE_LIGHT_SLEEP);
    int max_freq_mhz = esp_pm_impl_get_cpu_freq(PM_MODE_CPU_MAX);
    esp_wifi_internal_update_light_sleep_default_params(min_freq_mhz,
        max_freq_mhz);

    uint32_t sleep_delay_us = CONFIG_ESP_WIFI_SLP_DEFAULT_MIN_ACTIVE_TIME
        * 1000;
    esp_wifi_set_sleep_delay_time(sleep_delay_us);

    uint32_t keep_alive_time_us =
        CONFIG_ESP_WIFI_SLP_DEFAULT_MAX_ACTIVE_TIME * 1000 * 1000;
    esp_wifi_set_keep_alive_time(keep_alive_time_us);
#endif

    return result;
}
```

En el frágmento de código anterior se muestra un fragmento del código implementado en la libreria HAL esp-wifi utilizada en este proyecto, la cual esta alojada en ESP-IDF.

Durante todo el proceso de desarrollo del proyecto, se ha tenido en cuenta la escalabilidad y la reproducibilidad del código escrito, en otras palabras, se intentan crear interfaces de código abstractas las cuales sean capaces de ser compartidas entre diferentes proyectos, ocultando las dependencias de estos fragmentos de código en otros ficheros.

Igual que en otros fragmentos de software, para el wifi, se ha generado una interfaz la cual se muestra en el siguiente fragmento de código.

```
typedef struct _wifistanetworkifaceConfiguration
{
    NetworkIfaceConfiguration base;
    uint8_t ssid[32];
    uint8_t password[64];
} WifiSTANetworkIfaceConfiguration;

typedef struct _wifiAPList
{
    uint8_t ssid[MAX_AP_SCAN_COUNT][33]; /**< SSID of AP */
    int8_t rssi[MAX_AP_SCAN_COUNT]; /**< signal strength of AP */
    int8_t apCount;
} WifiAPList;

typedef struct _wifistanetworkiface
{
    NetworkIface mInterface;
    wifi_sta_config_t mConfig;
    WifiAPList *mWifiScanParams;
    EventGroupHandle_t mWifiEventGroup;
```

```
    bool (*startWifiScan)();
    bool (*wifiScanListCallback)(WifiAPList *);
    SemaphoreHandle_t mMutex;
} WifiSTANetworkIface;

WifiSTANetworkIface *newWifiSTANetworkIface(char *interfaceName);
```

Tal y como se puede observar, a través de una interfaz abstracta se ha intentado abstraer la implementación específica de una librería como puede ser esp-wifi. La interfaz está compuesta por diferentes punteros a funciones, las cuales se asignan a las funciones propias de la librería proporcionada, tal y como se puede ver en el siguiente fragmento de código.

```
WifiSTANetworkIface *newWifiSTANetworkIface(char *interfaceName)
{
    WifiSTANetworkIface *this = (WifiSTANetworkIface *)calloc(1, sizeof(
        WifiSTANetworkIface));
    this->mInterface.observable = newObservable(this);
    this->mInterface.connect = wifiSTANetworkIfaceConnect;
    this->mInterface.disconnect = wifiSTANetworkIfaceDisconnect;
    this->mInterface.getStatus = wifiSTANetworkIfaceGetStatus;
    this->mInterface.setConfiguration =
        wifiSTANetworkIfaceSetConfiguration;
    strncpy(this->mInterface.interfaceName, interfaceName,
        NETW_IFACE_NAME_MAX_LENGTH);
    this->mInterface.status = NETW_IFACE_DISCONNECTED;
    this->mWifiScanParams = &wifiScanParams;
    this->startWifiScan = wifiScanForAPList;

    return this;
}
```

Un ejemplo por tanto de una de las funciones que se asignan a los punteros de la interfaz, puede ser por ejemplo la función connect, la cual se asigna a la dirección de memoria donde se aloja la función wifiSTANetworkIfaceConnect, por tanto al llamar al método **connect** de la estructura que forma la interfaz proporcionada, automáticamente se ejecuta el fragmento de código correspondiente a **wifiSTANetworkIfaceConnect**.

Para poder llevar a cabo todo el proceso de conexión a la red, se ha diseñado una infraestructura, la cual cuenta de diferentes estructuras o objetos, dependientes entre si, los cuales proporcionaran al dispositivo de la capacidad de gestionar el WiFi en cualquiera de sus módulos, así como de poder gestionar los eventos de conexión y desconexión en todo momento. Al igual que el resto de la estructura del proyecto, se han utilizado técnicas de programación orientada a objetos para gestionar las dependencias de todos los módulos entre si. En la figura 3.16 se representa un diagrama con las conexiones entre módulos.

Como se puede observar en la figura 3.16, la estructura planteada para realizar la conexión de Network cuenta con diferentes clases o diferentes objetos, los cuales son dependientes entre ellos e imprescindibles para el correcto funcionamiento del sistema.

Los diferentes objetos son los siguientes:

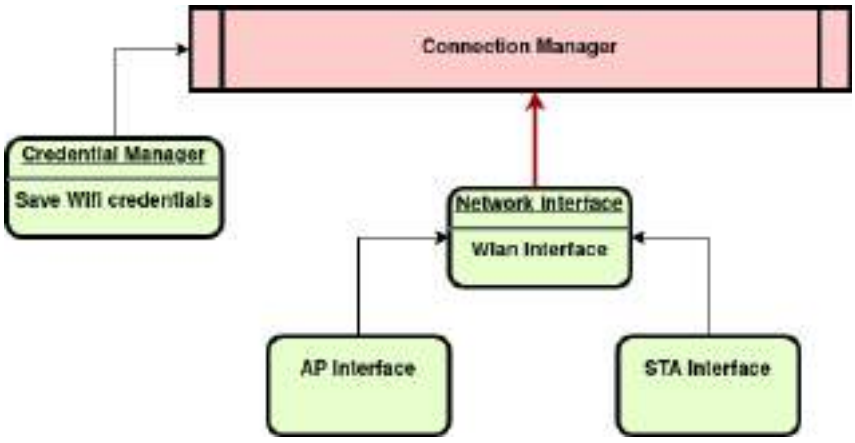


Figura 3.16: Diagrama objetos network

- **Connection Manager**: Este componente de es el encargado de gestionar todas las conexiones del sistema, mediante una máquina de estados, es el encargado de gestionar la interfaz WLAN del dispositivo. A este objeto se le inyectan las dependencias de los objetos Network Interface y Credential Manager que a continuación se describirá su funcionamiento.

  Este componente es el puente y por tanto la interfaz de comunicación del resto del programa. En todo momento debe de mantener informados al resto de los objetos del estado de conexión con la red, ya que hay muchas de las funcionalidades que carecen de sentido si no se tiene conexión a la red.

- **Network Interface**: El componente Network interface es una implementación abstracta de una interfaz de red. Tal y como se ha descrito con anterioridad, la interfaz de red es la encargada de comunicar el software programado con el hardware con el cuenta el microcontrolador. Dicha interfaz cuenta con dos modos de funcionamiento, los cuales se han descrito en dos objetos diferentes los cuales se describirán a continuación. El concepto de implementación abstracta, viene dado porque el componente simplemente implementa una estructura la cual debe de ser asignada con los punteros a función correspondientes. Los diferentes punteros a función que son asignados, provienen de los objetos WifiSTAInterface o bien WifiAPInterface.

- **AP Interface**: La interfaz AP es el objeto que implementa las funciones necesarias para que el objeto Network Interface funcione como interfaz de red en modo punto de acceso.

- **STA Interface**: La interfaz STA es el objeto que implementa las funciones necesarias para que el objeto Network Interface funcione como interfaz de red en modo estación.

3.3.2. Bluetooth

El Bluetooth es una tecnología de comunicación inalámbrica que nos permite conectar dispositivos entre sí para compartir datos y transmitir información de manera conveniente. Es ampliamente utilizado en una variedad de dispositivos, como teléfonos inteligentes, auriculares, altavoces, teclados, ratones y muchos otros dispositivos electrónicos.

El funcionamiento del Bluetooth se basa en el estándar Bluetooth SIG (Special Interest Group), que establece las especificaciones técnicas para la comunicación inalámbrica de corto alcance. A diferencia del Wi-Fi, que se utiliza principalmente para conectarse a redes y acceder a internet, el Bluetooth se centra en la comunicación directa entre dispositivos cercanos.

El proceso de conexión Bluetooth implica la identificación y emparejamiento de dispositivos. Cada dispositivo Bluetooth tiene una dirección única conocida como dirección MAC. Cuando dos dispositivos desean comunicarse, primero deben descubrirse entre sí y establecer una conexión. Esto se logra a través del proceso de emparejamiento, que generalmente implica que los dispositivos intercambien claves de seguridad o PIN para verificar la autenticidad y asegurar la privacidad de la comunicación. En la figura 3.17 se muestran las funcionalidades que se pueden adquirir al incorporar el bluetooth en un sistema.

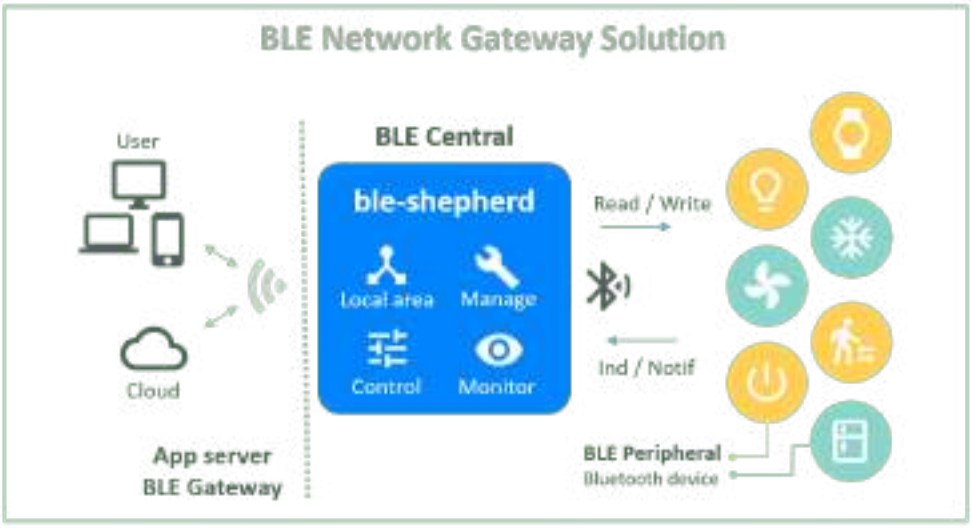


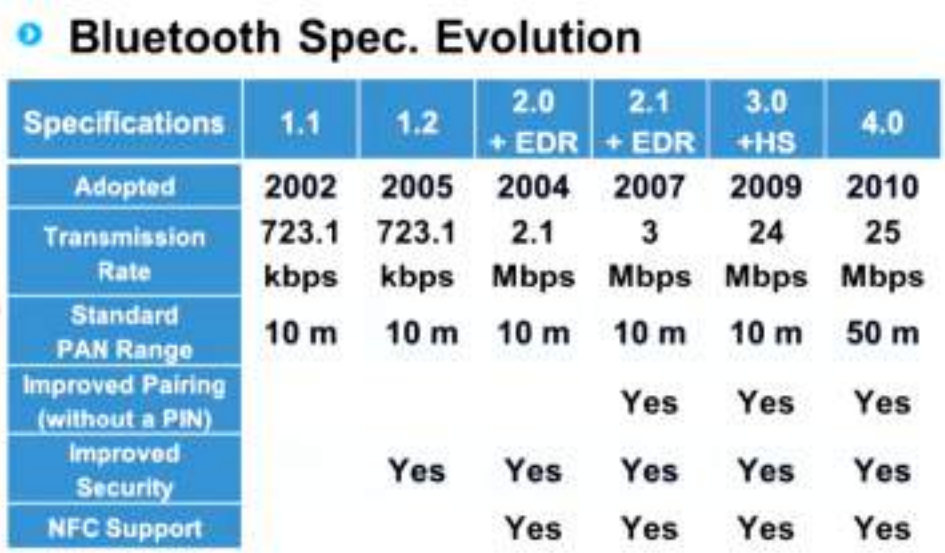
Figura 3.17: Bluetooth Low Energy.

Una vez emparejados, los dispositivos pueden establecer una conexión Bluetooth y comenzar a intercambiar datos. La comunicación Bluetooth se basa en la tecnología de radiofrecuencia y utiliza ondas de radio de corto alcance para transmitir datos. Esta tecnología permite la comunicación bidireccional, lo que significa que los dispositivos pueden enviar y recibir información entre sí.

El Bluetooth utiliza diferentes perfiles y protocolos para admitir una amplia gama de aplicaciones y servicios. Algunos perfiles comunes incluyen el perfil de manos libres (HFP) para llamadas telefónicas, el perfil de transferencia de archivos (FTP) para compartir archivos y el perfil de audio y video (A2DP) para transmitir música y video.

Además, el Bluetooth ha evolucionado a lo largo de los años para mejorar su rendimiento y eficiencia energética. Las versiones más recientes, como Bluetooth 5.0, ofrecen velocidades de transferencia más rápidas, mayor alcance y capacidad de conexión simultánea de varios dispositivos.

Algunas de las diferentes caracteristicas y comparativas se pueden ecntrar en la figura 3.18. En resumen, el Bluetooth es una tecnología inalámbrica que nos permite conectar dispositivos cercanos y compartir datos de manera conveniente. A través del empareja-



Bluetooth Spec. Evolution

Specifications	1.1	1.2	2.0 + EDR	2.1 + EDR	3.0 +HS	4.0
Adopted	2002	2005	2004	2007	2009	2010
Transmission Rate	723.1 kbps	723.1 kbps	2.1 Mbps	3 Mbps	24 Mbps	25 Mbps
Standard PAN Range	10 m	10 m	10 m	10 m	10 m	50 m
Improved Pairing (without a PIN)				Yes	Yes	Yes
Improved Security		Yes	Yes	Yes	Yes	Yes
NFC Support			Yes	Yes	Yes	Yes

Figura 3.18: Avances Bluetooth.

miento y la comunicación inalámbrica, el Bluetooth nos brinda la capacidad de interconectar nuestros dispositivos y disfrutar de una amplia gama de servicios y funciones.

El Bluetooth Low Energy (BLE), también conocido como Bluetooth Smart, es una variante del estándar Bluetooth diseñada para aplicaciones que requieren una baja potencia de transmisión y una larga duración de la batería. A diferencia del Bluetooth clásico, que se enfoca en transferir grandes cantidades de datos a velocidades más altas, BLE se optimiza para aplicaciones de baja potencia y bajo consumo de energía.

Una de las principales características de BLE es su eficiencia energética. Para lograr esto, se han realizado varias optimizaciones en comparación con el Bluetooth clásico. El tiempo de transmisión se ha reducido significativamente y se han implementado mecanismos de ahorro de energía, como el modo de suspensión y la tecnología de enlace de datos asimétrica. Estas optimizaciones permiten a los dispositivos BLE funcionar con baterías pequeñas durante largos períodos de tiempo, lo que es ideal para dispositivos portátiles y de baja potencia.

BLE se basa en un sistema de comunicación maestro-esclavo. Un dispositivo maestro, como un teléfono inteligente o una computadora, controla uno o más dispositivos esclavos, como sensores o dispositivos periféricos. El maestro inicia y controla las conexiones, mientras que los dispositivos esclavos están en modo de espera hasta que se les solicite una acción o transmisión de datos.

BLE utiliza una arquitectura de perfil y servicio para definir las funciones y características de un dispositivo. Los perfiles describen una aplicación o caso de uso específico, como el perfil de frecuencia cardíaca para monitores de ritmo cardíaco o el perfil de salud para dispositivos de seguimiento de la actividad física. Cada perfil consta de uno o más servicios, que a su vez contienen una o más características. Las características representan datos o acciones específicas que se pueden leer, escribir o notificar entre dispositivos BLE.

BLE también utiliza anuncios y exploraciones para descubrir y conectar dispositivos. Los dispositivos pueden transmitir anuncios que contienen información básica sobre ellos mismos, como su nombre o servicios ofrecidos. Otros dispositivos realizan exploraciones para detectar y recopilar información sobre los dispositivos cercanos. Una vez que se encuentra un dispositivo de interés, se puede establecer una conexión BLE entre los dis-

positivos maestro y esclavo.

En cuanto a la seguridad, BLE implementa medidas de protección para garantizar la privacidad y la integridad de los datos transmitidos. Utiliza encriptación de datos para evitar escuchas no autorizadas y autenticación de dispositivos para garantizar que solo los dispositivos confiables puedan establecer una conexión.

El Bluetooth Low Energy ha encontrado aplicaciones en una amplia gama de industrias, como la salud y el cuidado personal, la domótica, los dispositivos portátiles y la Internet de las cosas (IoT). Su eficiencia energética y su capacidad para transmitir datos de forma inalámbrica a distancias cortas lo convierten en una opción popular para dispositivos de bajo consumo de energía y conectividad inalámbrica de corto alcance.

En resumen, el Bluetooth Low Energy (BLE) es una variante del estándar Bluetooth diseñada para aplicaciones de baja potencia y bajo consumo de energía. Mediante optimizaciones y tecnologías de ahorro de energía, BLE permite a los dispositivos funcionar durante largos períodos de tiempo con baterías pequeñas. Con su arquitectura de perfil y servicio, BLE proporciona una forma eficiente de conectar dispositivos y transmitir datos de forma inalámbrica en aplicaciones diversas.

3.4. Implementación de la captura de las fotos

En este capítulo, se abordará el tema de la captura de imágenes mediante la cámara del kit ESP32-CAM mostrado en la figura 3.19. La cámara es un componente esencial en numerosas aplicaciones, desde sistemas de vigilancia hasta proyectos de visión artificial. En el contexto de este trabajo, la captura de imágenes es fundamental para la implementación de una funcionalidad específica que requiere la adquisición de imágenes en tiempo real.

Figura 3.19: ESP32S-CAM.

En este sentido, se explorará en detalle la cámara utilizada en el kit ESP32-CAM, la cámara OV2640. Se describirán sus características técnicas, su forma de funcionamiento y cómo se integra con el microcontrolador ESP32. Además, se analizarán las diferentes opciones de configuración y control que ofrece la cámara, así como las funcionalidades adicionales que pueden ser aprovechadas para mejorar la calidad de las imágenes capturadas.

Asimismo, se presentarán distintas estrategias y técnicas para la captura de imágenes, desde la configuración básica de la cámara hasta la optimización de parámetros avanzados para adaptarse a diferentes escenarios y condiciones de iluminación. Se proporcionarán ejemplos prácticos y código de muestra para facilitar la implementación y comprensión de los conceptos abordados.

Figura 3.20: Camara OV2640.

- **Sensor de imagen**: La cámara utiliza un sensor de imagen CMOS (Complementary Metal-Oxide-Semiconductor) OV2640 de 1/4 de pulgada. Este sensor tiene una resolución de 2 megapíxeles, lo que permite capturar imágenes con una resolución de hasta 1600x1200 píxeles. La cámara se muestra en la figura 3.20.
- **Tamaño del píxel**: El tamaño de cada píxel en el sensor es de 2.2 micrómetros, lo que proporciona una buena sensibilidad a la luz y la capacidad de capturar detalles finos en las imágenes.

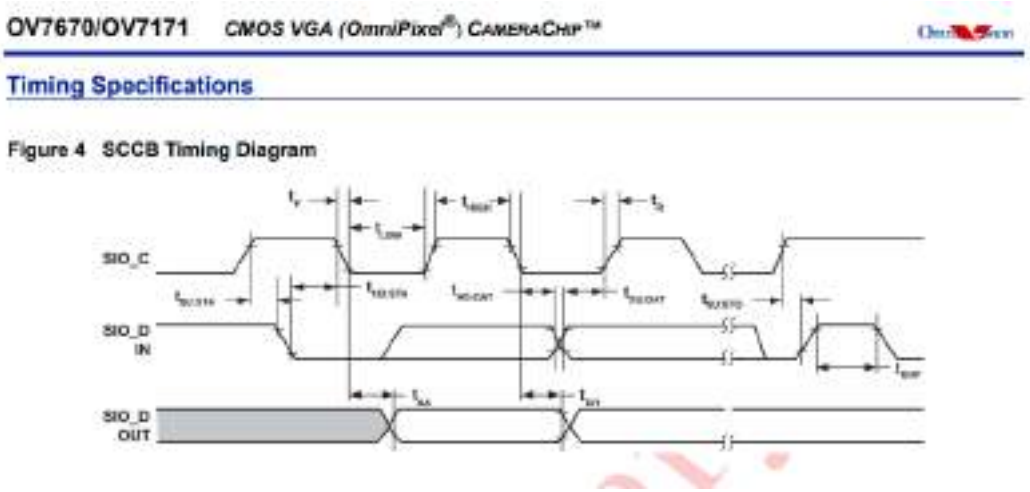


Figura 3.21: Protocolo SCCB.

- **Interfaz de comunicación**: La cámara se conecta al microcontrolador ESP32 a través de una interfaz Serial Camera Control Bus (SCCB). Esta interfaz permite la configuración y control de la cámara mediante comandos enviados a través de un bus de comunicación de dos hilos. En la figura 3.21 se muestra el funcionamiento del protocolo SCCB de forma gráfica.
- **Formato de salida**: La cámara puede proporcionar imágenes en varios formatos de salida, como JPEG, BMP, RGB565, YUV422 y otros. Esto permite una flexibilidad en la elección del formato de imagen según los requisitos del sistema.
- **Funciones adicionales**: La cámara OV2640 también ofrece algunas funciones adicionales, como el control de la exposición, balance de blancos automático, detección de rostros, control de saturación, brillo y contraste, entre otros. Estas funciones permiten ajustar los parámetros de captura de la imagen para obtener resultados óptimos en diferentes condiciones de iluminación y entornos.

3.4.1. Implementación Software

Siguiendo la tónica de todo el proyecto, se ha realizado una implementación modular basada en programación orientada a objetos, esto implica realizar una inyección de dependencias con los demás módulos del proyecto tal y como se ha visto en apartados anteriores.

En la implementación de la funcionalidad de captura de fotos, se ha adoptado un enfoque basado en estructuras abstractas en el lenguaje C. Estas estructuras abstractas proporcionan una forma flexible y extensible de encapsular la lógica de captura de fotos, permitiendo una fácil integración con otras partes del software. En la figura 3.22 se me muestra un esquema sobre la programación orientada a objetos.

Figura 3.22: Programación orientada a objetos.

Se han definido una serie de estructuras abstractas, cada una de las cuales representa una funcionalidad específica relacionada con la captura de fotos. Estas estructuras abstractas se han diseñado de manera modular, de modo que puedan ser fácilmente reemplazadas o extendidas sin afectar a otras partes del sistema.

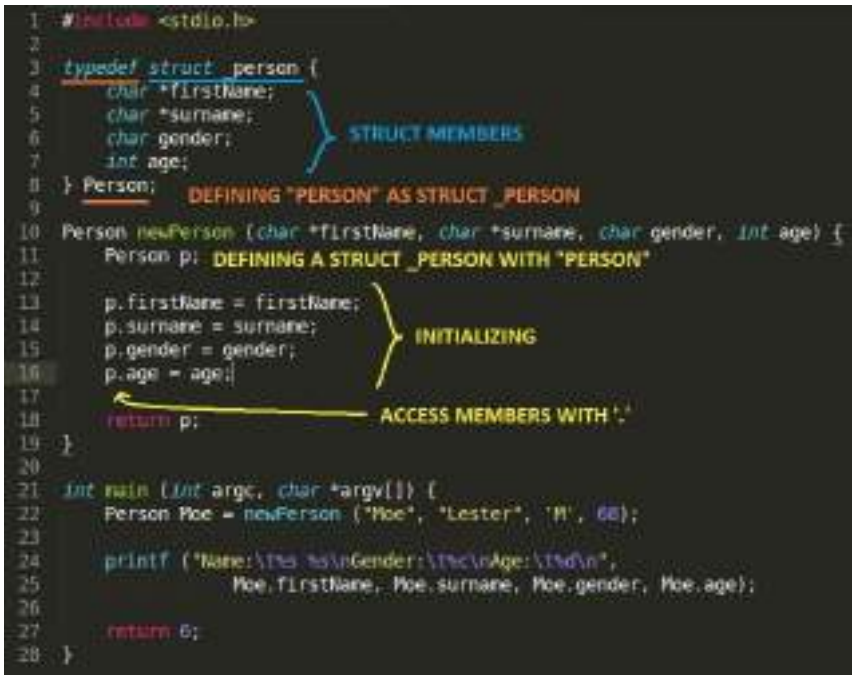


Figura 3.23: Ejemplo programación orientada a objetos.

Cada estructura abstracta contiene punteros a funciones que representan las diferentes operaciones relacionadas con la captura de fotos, como la inicialización, la captura de una imagen, el procesamiento de la imagen capturada, etc. Estos punteros a funciones permiten la inyección de dependencias, lo que significa que se puede proporcionar una implementación específica de cada operación mediante la asignación de punteros a funciones correspondientes. En la figura 3.23 se me muestra un ejemplo de implementación.

Una vez que se han definido las estructuras abstractas y se han cargado con punteros a funciones, se realiza la inyección de dependencias en las clases o módulos que requieren la funcionalidad de captura de fotos. Estas dependencias se utilizan para acceder a las operaciones de captura de fotos proporcionadas por las estructuras abstractas. De esta manera, se logra una separación clara entre las clases que utilizan la funcionalidad de captura de fotos y las implementaciones concretas de dicha funcionalidad.

Este enfoque basado en estructuras abstractas y la inyección de dependencias permiten una mayor modularidad y flexibilidad en la implementación de la captura de fotos. Se pueden agregar nuevas funcionalidades o modificar las existentes sin afectar el resto del sistema. Además, facilita la reutilización de código y promueve una mejor organización y mantenimiento del software.

En resumen, la implementación software de la captura de fotos se ha realizado mediante el uso de estructuras abstractas en C, que encapsulan la lógica y las operaciones relacionadas con la captura de fotos. Estas estructuras se cargan con punteros a funciones y se inyectan como dependencias en las clases que las necesitan, proporcionando así una solución modular y flexible para la funcionalidad de captura de fotos en el proyecto.

La primera de las estructuras y la mas importante, es la estructura *CameraInterface*, esta estructura contiene los punteros a función necesarios para poder tanto inicializar la cámara como para poder capturar imágenes. También contiene la API para poder cambiar la configuración de la cámara en tiempo de ejecución. La interfaz se puede observar en el siguiente fragmento de código:

```
typedef struct _cameraInterface
{
    camera_fb_t *(*pictureTake)();
    void (*pictureReturn)(camera_fb_t *);
    void (*set_pixformat)(struct _cameraInterface *, pixformat_t); //
       YUV422,GRAYSCALE,RGB565,JPEG
    void (*set_framesize)(struct _cameraInterface *,
                          framesize_t); // QQVGA-UXGA Do not use sizes above
                              QVGA when not JPEG
    void (*set_quality)(struct _cameraInterface *, int32_t); // 0-63 lower
        number means higher quality
    void (*set_brightness)(struct _cameraInterface *, int32_t); // -2 to 2
    void (*set_contrast)(struct _cameraInterface *, int32_t); // -2 to 2
    void (*set_saturation)(struct _cameraInterface *, int32_t); // -2 to 2
    void (*set_whitebal)(struct _cameraInterface *, int32_t); // 0 =
       disable , 1 = enable
    void (*set_awb_gain)(struct _cameraInterface *, int32_t); // 0 =
       disable , 1 = enable
    void (*set_wb_mode)(
        struct _cameraInterface *,
        int32_t); // 0 to 4 - if awb_gain enabled (0 - Auto, 1 - Sunny, 2
           - Cloudy, 3 - Office, 4 - Home)
    void (*cameraStop)(struct _cameraInterface *);
} CameraInterface;
```

El formato de inicialización se ha realiza a través de una estructura de configuración, la cual se ha de inicializar con los parámetros de configuración requeridos por la cámara, en el caso del proyecto aplicado y para un modelo de cámara OV2640 la estructura de configuración ha quedado de la siguiente forma:

```
static camera_config_t camera_config = {
    .pin_pwdn = CAM_PIN_PWDN,
    .pin_reset = CAM_PIN_RESET,
    .pin_xclk = CAM_PIN_XCLK,
    .pin_sscb_sda = CAM_PIN_SIOD,
    .pin_sscb_scl = CAM_PIN_SIOC,

    .pin_d7 = CAM_PIN_D7,
    .pin_d6 = CAM_PIN_D6,
    .pin_d5 = CAM_PIN_D5,
    .pin_d4 = CAM_PIN_D4,
    .pin_d3 = CAM_PIN_D3,
    .pin_d2 = CAM_PIN_D2,
    .pin_d1 = CAM_PIN_D1,
    .pin_d0 = CAM_PIN_D0,
    .pin_vsync = CAM_PIN_VSYNC,
    .pin_href = CAM_PIN_HREF,
```

```
    .pin_pclk = CAM_PIN_PCLK,
    .xclk_freq_hz = 20000000,
    .ledc_timer = LEDC_TIMER_0,
    .ledc_channel = LEDC_CHANNEL_0,
    .pixel_format = PIXFORMAT_JPEG,
    .frame_size = FRAMESIZE_VGA,
    .jpeg_quality = 20,
    .fb_count = 2,
    .grab_mode = CAMERA_GRAB_LATEST,
};
```

Como se puede apreciar, gran parte de la configuración del dispositivo está determinada por el hardware al que se conectará. En nuestro caso específico, se ha utilizado el protocolo SCCB para capturar imágenes del sensor.

En términos de resolución, hemos optado por VGA, ya que se ha considerado que es suficiente para detectar la mayoría de los modelos precompilados disponibles en la web. Esta resolución brinda un equilibrio adecuado entre calidad de imagen y eficiencia de procesamiento.

Tabla 3.1: Comparación de imágenes JPEG y RAW

JPEG	**RAW**
Tamaño de archivo más pequeño	Tamaño de archivo más grande
Compresión con pérdida de calidad	Sin compresión, calidad sin pérdida
Ampliamente utilizado en fotografía digital	Utilizado en fotografía profesional
Menor requerimiento de almacenamiento	Mayor requerimiento de almacenamiento
Procesamiento más rápido	Requiere procesamiento adicional

En cuanto al formato de la imagen, hemos seleccionado JPEG. La elección de la compresión JPEG se basa en la necesidad de reducir el tamaño del archivo para facilitar su transferencia al servidor y su almacenamiento en diferentes regiones de la memoria flash del microcontrolador. Al comprimir la imagen, se logra una mayor ligereza y eficiencia en el transporte de datos. En la tabla **??** se muestra una comparativa gráfica frente a una imagen RAW.

En resumen, la configuración del hardware, la elección de la resolución y el formato de imagen se han realizado considerando factores como la conectividad, la calidad visual y la eficiencia de almacenamiento. Estas decisiones nos permiten capturar imágenes de manera efectiva y prepararlas para su posterior procesamiento y transferencia.

El algoritmo de compresión JPEG se basa en la eliminación de información redundante y la optimización de la representación de los datos de la imagen. Esto se logra mediante la división de la imagen en bloques de píxeles y la aplicación de transformaciones matemáticas para reducir la cantidad de datos necesarios para representar la imagen. El algoritmo utiliza técnicas como la transformada de coseno discreta (DCT) y la cuantización para lograr una alta relación de compresión sin una pérdida significativa de calidad visual.

Cuando se captura una imagen, se realiza un proceso de muestreo para reducir la cantidad de información de color almacenada. El formato JPEG utiliza diferentes métodos de muestreo, como el submuestreo de color, que reduce la resolución de los componentes de color en comparación con los componentes de luminancia. Esto se basa en la percepción

visual humana, que es más sensible a los cambios en la luminancia que en el color. Al reducir la resolución del color, se logra una mayor compresión sin una pérdida visual perceptible.

Una vez que la imagen se ha comprimido en formato JPEG, se almacena en la memoria flash del microcontrolador. La compresión JPEG permite reducir el tamaño del archivo de imagen, lo que es beneficioso para el almacenamiento y la transmisión de imágenes a través de redes o dispositivos con recursos limitados. Además, el formato JPEG es ampliamente compatible y se puede visualizar en una amplia variedad de dispositivos y aplicaciones, lo que lo convierte en una opción versátil para la captura y visualización de imágenes.

3.5. Funcionamiento como cliente HTTP

En cuanto a la implementación del modo cliente HTTP en el microcontrolador ESP32, se ha desarrollado un sistema que permite el envío a un servidor de la última imagen capturada a través del protocolo HTTP. Esta funcionalidad se logra mediante la interacción de diferentes componentes y procesos en el microcontrolador.

El proceso comienza con la captura de la imagen utilizando la cámara del módulo ESP32. Una vez que se ha realizado la captura, la imagen se almacena en la memoria interna del microcontrolador. Por este motivo, es importante mantener un formato de compresión en todo momento, para que el almacenamiento de la imagen capturada sea lo mas liviano posible. A continuación, se inicia el proceso de envío de la imagen al servidor utilizando el cliente HTTP implementado. En la figura 3.24 se muestra una ejemplo gráfico del funcionamiento del protocolo http.

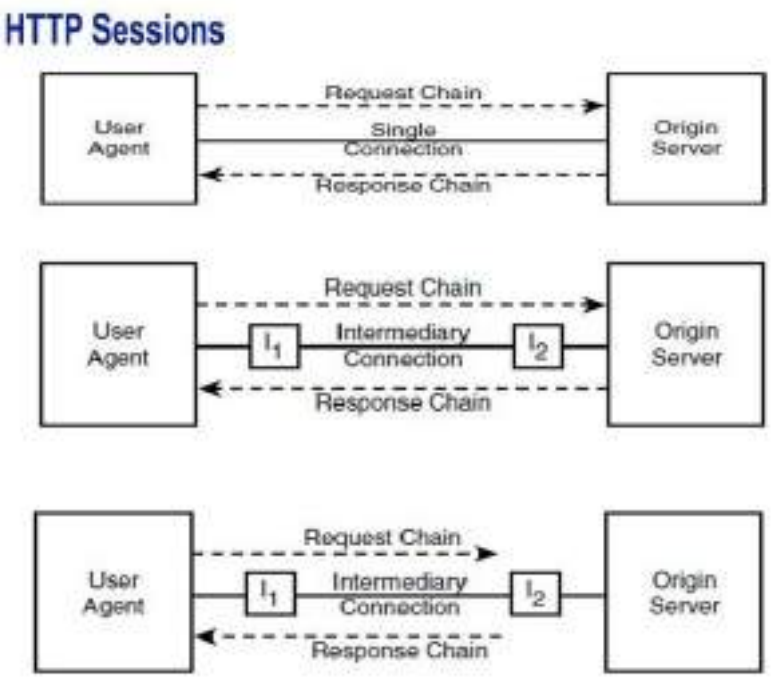


Figura 3.24: Protocolo http.

El cliente HTTP se encarga de establecer una conexión con el servidor remoto utilizando el protocolo HTTP. Esta conexión se establece a través de una dirección IP y un puerto específico. Una vez establecida la conexión, se envía una solicitud HTTP al servidor, que incluye la imagen capturada como parte de los datos de la solicitud. En la figura 3.25 se muestra el contenido de una petición http.

Para poder realizar la petición de forma satisfactoria, debe de coincidir el formato de la petición POST realizada del lado del cliente, con el formato de la petición que espera el endpoint específico en el lado del servidor.

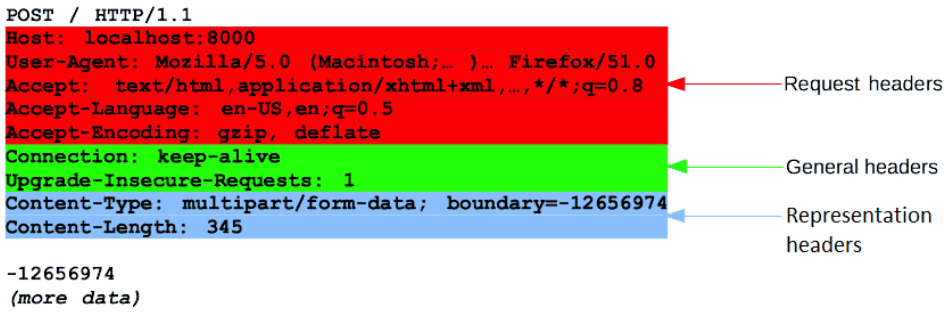


Figura 3.25: Contenido POST request http.

Para realizar correctamente la petición se han de configurar correctamente los encabezados así como el cuerpo o body de la petición. Los encabezados, son componentes de metadatos que se incluyen al hacer la solicitud HTTP. Proporcionan información sobre que tipo de contenido transportará la petición, para de esta manera que en la capa de transporte no se corrompa nada de información y pueda ser interpretada correctamente. Algunos ejemplos comunes de encabezados en una solicitud POST son los siguientes:

El encabezado *Content-Type* especifica el tipo de contenido del cuerpo o body que tendrá la petición. El encabezado Content/type es muy importante el cualquier petición POST, ya que tal y como se ha comentado anteriormente indica al lado del servidor cual va a ser el formato de los datos enviados y como se deben de interpretar. Por ejemplo, se utiliza el tipo de contenido application/json cuando se envía información en formato JSON.

Otro encabezado relevante es el *Content-Length*, ya que indica cual va a ser la longitud del cuerpo. El valor recibido son bytes y es esencial para saber cuanta memoria debe de ocupar el cuerpo de la solicitud y si se debe de fraccionar en diferentes partes para poder atenderla de forma correcta. En la figura 3.26 se muestra de forma gráfica la finalidad de las peticiones POST y GET.

Por otro lado, el body, o cuerpo de la petición, es el lugar de la petición donde se encuentran los datos que se han descrito en los encabezados y que van a llegar al servidor. En una solicitud POST, el body se utiliza para transmitir información estructurada o binaria, como formularios, archivos adjuntos o contenido JSON.

En el caso descrito en esta memoria, se envía una imagen JPEG en el cuerpo de una solicitud POST, se utiliza un encabezado multipart/form-data, para permitir el envío de archivos binarios, aunque también se ha podido realizar codificando la imagen JPEG en base64 para poder encapsularla en el cuerpo de la petición.

La imagen se transfiere al servidor utilizando técnicas de transferencia de datos en HTTP, como la codificación de datos en el cuerpo de la solicitud o el uso de métodos específicos, como POST. La imagen se divide en fragmentos o secciones más pequeñas,

Figura 3.26: Peticiones GET y POST.

dependiendo del tamaño de la imagen y las limitaciones del protocolo HTTP.

Una vez que se han enviado todos los fragmentos de la imagen al servidor, se completa la solicitud HTTP y se espera la respuesta del servidor. El servidor procesa los datos recibidos, que en este caso son la imagen capturada, y puede realizar acciones adicionales, como el almacenamiento de la imagen en una base de datos o su procesamiento posterior.

Es importante tener en cuenta que durante todo el proceso de envío de la imagen, se establece una comunicación bidireccional entre el módulo ESP32 y el servidor a través de la conexión HTTP. Esto permite la transferencia de datos de manera confiable y segura, y también proporciona la capacidad de recibir respuestas o notificaciones del servidor, como confirmaciones de recepción de la imagen o mensajes de error en caso de que se produzcan problemas durante la transferencia.

3.6. Funcionamiento como servidor HTTP

Tal y como se ha visto en apartados anteriores, el microcontrolador cuenta con la capacidad de establecer conexiones WiFi, por esta misma razón, igual que el dispositivo es capaz de actuar como cliente http y poder realizar peticiones a diferentes servidores, también es capaz de funcionar en modo servidor, y albergar la capacidad de recibir peticiones y generar respuestas.

Un servidor http o servidor web, puede albergar diferentes endpoints, un endpoint es una URL específica de un host que representa una funcionalidad específica que el servidor puede albergar en nada una de sus URLs. Cada endpoint está es capaz de realizar a una acción o conjunto de acciones definidas.

Utilizando lo endpoints podemos hacer que un mismo servidor sea capaz de poder albergar diferentes funcionalidades en cada una de las rutas establecidas. Cada endpoint tiene un nombre único el cual permite a cada cliente realizar solicitudes específicas. En el caso de este proyecto se han establecido diferentes endpoints para realizar dos funcionalidades principales, las cuales son: Captura y visualización de imágenes y streaming de vídeo en tiempo real. En la figura 3.27 se muestra una composición de endpoints de

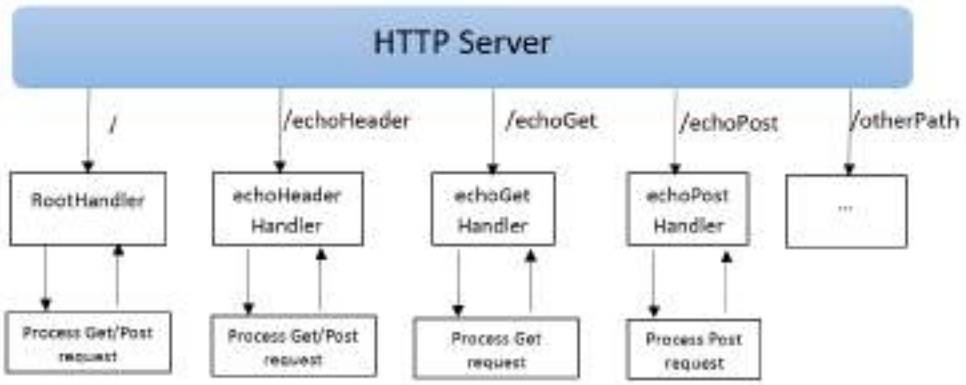


Figura 3.27: Servidor HTTP.

ejemplo.

Con el objetivo de poder poner a punto el servidor y hacerlo funcionar de la manera mas liviana y manejable posible, se ha escrito una interfaz, la cual utilizando la librería de código abierto proporcionada por el SDK de Espressif http-server. La interfaz se puede observar en el siguiente fragmento de código.

```
typedef struct _httpServerInterface
{
    bool (*registerNewUri)(struct _httpServerInterface *, const
       httpd_uri_t *);
    bool (*unregisterUri)(struct _httpServerInterface *, const char *);
    bool (*startServer)(struct _httpServerInterface *);
    bool (*stopServer)(struct _httpServerInterface *);
    void (*destroy)(struct _httpServerInterface *);
} HttpServerInterface;

typedef struct _HttpServerEsp
{
    HttpServerInterface mInterface;
    httpd_handle_t mServer;
} HttpServerEsp;
};
```

Tal y como se puede observar, a través de la interfaz, se pueden habilitar el servicio del servidor, deshabilitarlo así como registrar nuevos endpoints con los que añadir nuevas funcionalidades.

Los endopoints registrados han sido los siguientes:

- **streamHandler:** Proporciona la capacidad de conectarse con un navegador y poder visualizar a tiempo real las imágenes capturadas por la cámara.

```
esp_err_t streamHandler(void *args)
{
    httpd_req_t *req = (httpd_req_t *)args;

    esp_err_t res = ESP_OK;
    char *part_buf[128];

    res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);

    if(res != ESP_OK)
    {
        return res;
    }

    while(true)
    {
        picture = mCameraIface->pictureTake(mCameraIface);

        if(!picture)
        {
            ESP_LOGE(TAG, "Camera capture failed");
            res = ESP_FAIL;
        }

        size_t hlen = snprintf((char *)part_buf, 128, _STREAM_PART,
           picture->len);
        res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
        res = httpd_resp_send_chunk(req, (const char *)picture->buf,
           picture->len);
        res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(
           _STREAM_BOUNDARY));

        if(picture)
        {
            mCameraIface->pictureReturn(picture);
        }

        if(ESP_OK != res)
        {
            break;
        }
    }
    return res;
}
```

- **photoHandler:** Proporciona la capacidad de conectarse mediante un navegador poder capturar un imagen en un momento preciso a través de un botón en el navegador.

```
    httpd_req_t *req = (httpd_req_t *)args;

    httpd_resp_set_status(req, "307 Temporary Redirect");
    httpd_resp_set_type(req, "image/jpeg");

    ESP_LOGI(TAG, "picture take request fromweb server");

    if(NULL != picture)
    {
        mCameraIface->pictureReturn(picture);
    }

    picture = mCameraIface->pictureTake(mCameraIface);

    if(picture->buf == NULL)
    {
        ESP_LOGE(TAG, "Could not take picture");
        return ESP_ERR_NO_MEM;
    }

    httpd_resp_send_chunk(req, (const char *)picture->buf, picture->len);
    httpd_resp_sendstr_chunk(req, NULL);

    // CHECK_NULL_AND_FREE(imageToSend);

    return ESP_OK;
```

Utilizando esta configuración, se consigue poder gestionar diversas funcionalidades que dotan al usuario de poder almacenar imágenes bajo demanda, ya que con una simple petición, vía API Rest se obtiene una imagen capturada en tiempo real.

Capítulo 4

Servidor HTTP

4.1. Servidor HTTP externo

En este capítulo, se explorará la implementación y configuración del servidor Python que desempeña un papel central en el proyecto. Se abordarán varios aspectos fundamentales, desde la elección de la Raspberry Pi como plataforma para alojar el servidor hasta el procesamiento de imágenes mediante inteligencia artificial.

En primer lugar, se examinará la razón detrás de la elección de una Raspberry Pi como la base del servidor. Esta pequeña pero potente placa de desarrollo ofrece numerosas ventajas, como su versatilidad, bajo consumo de energía y escalabilidad. Exploraremos cómo la Raspberry Pi se convierte en el compañero ideal para alojar y gestionar el servidor Python, proporcionando un entorno flexible y de alto rendimiento.

Además, se profundizará en la configuración del servidor Python en la Raspberry Pi. Se explorarán los pasos necesarios para instalar y configurar adecuadamente el entorno de ejecución, incluyendo la instalación de bibliotecas y dependencias específicas. También se abordarán aspectos de seguridad y se discutirán las mejores prácticas para garantizar la protección y estabilidad del servidor.

Una vez que el servidor esté en funcionamiento, se analizará el proceso de procesamiento de imágenes a través de una IA en la Raspberry Pi. Se explicará cómo se integra la inteligencia artificial en el flujo de trabajo del servidor, permitiendo realizar tareas como reconocimiento de objetos, detección de rostros o cualquier otro tipo de análisis de imágenes. Se explorarán las bibliotecas y herramientas utilizadas para implementar la IA en la Raspberry Pi, y se discutirán las consideraciones y desafíos asociados con el procesamiento de imágenes en tiempo real.

4.1.1. Plataforma de desarrollo del servidor

Una de las ventajas destacadas de utilizar una Raspberry Pi como servidor para el proyecto radica en su capacidad para brindar un entorno de servidor altamente personalizable y adaptable. La Raspberry Pi, con su potente procesador y su capacidad de ejecutar sistemas operativos completos, se convierte en una elección ideal para alojar y gestionar un servidor web. Al utilizar la Raspberry Pi como servidor, se puede aprovechar su amplia gama de opciones de conectividad, incluyendo puertos Ethernet y Wi-Fi integrados, lo que facilita la comunicación con el módulo ESP32 y otros dispositivos conectados.

Otra ventaja significativa es el bajo consumo de energía de la Raspberry Pi. Este dispositivo ha sido diseñado para ser altamente eficiente en términos de energía, lo que implica un menor costo operativo y una menor huella ecológica en comparación con servidores convencionales. Además, su tamaño compacto y su diseño de bajo perfil permiten instalarlo fácilmente en diferentes entornos, ya sea en un laboratorio, un entorno de producción o incluso en un despliegue móvil. La Raspberry Pi también se destaca por su comunidad

Figura 4.1: Raspberry pi 3B.

activa y su amplia disponibilidad de software y herramientas. Con una gran cantidad de desarrolladores y entusiastas trabajando en proyectos relacionados con la Raspberry Pi, se puede acceder a una amplia variedad de recursos, tutoriales y documentación. Esto proporciona un sólido respaldo y soporte para la implementación del servidor, así como la capacidad de aprovechar soluciones preexistentes o adaptarlas según las necesidades específicas del proyecto. En la figura 4.1 se observa el dispositivo físico utilizado.

Además, la Raspberry Pi permite la escalabilidad del servidor. Su capacidad de procesamiento y almacenamiento se puede ampliar mediante la adición de módulos y periféricos externos, como discos duros USB o tarjetas de expansión. Esto brinda la flexibilidad necesaria para adaptarse al crecimiento de la aplicación y gestionar un mayor volumen de solicitudes y datos.

4.1.2. Python y Flask framework

Para levantar el en Flask juega un papel fundamental en la recepción y gestión de las imágenes enviadas por el módulo WiFi. Esta aplicación web, basada en Python, utiliza el framework Flask para crear un servidor HTTP robusto y flexible.

Al iniciar el servidor Flask, se establece un punto de entrada principal conocido como la ruta raíz". A partir de esta ruta raíz, se pueden definir distintos endpoints para manejar las solicitudes entrantes. Por ejemplo, se podría definir un endpoint /upload"para recibir las imágenes enviadas desde el módulo WiFi. En la figura 4.2 se muestra la composicion de una URL.

Cuando se realiza una petición HTTP POST al endpoint /upload", el servidor Flask recibe los datos en forma de un payload, que es la parte del mensaje HTTP que contiene la imagen en formato JPEG. El servidor procesa este payload para extraer la imagen y realizar las operaciones necesarias, como guardarla en un directorio específico o almacenarla en una base de datos.

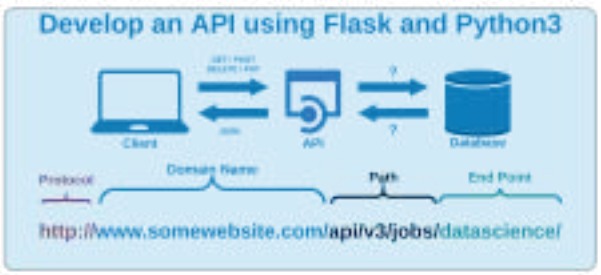


Figura 4.2: Logitipo de Flask.

Además de recibir y procesar las imágenes, el servidor Flask puede implementar otras funcionalidades, como autenticación de usuarios, gestión de sesiones, validación de datos y generación dinámica de respuestas. Esto permite crear una aplicación web completa y personalizada para interactuar con las imágenes capturadas por el módulo WiFi.

Una ventaja de utilizar Flask como framework para el servidor es su capacidad de extensión y modularidad. Flask ofrece una amplia gama de extensiones que facilitan tareas comunes en el desarrollo web, como el manejo de formularios, el enrutamiento avanzado, el almacenamiento en caché y la autenticación. Estas extensiones permiten mejorar la funcionalidad del servidor y adaptarlo a las necesidades específicas del proyecto.

En el caso de este proyecto, se ha levantado un endpoint en url /photo, tal y como se puede observar en la el siguiente fragmento de código:

```
#!/bin/python3

#  main.py
#  Date: Apr 17 07:49:03
#  Author: Rubén Garrido
#  Mail: rgarrido.rbn@gmail.com

import base64
from flask import Flask, request

app = Flask(__name__)

@app.route('/photo', methods=['POST'])
def photo():
    content_type = request.headers.get('Content-Type')
    if content_type == 'image/jpeg':
        #img_data = request.get_data() ## without base64
        img_data = request.get_data()
        with open('photo.jpg', 'wb') as f:
            f.write(img_data)
        return 'Imagen recibida y almacenada', 200
    else:
        return 'Se esperaba una imagen JPEG', 500
```

Tal y como se puede observar, al realizar una petición al endpoint, el servidor esperará como header una imagen en formato jpeg, una vez recibida, el servidor se encarga

de almacenar la imagen recibida como un fichero en el sistema de ficheros del sistema operativo, en este caso un sistema Linux.

También se puede contemplar en el fragmento de código anterior, como si la imagen se almacena correctamente y viene especificada con el formato correcto, el servidor devolverá al cliente un código http 200, cuya traducción directa sería, petición recibida y procesada correctamente.

En el caso contrario, si se produce algún error y el header no coincide con el esperado, el servidor devuelve al cliente un código de error 500, cuya traducción directa sería código de error interno en el servidor.

4.2. Procesado de imágenes con inteligencia artificial

En esta sección, se aborda uno de los componentes clave de este proyecto: la integración de un modelo de inteligencia artificial en el servidor central. La inteligencia artificial desempeña un papel fundamental al permitir la detección automatizada de objetos en las imágenes capturadas por el sistema. Dotando al sistema de múltiples funcionalidad y aplicaciones diferentes

El objetivo principal de esta sección es describir que modelo de inteligencia artificial y cómo se ha hecho funcionar, incluyendo los pasos necesarios y las herramientas utilizadas. Se abordarán aspectos técnicos relacionados con la descargar de los ficheros necesarios para el modelo así como su puesta a punto junto a todo el sistema.

A través de esta sección, se proporcionará una visión general completa de cómo la integración de un modelo de IA en el dispositivo Raspberry Pi ha contribuido a la detección automatizada de objetos en las imágenes capturadas, mejorando así la eficiencia y utilidad del sistema en todo el conjunto.

4.2.1. Funcionamiento de la Inteligencia Artificial

En este apartado se detalla el funcionamiento de la Inteligencia Artificial en el dispositivo. Para este proyecto se ha escogido la detección de objetos utilizando el modelo MobileNet. MobileNet es un modelo de inteligencia artificial especialmente diseñado para tareas de detección de objetos en imágenes, y ha sido adaptado para su uso en dispositivo Raspbery Pi que se ha desarrollado. En la figura 5.2 se muestra un breve esquema del funcionamiento del procesado de las imágenes.

Figura 4.3: Esquema funcionalidad IA.

El proceso comienza cuando el servidor Flask recibe una imagen capturada por el dispositivo. Esta imagen se pasa al modelo MobileNet, dicho modelo esta precompilado y se puede descargar de internet, tal y como se puede observar en la siguiente sección. El modelo procesa la imagen y realiza la detección de objetos en tiempo real, proporcionando

a la salida una etiqueta y una bounding box con el objeto detectado, tal y como se muestra en un ejemplo en la figura 4.4

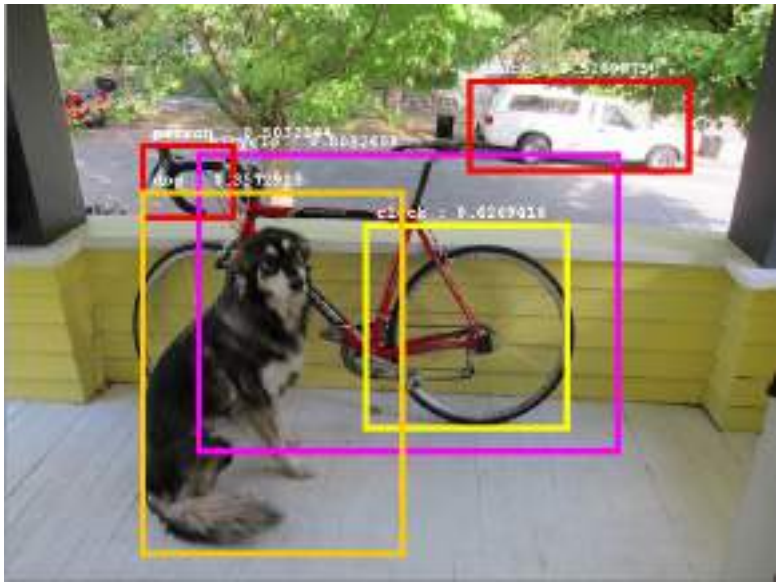


Figura 4.4: Ejemplo detección Inteligencia artificial TensorFlow.

El modelo MobileNet utiliza técnicas de aprendizaje profundo y convoluciones para analizar y categorizar los objetos presentes en la imagen. Está entrenado en una amplia variedad de categorías de objetos, lo que le permite reconocer y etiquetar de manera precisa diferentes tipos de objetos, como personas, animales, vehículos, etc.

Debido a la complejidad que conlleva el entrenamiento de un modelo de inteligencia artificial eficiente y que garantice un funcionamiento correcto, para realizar la prueba de concepto se ha decidido utilizar un modelo ampliamente probado por la comunidad como puede ser el ejemplo del modelo de MobileNet. Se puede encontrar mucha información acerca de este modelo en su web correspondiente, la cual se proporciona en esta memoria a través del siguiente enlace https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi.git.

La idea en todo momento de este proyecto ha sido dotar de modularidad y abstracción a todos y cada uno de los módulos que componen el proyecto, es por ello que si en un futuro, aparece una tecnología muy superior en alguno de los módulos, estos sean facilmente reemplazables. Tal y como puede ocurrir en ael campo de la inteligencia artificial.

Una vez que el modelo MobileNet ha realizado la detección de objetos en la imagen, se generan salidas que indican las ubicaciones de los objetos detectados, así como las etiquetas correspondientes a cada objeto. Estas salidas se utilizan para proporcionar información sobre los objetos detectados al servidor Flask y, en última instancia, al usuario.

Los desarrolladores de TensorFlow, proporcionan una serie de scripts para poder invocar a los modelos, dotando al proceso de diferentes funcionalidades, en el siguiente fragmento de código, se pueden observar todos los argumentos de los cuales se puede acompañar la llamada al script de detección de objetos.

```
# Import packages
import os
import argparse
import cv2
import numpy as np
import sys
import glob
import importlib.util


# Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is
    located in',
                    required=True)
parser.add_argument('--graph', help='Name of the .tflite file, if
    different than detect.tflite',
                    default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if
    different than labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--threshold', help='Minimum confidence threshold for
     displaying detected objects',
                    default=0.5)
parser.add_argument('--image', help='Name of the single image to perform
    detection on. To run detection on multiple images, use --imagedir',
                    default=None)
parser.add_argument('--imagedir', help='Name of the folder containing
    images to perform detection on. Folder must contain only images.',
                    default=None)
parser.add_argument('--save_results', help='Save labeled images and
    annotation data to a results folder',
                    action='store_true')
parser.add_argument('--noshow_results', help='Don\'t show result images (
    only use this if --save_results is enabled)',
                    action='store_false')
parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to
    speed up detection',
                    action='store_true')

args = parser.parse_args()


# Parse user inputs
MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
```

La integración de la inteligencia artificial en el dispositivo permite la detección de objetos en tiempo real a partir de las imágenes capturadas. Esto abre un amplio abanico de posibilidades para aplicaciones prácticas, como la seguridad, la monitorización y la automatización de tareas.

En el apartado siguiente se muestra en detalle como se ha realiza la puesta en marcha del modelo, la configuración que se ha decidido adoptar, así como la integración que se ha debido de adaptar al hardware utilizado.

4.2.2. Puesta a punto Coral TPU

Con el objetivo de poder llevar a cabo la detección de objetos a tiempo real, se ha hecho uso de un acelerador gráfico USB, en concreto un dispositivo TPU (Tensor Procesor Unit) el cual se encargará de poder llevar a cabo el cálculo y procesado relacionado con la detección de objetos y la aplicación y procesado del modelo matemático.

Para poder llevar a cabo el procesamiento de las imágenes y el cálculo estructural de la red neuronal, se utilizan dispositivos a los que comúnmente se les denomina acelerador gráfico. Estos dispositivos se encargar de soportar el peso computacional correspondiente a los cálculos asociados a los modelos matemáticos. En este caso, el gran parte del peso del procesado recae sobre el modelo matemático y su procesado en CPU, es por ello, que para obtener unos mejores tiempos de respuesta y poder llevar a cabo un procesado en tiempo real y con una latencia baja se utilice una TPU.

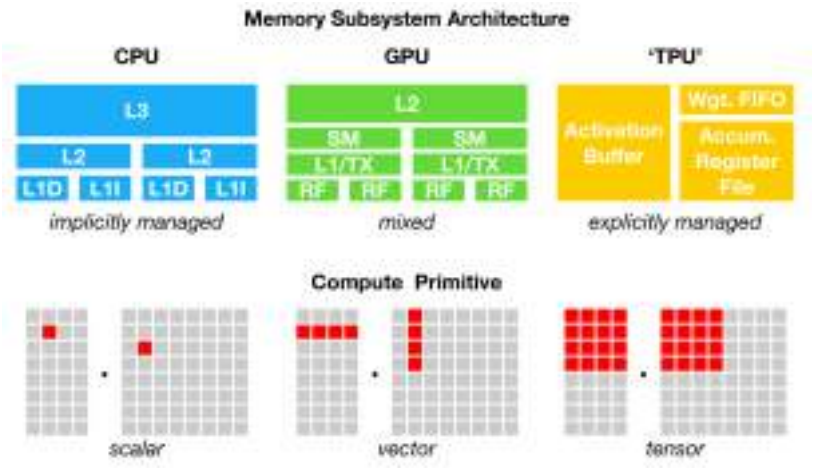


Figura 4.5: Comparativa CPU, GPU y TPU.

Tal y como se puede observar en la figura 4.5, cada una de los diferentes tipos de procesadores destaca en un ámbito diferente y cada un o tiene su aplicaciones particular. En lo que atañe al proyecto, ya que la carga de trabajo con mayor peso recae sobre el modelo de inteligencia artificial, la mejor opción para poder ejecutar todos estos cálculos de la manera más eficiente es utilizar una TPU.

Las TPU son capaces de procesar la información de modelos matemáticos, como por ejemplo las redes neuronales, de 15 a 30 veces mas rápido que una GPU convencional, además de ser mas eficientes energéticamente debido a su bajo consumo. El inconveniente de estos dispositivos es la baja versatilidad a la hora de incluirlos en el proyecto. Para la aplicación deseada encaja perfectamente debido al tipo de procesado que realizará, el bajo coste económico y la fácil instalación del dispositivo en el sistema operativo gracias

a su conexión USB 3.0. En la figura 4.6 se muestra el dispositivo físico.

Figura 4.6: Google Coral TPU USB Accelerator.

Para poder poner en funcionamiento la TPU se deben de cumplir una serie de requisitos software para que el sistema operativo sea capaz de trabajar con el hardware. Es necesario instalar una serie de drivers para que se pueda establecer un entendimiento hardware-software y de esa forma poder realizar la mayor parte del procesado en este hardware.

Para poner a punto la Coral TPU y poder comprobar su correcto funcionamiento, el primero de los pasaos a seguir es instalar un gestor de paquetes para poder resolver todas las dependencias que se vayan necesitando a lo largo de la instalación y puesta a punto. En el caso de los proyectos, se ha instalado conda para poder gestionar entornos e instalar los diferentes paquetes para python que se van necesitando a lo largo de la puesta en marcha. En la figura 4.7 se muestra el logotipo que se puede encontrar por la red fácilmente.

Figura 4.7: Gestor de paquetes y entornos Conda.

Se ha de tener en cuenta la distribución y arquitectura que presentará el proyecto a la hora de poder instalar un paquete determinado ya que dependiendo de cada arquitectura, distribución y kernel de cada uno de los sistemas operativos y hardware que se esté utilizando se deberá de adaptar a un software y otro.

Para el caso del proyecto, se esta ejecutando todo el código sobre un linux, con una distribución basada en ubuntu server al que se le ha instalado un entorno gráfico. A la

hora de instalar el gestor de paquetes Conda hay que tener en cuenta diversos factores diferenciales.

Primero debemos de conocer la arquitectura y la versión del sistema operativo sobre el cual estamos ejecutando, para ello se pude abrir una ventana de comandos o terminal y ejecutar el siguiente comando: *uname -m*.

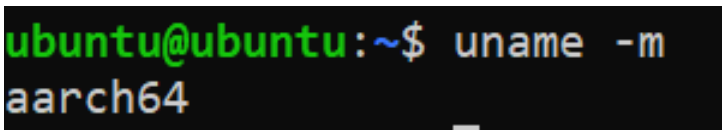


Figura 4.8: Ejecución de comando `uname -r`.

Tal y como se puede observar en la figura 4.8, el comando *uname -r* nos muestra en la terminal el tipo de arquitectura sobre el que estamos trabajando. Como era de esperar, Raspberry trabaja con una CPU ARM, por lo cual el resultado de ejecutar el comando es aarch64.

Para poder instalar conda sobre procesadores ARM habrá que recurrir a una versión de conda adaptada para estas arquitecturas, ya que los gestores mas utilizados para poder trabajar con conda, bien sea **Anaconda** o bien Miniconda únicamente están disponibles para poder instalarlos sobre procesadores AMD, por lo cual hay que buscar una alternativa o solución a este pequeño inconveniente.

Tras investigar las diferentes opciones que se encuentran como software abierto por la red, se ha escogido Archiconda como gestor de entornos y paquetes de conda. Archiconda es un software libre y abierto para poder instalar en dispositivos con arquitectura ARM como suelen ser la mayoría de placas de desarrollo para sistemas embebidos.

Para poder instalar archiconda es necesario descargar un repositorio para posteriormente ejecutar los diferentes comandos necesarios. Con el objetivo de simplificar toda la instalación se ha cogido de un repositorio un bash script para linux que se puede o bien descargar o bien generar desde la propia consola.

Ya que el objetivo es poder dejar toda la información necesaria en un mismo repositorio, se ha decidido generar un fichero bash script propio para poder almacenarlo en un repositorio propio.

Para poder crear el fichero se debe de realizar el siguiente procedimiento:

En primer lugar se debe de generar un archivo en linuz con extension .sh, a este fichero se le dará el nombre que el usuario desee, en este caso *installArchiconda.sh*.

Una vez creado este fichero, se rellenará con el código o acciones que se deseen ejecutar, en este caso el extracto de código que se desea introducir se puede muestra a continuación.

Una vez el archivo queda generado, se guarda para su posterior ejecución. Para poder dar permiso de ejecución en un sistema Linux se utilizará el comando *sudo chmod +x installArchiconda.sh*. De este modo estamos añadimos permiso de ejecución para el fichero y se podrá ejecutar desde la terminal utlizando *./installArchiconda.sh*. En las siguientes figuras se puede observar el proceso de creación del fichero así como su ejecución.

```
#! /bin/bash

cd ${HOME}
wget https://github.com/Archiconda/build-tools/releases/download/0.2.3/
    Archiconda3-0.2.3-Linux-aarch64.sh
sudo sh Archiconda3-0.2.3-Linux-aarch64.sh
rm -rf Archiconda3-0.2.3-Linux-aarch64.sh
cd ~
sudo chown -R $USER archiconda3/
export "PATH=/bin:/usr/bin:$PATH" >> ~/.bashrc
source ~/.bashrc
conda config --add channels conda-forge
conda -V
```

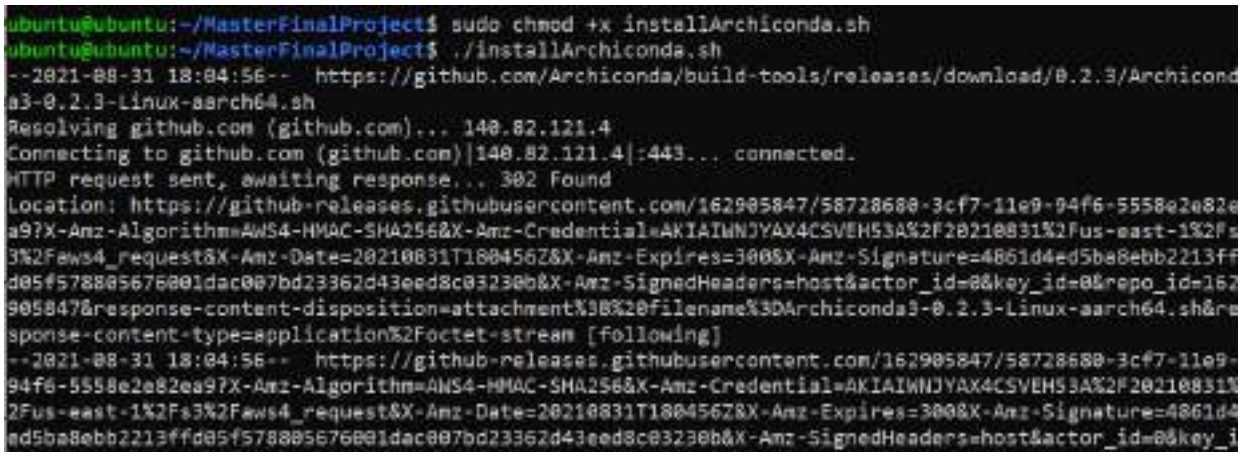


Figura 4.9: Ejecución archivo instalador archiconda.

Una vez se ha instalado archiconda, como se muestra en la figura 4.9 se puede comprobar que se ha instalado correctamente ejecutando el comando *conda –version* o bien *conda -V*. Si este comando devuelve la versión de conda significará que se ha instalado correctamente y estará todo listo para crear un entorno e instalar los diferentes paquetes necesarios.

Con el gestor de paquetes conda se puede proceder a la instalación y preparación de todo el entorno de desarrollo. El primer de los pasos a seguir para poder poner a punto el proyecto, es crear un entorno aislado donde poder compilar y donde poder alojar todos los paquetes necesarios para que a la hora de ejecutar se puedan encontrar todas las librerías en el entorno adecuado. Para poder hacer esto el primero de los comando a ejecutar es *conda create –name tflite1-env*. Como se muestra en la figura 4.10.

Una vez creado el entorno conda donde poder ejecutar los comandos, el siguiente paso será poder ejecutar *conda activate tflite1-env*. De esta forma se activa el entorno conda donde se quiere trabajar y estará todo preparado para poder empezar a instalar dependencias. Como se muestra en la figura 4.11.

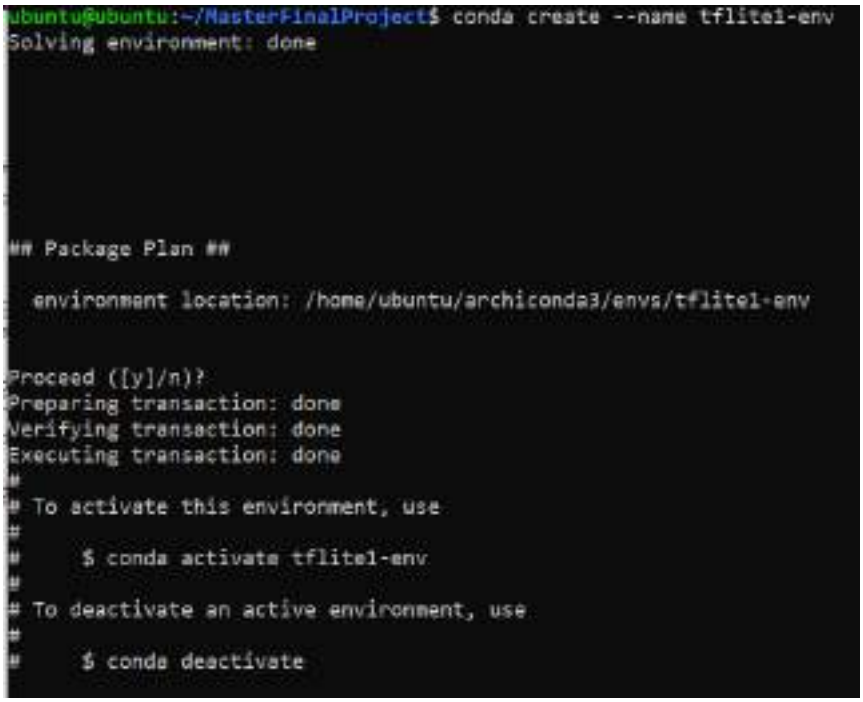


Figura 4.10: Ejecución archivo instalador archiconda.

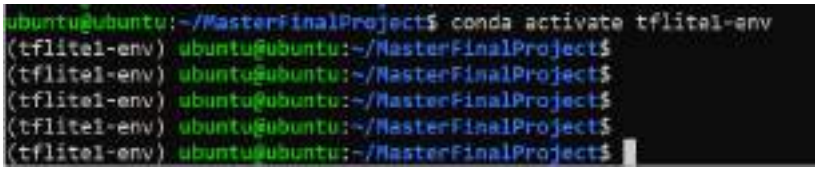


Figura 4.11: Ejecución archivo instalador archiconda.

El siguiente paso es instalar las dependencias, para poder instalar las dependencias desde un mismo scripts se han seguido los mismos pasos que se han seguido para la instalación de archiconda. Los requerimientos para poder ejecutar la detección de objetos vienen en el siguiente archivo ejecutable:

```
#!/bin/bash

# Get packages required for OpenCV

sudo apt-get -y install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
sudo apt-get -y install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get -y install libxvidcore-dev libx264-dev
sudo apt-get -y install qt4-dev-tools libatlas-base-dev

# Need to get an older version of OpenCV because version 4 has errors
pip3 install opencv-python
# pip install opencv-python (If pip3 drops you an error)

# Get packages required for TensorFlow
# Using the tflite_runtime packages available at https://www.tensorflow.org/lite
    /guide/python
# Will change to just 'pip3 install tensorflow' once newer versions of TF are
    added to piwheels
```

```
#pip3 install tensorflow
# Modded for installin tflite on a RPI4 running ubuntu mate 20.04
version=$(python -c 'import sys; print(".".join(map(str, sys.version_info[:2])))
    ')

if [ $version == "3.7" ]; then
pip3 install --extra-index-url https://google-coral.github.io/py-repo/
    tflite_runtime
fi

if [ $version == "3.5" ]; then
pip3 install --extra-index-url https://google-coral.github.io/py-repo/
    tflite_runtime
fi
```

Listado 4.2: Código bash

Una vez instalado el fichero de requirements, la raspberry estará lista para poder ejecutar una red neuronal precompilada. El siguiente de los pasos consiste en poder establecer la Coral TPU como dispositivo que será encargado de poder procesar toda esta información y poder realizar la detección de objetos.

En este punto la Raspberry es capaz de compilar y poder ejecutar la detección de objetos. Esto se puede conseguir tanto desde una entrada de vídeo como puede ser por ejemplo una webcam, o bien se puede conseguir procesar una imagen o vídeo y aplicar los coeficientes de la red neurona para poder clasificar los diferentes inputs que puedan llegar a través del vídeo.

Capítulo 5

Resultados y discusión

5.1. Obtención de resultados

En este capítulo se presentarán los resultados obtenidos y se describirá la puesta en marcha del sistema completo desarrollado en el marco de este proyecto. Se discutirán los logros alcanzados, las pruebas realizadas y se analizará el rendimiento del sistema en diferentes escenarios.

Para comenzar, se mostrarán los resultados de la implementación del módulo ESP32 y su funcionamiento en los distintos modos: servidor, cliente y captura de fotos. Se analizará el desempeño del módulo, la estabilidad de la conexión Wi-Fi y la calidad de las imágenes capturadas. Además, se evaluará el consumo de energía y se realizarán pruebas de interoperabilidad con otros dispositivos y protocolos.

A continuación, se presentarán los resultados de la integración del módulo ESP32 con el servidor Python basado en Raspberry Pi. Se mostrará cómo se establece la comunicación entre ambos componentes y se discutirán los resultados de las pruebas de envío y recepción de imágenes a través del protocolo HTTP. Se analizará la eficiencia del servidor en términos de tiempos de respuesta y carga de trabajo.

Posteriormente, se abordará la implementación de la inteligencia artificial en la Raspberry Pi y se mostrarán los resultados obtenidos en el procesamiento de imágenes. Se discutirán las técnicas utilizadas, como el reconocimiento de objetos o la detección de rostros, y se evaluará la precisión y el rendimiento del sistema en diferentes escenarios.

Finalmente, se describirá el proceso de puesta en marcha del sistema completo. Se detallarán los pasos necesarios para configurar y desplegar tanto el módulo ESP32 como el servidor Python en la Raspberry Pi. Se discutirán las consideraciones de seguridad y se proporcionarán recomendaciones para garantizar un despliegue exitoso.

5.1.1. Resultados de módulo ESP32

En primer lugar, la primera de las pruebas de funcionamiento, ha consistido en comprobar la capacidad del dispositivo de conectarse al router configurado y que este mismo, mediante el servdor DHCP que tiene, sea capaz de darle una IP y tenerlo en la red.

A continuación se muestran los logs del sistema mediante los cuales se ha completado el proceso de depuración de las tareas realizadas:

```
I (1980) wifi:wifi firmware version: 7e67c79
I (1980) wifi:wifi certification version: v7.0
I (1980) wifi:config NVS flash: enabled
I (1980) wifi:config nano formating: disabled
I (1980) wifi:Init data frame dynamic rx buffer num: 32
I (1990) wifi:Init management frame dynamic rx buffer num: 32
I (1990) wifi:Init management short buffer num: 32
I (2000) wifi:Init dynamic tx buffer num: 32
I (2000) wifi:Init tx cache buffer num: 32
I (2010) wifi:Init static rx buffer size: 1600
I (2010) wifi:Init static rx buffer num: 10
I (2010) wifi:Init dynamic rx buffer num: 32
I (2020) wifi_init: rx ba win: 6
I (2020) wifi_init: tcpip mbox: 32
I (2030) wifi_init: udp mbox: 6
I (2030) wifi_init: tcp mbox: 6
I (2030) wifi_init: tcp tx win: 5744
I (2040) wifi_init: tcp rx win: 5744
I (2040) wifi_init: tcp mss: 1440
I (2050) wifi_init: WiFi IRAM OP enabled
I (2050) wifi_init: WiFi RX IRAM OP enabled
I (2060) phy_init: phy_version 4670,719f9f6,Feb 18 2021,17:07:07
I (2160) wifi:mode : sta (94:b5:55:2c:9c:a0)
I (2160) wifi:enable tsf
I (2170) WifiSTANetwIface: STA started
I (2170) WifiSTANetwIface: STA configured successfully
```

Tal y como se observa, la interfaz wifi ha sido configurada correctamente, con lo cual, el dispositivo estaría listo para realizar en intercambio de información con el punto de acceso el cual se ha configurado.

El siguiente paso, es comprobar que el dispositivo se puede conectar de forma satisfactoria al punto de acceso, y tal como se comenta con anterioridad, este deberia de asignarle una IP a nuestro dispositivo.

```
I (2440) WifiSTANetwIface: Configure STA to connect to SSID garrido2
I (2470) WifiSTANetwIface: Start connecting to AP
I (2480) wifi:new:<8,2>, old:<1,0>, ap:<255,255>, sta:<8,2>, prof:1
I (2480) wifi:state: init -> auth (b0)
I (2490) wifi:state: auth -> assoc (0)
I (2500) wifi:state: assoc -> run (10)
W (2510) cam_hal: NO-SOI
I (2510) HttpServerEsp: Registering URI on: /photo
I (2520) wifi:connected with garrido2, aid = 2, channel 8, 40D, bssid =
   f8:1a:67:8c:60:96
I (2530) wifi:security: WPA2-PSK, phy: bgn, rssi: -67
I (2540) wifi:pm start, type: 1
I (2540) WifiSTANetwIface: Connected to AP
I (2540) HttpServerEsp: Registering URI on: /photoInfo
I (2540) HttpServerEsp: Registering URI on: /photoServer
```

```
I (2550) HttpClientEsp: HTTP_EVENT_DISCONNECTED
I (2550) HttpClientEsp: Last esp error code: 0x103
I (2560) HttpClientEsp: Last mbedtls failure: 0x0
I (2570) HttpClientEsp: HTTP_EVENT_DISCONNECTED
I (2570) HttpClientEsp: Last esp error code: 0x103
I (2580) HttpClientEsp: Last mbedtls failure: 0x0
E (2590) gpio: gpio_install_isr_service(449): GPIO isr service already
   installed
I (2600) gpio: GPIO[15]| InputEn: 1| OutputEn: 0| OpenDrain: 0| Pullup:
   1| Pulldown: 0| Intr:1
I (2600) wifi:AP's beacon interval = 102400 us, DTIM period = 1
I (2610) ButtonTrigger: Configuring gpioNum: 15, on index: 0
I (3430) WifiSTANetwIface: got ip:192.168.204.101
I (3430) ConnManager: Connection to AP established
I (3430) ConnManager: connManagerFSMProcessLoop: New event
   CONNECTION_SUCCESS received
I (3430) esp_netif_handlers: sta ip: 192.168.204.101, mask:
   255.255.255.0, gw: 192.168.204.1
```

Como se puede observar en el fragmento de logs de sistema anterior, el dispositivo ha sido capaz de conectarse al punto de acceso configurado, en este caso con SSID garrido2, recibiendo una IP, en este caso, la ip 192.168.204.101, mask: 255.255.255.0, gw: 192.168.204.1. Si desde el PC analizamos los direntes dispositivos que comparten subred con el módulo wifi, se puede observar como se encuentra en la red con la misma IP que se muestra con anterioridad.

```
Nmap scan report for 192.168.204.101
Host is up (0.0035s latency).
Not shown: 999 closed ports
PORT   STATE        SERVICE
80/tcp open|filtered http
MAC Address: 94:B5:55:2C:9C:A0 (Unknown)
```

Tal y como se menciona, la salida del comando nmap en linux, arroja la información necesaria para saber que hay un dispositivo con una mac concreta en la red y que además tiene el puerto 80 abierto a conexiones TCP, lo cual está indicando que la cámara en modo servidor esta funcionando correctamente.

```
I (2500) HttpServerEsp: Registering URI on: /
I (2500) HttpServerEsp: Registering URI on: /capture
I (2500) HttpServerEsp: Registering URI on: /stream
I (2510) HttpServerEsp: Registering URI on: /photo
I (2520) wifi:connected with garrido2, aid = 2, channel 8, 40D, bssid =
   f8:1a:67:8c:60:96
I (2530) wifi:security: WPA2-PSK, phy: bgn, rssi: -67
I (2540) wifi:pm start, type: 1

I (2540) WifiSTANetwIface: Connected to AP
I (2540) HttpServerEsp: Registering URI on: /photoInfo
I (2540) HttpServerEsp: Registering URI on: /photoServer
```

En el fragmento de logs anterior está la evidencia de que el servidor http está siendo levantado y se estan registrando los diferentes endpoints configurados.

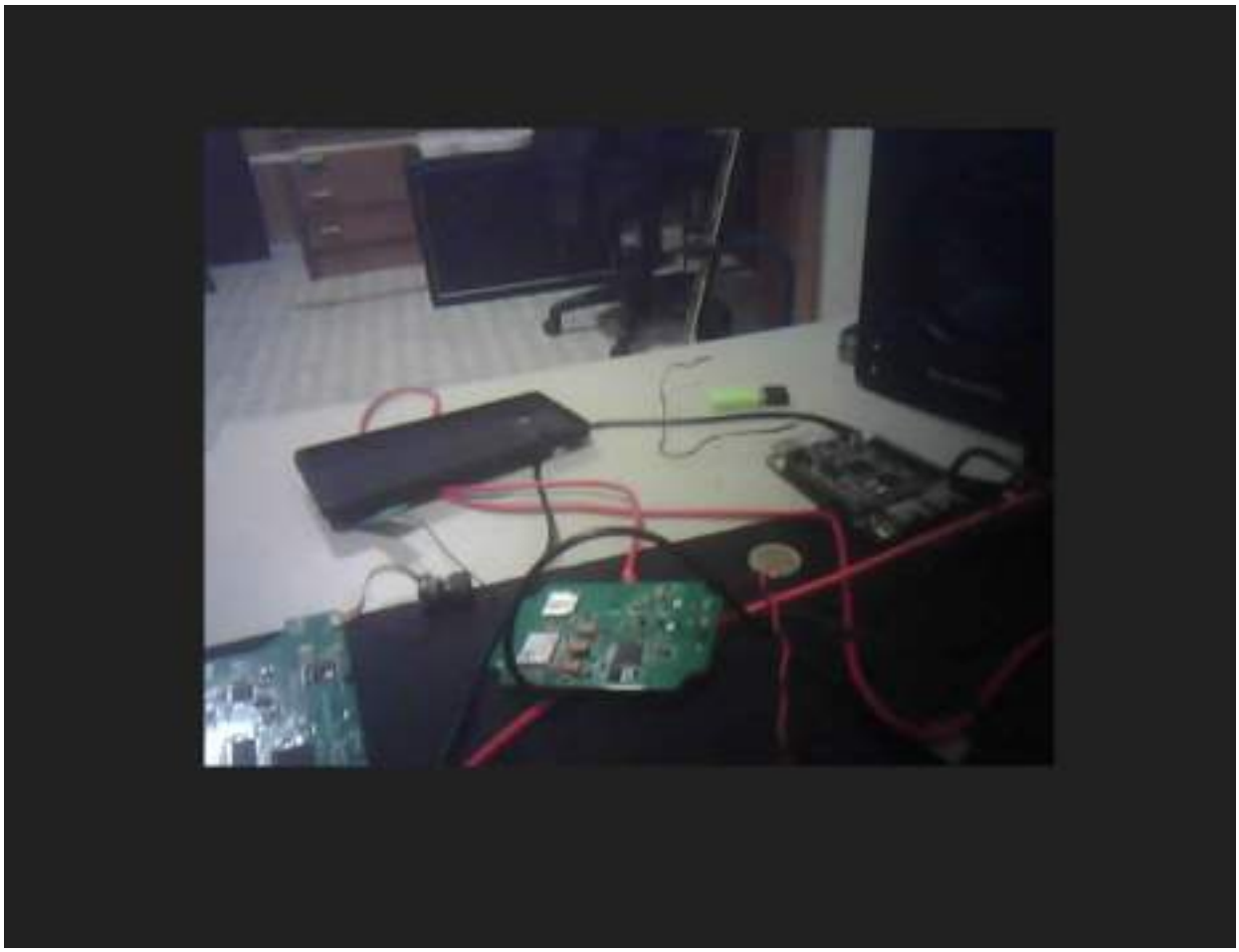


Figura 5.1: Captura realizada en modo stream.

En la figura 5.1 se muestra una captura del navegador realizando streaming.

Una vez comprobado que el stream funciona, vamos a comprobar que la parte del cliente http funciona correctamente, para poder enviar la petición de envío de foto al servidor se ha utilizado un endpoint en el servidor montado en el microcontrolador, de forma que al llegarle una petición POST, este lanzará un hilo en el microcontrolador que realizará la captura de la foto y su posterior envío al servidor.

En el siguiente fragmento de código se observar como desde un dispositivo se realiza la petición POST para activar al microcontrolador:

```
~/tfm_sandbox/cameraPyServer (develop*) $ curl --request POST \
  --url http://192.168.204.101:80/photoServer
{msg: "Request sent"}%
```

Por parte del microcontrolador se ha recibido la petición, se ha capturado la foto así como se ha realizado en envío correcto al servidor:

```
I (15363) httpCamera: picture take request fromweb server
I (15363) CameraClient: Taking picture...
I (15363) CameraClient: Picture taken! Its size was: 8403 bytes
I (15373) CameraClient: picture sent to the queue
I (15403) HttpClientEsp: HTTP_EVENT_DISCONNECTED
I (15403) CameraClient: La respuesta del servidor es Imagen recibida y
   almacenada

I (15403) CameraClient: Finishing send task
```

Con esto se tienen verificado el funcionamiento correcto del módulo ESP32, tanto por la conexión a la red, tanto por su funcionamiento en modo cliente y en modo servidor.

5.1.2. Resultados Servidor Python

En cuanto al servidor Python, se ha comprobado que los logs proporcionados por el sistema python a la hora de levantar el servidor sean correctos.

```
~/tfm_sandbox/cameraPyServer (develop*) \$ python3 main.py
 * Serving Flask app 'tfm_server'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production
   deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:3000
 * Running on http://192.168.204.103:3000
```

Tal y como se puede observar, el servidor se levanta correctamente en todas las interfaces de red, de forma que puede recibir peticiones entrantes en todas y cada una de las interfaces de red.

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address       Foreign Address       State
tcp       0      0 127.0.0.1:36329       0.0.0.0:*             LISTEN
tcp       0      0 127.0.0.1:631         0.0.0.0:*             LISTEN
tcp       0      0 0.0.0.0:22            0.0.0.0:*             LISTEN
tcp       0      0 127.0.0.53:53         0.0.0.0:*             LISTEN
tcp       0      0 0.0.0.0:3000          0.0.0.0:*             LISTEN
```

Para ello, se ha generado otro script de testeo, el cual se lanzará desde el mismo dispositivo, realizando una petición a la interfaz 127.0.0.1 puerto 3000 para comprobar su correcto funcionamiento:

```
#!/bin/python3
```

```
#  photo_request.py
#  Date: Apr 17 12:06:39
#  Author: Rubén Garrido
#  Mail: rgarrido.rbn@gmail.com

import requests

# Archivo de imagen para enviar
file_path = 'sample.jpg'

# URL del servidor para hacer la petici\u00f3n POST
url = 'http://localhost:3000/photo'

# Abrir y leer el archivo de imagen
with open(file_path, 'rb') as f:
    img_data = f.read()

# Hacer la peticion POST con los datos de la imagen
response = requests.post(url, data=img_data, headers={'Content-Type': '
    image/jpeg'})

# Imprimir la respuesta del servidor
print(response.text)
```

Al ejecutar este script, se puede ver como la respuesta del servidor es un código 200, haciendo referencia a que la petición se ha ejecutado correctamente.

```
~/tfm_sandbox/cameraPyServer/test (develop*) $ python3 photo_request.py
200 - Imagen recibida y almacenada
```

Visto desde el lado del servidor, esto es lo que ocurre cuando se le lanza una petición desde el módulo wifi:

```
~/tfm_sandbox/cameraPyServer (develop*) $ python3 main.py
 * Serving Flask app 'tfm_server'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production
    deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:3000
 * Running on http://192.168.204.103:3000

 192.168.204.101 - - [26/Jun/2023 18:43:21] "POST /photo HTTP/1.1" 200 -
```

De este modo se comprueba que el servidor esta funcionando correctamente.

5.1.3. Resultados procesado Inteligencia Artificial

El siguiente paso en el flujo del proyecto, consiste en procesar las imágenes capturadas por el microcontrolador ESP32 y posteriormente recibidas por el servidor, a un modelo de inteligencia artificial previamente entrenado. El objetivo era obtener información adicional y realizar tareas específicas sobre las imágenes capturadas, como por ejemplo poder realizar una detección de objetos y posicionarlos dentro de la imagen.

Para ello, se utilizó un modelo de clasificación de imágenes que proporciona tensorflow lite en uno de sus ejemplos, el cual cuenta con una amplia extensión de etiquetas posibles las cuales pueden ser detectadas por el modelo. A continuación se pueden ver algunas de las etiquetas disponibles.

```
~/tfm_sandbox/MasterFinalProject (master*) -> cat Sample_TFLite_model/
   labelmap.txt
person
bicycle
car
motorcycle
airplane
bus
train
truck
boat
traffic light
fire hydrant
stop sign
parking meter
bench
bird
cat
dog
horse
sheep
cow
elephant
bear
zebra
keyboard
giraffe
backpack
umbrella
handbag
tie
suitcase
frisbee
skis
snowboard
sports ball
kite
baseball bat
```

```
baseball glove
skateboard
surfboard
tennis racket
scissors
teddy bear
hair drier
toothbrush
```

Tras la comprobación y almacenaje de cada una de las imágenes que se han capturado con el microcontrolador, se realiza en el mismo proceso del dispositivo, el procesamiento de las imágenes mediante el modelo de inteligencia artificial del cual se obtuvieron resultados satisfactorios. El modelo fue capaz de clasificar correctamente las imágenes en diferentes categorías, lo que permitió identificar objetos, personas o situaciones específicas presentes en las imágenes capturadas.

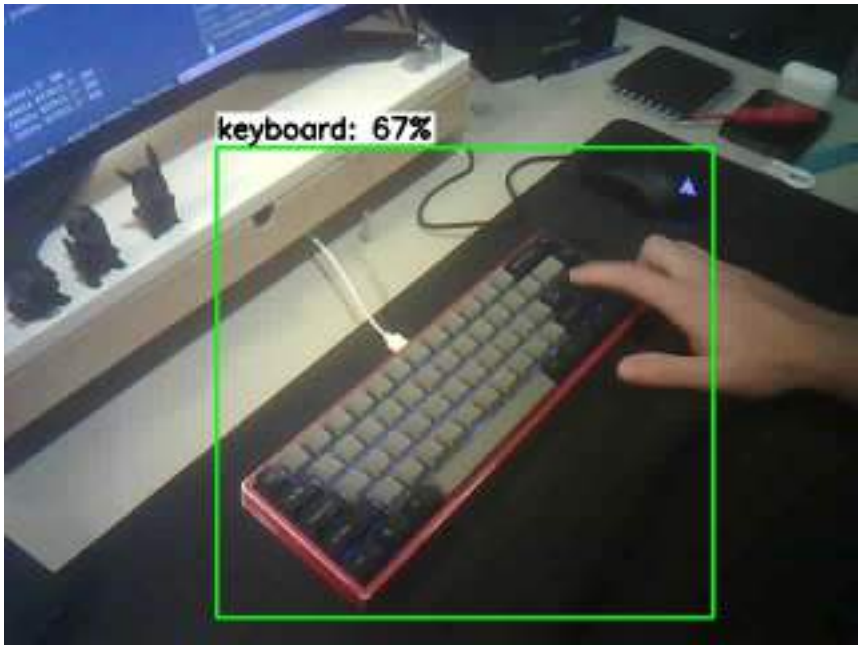


Figura 5.2: Resultado post procesado IA.

En la figura 5.2 se muestra como en una captura al escritorio desde donde se está desarrollando el proyecto, es capaz de poder reconocer un teclado.

El resultado visual resulta muy útil para la percepción humana y para darnos una idea de lo que esta procesado el modelo de inteligencia artificial, pero obviamente, para los diferentes dispositivos que puedan realizar una acción con el resultado de este procesado, necesitan una información interpretable. Es por ello que también se almacena el procesador de esta imagen con la siguiente información:

```
File: results/photo.txt
   1 - keyboard 0.6797 159 107 531 459
```

Como se puede ver en el fragmento anterior, los dos primeros campos corresponden a la etiqueta que se ha detectado y dentro de las etiquetas disponibles del modelo el porcentaje de probabilidad que le da a la etiqueta teclado, como se puede imaginar, los

modelos no son perfectos y como salida también proporcionan este tipo de información para que se les pueda añadir un umbral de fiabilidad a nuestro proyecto.

Los siguientes cuatro campos, corresponden a la *bounding box* que es el nombre que recibe la caja que encuadra el objeto detectado, los cuatro campos hacen referencia a las coordenadas de los píxeles de las esquinas.

Este resultado demuestra la viabilidad y efectividad de integrar la inteligencia artificial en el sistema, ampliando su capacidad de análisis y permitiendo la extracción de información relevante de las imágenes capturadas. Estos datos procesados pueden ser utilizados posteriormente para aplicaciones como detección de objetos, reconocimiento facial o análisis de patrones en tiempo real.

En resumen, la integración de un modelo de inteligencia artificial en el sistema permitió obtener resultados procesados y enriquecidos a partir de las imágenes capturadas, mejorando así la capacidad de análisis y apertura de nuevas posibilidades para aplicaciones futuras.

Capítulo 6

Conclusiones

6.1. Conclusiones

En este proyecto se ha desarrollado un sistema completo basado en el módulo ESP32, Raspberry Pi envolviendo diferentes aspectos, tanto a nivel de realizar elecciones claves en cuanto a software así como decisiones a nivel de componentes hardware. A lo largo de este trabajo, se han logrado los diferentes objetivos propuestos y se han obtenido resultados gratificantes en cada una de las etapas de implementación.

En primer lugar, la implementación del módulo ESP32 ha demostrado ser eficiente y versátil, permitiendo su funcionamiento tanto en modo servidor como en modo cliente. La captura de imágenes a través de la cámara del kit ESP32-CAM ha sido exitosa, obteniendo en una resolución más que suficiente para la aplicación de este proyecto. Además, la comunicación mediante el protocolo HTTP ha permitido el envío de imágenes de manera ágil dotando al proyecto de la capacidad de funcionamiento deseada así como un resultado escalable y facilmente ampliable para el futuro.

La integración del módulo ESP32 con el servidor Python utilizando el framwork Flask en el sistema Linux ha resultado exitosa. La Raspberry Pi ha demostrado ser una opción adecuada para alojar el servidor, lo cual demuestra que la elección de los componentes software como pueden ser lenguajes utilizados, elección de frameworks y diseño del código arroja resultados satisfactorios. Se ha conseguido una plataforma estable y de bajo consumo de energía. La comunicación entre el módulo ESP32 y el servidor se ejecutaba de forma correcta y ha funcionado tal y como se esperaba.

En cuanto a la puesta en marcha del sistema, se han proporcionado instrucciones detalladas para su configuración y despliegue. Se han abordado consideraciones de seguridad y se han ofrecido recomendaciones para garantizar un funcionamiento óptimo y proteger la integridad de los datos transmitidos.

En conclusión, el sistema desarrollado ha cumplido con éxito los objetivos establecidos. La combinación del módulo ESP32, la Raspberry Pi y las tecnologías elegidas tanto a nivel software como hardware han permitido crear un sistema eficiente y versátil para la captura y transmisión de imágenes. La integración de la inteligencia artificial ha enriquecido el sistema, abriendo la puerta a aplicaciones más avanzadas.

El proyecto ha sentado las bases para futuras mejoras y ampliaciones, ofreciendo un potencial significativo en el campo de la visión artificial y el IoT. A lo largo de todo el proyecto se han utilizado tecnologías open source y con un diseño completamente modular.

La tecnología en el campo del IoT avanza a pasos agigantados, pero con un diseño como el planteado en este proyecto, cada uno de los módulos es fácilmente sustituible por una tecnología emergente que sea superior.

6.2. Trabajo futuro

En cuanto al trabajo a futuro, existen diversas áreas que podrían ser objeto de investigación y desarrollo continuo para mejorar y ampliar el sistema implementado. Algunas de las posibles líneas de trabajo incluyen:

- Mejora de la capacidad de procesamiento de la Raspberry Pi para abordar tareas más complejas y exigentes en el análisis de imágenes, existen algunos dispositivos mas complejos pero con mayor capacidad de cómputo que podrían albergar el sistema, dotando de más funcionalidades.

- Exploración de la viabilidad de ejecutar modelos de aprendizaje automático directamente en el módulo ESP32 para mejorar la eficiencia y autonomía del dispositivo.

- Ampliación de las funcionalidades del servidor Python, como el almacenamiento en una base de datos, generación de estadísticas o integración con servicios en la nube, esto puede abrir la puerta a comercializar un producto junto con servicios configurables.

- Investigación en técnicas de seguridad para garantizar la confidencialidad e integridad de las imágenes transmitidas y la protección de datos.

- Exploración de técnicas de compresión de imágenes para optimizar la transferencia de datos y mejorar la eficiencia del sistema. Si se quiere abstraer el sistema y eliminar la dependencia de una red wifi, es imprescindible optimizar cada mensaje que se envía y se recibe del servidor.

Estas áreas de trabajo proporcionan oportunidades para seguir innovando y aplicando el sistema desarrollado en diversos campos, desde la vigilancia y seguridad hasta la automatización industrial o el análisis de datos en tiempo real.

Capítulo 7

Bibliografía

[1] Espressif Systems. *ESP32 Technical Reference Manual*. Disponible en: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf.

[2] Espressif Systems. *ESP-IDF Programming Guide*. Disponible en: https://docs.espressif.com/projects/esp-idf/en/latest/.

[3] FreeRTOS. *FreeRTOS API Reference Manual*. Disponible en: https://www.freertos.org/Documentation/RTOS_book.html.

[4] Docker. *Docker Documentation*. Disponible en: https://docs.docker.com/.

[5] Flask. *Flask Documentation*. Disponible en: https://flask.palletsprojects.com/en/2.1.x/.

[6] Raspberry Pi Foundation. *Raspberry Pi Documentation*. Disponible en: https://www.raspberrypi.org/documentation/.

[7] OpenCV. *OpenCV Documentation*. Disponible en: https://docs.opencv.org/.

[8] TensorFlow. *TensorFlow Documentation*. Disponible en: https://www.tensorflow.org/api_docs.

Apéndice A

Apéndice

A.1. Ficheros Docker

Dockerfile:

```
[basicstyle=\ttfamily, language=Docker, keywordstyle=\color{blue},
  commentstyle=\color{green!60!black}, stringstyle=\color{red}, showstringspaces
     =false]
FROM ubuntu:20.04

ENV TZ=Europe/Spain
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
ARG DEBIAN_FRONTEND=noninteractive

# We need libpython2.7 due to GDB tools
RUN : \
  && apt-get update \
  && apt-get install -y \
    apt-utils \
    bison \
    ca-certificates \
    ccache \
    cmake \
    check \
    curl \
    flex \
    git \
    gperf \
    lcov \
    libffi-dev \
    libncurses-dev \
    libpython2.7 \
    libusb-1.0-0-dev \
    make \
    ninja-build \
    vim \
    python3 \
    python3-pip \
```

```
    unzip \
    wget \
    xz-utils \
    zip \
  && apt-get autoremove -y \
  && rm -rf /var/lib/apt/lists/* \
  && update-alternatives --install /usr/bin/python python /usr/bin/python3 10 \
  && python -m pip install --upgrade \
    pip \
    virtualenv \
  && :

# To build the image for a branch or a tag of IDF, pass --build-arg
    IDF_CLONE_BRANCH_OR_TAG=name.
# To build the image with a specific commit ID of IDF, pass --build-arg
    IDF_CHECKOUT_REF=commit-id.
# It is possibe to combine both, e.g.:
#   IDF_CLONE_BRANCH_OR_TAG=release/vX.Y
#   IDF_CHECKOUT_REF=<some commit on release/vX.Y branch>.

ARG IDF_CLONE_URL=https://github.com/espressif/esp-idf.git
ARG IDF_CLONE_BRANCH_OR_TAG=master

ENV IDF_PATH=/opt/esp/idf
ENV IDF_TOOLS_PATH=/opt/esp

RUN echo IDF_CHECKOUT_REF=$IDF_CHECKOUT_REF IDF_CLONE_BRANCH_OR_TAG=
    $IDF_CLONE_BRANCH_OR_TAG && \
    git clone --recursive \
      ${IDF_CLONE_BRANCH_OR_TAG:+-b $IDF_CLONE_BRANCH_OR_TAG} \
      $IDF_CLONE_URL $IDF_PATH && \
    if [ -n "$IDF_CHECKOUT_REF" ]; then \
      cd $IDF_PATH && \
      git checkout $IDF_CHECKOUT_REF && \
      git submodule update --init --recursive; \
    fi

# Install all the required tools
RUN : \
  && update-ca-certificates --fresh \
  && $IDF_PATH/tools/idf_tools.py --non-interactive install required \
  && $IDF_PATH/tools/idf_tools.py --non-interactive install cmake \
  && $IDF_PATH/tools/idf_tools.py --non-interactive install-python-env \
  && rm -rf $IDF_TOOLS_PATH/dist \
  && :

# Ccache is installed, enable it by default
ENV IDF_CCACHE_ENABLE=1

ARG USERNAME
ARG GROUPNAME
ARG USERID
```

```
ARG GROUPID
RUN echo $GROUPID
RUN echo $GROUPNAME
RUN groupadd -g $GROUPID $GROUPNAME && \
    useradd -m -r -u $USERID -g $GROUPNAME $USERNAME

# Builder with depot tools used for building
ARG USERNAME
USER $USERNAME
WORKDIR /home/$USERNAME
RUN git clone https://chromium.googlesource.com/chromium/tools/depot_tools.git .
    depot_tools
ENV PATH="/home/$USERNAME/.depot_tools:${PATH}"

USER root
ARG USERNAME

RUN dpkg --add-architecture i386

ENV GOSU_VERSION=1.10

RUN dpkgArch="$(dpkg --print-architecture | awk -F- '{ print $NF }')" \
       && wget -q -O /usr/local/bin/gosu "https://github.com/tianon/gosu/
           releases/download/$GOSU_VERSION/gosu-$dpkgArch" \
       && chmod +sx /usr/local/bin/gosu

RUN cd /root && git clone https://github.com/Distrotech/flux && cd flux && \
  autoreconf -fi && ./configure --host=arm-linux-gnueabihf && make && make
      install

VOLUME ["/rbn"]

WORKDIR /rbn

COPY entrypoint.sh /usr/local/bin/entrypoint.sh

RUN chmod +x /usr/local/bin/entrypoint.sh
RUN chmod +s /usr/local/bin/gosu

ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]

CMD ["/bin/bash"]

ENV USER_NAME=$USERNAME

WORKDIR /rbn
```

Scripts uso Docker:

```
#!/bin/bash

if [ "$0" = "$BASH_SOURCE" ]; then
    echo "THIS SCRIPT MUST BE SOURCED!"
    exit 1
fi

export DOCKER_DIR=$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)

echo ">>>>>>>>>>>>>>>> AVAILABLE COMMANDS "
echo
echo "   docker_permision --> Add permission to docker after installation
    , use it only if docker throws permission errors"
echo "   docker_build {image} --> BUILD docker image"
echo "   docker_run {image} --> RUN docker image"
echo "   list_docker --> list different docker images ready to build on
   this repository"

list_docker ()
{
    echo ">>>>>>>>>>>>>>>> Available images"
    echo "   - ubuntu20 -> Ubuntu 20.04 image with necessary tools for
       cross compiling for differents architectures"
    echo "   - ubuntu18 -> Ubuntu 18.04 image with necessary tools for
       cross compiling for differents architectures"
    echo "   - ubuntu16 -> Ubuntu 16.04 image with necessary tools for
       cross compiling for differents architectures"
    echo "   - esp32 -> esp-idf enviroment ready to build esp-idf based
       projects"
    echo "   - rust -> debian enviroment ready to build rust projects"
}
# Build docker image
docker_build ()
{
    IMAGE_NAME="$1"
    docker build --build-arg USERNAME="$(id -un)" \
                 --build-arg GROUPNAME="$(id -gn)" \
                 --build-arg USERID="$(id -u)" \
                 --build-arg GROUPID="$(id -g)" \
                 -t $IMAGE_NAME $DOCKER_DIR/$IMAGE_NAME \
}
# Run docker image
docker_run ()
{
    IMAGE_NAME="$1"
    docker run --privileged -it -v $HOME/Projects/:/${USER} $IMAGE_NAME /
       bin/bash
}
```

A.2. Ficheros Servidor

Script principal servidor:

```
#!/bin/python3

#  main.py
#  Date: Apr 17 07:49:03
#  Author: Rubén Garrido
#  Mail: rgarrido.rbn@gmail.com

import base64
from flask import Flask, request

app = Flask(__name__)

@app.route('/photo', methods=['POST'])
def photo():
    content_type = request.headers.get('Content-Type')
    if content_type == 'image/jpeg':
        #img_data = request.get_data() ## without base64
        img_data = request.get_data()
        with open('photo.jpg', 'wb') as f:
            f.write(img_data)
        return 'Imagen recibida y almacenada', 200
    else:
        return 'Se esperaba una imagen JPEG', 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=3000)
```

Script testeo servidor:

```
#!/bin/python3

#  photo_request.py
#  Date: Apr 17 12:06:39
#  Author: Rubén Garrido
#  Mail: rgarrido.rbn@gmail.com

import requests

file_path = 'sample.jpg'

url = 'http://localhost:3000/photo'

# Abrir y leer el archivo de imagen
with open(file_path, 'rb') as f:
    img_data = f.read()

response = requests.post(url, data=img_data, headers={'Content-Type': '
    image/jpeg'})

# Imprimir la respuesta del servidor
print(response.text)
```

A.3. Ficheros microcontrolador

```
#include "httpCamera.h"
#include "cJSON.h"
#include "camClient.h"
#include "ceco_utils.h"
#include "esp_err.h"
#include "esp_http_server.h"
#include "esp_log.h"
#include "img_converters.h"
#include "iotUtils.h"
#include <stdint.h>
#include <string.h>

static const char *TAG = "httpCamera";

#define PART_BOUNDARY "123456789000000000000987654321"
#define PHOTO_LABEL "img_data"
static const char *_STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;
    boundary=" PART_BOUNDARY;
static const char *_STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char *_STREAM_PART = "Content-Type: image/jpeg\r\nContent-
    Length: %u\r\n\r\n";
```

```
esp_err_t rootUriHandler(void *args);
esp_err_t captureUriHandler(void *args);
esp_err_t streamHandler(void *args);
bool registerRootUri();
ceco_camera_fb_t *picture;

HttpServerInterface *mServerIface = NULL;
CameraInterface *mCameraIface = NULL;
char *mHtmlFile = "<h1> INDEX.HTML MUST BE PLACED HERE </h1>";
unsigned char *imageToSend;

esp_err_t streamHandler(void *args)
{
    httpd_req_t *req = (httpd_req_t *)args;

    esp_err_t res = ESP_OK;
    char *part_buf[128];

    res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);

    if(res != ESP_OK)
    {
        return res;
    }

    while(true)
    {
        picture = mCameraIface->pictureTake(mCameraIface);

        if(!picture)
        {
            ESP_LOGE(TAG, "Camera capture failed");
            res = ESP_FAIL;
        }

        size_t hlen = snprintf((char *)part_buf, 128, _STREAM_PART,
           picture->len);
        res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
        res = httpd_resp_send_chunk(req, (const char *)picture->buf,
           picture->len);
        res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(
           _STREAM_BOUNDARY));

        if(picture)
        {
            mCameraIface->pictureReturn(picture);
        }
```

```
        if(ESP_OK != res)
        {
            break;
        }
    }
    return res;
}

esp_err_t photoUriHandler(void *args)
{
    httpd_req_t *req = (httpd_req_t *)args;

    httpd_resp_set_status(req, "307 Temporary Redirect");
    httpd_resp_set_type(req, "image/jpeg");

    ESP_LOGI(TAG, "picture take request fromweb server");

    if(NULL != picture)
    {
        mCameraIface->pictureReturn(picture);
    }

    picture = mCameraIface->pictureTake(mCameraIface);

    if(picture->buf == NULL)
    {
        ESP_LOGE(TAG, "Could not take picture");
        return ESP_ERR_NO_MEM;
    }

    httpd_resp_send_chunk(req, (const char *)picture->buf, picture->len);
    httpd_resp_sendstr_chunk(req, NULL);

    // CHECK_NULL_AND_FREE(imageToSend);

    return ESP_OK;
}

esp_err_t photoInfoUriHandler(void *args)
{
    httpd_req_t *req = (httpd_req_t *)args;

    httpd_resp_set_status(req, "307 Temporary Redirect");
    httpd_resp_set_type(req, "application/json");

    ESP_LOGI(TAG, "picture take request fromweb server");
```

```
    if(NULL != picture)
    {
        mCameraIface->pictureReturn(picture);
    }

    picture = mCameraIface->pictureTake(mCameraIface);

    if(picture->buf == NULL)
    {
        ESP_LOGE(TAG, "Could not take picture");
        return ESP_ERR_NO_MEM;
    }

    CHECK_NULL_AND_FREE(imageToSend);

    imageToSend = base64encode(picture->buf, picture->len);

    cJSON *root = cJSON_CreateObject();

    cJSON *imageBase64 = cJSON_CreateStringReference((char *)imageToSend);
    cJSON_AddItemToObject(root, PHOTO_LABEL, imageBase64);

    httpd_resp_sendstr_chunk(req, cJSON_Print(root));
    httpd_resp_sendstr_chunk(req, NULL);
    cJSON_Delete(root);

    // CHECK_NULL_AND_FREE(imageToSend);

    return ESP_OK;
}

esp_err_t rootUriHandler(void *args)
{
    httpd_req_t *req = (httpd_req_t *)args;

    httpd_resp_set_status(req, "307 Temporary Redirect");
    httpd_resp_set_type(req, "text/html");

    if(mHtmlFile == NULL)
    {
        httpd_resp_send_err(req, HTTPD_500_INTERNAL_SERVER_ERROR, "index.
            html could not be loaded");
        return ESP_ERR_HTTPD_BASE;
    }

    /* Set HTML file content */
    httpd_resp_sendstr_chunk(req, (char *)mHtmlFile);
```

```
    if(imageToSend != NULL)
    {
        httpd_resp_sendstr_chunk(req, "<img src=\""
            );
        httpd_resp_send_chunk(req, (char *)imageToSend, strlen((char *)
            imageToSend));
        httpd_resp_sendstr_chunk(req, "\"id=\"camImage \"/>");
    }

    /* Send empty chunk to signal HTTP response completion */
    httpd_resp_sendstr_chunk(req, NULL);
    return ESP_OK;
}

esp_err_t captureUriHandler(void *args)
{
    httpd_req_t *req = (httpd_req_t *)args;

    ESP_LOGI(TAG, "picture take request fromweb server");

    if(NULL != picture)
    {
        mCameraIface->pictureReturn(picture);
    }

    picture = mCameraIface->pictureTake(mCameraIface);

    if(picture->buf == NULL)
    {
        ESP_LOGE(TAG, "Could not take picture");
        return ESP_ERR_NO_MEM;
    }

    CHECK_NULL_AND_FREE(imageToSend);

    imageToSend = base64encode(picture->buf, picture->len);

    /* Send empty chunk to signal HTTP response completion */
    httpd_resp_sendstr_chunk(req, NULL);

    return ESP_OK;
}
```

```
bool newHttpCameraServer(HttpServerInterface *server, CameraInterface *
    camera)
{
    if(server == NULL || camera == NULL)
    {
        ESP_LOGE(TAG, "Empty parameters on http server camera init");
        return false;
    }

    mServerIface = server;
    mCameraIface = camera;

    mServerIface->startServer(mServerIface);

    if(!registerRootUri())
    {
        return false;
    }

    return true;
}

bool registerRootUri()
{
    ceco_httpd_uri_t rootUri = {
        .uri = "/",
        .method = HTTP_GET,
        .handler = rootUriHandler,
    };

    ceco_httpd_uri_t captureUri = {
        .uri = "/capture",
        .method = HTTP_GET,
        .handler = captureUriHandler,
    };

    ceco_httpd_uri_t streamUri = {
        .uri = "/stream",
        .method = HTTP_GET,
        .handler = streamHandler,
    };

    ceco_httpd_uri_t photoUri = {
        .uri = "/photo",
        .method = HTTP_GET,
        .handler = photoUriHandler,
    };
```

```
    ceco_httpd_uri_t photoInfoUri = {
        .uri = "/photoInfo",
        .method = HTTP_GET,
        .handler = photoInfoUriHandler,
    };

    if(!mServerIface->registerNewUri(mServerIface, &rootUri))
    {
        return false;
    }
    if(!mServerIface->registerNewUri(mServerIface, &captureUri))
    {
        return false;
    }
    if(!mServerIface->registerNewUri(mServerIface, &streamUri))
    {
        return false;
    }
    if(!mServerIface->registerNewUri(mServerIface, &photoUri))
    {
        return false;
    }
    if(!mServerIface->registerNewUri(mServerIface, &photoInfoUri))
    {
        return false;
    }

    return true;
}
```