Universidad de Cantabria

Facultad
de
Ciencias

PROTOTIPO DE SISTEMA DOMÓTICO INTERACTIVO EN PLATAFORMAS DE BAJO COSTE

(Interactive home automation system prototype on low-cost platforms)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Óscar Ortiz Belén

Director: Héctor Pérez Tijero

Co-Director: Mario Aldea Rivas

JUNIO – 2023

Resumen

El rápido avance de las nuevas tecnologías deja obsoletas a sus predecesoras a un ritmo vertiginoso. Sin embargo, esto no implica que solo las tecnologías más recientes sean las únicas capaces de cumplir los propósitos deseados. Las tecnologías más antiguas disminuyen su precio con el tiempo, lo que las hace más asequibles para cualquier comprador.

En este contexto, se presenta EcoTronix, un *software* de código abierto para Linux que aprovecha las plataformas de bajo coste y permite convertir las interacciones implícitas de los usuarios en acciones domóticas configurables. Los usuarios podrán controlar estas acciones a través de reconocimiento facial y de voz.

El sistema se basa en un nodo central al que se conectan de forma jerárquica, a través de una red Wi-Fi, una serie de pequeños dispositivos. Estos, requieren únicamente alimentación y conexión inalámbrica a la red, y pueden funcionar tanto como sensores como actuadores. Es importante destacar que se trata de prototipos, ya que no es posible, por razones temporales, abarcar todas las variables que podrían ser monitoreadas, como la luz, la temperatura, el movimiento, entre otros. El nodo central se encarga del reconocimiento facial y de voz, y ejecuta las acciones deseadas por los usuarios tanto en el nodo local como de forma remota en los dispositivos inalámbricos.

Por último, se ha puesto especial énfasis en la automatización de la instalación y en la facilidad de configuración, con el objetivo de que cualquier usuario, sin necesidad de conocimientos previos, pueda utilizar el sistema siguiendo unos sencillos pasos. Con esto se pretende alcanzar el mayor número de usuarios posible, gracias a la disponibilidad del sistema en un repositorio público en internet.

. . . . . .

PALABRAS CLAVE

Comando; Raspberry Pi; Raspberry Pi Pico W; Reconocimiento facial; Reconocimiento de voz;

Abstract

The rapid advance of new technologies renders their predecessors obsolete at a dizzying pace. However, this does not imply that only the latest technologies can fulfil the desired purposes. Older technologies decrease in price over time, making them more affordable for any buyer.

In this context, we present EcoTronix, an open-source software for Linux that takes advantage of low-cost platforms and allows converting implicit user interactions into configurable home automation actions. Users will be able to control these actions through facial and voice recognition.

The system is based on a central node to which a series of small devices are hierarchically connected via a Wi-Fi network. These require only power and a wireless connection to the network and can function as both sensors and actuators. It is important to note that these are prototypes, as it is not possible, for time reasons, to cover all the variables that could be monitored, such as light, temperature, movement, among others. The central node oversees facial and voice recognition, and executes the actions desired by the users both in the local node and remotely in the wireless devices.

Finally, special emphasis has been placed on the automation of the installation and the ease of configuration, with the aim that any user, without the need for prior knowledge, can use the system by following a few simple steps. This is intended to reach as many users as possible, thanks to the availability of the system in a public repository on the internet.

. . . . . .

KEYWORDS

Command; Raspberry Pi; Raspberry Pi Pico W; Facial recognition; Voice recognition;

Índice de contenido

Índice de tablas

Índice de figuras

1. Introducción

1.1. Motivación

Muchas veces se tiene tecnología olvidada en algún rincón de la casa a la que no se les está sacando provecho. Un ejemplo de esto es la Raspberry Pi, un ordenador a pequeña escala líder en el sector de computadores monoplaca. Debido a su bajo coste, es muy probable que muchos hogares tengan alguna, más antigua o reciente, que haya sido utilizada en el pasado como reproductor multimedia o servidor de archivos, por ejemplo.

En este sentido, surge una necesidad latente de aprovechar al máximo los recursos tecnológicos disponibles en los hogares y buscar nuevas formas de utilizarlos de manera efectiva. Una de las áreas en las que se puede aplicar el uso de la Raspberry Pi es la domótica, que consiste en el control automatizado de los dispositivos del hogar. Estos sistemas inteligentes proporcionan ventajas significativas en términos de seguridad, comodidad, bienestar y eficiencia energética, mejorando la calidad de vida de los residentes en la vivienda. Sin embargo, a pesar de la existencia de diversas soluciones en el mercado, actualmente suelen ser costosas y requieren una inversión considerable, lo que puede dificultar su adquisición para muchos hogares.

Por otro lado, el avance del *software* para plataformas como la Raspberry Pi ha permitido el procesamiento de datos que hace años eran impensables incluso en un ordenador de uso común. Esto incluye capacidades como el reconocimiento facial, gestual y de voz, lo cual ha abierto nuevas posibilidades en términos de interacciones con la tecnología.

En este contexto, se hace referencia a las interacciones implícitas como la capacidad de capturar y procesar información a partir de acciones o comportamientos que no requieren una interacción directa con la tecnología. En lugar de depender exclusivamente de comandos introducidos explícitamente en el computador, el *software* especializado puede analizar datos provenientes de la cámara o el micrófono, permitiendo una respuesta inteligente ante estas señales. Estas interacciones implícitas hacen que la experiencia sea más intuitiva y fluida, ya que el sistema puede anticipar o agilizar las necesidades de los usuarios sin requerir una interacción explícita constante.

Considerando todo lo expuesto, se propone la creación de una aplicación que aproveche al máximo estas Raspberry Pi y haga uso de *software* especializado en el procesamiento de interacciones implícitas. Al aprovechar estos dispositivos ya existentes en los hogares, se evita la necesidad de adquirir equipos dedicados o soluciones comerciales más costosas. Además, al integrar las interacciones implícitas, se logra una experiencia domótica más intuitiva y personalizada, mejorando la eficiencia y comodidad en el hogar.

1.2. Objetivos

La finalidad del proyecto es habilitar la capacidad de realizar acciones domóticas mediante interacciones implícitas de los usuarios, aprovechando las plataformas empotradas y de bajo coste existentes.

Con el fin de asegurar un aplicativo exitoso y satisfacer las necesidades de los usuarios, es fundamental establecer objetivos claros que brinden valor y funcionalidad. Los principales que se pretenden alcanzar con cualquier aplicación son los siguientes:

- Proporcionar una experiencia de usuario óptima, asegurando que la aplicación sea rápida y eficiente en su funcionamiento, permitiendo a los usuarios lograr sus objetivos de manera ágil y sin contratiempos.
- Ofrecer versatilidad y adaptabilidad, brindando a los usuarios un amplio abanico de posibilidades, permitiéndoles personalizar la aplicación según sus necesidades individuales, lo cual es esencial para lograr una experiencia satisfactoria.
- Garantizar la facilidad de uso de la aplicación, con una interfaz intuitiva y amigable que sea accesible incluso para aquellos usuarios sin experiencia técnica, de manera que puedan comprender y utilizar fácilmente todas las funciones principales.

En cuanto a los objetivos específicos del proyecto, se definen dos metas principales:

- Desarrollar la capacidad de reconocer y procesar interacciones implícitas, capturando y comprendiendo acciones o comportamientos de los usuarios sin una interacción explícita, como detección de rostros o comandos de voz.
- Lograr la capacidad de realizar acciones domóticas prototípicas, permitiendo el control automatizado de dispositivos del hogar, como luces, utilizando las interacciones implícitas detectadas previamente.

1.3. Conceptos clave

Además de la motivación y los objetivos del presente trabajo, es necesario establecer algunos conceptos clave que sentarán las bases para comprender en detalle la visión global y el análisis de requisitos y casos de uso que se presentan más adelante. En esta sección, se introducirán brevemente los siguientes conceptos fundamentales y la terminología que se utilizará a lo largo del documento:

- Nodo central → Se refiere a la Raspberry Pi que actúa como núcleo del sistema, proporcionando inteligencia centralizada y capacidad de cómputo de reconocimiento de voz y facial. Es el punto central desde el cual se controlan y coordinan las interacciones y acciones del sistema.
- Sensores y actuadores → Son las funcionalidades de un periférico que se conecta a un microcontrolador y son controlados remotamente mediante comandos desde el nodo central. Los sensores son capaces de recolectar datos del entorno, como temperatura, luz o movimiento. Los actuadores, por

otro lado, son capaces de realizar acciones físicas en respuesta a comandos, como encender o apagar luces, abrir o cerrar puertas, entre otros. Los periféricos pueden ser mixtos, es decir, que incluyan ambas funcionalidades, tanto de sensor como de actuador.

- Dispositivos → Los periféricos no son capaces por sí mismos de realizar ninguna tarea. Para que sus sensores y actuadores puedan trabajar, requieren de un microcontrolador como una Raspberry Pi Pico W. Estos dispositivos recibirán las órdenes del nodo central y controlarán los sensores y actuadores de los periféricos que tiene conectados.
- Reconocimiento de voz → Es el proceso mediante el cual el sistema recibe audio y lo procesa para extraer una frase comprensible. El reconocimiento de voz permite a los usuarios interactuar con el sistema mediante frases asignadas a comandos.
- Reconocimiento facial → Consiste en recibir imágenes y procesarlas para identificar y reconocer caras específicas. El sistema utiliza algoritmos para detectar y reconocer rostros, comparándolos con los ya almacenados para determinar si coinciden.
- Rol → Se refiere a una representación abstracta de una función humana dentro del contexto del sistema domótico. Puede ser un rol familiar, como padre o hijo, un rol laboral, como trabajador o jefe, etc.
- Comando → Es la unidad mínima de trabajo y define la acción específica en el sistema que se debe llevar a cabo. Respecto al destinatario de los comandos, se podrán diferenciar entre locales y remotos.
- Local → Se refiere a un comando que se ejecuta directamente en el nodo central. El destinatario del comando es el mismo nodo central y no se requiere que viaje a través de la red. Los comandos locales son procesados internamente por el nodo central.
- Remoto → Se refiere a un comando que se ejecuta en los periféricos de los dispositivos. Estos comandos deben ser enviados a través de la red para llegar al sensor o actuador específico que se desea controlar. Los comandos remotos permiten la interacción con los periféricos conectados a los dispositivos mediante la red inalámbrica.

En los siguientes apartados, se explorarán en detalle estos conceptos clave y su aplicación en el contexto del proyecto.

1.4. Visión global

La finalización del desarrollo de este proyecto da lugar a EcoTronix, una solución domótica diseñada para adaptarse a hogares, oficinas y otros entornos a un bajo coste.

Su objetivo principal es brindar capacidades personalizadas de automatización controladas exclusivamente desde el nodo central mediante comandos de voz y reconocimiento facial, tal y como se esquematiza en la Figura 1. Los usuarios podrán descargar el *software* desde el repositorio público y, a través de la herramienta

suministrada, instalarlo en el nodo central. Una vez que un primer usuario haya configurado el sistema, estará listo para su uso.

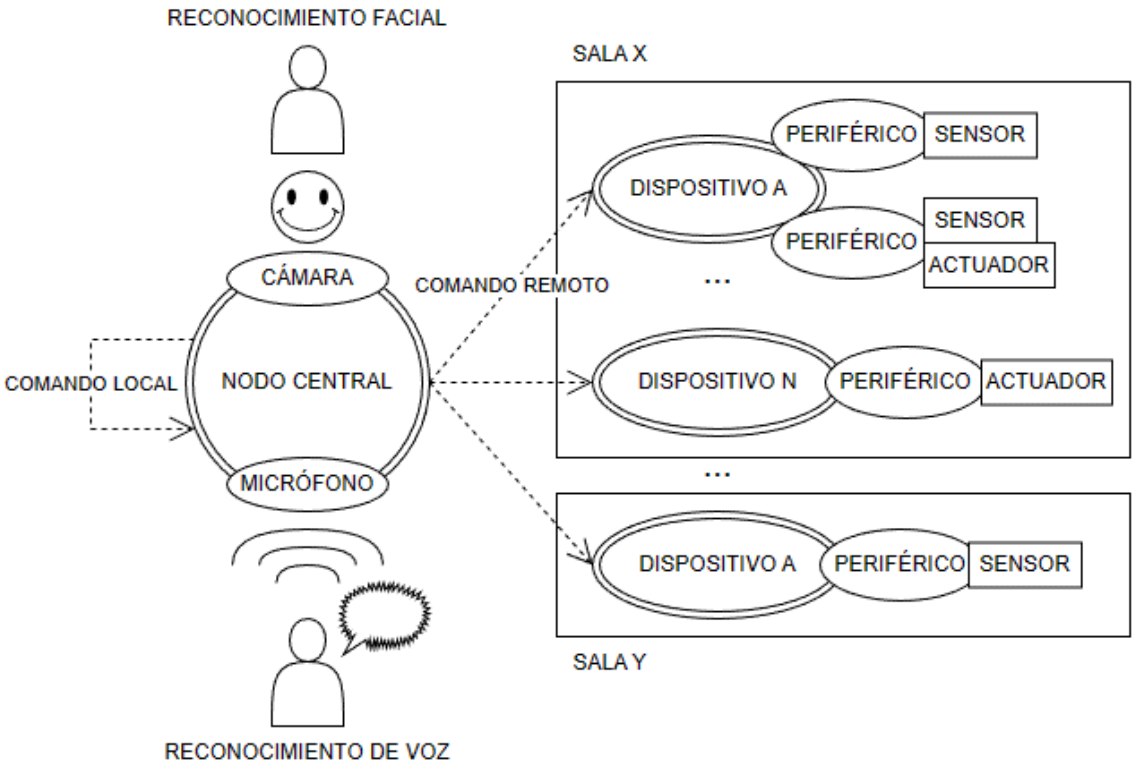

Figura 1. Diagrama de visión global

El nodo central operará como el punto de control principal, monitoreando constantemente los rostros de los usuarios y capturando frases a través del micrófono. Si no se reciben órdenes, el sistema permanecerá en espera. Sin embargo, cuando se detecte una frase asignada a un comando, el sistema verificará si requiere de permisos para llevar a cabo esa acción en particular.

En caso de que el comando requiera de algún permiso, el sistema pondrá el comando en espera hasta que se presente el rostro de un usuario con los permisos adecuados mediante el reconocimiento facial. Una vez que se verifique la identidad del usuario con los permisos necesarios, se ejecutará la acción solicitada.

La asignación de permisos se realiza a través de roles, los cuales agrupan conjuntos de permisos relacionados. Los usuarios se asignan a roles específicos, lo que determina los permisos que tienen dentro del sistema. Esta estructura de roles y permisos garantiza un control de acceso adecuado y una gestión eficiente de los usuarios.

Además, EcoTronix permite a los usuarios ejecutar comandos locales, como programas o *scripts*, que se ejecutan en el propio nodo central, brindando flexibilidad y control directo sobre la Raspberry Pi y sus funcionalidades.

La arquitectura del aplicativo incluye la organización de dispositivos con sensores y actuadores en una jerarquía de salas y posiciones específicas. Esto facilita la identificación y diferenciación de los dispositivos, evitando conflictos entre aquellos con funciones similares en diferentes ubicaciones. Cada microcontrolador encargado de gestionar los sensores y actuadores estará plenamente informado de los periféricos conectados gracias a una configuración inicial y estará a la espera de comandos remotos para atenderlos. El *firmware* y los códigos necesarios se instalarán automáticamente al conectarlos al nodo central mediante la herramienta de configuración suministrada.

En resumen, EcoTronix ofrece una solución intuitiva que simplifica la vida de los usuarios al permitirles automatizar tareas domóticas mediante sus voces y rostros. Su arquitectura basada en salas y posiciones garantiza un control preciso y centralizado de los dispositivos, mejorando la eficiencia y la comodidad en la gestión de la domótica.

2. Análisis de requisitos y casos de uso

Las pautas que debe seguir el sistema son recogidas mediante un proceso de captura de requisitos y clasificadas en dos tipos: funcionales y no funcionales.

Además de la clasificación anterior, es necesario diferenciar dos partes en base al ámbito de uso: subsistema de configuración y subsistema de ejecución.

2.1. Requisitos funcionales

Los requisitos funcionales referentes al subsistema de configuración son recogidos en la Tabla 1.

Identificador	Descripción
RF-C-01	Permitirá crear, editar, eliminar y listar usuarios
RF-C-02	Cada usuario será identificado por un nombre y tendrá una edad y una imagen de su rostro
RF-C-03	Permitirá crear, editar, eliminar y listar roles
RF-C-04	A un rol se le podrá aplicar restricción de edad o privilegios
RF-C-05	Los usuarios se asignarán a roles y podrán pertenecer a varios
RF-C-06	Un usuario mayor de edad no podrá ser asignado a un rol con restricción de edad
RF-C-07	Permitirá crear, editar, eliminar y listar salas y posiciones
RF-C-08	Permitirá instalar, crear, editar, eliminar y listar dispositivos
RF-C-09	Cada dispositivo será identificado mediante las salas y las posiciones
RF-C-10	Permitirá listar periféricos internos y externos y sus acciones posibles
RF-C-11	Los periféricos externos se asignarán a dispositivos concretos
RF-C-12	Permitirá crear, editar, eliminar y listar comandos, tanto locales como remotos
RF-C-13	Cada comando requerirá una descripción, una respuesta y una o varias frases posibles
RF-C-14	Para un mismo comando, cada idioma podrá tener sus propias respuestas y frases
RF-C-15	A un comando se le podrá aplicar restricción de edad o privilegios
RF-C-16	Los permisos de los comandos se tratarán a nivel de rol
RF-C-17	Los comandos remotos requerirán la especificación de la sala, posición en la sala, sensor o actuador y acción a realizar
RF-C-18	Permitirá seleccionar el idioma por defecto para el reconocimiento de voz

Tabla 1. Requisitos funcionales del subsistema de configuración

En cuanto a los del subsistema de ejecución, se recogen en la Tabla 2.

Identificador	Descripción
RF-E-01	Los usuarios ejecutarán comandos desde el nodo central
RF-E-02	Los comandos serán detectados mediante reconocimiento de voz
RF-E-03	El reconocimiento de voz deberá ser capaz de detectar cualquier frase preconfigurada en el idioma por defecto
RF-E-04	El reconocimiento por voz deberá poder funcionar, al menos, para los idiomas del castellano e inglés
RF-E-05	Un comando que requiera de permisos permanecerá a la espera de autorización temporalmente
RF-E-06	Los comandos a la espera de autorización se confirmarán mediante los permisos de los roles a los que pertenece el usuario detectado mediante reconocimiento facial
RF-E-07	El reconocimiento facial deberá ser capaz de detectar el rostro de cualquier usuario previamente configurado
RF-E-08	El propio nodo central ejecutará los comandos locales
RF-E-09	El nodo central se comunicará con los dispositivos configurados para la ejecución de los comandos remotos

Tabla 2. Requisitos funcionales del subsistema de ejecución

2.2. Requisitos no funcionales

Se tomarán en cuenta siete aspectos para definir los requisitos no funcionales: compatibilidad, disponibilidad, interfaz, rendimiento, seguridad, tecnología y usabilidad.

En la Tabla 3 se recogen los referentes al subsistema de configuración.

Identificador	Tipo	Descripción
RNF-C-01	Interfaz	Las configuraciones se realizarán a través de una interfaz gráfica
RNF-C-02	Seguridad	Solo el nodo central y los dispositivos podrán hacer uso de las comunicaciones MQTT mediante credenciales
RNF-C-03	Tecnología	Los comandos locales serán comandos de BASH que podrán lanzar programas o *scripts*
RNF-C-04	Usabilidad	La configuración de sensores y actuadores en las Raspberry Pi Pico W será un proceso automático
RNF-C-05	Usabilidad	El proceso de instalación será automático

Tabla 3. Requisitos no funcionales del subsistema de configuración

Los del subsistema de ejecución en la Tabla 4.

Identificador	Tipo	Descripción
RNF-E-01	Compatibilidad	El sistema funcionará en una Raspberry Pi 3 B
RNF-E-02	Compatibilidad	La Raspberry Pi 3 B funcionará con Raspbian OS
RNF-E-03	Compatibilidad	Los periféricos se usarán en una Raspberry Pi Pico W
RNF-E-04	Disponibilidad	El sistema deberá funcionar sin interrupciones
RNF-E-05	Disponibilidad	El sistema será detenido por el usuario
RNF-E-06	Rendimiento	Se podrán ejecutar acciones de forma consecutiva sin tiempos de espera
RNF-E-07	Rendimiento	El lapso temporal entre el reconocimiento de un comando y su lanzamiento deberá ser inapreciable
RNF-E-08	Seguridad	El sistema funcionará sin conexión a internet
RNF-E-09	Seguridad	Las comunicaciones entre el nodo central y los dispositivos se realizarán encriptadas
RNF-E-10	Tecnología	Ambos reconocimientos, facial y de voz, se procesarán localmente
RNF-E-11	Tecnología	Los comandos remotos serán enviados mediante MQTT

Tabla 4. Requisitos no funcionales del subsistema de ejecución

2.3. Casos de uso

En las Figuras 2 y 3 se muestran los casos de uso del subsistema de configuración y del de ejecución respectivamente.

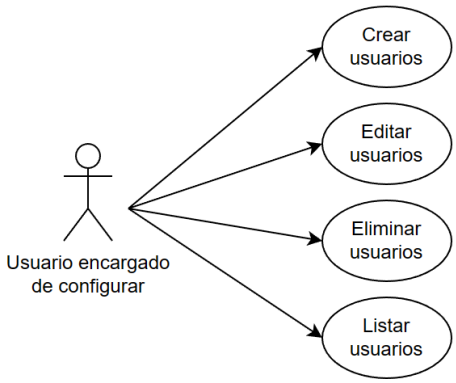

Figura 2. Casos de uso del subsistema de configuración

Respecto al del subsistema de configuración, los mismos casos de uso son aplicables al resto de entidades (roles, salas, posiciones, dispositivos, comandos locales,

comandos remotos), excepto los periféricos, que tan solo será posible listarlos. Su creación o edición supondría la adición y alteración, respectivamente, de las funciones de los periféricos del código base que ejecutará cada uno de los dispositivos, por lo que se descarta esta opción.

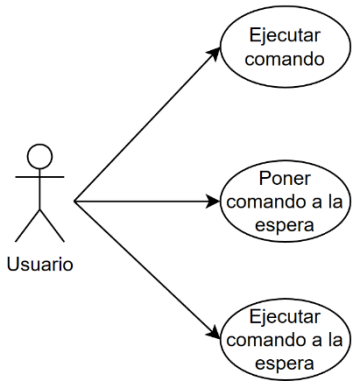

Figura 3. Casos de uso del subsistema de ejecución

En los del subsistema de ejecución, los dos primeros se refieren al reconocimiento de voz y el tercero al facial. Si el comando que trata de ejecutar un usuario (no tiene por qué estar registrado en el sistema mediante el subsistema de configuración) mediante reconocimiento de voz no posee permisos, se ejecutará directamente. En caso contrario, se pondrá a la espera de que el propio usuario u otro del sistema lo valide. Aquí es donde entra en juego el tercer caso de uso, confirmando aquellos comandos pendientes, si el usuario detectado tiene los permisos suficientes para ello.

3. Tecnologías y materiales

En este apartado se definirán tanto las tecnologías y herramientas *software* empleadas como los materiales y *hardware* necesario para llevar a cabo el proyecto.

3.1. Tecnologías

En el despliegue del sistema, se utiliza el lenguaje de comandos BASH, como es habitual en entornos Linux, para instalar los paquetes necesarios que permiten la ejecución del *software*.

Figura 4. BASH

El lenguaje de programación escogido para el desarrollo íntegro del nodo central del proyecto es Python, más concretamente su versión 3. Aunque el rendimiento que proporciona este lenguaje es relativamente bajo con respecto a otros de más bajo nivel, como pueden ser C o Ada, compensa la facilidad de uso y la cantidad y variedad de librerías que ofrece en cualquier ámbito.

Figura 5. Python

MicroPython es la elección ideal para dispositivos que manejan sensores y actuadores, ya que combina la facilidad de uso de Python con un rendimiento óptimo en microcontroladores. Esta implementación optimizada de Python 3, escrita en C, permite ejecutar código en dispositivos con recursos limitados, como microcontroladores, ofreciendo una combinación perfecta entre eficiencia y facilidad de desarrollo.

Figura 6. MicroPython

MQTT o Transporte de Telemetría de Mensajería en Cola, juega un papel fundamental en el proyecto. Se trata de un protocolo de conexión que facilita la comunicación entre los dispositivos emisores y receptores de datos. Su principal ventaja radica en su eficiencia y ligereza, lo cual lo hace especialmente adecuado para entornos con recursos limitados. En el contexto de este aplicativo, MQTT se utiliza para establecer la conexión entre los diferentes dispositivos encargados de enviar información (publicadores) y aquellos responsables de recibirla y procesarla (suscriptores). Permite jerarquizar los sensores y actuadores mediante temas o *topics* y enviarles información mediante mensajes, lo que facilita la organización escalable de los componentes del sistema. Además, MQTT evita la necesidad de realizar programación a bajo nivel utilizando *sockets*, lo cual simplifica el desarrollo y mejora la interacción entre los dispositivos involucrados.

Figura 7. MQTT

Como es común en cualquier *software* que requiere configuraciones, es esencial contar con un mecanismo que permita almacenar y gestionar los datos de forma sencilla, incluyendo su creación, modificación y eliminación. Dado que las bases de datos suelen ser demasiado pesadas para pequeñas cantidades de datos, a excepción de las embebidas, y los archivos de texto plano carecen de estructura, la elección ideal para mantener las configuraciones y datos de la aplicación es utilizar JSON.

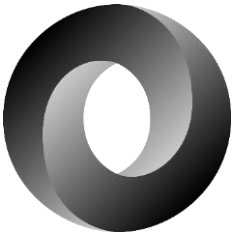

Figura 8. JSON

Para el control de versiones, se utiliza Git, una herramienta ampliamente reconocida en la industria. La plataforma GitHub ofrece un repositorio remoto y público que facilita el almacenamiento del proyecto de manera totalmente *open-source*. Utilizar Git y GitHub de manera conjunta evita la complejidad y la dificultad de trabajar con archivos locales en diferentes versiones, permitiendo una gestión eficiente de las modificaciones, recuperación de código y deshacer cambios. El repositorio se encuentra disponible en la siguiente dirección web: github.com/Osorbe10/EcoTronix.

Figura 9. Git y GitHub

3.2. Materiales

Este apartado es crucial para exponer los materiales necesarios para desplegar el sistema, destacando el bajo costo y la accesibilidad del proyecto. A continuación, se describen los componentes necesarios para garantizar un uso completo del aplicativo.

El nodo central del sistema se basará en una Raspberry Pi 3 B, que es el modelo mínimo requerido. Versiones inferiores no serían capaces de ejecutar el sistema con un buen rendimiento debido a la limitación en la cantidad de núcleos de procesamiento. La Raspberry Pi 3 B [1] cuenta con cuatro núcleos, lo que permite paralelizar el *software* y optimizar su ejecución.

La elección de la Raspberry Pi 3 B se justifica por su capacidad para manejar los requisitos de procesamiento de audio e imagen del sistema. Estas tareas requieren potencia de procesamiento, y los modelos inferiores de Raspberry Pi no ofrecen la suficiente para realizarlas de manera eficiente.

Figura 10. Raspberry Pi 3 B

Por lo tanto, es necesario utilizar una Raspberry Pi 3 B o un modelo superior para asegurar un rendimiento adecuado del sistema y garantizar un funcionamiento fluido y una respuesta rápida del aplicativo.

En cuanto a la cámara utilizada para el reconocimiento facial, se optó por la Raspberry Pi Camera. Aunque existen diferentes gamas y modelos disponibles, incluso el modelo más básico proporciona una resolución adecuada para capturar los rasgos faciales necesarios para la detección. No obstante, es importante tener en cuenta que los modelos más económicos tienden a generar más calor y pueden experimentar fallos después de un uso prolongado. Por esta razón, se recomienda invertir un poco más en una cámara de mayor calidad para evitar problemas de temperatura y riesgos de interrupción en el sistema de reconocimiento facial. Este tipo de cámaras funcionan mediante una conexión al bus CSI de la Raspberry Pi, por lo que el sistema operativo debe habilitar este puerto para poder ser usadas.

En este proyecto en particular, se utilizó el modelo Raspberry Pi Camera NoIR V2 [2], una versión más reciente que ofrece una mayor calidad en comparación con el modelo más básico mencionado anteriormente. Este cambio se realizó para evitar problemas de sobrecalentamiento que se presentaban con el modelo anterior después de unos minutos de uso.

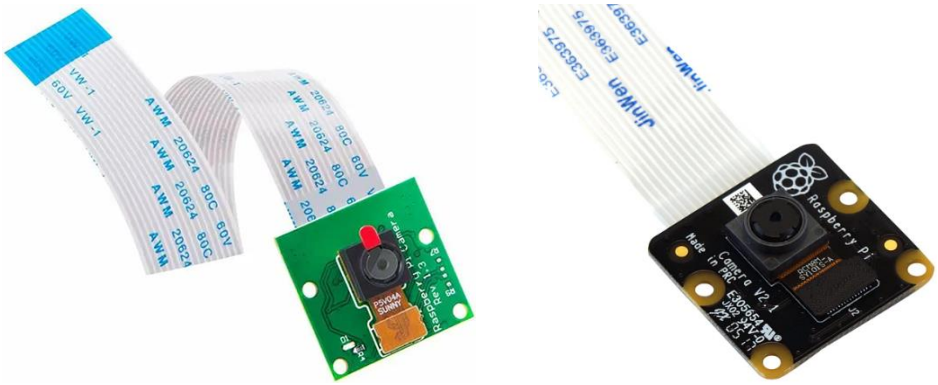

Figura 11. Raspberry Pi Camera y Raspberry Pi Camera NoIR V2

Para el manejo de los diferentes periféricos que contienen sensores y actuadores, se seleccionó la Raspberry Pi Pico W [3]. Este microcontrolador RP2040 cuenta con una alimentación a través de un puerto USB Micro B y dispone de cuarenta pines GPIO para la conexión de periféricos. Además, está equipada con un interfaz Wi-Fi que cumple con los requisitos de ancho de banda necesarios para la tecnología MQTT utilizada en este proyecto. También incluye un botón BOOTSEL que facilita la instalación del *firmware*.

El arranque del dispositivo puede realizarse de dos maneras, dependiendo del estado del botón BOOTSEL al instante de conexión a la alimentación:

- Si el botón está presionado, el dispositivo se puede utilizar como un medio exportable para importar el *firmware* necesario.
- Si el botón no está presionado, el dispositivo se ejecutará utilizando los códigos y programas previamente cargados.

Figura 12. Raspberry Pi Pico W

Para prototipar los elementos del hogar se ha elegido la BerryClip [4], una placa de expansión básica de doce pines GPIO con un costo reducido. Dado que gran parte del tiempo se invierte en el desarrollo del *software*, no ha sido posible abordar una gran cantidad y variedad de *hardware*. Sin embargo, la BerryClip es más que suficiente para simular el control de varias luces de diferentes colores y un zumbador por parte de una Raspberry Pi Pico W.

En un escenario real, el zumbador podría ser, por ejemplo, una alarma que se activa mediante un detector de movimiento, y las luces podrían ser bombillas capaces de adquirir cualquier color. Lo que realmente se necesita es una forma de ejecutar estas acciones, ya que una vez que esto se logre, solo será necesario modificar su comportamiento con otro periférico y añadir tantos cuanto se quiera.

Es importante tener en cuenta que la BerryClip está diseñada específicamente para la Raspberry Pi [5], por lo que la distribución de los pines GPIO en la placa difiere de la de cualquier Raspberry Pi Pico W. Esto implica que no es posible realizar una conexión directa utilizando la base soldada de doce pines, y es necesario utilizar cableado adicional para cada uno de ellos.

Figura 13. BerryClip

A continuación, además de representar visualmente el conexionado en la Figura 14, se muestra en la Tabla 3 una lista de las conexiones mediante *jumpers* entre los pines GPIO de la Raspberry Pi Pico W y los de la BerryClip, en orden descendente,

comenzando con el pin más cercano al primer LED rojo, y excluyendo los pines referentes a los botones:

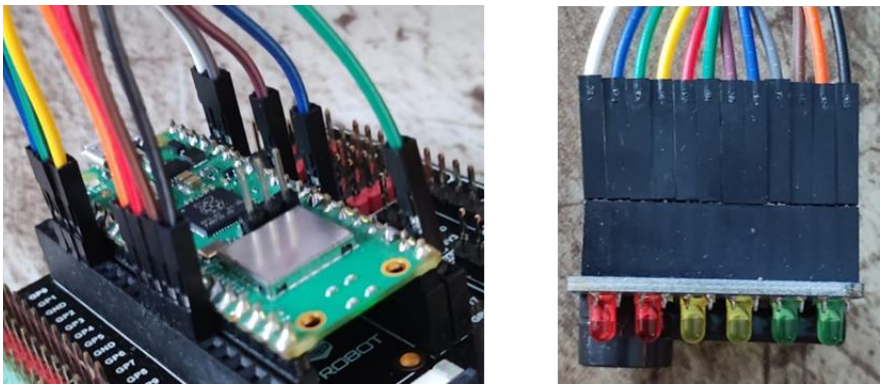

Figura 14. Conexionado entre la Raspberry Pi Pico W y la BerryClip

Pin BerryClip	Pin Raspberry Pi Pico W	Función
0	Alimentación 3V3	
3	4	Zumbador
4	Tierra	
5	17	Segundo LED amarillo
6	27	Segundo LED verde
7	22	Primer LED verde
8	Alimentación 3V3	
9	10	Segundo LED rojo
10	9	Primer LED rojo
11	11	Primer LED amarillo

Tabla 5. Mapeo de pines entre la BerryClip y la Raspberry Pi Pico W

Por último, en cuanto a las entradas y salidas de audio, es importante mencionar que esta segunda es opcional, mientras que la primera es fundamental. Para el micrófono, se recomienda utilizar uno capaz de captar audio en distancias que no estén demasiado cerca de la boca. Cuanto más lejos se capte el audio, mayor será el alcance de detección de las órdenes. Es recomendable evitar los micrófonos de cuello, ya que suelen captar audio solo a una distancia de unos pocos centímetros.

En este caso, se utiliza un micrófono de un auricular Logitech Plantronics, conectado mediante USB. Además, este casco también puede usarse como salida de audio a través de la misma conexión USB.

Si bien podría hacerse una tabla con el material empleado y sus precios, es importante tener en cuenta que el mercado ofrece una amplia gama de distribuidores y precios para cada producto, lo que dificulta obtener un promedio representativo. No obstante, hay que destacar que los costos de los componentes utilizados son relativamente bajos en comparación con las posibilidades que ofrece el sistema.

4. Arquitectura del sistema

En la Figura 15 se muestra una representación gráfica de un sistema de ejemplo que ilustra los componentes y sus relaciones. La Raspberry Pi está representada como el nodo central, rodeada por una nube, lo cual simboliza la centralización y estabilidad de los componentes esenciales, como el micrófono, la cámara y el altavoz (este último es opcional, como se mencionó anteriormente). Además, se puede entender que los comandos locales se ejecutan dentro de la nube, sin depender de la red en el propio nodo central.

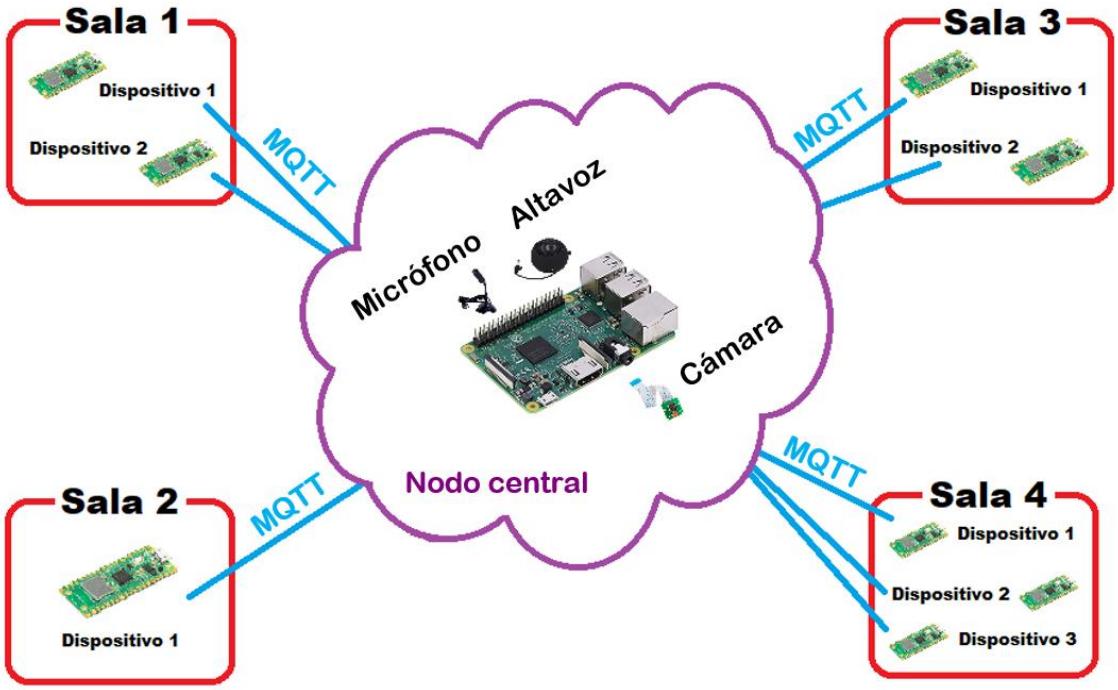

Figura 15. Sistema de ejemplo

4.1. Jerarquización de los dispositivos y comunicaciones MQTT

Los dispositivos de cada sala se organizan en una estructura jerárquica, donde el nodo central actúa como intermediario de las comunicaciones utilizando el protocolo MQTT. A continuación, se describen los niveles de jerarquización en detalle, identificando de manera unívoca cualquier periférico conectado a un dispositivo:

- Primer nivel: las salas. Dependiendo del entorno de uso del sistema, las salas pueden representar las habitaciones de una casa o las áreas de una oficina.
- Segundo nivel: la posición del dispositivo dentro de la sala. Este nivel es esencial para identificar de manera única varios dispositivos ubicados en el mismo primer nivel.
- Tercer nivel: en el contexto de un dispositivo, se requiere distinguir entre las diferentes opciones y funciones de un periférico, pudiéndose considerar las variables físicas asociadas. Por ejemplo, la BerryClip dispone de luces,

zumbador y botones. Por lo tanto, es necesario identificar la variable física específica que se utilizará para un periférico en particular.

- Cuarto nivel: ya identificado unívocamente un periférico, puede ser necesario distinguir entre varias opciones disponibles. Por ejemplo, en el caso de las luces de la BerryClip, se pueden diferenciar por colores. No siempre es necesario este nivel adicional, ya que con tres niveles se puede manejar la mayoría de los periféricos.

Un *topic* en MQTT no es más que una cadena de caracteres que representa un tema en el que publicar datos. Si los cuatros niveles de jerarquización anteriores se juntan en una única cadena usando un separador concreto, es posible componer un *topic* que identifique unívocamente el sensor o actuador deseado.

Ahora bien, ¿cómo diferenciamos entre acciones como encender o apagar, mover o detener, obtener datos, etc.? La solución propuesta hasta ahora solo permite identificar de manera unívoca el sensor o actuador que recibirá la acción proveniente de un comando remoto.

Para lograrlo, junto con la ruta compuesta por los cuatro (o tres en caso de que no haya cuarto nivel) niveles mencionados anteriormente, MQTT envía un mensaje que contiene la información clave, en este caso, la acción deseada.

A modo de ejemplo, y tomando como referencia la Figura 15, supongamos que “Sala 1” representa la cocina. En esta sala, hay dos dispositivos: “Dispositivo 1” y “Dispositivo 2”, ubicados en el fondo y en la entrada, respectivamente. Si deseamos enviar una orden al segundo dispositivo, la jerarquización base del sistema sería “cocina/entrada”. Si queremos interactuar con las luces de la BerryClip, que pueden ser de tres colores (rojo, verde y amarillo), y las acciones posibles son encender, apagar u obtener estado, la siguiente ruta y mensaje definirían el comando remoto deseado para encender el LED amarillo de la BerryClip del dispositivo situado en la entrada de la cocina:

- Ruta: “cocina/entrada/berryclip_led/amarillo”
- Mensaje: “encender”

Con esta estructura jerárquica de rutas y el envío de mensajes, es posible enviar acciones a los dispositivos dentro de cada sala de manera precisa y eficiente. Después de considerar estos aspectos, surgen algunas inquietudes relacionadas con la seguridad: ¿es solo la Raspberry Pi capaz de enviar comandos remotos? En caso de utilizar credenciales, ¿se transmitirán a través de la red y podrían ser leídas en texto plano?

4.1.1. Seguridad en las comunicaciones

Si no se utilizan credenciales, cualquier dispositivo conectado a la red local podría enviar paquetes MQTT. Aunque esto no sea necesariamente un problema, ya que permite la interacción con otras tecnologías como un teléfono móvil, no es el enfoque deseado para esta aplicación en particular. El objetivo es utilizar el reconocimiento facial para determinar si se debe permitir o no la ejecución de comandos remotos en

los dispositivos. Por lo tanto, se generan credenciales aleatorias durante la instalación y se asignan a las Raspberry Pi Pico W junto con los códigos correspondientes. De esta manera, solo la Raspberry Pi y todas las Raspberry Pi Pico W podrán interactuar a través de MQTT.

En la actualidad, la mayoría de las redes Wi-Fi domésticas u organizacionales utilizan al menos el protocolo de seguridad WPA-2, y existen versiones más actualizadas como WPA-3. Estos protocolos proporcionan un nivel adecuado de encriptación para la comunicación en una red Wi-Fi local. El tráfico que se transmite a través de estas redes viaja encriptado, lo que significa que se requiere una clave que se obtiene durante el establecimiento de la conexión para poder leer el tráfico que se envía de un punto a otro.

Dado que EcoTronix utiliza MQTT a través de Wi-Fi, y considerando que está protegida por al menos WPA-2, no es necesario implementar ningún mecanismo adicional de SSL/TLS. La encriptación proporcionada es suficiente para asegurar la confidencialidad de las credenciales MQTT intercambiadas en el momento del establecimiento de la conexión.

4.1.2. Identificación del nodo central

Un último aspecto para tener en cuenta sobre la interacción entre la Raspberry Pi y cada una de las Raspberry Pi Pico W, es cómo los dispositivos pueden reconocer al nodo central, necesario para que actúe como intermediario (*broker*) de MQTT. En este protocolo, son los clientes los que inician la conexión y se conectan al *broker*, lo que significa que la detección se realiza en un único sentido. Por ello, se exploran dos opciones, esquematizándolas en la Figura 16: el uso de una dirección IP estática o el uso de un nombre de *host* con parámetros de red dinámicos.

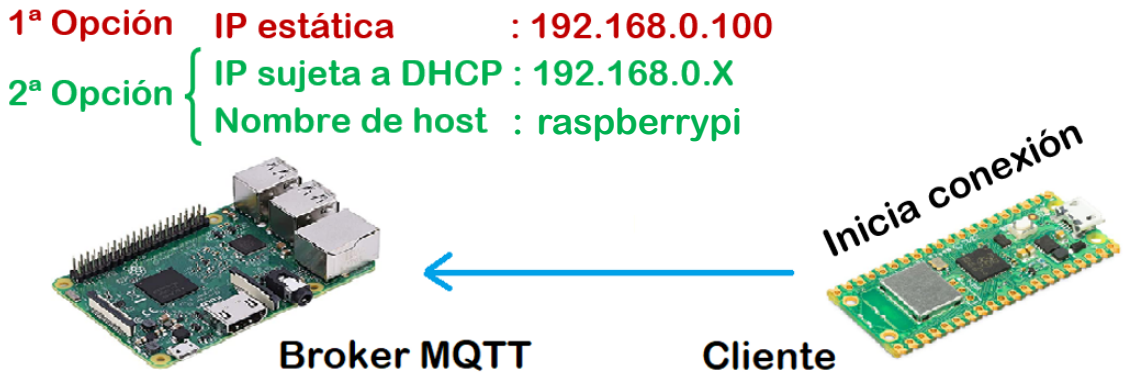

Figura 16. Esquema de establecimiento de conexión MQTT

Se opta por el uso de nombres de *host* en lugar de utilizar direcciones IP, ya que estas pueden cambiar con el tiempo debido al servidor DHCP. En cambio, al emplear un nombre de *host*, este permanece inalterable, evitando la necesidad de reconfigurar los

dispositivos cada vez que el enrutador cambie la IP asignada al nodo central. Además, la alternativa de asignar una dirección IP estática puede resultar complicada para usuarios con poca experiencia en redes. En otras palabras, el nodo central no requiere una dirección IP estática en la red local y puede estar sujeto a parámetros de red dinámicos.

Para que la detección de *hosts* sea posible, es necesario que el enrutador admita esta funcionalidad. Algunos enrutadores, como el LiveBox de la compañía Jazztel, tienen la configuración predeterminada que permite la detección de *hosts* mediante el uso del protocolo mDNS. Sin embargo, en otros casos, puede ser necesario modificar algunas opciones del enrutador o utilizar un servidor DNS externo para habilitar esta función. En ausencia de un servidor DNS en la red local, sería imposible reconocer al nodo central desde los dispositivos utilizando nombres de *host*.

4.2. Procesamiento de voz local

Otra consideración sobre el diseño de EcoTronix trata sobre al reconocimiento de voz, optándose por realizarlo localmente para evitar dependencias de servidores externos.

Esta elección se basa en consideraciones de seguridad y fiabilidad ya que, aunque los datos no contengan información sensible, el envío de audio a un servidor externo puede comprometer datos personales si se captura más audio del necesario. Además, la caída del servidor resultaría en la interrupción del servicio. Por lo tanto, resulta más favorable procesar el audio localmente, aprovechando la capacidad de procesamiento adecuada de la Raspberry Pi 3 B para extraer las frases necesarias.

4.3. Capas de *software*

Una vez explicados los principales principios arquitectónicos y de diseño, es necesario comentar cómo los usuarios interactúan con el sistema. Para ello, se requerirán tres programas diferentes que compondrán la capa de aplicación del nodo central:

- Herramienta de instalación: realizará la instalación de los paquetes y los lenguajes y la preparación del ambiente de ejecución.
- Herramienta de configuración: proporcionará las configuraciones mediante una interfaz gráfica.
- Aplicación EcoTronix: lanzador del aplicativo en sí.

En cuanto a los dispositivos, los usuarios no interactuarán con ello directamente y se proporcionará un único programa.

En la Figura 17 se representan las diferentes capas de *software* que serán usadas en el aplicativo, mostrando en color verde la comentada anteriormente.

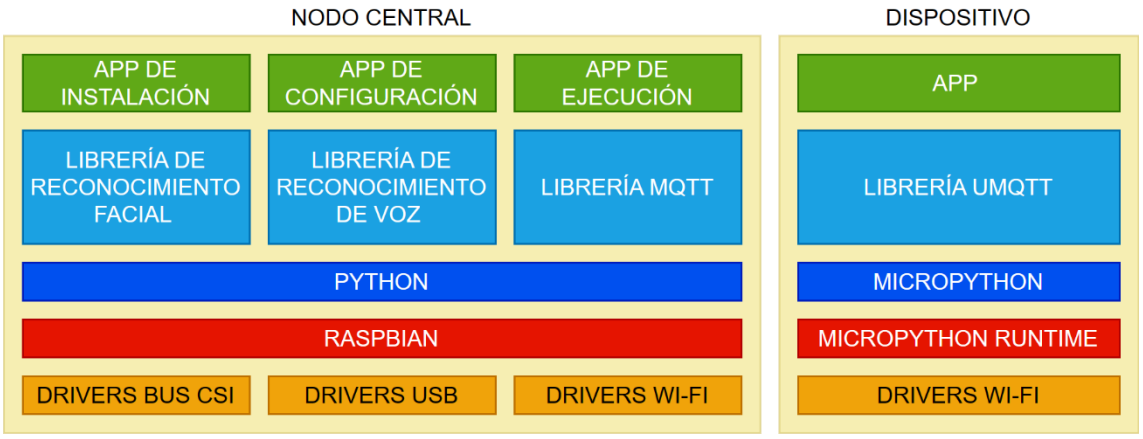

Figura 17. Capas del software

En el siguiente apartado se desgranará más en detalle cada uno de los programas y su interacción con el resto de las capas, más concretamente con las tres inmediatamente inferiores (librerías, lenguaje de programación y sistema operativo o *firmware*).

5. Implementación

En esta sección, se presentará una exhaustiva descripción de la programación del sistema, abordando los aspectos fundamentales de su implementación. EcoTronix consta de aproximadamente cuatro mil líneas de código, lo que hace prácticamente imposible comentar cada uno de los archivos y funciones que lo conforman. Por esta razón, el propio código incluye documentación asociada a cada función, clase o bloque de código, proporcionando especificaciones concretas y detalladas.

5.1. Organización del *software*

El repositorio o directorio de trabajo de EcoTronix contiene una variedad de archivos que desempeñan diferentes funciones en el sistema. Con el objetivo de organizar de manera clara y comprensible los componentes del proyecto, se ha establecido la siguiente estructura:

1. Los programas principales para la ejecución por parte de un usuario:
    - *install.sh*: herramienta de instalación.
    - *config.py*: herramienta de configuración.
    - *start.py*: programa principal.
2. *config.json*: fichero de configuración principal.
3. Directorio *Common*: contiene los archivos Python que gestionan la lógica de interacción con los ficheros de configuración. Cada uno de ellos se encarga de manipular aspectos específicos de ciertas configuraciones, como obtención, listado, creación, edición, borrado o comprobación de atributos. A continuación, se detallan los ficheros contenidos en este directorio:
    - *commands.py*: gestión de los comandos, tanto locales como remotos.
    - *constants.py*: definición de múltiples constantes que son importadas por los programas, como nombres de archivos, rutas, extensiones, formatos, valores numéricos o códigos de color de terminal.
    - *devices.py*: gestión de los dispositivos.
    - *general.py*: configuraciones varias que no se clasifican en los otros ficheros, como credenciales.
    - *languages.py*: gestión de los lenguajes utilizados en el reconocimiento de voz.
    - *peripherals.py*: gestión de los periféricos que serán utilizados por los dispositivos.
    - *positions.py*: definición de las posiciones en la sala de los dispositivos.
    - *roles.py*: definición de los roles de los usuarios.
    - *rooms.py*: gestión de las salas de los dispositivos.
    - *users.py*: gestión de los usuarios y caras utilizadas en el reconocimiento facial.
4. Directorio *Pico*: contiene todo el contenido relacionado con la Raspberry Pi Pico W. Aquí se incluyen los códigos, así como el *script* de instalación, la imagen del *firmware* y dos archivos de configuración.

- Directorio *Codes*: contiene el código fuente principal para la Raspberry Pi Pico W.
  - *main.py*: programa principal iniciado automáticamente al arranque.
  - *umqtt.simple.py*: librería MQTT utilizada.
- *setup.sh*: herramienta de instalación de un dispositivo.
- *config_template.json*: plantilla del fichero de configuración que se adaptará según el dispositivo en el que se importe.
- *peripherals.json*: definición de los periféricos disponibles.
- *micropython.uf2*: *firmware* MicroPython.

Se recomienda descargar la imagen [6] del *firmware* en el momento en que se necesite, en lugar de depender de una copia almacenada localmente. Sin embargo, existe la posibilidad de que la página oficial que aloja el recurso cambie constantemente el identificador de versión de sus *firmwares*, lo cual puede generar problemas de acceso al recurso si se depende de la dirección de descarga. Además, nadie garantiza que las siguientes versiones del firmware sigan siendo compatibles y no generen problemas en base al código desarrollado.

5. Directorio *Faces*: alberga las imágenes con los rostros de los usuarios.
   - Directorio *Encoded*: para cada imagen, se almacena un fichero con su codificación.
6. Directorio *Languages*: contiene subdirectorios correspondientes a cada uno de los lenguajes instalados. Cada subdirectorio incluye modelos de lenguaje, diccionarios y una lista de frases sensibles asociadas.

Una vez situados cada uno de los elementos que componen el sistema, se desarrollan a lo largo de los siguientes apartados.

5.2. *Software* del nodo central

5.2.1. Herramienta de instalación

La instalación de EcoTronix se realiza a través de la terminal de Linux, que sigue siendo la principal vía para gestionar el *software* y el sistema operativo. Para automatizar este proceso, se ha desarrollado un *script* en BASH.

En primer lugar, el *script* verifica que sus permisos de ejecución sean suficientes para llevar a cabo las tareas sin problemas. Si no dispone de acceso de *root* o si el módulo CSI (bus serie de la Raspberry Pi para el conexionado con la Raspberry Pi Camera) está deshabilitado, se detiene el proceso de instalación. A continuación, procede a obtener las dependencias y paquetes APT necesarios.

Una vez completada esta fase, la herramienta descarga la biblioteca dlib desde GitHub y la compila e instala. Es importante destacar que realiza temporalmente una modificación en el espacio de intercambio (*swap*) para asegurar que la instalación de

dlib cuente con suficiente memoria. A continuación, obtiene los paquetes gestionados por PIP.

Si una dependencia ya está satisfecha, se considera instalada y pasa a la siguiente. Todas las dependencias del proyecto se especifican con sus respectivas versiones en el apartado de control de versiones de los paquetes de software.

Posteriormente, instala los paquetes de idioma necesarios para el reconocimiento de voz, en este caso el castellano, pues el inglés funciona por defecto. Se ha decidido instalar estos idiomas en el directorio de trabajo del proyecto en lugar de utilizar el directorio predeterminado del *software* de reconocimiento de voz, más concretamente se crea el directorio *Languages*. Esta elección se basa en posibles problemas al cargar los lenguajes desde el programa principal como usuario no privilegiado, ya que el directorio predeterminado es gestionado por *root*. Además, se realiza una copia del idioma inglés estadounidense por defecto en el nuevo directorio de trabajo, para mantenerlo en una ubicación coherente con los demás idiomas. El procedimiento consiste en descargar el lenguaje desde el servidor web utilizando wget, descomprimirlo con tar, copiarlo al directorio de lenguajes y almacenar las rutas del modelo de lenguaje en el archivo de configuración mediante jq [7]. Previamente, se solicita al usuario que seleccione los lenguajes que desea descargar de una lista mostrada por consola.

Finalmente, la herramienta configura el directorio de trabajo de la aplicación para que todos los programas puedan acceder a los recursos necesarios. Esto implica la creación de los directorios *Faces* y *Faces/Encoded* para almacenar las imágenes con los rostros de los usuarios, sus codificaciones y las credenciales MQTT. Además, configura el *broker* MQTT [8] y establece los permisos de ejecución para los programas de usuario.

Cabe mencionar que, en caso de desear utilizar certificados SSL/TLS, se incluye un bloque de código comentado que permite la generación automática de certificados de servicio, CA y claves privadas. Sin embargo, debido a la implementación de los *sockets* proporcionada por el *firmware* MicroPython en la Raspberry Pi Pico W, no es posible utilizar SSL/TLS en el microcontrolador RP2040, por lo que dicho bloque de código se mantiene comentado para su uso en otros microcontroladores en caso de ser necesario en el futuro.

5.2.2. Herramienta de configuración

En la actualidad, casi todas las aplicaciones complejas que se ejecutan en sistemas operativos con interfaz gráfica cuentan con una interfaz gráfica de usuario (GUI) para gestionar su configuración. Esto evita la necesidad de que los usuarios tengan conocimientos de línea de comandos, lo cual puede resultar engorroso en algunos casos. Sin embargo, en situaciones donde se utiliza un sistema operativo basado en línea de comandos sin interfaz gráfica, el uso de la línea de comandos es inevitable.

En el caso de EcoTronix, se ha decidido utilizar el módulo tkinter [9] de Python, una librería capaz de proporcionar interfaces gráficas sencillas y adecuadas para los

requisitos del proyecto. La capa de presentación que define la interfaz gráfica se encuentra en el archivo *config.py*, el cual implementa ventanas, botones, cajas de texto y otras funcionalidades.

Además de la parte *frontend*, la GUI necesita acceder y manipular datos. Para ello, se utilizan tres ficheros de configuración, el primero principal y el resto auxiliares: *config.json*, *Pico/config_template.json* y *Pico/peripherals.json*. Para más información sobre su estructura ver ANEXO A. Estos archivos almacenan las configuraciones que serán extraídas tanto por la GUI como por la propia aplicación. Además, el contenido del directorio *Common*, desarrollado en la sección de organización del software, se encarga de realizar tareas como la obtención, creación, edición y eliminación de datos, completando así el *backend* de la configuración. En la Figura 18 se esquematizan ambas partes y sus componentes.

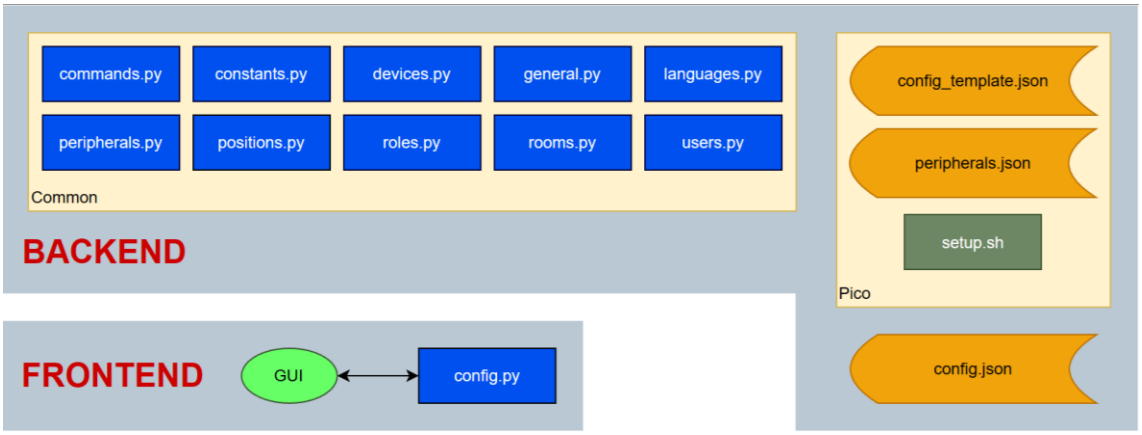

Figura 18. Esquematización de la herramienta de configuración

5.2.2.1. Interfaz gráfica

Es importante destacar que la interfaz gráfica no tiene como objetivo ofrecer un diseño y experiencia de usuario similar a las aplicaciones comerciales. El objetivo principal es poder gestionar las configuraciones de los archivos de manera gráfica, efectiva y sencilla. Cumplir con esta funcionalidad es más que suficiente para el propósito del proyecto.

La GUI consta de una ventana principal que contiene ocho secciones diferentes, tal y como se ilustra cada una de ellas en las Figuras 19 - 26, las cuales se pueden seleccionar mediante pestañas. Cada sección sigue una estructura similar, ofreciendo cajas de texto y menús desplegables para ingresar atributos o seleccionar opciones, así como listas que muestran los datos existentes. A continuación, se detallan las opciones que ofrece y atributos requeridos para cada una de las secciones:

1. ***Menú sección general*** (Figura 19):
    - *Selección del lenguaje por defecto*: permite elegir el idioma en el que se realizará el reconocimiento de voz y configurar las frases y respuestas

específicas para ese idioma. Solo se mostrarán los idiomas instalados y preparados para su uso.

- *Edición de la edad legal*: permite establecer la edad mínima requerida para ciertos comandos mediante el permiso de restricción de edad. Este valor se utilizará para verificar si un usuario cumple con la edad necesaria para omitir la restricción. Dado que EcoTronix puede utilizarse en diferentes entornos y países, este valor numérico debe ser configurable y no estático.
- *Edición de las credenciales Wi-Fi*: permite ingresar y almacenar las credenciales de la red Wi-Fi que serán usadas durante la instalación de una Raspberry Pi Pico W, evitando así la necesidad de introducirlas manualmente para cada instalación. El SSID y la contraseña se almacenan en el fichero de configuración *Pico/config_template.json* y la contraseña se muestra de forma oculta en la caja de texto para mayor seguridad.

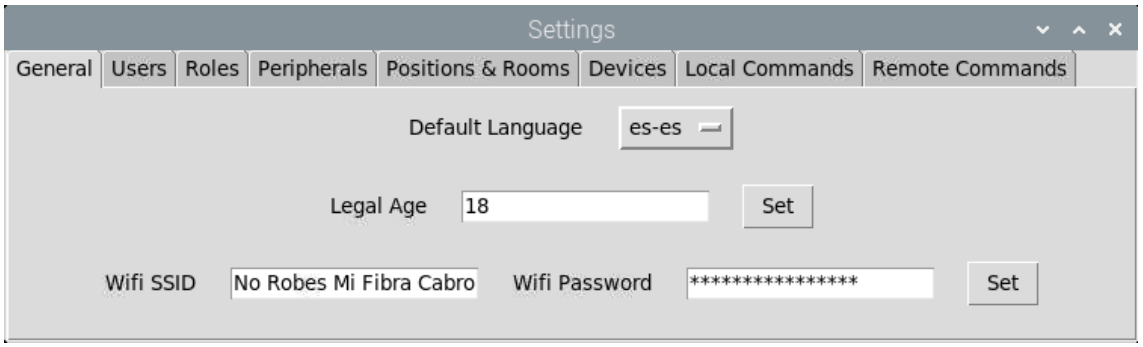

Figura 19. Sección general de la herramienta de configuración

2. ***Menú sección de usuarios*** (Figura 20):
    - *Creación de un usuario*:
        - Nombre: permite ingresar un nombre único para el usuario, con caracteres alfabéticos solamente.
        - Fecha de nacimiento: se ingresa la fecha de nacimiento en el formato "día-mes-año", que es configurable en el fichero *Common/constants.py*.

      Al presionar el botón de crear, se captura una fotografía intentando detectar el rostro del usuario. En caso de que la detección no sea correcta o no se pueda detectar el rostro, se puede repetir el proceso las veces necesarias hasta completar la creación del usuario.

    - *Listado de usuarios*: muestra una lista de usuarios ordenados alfabéticamente por nombre, junto con su edad correspondiente.

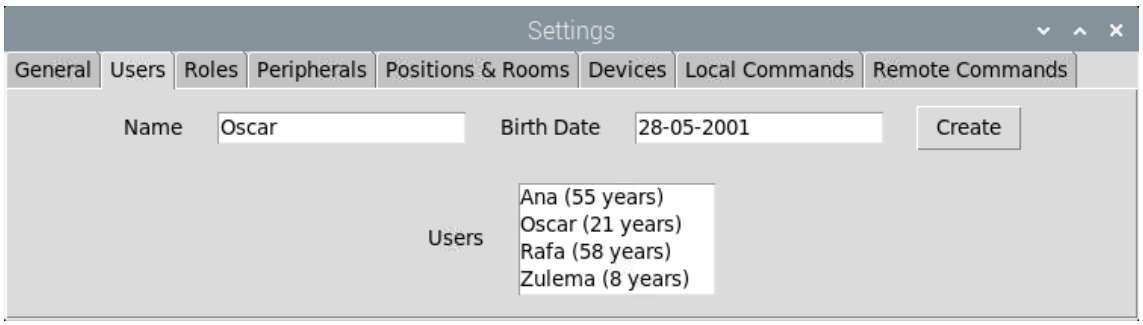

Figura 20. Sección de usuarios de la herramienta de configuración

3. ***Menú sección de roles*** (Figura 21):
    - *Creación de un rol*:
        - Rol: permite ingresar un nombre único para el rol, con caracteres alfabéticos solamente.
        - Selección de permisos: permite elegir los privilegios y la restricción de edad para el rol, con la posibilidad de seleccionar ambos.
    - *Asignación y desasignación de usuarios a roles*:
        - Selección del rol: desplegable que muestra los roles existentes.
        - Selección del usuario: desplegable que muestra los usuarios existentes.

        No es posible asignar un usuario con una edad superior a la edad mínima establecida a un rol con restricción de edad.

    - *Listado de roles*: muestra una lista de roles ordenados alfabéticamente, indicando los permisos y la lista de usuarios asignados a cada rol.

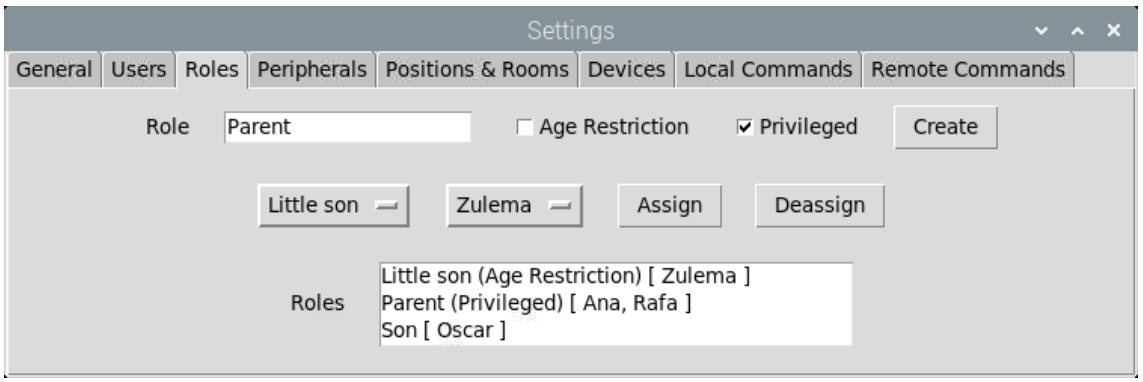

Figura 21. Sección de roles de la herramienta de configuración

4. ***Menú sección de periféricos*** (Figura 22):
    - *Listado de periféricos internos*: muestra una lista de los periféricos instalados en la Raspberry Pi Pico W de forma predeterminada.

- *Listado de periféricos externos*: muestra una lista de los periféricos que se pueden conectar a la Raspberry Pi Pico W a través de los pines GPIO.

No se permite la interacción con estas listas, ya que aparecerán deshabilitadas. Los periféricos internos y externos se muestran en orden alfabético, junto con las acciones posibles. Todos los datos de esta sección se obtienen del fichero *Pico/peripherals.json*. Para definir nuevos periféricos, es necesario modificar dicho archivo de configuración y añadir las funciones correspondientes en el código de la Raspberry Pi Pico W.

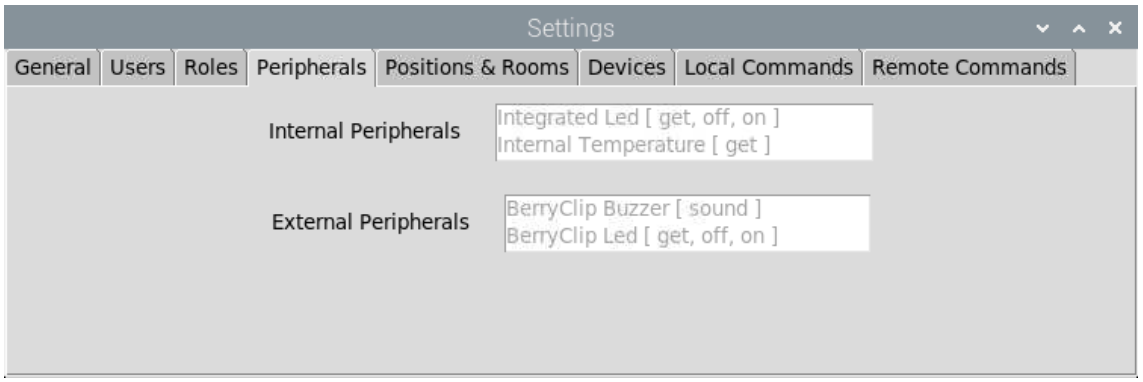

Figura 22. Sección de periféricos de la herramienta de configuración

5. ***Menú sección de posiciones y salas*** (Figura 23):
   - *Definición de posiciones*.
   - *Definición de salas*.
   - *Listado de posiciones*.
   - *Listado de salas*.

Estas definiciones son importantes para componer los temas MQTT, que actúan como rutas e identificadores únicos para los dispositivos remotos.

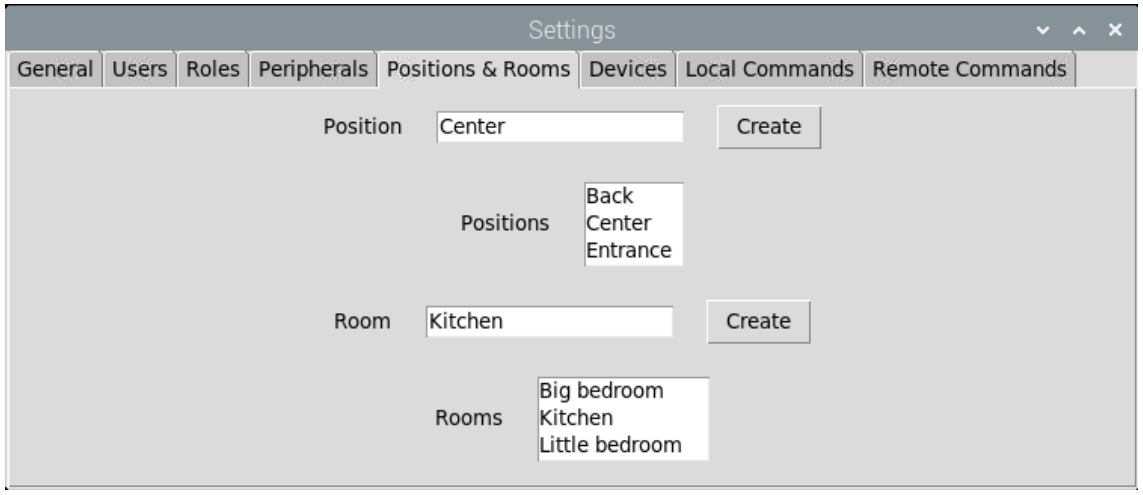

Figura 23. Sección de posiciones y salas de la herramienta de configuración

6. ***Menú sección de dispositivos*** (Figura 24):
    - *Creación de un dispositivo*:
        - Selección de la sala: desplegable que muestra las salas existentes.
        - Selección de la posición: desplegable que muestra las posiciones existentes.
    - *Instalación de un dispositivo*: Permite lanzar el script de instalación de una Raspberry Pi Pico W. Este proceso se explica con más detalle en la sección de la herramienta de instalación de un dispositivo. Se utiliza la misma selección de desplegables que en el proceso de creación.
    - *Asignación y desasignación de periféricos externos a dispositivos*:
        - Selección del periférico: desplegable que muestra los periféricos externos existentes.
        - Selección del dispositivo: desplegable que muestra los dispositivos existentes.
    - *Listado de dispositivos*: muestra un listado de los dispositivos existentes ordenados alfabéticamente por las salas y posiciones. Indica si el dispositivo está instalado o no, así como los periféricos externos asignados a cada dispositivo.

    Cualquier edición o eliminación que suponga la reinstalación del fichero de configuración de un dispositivo supondrá alterar el estado a no instalado automáticamente.

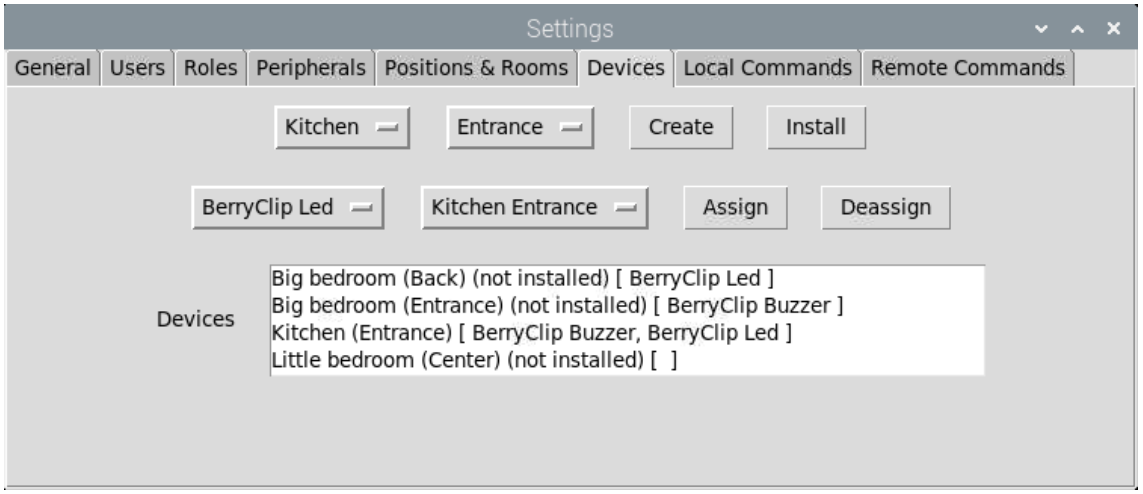

Figura 24. Sección de dispositivos de la herramienta de configuración

7. ***Menú sección de comandos locales*** (Figura 25):
    - *Creación de un comando local*:
        - Comando de BASH: permite ejecutar programas y *scripts*. Esto proporciona flexibilidad en la ejecución de acciones personalizadas.
        - Descripción: proporciona una breve descripción de la funcionalidad principal del comando.

- Respuesta: define la frase que se reproducirá a través del altavoz cuando se lance el comando. La respuesta se basará en el lenguaje por defecto seleccionado y cada lenguaje puede tener una respuesta diferente para el mismo comando.
- Lista de frases: permite definir las frases que el sistema reconocerá como activadoras del comando a través del micrófono. Cualquiera de las frases definidas en esta lista puede usarse para lanzar el mismo comando. Estas frases están asociadas a un único lenguaje. Si se cambia el lenguaje por defecto en la sección general, las frases y respuestas creadas para el idioma anterior se conservarán, pero no se utilizarán con la configuración actual.
- Permisos: permite asignar restricción de edad o privilegios.

- *Listado de comandos locales*: muestra un listado de los comandos locales existentes ordenados alfabéticamente por su descripción. Además, indica los permisos asignados a cada comando.

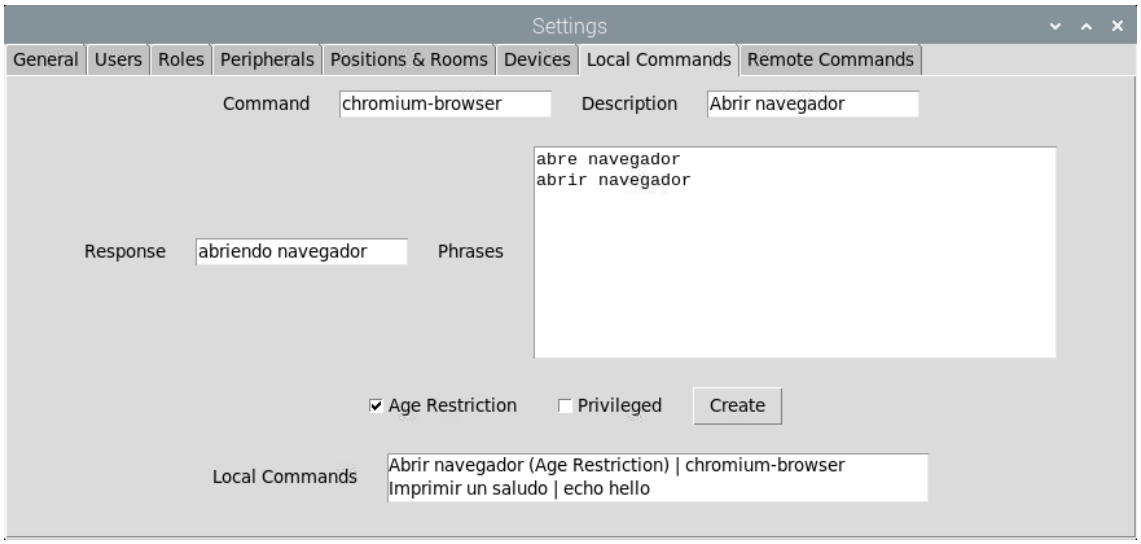

Figura 25. Sección de comandos locales de la herramienta de configuración

8. ***Menú sección de comandos remotos*** (Figura 26):

Esta sección es similar a la anterior, pero en lugar de definir comandos locales, se definen comandos remotos. Los comandos remotos se basan en tres niveles (o cuatro si hay subtipos) y un mensaje para definir una acción remota específica, por lo que se proporcionan cuatro desplegables para seleccionar cada uno de los valores que definen la acción remota:

- El primer desplegable permite seleccionar el dispositivo específico al cual se enviará el comando remoto, unificando los dos primeros niveles de enrutado.
- El segundo y tercer desplegables corresponden al tercer y cuarto nivel de la arquitectura respectivamente. En caso de que no haya subtipos para el tercer nivel, el desplegable se deshabilitará.

- El cuarto desplegable permite seleccionar la acción que se realizará en el periférico seleccionado. Dependiendo del valor seleccionado en el segundo desplegable, las acciones disponibles serán diferentes.

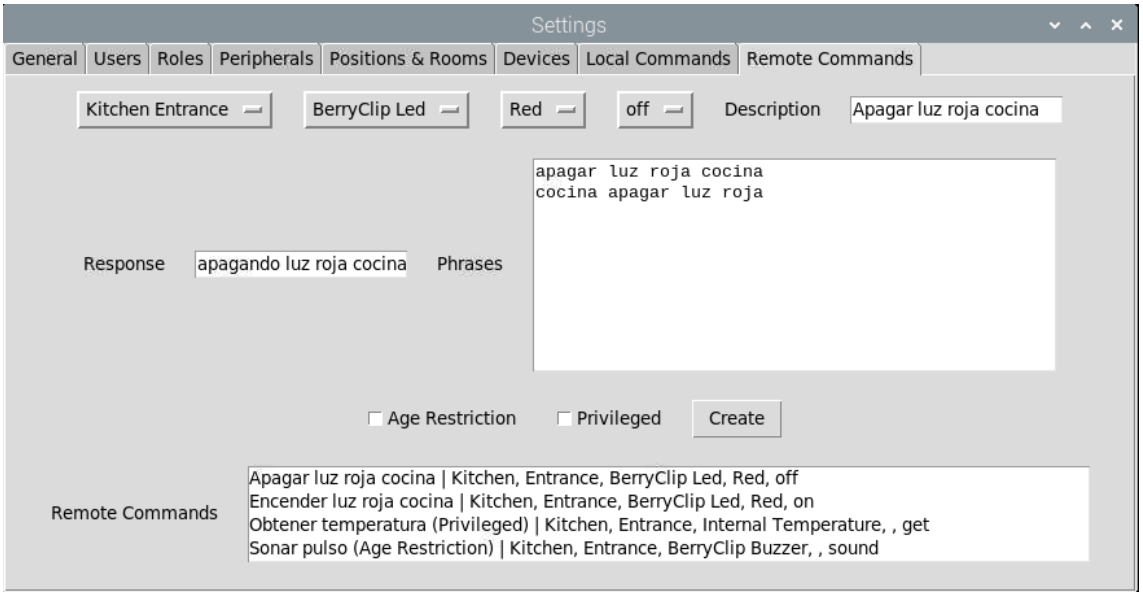

Figura 26. Sección de comandos remotos de la herramienta de configuración

En cada una de las listas de las diferentes secciones (a excepción de la de los periféricos o aquellas que no posean entradas), como se muestra en la Figura 27, es posible eliminar o editar una entrada seleccionada haciendo clic con el botón derecho del ratón después de haberla seleccionado con el botón izquierdo.

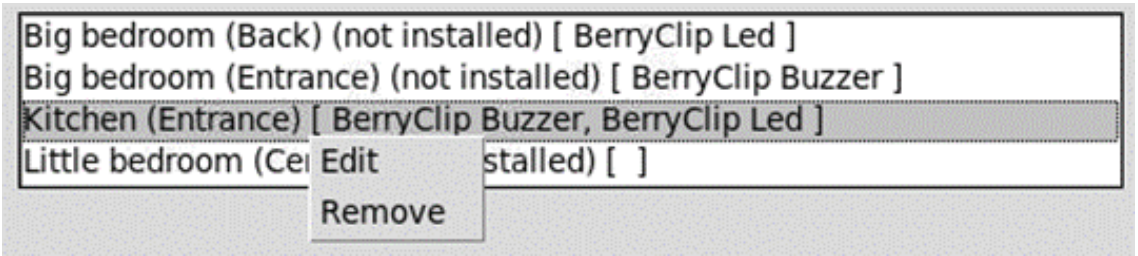

Figura 27. Menú desplegable para editar y eliminar entradas

A continuación, se describen las opciones disponibles:

- La opción de eliminar borra la entrada seleccionada del fichero de configuración, así como todas las entradas dependientes de ella en otras secciones. Por ejemplo, si se elimina una sala, también se eliminarán los dispositivos que estaban ubicados en esa sala, o si se elimina un usuario, se desasignará automáticamente de todos los roles en los que estaba asignado.

- La opción de editar rellena los campos de texto y desplegables con los datos de la entrada seleccionada e inhabilita el botón de creación. Una vez en modo de edición, se habilitan los botones para confirmar la edición y cancelarla. Esto permite diferenciar entre la edición de una entrada existente y la creación de una nueva entrada. Una vez que se ha editado o cancelado una entrada, se vuelve al modo de creación, como se muestra en la Figura 28.

Figura 28. Modo creación y modo edición respectivamente

Para finalizar con la implementación de este programa de configuración, hay que destacar que las salidas de éxito o error para cada cambio realizado en la configuración son mostradas a través de la consola donde se abrió el programa. Es decir, las indicaciones sobre si los cambios se realizaron correctamente o no, se muestran por la salida estándar y no existe una ventana gráfica que lo indique al momento de cambio. Por simplicidad es más que suficiente, ya que la ventana no se proporciona maximizada y el estado de los cambios puede verse en la terminal de fondo maximizada.

5.2.3. Programa principal

Este es el programa que lleva a cabo la operación del sistema una vez instalado y configurado.

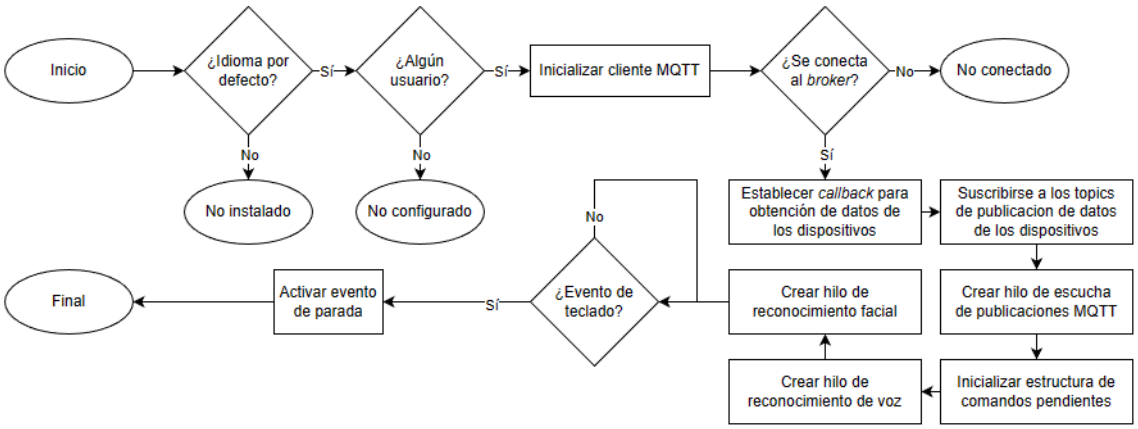

Figura 29. Diagrama de flujo del hilo principal del subsistema de ejecución

Comienza con un único hilo de ejecución, encargado de inicializar todos los recursos necesarios para su funcionamiento. En la Figura 29 se ilustra un diagrama de flujo con los pasos que sigue para su funcionamiento. En caso de que no se haya ejecutado previamente el programa de instalación o no se haya configurado el primer usuario mediante el programa de configuración, se detectará la ausencia de un idioma por defecto o de un usuario, respectivamente.

Posteriormente, se procede a la inicialización del cliente MQTT mediante la librería paho-mqtt [10]. Aunque el *broker* MQTT y este cliente se ejecuten en el mismo nodo, son procesos completamente independientes. El *broker* actúa como intermediario para las comunicaciones MQTT, mientras que el cliente se encarga de publicar mensajes en temas específicos. El cliente MQTT se inicializa utilizando la versión más reciente, MQTT v5, y se identifica con el nombre de *host* de la Raspberry Pi. Una vez creado, se intenta asignar al cliente las mismas credenciales (nombre de usuario y contraseña) que utilizan los dispositivos, y se establece la conexión con el *broker* a través de la interfaz de *loopback* y el puerto predeterminado para conexiones MQTT sin SSL/TLS. En caso de que la conexión no se pueda completar, el programa finaliza. De lo contrario, se define una función de devolución de llamada (*callback*) que muestra por consola los datos devueltos por los sensores de los dispositivos. Por ejemplo, si se solicita la temperatura, el dispositivo correspondiente publicará los grados y mediante esta función de *callback*, la Raspberry Pi los mostrará al usuario.

A continuación, el cliente se suscribe a todos los temas necesarios para obtener datos de los sensores, es decir, que se suscribe a todos aquellos periféricos (o subtipos de periféricos en caso de existir), de dispositivos activos que tengan acciones de obtención de datos o "get". De esta manera, el cliente del nodo central estará capacitado para recibir las respuestas de los sensores a los que les ha solicitado alguna medición o estado.

Una vez suscrito a todos los temas necesarios, se llama a una función de la librería que crea un hilo con un bucle infinito, para que el cliente esté constantemente escuchando las publicaciones en los temas suscritos, es decir, las respuestas de los sensores de los dispositivos. Esta llamada a la función de librería es esencial, ya que, haciéndolo de manera manual, el programa mezclaría toda la lógica restante con MQTT, lo cual debe evitarse en todo momento.

Después, se inicializa de la estructura de datos encargada de gestionar los comandos que requieren confirmación mediante reconocimiento facial. Esta estructura es fundamental, ya que, sin ella, el reconocimiento facial tendría que llevarse a cabo inmediatamente después de detectar un comando, sin posibilidad de mantener esos comandos pendientes a la espera de un usuario autorizado de manera temporal. La estructura proporciona funciones para agregar, obtener y eliminar comandos, así como el número total de comandos pendientes. Su funcionamiento interno es sencillo: una vez inicializada, se crea un hilo que permanece en espera indefinida, y se activa periódicamente (según un parámetro definido en el archivo *Common/constants.py*). Durante cada activación, se eliminan de la lista los comandos pendientes que no han sido autorizados, siempre y cuando haya excedido el valor de *timeout*, utilizando una marca temporal asignada en el momento de agregarlos a la lista.

Por último, queda definir el funcionamiento de las interacciones implícitas en las que se basa el proyecto. Ambos reconocimientos, facial y de voz, se ejecutarán en hilos independientes, lo que permite que cada uno funcione de forma separada, maximizando su rendimiento. Esta implementación no solo aumenta la concurrencia, sino que también aprovecha los núcleos de la Raspberry Pi para obtener el máximo paralelismo posible. En los siguientes apartados se expone el funcionamiento interno de estas piezas clave de interacción implícita con el sistema.

Todo el flujo de estos dos hilos finalizará mediante un evento, es decir, cuando este evento se active, cada uno de los hilos finalizará su flujo de ejecución de manera totalmente controlada. Este evento será activado única y exclusivamente mediante una interrupción de teclado.

5.2.3.1. Reconocimiento de voz

Una de las opciones más populares para el reconocimiento de voz es el uso de un módulo de Python llamado SpeechRecognizer. Este paquete ofrece una *suite* de tecnologías con las que realizar reconocimiento por voz en diferentes ámbitos. Tras probarlo, se llega a la conclusión de que las funciones que ofrece son de muy alto nivel, no adecuadas para lo que realmente se pretende hacer, por lo que es necesario usar una librería aislada que permita funciones más específicas. Además, muchas de esas tecnologías trabajan de forma remota en un servidor y, como se comentó en apartados previos, se busca modelar las frases de los usuarios de manera local.

Por ello, la siguiente alternativa es la de trabajar con una librería que ofrezca funcionalidades de más bajo nivel, como es en este caso pocketsphinx [11]. Este módulo permite especificar diferentes modelos de lenguaje, diccionarios y listas de frases sensibles y es muy cómodo para instalar nuevos idiomas, todo ello además procesando de manera local.

En primera instancia se usa la clase LiveSpeech. Aunque las primeras pruebas de uso fueron satisfactorias, se detectaron limitaciones importantes durante su funcionamiento. Por ejemplo, al tratar de detener el flujo del programa a razón de los hilos en ejecución, y la capacidad de trabajar con las frases detectadas era limitada, ya que es un iterador infinito cerrado. Por ello, se decide proceder a la creación de un decodificador de audio. Este, mediante una configuración inicial (rutas al modelo de lenguaje *Hidden Markov*, a la lista de frases sensibles y al diccionario para el lenguaje configurado por defecto), es capaz de leer un buffer que contiene audio y procesarlo. En la Figura 30 se representa el pseudocódigo del algoritmo utilizado para el reconocimiento de voz.

La recolección del *stream* de audio y volcado en un *buffer* se realiza mediante el módulo PyAudio [12]. Este, produce una serie de *warnings* en consola a causa de la librería ALSA de Linux, por lo que, temporalmente durante la creación del objeto PyAudio, se redirige la salida de errores hacia la salida nula para así no ensuciar la salida estándar con texto que no conviene.

```
hilo de reconocimiento de voz (evento de parada):
    configuración = (ruta del modelo de lenguaje, ruta del diccionario, ruta del fichero de frases sensibles)
    crear decodificador (configuración)
    activar decodificador
    audio = canal de obtención de audio del micrófono 2> /dev/null
    stream de audio = abrir audio (parámetros físicos del muestreo)
    mientras no se active el evento de parada:
        buffer = lectura de stream de audio
        si buffer contiene datos:
            decodificador procesa audio del buffer
            si decodificador detecta hipótesis:
                frase = decodificador obtiene hipótesis
                lanzar comando o añadirlo a los pendientes (frase, lenguaje por defecto)
                desactivar decodificador
                activar decodificador
        en caso contrario:
            el audio se detectó mal
    cerrar decodificador
    cerrar stream de audio
```

Figura 30. Pseudocódigo del hilo de reconocimiento de voz

Con el *stream* de audio abierto, de manera indefinida hasta que se lance el evento de parada desde del *main*, se buscan nuevos datos en el *buffer* del *stream* de audio, y en caso de haberlos, el decodificador lo procesa y extrae la hipótesis, que sería una de las frases configuradas en la lista de frases sensibles. Ya disponiendo de la frase, mediante la misma y el lenguaje por defecto, se extrae el comando asociado. En base a los permisos de los que disponga ese comando se decide si ejecutarlo o añadirlo a la estructura de comandos pendientes a la espera de un rostro que pueda confirmar su ejecución, trabajo realizado por el otro hilo que se desarrollará en el siguiente apartado. En la Figura 31 se indica el pseudocódigo para determinar si se debe lanzar el comando o añadirlo a la estructura de comandos pendientes. Además, dependiendo de si se permite lanzar el comando o añadirlo a la lista de pendientes, suena por el altavoz un sonido distinto.

```
lanzar comando o añadirlo a los pendientes (frase, lenguaje por defecto):
    comando = obtener comando (frase, lenguaje por defecto)
    si no existe comando:
        se reconocieron varias frases válidas en la misma frase
    si comando no requiere de permisos:
        sonar alerta de lanzamiento de comando
        lanzar comando (comando, lenguaje por defecto)
    si no:
        sonar alerta de comando pendiente de confirmación
        añadir comando a la lista de comandos pendientes de confirmación
```

Figura 31. Pseudocódigo para decidir si lanzar un comando o añadirlo a la lista de comandos pendientes

En caso de que el comando no requiera de ningún permiso y pueda ser lanzado, se toman en cuenta dos aspectos para ejecutarlo:

- Si es un comando local: se crea un nuevo proceso independiente que ejecuta exactamente el comando de BASH tal como se encuentra configurado, al igual que si se llamase desde otra terminal.
- Si es un comando remoto: se publica un mensaje en el *topic* compuesto por los atributos del comando remoto, como ya se desarrolló en el apartado de arquitectura del sistema.

Además, se crea un proceso que llama a *espeak*, un programa que recibe una frase y un lenguaje y convierte texto a voz. Este mecanismo es suficiente para reproducir por el altavoz las respuestas previamente configuradas para el comando que se trata de ejecutar, en base al idioma por defecto usado. Todo el mecanismo de lanzamiento de un comando se esquematiza en la Figura 32.

```
lanzar comando (comando, lenguaje por defecto):
    si comando es un comando local:
        crea un proceso que ejecute el comando de BASH
        crea un proceso que ejecute espeak (respuesta del comando, lenguaje por defecto)
    si comando es un comando remoto:
        topic = sala del comando + "/" + posición del comando + "/" + periférico del comando
        si comando tiene subtipos:
            topic += "/" + subtipo del comando
        publicar tema y mensaje en MQTT (topic, acción del comando)
        crea un proceso que ejecute espeak (respuesta del comando, lenguaje por defecto)
```

Figura 32. Pseudocódigo del lanzamiento de un comando

Una vez es recibido el evento de finalización, se liberan los recursos asociados al *stream* de audio y al decodificador.

5.2.3.2. Reconocimiento facial

Para las labores de reconocimiento facial, el módulo face_recognition [13] de Python es el claro ganador, no solo por su rendimiento, sino por las posibilidades y sencillez que ofrece.

Una de las primeras versiones del reconocimiento facial carga todas las imágenes de los rostros de los usuarios y procesa sus codificaciones, haciendo extremadamente lenta la fase de inicialización. Estas codificaciones, basadas en la conversión de la imagen original a un archivo binario de extensión *npy* (trabajo realizado por una función de la librería), no varían, por lo que era posible almacenarlas durante la fase de configuración al crear o editar un usuario. De esta manera, se ahorra todo el tiempo de procesamiento y se delegan estas tareas de codificación a la herramienta de configuración. Esta es la finalidad del directorio *Faces/Encoded*, y su uso hace la inicialización de este hilo de ejecución extremadamente ligera.

Con las codificaciones cargadas en memoria, y al igual que en el reconocimiento de voz, el hilo entra en un ciclo que solo se detiene en caso de que el hilo *main* active el evento de detención, liberando los recursos asociados a la cámara abierta. De manera

indefinida, el módulo de manejo de imágenes CV2 de Python, es usado para obtener imágenes mediante la cámara. En caso de no poder obtener un *frame*, como por ejemplo a causa del sobrecalentamiento de una Raspberry Pi Camera de mala calidad como se habló en el apartado de materiales, se detiene el flujo de ejecución del reconocimiento facial, pero sigue siendo posible seguir usando el reconocimiento de voz, aunque solo para comandos libres de permisos. Al ser flujos de ejecución diferentes, un problema en uno de los hilos no tiene por qué afectar al otro si no es estrictamente necesario finalizar el programa.

Ya obtenido el *frame*, mediante una serie de funciones que ofrecen las librerías face_recognition y CV2, se realizan las siguientes funciones:

- Se redimensiona el *frame* a un cuarto de su tamaño original para aumentar la velocidad de procesamiento y transforma de BGR a RGB.
- Se localizan y codifican los rostros en el *frame*.
- Se comparan los rostros codificados del *frame* con los rostros almacenados ya codificados.

Una vez reconocido un usuario, se busca en la lista de comandos pendientes si alguno de ellos puede ser aceptado y por consiguiente lanzado dados los permisos de los que dispone el usuario mediante los roles a los que pertenece. En caso negativo, el comando permanece a la espera (siempre que aun esté dentro del *timeout*) y se pasa al siguiente. En caso positivo, el procedimiento de lanzamiento de comando es el mismo que se explicó en el apartado anterior mediante la Figura 32, componiendo una ruta MQTT y mensaje si es comando remoto o creando un proceso si es comando local, además de la correspondiente respuesta por voz.

Además, dada la previsible adición de carga de perfiles y comandos preasignados a un usuario, se implementa mediante una lógica sencilla una solución al problema que pudiera surgir al detectar un usuario múltiples veces seguidas. Esto ocasionaría la repetición innecesaria de acciones tras detectar el mismo rostro una y otra vez, ya que mientras no se detecte un nuevo usuario diferente del anterior, el estado no debería variar. Todo el funcionamiento del hilo explicado en este apartado se recoge en el pseudocódigo de la Figura 33.

Se puede suponer que no es necesario disponer del reconocimiento facial activado en todo momento, siendo solo necesario cuando haya un comando pendiente de confirmación. En realidad, mediante este mecanismo, es posible cargar perfiles y lanzar acciones por defecto asociadas a un usuario. Es decir, que, aunque no se lancen comandos implícitamente mediante voz, permite lanzar comandos predeterminados asociados a un usuario con tan solo detectar su rostro. Esta funcionalidad no ha sido implementada en la versión actual de EcoTronix, y su desarrollo se deja para trabajo futuro.

```
hilo de reconocimiento facial (evento de parada):
    para cada uno de los usuarios:
        añadir a la lista de nombres de usuario el usuario actual
        añadir a la lista de codificaciones la codificación de la imágen de la cara del usuario actual
    crear cámara
    mientras no se active el evento de parada:
        frame = leer cámara
        si no frame:
            no se pudo leer la cámara
        redimensionar frame y transformar la composición de color
        localizar y codificar rostros en el frame
        para cada uno de los rostros codificados del frame:
            si coincide el rostro codificado del frame con alguno de los almacenados:
                si último usuario detectado != usuario del rostro:
                    cargar perfil del usuario del rostro
                si usuario del rostro tiene privilegios o no tiene restricción de edad:
                    para cada uno de los comandos pendientes:
                        si usuario del rostro está autorizado para ejecutar el comando pendiente:
                            sonar alerta de lanzamiento de comando
                            lanzar comando (comando pendiente, lenguaje por defecto)
                            eliminar comando de la lista de pendientes
    cerrar cámara
```

Figura 33. Pseudocódigo del hilo de reconocimiento facial

5.3. *Software* de los dispositivos

5.3.1. Herramienta de instalación

El proceso de instalación del *firmware* y los códigos necesarios en la Raspberry Pi Pico W se lleva a cabo a través de la configuración de EcoTronix, más concretamente en la sección de dispositivos. Al seleccionar el dispositivo mediante los desplegables y presionar en el botón de instalación, se inicia la ejecución del *script* BASH *setup.sh*. Es importante examinar en detalle este pequeño programa, ya que presenta particularidades y funcionalidades exclusivas.

En primer lugar, se solicita conectar la Raspberry Pi Pico W a la Raspberry Pi presionando el botón BOOTSEL, tal como se explicó en la sección de materiales. Este proceso aprovecha que, al conectar la Raspberry Pi Pico W [14] como un medio extraíble, se crea un nuevo directorio en /media/${USER}/RPI-RP2 (siendo ${USER} el usuario Linux que ejecuta el programa de configuración). El script espera indefinidamente la creación de este directorio. Una vez detectado, se guarda el estado actual de los puertos de la Raspberry Pi en un archivo temporal, que será utilizado posteriormente para determinar el puerto al que está conectado el dispositivo.

A continuación, se copia el *firmware* desde el directorio Pico al dispositivo, simulando una operación de copia en un medio extraíble. Después de esta operación, la Raspberry Pi Pico W deja de ser reconocida como un medio extraíble y se convierte en una conexión USB serie gracias al *firmware*. En este punto, se compara el estado actual de los puertos con el estado previamente guardado en otro archivo temporal. Cuando se detecta una diferencia entre ambos estados, se identifica el nuevo puerto resultante de esta diferencia y se procede a eliminar los archivos temporales.

Por último, utilizando la herramienta *rshell*, una consola remota que permite trabajar con el sistema de archivos del dispositivo sin acceso local, se establece una conexión basada en el puerto exacto al que está conectada la Raspberry Pi Pico W. Una vez establecida la conexión, se borran los contenidos existentes en el sistema de archivos y se copia todo el directorio de códigos en la Raspberry Pi Pico W, incluyendo tanto los dos ficheros MicroPython como el fichero de configuración modificado y basado en la plantilla. Es importante tener en cuenta que las copias largas pueden presentar problemas debido a la falta de espacio en los búferes. Por esta razón, se recomienda realizar el proceso de copia en tres pasos, separando los contenidos, para evitar cualquier inconveniente.

5.3.2. Programa principal

Una de las particularidades del *firmware* MicroPython es que, al detectar un archivo llamado *main.py* en el sistema de ficheros, este se ejecuta automáticamente al momento de conectar el dispositivo a la alimentación. Aprovechando esta característica, se puede emular una funcionalidad *Plug and Play*, asegurando que el programa se inicie instantáneamente al conectar el dispositivo a una fuente de energía.

El archivo *umqtt/simple.py* [15] es una biblioteca liviana que implementa funcionalidades MQTT de bajo nivel. Estas funcionalidades, como la creación de un cliente MQTT, establecer conexiones, suscribirse y publicar en un tema, y verificar si hay nuevos mensajes, serán suficientes que el dispositivo pueda interactuar mediante este protocolo.

Los dos ficheros comentados anteriormente, junto con el archivo de configuración JSON manejado mediante el módulo ujson [16], serán responsables de llevar a cabo el funcionamiento de cada uno de los dispositivos.

El fichero *main.py*, cuyo pseudocódigo puede encontrase en la Figura 34, comienza inicializando tanto la conexión Wi-Fi como un cliente MQTT. Para ello, obtiene las credenciales Wi-Fi (nombre de red SSID y contraseña) del archivo de configuración *config.json* y espera una conexión exitosa. En caso de no recibir respuesta, permanecerá en espera hasta que sea posible establecer la conexión, ya que sin una red Wi-Fi disponible, no tendría sentido continuar con los siguientes pasos. Para realizar esta conexión, MicroPython proporciona un módulo de red [17] sencillo con unas funciones que facilitan el proceso.

Una vez establecida la conexión Wi-Fi, se procede a crear el cliente MQTT utilizando las credenciales MQTT (nombre de usuario y contraseña) y el nombre de *host* que se encuentran también en el fichero de configuración. Además, para crear el cliente, ha sido necesario proporcionar un identificador de cliente. En este caso, ha sido los dos primeros niveles de la ruta o *topic*, sala y posición. De esta manera se le identificará al dispositivo unívocamente.

```
si no se está conectado al Wi-Fi:
    mientras no se esté conectado al Wi-Fi:
        tratarse de conectar al Wi-Fi (SSID, contraseña Wi-Fi)
identificador de cliente = sala + "/" + posición
cliente = crear cliente (identificador de cliente, nombre de host del broker, usario MQTT, contraseña MQTT)
definir función de callback que procesará los mensajes MQTT a los que se está suscrito
mientras no se este conectado al broker MQTT:
    tratarse de conectar al broker MQTT
encender led integrado
para cada uno de los periféricos:
    si el periférico tiene definidos subtipos:
        suscribirse a identificador de cliente + "/" + periférico + "/" + subtipo
    en caso contrario:
        suscribirse a identificador de cliente + "/" + periférico
de manera indefinida:
    comprobar si hay nuevos mensajes MQTT
    en caso de desconexión:
        apagar led integrado
        mientras no se este conectado al broker MQTT:
            tratarse de conectar al broker MQTT
        encender led integrado
        para cada uno de los periféricos:
            si el periférico tiene definidos subtipos:
                suscribirse a identificador de cliente + "/" + periférico + "/" + subtipo
            en caso contrario:
                suscribirse a identificador de cliente + "/" + periférico
```

Figura 34. Pseudocódigo del flujo de ejecución del programa de los dispositivos

Con el cliente creado, se configura una función de *callback* para que cada vez que se detecte un nuevo mensaje MQTT entrante, se llame a esta función como si de una interrupción se tratase. Esta, a su vez, en base a la acción indicada en el mensaje y al periférico indicado en el último nivel del *topic*, deducirá que función concreta que maneje el periférico debe llamar. En la Figura 35 se detalla su funcionamiento, destacando la capacidad de añadir nuevos periféricos mediante sus funciones asociadas.

Posteriormente, ya creada la estructura del cliente, se procede a la conexión con el *broker* MQTT. Al igual que con la red Wi-Fi, si no es posible conectarse, permanecerá a la espera. Una vez conectado, se encenderá el led integrado para indicar que ya está conectado al nodo central.

Establecida la conexión no es posible permanecer a la espera de mensajes si antes no se suscribe a ningún *topic*. Por ello, se procede a la lectura de todos los periféricos asignados en el dispositivo en el fichero de configuración y se suscribe a ellos, incluyendo los internos y externos. En caso de tener subtipos como los leds, no se suscribiría al periférico, sino a cada uno de los subtipos del periférico. Ya inicializado todo lo necesario, se entra en un bucle infinito de escucha de mensajes sobre los *topics* suscritos.

En caso de caída del bróker MQTT, el dispositivo tratará de conectarse de manera indefinida. Lógicamente, si estos dispositivos se sitúan en lugares de difícil acceso, como por ejemplo el techo, no tendría sentido tener que conectarlos y desconectarlos de la alimentación cada vez que se quieran usar. Por ello, podrán permanecer a la espera de la reconexión de manera indefinida bajo un mínimo gasto energético.

```
callback (topic, mensaje):
    dividir topic en tres niveles, o cuatro si presenta subtipos
    si tercer nivel == "berryclip_led":
        berryclip_led(cuarto nivel, mensaje)
    si tercer nivel == "berryclip_buzzer":
        berryclip_buzzer(mensaje)
    si tercer nivel == "integrated_led":
        integrated_led(mensaje)
    si tercer nivel == "internal_temperature":
        internal_temperature(mensaje)

def integrated_led(acción):
    si acción == "obtener":
        publicar en el topic identificador de cliente + "/" + periférico + "/" + "obtener"
    si acción == "apagar":
        apagar led integrado
    si acción == "encender":
        encender led integrado

def internal_temperature(acción):
    ...

def berryclip_buzzer(acción):
    ...

def berryclip_led(subtipo, acción):
    ...
```

Figura 35. Pseudocódigo de la función de callback del programa de los dispositivos

5.4. Control de versiones de los paquetes de *software*

En las tablas 4, 5 y 6 se presenta todo el *software* utilizado a lo largo del proyecto acompañado de su versión específica.

Paquete APT	Versión	Necesidad
espeak	1.48.15+dfsg-2	Conversión de texto a voz
jq	1.6-2.1	Manejo de ficheros JSON en BASH
libatlas-base-dev	3.10.3-10+rpi1	Requerida por *dlib*
mosquitto	2.0.11-1	Demonio del *broker* MQTT
python3.9	3.9.2-1+rpi1	Lenguaje utilizado
python3-opencv	4.5.1+dfsg-5	Requerida por face-recognition
python3-pip	20.3.4-4+rpt1+deb11u1	Descarga de los módulos de python
python3-tk	3.9.2-1	Interfaz gráfica de la configuración
tar	1.34+dfsg-1	Descompresión de lenguajes
wget	1.21-1+deb11u1	Descarga de lenguajes

Tabla 6. Listado de paquetes APT

Paquete PIP	Versión	Necesidad
chime	0.5.3	Sonidos de alerta y notificación
face-recognition	1.3.0	Reconocimiento facial
pocketsphinx	5.0.0	Reconocimiento de voz
PyAudio	0.2.13	Lectura de audio del micrófono
rshell	0.0.31	Importado de códigos a la Raspberry Pi Pico W
paho-mqtt	1.6.1	Cliente MQTT en la Raspberry Pi

Tabla 7. Listado de paquetes PIP

Resto del *software*	Versión	Necesidad
dlib	19.24.0	Librería de procesamiento gráfico
Raspbian GNU/Linux 11 (bullseye)	6.1.21-v7+	Sistema operativo para la Raspberry Pi
MicroPython para RP2	1.19.1-995-g0a3600a9a	*Firmware* para la Raspberry Pi Pico W

Tabla 8. Listado del resto de software

6. Pruebas

6.1. Pruebas de rendimiento

La Raspberry Pi 3 B puede presentar limitaciones en términos de rendimiento y recursos disponibles. Aunque EcoTronix se diseñó para ser la única aplicación en ejecución, es importante tener en cuenta que esta plataforma no puede soportar cargas muy exigentes. Por lo tanto, es crucial evitar sobrecargar la Raspberry Pi para garantizar un rendimiento adecuado y una buena experiencia para los usuarios.

Para evaluar el rendimiento y evitar retrasos perceptibles, se utiliza una herramienta llamada *stress*. Esta aplicación aplica cargas específicas a los componentes de la Raspberry Pi, como la CPU, la memoria, la entrada/salida y el disco (en este caso, la microSD). Unos parámetros adaptados a las capacidades de la Raspberry Pi 3 B y que podrían ser perfectamente reales por cualquier aplicación de hoy en día, considerando que tiene cuatro núcleos y 1 GB de memoria (compartida entre la memoria del sistema y la memoria de la GPU), son los siguientes:

- Dos hilos de ejecución.
- Dos operaciones de entrada/salida.
- Dos operaciones de memoria virtual de 128 MB cada una.
- Dos operaciones de disco de 16 MB cada una.

Durante la ejecución de EcoTronix, se realizan cargas en los recursos del sistema utilizando el comando *stress --cpu 2 --io 2 --vm 2 --vm-bytes 128M --hdd 2 --hdd-bytes 16M*. A lo largo de esta ejecución, no se observan retrasos significativos que afecten la usabilidad del sistema en cuanto al lanzamiento de comandos.

Es importante tener en cuenta que, aunque EcoTronix es capaz de lanzar los comandos solicitados de manera inmediata, en condiciones adversas de disponibilidad de recursos, no se garantiza que los comandos locales se ejecuten instantáneamente. Esto se debe a que estos comandos son lanzados como procesos por el sistema operativo y quedan fuera del alcance directo del aplicativo, dependiendo de la disponibilidad de recursos para su uso.

6.2. Pruebas de uso

Se han llevado a cabo una serie de pruebas básicas para validar el funcionamiento completo y correcto de cada uno de los programas suministrados:

- Para verificar la instalación de EcoTronix, se ha utilizado una microSD con Raspbian idéntico al utilizado durante el desarrollo del proyecto, pero en un entorno separado.
- En el programa de configuración, se han creado, editado y eliminado múltiples entradas. Se ha comprobado que todos los desplegables y listas se actualizan correctamente al crear, editar o eliminar ciertas entradas.

- En el programa principal, se ha verificado que permite la ejecución secuencial de comandos y valida aquellos que requieren permisos mediante reconocimiento facial. Si el micrófono captura dos frases válidas como una sola, se anulan ambas (la ejecución de dos comandos consecutivos no implica decir dos frases seguidas, sino decir la segunda después de haber captado la primera). Se ha utilizado un usuario diferente para cada combinación de permisos: uno para la restricción de edad, otro con privilegios, un tercero con ambos y un cuarto sin restricciones. Cabe mencionar que los permisos se asignan a roles, no a usuarios, por lo que se han creado cuatro roles distintos con un usuario asignado a cada uno de ellos.
- En el programa de los dispositivos, se han probado todas las acciones posibles de forma independiente y aislada para cada periférico.

Es importante tener en cuenta que las pruebas se han realizado utilizando únicamente una Raspberry Pi Pico W, por lo que no se ha verificado simultáneamente el funcionamiento de varios dispositivos remotos. No obstante, en la configuración se han creado varios dispositivos y se han instalado y probado de manera secuencial en el mismo dispositivo físico.

7. Trabajos futuros

Para dejar de tratar el sistema como un prototipo y abordar funcionalidades que se asemejen más a la realidad, sería recomendable llevar a cabo una serie de mejoras que por razones de tiempo no han sido posibles abordar:

- Definir más comportamientos en los dispositivos controlados remotamente, de tal manera que permitan trabajar con cualquier variable física dado un periférico concreto. De esa forma, sí que sería realmente posible un manejo inteligente realista del hogar de por ejemplo motores de persiana, alarmas con sensores de movimiento, etc.
- Otra buena característica para implementar sería una versión por consola de la configuración. De esta manera, se podría reutilizar un sistema operativo Raspbian Lite sin interfaz gráfica.
- Podría ponerse sobre la mesa la opción de añadir reconocimiento gestual. De esta manera, se complementaría con el reconocimiento por voz como una alternativa, aunque habría que estudiarla, ya que el modelo de Raspberry Pi utilizado podría, a priori, no ser soportado junto al resto del *software* en términos de rendimiento.
- La creación de plantillas supondría configuraciones y comandos prediseñados. Mediante su uso, los usuarios podrían buscar la plantilla e importarla, evitando tener que perder tiempo en configurar. Junto a lo anterior, también sería muy buena idea la posibilidad de poder importar scripts representando comandos locales complejos, como por ejemplo buscar la previsión meteorológica, cargar un perfil en una aplicación o acciones similares.
- La adición de comandos predeterminados a usuarios, de tal manera que no solo el reconocimiento de voz sirva para ejecutar comandos. Tras detectar un rostro, automáticamente llevar a cabo estas acciones predeterminados asociadas.

8. Conclusión

Durante el desarrollo de este proyecto y a lo largo de este documento, se ha demostrado la capacidad de las plataformas de bajo coste para realizar tareas domóticas mediante interacciones implícitas. Utilizando una Raspberry Pi, junto con múltiples Raspberry Pi Pico W, es posible ejecutar acciones domóticas mediante el reconocimiento de voz y facial.

Se han seguido diversas etapas, comenzando por una primera aproximación genérica del desarrollo y la aplicación de la ingeniería de software para plasmar las ideas y encontrar formas de llevarlas a cabo. Se han abordado aspectos concretos de diseño e implementación, aunque en esta memoria se describen de manera general debido a la limitación de este documento. Además, las pruebas de rendimiento y uso garantizan un funcionamiento relativamente óptimo y satisfactorio del sistema.

Se han utilizado una amplia variedad de tecnologías, no solo en términos de lenguajes de programación, sino también en el ámbito de las bibliotecas y su interacción para lograr homogeneidad. Además, se han explorado otras tecnologías, como MQTT y JSON, que han sido fundamentales para las comunicaciones a través de la red Wi-Fi y el mantenimiento de las configuraciones respectivamente.

Desde el inicio, se ha considerado al usuario como el enfoque principal del proyecto, buscando en todo momento facilitar tareas como la instalación y la configuración, y garantizando que los usuarios experimenten efectividad, utilidad y personalización en el uso del sistema.

Por último, el código fuente completo de EcoTronix se proporciona en un repositorio público alojado en la dirección web github.com/Osorbe10/EcoTronix. Esto permitirá realizar los futuros trabajos expuestos y transformar el proyecto de un prototipo a un sistema funcional en el mundo real.

9. Bibliografía

[1] RASPBERRY PI, 3 Model B [Consulta: 02/02/2023] Disponible en: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

[2] RASBERRY PI, Camera Module 2 NolR [Consulta: 23/04/2023] Disponible en: https://www.raspberrypi.com/products/pi-noir-camera-v2/

[3] RASBERRY PI, Microcontrollers [Consulta: 09/03/2023] Disponible en: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html

[4] TECNÓFILO, Conexionado de la placa RaspBerry Clip [Consulta: 09/03/2023] Disponible en: https://www.tecnofilo.es/raspberry-pi/235-placa-de-expansion-para-raspberry-similar-a-berryclip.html?search_query=BERRYCLIP&results=1

[5] RASBERRY PI, Computers [Consulta: 09/03/2023] Disponible en: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

[6] MICROPYTHON, Download RP2 Pico W [Consulta: 27/02/2023] Disponible en: https://micropython.org/download/rp2-pico-w/

[7] JQLANG, Documentación de jq [Consulta: 29/01/2023] Disponible en: https://jqlang.github.io/jq/manual/

[8] MOSQUITTO, mosquitto.conf man page [Consulta: 02/05/2023] Disponible en: https://mosquitto.org/man/mosquitto-conf-5.html

[9] PYTHON, Documentación de tkinter [Consulta: 04/02/2023] Disponible en: https://docs.python.org/es/3/library/tkinter.html

[10] PYPI, Documentación de paho-mqtt [Consulta: 24/02/2023] Disponible en: https://pypi.org/project/paho-mqtt/

[11] POCKETSPHINX, Documentación de PocketSphinx [Consulta: 30/01/2023] Disponible en: https://pocketsphinx.readthedocs.io/en/latest/

[12] PYPI, Documentación de PyAudio [Consulta: 13/02/2023] Disponible en: https://pypi.org/project/PyAudio/

[13] PYPI, Documentación de face-recognition [Consulta: 28/01/2023] Disponible en: https://pypi.org/project/face-recognition/

[14] RASBERRY PI, Microcontrollers [Consulta: 27/02/2023] Disponible en: https://www.raspberrypi.com/documentation/microcontrollers/micropython.html

[15] MPYTHON, Documentación de umqtt.simple [Consulta: 27/02/2023] Disponible en: https://mpython.readthedocs.io/en/master/library/mPython/umqtt.simple.html

[16] MICROPYTHON, Documentación de ujson [Consulta: 10/03/2023] Disponible en: https://docs.micropython.org/en/v1.15/library/ujson.html

[17] MICROPYTHON, Documentación de network [Consulta: 09/03/2023] Disponible en: https://docs.micropython.org/en/latest/esp8266/tutorial/network_basics.html

ANEXO A. Estructura de los ficheros de configuración

El fichero de configuración *Pico/peripherals.json* (Figura A.1) define tanto los periféricos internos como los externos. En caso de querer añadir nuevos periféricos, además de añadir las funciones pertinentes en el código *Pico/main.py*, habría que definir su interfaz en este fichero.

```
{
    "external": [
        { "type": "BerryClip Buzzer", "actions": [ "sound" ], "pins": [ 4 ] },
        { "type": "BerryClip Led", "actions": [ "on", "off", "get" ],
          "subtypes": [
              { "subtype": "Green", "pins": [ 22, 27 ] },
              { "subtype": "Red", "pins": [ 9, 10 ] },
              { "subtype": "Yellow", "pins": [ 11, 17 ] }
          ]
        }
    ],
    "internal": [
        { "type": "Integrated Led", "actions": [ "on", "off", "get" ] },
        { "type": "Internal Temperature", "actions": [ "get" ] }
    ]
}
```

Figura A.1. Fichero de configuración Pico/peripherals.json

```
{
    "wifi": { "ssid": "", "password": "" },
    "mqtt": { "server": "", "user": "", "password": "", "room": "", "position": "" },
    "peripherals": { "external": [], "internal": [] }
}
```

Figura A.2. Plantilla de fichero de configuración Pico/config_template.json

Los dispositivos usan un fichero de configuración basado en *Pico/config_template.json* (Figura A.2), siendo rellenados los datos específicos del dispositivo al momento de su instalación. Como su propio nombre indica, este archivo es la plantilla para cada uno de los dispositivos, conteniendo información genérica como credenciales Wi-Fi y MQTT y *hostname* del nodo central (no mostrados en la Figura A.2, pero almacenados en la propia plantilla mediante la herramienta de instalación y configuración) e información concreta del dispositivo como la sala y la posición dentro de la misma y extractos de los periféricos asignados al dispositivo definidos en *Pico/peripherals.json*.

El fichero de configuración restante es el más importante, y el usado por el nodo central. *config.json*, en la Figura A.3, almacenará todas las configuraciones de usuarios, roles, salas, posiciones, dispositivos, comandos locales y remotos, lenguaje por defecto y rutas a todos ellos y edad legal para la restricción de edad.

Figura A.3. Fichero de configuración config.json

Todas las secciones se basan en listas de entradas. La sección de lenguajes es rellenada a través de la herramienta de instalación, así como el lenguaje por defecto al inglés. El resto, son gestionadas por la herramienta de configuración.

Cada una de las secciones, por extensión, sería prácticamente imposible mostrarlas en forma de ejemplo. Para hacerse una idea, cada una de las entradas de su sección, posee unos atributos tal y como se definieron en el apartado de implementación de la interfaz gráfica de la herramienta de configuración del nodo central.