Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Desarrollo de un sistema de Visión Artificial para ayuda a personas con dificultad motora en las extremidades superiores

Autor:

Bouaouda, Smail

Tutores:

Gómez García-Bermejo, Jaime

Zalama Casanova, Eduardo

Ingeniería de Sistemas y Automática

Valladolid, junio 2022

AGRADECIMIENTOS

En primer lugar, agradecer a mis padres el gran esfuerzo que han realizado abandonando su país de origen con el único propósito de darnos tanto a mí como a mi hermano un futuro mejor. La educación y la motivación que me han otorgado es la que ha hecho que sea quien soy y llegue a estar donde estoy. Lo mejor que nos pueden ofrecer nuestros padres y podemos ofrecer a nuestros hijos es una buena educación y unos valores que te formen como persona, ninguna cosa material puede superar esto. Puedo sentirme afortunado de que mis padres me hayan inculcado ambas cosas. Siempre les estaré agradecido porque sin sus "charlas" no sabría que habría sido de mí, lo que es seguro, es que sin su apoyo diario no habría hecho ni la mitad de las cosas que hecho. GRACIAS.

A lo largo de la vida conocemos a personas, algunas de ellas pasan a ser nuestros amigos. Es sabido que las relaciones y las personas con las que te juntas pueden encaminar tu futuro. Yo he tenido la suerte de coincidir con unas personas maravillosas, tanto aquellas que conocí de pequeño como los que conocí en la propia universidad, personas responsables y con las ideas claras. Su compañía ha hecho que este trayecto sea mucho más fácil y llevadero. Gracias por ser mis amigos.

Por último y no menos importante, agradecer a mis tutores de TFG Jaime Gómez y Eduardo Zalama, primero por darme la oportunidad de realizar este proyecto tan bello y útil, con el que poder ayudar a aquellas personas más desfavorecida, y segundo por su implicación y preocupación para que el proyecto se llevara a cabo de la mejor manera posible, gracias a sus directrices y ayuda ha sido más fácil realizar el proyecto.

RESUMEN

En el presente proyecto, se desarrolla un sistema de visión artificial para la ayuda a aquellas personas con problemas motoras que afecten a sus extremidades superiores. Para la realización del sistema, se ha programado la placa de desarrollo Maixduino mediante el software Maixpy IDE. La placa Maixduino que es capaz de soportar inteligencia artificial va a permitir que, mediante una interfaz de cuatro botones digitales mostrados en su pantalla y su cámara se pueda usar la nariz como un puntero que pueda accionar dichos botones. La información de activación/desactivación de dichos botones digitales se enviará a una Raspberry Pi que contendrá el software domótico Home Assistant, en el cual programaremos una serie de automatizaciones, para el control de una serie de actuadores, como enchufes inteligentes, bombillas inteligentes, etc. Es decir, mediante un ligero movimiento de cuello, una persona podrá activar o desactivar una serie de dispositivos inteligentes.

PALABRAS CLAVE

Visión artificial. Discapacidad motora. Maixduino. MicroPython. Maixpy IDE. Raspberry Pi. Home Assistant

ABSTRACT

In this project, an artificial vision system is developed to help people with motor problems affecting their upper limbs. To create the system, the Maixduino development board has been programmed using the Maixpy IDE software. The Maixduino board, which is capable of supporting artificial intelligence, will allow the nose to be used as a pointer that can be activated by means of an interface of four digital buttons shown on its screen and its camera. The activation/deactivation information of these digital buttons will be sent to a Raspberry Pi that will contain the Home Assistant home automation software, in which we will program a series of automations to control a series of actuators, such as intelligent plugs, intelligent light bulbs, etc. In other words, by means of a slight neck movement, a person will be able to activate or deactivate a series of intelligent devices.

KEYWORDS

Artificial vision. Motor disability. Maixduino. MicroPython. Maixpy IDE. Raspberry Pi. Home Assistant.

Índice de contenidos

Índice de tablas

Índice de figuras

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

En este capítulo de introducción encontraremos la motivación que ha llevado a realizar el proyecto, los objetivos a conseguir, antecedentes al proyecto a realizar, así como el plan de trabajo y la organización de la memoria.

1.2 Motivación

El instituto nacional de estadística español (INE) proporciona una encuesta de discapacidad del año 2008, en la cual se puede observar el número de personas que padecen de alguna discapacidad que pueda afectar al movimiento de las extremidades superiores (**FIGURA 1**).

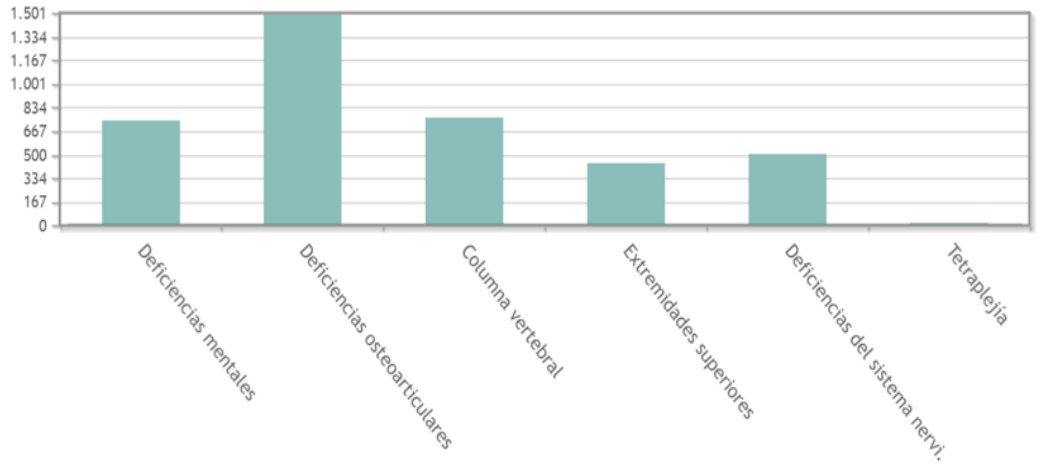

FIGURA 1. Encuesta de discapacidad, autonomía personal y situaciones de dependencia para ambos sexos [1]

Según el INE el 37% (1,4 millones) de las personas con discapacidad tienen dificultades a la hora de realizar actividades básicas de la vida diaria sin ayuda [1]. Por ello este sector de la población se encuentra con grandes dificultades para la realización de tareas de forma autónoma, lo que provoca una dependencia y disminución de la calidad de vida.

Aunque el número de personas que sufren esta problemática es reducido frente a la población total, necesitan la máxima ayuda para poder disminuir su grado de dependencia todo lo posible. Esto no solo ayudaría a las personas que sufren la propia discapacidad, sino también a las personas que las rodean y están a su cuidado.

Por tanto, el propósito de este proyecto, además del de poner a prueba todos los conocimientos adquiridos en el grado, también es, aprender, profundizar e investigar en un tema de gran interés personal, así como contribuir a ayudar a un sector que se encuentra en estado de vulnerabilidad, que sufre en su día a día el no poder realizar tareas básicas y necesarias para la supervivencia de forma autónoma. La utilidad del desarrollo del proyecto que se va a llevar a cabo va a ser máxima y va a tener resultados que aportan a mejorar la sociedad en la que vivimos.

La tecnología debe estar siempre al servicio de la sociedad.

1.3 Objetivos

El objetivo del presente proyecto es ayudar a personas que sufran dificultades motoras que afecten a las extremidades superiores a través del desarrollo e implementación de un sistema de visión artificial y domótico. El sistema va a consistir en que, mediante un leve movimiento del cuello, los usuarios puedan realizar una serie de acciones domóticas de forma autónoma, disminuyendo su dependencia y aumentando su calidad de vida.

El proceso seguido por el sistema va a consistir en detectar la punta de la nariz y a partir de su posición sobre una serie de botones digitales en una pantalla realizar unas determinadas acciones domótica como, por ejemplo, encender y apagar un televisor o las luces de una sala.

Algunos de los sectores que pueden beneficiarse de este sistema son las personas que padecen de:

- Amputación de las extremidades superiores.
- Distrofia muscular.
- Parálisis cerebral.
- Lesión modular.
- Esclerosis múltiple.

Hay que tener en cuenta, que este sistema va a dirigido en especial a personas que además de sufrir problemas que afecten a sus extremidades superiores, tienen problemas de dicción y habla, ya que, se podría pensar que una alternativa a este sistema podría ser por ejemplo el dispositivo que proporciona Amazon denominado Alexa, entre muchos otros.

1.4 Metodología y plan de trabajo

Para realizar el proyecto de una forma organizada y cumplir con los objetivos propuestos, se ha realizado un plan de trabajo a seguir que es el siguiente:

- **Estudio del estado de la tecnología.** Esta etapa del proyecto consiste en recopilar toda aquella información que pueda ser de interés y ayuda, en la investigación de la rama de la visión artificial y domótica para la adquisición de los conocimientos necesarios, además del análisis y estudio de los

componentes físicos y aplicaciones informáticas que se usarán para conseguir el desarrollo y montaje del sistema final que cumpla el objetivo propuesto.

- **Desarrollo y montaje del sistema.** El desarrollo y montaje se va a llevar a cabo en distintas etapas, las cuales irán sumando hasta llegar al desarrollo final. Las etapas son las siguientes:

1. La primera etapa consiste en configurar los dispositivos para su correcto funcionamiento.

2. Una vez realizado el proceso anterior, programar el dispositivo de visión artificial para detectar la posición de la nariz, que usaremos como puntero para navegar por la pantalla LCD de la que dispone esta placa.

3. Una vez detectada la nariz, se va a desarrollar una interfaz con botones digitales, para que el usuario mediante la punta de la nariz los pueda activar y desactivar.

4. En la cuarta etapa, se procederá a instalar y poner en marcha el software domótico en un ordenador de placa reducida (Raspberry pi).

5. La quinta etapa va a consistir en la comunicación del dispositivo de visión artificial con el ordenador de placa reducida (Raspberry pi), con el objetivo de transmitir la información de un dispositivo a otro. Para ello se deben seguir los siguientes pasos:

   5.1. Conectar la placa de visión artificial a la red wifi.
   5.2. Programar el protocolo de comunicación MQTT en la placa de visión artificial.
   5.3. Integrar el protocolo de comunicación MQTT en el programa domótico.

6. En la sexta etapa se va a comprobar que la comunicación entre ambos dispositivos es correcta.

7. Una vez enviada la información y visualizada de forma correcta en el software domótico instalado en el ordenador de placa reducida, procedemos a realizar una automatización en su entorno.

8. Por último, se va a integrar y automatizar en el programa domótico un actuador, que deberá conmutar en función de la activación y desactivación de los botones, con el objetivo de poder probar el funcionamiento del sistema.

- **Comprobación del desarrollo y montaje del sistema.** Una vez desarrollado y montado todo el sistema, se va a poner en marcha para probar que su funcionamiento es el esperado.

- **Memoria.** Como parte final del trabajo de fin de grado, queda explicar en la memoria la información necesaria y los pasos seguidos para desarrollar el sistema al completo, de forma que se pueda comprender y desarrollar por cualquier otra persona con solo la lectura de esta.

1.5 Organización de la memoria

La memoria constará de los siguientes capítulos:

1. **Introducción.** En este capítulo se explican los objetivos y motivaciones que han llevado a realizar el proyecto, la metodología seguida para llevar a cabo el desarrollo del proyecto y el modo en el que se organiza la memoria.

2. **Marco teórico.** Encontraremos aquellos conceptos teóricos necesarios para entender el desarrollo del trabajo.

3. **Recursos y herramientas.** Aquí veremos todo el hardware y software utilizado para desarrollar y ejecutar el proyecto.

4. **Configuración de la placa de visión artificial y Raspberry Pi.** Analizaremos cómo se han de configurar dichos dispositivos para su uso.

5. **Desarrollo del sistema.** En este capítulo se describe el desarrollo y realización completa del sistema de visión artificial, de tal forma que se pueda obtener unos resultados deseables y que cumplan con el objetivo propuesto.

6. **Resultados y experimentos.** Se probará el sistema al completo para observar los resultados obtenidos y se realizarán una serie de

experimentos para comprobar que el funcionamiento es el adecuado.

7. **Estudio económico.** En este capítulo se realizará un presupuesto económico del coste que lleva realizar todo el proyecto.

8. **Conclusiones y líneas futuras.** Se presenta un breve resumen del trabajo realizado, en el que se explica las características principales del proyecto realizado y sus ventajas frente a otros sistemas similares.

9. **Bibliografía.** Compilación de todas aquellas referencias que han sido utilizadas para la ejecución del proyecto.

10. **Anexos.** Aquí se recoge información que, por su extensión o carácter complementario, pudiera entorpecer la lectura secuencial de la memoria.

2 VISIÓN ARTIFICIAL Y DISCAPACIDAD

2.1 Introducción

Para el desarrollo del proyecto se necesita inicialmente adquirir conocimientos e información acerca del tema que se quiere desarrollar. Para adquirir estos conocimientos se ha realizado una investigación sobre aquellos temas que van a intervenir a la hora del desarrollo del proyecto, desde temas básicos hasta otros más complejos.

2.2 Visión artificial

2.2.1 Que es y funcionamiento de la visión artificial

La visión artificial es una disciplina científica que consiste en adquirir, procesar y analizar imágenes, por lo que trata de emular la visión de los seres vivos, en el que

un objeto es captado por los receptores de la retina y es convertido en impulsos nerviosos procesados por el cerebro. Las imágenes se captan mediante cámaras y son tratadas mediante técnicas de análisis de imagen, con el fin de obtener información que pueda ser analizada por una máquina [2].

En general, los sistemas de visión artificial se componen de los siguientes elementos básicos [3]:

- Sistema de iluminación
- Cámara
- Sistema de captura
- Procesamiento de imagen
- Sincronía
- Actuadores

A continuación, se muestra en la **FIGURA 2** un sistema con las etapas y componentes que son necesarios para el adecuado procesamiento de imágenes anteriormente mencionados.

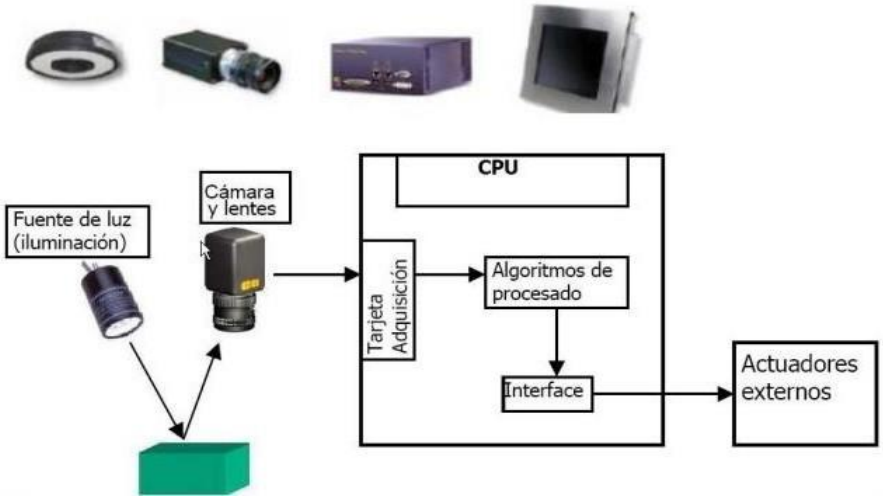

FIGURA 2. Sistema y componentes para el procesamiento de imágenes [4]

En un diagrama de bloques el funcionamiento podría ser el siguiente (**FIGURA 3**):

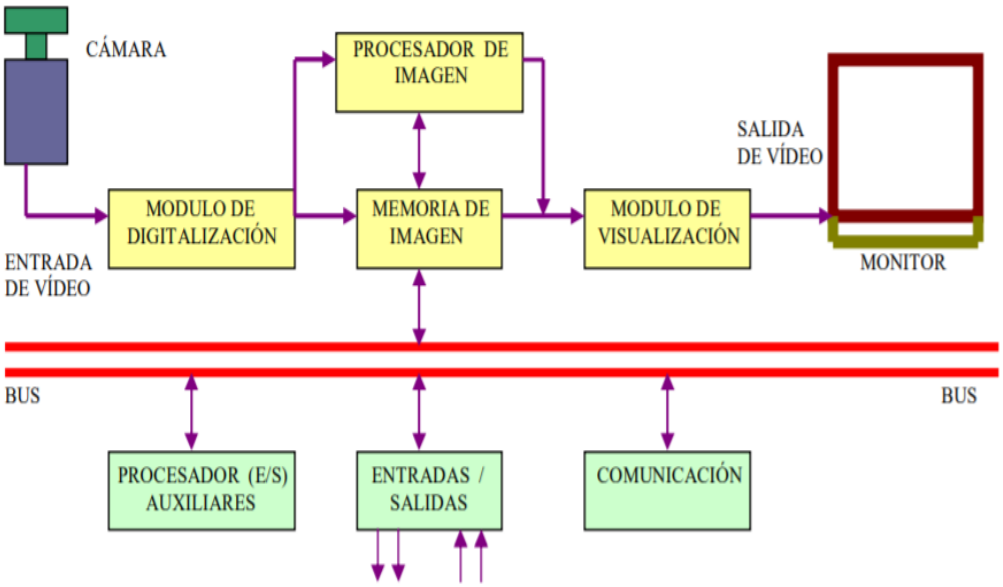

FIGURA 3. Diagrama de bloques del procesamiento digital de imágenes [5]

La explicación correspondiente a cada bloque es la siguiente:

- **Módulo de digitalización:** Transforma la señal analógica proporcionada por la cámara a una señal digital.
- **Memoria de imagen:** Almacena la señal procedente del módulo de digitalización.
- **Módulo de visualización:** Transforma la señal digital a señal analógica para poder ser visualizada en el monitor
- **Procesador de imagen:** Procesa las imágenes captadas por la cámara.
- **Módulo de entradas/salidas:** Gestiona la entrada de sincronismo de imagen y las salidas hacia actuadores.
- **Comunicaciones:** Protocolo de comunicación usado.

Algunas aplicaciones que se le puede dar a la visión artificial pueden ser las siguientes:

- Control de calidad.
- Clasificación.
- Detección de objetos.
- Reconocimiento facial y de objetos.
- Visión artificial + Robótica en procesos industriales.

2.2.2 Funcionamiento de la tecnología de detección de rostros

La tecnología que permite la detección de rostros utiliza algoritmos para encontrar rostros humanos en imágenes. Dichos algoritmos de detección pueden detectar las cejas, la boca, la nariz y los ojos [6].

Los algoritmos deben ser entrenados en conjuntos de datos que incorporen miles y miles de imágenes positivas y negativas, a mayor entrenamiento de estos algoritmos, mayor es la precisión de detección. Mediante estos entrenamientos obtenemos modelos neuronales que permiten la detección y reconocimiento facial.

Los métodos utilizados en la detección facial son los siguientes:

- Los métodos basados en conocimientos o en reglas describen un rostro en función de las reglas.
- Los métodos de características invariantes que usan características como ojos humanos o nariz para detecciones faciales.
- Los métodos de coincidencia de plantillas basadas en la comparación de imágenes previamente almacenadas para detectar un rostro en una imagen.
- Los métodos basados en la apariencia emplean el análisis estadístico y aprendizaje automático para hallar características relevantes en las imágenes faciales.

Algunas de las técnicas utilizadas en la detección facial incluyen:

- Eliminar el fondo puede ayudar a determinar el contorno facial.
- Uso del color de piel para detectar el rostro.
- Uso del movimiento para determinar la posición facial.

Para la mejora de la aplicación de detección de rostros, se han desarrollado algoritmos como el detector de disparo único (SSD) y la red neuronal convolucional (CNN) de la que hablaremos a continuación debido a que es el algoritmo utilizado por nuestra placa de desarrollo de visión artificial.

2.2.3 Redes neuronales convolucionales (CNN)

El dispositivo de visión artificial utilizado es compatible con aplicaciones de computación de borde de visión artificial basado en redes neuronales convolucionales, de hecho, cargaremos en dicho dispositivo modelos neuronales capaces de la detección y puntos clave del rostro.

Para ello, necesitamos conocer que son y cómo funcionan las redes neuronales convolucionales. Estas consisten en una semejanza tecnológica a las neuronas en la corteza visual primaria de un cerebro biológico y son muy efectivas para tareas de visión artificial [7].

2.2.3.1 Estructura y funcionamiento de este tipo de redes

Estas redes consisten en múltiples capas de filtros convolucionales unidimensionales o multidimensionales. Después de cada capa, se agrega una función para realizar un mapeo causal no lineal.

La primera fase de funcionamiento es la extracción de características, compuesta de neuronas convolucionales, la siguiente fase es la reducción por muestreo y finalmente tenemos neuronas más sencillas para la clasificación final.

2.2.3.2 Aprendizaje de las redes neuronales convolucionales

Las CNN aprenden a reconocer una amplia variedad de objetos contenidos en una imagen, pero previamente necesitan ser “entrenadas” mediante una gran cantidad de "muestras” (10.000), para que las neuronas sean capaces de capturar las singularidades de cada objeto, este proceso se le denomina “algoritmo de aprendizaje”.

El seguimiento para el aprendizaje de las CNN es el siguiente:

- **Píxeles y neuronas:** En primer lugar, la red toma los píxeles de la imagen como entrada. Por ejemplo, si tenemos una imagen de 28x28 píxeles de alto y ancho, se tienen que usar 784 neuronas. Teniendo en cuenta que solo se usa un color (escala de grises). Si la imagen es a color, necesitaremos 3 canales RGB, por lo que ahora se necesitarán 2352 neuronas (28x28x3), que componen nuestra capa de entrada.

- **Pre-procesamiento (FIGURA 4):** Anteriormente, como entrada debemos

tener valores entre 0 y 1, por tanto, debemos dividirlos todos entre 255 (colores de los píxeles).

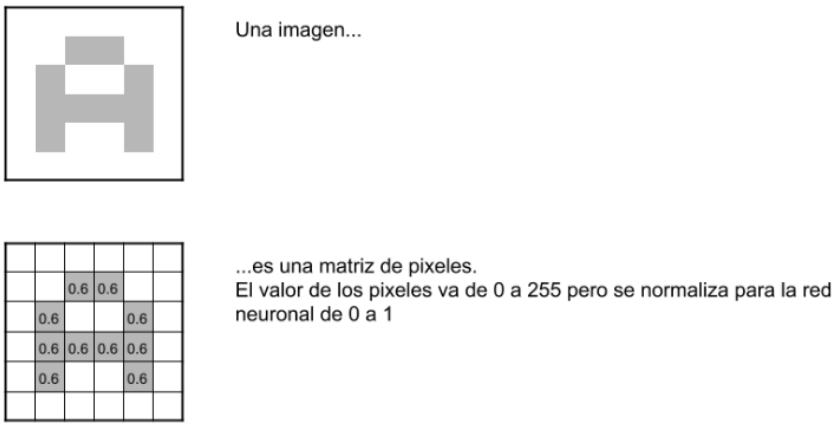

FIGURA 4. Imagen normalizada para la red neuronal [7]

- **Convoluciones:** Consiste en tomar grupos de píxeles cercanos de la imagen de entrada e ir operando contra una matriz denominada kernel (matriz con valores iniciales aleatorias y que se irán ajustando mediante *backpropagation*) mediante el producto escalar. Mediante esta operación obtenemos una nueva matriz de salida, que será nuestra nueva capa de neuronas ocultas.

- **Filtros:** No solo constará de un kernel, sino de un conjunto de ellos (filtros). Por ejemplo, si tenemos 32 filtros conseguiremos 32 matrices de salida, por lo que tendremos 25088 neuronas (28x28x32) para la primera capa oculta de neuronas.

- **Función de activación:** La más utilizada es la denominada ReLu (*Rectifier Linear Unit*) que consiste en una función f(x)=max (0,x).

- **Muestreo:** En el siguiente paso, se toma una muestra de las neuronas más representativas, antes de realizar una convolución de nuevo. El muestro nos permite reducir el tamaño de la siguiente capa de neuronas, ya que no daríamos uso a todas las neuronas obtenidas anteriormente, sino solo aquellas que nos proporcionan las características más importantes.

- **Conexión con una red neuronal tradicional:** Finalmente, se toma la última capa muestreada, la cual es tridimensional (alto, ancho, mapas) y se aplana, pasando a una capa de neuronas tradicional y así conectamos una nueva capa oculta de neuronas tipo *feedforward*. A esta capa oculta, se le

aplica la función *Softmax* que conecta con la salida final. La estructura por tanto es la que se muestra en la **FIGURA 5**.

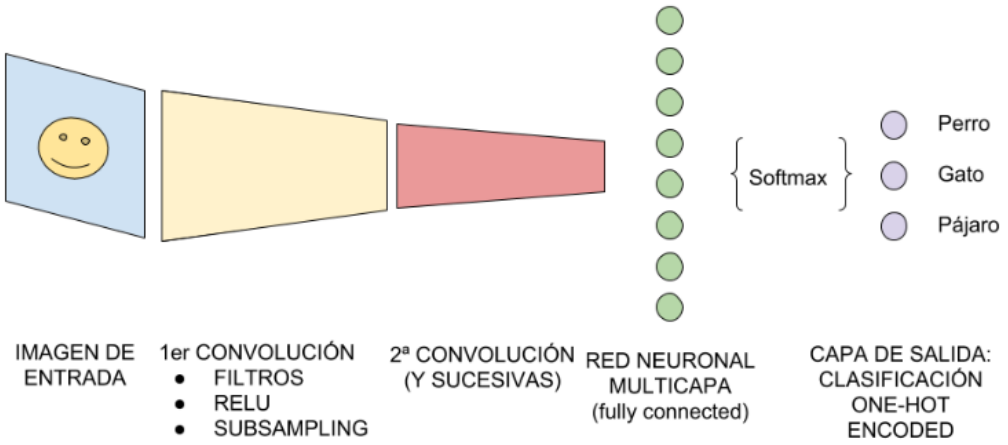

FIGURA 5. Estructura de una CNN [7]

2.3 Discapacidad motora

En este apartado se va a conocer en qué consisten las discapacidades motoras y como éstas afectan a las personas que la sufren, contemplando qué tipo de discapacidades motoras existen según una clasificación. El análisis se centrará particularmente en las discapacidades motoras que afectan a las extremidades superiores en especial, ya que el sistema a desarrollar va enfocado a personas que sufran este tipo de problemática.

2.3.1 ¿Qué es la discapacidad motora?

La discapacidad motora es la dificultad que presentan algunas personas para realizar acciones propias de la vida cotidiana, afectando a la ejecución de movimientos y a la motricidad en general. Este tipo de discapacidad contiene todas las alteraciones orgánicas del aparato motor, que afectan a articulaciones, nervios, sistema óseo y/o músculos.

2.3.2 Clasificación de la discapacidad motora

La discapacidad motora se clasifica según el momento en el que se dio la lesión, las causas por las que se dio la discapacidad (etiopatología), la localización y extensión de la discapacidad (topografía) y el origen de las deficiencias [8].

- Según el **momento de la lesión**:
    - Originadas en el nacimiento.
    - Originadas después del nacimiento.
    - Originadas a lo largo de la vida.

- Según la **etiopatología**:
    - Discapacidad motora por transmisión genética.
    - Discapacidad motora por infecciones.

- Según la **topografía** (**FIGURA 6**):

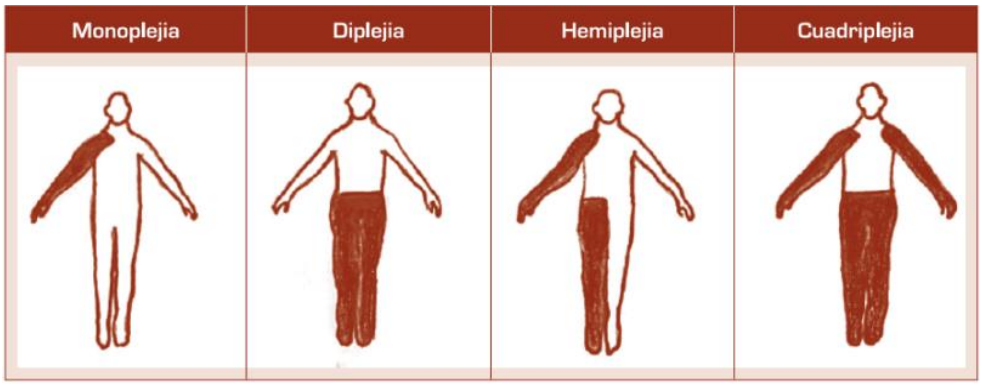

FIGURA 6. Parálisis total según la topografía [8]

El sistema a desarrollar se enfoca a la ayuda a aquellas personas que sufren cuadriplejia, pacientes que están un porcentaje muy elevado del día en una camilla, con una movilidad que se reduce al cuello y, por tanto, su dependencia es total.

2.4 Dispositivos para la ayuda a personas con discapacidad motora que afecta a las extremidades superiores

En el presente apartado, se mostrarán aquellos dispositivos o sistemas informáticos del mercado cuya función es la ayuda a personas con discapacidad motora que afecta al movimiento de las extremidades superiores y en general a todo el cuerpo, por lo que estas personas solo tienen control del movimiento de la cabeza. Estos dispositivos nos servirán como antecedente a nuestro sistema desarrollado.

2.4.1 LifewareIntegra

LifewareIntegra (**FIGURA 7**) consiste en un sistema desarrollado por la empresa Emotiv System, gracias al uso de electrodos y un giroscopio permite utilizar una computadora a personas que sufren alguna discapacidad física. El control del computador se realiza mediante movimientos de la cabeza y expresiones faciales [9]. El sistema por tanto consta de un programa y un dispositivo físico, que mediante la interacción de ambos permiten el control de una computadora por personas con discapacidad.

Los principales usos que permite LifewareIntegtra son los siguientes:

- Uso del teclado, mediante un teclado de pantalla.
- Uso completo del ratón.
- Reconocimiento de voz.

FIGURA 7. Sistema para el control de computador LifewareIntegra [9]

La limitación de este sistema frente al nuestro tiene lugar en que este sistema no puede ser usado por personas que se encuentren completamente tumbadas en una camilla, cuya única movilidad es la del cuello, por ende, este sistema limita su uso a un grupo de personas que tienen una movilidad elevada, personas que pueden estar sentadas y cuya dependencia es menor.

Además, este dispositivo tiene que estar continuamente colocado en la cabeza, lo que disminuye la manejabilidad y comodidad para el usuario y el coste es elevado debido a que se necesita además un ordenador que soporte el software.

2.4.2 Sip/Puff Switch

El pulsador Sip/Puff Switch (**FIGURA 8**) consiste en un sistema que, mediante el soplido o la absorción realizados por una persona en un tubo introducido en la boca, permite el accionamiento de un pulsador de forma neumática para la activación de dispositivos como ordenadores, juguetes adaptados, dispositivos de comunicación aumentativa y acceso y sistemas de control de entorno. Por tanto, este dispositivo es muy útil para aquellas personas con grandes dificultades en la movilidad de las extremidades superiores, inferiores e incluso de cabeza ya que la activación de accionadores [10].

FIGURA 8.Sistema para el accionamiento de un pulsador Sip/Puff Switch [10]

Un sector de la población con discapacidad motora que afecta a casi la totalidad de la movilidad del cuerpo también puede sufrir parálisis facial, lo que puede provocar que uno de los extremos de la boca caiga y se tenga problemas para la retención de saliva, por tanto, este tipo de dispositivo quedaría inutilizado para este grupo de personas, o simplemente los pacientes no tienen la suficiente fuerza para que el soplido sea detectado. Este problema no tiene lugar para nuestro sistema, debido a que el uso de la nariz como puntero para detonar acciones domóticas permite que personas que sufren esta problemática puedan usarlo de forma cómoda y sencilla.

2.4.3 Maltron head / Mouth Stick Keyboard

El Maltron Head consiste en un teclado especial para aquellas personas que no pueden usar las extremidades superiores para la escritura en ordenador [11].

Su uso consiste en ir tecleando mediante un palo que es soportado por la boca del usuario.

El diseño es de tal forma que reduce el movimiento realizado por la cabeza aumentando la comodidad. La orientación del teclado es vertical a diferencia de un teclado convencional, lo que facilita su uso como se puede observar en la **FIGURA 9**.

FIGURA 9.Maltron Head [11]

Este tipo de sistema recoge tanto el problema encontrado en el sistema LifewareIntegra debido a que necesitamos soportar con la boca un pequeño palo y el problema del Sip/Puff Switch, debido a que personas con parálisis facial no pueden realizar la acción de soportar el palo mediante la boca, por lo que su utilidad es mucho menor que los dos sistemas mencionados anteriormente.

2.4.4 Head Wand

El head wand (**FIGURA 10**) consiste en un dispositivo que es colocado en la cabeza y consta de una varita que permite asistir a las personas con un movimiento limitado de las extremidades superiores, pero con capacidad de movimiento de la cabeza. El objetivo de este dispositivo es ayudar a las personas a escribir en un ordenador u otros dispositivos [12]. La ventaja de este dispositivo frente al mencionado anteriormente (Sip/puff Switch) es el soporte que incorpora la varilla y que va enganchado a la cabeza, lo que evita el uso de la boca, disminuyendo la fatiga e incomodidad, permitiendo que su uso pueda ser prolongado.

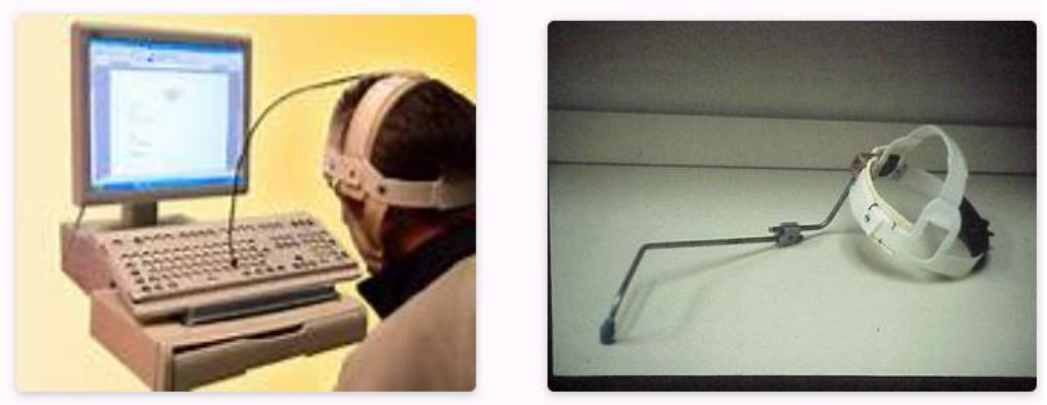

FIGURA 10.Head Wand [12]

La principal desventaja de este dispositivo tiene que ver con que no puede ser usado por personas que están completamente tumbadas y con la cabeza apoyada en la camilla, ya que el soporte no lo permite, debido a la incomodidad y la poca manejabilidad que tendría lugar, lo que reduce su rango de funcionalidad entre las personas con discapacidades motoras que afecten a las extremidades superiores.

Nuestro sistema presenta una serie de ventajas frente a todos los dispositivos mostrados anteriormente. Una de las ventajas principales, es que puede ser usado por cualquier tipo de persona con discapacidad motora que afecte a la movilidad de las extremidades superiores independientemente de su situación y posición, personas en sillas de ruedas, personas tumbadas en camilla, etc.

Su pequeño tamaño le proporciona manejabilidad a la hora de su colocación para proporcionar la mayor comodidad posible al usuario a la hora de su uso.

Otra ventaja es el funcionamiento del dispositivo, el cual es muy intuitivo y fácil de comprender, lo que le otorga accesibilidad a cualquier tipo de persona, además de su reducido coste.

3 RECURSOS Y HERRAMIENTAS

3.1 Introducción

En este apartado se contemplará todo el hardware y el software necesario para llevar a cabo el proyecto. El primer punto corresponde al Hardware utilizado y el segundo punto al Software.

3.2 Dispositivos físicos

En el presenta apartado, se indicarán aquellos dispositivos físicos utilizados y que mejor se adaptan para desarrollar el sistema.

El sistema consta de una pequeña placa de desarrollo capaz de soportar visión artificial, que nos va a permitir programar la interfaz de usuario y la detección de la nariz usada como puntero. La elección de dicha placa (Maixduino) se debe

a su pequeño tamaño, lo que permite su manejabilidad y facilidad de instalación en posiciones cómodas para el usuario. Además, su bajo coste con todas las propiedades que incorpora dicha placa, la hace ideal para desarrollo de proyectos que empleen inteligencia artificial e internet de las cosas como nuestro caso.

Para las automatizaciones domóticas necesitaremos el uso de un ordenador de placa reducida en el que cargaremos el software domótico (Home Assistant), el mejor dispositivo para la instalación de este software es una Raspberry Pi como nos indica la propia página oficial de Home Assistant. En este caso, usaremos concretamente la Raspberry Pi 4 Model B, que permite integrar un mayor número de dispositivos debido a su mayor capacidad, frente a versiones anteriores.

Por último, necesitaremos actuadores, estos actuadores pueden variar en función de las necesidades de cada usuario. Los actuadores serán integrados en el programa domótico Home Assistant para poder ser automatizados y serán los encargados de realizar las distintas acciones.

Para las pruebas de funcionamiento del sistema al final del desarrollo del proyecto solo usaremos un actuador, debido a la gran variedad de actuadores que existen y que dependerá el uso de unos u otros en función de las necesidades del usuario, como se ha dicho anteriormente. Indicaremos algunos ejemplos de actuadores que podrían ser utilizados.

A continuación, se van a detallar las características de cada uno de los dispositivos mencionados con anterioridad.

3.2.1 Sipeed Maixduino Kit para RISC-V AI + IoT

El Sipeed Maixduino kit para RISC-V AI + IoT (**FIGURA 11**) consta de conectores para una pantalla LCD de 2,4 pulgadas que permite mostrar imágenes, gráficos y textos y un conector para el módulo de cámara digital GC0328 para visión artificial. Además, la placa consta de un micrófono para el reconocimiento de voz y procesamiento de sonidos y un conector de salida de audio al que poder conectar un altavoz [13].

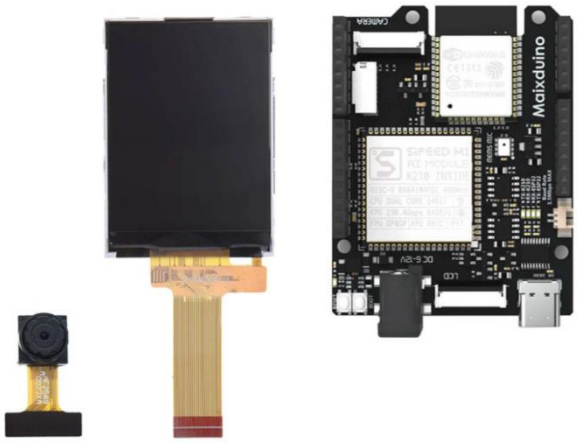

FIGURA 11.Sipeed Maixduino Kit para RISC-V AI + IoT [13]

3.2.1.1 Detalles del producto

Sipeed Maixduino es una placa de desarrollo de 68mm x 54mm basada en el módulo Sipeed MAIX con una CPU RISC-V de doble núcleo de 64 bits junto con un procesador de red neuronal de 400 MHz. El procesador de red neuronal permite aplicaciones de IA (Inteligencia Artificial). Además, el Maixduino integra un módulo ESP32 el cual permite conectividad inalámbrica (Wifi + Bluetooth).

La alimentación es mediante cable USB tipo C (5 Voltios) o un conector de entrada de tipo CC (6-12 Voltios) [A1]. En la **FIGURA 12** y **FIGURA 13** respectivamente, se pueden observar los detalles de los que se compone la placa de desarrollo Maixduino y los recursos de pines. El Maixduino es compatible con Arduino IDE, PlataformIO IDE, MicroPython, OpenMV IDE Grove y TensorFlow [13].

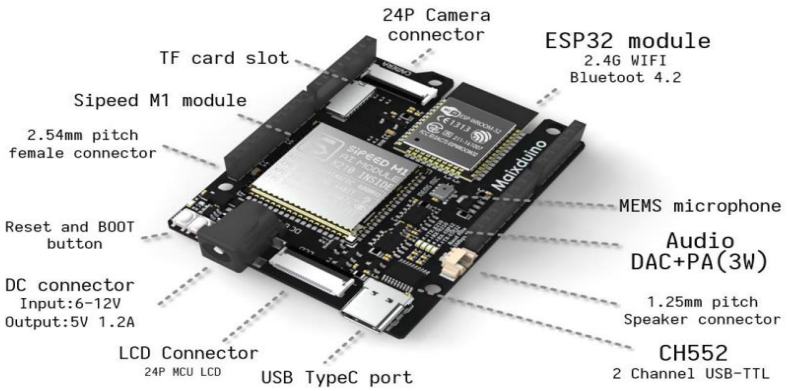

FIGURA 12. Detalles de la placa Maixduino [14]

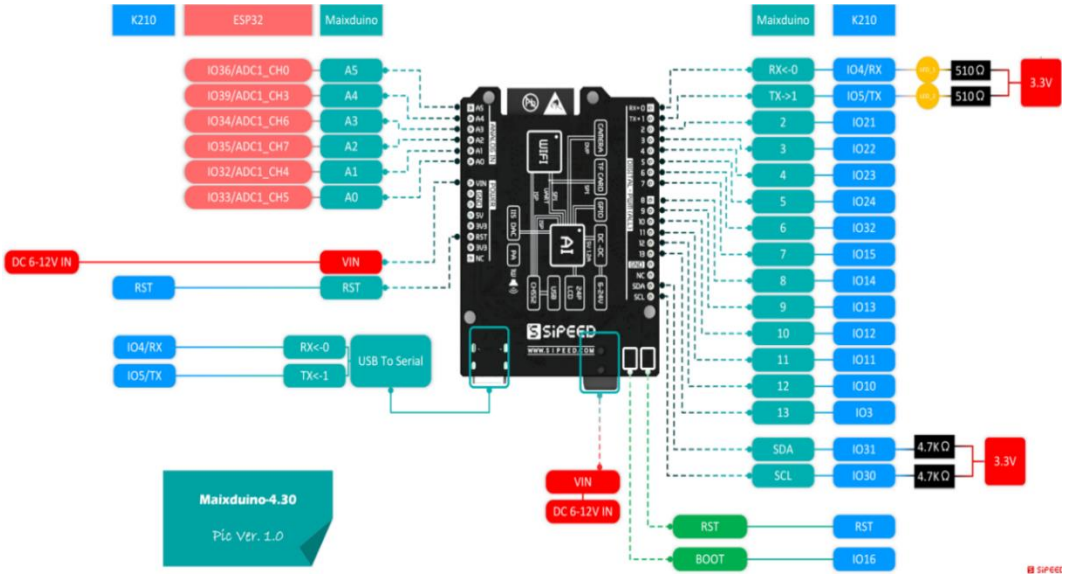

FIGURA 13. Recursos de pines [15]

3.2.1.2 Especificaciones

Las especificaciones de mayor interés de la placa de desarrollo Sipeed Maixduino se recogen en la **TABLA 1**.

Módulo maestro	**Módulo AIot MAIX-I sipeed**
Entrada de energía	**Usb Type-CDC-DC** **circuito reductor: soporte de entrada de 6-12V** **Proporcionar salida de 5V 1.2A**
Ranura para tarjeta Micro SD	**Soporte de tarjeta**
Micrófono MEMS integrado	**MSM261S4030H0 es un micrófono MEMS de salida digital I2S omnidireccional, con puerto inferior.**
Interfaz de cámara DVP	**Conector FPC 24P de 0,5 mm**
Conector LCD	**Conector FPC MCU LCD 24P de 0,5 mm de 8 bits**
Salida de audio	**DAC+ PA:TM8211: rango dinámico de 16 bits** **Baja distorsiónarmónicaNS4150:3W potencia de salida;** **Hasta un 90% de eficiencia**
Módulo ESP32	**Soporte 2.4G 802.11.b / g / n802.11 n (2.4 GHz) velocidades de hasta 150 MbpsBluetooth v4.2 estándar completo, incluyendo Bluetooth tradicional (BR / EDR) y Bluetooth de baja energía (BLE)**
Tensión de alimentación de la fuente de alimentación externa	**4.8V ~ 5.2V**

TABLA 1. Especificaciones de la placa Maixduino [14]

En las especificaciones mostradas en la tabla anterior, podemos observar que la placa Maixduino elegida para nuestro sistema, se adapta perfectamente a nuestras necesidades, gracias a su módulo MAIX-I Sipeed nos va a permitir el uso de redes neuronales capaces de la detección de puntos clave del rostro. Asimismo, incorpora el módulo ESP32 para la conexión wifi, permitiendo la comunicación con el entorno domótico instalado en la Raspberry Pi.

En cuanto a la alimentación del dispositivo, se puede alimentar mediante una batería portátil, o un cargador como el usado para cualquier teléfono móvil gracias a su puerto tipo C.

3.2.1.3 Aplicaciones

El dispositivo Maixduino tiene un gran número de aplicaciones, algunas de ellas se muestran a continuación:

- Detección y reconocimiento facial.
- Aplicación en la industria para la clasificación de productos.
- Grabación y reproducción de vídeos.
- Emulador de videojuegos.
- Reconocimiento de voz.
- Incorporado a robots industriales en procesos productivos.

3.2.1.4 Accesorios para Maixduino

Para el funcionamiento de la placa Maixduino necesitaremos los componentes siguientes:

- **Cable USB-C:** Este cable lo usaremos tanto para cargar el programa desarrollado en la placa como para la alimentación de la misma.

- **Batería portátil:** Para alimentar el dispositivo, usaremos una batería portátil cuyo voltage de salida sea de 5V, lo que va a proporcionar movilidad a nuestro dispositivo, ya que no depende de la alimentación a la red.

3.2.2 Raspberry pi

Raspberry pi (**FIGURA 14**) es un miniordenador compacto y de bajo coste que permite el desarrollo de una gran cantidad de proyectos.

La salida al mercado de la primera Raspberry pi tuvo lugar en febrero del año 2012 a través de la Raspberry pi Foundation. El objetivo de esta fundación era el de hacer llegar la informática y aumentar su accesibilidad a todo el público gracias al bajo coste de este dispositivo [16].

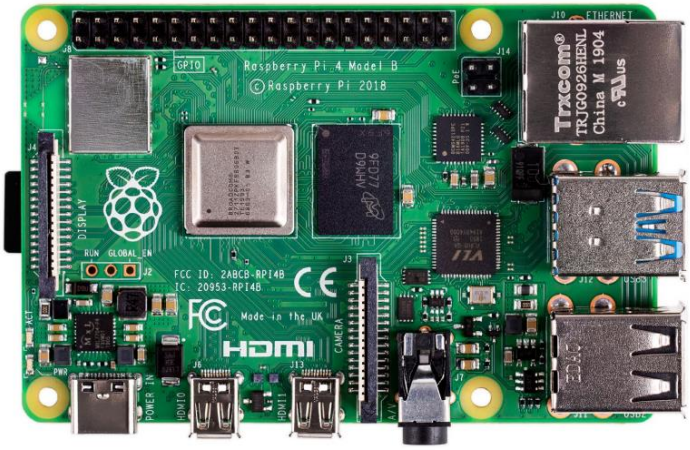

FIGURA 14. Raspberry pi 4 Model B [17]

Existen gran variedad de modelos de Raspberry, a continuación, se muestra un pequeño resumen de las características de cada una de estas versiones.

- **Versión Pi Model B:** Son placas más completas ya que incorporan puertos Ethernet y puertos USB. Las versiones más actualizadas disponen de conectividad Bluetooth y Wifi.

- **Versión Pi Model A:** Placas más compactas que las Model B, suelen ser versiones sencillas ya que tienen solo un puerto USB y no disponen de puerto Ethernet.

- **Versión Pi Zero (FIGURA 15):** Placas de un tamaño muy reducido. Disponen de un microUSB y un microHDMI. Cuentan con conexión Wifi y Bluetooth.

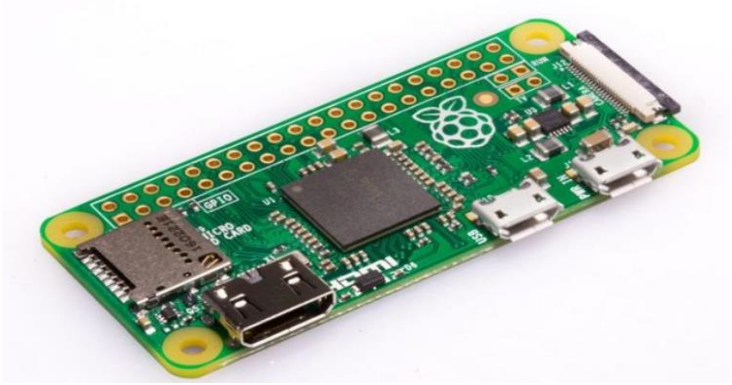

FIGURA 15.Raspberry pi Zero [16]

- **Versión Pi 400 (FIGURA 16):** Se trata de un ordenador compacto ya que incorpora un teclado. Dispone de conexión Wifi y Bluetooth, dos puertos USB 3.0, un puerto USB 2.0, dos microHDMI y un puerto Gigabit Ethernet, su procesador es de 4 núcleo y una RAM de 4 GB.

FIGURA 16. Rspberry pi 400 [17]

- **Versión Pi Pico (FIGURA 17):** Microcontrolador con procesador RP2040. No dispone de ningún puerto de conexión y carece de conectividad inalámbrica.

FIGURA 17. Raspberry pi Pico [16]

Para el desarrollo de nuestro sistema, vamos a emplear la Raspberry pi 4 Model B de 4GB de RAM, ya que cumple con todos los requisitos de especificaciones que nos son necesarios y es muy completa en cuanto a recursos se refiere, por lo que en los siguientes apartados se indicarán aquellos detalles y especificaciones importantes de esta placa.

3.2.2.1 Detalles del producto Raspberry Pi 4 Model B

La Raspberry Pi 4 Model B (**FIGURA 18**) consta de un procesador ARM Cortex-A72 con 4 núcleo a 1.5GHz. La placa permite la conexión inalámbrica de Bluetooth 5.0 y Wifi 802.11ac. La alimentación de la placa es mediante USB tipo C. Algunos detalles extra con los que cuenta este modelo es el de poder soportar doble monitor con resolución 4K.

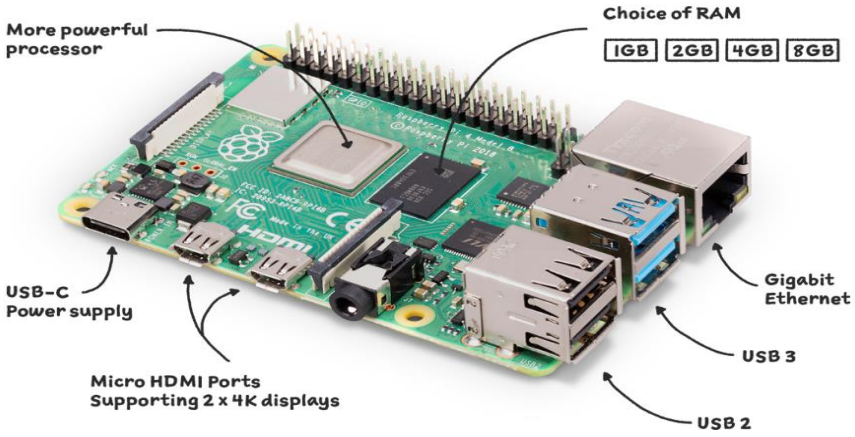

FIGURA 18. Detalles de la Raspberry Pi 4 Model B [18]

3.2.2.2 Especificaciones de la Raspberry Pi 4 Model B

Las especificaciones técnicas de la Raspberry 4 model B (4GB) se encuentran en la **TABLA 2 .**

Procesador	**Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz**
Memoria	**1GB, 2GB, 4GB o 8GB LPDDR4 (depende del modelo).**
Conectividad	**2.4 GHz y 5.0 GHz IEEE 802.11b/g/n/ac inalámbrico LAN, Bluetooth 5.0, BLE Gigabit Ethernet 2×USB 3.0 2×USB 2.0**
GPIO	**Cabezal GPIO estándar de 40 pines**
Video & Sonido	**2 × puertos micro HDMI (Compatible con hasta 4Kp60) Puerto de visualización MIPI DSI de 2 carriles Puerto de cámara MIPI CSI de 2 carriles Puerto de audio estéreo y vídeo compuesto de 4 polos**
Multimedia	**H.265 (decodificación 4Kp60) H.264 (decodificación 1080p60, codificación 1080p30) Gráficos OpenGL ES, 3.0**
Tarjeta SD	**Ranura para tarjeta Micro SD**
Alimentación	**5V/3A vía USB-C, 5V vía cabezal GPIO**
Expansión	**Cabezal GPIO de 40 pines**

TABLA 2. Especificaciones de la Raspberry pi 4 Model B [17]

La especificación de mayor interés para nuestro sistema es la capacidad de la que dispone este modelo de Raspberry, lo que va a permitir una integración de dispositivos mucho mayor (actuadores), por lo que, a una sola Raspberry Pi, podemos conectar un gran número de placas Maixduino (según el número de pacientes en una residencia) que controlen un gran número de actuadores.

Además, consta de conectividad para la comunicación con la placa de visión artificial y su rendimiento es el adecuado para soportar Home Assistant.

3.2.2.3 Aplicaciones

La Raspberry pi tiene una infinidad de aplicaciones gracias a su versatilidad. A continuación, se muestran algunos ejemplos de uso que se le puede dar:

- Ordenador de mesa.
- Controlador para robots.
- Servidor web o FTP.
- Centro multimedia.
- Domotización del hogar.
- Sistema de seguridad y vigilancia.

3.2.2.4 Accesorios para la Raspberry Pi 4

Para el funcionamiento de la Raspberry Pi 4 necesitamos una serie de componentes que se muestran a continuación.

- **Fuente de alimentación USB-C:** Para la alimentación de la placa Raspberry necesitaremos un alimentador de 5V/3A de tipo C.

- **Cable Ethernet:** Para conectar la Raspberry al Router y así dotar de wifi a la Raspberry, necesitaremos un cable Ethernet.

- **Tarjeta SD:** Para cargar el software domótico Home Assistant con el que realizaremos las automatizaciones, necesitaremos una tarjeta SD SanDisk de 32 GB (recomendada por Home Assistant).

- **Memoria USB:** Para la configuración de la IP del servidor de Home Assitant necesitaremos una memoria USB. La memoria del USB es indiferente, ya que, solo usaremos un archivo de texto para dicha configuración.

- **Carcasa de protección (FIGURA 19):** Para proteger la placa Raspberry de posibles caídas y agentes externos, necesitamos una carcasa que envuelve la misma.

FIGURA 19. Carcasa protectora Raspberry Pi 4 [18]

3.2.3 Actuadores inteligentes

A continuación, vamos a analizar algunos actuadores, aunque para los experimentos y resultados del proyecto solo se ha usado un actuador (enchufe Tapo P100), ya que la incorporación de cualquier otro actuador resultaría directa y muy similar a la utilizada para el actuador con el que se va a probar el funcionamiento del sistema. Además, hay infinidad de actuadores que pueden ser integrados, cuya elección depende de las necesidades y deseos de cada usuario.

3.2.3.1 TP-Link Tapo P100 mini Smart wifi enchufe inteligente

El enchufe Tapo P100 (**FIGURA 20**) consiste en un enchufe inteligente con conectividad Wifi. Este pequeño enchufe tiene las siguientes propiedades [19]:

- **Control Remoto:** Control de los dispositivos conectados en cualquier lugar a través de la aplicación Tapo.
- **Programación:** Automatización del dispositivo.
- **Temporizador:** Crea una cuenta atrás para apagar o encender los dispositivos electrónicos conectados.
- **Control de Voz**: Control mediante voz a través de Amazon Alexa o Google Assistant.
- **Modo Fuera de Casa**: Enciende y apaga en diferentes horarios para dar la apariencia de que alguien está en casa.
- **Diseño Compacto:** Pequeño tamaño.
- **Fácil de Utilizar e Instalar:** Configuración rápida mediante la aplicación Tapo.

FIGURA 20. Enchufe inteligente Tapo P100 [19]

En la **TABLA 3** se indican las especificaciones técnicas del enchufe inteligente Tapo P100.

Red	**Protocolo: IEEE 802.11 b/g/n, Bluetooth 4.2 (Solo para la primera configuración)** **2.4 GHz Wifi** **Android 4.3 o superiores** **iOS o superiores**
General	**Dimensiones: 51 x 72 x 40 mm** **Material: PC** **Botón de Power** **Led indicador de estado**
Carga máxima	**2300 W/10 A**
Alimentación	**AC 220-240 V ~ 50/60 Hz, 10 A**

TABLA 3.Especificaciones del enchufe Tapo P100 [19]

Como se puede observar en la tabla anterior, las pequeñas dimensiones de este dispositivo y su conectividad a la red wifi lo hacen ideal para su control.

3.2.3.2 Bombilla inteligente Wifi Tapo L510E

La bombilla L510E (**FIGURA 21**) consiste en una bombilla inteligente con conectividad Wifi. Sus propiedades son las siguientes [20]:

- **Control Remoto:** Control de los dispositivos conectados en cualquier lugar a través de la aplicación Tapo.
- **Programación:** Automatización del dispositivo.
- **Temporizador:** Crea una cuenta atrás para apagar o encender los dispositivos electrónicos conectados.
- **Control de Voz**: Control mediante voz a través de Amazon Alexa o Google Assistant.
- **Modo Fuera de Casa**: Enciende y apaga en diferentes horarios para dar la apariencia de que alguien está en casa.

- **Fácil de Utilizar e Instalar:** Configuración rápida mediante la aplicación Tapo.
- **Modo calendario:** Apagar y encender la bombilla en función del atardecer/amanecer o programar horario de encendido y apagado.
- **Regulable:** Intensidad lumínico ajustable.

FIGURA 21. Bombilla Inteligente Tapo L510E [20]

En la **TABLA 4** se indican las especificaciones técnicas de la bombilla inteligente L510E.

Red	**Protocolo: IEEE 802.11 b/g/n, Bluetooth 4.2 (Solo para la primera configuración) 2.4 GHz Wifi**
Potencia de entrada	**8,3 W**
Alimentación	**AC 220-240 V ~ 50/60 Hz 73 mA**

TABLA 4.Especificaciones de la bombilla Tapo L510E [20]

3.2.3.3 SwitchBot pulsador inteligente

El SwitchBot (**FIGURA 22**) consiste en un mini robot pulsador compatible con la mayoría de los interruptores basculantes y botones de electrodomésticos. Este pequeño dispositivo permite convertir cualquier interruptor en un interruptor inteligente, por lo que es muy funcional [21].

La instalación es sencilla, se pega el dispositivo al interruptor o botón deseado mediante una pegatina 3M, que permite la adhesión del dispositivo a cualquier superficie, sin la necesidad de perforaciones.

FIGURA 22. SwitchBot [21]

Los tipos de interruptores con los que es compatible este dispositivo se muestran en la **FIGURA 23.**

FIGURA 23.Interruptores compatibles con el SwitchBot [21]

En la **TABLA 5** se indican las especificaciones técnicas del dispositivo SwitchBot.

Red	**2.4 GHz Wifi**
Voltaje	**3 V**
Dimensiones	**4,25 x 3,65 x 2,4 cm; 39 gramos**
Material	**Plástico**
Pilas	**1 Litio-metal**

TABLA 5.Especificaciones del SwitchBot [21]

Sus dimensiones facilitan su colocación al lado de cualquier interruptor y al estar dotado de conexión wifi puede ser integrado y automatizado en Home Assistant.

A partir de los dispositivos inteligentes mostrados anteriormente, los cuales podemos dar distintos usos y funcionalidades, podemos realizar automatizaciones

como encender/apagar una luz, un televisor, una radio, subir/bajar una persiana, etc.

3.3 Software

En el presente apartado se indica el software utilizado para llevar a cabo todo el desarrollo del proyecto.

3.3.1 Kflash_gui V1.6

Kflash_gui (**FIGURA 24**) es una herramienta que nos va a permitir cargar el nuevo y actualizado firmware de MaixPy, así como, el modelo de detección de rostro y el modelo de detección de puntos clave del rostro. Se trata de dos modelos de red neuronales preconstruidos en la memoria flash de nuestra placa de desarrollo Maixduino, para que así el procesador de red neuronal pueda utilizarlos. Estos modelos descargados de la página de Sipeed han sido entrenados para la detección facial y la identificación de la nariz, boca, labios y ojos, para poder ser ejecutados por el procesador neuronal del que consta nuestro Maixduino.

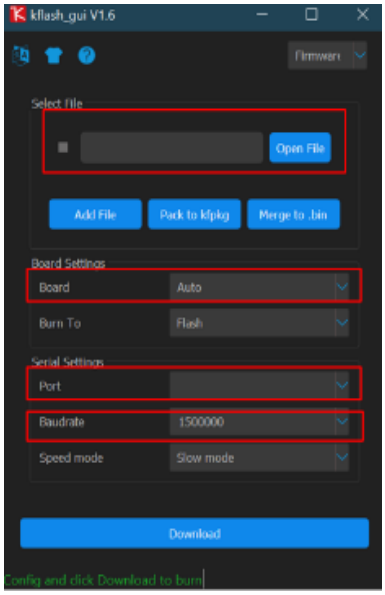

FIGURA 24. Herramienta Kflash_gui V1.6

En esta interfaz se configuran los siguientes aspectos respectivamente señalados:

1. Se cargan los archivos de firmware y modelos neuronales con sus respectivas direcciones de memoria.

2. Se selecciona el tipo de placa de desarrollo utilizada, en nuestro caso es la Sipeed Maixduino, aunque se puede dejar en modo "Auto" para que detecte de forma automática la placa.

3. Se selecciona el puerto serie perteneciente al procesador de visión artificial (COM 20).

4. Se selecciona la velocidad en baudios de carga de los archivos, en caso de error de carga, se recomienda bajar esta velocidad.

3.3.2 Flash download tool

Flash download tool es una herramienta que nos va a permitir la actualización del módulo ESP32 integrado en nuestra placa de Maixduino. Esta herramienta la podemos descargar de la página oficial de la compañía ESPRESSIF [22], como se puede observar en la **FIGURA 25**.

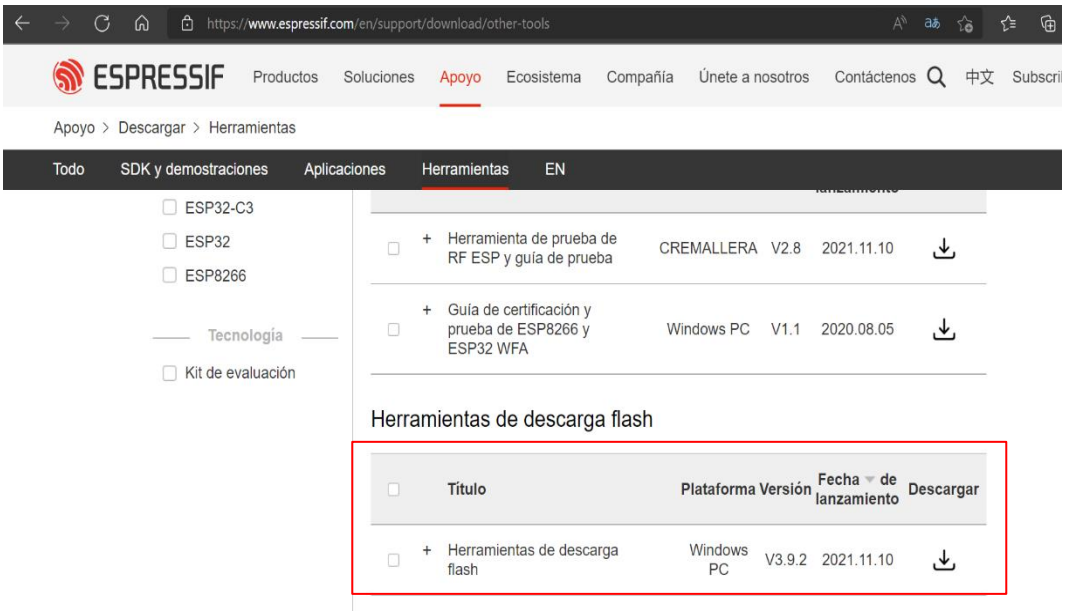

FIGURA 25. Descarga de la herramienta flash download tool [22]

La herramienta nos proporciona la opción de seleccionar entre los distintos módulos ESP y el modo de trabajo (**FIGURA 26**). Usaremos siempre el modo "develop", ya que el modo "factory" es para actualizar varias placas simultáneamente.

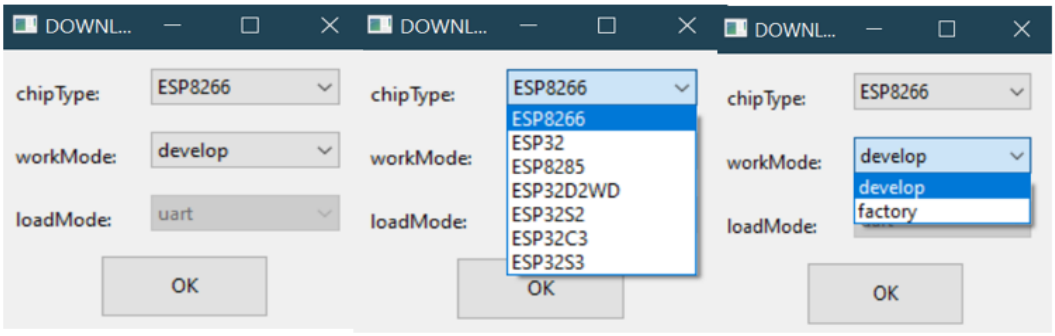

FIGURA 26. Menú de la herramienta flash download tool

3.3.3 MaixPy IDE

Para desarrollar el software de aplicación para Maixduino usaremos la herramienta Maixpy, proyecto lanzado en el año 2018. Maixpy es MicroPython portado al procesador k210, contenido en el módulo MAIX. Contiene paquetes de biblioteca prediseñados que admiten el funcionamiento de la placa, incluida la inicialización, los periféricos de entrada/salida y el procesamiento de datos del sensor [23], lo que permite que la programación en k210 sea más sencilla y rápida. Por este motivo, se ha seleccionado este software para llevar a cabo la programación de la placa Maixduino.

Maixpy IDE (**FIGURA 27**) se puede ejecutar en PC con sistemas opertaivos Windows, macOs o Linuxc. El IDE consiste en un editor y descargador que permite cargar, ejecutar y archivar en la placa de desarrollo los scripts, así como ver la imagen y flujo de datos de la cámara en tiempo real en el PC.

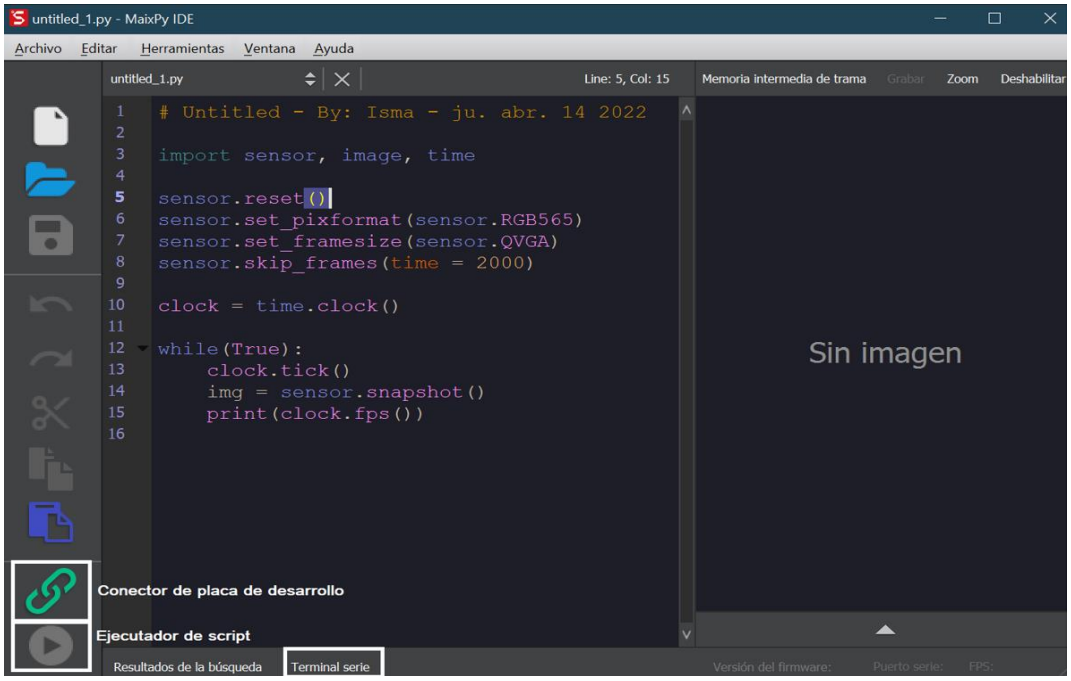

FIGURA 27. Maixpy IDE

3.3.4 Home Assistant

Home Assistant (**FIGURA 28**) es un software gratuito de código abierto que se ejecuta bajo el lenguaje Python [24] y permite controlar y monitorizar dispositivos inteligentes.

FIGURA 28. Software domótico Home Assistant [24]

Algunas de las funciones que tiene Home Assistant son las siguientes:

- Puede integrar más de 1000 dispositivos inteligentes.
- Puede realizar infinidad de automatizaciones.
- Permite instalar otras aplicaciones que faciliten la administración del hogar.
- Todos los datos permanecen locales.
- Tiene aplicación para móvil.

Estos son los dispositivos donde se puede instalar Home Assistant:

- Raspberry pi.
- Windows.
- Linux.
- MacOS.
- Android.
- Equipos C64 y X86.
- Asus Tinkerboard.

En nuestro caso, la instalación se realiza en una Raspberry Pi ya que nuestro sistema a desarrollar se basa en el uso de un ordenador de placa reducida, lo que permite una mayor facilidad a la hora de integrar dispositivos y programar automatizaciones.

Home Assistant ofrece cuatro métodos de instalación diferentes, los cuales, se diferencian en que unos métodos contienen un mayor número de propiedades que otros. Como se puede observar en la **FIGURA 29**, el método de instalación más completo y el recomendado por Home Assistant es de "SISTEMA OPERATIVO", por tanto, es el que hemos seleccionado para desarrollar el proyecto.

	SISTEMA OPERATIVO	Contenedor	Núcleo	Supervisado
Automatizaciones	✅	✅	✅	✅
Paneles	✅	✅	✅	✅
Integraciones	✅	✅	✅	✅
Planos	✅	✅	✅	✅
Utiliza contenedor	✅	✅	❌	✅
Supervisor	✅	❌	❌	✅
Complementos	✅	❌	❌	✅
Backups	✅	✅[1]	✅[1]	✅
Sistema operativo administrado	✅	❌	❌	❌

FIGURA 29. Comparación de los métodos de instalación de Home Assistant [24]

3.3.5 Tinkercad 3D

Tinkercad (**FIGURA 30**) es un programa web que pertenece a la compañía de desarrollo de software 2D y 3D Autodesk [25]. Tinkercad permite la realización de diseños de modelos 2D y 3D de forma gratuita y sencilla y se puede usar en cualquier dispositivo con conexión a internet, además de permitir el diseño 3D. En nuestro caso, usaremos este software para diseñar una serie de componentes para proteger nuestra placa Maixduino, para posteriormente proceder a su impresión mediante una impresora 3D.

FIGURA 30. Programa Tinkercad 3D [25]

3.3.6 TP-Link Tapo app

La aplicación Tapo (**FIGURA 31**) permite la configuración de dispositivos Tapo, lo cual resultará necesario para la configuración del enchufe Tapo P100 que usaremos como actuador para las pruebas de nuestro sistema [26].

También permite realizar las siguientes funciones:

- Controlar el dispositivo de forma remota desde cualquier lugar.
- Controlar el dispositivo mediante control de voz con Google Home y Amazon Echo.
- Realizar automatizaciones a los dispositivos inteligentes. Por ejemplo, encender las luces al atardecer, etc.

FIGURA 31. TP-Link Tapo app [26]

3.3.7 balenaEtcher

El software balenaEtcher (**FIGURA 32**) de código abierto, nos va a permitir escribir archivos .iso y .img, así como carpetas comprimidas en tarjetas microSD y USB. Es compatible con Windows, macOS y Linux [27]. En nuestro

caso, lo usaremos para grabar el programa de Home Assistant en una tarjeta microSD.

FIGURA 32. Software balenaEtcher [27]

3.3.8 MobaXterm

MobaXterm (**FIGURA 33**) es una herramienta que permite la computación remota, la cual proporciona todas las herramientas de red remotas como SSH, RDP, X11, … y comandos Unix (ls, cat, …). Esta herramienta nos va a servir para la creación de una llave autorizada para la herramienta "Terminal & SSH" que usaremos en el programa Home Assistant, para la descarga de la tienda de integraciones no oficiales en Home Assistant (HACS), que necesitaremos para integrar el enchufe Tapo P100. También nos va a permitir usar SSH para conectarnos al servidor de Home Assistant en caso de problemas con el acceso a Home Assistant debido al olvido de la contraseña o en caso de intrusión no deseado por alguna persona ajena [28].

FIGURA 33. Software MobaXterm [28]

4 CONFIGURACIÓN DE MAIXDUINO Y RASPBERRY PI

4.1 Introducción

En el presente capítulo, debido a la complejidad de las labores de configuración y puesta en marcha de los dispositivos que intervienen en el sistema a desarrollar, se indicará paso a paso el procedimiento de configuración seguido. Por ello, en el presente TFG, se ha desarrollado una descripción detallada de los pasos a seguir para configurar y desarrollar el sistema.

Los dispositivos principales por configurar son la placa de desarrollo Maixduino y la Raspberry Pi 4 Model B como se muestra a continuación.

4.2 Configuración de Maixduino

Se explicará los pasos seguidos para la configuración y puesta a punto de la placa de visión artificial Maixduino para su correcto funcionamiento.

4.2.1 Obtención de la clave máquina de nuestro Maixduino

Para la obtención de los modelos de red neuronal de detección de puntos clave del rostro, que veremos más adelante, debemos obtener la clave máquina de nuestra placa de desarrollo Maixduino, la cual es personal y única para cada placa.

Todo lo que realizaremos a continuación, es con la placa Maixduino conectada mediante el cable USB-C a la computadora.

Para ello, debemos abrir la herramienta Kflash_gui V1.6 que hemos mencionado en el capítulo de "Recursos y herramientas" y cargar un archivo denominado "key_gen_v1.2.bin" en la dirección de memoria 0x000. Este archivo lo podemos encontrar en la página oficial de MaixHub [29].

Disponemos de dos puertos virtuales, el "COM20" correspondiente al procesador k210 y el puerto "COM21" correspondiente al procesador ESP32, en este caso, seleccionamos el puerto "COM20" ya que es el perteneciente al procesador k210 (**FIGURA 34**). Los demás parámetros de la herramienta se dejan por defecto.

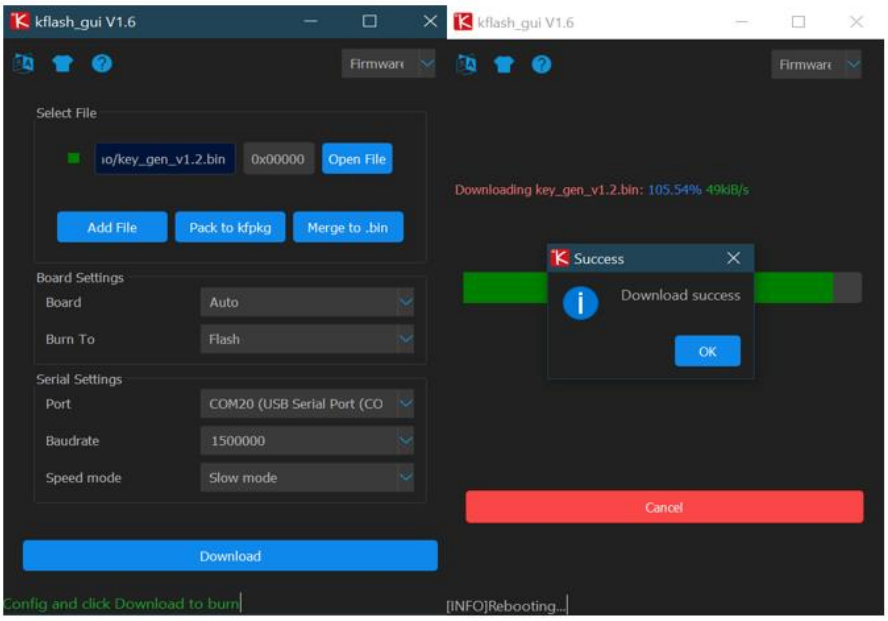

FIGURA 34. Flasheado del archivo "key_gen_V1.2.bin" para la clave máquina del Maixduino

Una vez cargado el archivo mencionado anteriormente de forma correcta, procedemos a abrir el programa Maixpy IDE. Con el programa Maixpy IDE abierto, seleccionamos el tipo de placa de desarrollo utilizada, que para nosotros es la placa "Sipeed Maixduino" (**FIGURA 35**). Ya seleccionada la placa procedemos a abrir el terminal serie como se indica en la **FIGURA 36** para conocer la clave máquina de nuestra placa Maixduino (**FIGURA 37**).

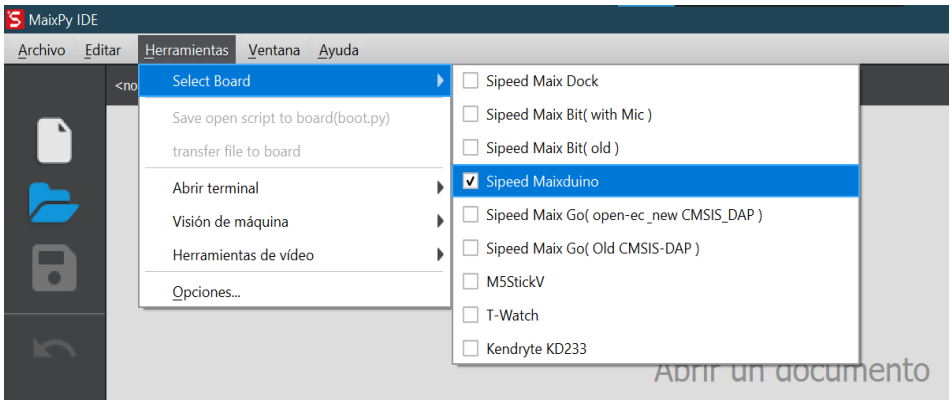

FIGURA 35. Selección del tipo de placa en el programa Maixpy

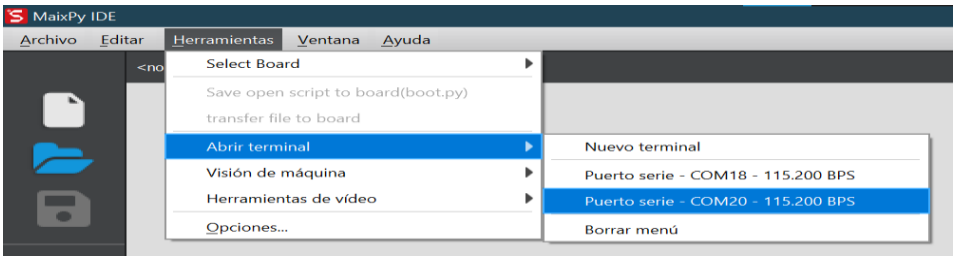

FIGURA 36. Abrir terminal serie en el programa Maixpy IDE

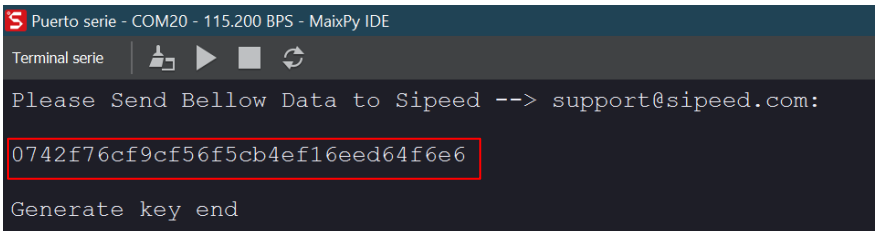

FIGURA 37. Clave máquina de la placa Maixduino

4.2.2 Actualización del firmware Maixpy

Mediante el mismo *modus operandi* que, en el apartado anterior, procedemos a cargar el firmware de Maixpy que se ejecuta en la placa. Las versiones de firmware están disponibles en la página oficial de Sipeed en la carpeta denominada MAIX/MaixPy/Release/master [30]. Hay distintas versiones de firmware de MaixPy, para este proyecto se ha utilizado la versión "maixpy_v0.6.2_72_g22a8555b5" debido a que es la versión más actualizada y,

por tanto, proporciona menos errores (**FIGURA 38**).

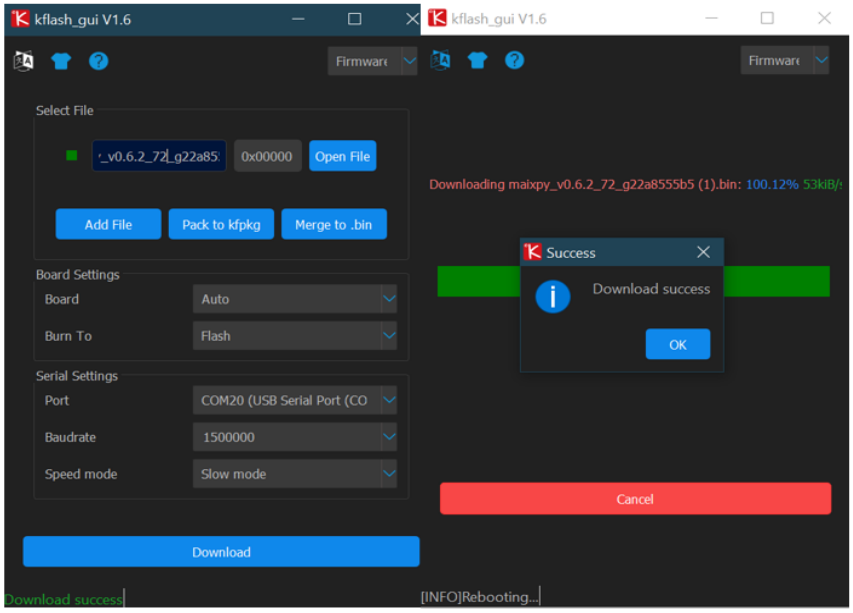

FIGURA 38. Flasheado del firmware MaixPy

Para comprobar que el flasheado de la nueva versión de firmware se ha realizado correctamente, accedemos a la aplicación Maixpy IDE y abrimos el terminal serie. Se puede observar que en la **FIGURA 39** la versión de firmware se ha actualizado satisfactoriamente.

El flasheado consiste en quemar en la placa de desarrollo, la versión de firmware más reciente, para así evitar errores futuros provocados por la desactualización.

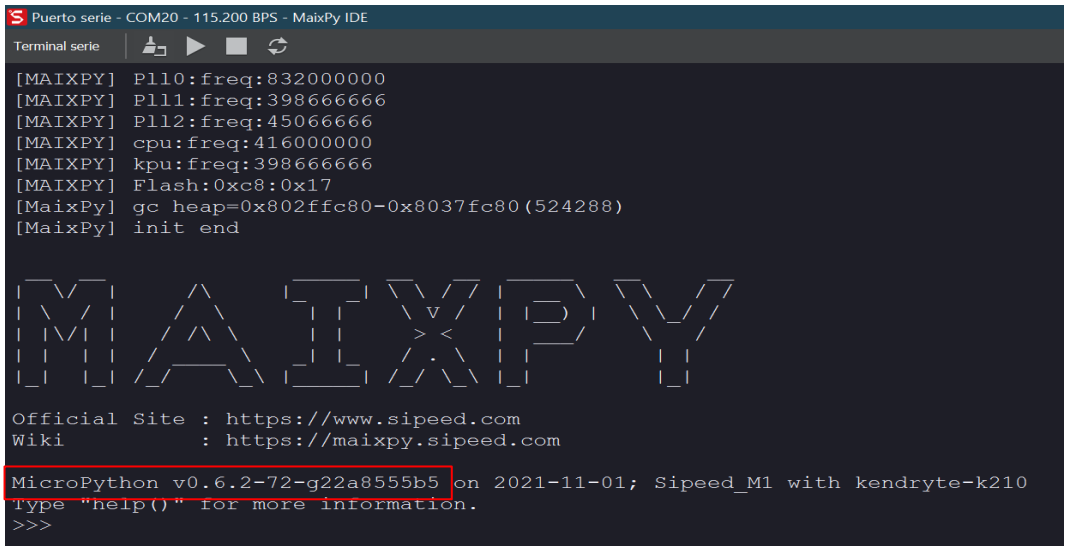

FIGURA 39. Terminal serie con la versión del firmware Maixpy

A continuación, vamos a cargar un pequeño programa que nos proporciona Maixpy IDE por defecto, con la finalidad de comprobar el funcionamiento de nuestra placa Maixduino. Para ello, pulsamos el botón conector, lo cual nos pedirá seleccionar el puerto del procesador k210 (puerto 20), una vez conecta la placa al entorno de desarrollo pulsamos el botón de ejecutar scripts señalado en la **FIGURA 27**.

A modo de prueba, ejecutamos el programa de la **TABLA 6**, que permite capturar imágenes mediante la cámara y mostrarlas en la pantalla LCD (**FIGURA 40**).

```
import sensor, image, time, lcd

lcd.init(freq=15000000)
sensor.reset()

sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

while(True):
    clock.tick()
    img = sensor.snapshot()
    lcd.display(img)
```

TABLA 6. Programa en Maixpy que toma imágenes y las muestra en la LCD

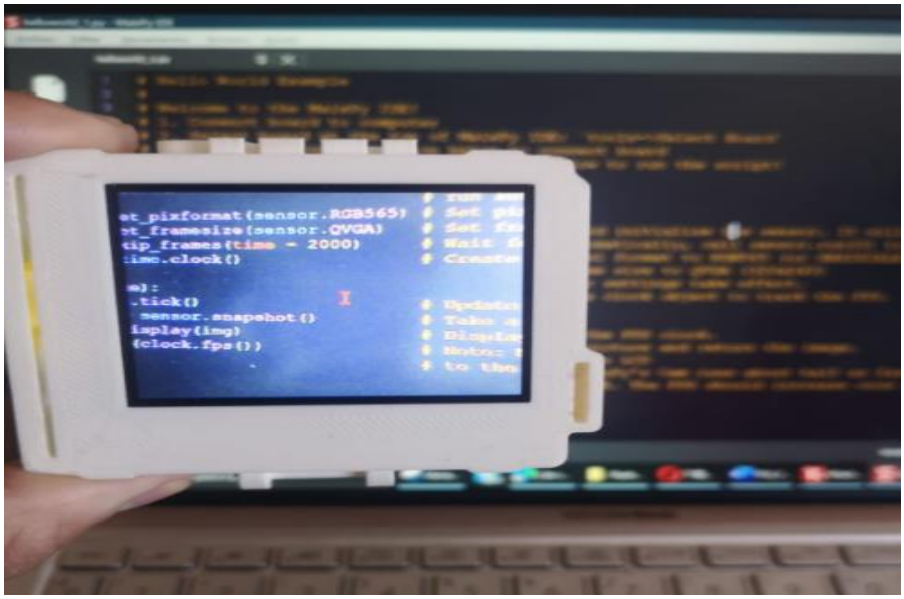

FIGURA 40. Maixduino capturando imágenes y mostrándolas en la LCD

4.2.3 Obtención de los modelos neuronales

Para obtener los modelos neuronales, debemos acceder a la página MaixHub [29], una vez en la página oficial, pinchamos en la pestaña de "Hogar" y "Visión" y buscamos el archivo denominado "Reconocimiento facial en el que podemos observar sus características como se muestra en la **FIGURA 41**. Como se puede contemplar, las características del archivo se adecuan a las especificaciones de nuestra placa, ya que el archivo está diseñado para funcionar en el procesador k210, procesador del que se compone nuestra placa y para ser ejecutado en la plataforma de software MaixPy IDE, utilizado para desarrollar el código de detección de la nariz.

FIGURA 41. Archivo de reconocimiento facial de la página MaixHub [29]

A continuación, procedemos a descargar el archivo correspondiente a los modelos de red neuronal mediante el botón "Descargar". Al pulsar este botón, nos pedirá el código máquina obtenido en el apartado **4.2.1** para poder identificar nuestra placa de desarrollo Maixduino y el cual consta de 32 caracteres (**FIGURA 42**).

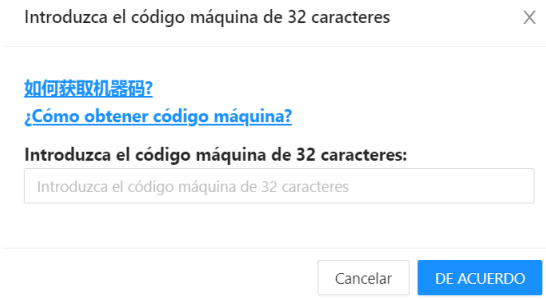

FIGURA 42. Clave máquina para descargar modelos neuronales [29]

Una vez introducidos los 32 caracteres de nuestra clave máquina pulsamos "DE ACUERDO", tras lo cual se empezarán a instalar tres modelos neuronales (**FIGURA 43**). De ellos solo nos interesan dos: el modelo de "FaceDetection" y "FaceLandmarkDetection", modelos neuronales encargados de la detección de rostro y de la detección de puntos clave del rostro respectivamente.

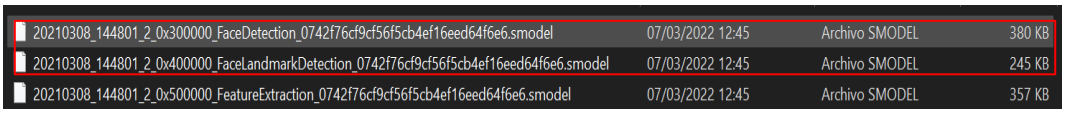

FIGURA 43. Modelos neuronales descargados

4.2.4 Carga de los modelos neuronales en la placa Maixduino

Una vez descargados los modelos neuronales correctamente, procedemos a cargar los modelos de red neuronal preconstruidos en la memoria flash de Maixduino para que el procesador k210 pueda hacer uso de ellos.

Para la carga de dichos modelos se usa de nuevo la herramienta Kflash_gui V1.6 del mismo modo visto anteriormente. Añadimos ambos archivos en la herramienta Kflash e introducimos la dirección de memoria donde se van a flashear dichos archivos como se puede observar en la **FIGURA 44**, las direcciones de memoria elegidas nos las indican los propios modelos descargados.

Por último, seleccionamos el puerto correspondiente al procesador k210 (puerto 20) dejando los demás parámetros por defecto. Pulsamos el botón “Download” y esperamos a que la carga se complete.

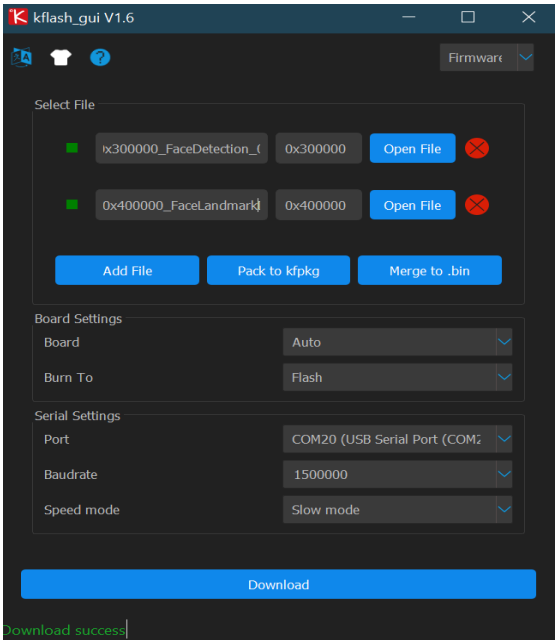

FIGURA 44. Flasheado de los modelos de detección y puntos clave del rostro

4.2.5 Actualización del firmware ESP32 integrado

Maixduino incorpora un módulo ESP32 WIFI SOC integrado, que permite la conexión inalámbrica Wifi y Bluetooth. En principio, no sería necesario actualizar este módulo ESP32, sin embargo, esto puede conducir errores a la hora de hacer uso de dicho módulo. Por ello, para evitar problemas futuros, hemos optado por actualizar su firmware a la versión más reciente.

En primer lugar, procedemos a obtener el firmware más actualizado del ESP32

para Maixduino. Para ello accedemos a la carpeta que contiene dicho firmware en la página oficial de Sipeed como se muestra en la **FIGURA 45** [30].

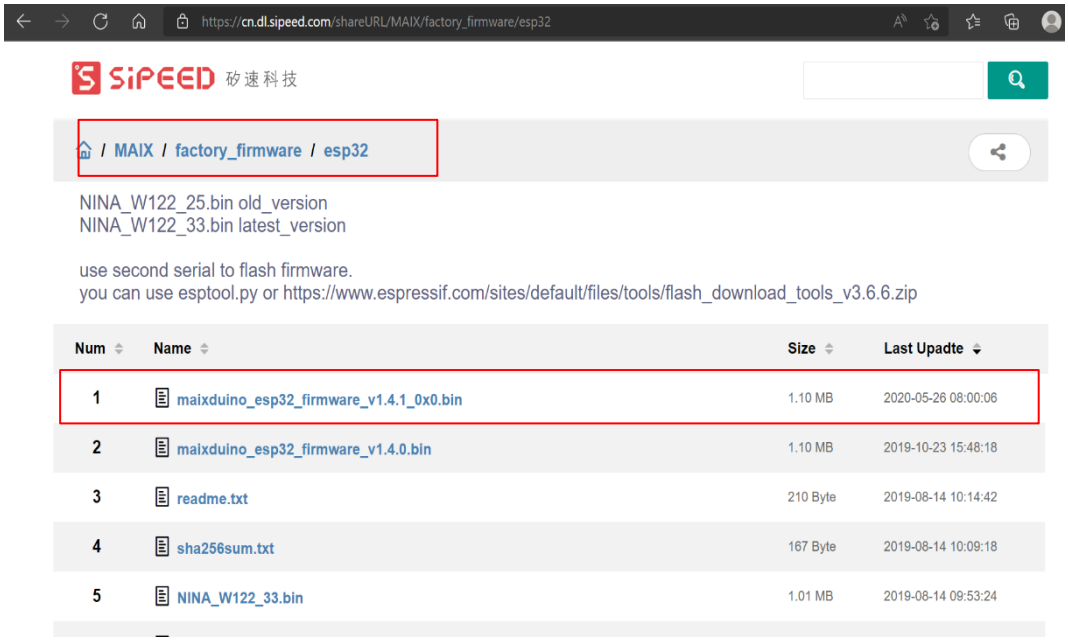

FIGURA 45. Descarga del firmware ESP32 para Maixduino [30]

Una vez obtenido el firmware a flashear, procedemos a conectar nuestra placa de Maixduino y a abrir la herramienta flash download tool descrita en el apartado **3.3.2**.

En el menú de la herramienta seleccionamos nuestro módulo, en este caso el ESP32, y pulsamos el botón OK, tal como se muestra en la **FIGURA 46**.

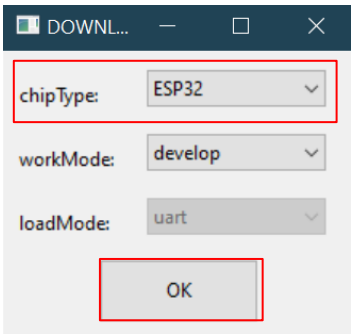

FIGURA 46. Selección del chip ESP32 en la herramienta flash download tool

Una vez en la herramienta, procedemos a su configuración. En primer lugar, abrimos el archivo del firmware descargado anteriormente "maixduino_esp32_firmware_v1.4.1_0x0.bin", el cual se va a flashear en la dirección de memoria 0x0 como nos indica el manual de flasheado. En la velocidad SPI seleccionamos 40 MHz y en el modo SPI seleccionamos DIO, automáticamente

se nos detectaran las características de nuestra placa en “DETECTED INFO”.

Una vez realizadas estas configuraciones, pulsamos “START” y esperamos a que la actualización se complete (**FIGURA 47**).

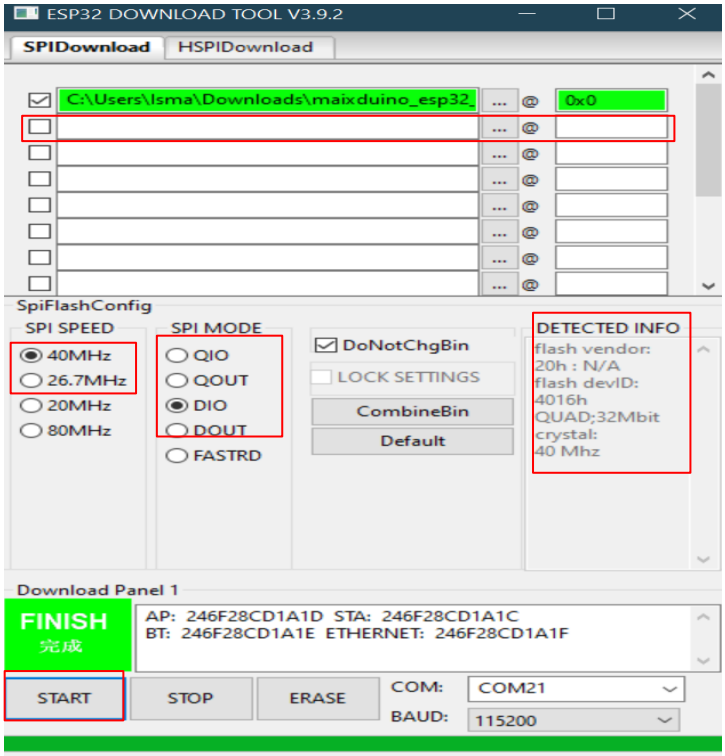

FIGURA 47. Configuración de flash download tool para la actualización del firmware ESP32

4.3 Configuración de la Raspberry Pi

Se explicará los pasos seguidos para la configuración y puesta a punto del ordenador de placa reducida Raspberry Pi.

4.3.1 Instalación de Home Assistant para la Raspberry PI

En el actual apartado, veremos cómo realizar la instalación y configuración del software domótico Home Assistant en la Raspberry pi 4 Model B.

Instalaremos la versión “SISTEMA OPERATIVO” como se indicó en el apartado **3.3.4** debido a su recomendación por el propio Home Assistant y a que disponemos de todas sus funcionalidades.

4.2.1.1 Hardware y software necesario para instalar Home Assistant

Para la instalación de Home Assistant debemos de disponer del siguiente Hardware:

- Raspberry pi 4 Model B
- Fuente de alimentación para la Raspberry pi
- Micro tarjeta SD
- Cable Ethernet
- Un ordenador

En cuanto al software, necesitaremos el programa balenaEtcher visto en el apartado **3.3.7**. Por tanto, realizamos su descarga desde la página web oficial de balenaEtcher (**FIGURA 48**). En nuestro caso, instalaremos la versión para Windows de 64 bits.

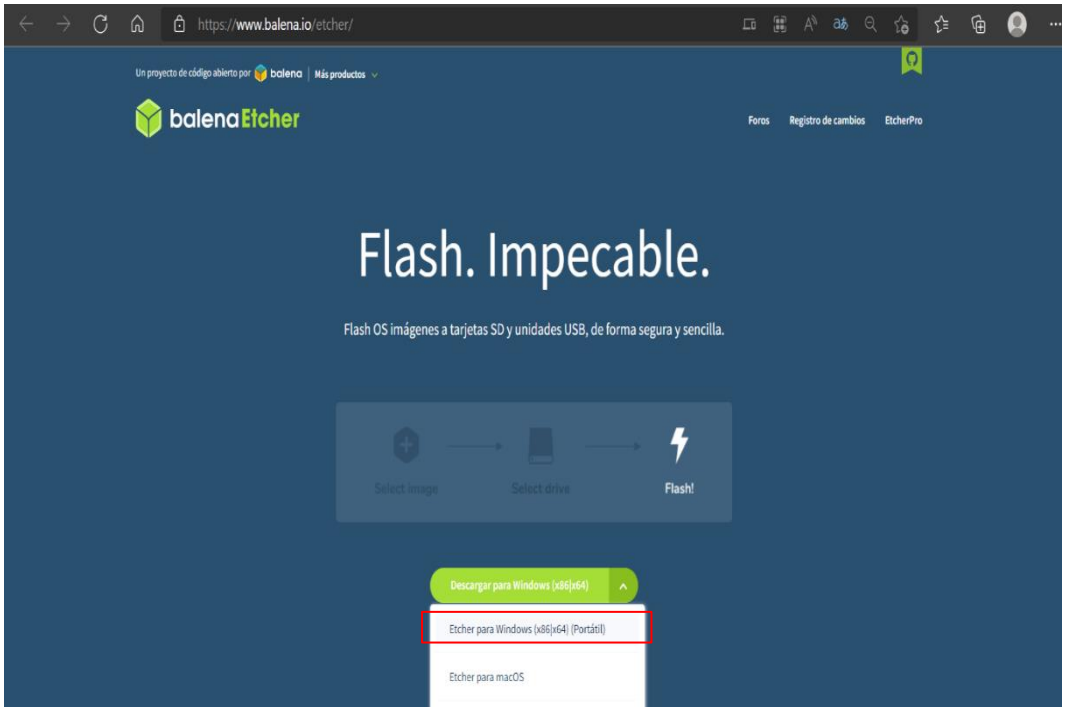

FIGURA 48. Instalación del software balenaEtcher [27]

4.2.1.2 Grabación de Home Assistant en Micro SD

Obtenida la aplicación balenaEtcher, la ejecutamos y seleccionamos "Flash from URL" (**FIGURA 49**) por lo que nos va a pedir una URL como se muestra en la **FIGURA 50**.

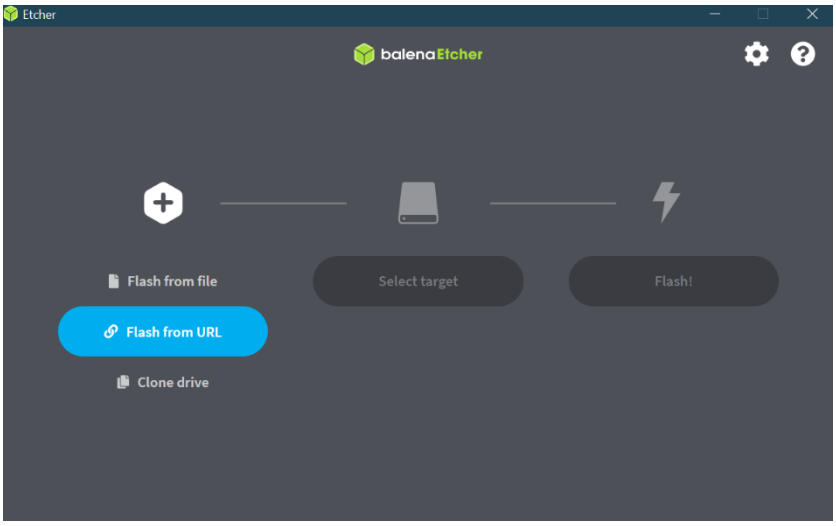

FIGURA 49. Grabación de Home Assistant a partir de una URL en balenaEtcher

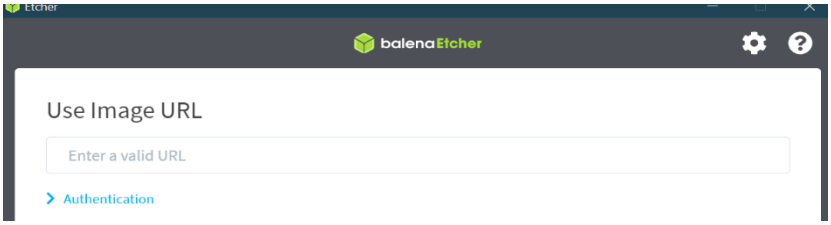

FIGURA 50. URL solicitada por balenaEtcher para la grabación de Home Assistant

La URL necesaria nos la proporciona el propio Home Assistant. Distingue entre 4 tipos posibles: de Raspberry pi 3 de 32 y 64 bits, y de Raspberry pi 4 de 32 y 64 bits.

En nuestro caso disponemos de una Raspberry pi 4, por lo que podemos elegir entre la opción de 64 bits o la opción de 32 bits. Seleccionaremos la opción de 64 bits debido a que es la recomendada por Home Assistant y permite el uso de todas las integraciones (**FIGURA 51**).

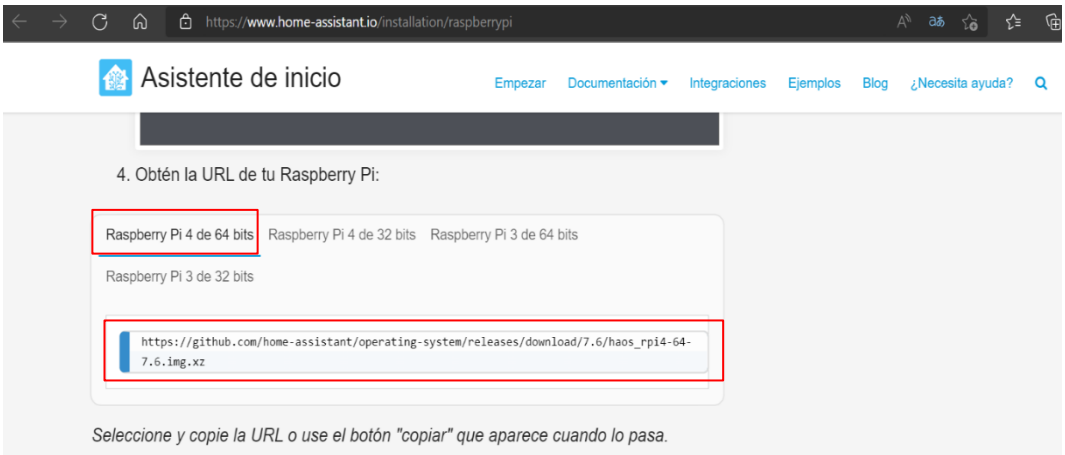

FIGURA 51. URL de la Raspberry pi 4 de 64 bits

Una vez copiada la URL que se muestra en la imagen anterior, procedemos a pegarla en el programa balenaEtcher y pulsamos "OK" (**FIGURA 52**).

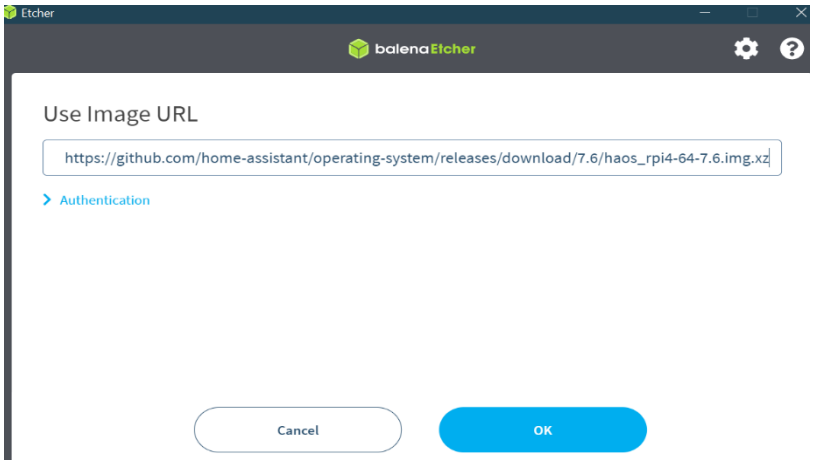

FIGURA 52. URL proporcionada por Home Assistant introducida en balenaEtcher

Aceptada la URL, el siguiente paso será seleccionar la tarjeta SD y pulsar el botón "Flash!" (**FIGURA 53**), lo que iniciará el proceso de grabación en la tarjeta Micro SD.

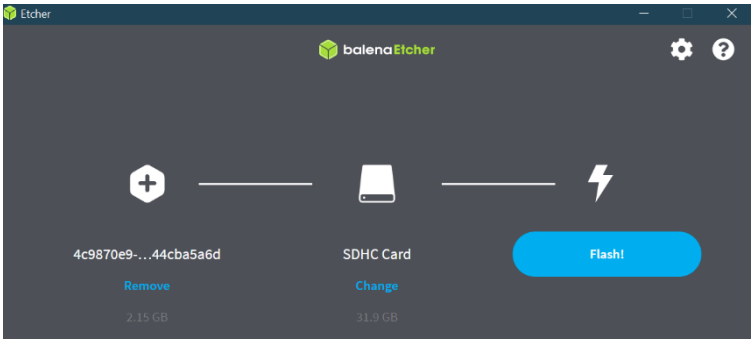

FIGURA 53. Selección de la tarjeta SD en balenaEtcher

Al finalizar el flasheado, obtendremos una imagen indicativa de que dicho proceso ha finalizado (**FIGURA 54**), con lo que ya podemos extraer la tarjeta Micro SD e introducirla en la Raspberry pi.

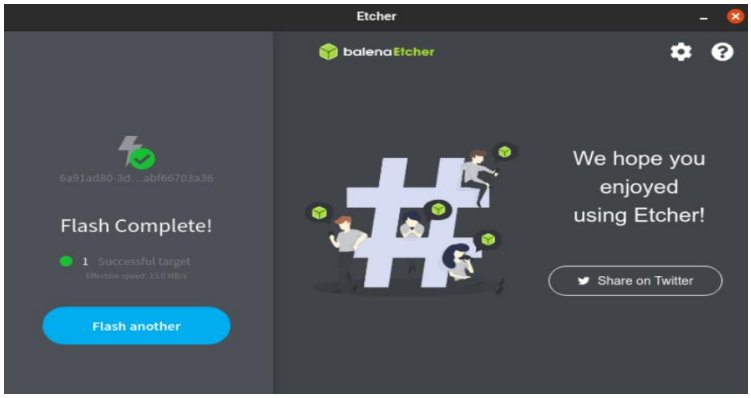

FIGURA 54. Flasheado finalizado en balenaEtcher

4.3.2 Configuración de la IP fija en Raspberry Pi

Para facilitar el acceso a Home Assistant, debemos realizar una configuración para que la Raspberry se inicie en una IP configurada previamente por nosotros. Para ello, necesitamos una memoria USB en la que copiaremos el archivo de texto con esta configuración.

En primer lugar, renombramos el nombre del dispositivo USB a “CONFIG” (**FIGURA 55**).

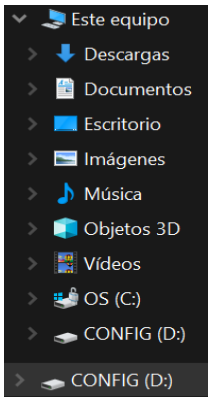

FIGURA 55. Renombramiento del dispositivo USB a "CONFIG"

En segundo lugar, crearemos una carpeta denominada “network” en el lápiz USB, la cual contendrá el archivo de texto al que denominaremos “my-network” con la configuración de la IP fija para nuestra Raspberry Pi (**FIGURA 56**).

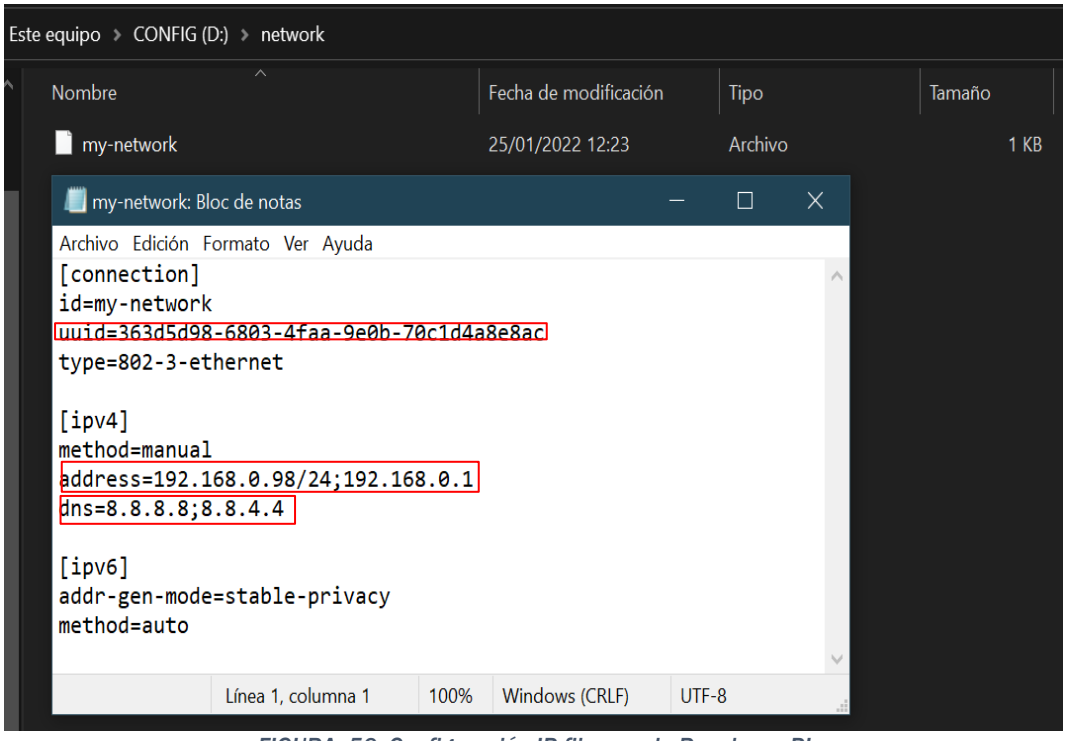

FIGURA 56. Configuración IP fija para la Raspberry PI

Para la obtención de la "uuid" usaremos un generador de UUID como se puede observar en la **FIGURA 57**. Esto nos va a permitir una identificación única e intransferible a nuestro dispositivo.

FIGURA 57. Generador de UUID para la configuración de la IP de la Raspberry Pi [31]

El siguiente paso es la configuración de la IP que va a tomar la Raspberry pi, esta dirección se configura en la parte de "address". Para dicha configuración accedemos al símbolo del sistema de nuestro ordenador e introducimos el comando "ipconfig", que nos muestra los valores de configuración completa de red de TCP/IP (**FIGURA 58**).

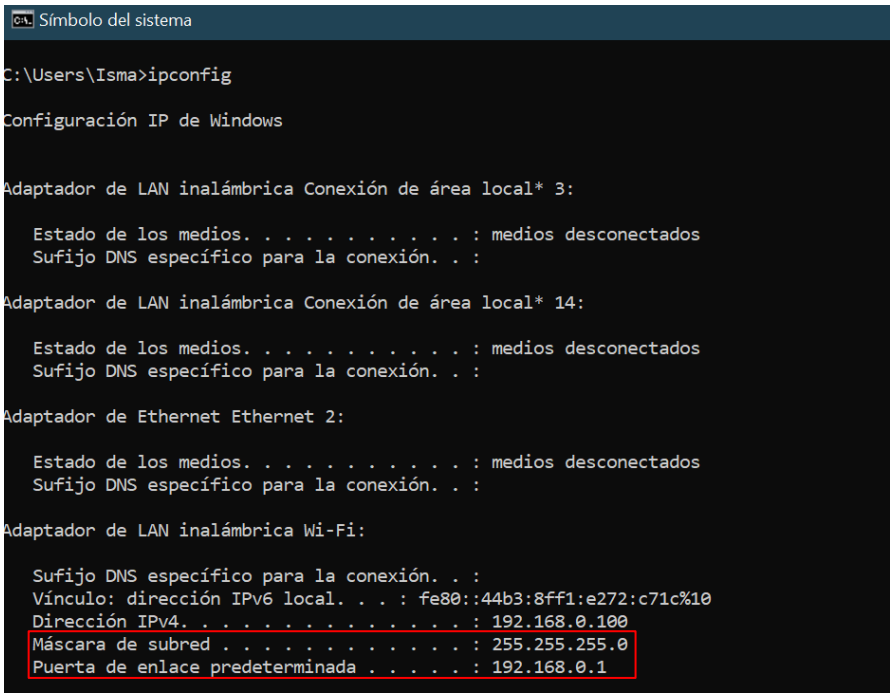

FIGURA 58. Configuración de red del TCP/IP de la red

Como se puede observar en la imagen anterior, la máscara de subred es la "255.255.255.0" y el puerto de enlace es el "192.168.0.1" (perteneciente al

router), la cual usaremos para la configuración del archivo. Por lo tanto, podemos usar de configuración de IP para la Raspberry Pi la siguiente dirección "192.168.0.xx" siendo las "xx" cualquier número comprendido entre 1 y 255, siempre y cuando dicho número no este ocupado por otro dispositivo, en este caso el 1 es el perteneciente al router y no se puede usar. Como se puede contemplar en la **FIGURA 56** hemos usado la dirección IP "192.168.0.98".

En cuanto a las "dns" usaremos las de Google (**FIGURA 59**), aunque se pueden usar cualquier otras.

FIGURA 59. DNS de Google [32]

4.3.3 Acceso a Home Assistant

Realizadas todas las acciones anteriores solo queda conectar los elementos necesarios y acceder al servidor de Home Assistant. Por consiguiente, debemos introducir la memoria USB configurada anteriormente en cualquiera de los puertos USB de los que dispone la Raspberry Pi 4, introducir la tarjeta Micro SD en la ranura para tarjetas Micro SD, conectar el cable ethernet al router y a la Raspberry y, por último, conectar el alimentador como se muestra en la **FIGURA 60**.

FIGURA 60. Conexión de todos los elementos a la Raspberry Pi

Una vez realizadas todas las conexiones, procedemos a ir al navegador y escribir la dirección IP asignada a nuestra Raspberry Pi, la cual configuramos en el apartado anterior.

Para acceder, introducimos en el navegador "192.168.0.98:8123", donde los dígitos "8123" que figuran después de la dirección IP de la Raspberry Pi indican el puerto predefinido para Home Assistant.

La primera vez que accedamos a Home Assistant, la preparación del entorno puede llevar hasta 20 minutos. Finalizada la preparación, nos pedirá crear un usuario y contraseña, para así dotar de seguridad al servidor de Home Assistant y tener un acceso personal. El siguiente paso será dar nombre a nuestra instalación y fijar la posición aproximada a nuestra localización como contemplamos en la **FIGURA 61**.

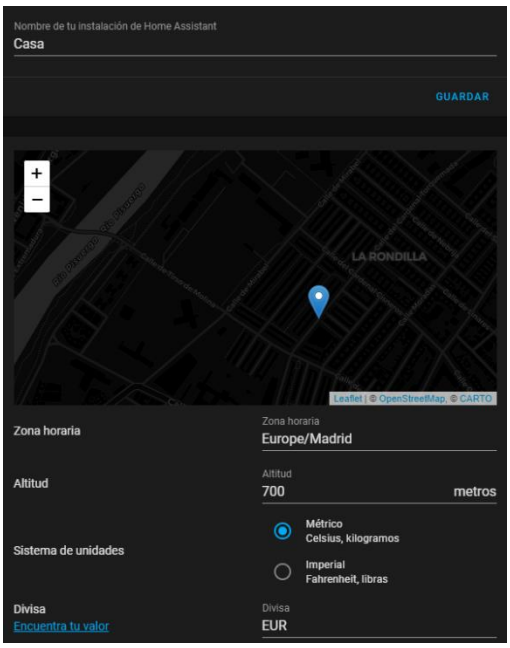

FIGURA 61. Nombre de la instalación y localización de Home Assistant

Todos estos parámetros pueden ser cambiados en cualquier momento. Una vez finalizadas estas configuraciones ya accederemos al entorno de Home Assistant (**FIGURA 62**).

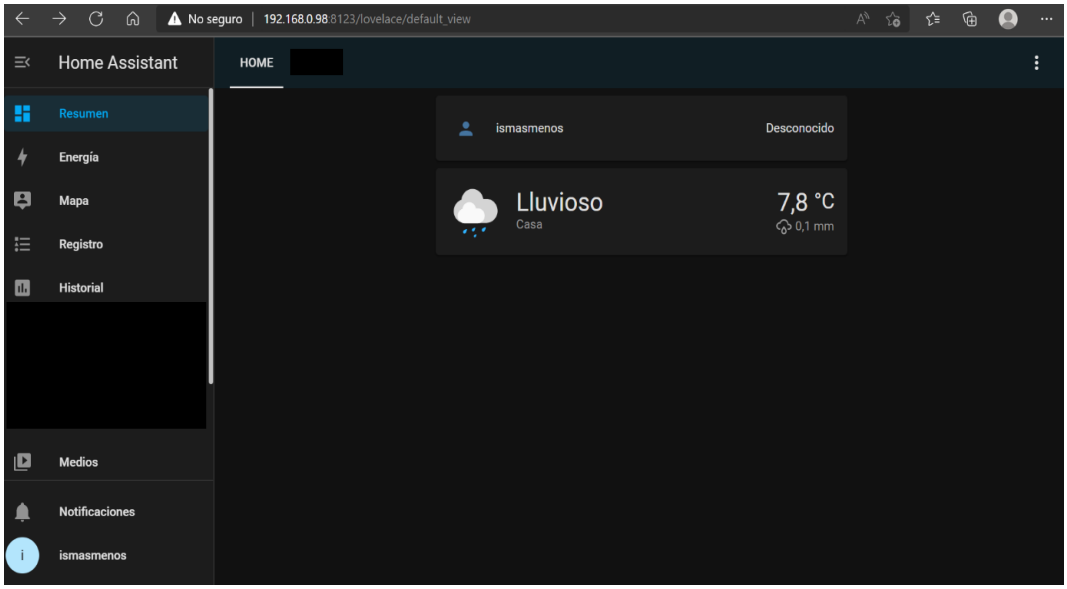

FIGURA 62. Entorno de Home Assistant

Ahora, cada vez que accedamos a Home Assistant, debemos iniciar sesión con los datos del nombre de usuario y contraseña registrados anteriormente (**FIGURA 63**).

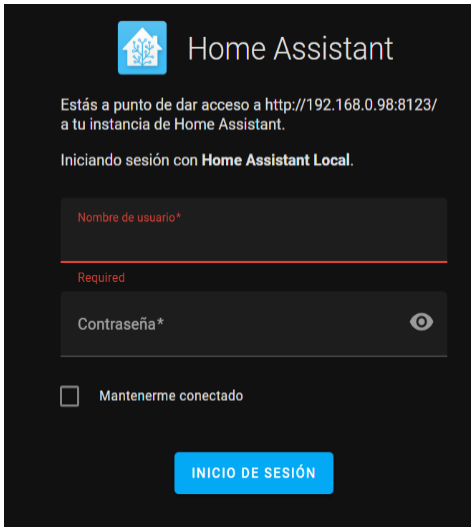

FIGURA 63. Nombre de usuario y contraseña para acceder a Home Assistant

Realizando todos los pasos anteriormente descritos, nos aseguramos de que la puesta en marcha de ambos dispositivos es correcta y su funcionamiento es el adecuado, pudiendo así empezar con el desarrollo del sistema, en lo que a código para la detección de la nariz e interfaz de usuario, comunicaciones y automatizaciones se refiere.

5 SISTEMA DESARROLLADO PARA LA ACTIVACIÓN Y DESACTIVACIÓN DE ACTUADORES MEDIANTE LA NARIZ

5.1 Introducción

En el presente capitulo, veremos cómo se ha realizado el desarrollo que permite la activación y desactivación de actuadores mediante el posicionamiento de la nariz en la pantalla del módulo Maixduino, gracias a la comunicación de dicha placa con el entorno domótico Home Assistant.

Debido a la complejidad del desarrollo, se explicará detalladamente el proceso seguido para la correcta puesta en marcha de cada uno de los elementos que componen el sistema.

5.2 Carcasa protectora para la placa de Maixduino

Para proteger, mejorar la manejabilidad y manipulación del kit Maixduino se ha impreso mediante una impresora 3D, una carcasa que envuelve a todo el dispositivo y protege la pantalla, cámara y placa de las que se compone el kit Sipeed Maixduino, evitando así posibles daños por caídas o mala manipulación.

Dicho modelo 3D está basado en un modelo de la página web dedicada a los archivos de diseño digital Thingiverse [33]. Se ha necesitado una adaptación del modelo para que cumpla con nuestros requisitos. Para la modificación de dicho modelo se ha usado la herramienta Tinkercad 3D con la que diseñaremos el logo de la UVa y el de la Escuela de Ingenierías Industriales, así como un pequeño enganche que nos va a permitir en un futuro diseñar un soporte para una batería externa para la alimentación de la placa. Este diseño dependerá del tamaño y forma de la batería seleccionada.

Para la impresión de la caja 3D que protege nuestro kit Maixduino se ha usado una impresora 3D de la que dispone el departamento de automática. El resultado de la impresión se muestra en la **FIGURA 64**.

FIGURA 64. Carcasa protectora del kit Sipeed Maixduino

5.3 Programa de detección de nariz desarrollado en el entorno MaixPy IDE

Una vez actualizado todo el firmware y flasheados todos los modelos neuronales necesarios, procedemos a desarrollar el programa capaz de detectar la nariz en el entorno de MicroPython para el procesador neuronal k210, Maixpy IDE.

Importamos la biblioteca de los sensores, imagen, lcd y tiempo, así como, la KPU (unidad de procesamiento Kendryte), perteneciente al procesador de red neuronal (**TABLA 7**), capaz de realizar cálculos de red neuronal, obtener coordenadas y tipos de objetos a detectar y detectar y clasificar caras. Se usa KPU para la aceleración de cálculo del modelo y por tanto la aceleración de hardware [23].

```
import sensor,image,lcd, time
import KPU as kpu
```

TABLA 7.Importación de las bibliotecas para la detección de la nariz en Maixpy IDE

Inicializamos y establecemos la orientación de la pantalla (**TABLA 8**).

```
lcd.init()
lcd.rotation()
```

TABLA 8. Ajuste de los parámetros de la pantalla en MaixPy IDE

A continuación, fijaremos los parámetros de los sensores. Junto a cada sentencia se incluyen los comentarios necesarios para clarificar su función (**TABLA 9**).

```
sensor.reset()#Inicializar el sensor de la cámara
sensor.set_pixformat(sensor.RGB565)#Establece el formato de
píxeles para el módulo de la cámara
sensor.set_framesize(sensor.QVGA)#Establece el tamaño de
fotograma para el módulo de cámara (320x240)
sensor.set_hmirror(1)#Configura el espejo de la cámara
sensor.run(1)#Habilita cámara
```

TABLA 9. Ajuste de los sensores en MaixPy IDE

Tras esto procede a la carga de los modelos neuronales flasheados con kflash_gui (**TABLA 10**).

```
task_fd = kpu.load(0x300000) # Cargue el modelo de detección de
rostros desde flash 0x300000
task_ld = kpu.load(0x400000) # Cargue el modelo de detección de
puntos clave de la cara desde el flash 0x400000
```

TABLA 10. Carga de los modelos neuronales de detección y puntos clave del rostro en MaixPy IDE

A continuación, establecemos los anclajes de detección de rostro. Estos anclajes consisten en un "cuadro" preestablecido para el modelo de detección de objetos YOLO v2, algoritmo de detección de objetos más utilizado en la actualidad y que

permite detectar múltiples objetos en una imagen [34]. Además, asignamos a una cierta variable ("IMDR": Inicializar el modelo de detección de rostros) el modelo de detección de rostros al cual pasamos el anclaje y el modelo neuronal correspondiente a la detección de rostros (**TABLA 11**).

```
anchor = (1.889, 2.5245, 2.9465, 3.94056, 3.99987, 5.3658,
5.155437, 6.92275, 6.718375, 9.01025) #Ancla para detección de
rostros
IMDR = kpu.init_yolo2(task_fd, 0.5, 0.3, 5, anchor)#Inicializar
el modelo de detección de rostros
```

TABLA 11. Ancla e inicialización del modelo de detección de rostros.

Por último, programamos el bucle que se va a estar ejecutando continuamente y cuyas funciones vienen especificadas en la **TABLA 12.**

```
while(1):
    imagen = sensor.snapshot()#Obtiene una foto de la cámara
    rostro = kpu.run_yolo2(task_fd, imagen) #Ejecute el modelo de
        detección de rostros para obtener la posición de las
        coordenadas del rostro

    if rostro: # Si se detecta un rostro
        for i in rostro:
            # Cortar la cara y cambiar el tamaño a 128x128
            corte_cara= imagen.cut(i.x(),i.y(),i.w(),i.h())
            corte_cara_128= corte_cara.resize(128,128)
            IMDR = corte_cara_128.pix_to_ai()#Convierta la imagen
            detectada al formato aceptado por kpu

            # Marcas de la cara
            # Ejecute el modelo de puntos clave del rostro
            mapa_cara = kpu.forward(task_ld, corte_cara_128)
            plist= mapa_cara[:] # Obtiene resultados clave de
            predicción de la nariz
            nariz=(i.x()+int(plist[4]*i.w()),
                   i.y()+int(plist[5]*i.h()))
            IMDR = imagen.draw_cross(nariz[0], nariz[1], 10,
            color=(255,0,0),thickness=2)# Dibuja sobre la nariz
             una cruz
            IMDR = lcd.display(imagen) # Mostramos por pantalla
la imagen
```

TABLA 12.Bucle que contiene las funciones para detectar la nariz y mostrarlo en la pantalla

El bucle principal captura la imagen y ejecuta el modelo de detección de rostro. En el caso de detectar algún rostro, se recorta la cara y se ajusta a un tamaño de 128x128, un tamaño aceptado por la unidad de procesamiento (KPU). A continuación, ejecuta el modelo de puntos clave del rostro, del cual extraemos la posición de la nariz. Por último, dibujamos una cruz sobre la nariz y lo mostramos por la pantalla (**FIGURA 65**).

FIGURA 65. Muestra de la detección de la nariz en la pantalla LCD del Maixduino

5.4 Desarrollo de la interfaz de usuario en MaixPy IDE

Tal como se ha indicado, la posición de la nariz será utilizada para la activación de una serie de botones digitales mostrados en la pantalla LCD de la que dispone nuestro kit de Maixduino.

La interfaz, en este caso, va a consistir en cuatro botones digitales (el número de botones puede variar en función de las necesidades de cada paciente), cada uno de los cuales realizará una acción diferente. Los botones, una vez se haya posicionado la nariz sobre ellos, se iluminarán en color rojo, permaneciendo así durante cuatro segundos, pasando a iluminarse en color verde, indicando así que la acción deseada ha tenido lugar. Para desactivar la acción realizada, el procedimiento es el mismo: permanecer con la nariz sobre el botón durante cuatro segundos hasta que se ilumine en rojo. Por lo tanto, la iluminación de los botones indica al usuario que está posicionado correctamente sobre ellos y, además, indica si la acción se ha activado o desactivado.

La elección del tiempo de activación/desactivación de cuatro segundos de los botones, se ha elegido de tal forma que, sea lo suficientemente grande para que no se produzcan activaciones/desactivaciones no deseadas y teniendo en cuenta que no puede ser una duración mucho más prolongado debido a la incomodidad que esto provocaría en el usuario.

El código para la interfaz de los cuatro botones se integrará dentro del bucle comentado en la **TABLA 12**. En la **TABLA 13** se muestra el resultado final, con el

nuevo código resaltado en letra negrita.

```
# importar bibliotecas relacionadas
import sensor,image,lcd, time
import KPU as kpu

# Globals
Boton_1 = 0
Boton_2 = 0
Boton_3 = 0
Boton_4 = 0
contador = 0
Tiempo_act_des = 4

lcd.init()
lcd.rotation()
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.set_hmirror(1)
sensor.run(1)

task_fd = kpu.load(0x300000)
task_ld = kpu.load(0x400000)

anchor = (1.889, 2.5245, 2.9465, 3.94056, 3.99987, 5.3658,
5.155437, 6.92275, 6.718375, 9.01025)
IMDR = kpu.init_yolo2(task_fd, 0.5, 0.3, 5, anchor)

while(1):
    imagen = sensor.snapshot()
    rostro = kpu.run_yolo2(task_fd, imagen)

    if rostro:
        for i in rostro:
            corte_cara=imagen.cut(i.x(),i.y(),i.w(),i.h())
            corte_cara_128= corte_cara.resize(128,128)
            IMDR = corte_cara_128.pix_to_ai()

            mapa_cara = kpu.forward(task_ld, corte_cara_128)
            plist= mapa_cara[:]
            nariz=(i.x()+int(plist[4]*i.w()),
                   i.y()+int(plist[5]*i.h()))
            IMDR = imagen.draw_cross(nariz[0], nariz[1], 10,
                             color=(255,0,0),thickness=2)

            if (Boton_1 == 0 and (nariz[0]>70 and nariz[0]<110)
                and (nariz[1]>110 and nariz[1]<150)):
                contador=contador+1
                tiempo=contador/6
                if (tiempo <= Tiempo_act_des):
                    IMDR = imagen.draw_circle(90, 130,20,
                       color=(255,0,0),thickness=12)
                if (tiempo >= Tiempo_act_des):
                    Boton_1 = 1
                    contador = 0
                    print('Boton 1: ON {}'.format(Boton_1))
```

```
            elif (Boton_1 == 1 and (nariz[0]>70 and
             nariz[0]<110) and (nariz[1]>110 and nariz[1]<150)):
                contador=contador+1
                tiempo=contador/6
                if (tiempo <= Tiempo_act_des):
                    IMDR = imagen.draw_circle(90, 130,20,
                                color=(0,255,0),thickness=12)
                if (tiempo >= Tiempo_act_des):
                    Boton_1 = 0
                    contador = 0
                    print('Boton 1: OFF {}'.format(Boton_1))

            elif (Boton_2 == 0 and (nariz[0]>100 and
             nariz[0]<140) and (nariz[1]>160 and nariz[1]<200)):
                contador=contador+1
                tiempo=contador/6
                if (tiempo <= Tiempo_act_des):
                    IMDR = imagen.draw_circle(120, 180,20,
                               color=(255,0,0),thickness=12)
                if (tiempo >= Tiempo_act_des):
                    Boton_2 = 1
                    contador = 0
                    print('Boton 2: ON {}'.format(Boton_2))
            elif (Boton_2 == 1 and (nariz[0]>100 and
             nariz[0]<140) and (nariz[1]>160 and nariz[1]<200)):
                contador=contador+1
                tiempo=contador/6
                if (tiempo <= Tiempo_act_des):
                    IMDR = imagen.draw_circle(120, 180,20,
                                color=(0,255,0),thickness=12)
                if (tiempo >= Tiempo_act_des):
                    Boton_2 = 0
                    contador = 0
                    print('Boton 2: OFF {}'.format(Boton_2))

            elif (Boton_3 == 0 and (nariz[0]>170 and
             nariz[0]<210) and (nariz[1]>160 and nariz[1]<200)):
                contador=contador+1
                tiempo=contador/6
                if (tiempo <= Tiempo_act_des):
                    IMDR = imagen.draw_circle(185, 180,20,
                               color=(255,0,0),thickness=12)
                if (tiempo >= Tiempo_act_des):
                    Boton_3 = 1
                    contador = 0
                    print('Boton 3: ON {}'.format(Boton_3))
            elif (Boton_3 == 1 and (nariz[0]>170 and
             nariz[0]<210) and (nariz[1]>160 and nariz[1]<200)):
                    contador=contador+1
                    tiempo=contador/6
                    if (tiempo <= Tiempo_act_des):
                        IMDR = imagen.draw_circle(185, 180,20,
                                    color=(0,255,0),thickness=12)
                    if (tiempo >= Tiempo_act_des):
                        Boton_3 = 0
                        contador = 0
                        print('Boton 3: OFF {}'.format(Boton_3))
```

```
            elif (Boton_4 == 0 and (nariz[0]>190 and
             nariz[0]<230) and (nariz[1]>110 and nariz[1]<150)):
                contador=contador+1
                tiempo=contador/6
                if (tiempo <= Tiempo_act_des):
                    a = imagen.draw_circle(210, 130,20,
                                color=(255,0,0),thickness=12)
                if (tiempo >= Tiempo_act_des):
                    Boton_4 = 1
                    contador = 0
                    print('Boton 4: ON {}'.format(Boton_4))
            elif (Boton_4 == 1 and (nariz[0]>190 and
            nariz[0]<230) and (nariz[1]>110 and nariz[1]<150)):
                    contador=contador+1
                    tiempo=contador/6
                    if (tiempo <= Tiempo_act_des):
                        IMDR = imagen.draw_circle(210, 130,20,
                                color=(0,255,0),thickness=12)
                    if (tiempo >= Tiempo_act_des):
                        Boton_4 = 0
                        contador = 0
                        print('Boton 4: OFF {}'.format(Boton_4))

            else:
                contador=0

            IMDR = imagen.draw_string(85,115, ("1"),
                                color=(255,128,0),scale=3)
            IMDR = imagen.draw_string(113,165,("2"),
                                color=(255,128,0),scale=3)
            IMDR = imagen.draw_string(175,165,("3"),
                                color=(255,128,0),scale=3)
            IMDR = imagen.draw_string(200,115,("4"),
                                color=(255,128,0),scale=3)
            IMDR = imagen.draw_circle(90, 130,20,
                                color=(0,0,0),thickness=8)
            IMDR = imagen.draw_circle(120, 180,20,
                                color=(0,0,0),thickness= 8)
            IMDR = imagen.draw_circle(185, 180,20,
                                color=(0,0,0),thickness= 8)
            IMDR = imagen.draw_circle(210, 130,20,
                                color=(0,0,0),thickness= 8)
            IMDR = lcd.display(imagen) # Mostramos por pantalla
                                        la imagen
```

TABLA 13.Código desarrollado para la creación de la interfaz de usuario en MaixPy IDE

Como se puede observar, el código para el desarrollo de la interfaz consiste mayormente en sentencias condicionales.

Las sentencias condicionales en función de la posición de la nariz y el estado del botón (activado/ desactivado) permiten activar un temporizador, el cual una vez pasados los cuatro segundos se produce un *reset* y cambia el estado del botón a rojo (desactivado) o a verde (activado).

El tiempo de posicionamiento de la nariz en cada botón para el cambio de estado

del mismo, se ha programado como una variable global (Tiempo_act_des), para su mayor facilidad de modificación y adaptación a cada usuario.

Para dibujar los botones en forma circular se ha usado la función "`imagen.draw_circle()`", a la cual se le han pasado como parámetros, la posición del centro, el diámetro(en nuestro caso 20 mm), el color (negro) y el grosor (8 mm), como se puede observar en las últimas sentencias mostradas en la **13**.

Para enumerar los botones se ha usado la función "`imagen.draw_string()`", a la que se han pasado como parámetros la posición del carácter, su forma (en este caso números), el color (naranja) y el escalado. En la **FIGURA 66** se muestran los botones iluminados en color rojo al estar posicionado con la nariz sobre ellos, mientras que en la **FIGURA 67** se muestran en color verde que indica el cambio de estado.

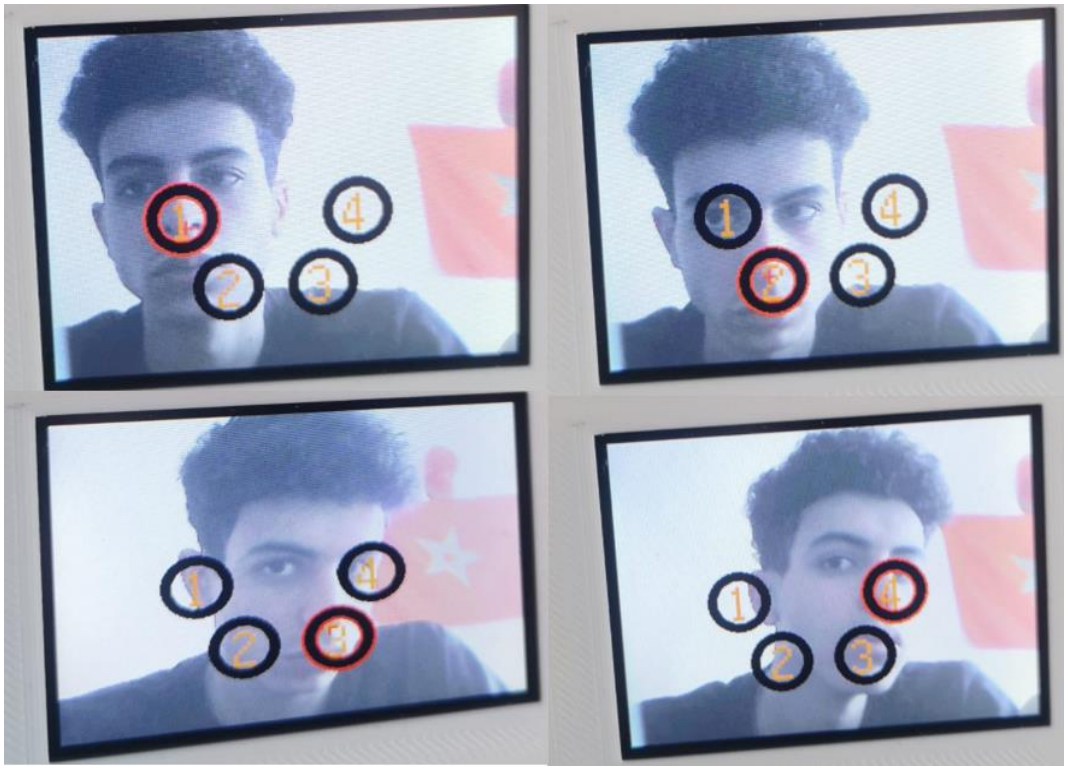

FIGURA 66. Interfaz de usuario mostrada en la pantalla del Maixduino (botones desactivados)

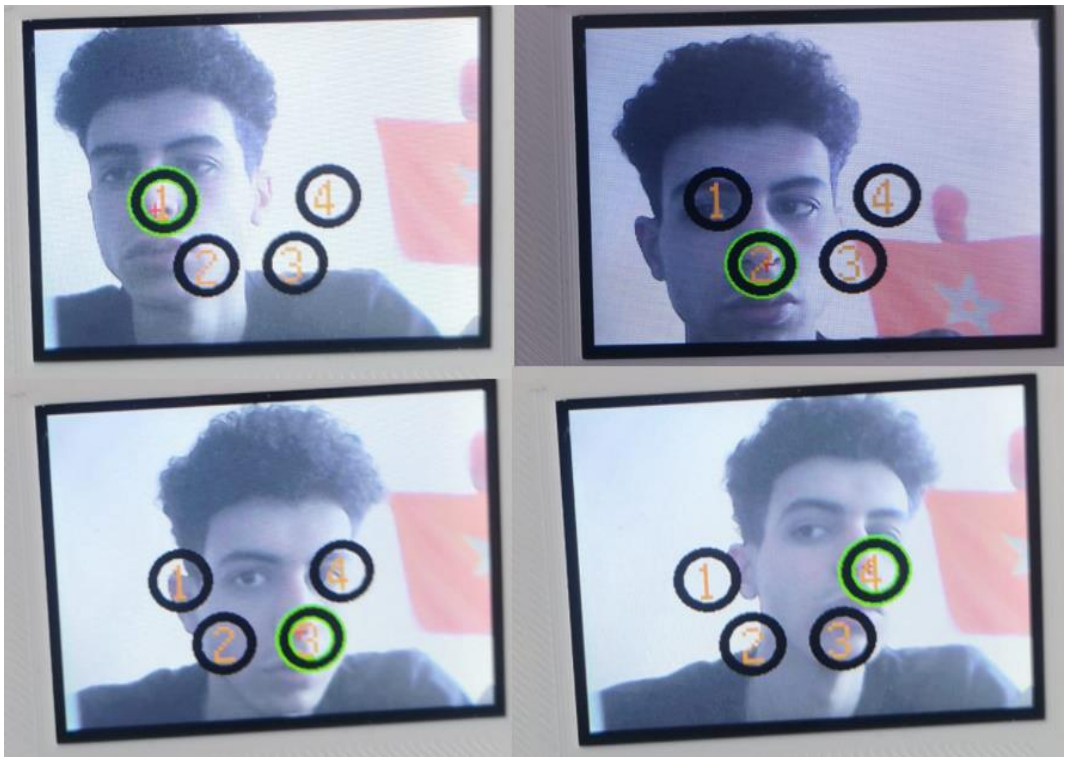

FIGURA 67. Interfaz de usuario mostrada en la pantalla del Maixduino (botones activados)

5.5 Conexión a Wifi del Maixduino

En el presente apartado mostraremos el programa que permite conectar nuestra placa Maixduino a la red Wifi, con el objetivo del envío de flujo de datos y la comunicación con la aplicación domótica Home Assistant. El código que permite realizar esta función es el mostrado en la **TABLA 14**.

```
import network, utime
from Maix import GPIO
from fpioa_manager import fm


WIFI_SSID = "ismasmenos"
WIFI_PASSWORD = "Bouaouda99"


class wifi():

    nic = None

    def reset(force=False, reply=5, is_hard=True):
        if force == False and __class__.isconnected():
            return True
        try:
            # IO map for ESP32 on Maixduino
```

```
            fm.register(25,fm.fpioa.GPIOHS10)#cs
            fm.register(8,fm.fpioa.GPIOHS11)#rst
            fm.register(9,fm.fpioa.GPIOHS12)#rdy

            if is_hard:
                print(" Hardware SPI para Maixduino")
                fm.register(28,fm.fpioa.SPI1_D0,
force=True)#mosi
                fm.register(26,fm.fpioa.SPI1_D1,
force=True)#miso
                fm.register(27,fm.fpioa.SPI1_SCLK,
force=True)#sclk
                __class__.nic =
network.ESP32_SPI(cs=fm.fpioa.GPIOHS10, rst=fm.fpioa.GPIOHS11,
rdy=fm.fpioa.GPIOHS12, spi=1)
                print("ESP32_SPI firmware version:",
__class__.nic.version())
            else:

                print("Use Software SPI for other hardware")
                fm.register(28,fm.fpioa.GPIOHS13,
force=True)#mosi
                fm.register(26,fm.fpioa.GPIOHS14,
force=True)#miso
                fm.register(27,fm.fpioa.GPIOHS15,
force=True)#sclk
                __class__.nic =
network.ESP32_SPI(cs=fm.fpioa.GPIOHS10,rst=fm.fpioa.GPIOHS11,rdy
=fm.fpioa.GPIOHS12,
mosi=fm.fpioa.GPIOHS13,miso=fm.fpioa.GPIOHS14,sclk=fm.fpioa.GPIO
HS15)
                print("ESP32_SPI firmware version:",
__class__.nic.version())

        except Exception as e:
            print(e)
            return False
        return True

    def connect(ssid="wifi_name", pasw="pass_word"):
        if __class__.nic != None:
            return __class__.nic.connect(ssid, pasw)

    def ifconfig():
        if __class__.nic != None:
            return __class__.nic.ifconfig()

    def isconnected():
        if __class__.nic != None:
            return __class__.nic.isconnected()
        return False

def connect_wifi():
    while not wifi.isconnected():
        print("--- Reset ESP32 ---")
        wifi.reset()
        print("--- Conectando a Wi-Fi ---")
        try:
            wifi.connect(WIFI_SSID, WIFI_PASSWORD)
```

```
        except Exception as e:
            print("Conexión fallida: {}".format(e))

    print("Wi-Fi conectado a:", wifi.isconnected(),
wifi.ifconfig())



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

TABLA 14. Código en Maixpy IDE para conectar a la red Wifi el Maixduino

El código se carga en el puerto correspondiente al chip neuronal K210 (puerto 20). Por tanto, para la conexión de la placa a la red Wifi, este procesador se debe comunicar con el chip ESP32, lo cual se consigue mediante una serie de pines integrados en la placa. Para la comunicación entre ambos módulos se utiliza el Bus SPI. El Bus SPI es un estándar de comunicación, usado para la transferencia de información entre circuitos integrados, por lo cual, para la comunicación entre ambos procesadores se asigna a la función SPI los pines que se muestran en la **FIGURA 68**.

Maixduino (PIN ASSIGNMENT TABLE)					
Maixduino Slik	K210 IO	ESP32 IO	Function	Remark	IO Voltage
RST	Dedicated pin		K210_RST	10K pull up	1.8V
	IO0		JTAG_TCK		
	IO1		JTAG_TDI		
	IO2		JTAG_TMS		
13	IO3		JTAG_TDO		
RX←0	IO4		K210_RX		
TX→1	IO5		K210_TX		
	IO6	IO1	ESP32_U0TX		
	IO7	IO3	ESP32_U0RX		
	IO8	Dedicated pin	ESP32_EN		
	IO9	IO25	ESP32_READY		
12	IO10				
11	IO11				
10	IO12		LED_G		
9	IO13		LED_R		
8	IO14		LED_B		
7	IO15				
	IO16		K210_BOOT	10K pull up	
	IO17		LCD_Backlight	10K pull down(on)	
	IO18		MIC_BCK	MEMS MIC	3.3V
	IO19		MIC_WS		
	IO20		MIC_DAT3		
2	IO21				
3	IO22				
4	IO23				
5	IO24				
	IO25	IO5	ESP32_SPI_CS		
	IO26	IO23	SPI0_MISO		
	IO27	IO18	SPI0_SCLK	TF Card	
	IO28	IO14	SPI0_MOSI		

FIGURA 68. Asignación de pines entre el procesador K210 y el módulo ESP32

Una vez enlazada la placa con el entorno de desarrollo, ejecutamos el script y obtenemos en el terminal serie que la conexión se ha realizado correctamente, además de la IP de la red a la que se ha conectado, la IP del dispositivo y la máscara de subred (**FIGURA 69**).

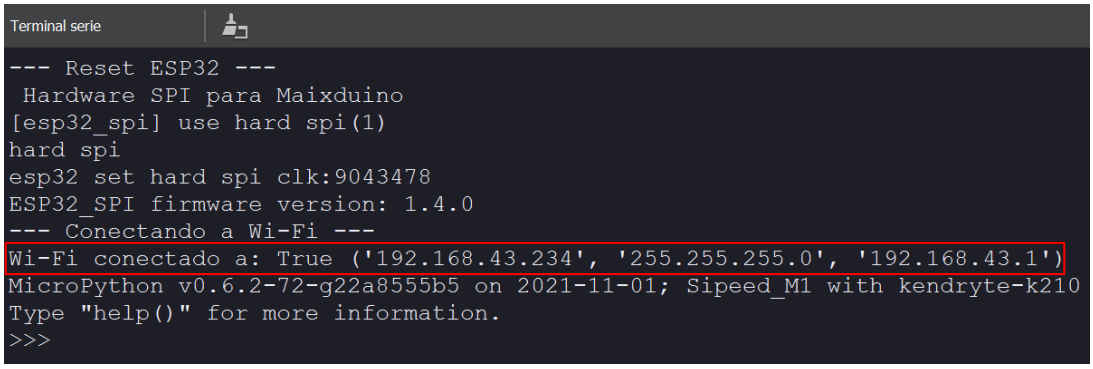

FIGURA 69. Conexión de la placa Maixduino a una red Wifi

5.6 Comunicación entre Maixduino y Raspberry Pi

Para el envío de información del módulo Maixduino a la placa Raspberry Pi para su visualización en Home Assistant, se ha utilizado el protocolo de comunicación MQTT. Se ha elegido el protocolo MQTT debido a su sencillez y robustez a la hora del envío de flujo de datos.

Para la correcta comunicación entre ambos dispositivos resulta necesario configurar el servidor de Home Assistant de forma que disponga del protocolo MQTT y se ha programado en Maixpy IDE una clase MQTT que permite el envío de flujo de datos desde el Maixduino a través de una conexión Wifi. Todo ello viene explicado detalladamente a continuación.

5.6.1 Protocolo MQTT

MQTT consiste en un protocolo de comunicación M2M (máquina a máquina), es decir, está diseñado para comunicar dispositivos entre sí. Está basado en la pila TCP/IP para la comunicación y utiliza la topología de publicar/suscribir.

Para poder comprender el funcionamiento de este protocolo hay que entender los siguientes conceptos [35]:

- **Broker** es el servidor que distribuye la información entre los distintos clientes.
- **Cliente** es el dispositivo que se conecta al servidor tanto para enviar como para captar información.
- **Topic** es el tema al que se suscriben o en el que envían información los clientes.

- **QoS** es la calidad del servicio. Éste puede tomar los valores 0, 1 o 2. En función de dicho valor se tiene una calidad de servicio u otra a la hora del envío de los mensajes. El nivel 2 es el que nos aporta una entrega asegurada y solo una copia del mensaje. El nivel 1 consiste en que se envía varias veces el mensaje hasta que se indica que se ha recibido dicho mensaje. El nivel 0 envía el mensaje solo una vez, lo que puede provocar que, en caso de no ser recibido, dicho mensaje se pierda.

Una vez comprendidos estos conceptos se puede explicar el funcionamiento de la arquitectura publicación/suscripción empleada por el protocolo MQTT.

Un cliente emisor publica al servidor Broker un mensaje e indica qué tema le corresponde a dicho mensaje, sin saber quién será el destinatario. Ahora un cliente se suscribe a dicho tema a partir del servidor Broker y obtiene la información del mensaje, este mensaje puede tener cualquier formato. Un ejemplo sencillo de este tipo de arquitectura se muestra en la **FIGURA 70**.

FIGURA 70. Arquitectura publicación/suscripción del protocolo MQTT [35]

En nuestro caso, el cliente que se va a encargar de publicar toda la información es el módulo Maixduino, mientras que el encargado de recibir dicha información es la placa Raspberry Pi. Por tanto, la Raspberry Pi que contiene el software Home Assistant se va a usar tanto de Broker (servidor de Home Assistant) como de cliente suscrito, ya que toda la información enviada por el módulo Maixduino se va a ver reflejada en el monitor de la interfaz de Home Assistant.

Un esquema explicativo de ello es el de la **FIGURA 71** , que indica los tópicos utilizados, cada uno correspondiente a uno de los botones programados en el Maixduino. De esta forma, cada vez que cambie de estado un botón se enviará dicha información a Home Assistant, mediante el tópico correspondiente.

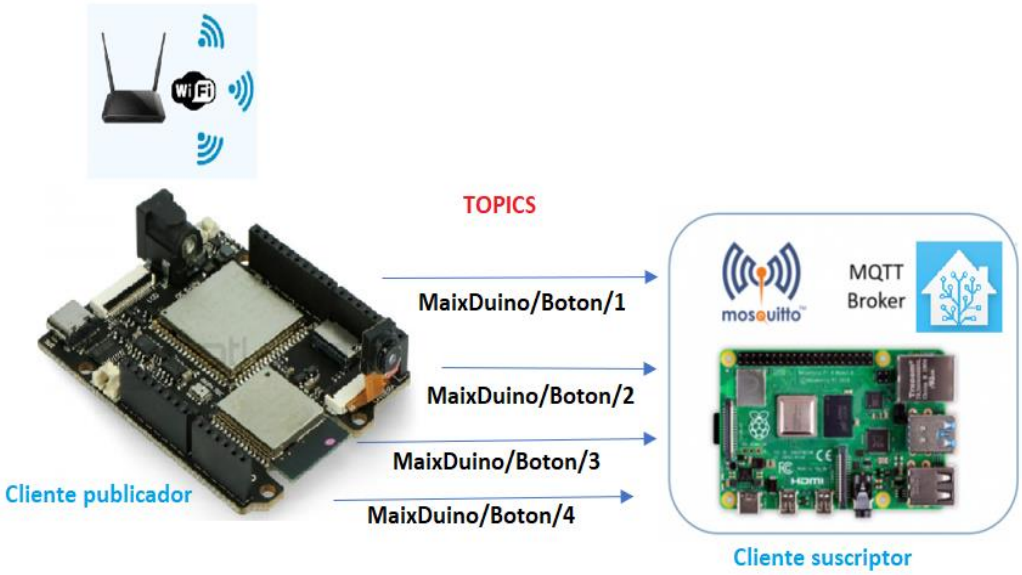

FIGURA 71. Arquitectura publicación/suscripción entre Maixduino y Raspberry Pi

A continuación, se mostrará tanto la configuración realizada en Home Assistant como el programa desarrollado en Maixpy IDE para conseguir que ambos dispositivos se comuniquen entre sí.

5.6.2 MQTT en Home Assistant

Para poder utilizar el protocolo MQTT debemos instalar el complemento MQTT disponible en la tienda oficial que nos proporciona el propio Home Assistant. Para ello, se debe acceder a **Configuración > Complementos, Copias de seguridad y Supervisor > Tienda de complementos**. En el buscador introducimos "MQTT" y seleccionamos e instalamos dicho complemento. Una vez realizada la instalación, procedemos a activar la "Vigilancia" que permite reiniciar MQTT en caso de caída o fallo de conexión del protocolo (**FIGURA 72**).

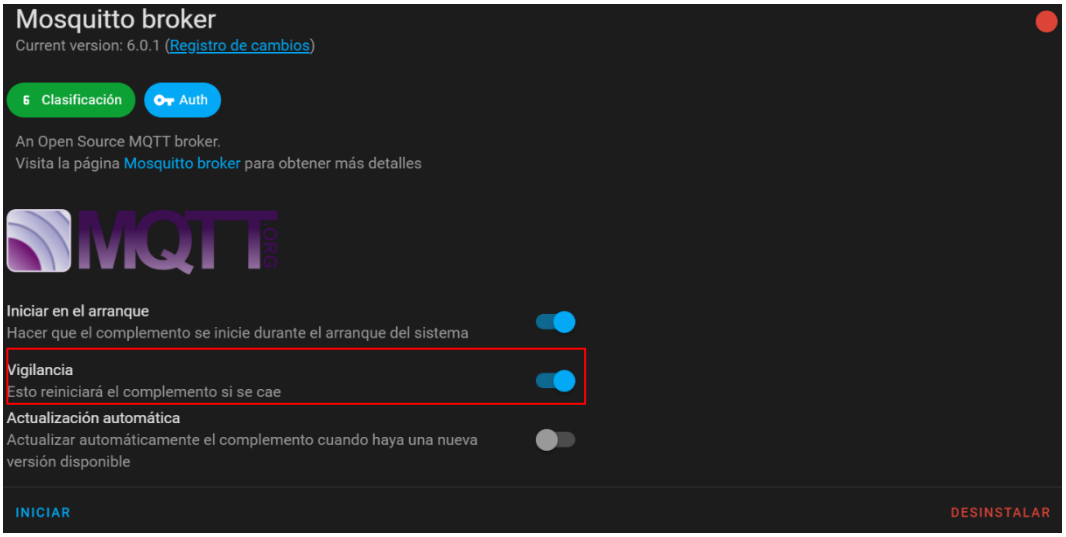

FIGURA 72. Instalación de MQTT en Home Assistant

Seguidamente, vamos a configuración y en "logins" pondremos un usuario y una contraseña (no se permite el uso de homeassistant ni addon como nombre de usuario), lo que dotará de seguridad a nuestra comunicación, posteriormente guardamos los cambios realizados (**FIGURA 73**). El resto de los parámetros se deben dejar por defecto ya que no afectan al funcionamiento. Una vez hechos estos ajustes, iniciamos MQTT accediendo a información y pulsando "INICIAR".

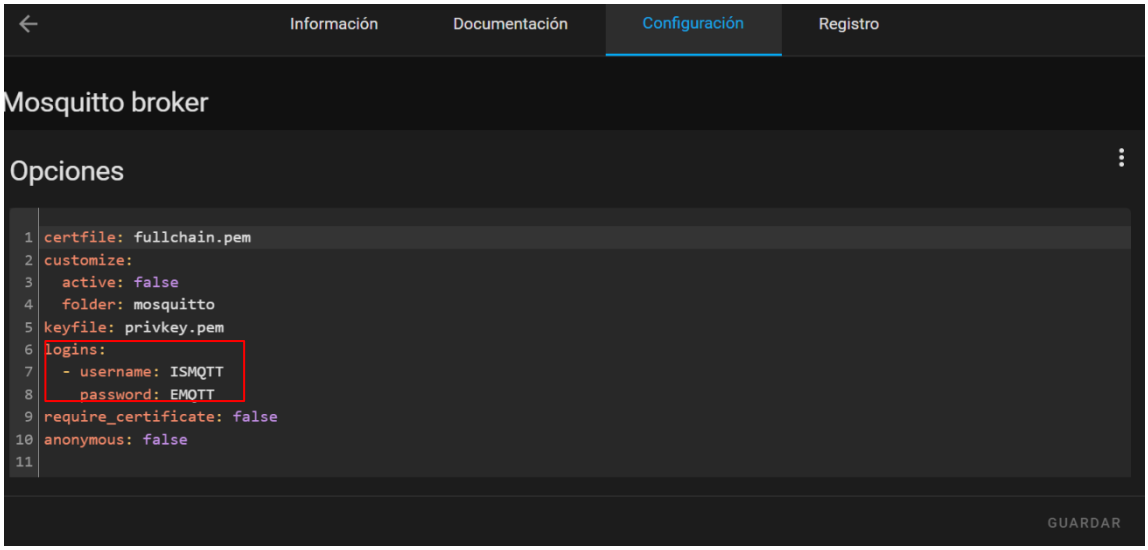

FIGURA 73. Usuario y contraseña para MQTT

Como puerto MQTT hemos optado por el proporcionado por defecto y que es el usado normalmente, como se indica en la **FIGURA 74**.

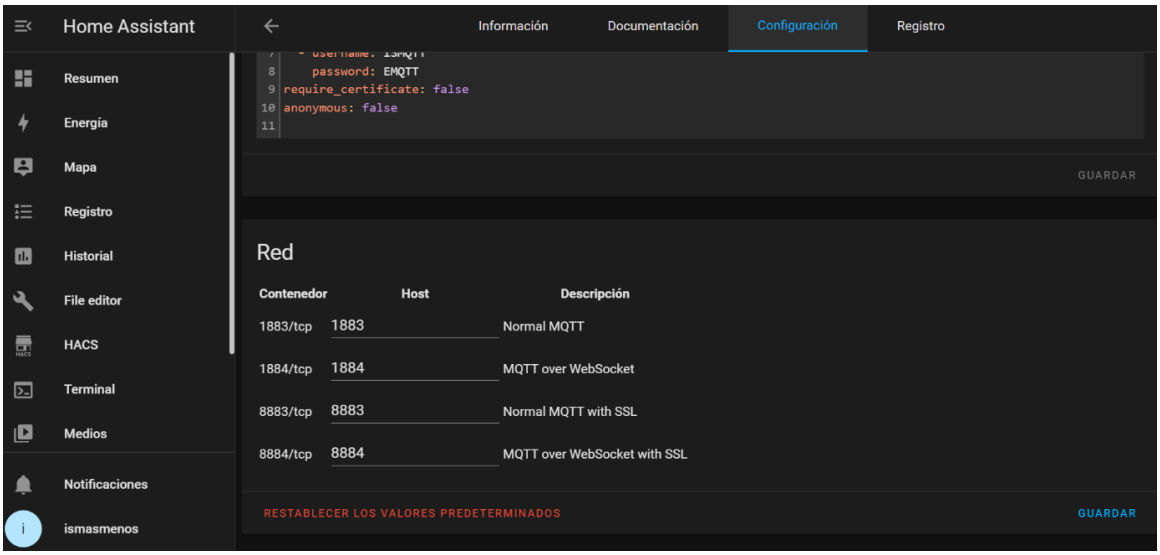

FIGURA 74. Configuración de Red del protocolo MQTT en Home Assistant

Ahora, al acceder a **Configuración > Dispositivos y servicios** se podrá comprobar que el complemento MQTT instalado ha sido detectado como una integración, (**FIGURA 75**).

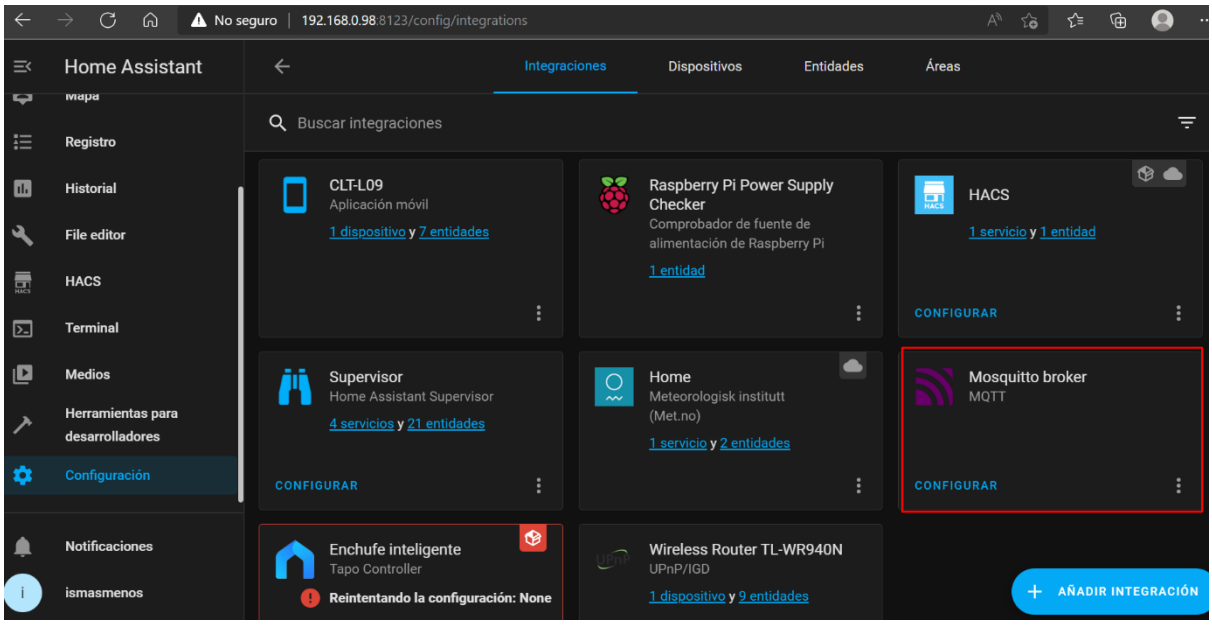

FIGURA 75. MQTT detectado como integración en Home Assistant

Para comprobar el funcionamiento del protocolo de comunicación MQTT, accedemos a la configuración de la integración MQTT y enviamos un mensaje desde la plantilla que nos ofrece Home Assistant como se muestra en la **FIGURA 76**.

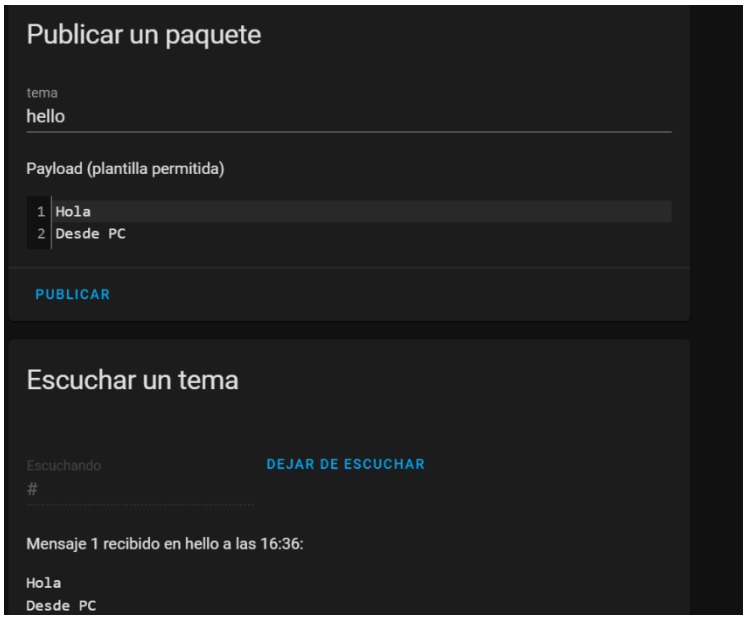

FIGURA 76. Envío de un mensaje mediante el protocolo MQTT en el propio Home Assistant

Tras comprobar el funcionamiento de la integración MQTT, procedemos a configurar los tópicos (canales) que se van a utilizar para el intercambio de datos entre Maixduino y Raspberry Pi.

Maixduino será el encargado de enviar la información del estado de los botones a través de los canales y la Raspberry Pi de recibirlos y mostrarlos en la interfaz de Home Assistant.

Para la configuración de los distintos tópicos accedemos a **File editor > configuration.yaml** como se indica en la **FIGURA 77**. En dicha carpeta indicamos el tipo de sistema, en este caso, es un sensor que cambia de estado de 0 a 1 (desactivado/activado) correspondiente al estado de los botones digitales. Por cada botón digital del que disponemos, debemos crear un tópico que usaremos para el envío del estado de cada botón como se contempla en la **FIGURA 78**, y al cual asignamos un nombre y una unidad de medida.

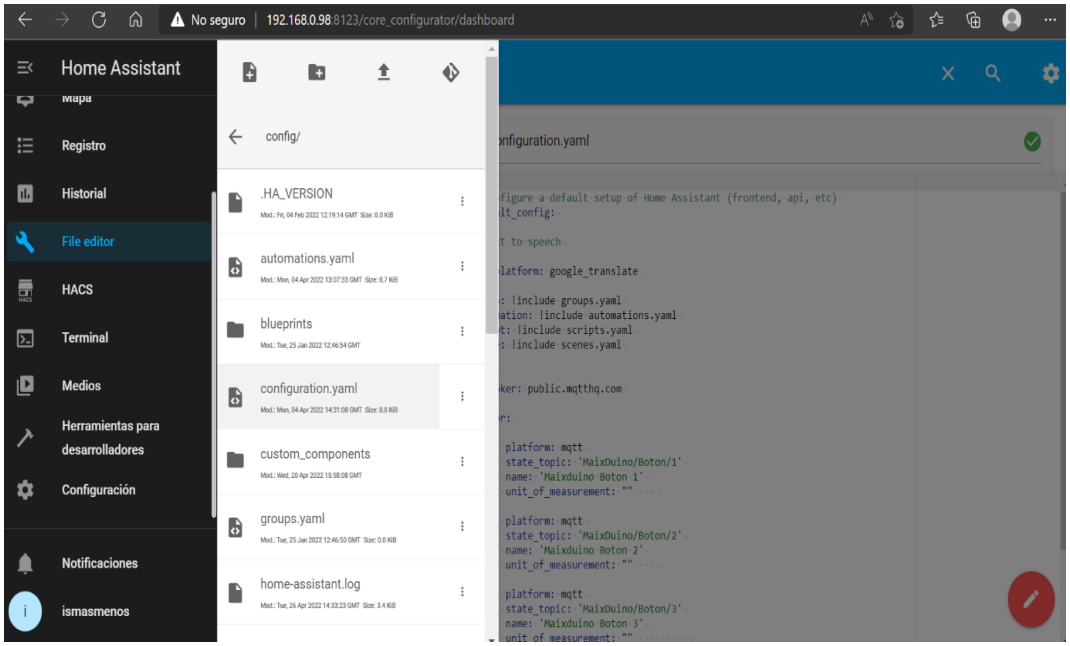

FIGURA 77. Acceso a la carpeta "configuration.yaml" desde File editor

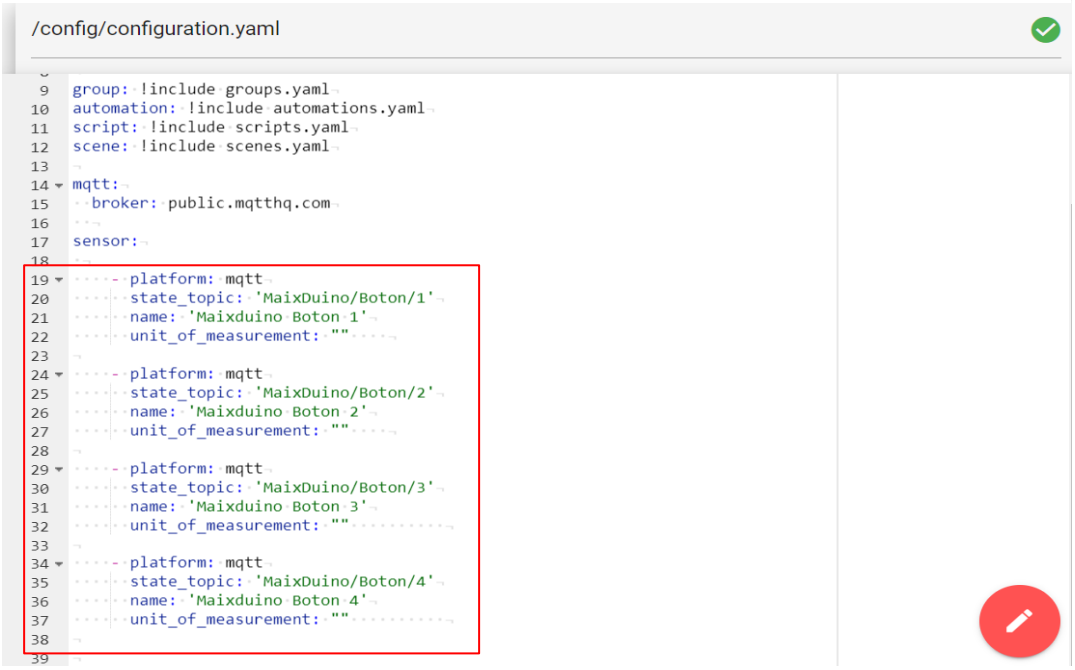

FIGURA 78. Configuración de los temas MQTT correspondientes a los botones digitales del Maixduino

Una vez asignados los tópicos y nombres a cada uno de los cuatro botones digitales, guardamos los cambios y reiniciamos el servidor desde el apartado de administración del servidor.

Por último, para visualizar el cambio de estado de los botones en la interfaz de Home Assistant, debemos editar el panel de control. Para ello, accedemos a **Resumen > 3 puntitos en el lateral derecho superior > editar panel de control > +** y añadimos una nueva configuración que denominaremos MQTT.

A continuación, en dicho panel pulsamos “AÑADIR TARJETA” y “POR ENTIDAD” como se contempla en la **FIGURA 79**, seleccionamos las cuatro entidades correspondientes a los cuatro botones digitales definidos con anterioridad en la carpeta “configuration.yaml”. Las tarjetas son una serie de pestañas que nos van a permitir tanto monitorizar datos, como controlar el estado de dispositivos.

FIGURA 79. Entidades correspondientes a los cuatro botones digitales en el panel MQTT de Home Assistant

Pulsamos “CONTINUAR” obteniendo el panel de la **FIGURA 80**, y “AÑADIR A LA IU LOVELACE”.

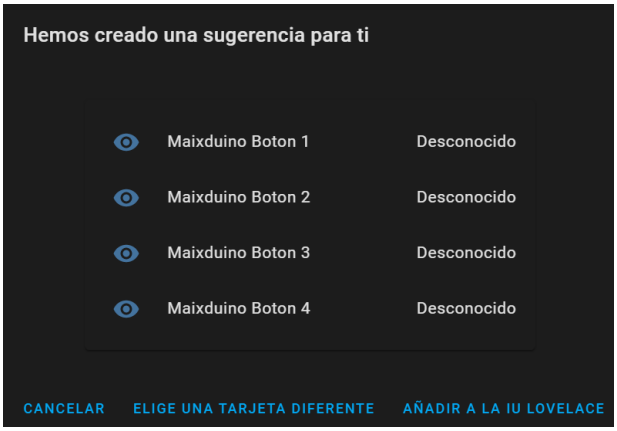

FIGURA 80. Tarjetas correspondientes a los cuatro botones digitales

Con los pasos realizados anteriormente obtenemos la interfaz en Home Assistant con lo que podremos monitorizar el estado de los cuatro botones. Con ello, podremos comprobar que el envío y recepción de los mensajes transcurre correctamente.

5.6.3 Desarrollo del protocolo MQTT en Maixpy IDE

El código desarrollado al completo se encuentra en los anexos debido a su extensión, lo que interrumpiría la fluidez de la lectura y comprensión de la memoria, por ende, en el presente apartado nos centraremos en la parte del código de mayor interés para la correcta configuración del protocolo.

La clase MQTT al completo se puede encontrar en los anexos de la memoria como se ha mencionado anteriormente. A continuación, se van a explicar aquellos detalles de mayor importancia de dicha clase, para su correcto ajuste a nuestras necesidades.

Definimos una serie de parámetros como son el Broker, el puerto MQTT, el usuario y contraseña y los tópicos, todos ellos obtenidos a la hora de configurar e instalar MQTT en Home Assistant en el apartado anterior. En cuanto al identificador del cliente se puede dar cualquier nombre, el objetivo de este parámetro es identificar el dispositivo que se ha conectado al protocolo (**TABLA 15**).

```
#Parámetros MQTT
SERVIDOR_MQTT     = "192.168.0.98"
PUERTO_MQTT       = 1883
USUARIO_MQTT   = "ISMQTT"
CONTRASE_MQTT   = "EMQTT"
IDENTIFICADOR_CLIENTE_MQTT = "MaixDuino"
MQTT_PUB_TOPIC_BOTON_1 = "MaixDuino/Boton/1"
MQTT_PUB_TOPIC_BOTON_2 = "MaixDuino/Boton/2"
MQTT_PUB_TOPIC_BOTON_3 = "MaixDuino/Boton/3"
MQTT_PUB_TOPIC_BOTON_4 = "MaixDuino/Boton/4"
```

TABLA 15. Parámetros para la comunicación MQTT

Para la conexión y desconexión al servidor se han definido las funciones indicadas en la **TABLA 16**. Dichas funciones usan la clase MQTT e indican si se ha conectado o desconectado el Maixduino al servidor. Para la conexión al servidor deseado se le envían los parámetros correspondientes del mismo, como es la IP del servidor, el puerto, el usuario y contraseña indicadas anteriormente a la variable "cliente", para posteriormente llamar a la función "`def Conexion_MQTT(cliente):`" encargada de realizar la conexión con el servidor.

```
# Conectarse al servidor MQTT
def Conexion_MQTT(cliente):
    global MQTT_Conectado
    if not MQTT_Conectado:
        try:
            cliente.connect()
            MQTT_Conectado = True
            print("MQTT conectado {}".format(SERVIDOR_MQTT))
```

```
        except Exception as e:
            MQTT_Conectado = False
            print("Fallo en la conexion MQTT: {}".format(e))
    else:
        print("MQTT Conectado")

#Desconectarse del servidor MQTT
def Desconexion_MQTT(cliente):
    global MQTT_Conectado
    if MQTT_Conectado:
        try:
            cliente.disconnect()
        except:
            pass
  MQTT_Conectado = False

# Paso de los parámateros para la conexión al servidor MQTT
def Conectar_MQTT():
    global cliente, MQTT_Conectado
    cliente = MQTTClient(cliente_id=IDENTIFICADOR_CLIENTE_MQTT,
server=SERVIDOR_MQTT, port=PUERTO_MQTT,
                         user=USUARIO_MQTT,
password=CONTRASE_MQTT)
    while not MQTT_Conectado and wifi.isconnected():
        print("--- Conectando al broker MQTT  {} ---
".format(SERVIDOR_MQTT))
        Conexion_MQTT(cliente)
        if not MQTT_Conectado:
      time.sleep(1)
```

TABLA 16.Código para la conexión y desconexión del servidor MQTT

Para la publicación de los mensajes (cambios de estado de cada botón) se ha definido la función indicada en la **TABLA 17**, en la cual se pasan los tópicos, es decir, los canales por los que se transmiten los mensajes, así como el estado del botón (“1” activo y ”0” desactivado).

```
#Publicar mensajes al servidor MQTT
def Publicar_MQTT():
    global cliente,
MQTT_Conectado,Boton_1,Boton_2,Boton_3,Boton_4
    if not wifi.isconnected():
        if MQTT_Conectado:
            Desconexion_MQTT(cliente)
            MQTT_Conectado = False
        Conexion_Wifi()
    Intentos = 3 #Intentos para conectarse al servidor
    while Intentos > 0:
        if not MQTT_Conectado:
            Conexion_MQTT(cliente)
        if MQTT_Conectado:
            try:
                #Una publicación por tópico
                cliente.publish(MQTT_PUB_TOPIC_BOTON_1,
                                json.dumps(Boton_1))
                cliente.publish(MQTT_PUB_TOPIC_BOTON_2,
                                json.dumps(Boton_2))
```

```
                cliente.publish(MQTT_PUB_TOPIC_BOTON_3,
                                json.dumps(Boton_3))
                cliente.publish(MQTT_PUB_TOPIC_BOTON_4,
                                json.dumps(Boton_4))
                print("Mensaje publicado")
                break
            except Exception as e:
                print("MQTT publicacion fallida")
                Desconexion_MQTT(cliente)
        else:
            print("MQTT no conectado, publicacion fallida")
        Intentos = Intentos - 1
        if Intentos > 0:
    print("Se intentara conectar {} mas".format(Intentos))
```

TABLA 17. Código para la publicación de mensajes mediante el protocolo MQTT

5.7 Control del enchufe Tapo P100 desde Home Assistant

Para poder automatizar el enchufe inteligente seleccionado en el presente TFG, el Tapo P100, y activarlo o desactivarlo en función de la activación o desactivación de los botones digitales del módulo Maixduino, primero debemos integrar dicho enchufe en la herramienta domótica Home Assistant. Para ello, accedemos a Home Assistant y en **Configuraciones > Dispositivos y servicios > AÑADIR INTEGRACIÓN**; se abriría un panel para la búsqueda de integraciones donde buscaremos el controlador “Tapo”. Sin embargo, la integración del Tapo P100 no se encuentra de manera oficial entre las integraciones de Home Assistant como se puede observar en la **FIGURA 81**.

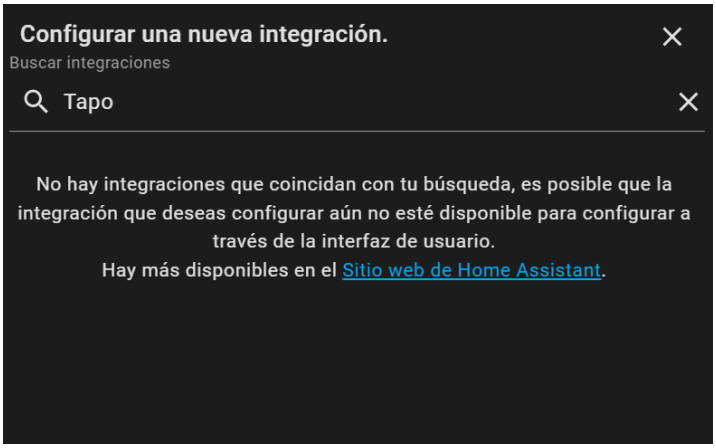

FIGURA 81. Búsqueda de la integración "Tapo" en el panel de búsqueda de integraciones de Home Assistant

Para solucionar este problema, debemos instalar en Home Assistant la herramienta HACS (Home Assistant Community Store). HACS es una tienda de

dispositivos personalizados, plantillas y tarjetas desarrollada que permite que cualquier desarrollador pueda añadir sus componentes, por lo que esta tienda nos ofrece la integración e instalación de forma sencilla de un mayor número de componentes de los que no disponen la tienda de complementos oficial de Home Assistant [36].

Debe tenerse en cuenta que todas aquellas instalaciones de componentes a través de HACS están en fase de desarrollo y pueden no funcionar correctamente.

5.7.1 HACS en Home Assistant

En el siguiente apartado se indicará el modo de operar para la instalación de HACS de manera correcta. Para la instalación debemos cumplir los siguientes prerrequisitos:

- La versión de Home Assistant debe ser la 2022.3.0 o posterior.
- Conexión a internet estable.
- Cuenta en GitHub.

Para continuar, debemos conocer el tipo de instalación de Home Assistant de la que disponemos, en nuestro caso, sabemos que es la versión 2022.2.1 y el modo de instalación es "SISTEMA OPERATIVO (OS)", para realizar la comprobación accedemos a **Configuración > Configuración > Información** (**FIGURA 82**).

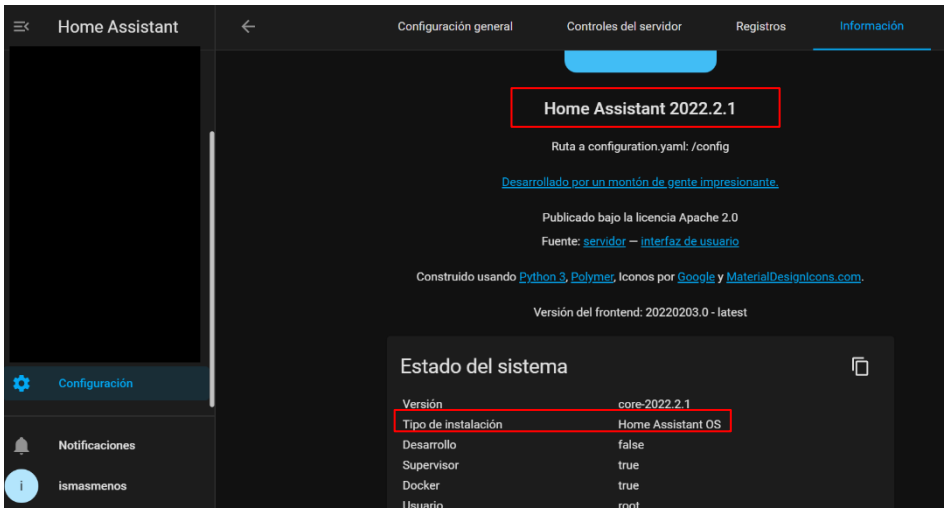

FIGURA 82. Versión y tipo de instalación de Home Assistant

El siguiente paso será habilitar el modo avanzado desde el perfil de usuario como se muestra en la **FIGURA 83**.

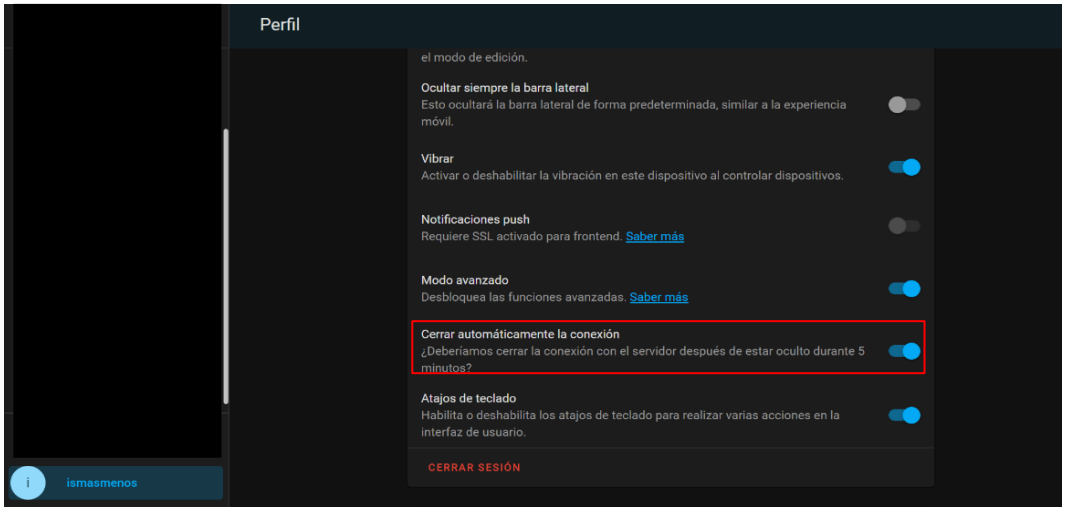

FIGURA 83. Habilitación del modo avanzado en Home Assistant

5.7.1.1 Instalación de la herramienta Terminal & SSH

La instalación de HACS se hace a través del terminal SSH, por tanto, antes de instalar HACS debemos instalar dicha herramienta, por ende, accedemos a la tienda de complementos y buscamos "Terminal & SSH" (**FIGURA 84**) y procedemos a su instalación.

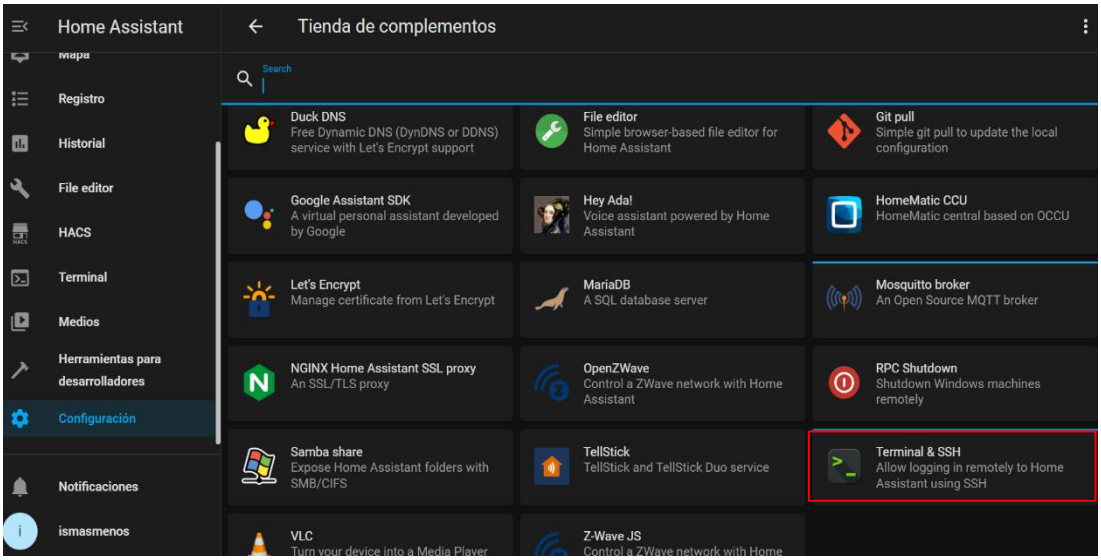

FIGURA 84. Búsqueda de Terminal & SSH en la tienda de complementos de Home Assistant

Una vez instalado, activamos la opción de vigilancia que permitirá reiniciar el complemento en caso de caída y mostramos este complemento en la barra lateral de nuestro Home Assistant, para un acceso directo (**FIGURA 85**).

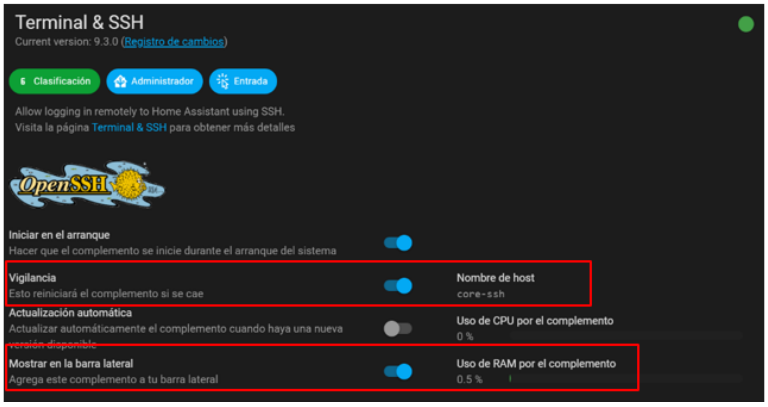

FIGURA 85. Activación de la vigilancia y muestra en la barra lateral del Terminal & SSH en Home Assistant

A continuación, debemos realizar una configuración del Terminal & SSH. Accedemos desde la barra superior a "Configuración".

La configuración consiste en poner una contraseña o una llave autorizada al terminal y habilitar el puerto SSH. Para una mayor seguridad, se ha optado por una llave autorizada y se ha habilitado como puerto el puerto 22, ya que es el que se usa como norma general, aunque, se puede usar cualquier otro (**FIGURA 86**).

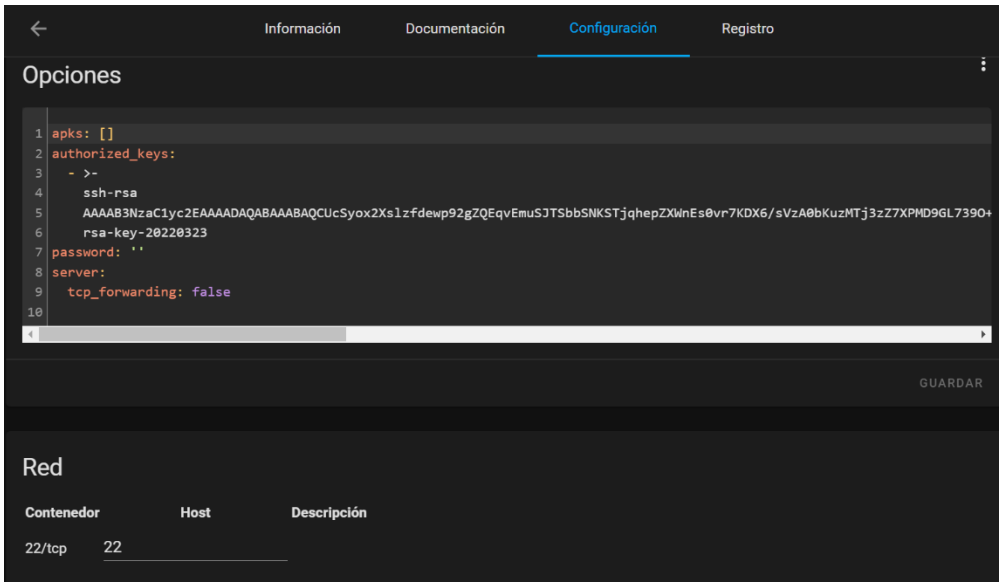

FIGURA 86. Configuración del Terminal & SSH en Home Assistant

5.7.1.1.1 Generación de clave autorizada mediante MobaXterm

Para aumentar la seguridad a la hora de la conexión remota mediante SSH a nuestro Home Assistant, generaremos una clave autorizada a través del programa MobaXterm.

En el programa accedemos a **Tools > MobaKeyGen**. Una pestaña nos indica que no disponemos de ninguna llave (**FIGURA 87**). Para generar esta llave pulsamos "Generate" (**FIGURA 88**).

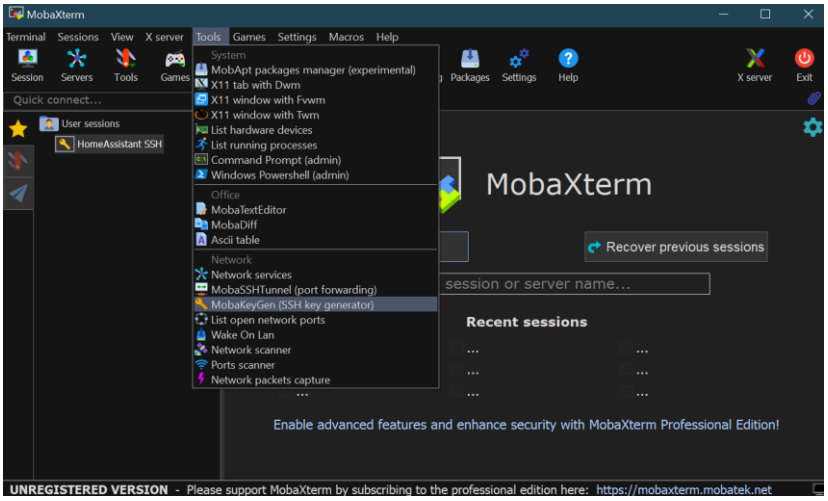

FIGURA 87. Acceso a la generación de calve autorizada en MobaXterm

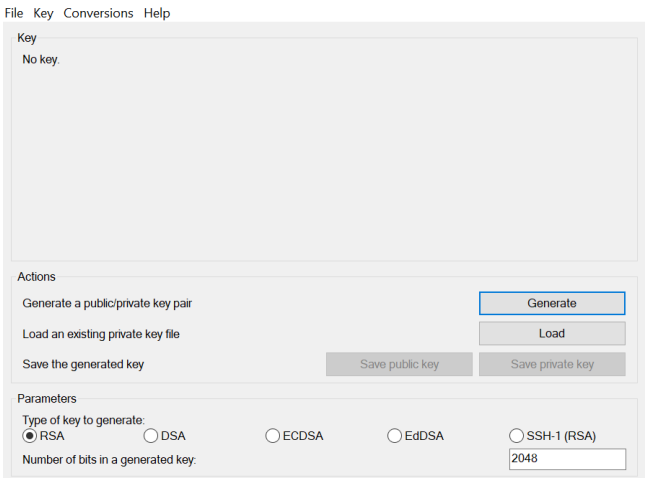

FIGURA 88. Clave de autorización generada en MobaXterm

Al finalizar el proceso de carga se generará la clave (**FIGURA 89**), que a continuación deberemos copiar y pegar en la configuración del Terminal & SSH de Home Assistant como se mostró en la **FIGURA 86** y guardamos las modificaciones.

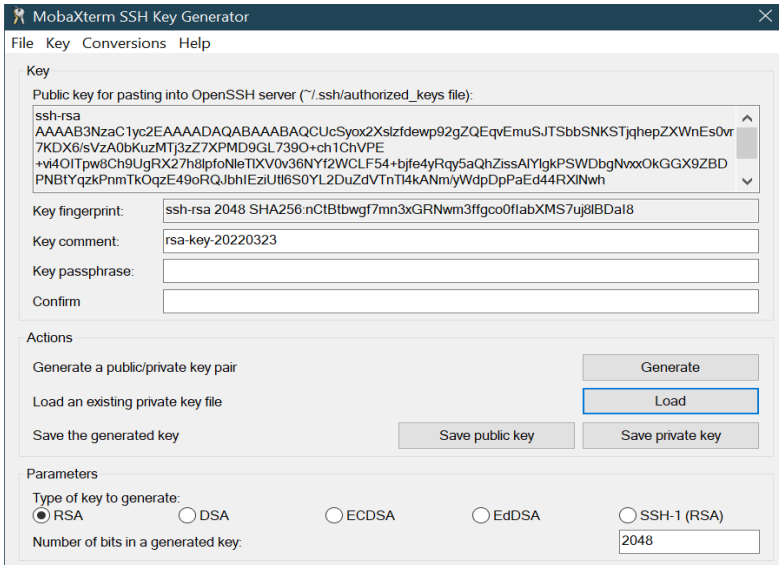

FIGURA 89. Clave de autorización generada en MobaXterm

Con este tipo de configuración disminuye la vulnerabilidad del sistema, ya que la clave autorizada proporciona una mayor seguridad y es más difícil de hackear que una simple contraseña, evitando así, que alguien pueda acceder a nuestro Home Assistant.

Además, la herramienta MobaXterm nos permitirá acceder de forma remota a nuestro Home Assistant, pudiendo así realizar configuraciones en Home Assistant sin la necesidad de estar conectados a la misma red Wifi a la que está conectada la Rasberry Pi.

Ahora ya podemos acceder al Terminal & SSH.

5.7.2 Instalación de HACS desde el terminal

Ahora que ya disponemos del terminal en funcionamiento, podemos realizar la instalación de HACS.

Desde la página de HACS [36] copiamos el comando correspondiente a nuestro tipo de instalación de Home Assistant como se observa en la **FIGURA 90**.

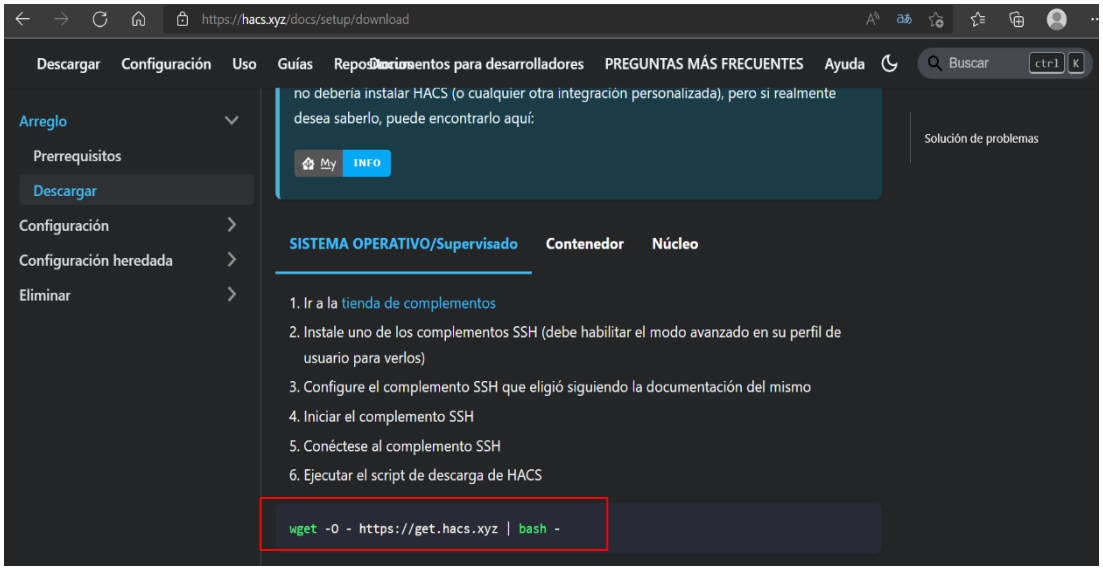

FIGURA 90. Comando para la instalación de HACS en Home Assistant

El comando copiado anteriormente procedemos a pegarlo en el terminal y pulsamos "intro", obteniendo la instalación de HACS como se observa en la **FIGURA 91**.

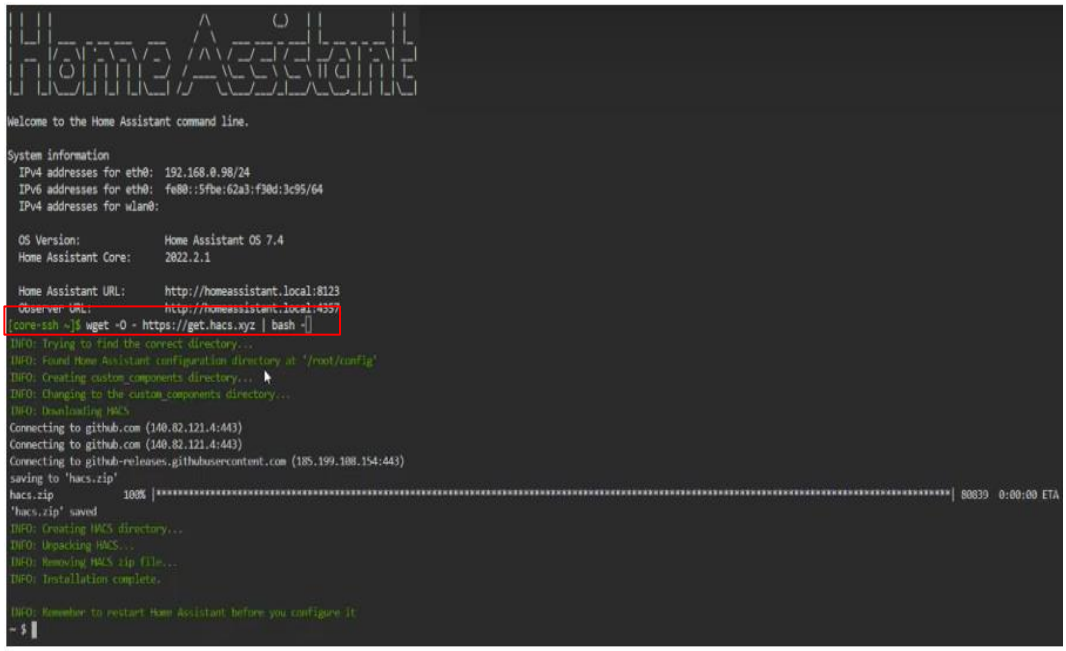

FIGURA 91. Instalación de HACS a través del terminal

El siguiente paso es reiniciar Home Assistant, accediendo a **Configuración > Configuración > Controles del servidor** y pulsando "REINICIAR" (**FIGURA 92**).

Una vez el servidor se haya puesto en marcha accedemos a **Configuración > Dispositivos y servicios, pulsamos en "AÑADIR INTEGRACIONES"** y buscamos HACS (**FIGURA 93**).

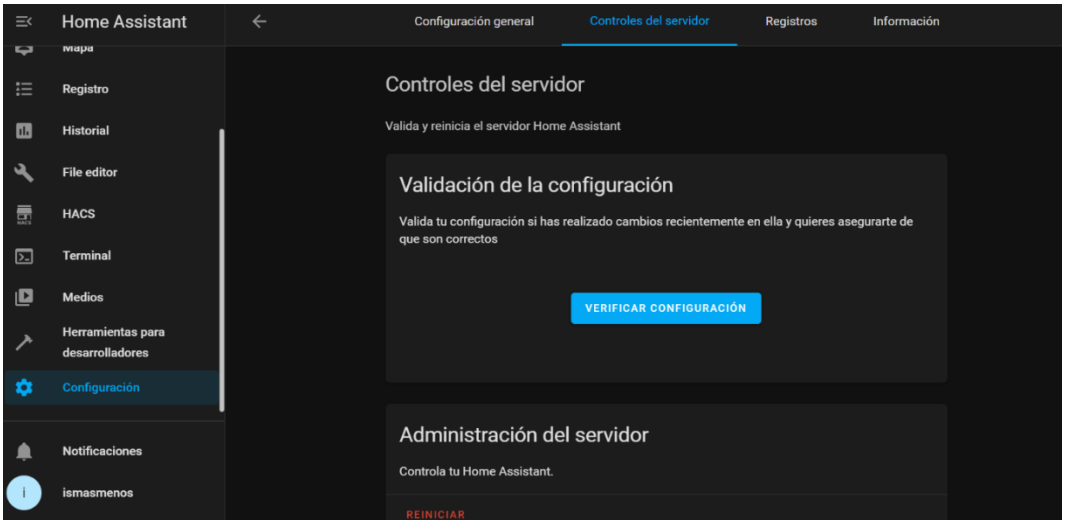

FIGURA 92. Reinicio del servidor de Home Assistant

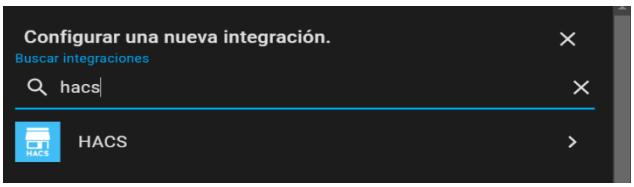

FIGURA 93. Búsqueda de HACS en el buscador de integraciones de Home Assistant

Como se puede observar, ya disponemos de la herramienta HACS para su instalación.

Una vez realizada la instalación obtendremos una pestaña en la que tendremos que aceptar todas las condiciones y que nos mandará a GitHub para introducir el código que nos ofrecen para la verificación de la cuenta creada, como se puede contemplar en la **FIGURA 94**.

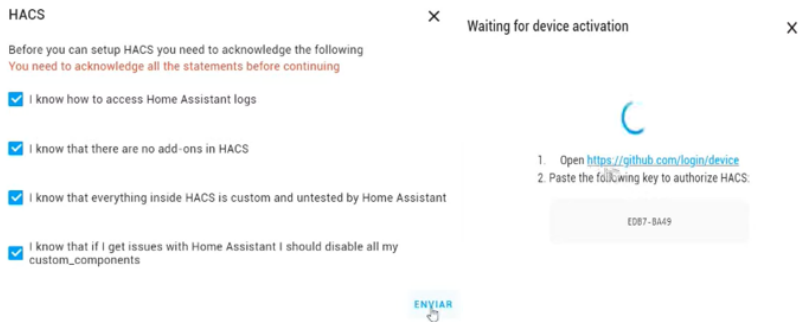

FIGURA 94. Condiciones de HACS y verificación en GitHub

Una vez finalizado el proceso anterior, ya disponemos de HACS en nuestro Home Assistant, y todos los repositorios que contiene.

5.7.3 Descarga de Tapo Controller desde HACS

Ahora procedemos a descargar el repositorio de Tapo controller, para poder así integrar dispositivos de la marca Tapo. Para ello, accedemos a **HACS > Integraciones > EXPLORAR Y DESCARGAR REPOSITORIOS** en la búsqueda de repositorios buscamos "tapo".

Como se puede observar en la **FIGURA 95** disponemos de dos tipos de repositorios Tapo, usaremos el repositorio "Tapo Controller", que permite incorporar integraciones para controlar dispositivos tapo desde Home Assistant.

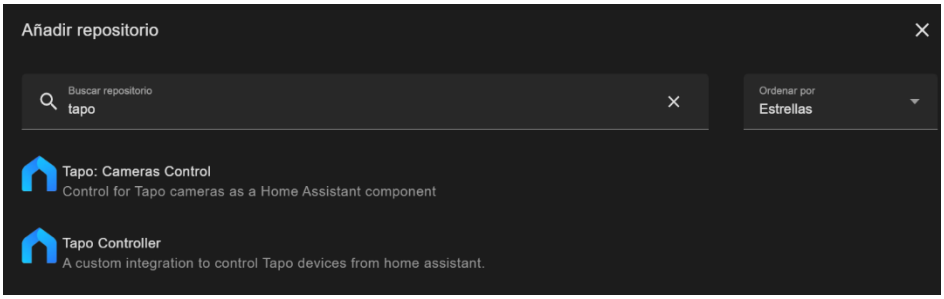

FIGURA 95. Búsqueda de "Tapo" en el repositorio de HACS

Accedemos a "Tapo Controller" y procedemos a su descarga pulsando en "DESCARGAR ESTE REPOSITORIO CON HACS" como se indica en la **FIGURA 96**.

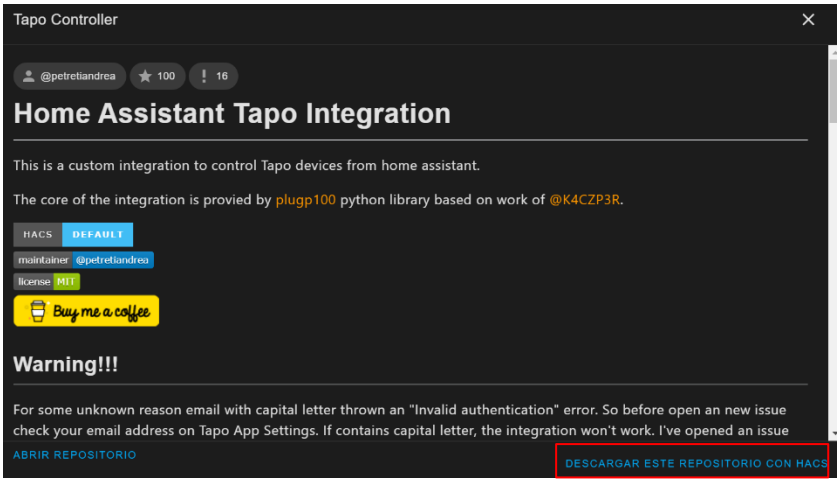

FIGURA 96. Descarga del repositorio "Tapo Controller" desde HACS

Una vez realizada la descarga ya dispondremos de la integración de Tapo, que nos va a permitir integrar una serie de dispositivos de la familia Tapo, por lo que el siguiente paso será configurar e integrar nuestro enchufe P100 como se indica en el siguiente apartado.

5.7.4 Integración del enchufe Tapo P100 en Home Assistant

Vamos a integrar el enchufe P100 para su control desde Home Assistant, para ello vamos a **Configuración > Dispositivos y servicios > AÑADIR INTEGRACIÓN** Y buscamos "Tapo Controller", al haber descargado su repositorio podremos encontrarlo en el buscador. Accedemos a dicha integración y se abrirá una pestaña como la que se muestra en la **FIGURA 97**.

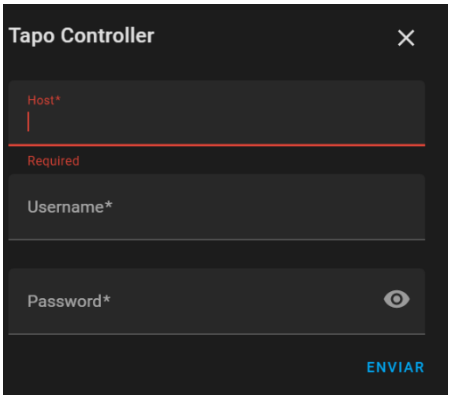

FIGURA 97. Pestaña para de integración de dispositivos Tapo

Para obtener los parámetros que se nos indican para poder integrar cualquier dispositivo Tapo compatible, debemos acceder a la app de Tapo que encontramos en el "Play Store" del teléfono móvil. La primera vez que accedamos a esta aplicación nos pedirá un registro mediante un correo electrónico y una contraseña, estos datos serán el "Username" y "Password" respectivamente.

5.7.4.1 Configuración del enchufe Tapo p100 en la app Tapo.

Una vez dentro de la app del móvil, procedemos a añadir un dispositivo, en este caso, nuestro dispositivo Tapo P100.

Ya seleccionado el dispositivo a añadir, conectamos a la alimentación el enchufe.

El enchufe dispone de un pequeño led y un botón en el lateral. Para entrar en el modo de localización del dispositivo debemos mantener pulsado dicho botón durante 5 segundos, hasta que observemos el comienzo del parpadeo en naranja y verde. A continuación, se indica a la aplicación que el enchufe está en el modo de parpadeado. Inmediatamente se pedirá habilitar el Bluetooth y la localización de nuestro dispositivo móvil.

El Bluetooth solo se usará la primera vez para configurar y localizar el enchufe, posteriormente no será necesario su uso.

Una vez realizados los pasos descritos anteriormente, obtendremos una notificación de que la conexión se ha realizado de forma correcta. Tras esto, solo resta conectarlo a la red Wifi y asignarle un nombre a nuestro dispositivo Tapo.

Todo este proceso se muestra de forma más visual en la **FIGURA 98**.

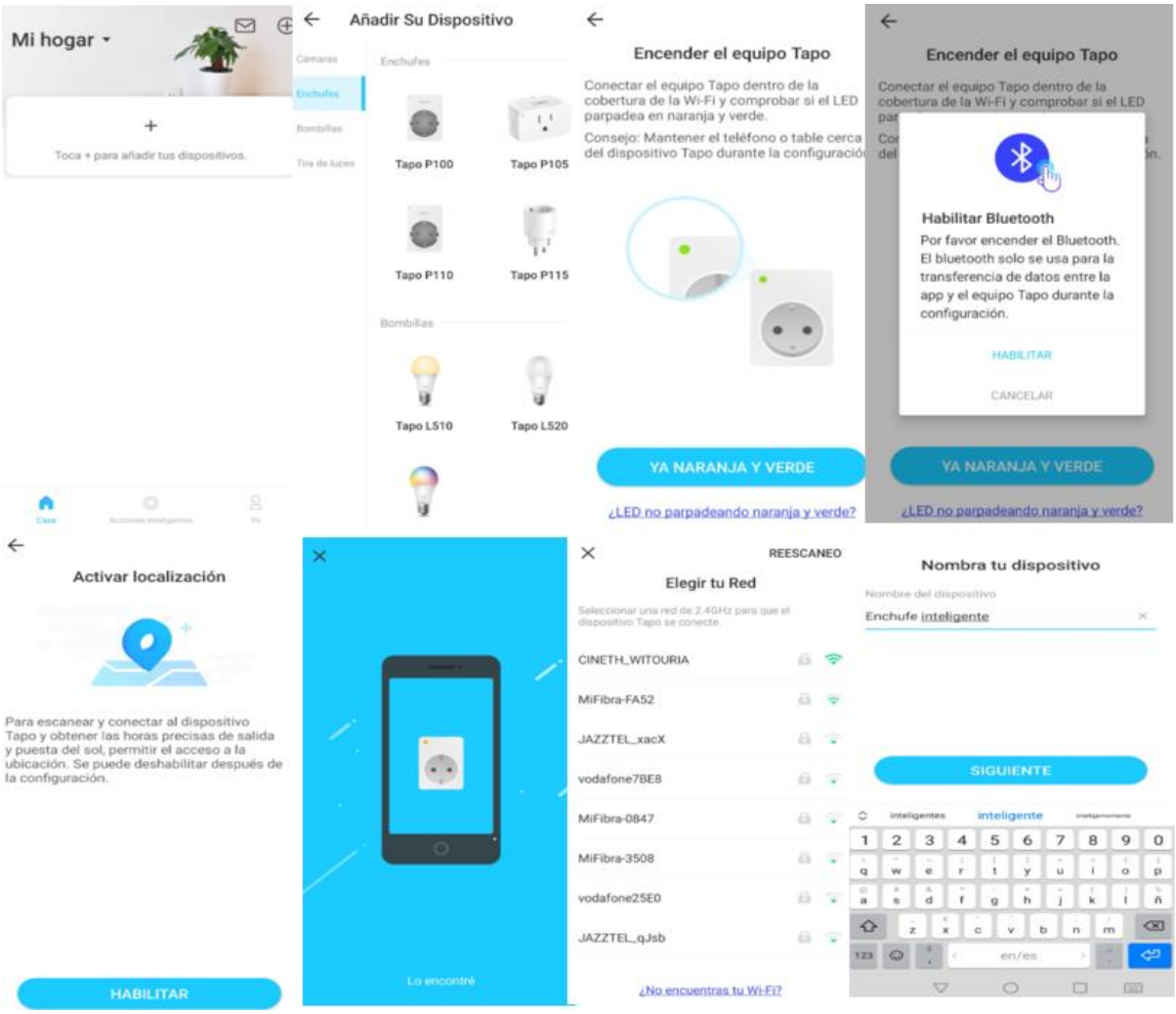

FIGURA 98. Configuración del enchufe Tapo P100 en la app Tapo

Para comprobar que el dispositivo se ha conectado correctamente, probamos a encender y apagar el enchufe desde la propia aplicación Tapo, como se observa sutilmente en la **FIGURA 99**, cuando el enchufe está apagado, el led está apagado y cuando el enchufe es encendido, el led es de color verde.

FIGURA 99. Control del enchufe Tapo P100 desde la app Tapo

Una vez comprobado que el enchufe funciona correctamente, el último dato necesario para su integración en Home Assistant es el "HOST", es decir, la IP del enchufe inteligente. La obtención de esta IP se hace mediante la propia aplicación Tapo, y para su obtención, accedemos a la **ruleta de ajustes del enchufe inteligente > Información del dispositivo** (FIGURA 100).

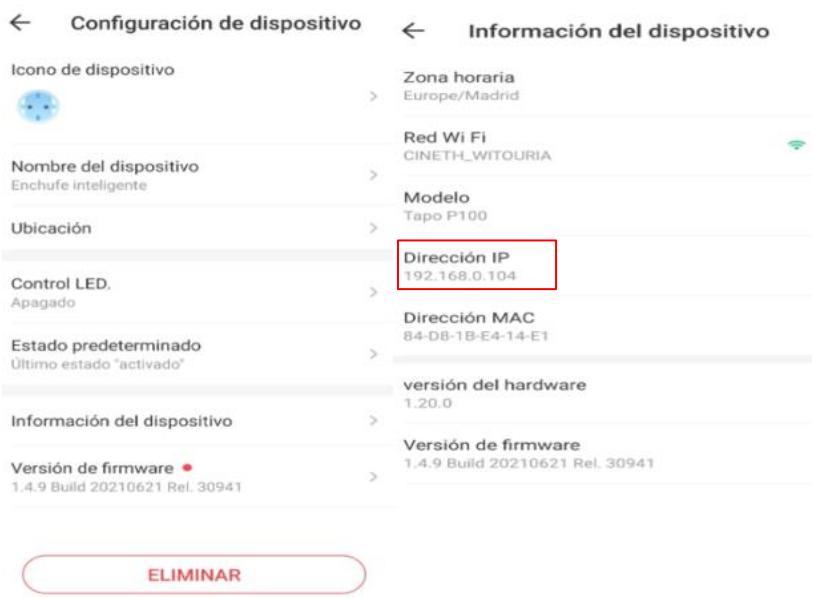

FIGURA 100. Obtención de la IP del enchufe Tapo P100

Ya conocidos todos los datos para su integración en Home Assistant procedemos a completar estos datos en la pestaña de la **Figura 100** y enviamos. Se muestra en la **FIGURA 101** la pestaña indicativa de que la integración se ha realizado correctamente.

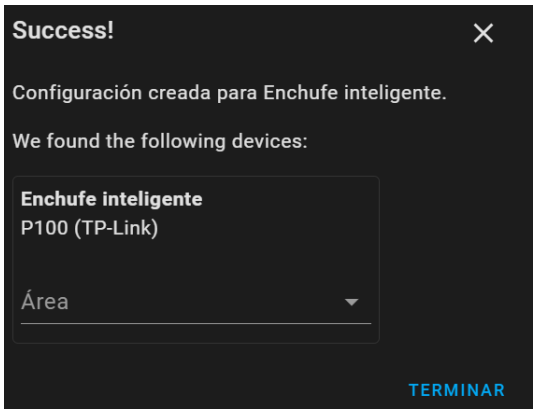

FIGURA 101. Integración exitosa del enchufe Tapo P100 en Home Assistant

Ahora ya podemos controlar el enchufe inteligente a partir de Home Assistant al igual que lo hacemos desde la aplicación móvil. Esto nos va a permitir incluir el enchufe en automatizaciones y, por tanto, que el enchufe se encienda y apague en función de la activación o desactivación de los botones digitales programados en la placa de desarrollo Maixduino.

57.4.2 Tarjeta de la entidad enchufe

De igual forma que añadimos una serie de tarjetas para monitorizar el estado de los botones digitales del Maixduino, podemos hacerlo para el enchufe, lo que nos va a permitir conocer el estado y control de este. Para la obtención de dicha tarjeta seguimos el mismo *modus operandi* que para la configuración los cuatro botones.

En este caso, buscaremos en la pestaña entidades: "Enchufe Inteligente", seleccionamos y pulsamos "CONTINUAR"; ahora tendremos en la interfaz MQTT información del estado de los botones y control e información del estado del enchufe como se puede observar en la **FIGURA 102**.

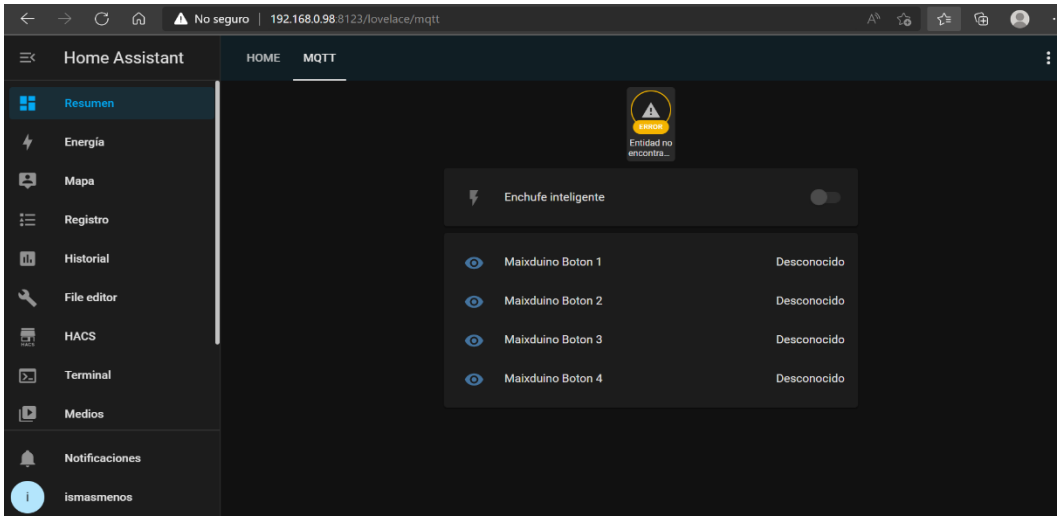

FIGURA 102. Tarjetas de entidad de los botones digitales y el enchufe inteligente en la interfaz MQTT

5.8 Automatización del enchufe Tapo P100

Conseguida la transmisión de información del Maixduino a la Raspberry Pi, mediante la codificación a binario del estado de los botones: "0" correspondiente a la desactivación del botón digital y "1" correspondiente a la activación.

Ahora podemos realizar una automatización que consista en activar el enchufe cuando se active el botón digital del Maixduino y desactivar el enchufe cuando se desactive el botón digital.

Para realizar la automatización en Home Assistant accedemos a **Configuración > Automatizaciones y escenas > AÑADIR AUTOMATIZACIÓN > Empezar con una automatización vacía**.

Hay que realizar dos automatizaciones, una correspondiente al encendido del enchufe y otra al apagado del enchufe, como se indica en los apartados siguientes.

5.8.1 Automatización para el encendido del enchufe

En primer lugar, proporcionamos un nombre a la automatización para su identificación, como se puede observar en la **FIGURA 103**.

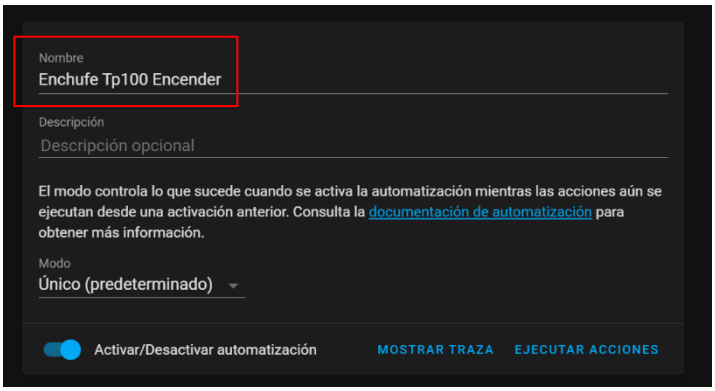

FIGURA 103. Nombre de la automatización para el encendido del enchufe Tapo P100

A continuación, configuramos el desencadenante que inicia una regla de automatización.

El botón digital cuando se activa envía un “1” a Home Assistant. Por tanto, queremos que cuando se reciba un “1” se encienda el enchufe o es lo mismo que decir, cuando se reciba algo por encima de 0. Para realizar esta configuración, elegimos como tipo de desencadenante el “Estado numérico”, elegimos la entidad que recibe este cambio de estado, en este caso elegimos el botón 1 y que se desencadene la automatización cuando el número recibido este por encima de 0 como se observa en la **FIGURA 104**.

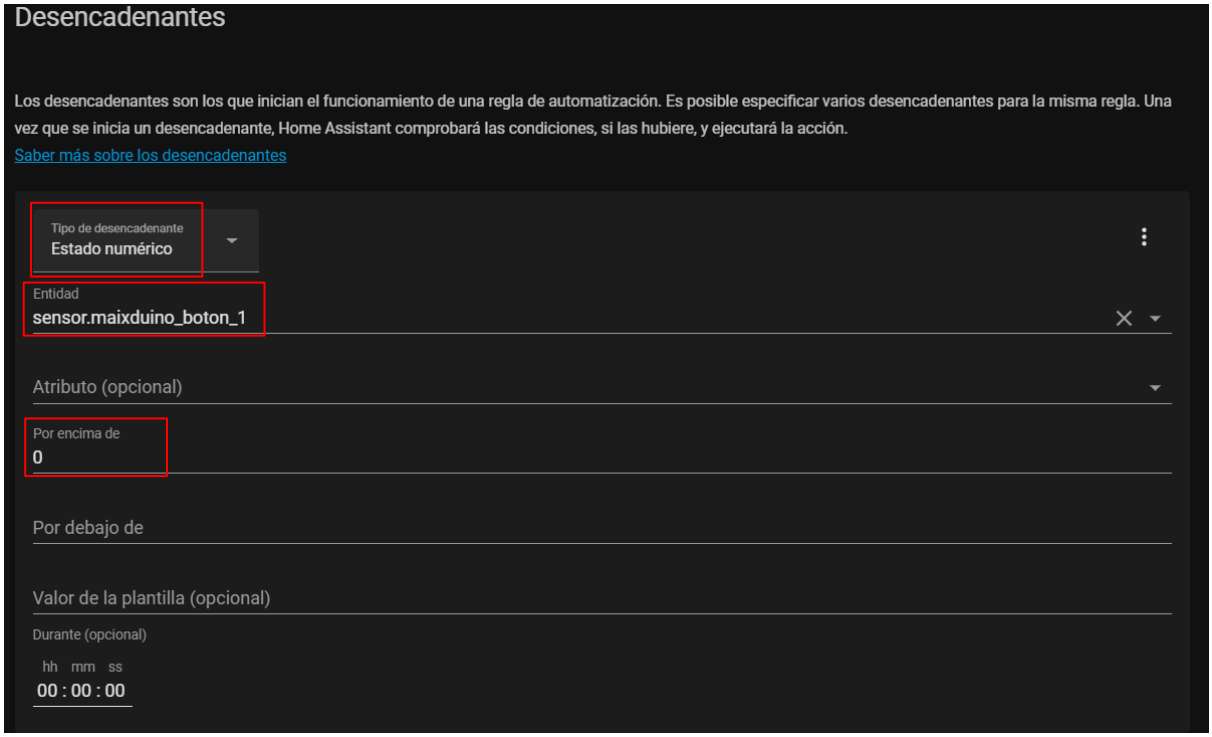

FIGURA 104. Configuración del desencadenante que provoca el encendido del enchufe

Por último, configuramos la acción dada cumplido el desencadenante, en este caso, la acción es el encendido del enchufe. Para ello, elegimos el tipo de acción, el nombre del dispositivo que hemos integrado anteriormente (Tapo P100) y la acción que se tiene que dar, que es la de encendido (**FIGURA 105**) y guardar la automatización.

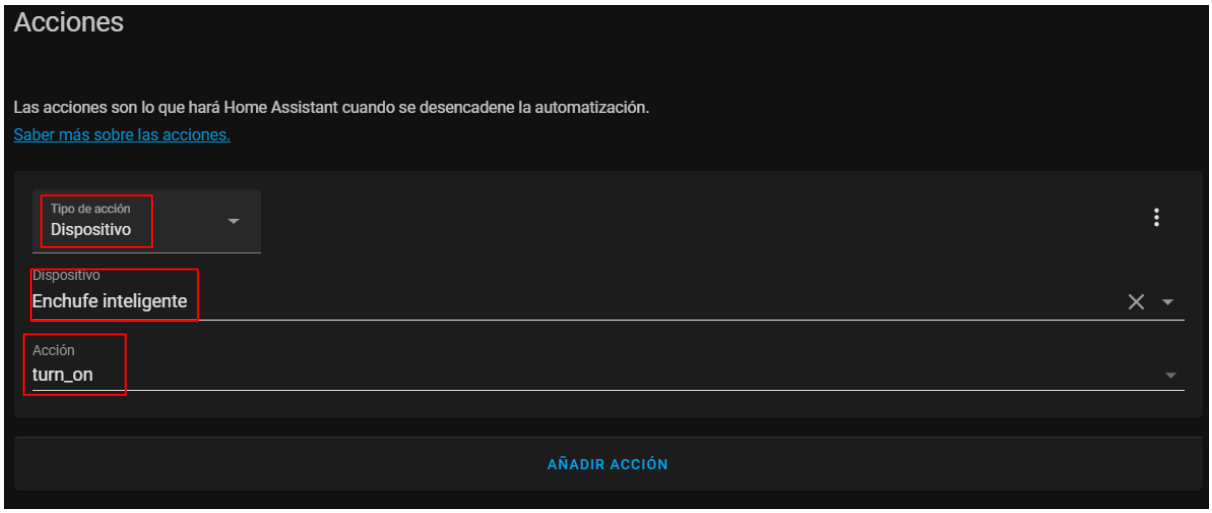

FIGURA 105. Configuración de la acción que provoca el encendido del enchufe

5.8.2 Automatización para el apagado del enchufe

El modo de operar es igual que el usado para la automatización de encendido del

enchufe. Proporcionamos un nombre a la automatización para su identificación (**FIGURA 106**).

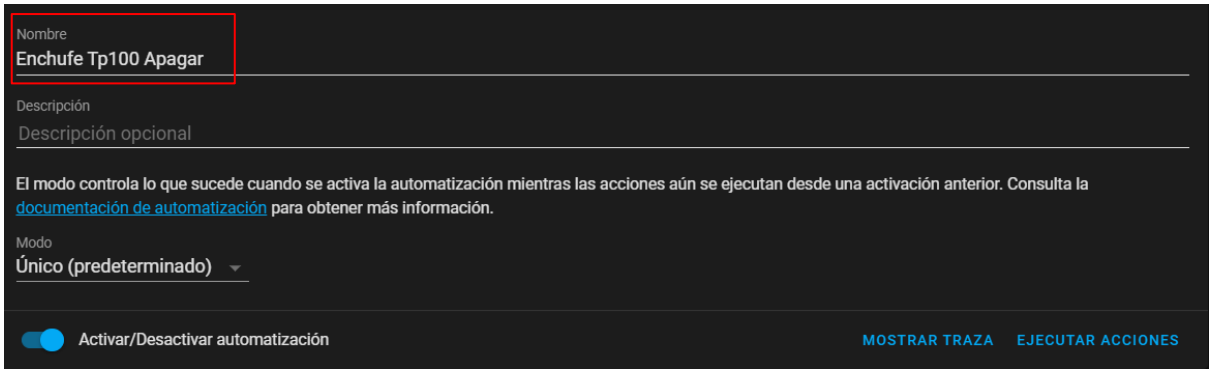

FIGURA 106. Nombre de la automatización para el apagado del enchufe Tapo P100

De igual forma que el apartado anterior, configuramos el desencadenante. El botón digital cuando se desactiva manda un "0" a Home Assistant, por tanto, queremos que cuando se reciba un "0" se apague el enchufe o es lo mismo que decir, cuando se reciba algo por debajo de 1. Elegimos como tipo de desencadenante el "Estado numérico", elegimos la entidad que recibe este cambio de estado, que es el botón 1 y que se desencadene la automatización cuando el número recibido este por debajo de 1 como se observa en la **FIGURA 107**.

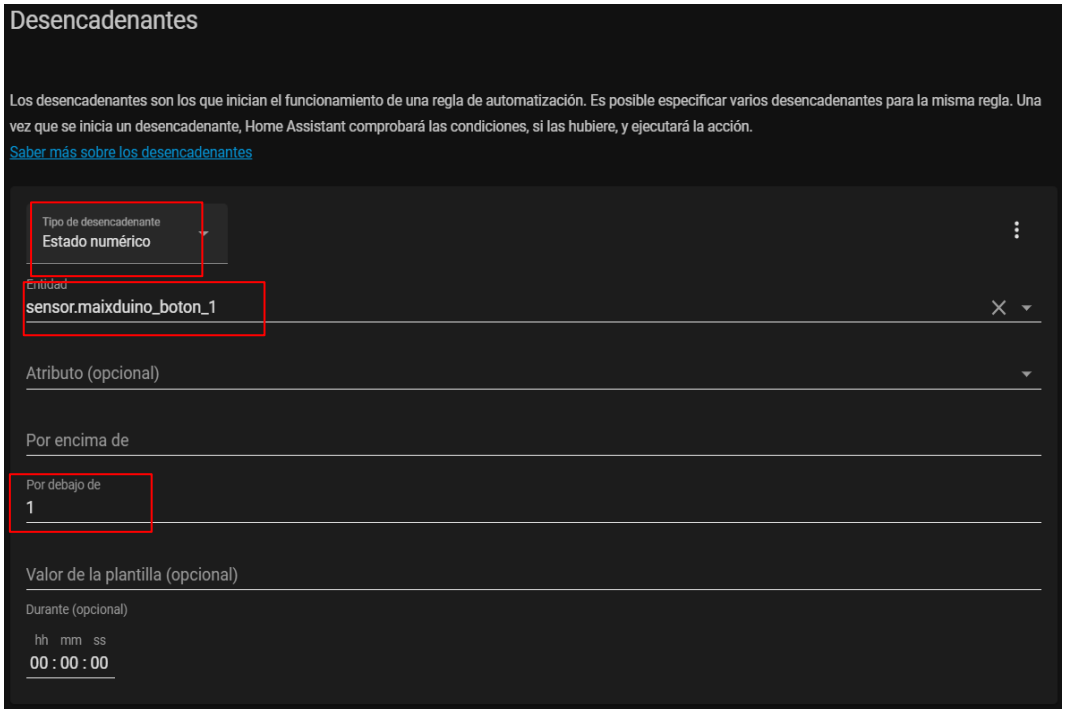

FIGURA 107. Configuración del desencadenante que provoca el apagado del enchufe

Por último, configuramos la acción dada cumplido el desencadenante, en este caso, la acción es el apagado del enchufe; para ello, elegimos el tipo de acción, el nombre del dispositivo que hemos integrado anteriormente y la acción que se tiene que dar, que es la de apagado (**FIGURA 108**) y guardamos la automatización.

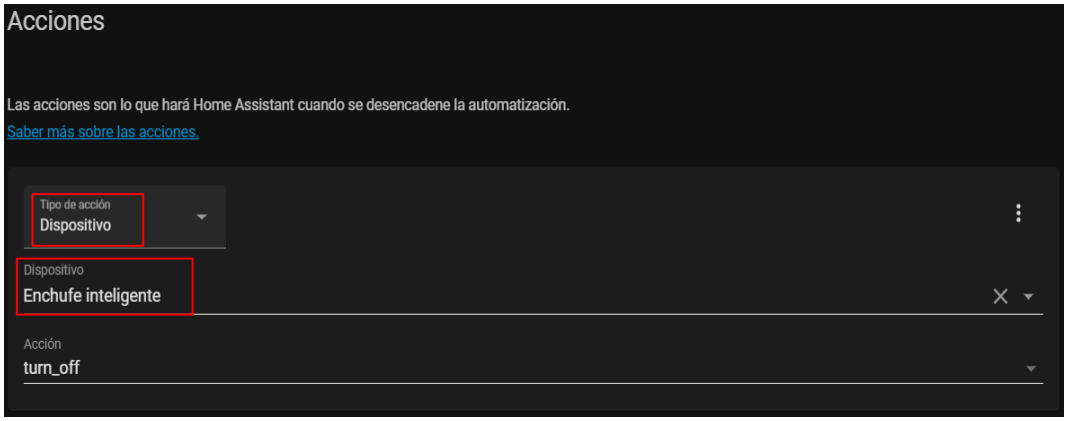

FIGURA 108. Configuración de la acción que provoca el apagado del enchufe

6 RESULTADOS Y EXPERIMENTOS DEL SISTEMA DESARROLLADO

6.1 Introducción

En el presente capitulo, mostraremos los distintos resultados obtenidos del desarrollo del sistema, así como los experimentos realizados para demostrar el cumplimiento de los objetivos propuestos y el correcto funcionamiento del proyecto.

6.2 Resultados

Para comprobar que el sistema desarrollado cumple con los objetivos inicialmente propuestos procedemos a probar su funcionamiento en conjunto.

Ejecutamos el programa desde MaixPy y procedemos a activar y desactivar los botones digitales como se muestra en la **FIGURA 109**.

FIGURA 109.Botones digitales del Maixduino activados y desactivados

Se puede observar, que la interfaz de usuario funciona correctamente y la disposición de los botones en la pantalla es la adecuada, ya que permite al usuario llegar a los botones sin la necesidad de hacer grandes movimientos de cuello.

La parte central de la pantalla queda liberada para evitar que el usuario active el botón por error.

Cabe destacar que el número de botones y las funciones de los mismos es totalmente personalizable en función de las necesidades y gustos de cada usuario.

En la **FIGURA 110**, a través del terminal serie del que dispone el programa MaixPy, podemos observar el proceso de ejecución del código que consiste en: Conexión a la red Wifi, seguidamente al servidor de Home Assistant y finalmente ejecuta el modelo de detección de rostros y la interfaz con los botones digitales.

Se puede contemplar de igual forma, como se detecta la activación (ON) y desactivación (OFF) de los distintos botones digitales y los mensajes que indican que el estado de estos botones se ha publicado en el servidor de Home Assistant.

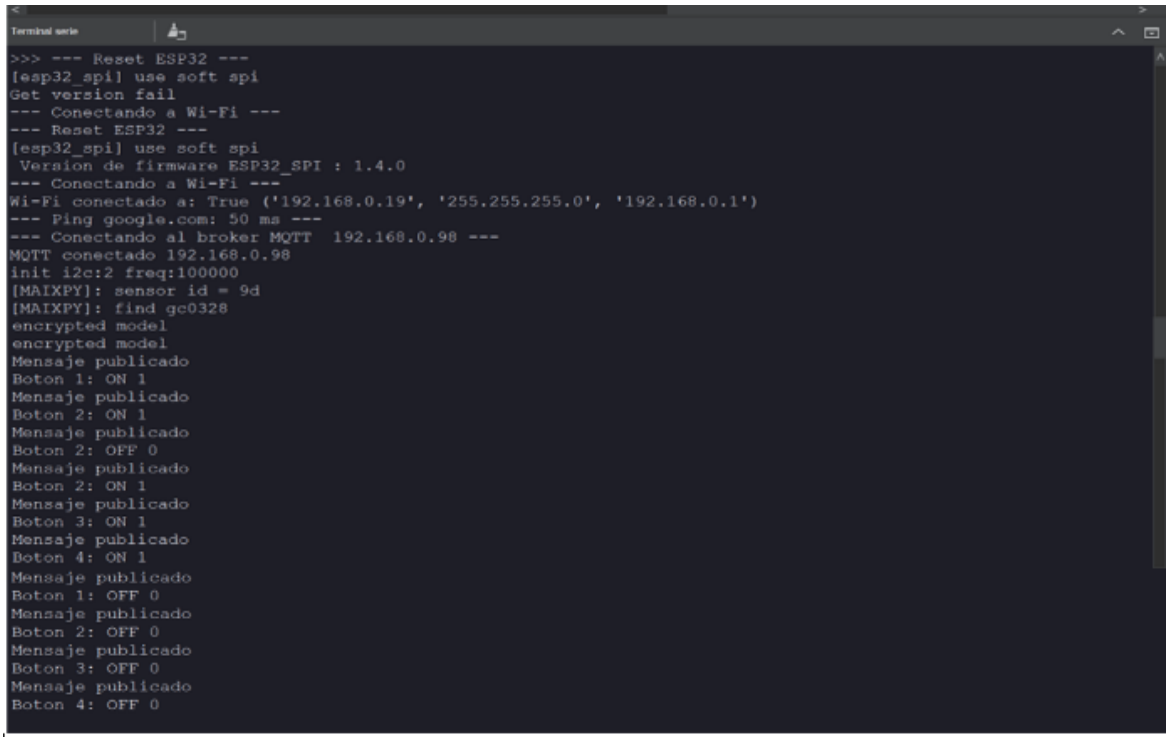

FIGURA 110. Muestra de la ejecución del programa mediante el terminal serie de Maixpy IDE

El terminal serie nos proporciona también información de la IP del servidor al que con el que se realiza la comunicación (misma IP que la asignada a nuestro servidor Home Assistant) y la IP de la red Wifi a la que nos conectamos.

Como se muestra en la **FIGURA 111**, el servidor recibe el estado de los botones de forma correcta, recibe un “1” indicativo de la activación de los botones y un “0” indicativo de la desactivación de los botones.

A partir de esta codificación podemos realizar una infinidad de automatizaciones desde Home Assistant, en función de las necesidades de cada usuario como, por ejemplo, la automatización del enchufe tapo P100 del apartado **5.8**.

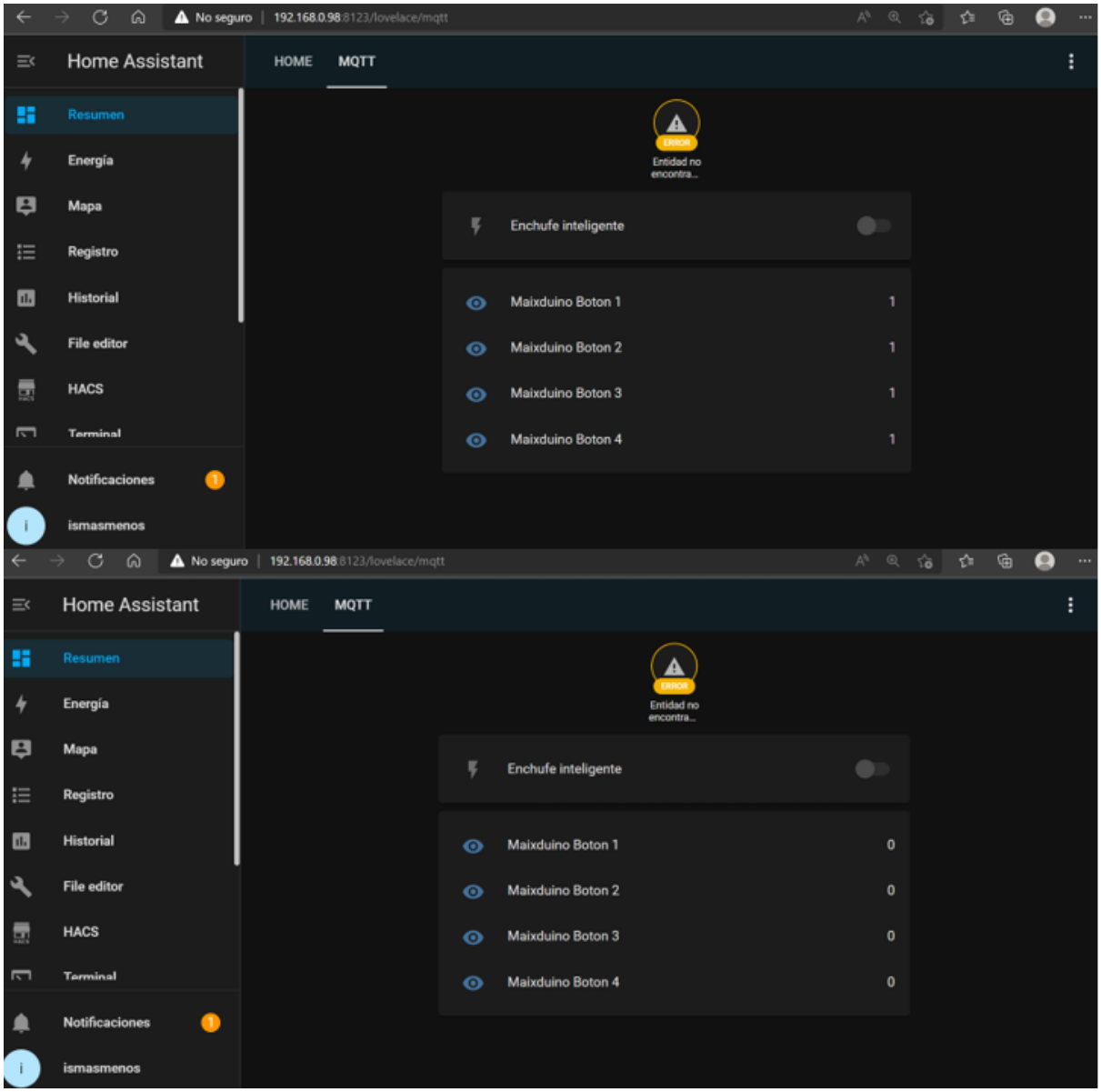

FIGURA 111. Estado de los botones digitales en la interfaz de Home Assistant

La importancia de usar como entorno domótico el software Home Assistant tiene lugar en que se pueden integrar y por tanto automatizar la gran parte de los dispositivos inteligentes del mercado, aumentando así el abanico de posibilidades de implementación de nuestro sistema al mundo real y las necesidades de cada persona.

Podemos concluir que, el sistema cumple con el objetivo de controlar actuadores mediante el posicionamiento de la nariz en los distintos botones digitales de la que

dispone la pantalla de nuestro Maixduino, permitiendo que aquellas personas con discapacidad motora que reduce su movilidad a tan solo el movimiento del cuello puedan controlar una serie de dispositivos de forma autónoma.

6.3 Experimentos

Para comprobar el funcionamiento de nuestro sistema, su comprensión por los usuarios, su manejabilidad y fiabilidad se ha realizado una serie de pruebas con distintos conocidos, de distintas edades, géneros y estudios, por ende, la finalidad de esta serie de pruebas es examinar la ergonomía del producto desarrollado.

La prueba ha consistido en que, una vez explicado el funcionamiento del sistema y el modo de empleo del mismo, los usuarios se colocan enfrente del dispositivo Maixduino y mediante un ligero movimiento de cabeza intentan activar y desactivar el botón 1, mandando la orden automatizada en Home Assistant de activar/desactivar el enchufe inteligente tapo P100, al cual se le ha enchufado una pequeña lámpara para que sea más visual el resultado.

Tras las pruebas, se ha realizado una breve encuesta (**TABLA 18**) a cada sujeto donde se ha recogido la facilidad de comprensión del funcionamiento, la facilidad de manejo de la interfaz, la comodidad a la hora de realizar el cambio de estado de los botones, el grado de utilidad que le ven al sistema y, por último, una valoración final.

Sujeto	**1**	**2**	**3**	**4**	**5**	**6**
Edad	44	50	16	22	22	79
Género	Mujer	Hombre	Hombre	Mujer	Hombre	Mujer
Nivel educativo	Ninguno	Bachillerato	ESO	Universidad	Universidad	ESO
Comprensión del funcionamiento	4	5	5	5	5	5
Manejo del dispositivo	4	5	5	5	5	5
Comodidad de uso	5	5	5	5	5	5
Utilidad	5	5	5	5	5	5
Valoración final	**4**	**5**	**5**	**5**	**5**	**5**

TABLA 18. Encuesta sobre la ergonomía del sistema

La valoración consiste en: 0 es muy desfavorable y un 5 muy favorable.

La prueba de mayor interés fue la prueba realizada en la residencia de la 'Asociación salamantina de esclerosis múltiple'. Asociación creada en 1994 de la familiares y mano de personas que sufren esclerosis múltiple.

La asociación ha ofrecido la oportunidad de que una persona que sufre esclerosis múltiple, que afecta a su movilidad, reduciéndola a tan solo el movimiento del cuello, pueda probar nuestro sistema, para así poder observar la comprensión del funcionamiento del dispositivo, el manejo y la desenvoltura que tiene con el dispositivo.

La mujer de 79 años ha tardado en comprender el funcionamiento y el modo de usos del dispositivo solo dos minutos aproximadamente, lo cual nos indica que nuestro sistema es sencillo de comprender y utilizar por cualquier persona.

Tras una serie de pruebas, pudo encender y apagar una lámpara conectada a nuestro enchufe Tapo sin problema alguno. Su encuesta es la correspondiente a la del sujeto 6. Dichas pruebas han permitido comprobar el funcionamiento de todo el sistema en una situación real, añadiendo valor a la experimentación.

A la mujer le interesa poder encender y apagar el televisor, acción que se puede programar y automatizar de la misma forma que se ha automatizado el enchufe Tapo, integrando un dispositivo con infrarrojos que permita el control de televisores y programando los botones necesarios para el control. Por lo que nuestro sistema se puede adaptar a las necesidades y gustos de cada usuario, cambiando el número de botones digitales, utilizando distintos actuadores, etc.

La asociación probó un sistema similar en el que se necesitaba de una pegatina verde colocada en la nariz para poder detectar la misma y poder realizar una seri de actividades. Como se puede observar, el sistema desarrollado en el presente proyecto es capaz de detectar la nariz sin la necesidad de depender de elementos o accesorios externos.

La valoración final dada por los usuarios es alta debido a que les sorprendió bastante la originalidad y funcionamiento del sistema, el poder encender una lámpara con un pequeño movimiento de cabeza les pareció de gran utilidad para ayudar a disminuir la dependencia y aumentar la calidad de vida de aquellas personas que sufren una discapacidad que afecta a sus extremidades superiores.

En cuanto a la comprobación de la fiabilidad del sistema, se ha hecho la prueba de activar/desactivar la lámpara conecta al enchufe tapo un número de cien veces de forma seguida, dando lugar el correcto funcionamiento en todas ellas, por lo que la fiabilidad del sistema se puede decir que es del 100%.

7 ESTUDIO ECONÓMICO

7.1 Introducción

En el capítulo siguiente, realizaremos un estudio económico de todo el proyecto desarrollado. La finalidad de este estudio económico es estimar el costo que tiene llevar a cabo dicho proyecto.

El costo de materiales se ha extraído de la página oficial de sus fabricantes con el IVA incluido, así como la mano de obra se ha calculado según el sueldo medio que se cobra en España en la actualidad para las categorías de ingeniero junior e ingeniero experto.

Para el cálculo de la amortización del computador utilizado para la configuración de los dispositivos hardware y la programación, se ha considerado que tiene una vida útil de 6 años.

7.2 Tareas realizadas durante el proyecto

En el presenta apartado, se describen todas las tareas llevadas cabo durante la realización del proyecto. Las tareas y el tiempo empleado en ellas se van indicando en la **TABLA 19**, **TABLA 20**, **TABLA 21**, y **TABLA 22** en orden cronológico a su realización.

Descripción	Unidad	Cantidad
Reunión inicial tutor TFG	h	1
Estudio de los antecedentes	h	10
Reuniones de control de actividad	h	8
Reuniones selección del material	h	8
Reunión para mostrar el funcionamiento del sistema	h	2
Total	**h**	**29**

TABLA 19. Horas empleadas para la preparación del proyecto

Descripción	Unidad	Cantidad
Organización de la memoria	h	6
Investigación teórica	h	20
Selección de hardware y software a usar	h	30
Aprendizaje del manejo del hardware y software	h	50
Total	**h**	**106**

TABLA 20. Horas empleadas para la investigación del estado del arte

Descripción	Unidad	Cantidad
Configuración del Maixduino	h	45
Configuración de la Raspberry Pi	h	16
Desarrollo del código en Maixpy IDE	h	75
Ajuste de la Raspberry Pi y Maixduino para su comunicación	h	31
Control y automatización del enchufe Tapo P100	h	8
Realización de pruebas y corrección de errores	h	34
Total	**h**	**209**

TABLA 21. Horas empleadas para el desarrollo del sistema de visón artificial

Descripción	Unidad	Cantidad
Redacción de la memoria	h	95
Preparación de la exposición	h	6
Total	**h**	**101**

TABLA 22. Horas empleadas para la redacción y defensa del TFG

El total de horas empleadas en el desarrollo completo del proyecto es de 445 horas.

7.3 Costo de la mano de obra, recursos y herramientas utilizadas para desarrollar el proyecto

En el actual apartado, se indica el costo de toda la mano de obra que ha intervenido en el desarrollo del proyecto. Se ha considerado al alumno como un ingeniero junior y a los tutores como ingenieros expertos.

En las **TABLA 23**, **TABLA 24** y **TABLA 25** se recogen todos los costes que han tenido lugar para la realización del proyecto.

El costo total que lleva hacer el proyecto se recoge en la **TABLA 26**.

Descripción	Precio (€/h)	Cantidad	Amortización	Total (€)
Ingeniero Electrónico Junior	13	445	-	5.785
Ingeniero tutor	31	19	-	589
Ingeniero cotutor	31	19	-	589
			Total (€)	**6.963**

TABLA 23. Costo de la MANO DE OBRA que interviene en la realización del proyecto

Descripción	Precio (€)	Cantidad	Amortización	Total
Placa Raspberry Pi 4 Model B (4 GB RAM)	51,22	1	-	51,22
Cable alimentación Raspberry Pi	8,80	1	-	8,80
Kit Maixduino	75,99	1	-	75,99
Cable Ethernet	3,99	1	-	3,99
Tarjeta MicroSD SanDisk (32 GB)	17,58	2	-	35,16
Lápiz USB	2,99	1	-	2,99
Carcasa protectora Raspberry Pi	4,56	1	-	4,56
Carcasa protectora Maixduino	2,00	1	-	2,00
Enchufe Tapo P100	9,07	1	-	9,07
Portátil Asus ZenBook 13	699	1	0,013	58,25
Cable USB-C	0,99	1	-	0,99
			Total (€)	**253,02**

TABLA 24. Costo del HARDWARE usado en el desarrollo del proyecto

Descripción	Precio (€)	Cantidad	Amortización	Total
Kflash_gui V1.6	0	1	-	0
Flash download tool	0	1	-	0
Maixpy IDE	0	1	-	0
Home Assistant	0	1	-	0
Tinkercad 3D	0	1	-	0
Tapo app	0	1	-	0
BalenaEtcher	0	1	-	0
MobaXterm	0	1	-	0
			Total (€)	**0**

TABLA 25. Costo del SOFTWARE usado en el desarrollo del proyecto

Descripción	Total (€)
Mano de obra	6.963,00
Materiales	253,02
Herramientas	0
Total (€)	**7.222,02**

TABLA 26. Resumen del costo total de la realización del proyecto

Llevar a cabo el proyecto tiene un coste de 7.222,02 euros.

8 CONCLUSIONES Y LÍNEAS FUTURAS

8.1 Conclusiones

El sistema desarrollado cumple con el objetivo de ayudar a personas que sufren discapacidad motora que afecte a sus extremidades superiores, ya que permite mediante la detección de la nariz y una interfaz de usuario poder controlar una serie de actividades domóticas de forma autónoma.

Las distintas propiedades de la placa de visión artificial Maixduino han cumplido con el propósito de la detección de la nariz, así como la conexión wifia para una comunicación con el entorno domótico. Gracias a su pequeño tamaño y al microprocesador de visión artificial que incorpora ha permitido lograr un sistema embebido.

El protocolo de comunicación MQTT permite que el sistema sea desacoplado, pudiendo configurar cada botón de forma independiente, consiguiendo que se puedan hacer diferentes acciones domóticas en función de cada botón.

El uso del entorno domótico Home Assistant aumenta el abanico de posibilidades a la hora de la integración de los dispositivos inteligentes y automatización de los mismos.

El uso de Maixduino, MQTT y Home Assistant para desarrollar el proyecto, otorga al sistema adaptabilidad mediante el cambio del número de botones, su posición en la pantalla y su función domótica, y, mediante la integración de una gran cantidad de dispositivos inteligentes, con sus respectivas automatizaciones.

La adaptabilidad a las distintas necesidades de cada usuario hace que nuestro sistema pueda ser usado en cualquier entorno y por cualquier tipo de persona.

La selección de los dispositivos físicos y programas informáticos, de tal forma, que sean las más adecuadas y que nos permitan cumplir con el objetivo propuesto fue de gran importancia para un desarrollo compacto, robusto, manejable y sencillo de nuestro sistema.

Las distintas pruebas y experimentos realizados al sistema han permitido observar su facilidad de uso, manejabilidad y su fácil y reducida instalación, además de su bajo coste debido a que todo todos los programas utilizados son libres y de coste cero.

Se ha tenido que estudiar en profundidad la placa de visión artificial Maixduino, para asegurar que cumplía con los requisitos necesarios para nuestro proyecto. Una vez elegida la placa, ha sido necesario profundizar en el lenguaje de programación de MicroPython, lenguaje más adecuado para la programación de esta debido al procesador k210 que incorpora.

La mayor dificultad encontrada a la hora de realizar el presente proyecto ha sido a la hora de la configuración y programación de la placa, ya que el lenguaje usado no ha sido cursado en el grado y la placa disponía de poca información debido a su novedad en el mercado, lo que ha provocado que la mayor parte del tiempo empleado en desarrollar el proyecto se haya utilizado para ello.

Ha sido gratificante saber que el proyecto que se estaba labrando tiene una gran utilidad y finalidad moral, la finalidad de ayudar a aquellas personas que sufren dificultades para realizar acciones imprescindibles en la vida diaria. El poder mejorar la calidad de vida de aquellas personas con discapacidad y, por tanto, de las personas que las rodean y ayudan diariamente es un deber de todos. El uso de la tecnología para cumplir con este tipo de problemáticas debe ser siempre una

prioridad.

Finalmente, cabe decir que la realización del trabajo de fin de grado implica un gran esfuerzo y muchas horas de trabajo, durante la realización de dicho proyecto, se han puesto aprueba todos aquellos conocimientos adquiridos durante el grado para llevarlos a un entorno más práctico mediante el desarrollo de este sistema. A medida que se iba avanzando y superando los objetivos propuestos las ganas de seguir trabajando iban también en aumento.

8.2 Líneas futuras

Como línea futura se puede realizar la implementación de este sistema durante un largo periodo de tiempo en una residencia con personas que sufran discapacidad motora o en un domicilio, para así poder observar cómo se desenvuelven con el manejo del dispositivo a lo largo del tiempo y las distintas funcionalidades que se le puede dar al dispositivo desarrollado.

Un proyecto futuro de interés, puede ser el desarrollo de un sistema para aquellas personas cuya discapacidad motora afecta a la totalidad del movimiento del cuerpo, y, por tanto, no tiene la posibilidad del movimiento del cuello. Un sistema que permita mediante el posicionamiento y movimiento de las pupilas poder realizar una serie de acciones domóticas, las cuales consigan disminuir la dependencia de las personas que sufren esta problemática, al igual que el objetivo del presente proyecto.

9 BIBLIOGRAFÍA

[1] INE. (2008). [En línea]. Encuesta de Discapacidad, Autonomía personal y situaciones de Dependencia (EDAD). Recuperado de https://ine.es/prensa/np524.pdf [Consulta: 09- Abr- 2022].

[2] Contaval. (2016). [En línea]. ¿Qué es la visión artificial y para qué sirve? Recuperado de https://www.contaval.es/que-es-la-vision-artificial-y-para-que-sirve/. [Consulta: 23- Abr- 2022].

[3] Porras, José. (2022). [En línea]. CLASIFICATION SYSTEM BASED ON COMPUTER VISION. [Curso: CE 1002 Taller de Electrónica IV, Escuela Profesional de Ingeniería Electrónica. Universidad Ricardo Palma]. Recuperado de https://www.urp.edu.pe/pdf/id/2881/n/clasification-system-based-on-computer-vision. [Consulta: 23- Abr- 2022]

[4] INFAIMON. (2022). [En línea]. ILUMINACIÓN FRONTAL. Recuperado de

https://infaimon.com/enciclopedia-de-la-vision/iluminacion-frontal/ Consulta: 25- Abr- 2022].

[5] Visión artificial. (2022). [En línea]. En CIP ETI Tudela. Recuperado de http://www.etitudela.com/celula/downloads/visionartificial.pdf [Consulta: 25-Abr- 2022].

[6] Equipo técnico. (2022). [En línea]. "¿Qué es la detección de rostros y cómo funciona?. *Krypton Solid*. Recuperado de https://kryptonsolid.com/que-es-la-deteccion-de-rostros-y-como-funciona/#:~:text=Las%20aplicaciones%20de%20detecci%C3%B3n%20de%20rostros%20utilizan%20algoritmos,partes%20del%20cuerpo%20humano%2C%20como%20pies%20o%20manos. [Consulta: 19 de mayo de 2022].

[7] Barrios, Juan. (2022). [En línea]. Redes neuronales convolucionales son un tipo de redes neuronales. *Health big data.* [En línea]. Recuperado de https://www.juanbarrios.com/redes-neurales-convolucionales/#Arquitectura_basica_RESUMEN. [Consulta: 17 de mayo de 2022].

[8] Dirección general de formación de maestros. (2022). [En línea]. *Comprensión de la discapacidad VII para el proceso formativo de las y los estudiantes de las ESFM.* Ministerio de educación, estado plurinacional de Bolivia. Recuperado de https://www.minedu.gob.bo/files/publicaciones/veaye/dgee/jica12_DISCAPACIDAD_FISICO-MOTORA.pdf. [Consulta: 23- Abr- 2022].

[9] Desconocido. (2022). [En línea]. LIFEWAREINTEGRA, *Tecnología asistiva para personas con discapacidad motora.* Recuperado de: https://tecnologiasistiva5.blogspot.com/2016/07/lifewareintegra-una-util-herramienta.html. [Consulta: 25- Abr- 2022].

[10] "Sip/Puff Switch", (2022). [En línea]. Recuperado de https://tecnoaccesible.net/catalogo/sippuff-switch. [Consulta: 25- Abr- 2022].

[11]"Head Mouth Stick Keyboard Info", *Maltron Ergonomic Keyboards*. (2022). [En línea]. Recuperado de https://www.maltron.com/head-mouth-stick-keyboard-info.html. [Consulta: 25- Abr- 2022].

[12] "Head Wand", *Christina Giambrone*. (2022). [En línea]. Recuperado de

https://cgiambrone.wordpress.com/2014/09/23/head-wand/. [Consulta: 19 de mayo de 2022].

[13] "Sipeed Maixduino Kit for RISC-V AI + IoT", *Seeedstudio.com*. (2022). [En línea]. Recuperado de https://www.seeedstudio.com/Sipeed-Maixduino-Kit-for-RISC-V-AI-IoT-p-4047.html. [Consulta: 12- Abr- 2022].

[14] "Sipeed Maixduino Datasheet", *Mantech.co.za*. (2022). [En línea]. Recuperado de https://www.mantech.co.za/Datasheets/Products/MAIXDUINO-20100410A.pdf. [Consulta: 12- Abr- 2022].

[15] "MaixDuino - Sipeed Wiki", *Wiki.sipeed.com*. (2022). [En línea]. Recuperado de https://wiki.sipeed.com/soft/maixpy/en/develop_kit_board/maix_duino.html. [Consulta: 12- Apr- 2022].

[16] "Raspberry Pi: Qué es, para qué sirve y qué podemos hacer", *Profesional Review*. (2022). [En línea]. Recuperado de https://www.profesionalreview.com/2021/07/18/que-es-raspberry-pi/#Que_es_Raspberry_Pi. [Consulta: 13- Apr- 2022].

[17] "Raspberry PI 4 Computer, Model B", *Datasheets.raspberrypi.com*. (2022). [En línea]. Recuperado de https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf. [Consulta: 13- Apr- 2022].

[18] "Buy a Raspberry Pi 4 Model B – Raspberry Pi", *Raspberry Pi*. (2022). [En línea]. Recuperado de https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. [Consulta: 13- Apr- 2022].

[19] "Tapo P100 | Mini Enchufe Inteligente Wi-Fi | Tapo", *Tapo.com*. (2022). [En línea]. Recuperado de https://www.tapo.com/es/product/smart-plug/tapo-p100/. [Consulta: 13- Apr- 2022].

[20] "Tapo L510E | Bombilla Smart Wi-Fi, regulable | Tapo", *Tapo.com*. (2022). [En línea]. Recuperado de https://www.tapo.com/es/product/smart-light-bulb/tapo-l510e/. [Consulta: 14- Abr- 2022].

[21] "SwitchBot Bot", *SwitchBot Global*. (2022). [En línea]. Recuperado de https://www.switch-bot.com/products/switchbot-bot. [Consulta: 14- Abr- 2022].

[22] "工具 | 乐鑫科技", *Espressif.com*. (2022). [En línea]. Recuperado de https://www.espressif.com/zh-hans/support/download/other-tools. [Consulta: 17- Abr- 2022].

[23] "MaixPy 文档简介 - Sipeed Wiki", *Wiki.sipeed.com*. (2022). [En línea]. Recuperado de https://wiki.sipeed.com/soft/maixpy/zh/index.html. [Consulta: 14- Abr- 2022].

[24] "Home Assistant", *Home Assistant*. (2022). [En línea]. Recuperado de https://www.home-assistant.io/. [Consulta: 14- Abr- 2022].

[25] "Tinkercad | De la mente al diseño en minutos", *Tinkercad*. (2022). [En línea]. Recuperado de https://www.tinkercad.com/dashboard. [Consulta: 16- Abr- 2022].

[26] "TP-Link Tapo", *App Store*. (2022). [En línea]. Recuperado de https://apps.apple.com/es/app/tp-link-tapo/id1472718009. [Consulta: 16- Abr- 2022].

[27] "balenaEtcher - Flash OS images to SD cards & USB drives", *balenaEtcher*. (2022). [En línea]. Recuperado de https://www.balena.io/etcher/. [Consulta: 19- Abr- 2022].

[28] "MobaXterm free Xserver and tabbed SSH client for Windows", *Mobaxterm.mobatek.net*. (2022). [En línea]. Recuperado de https://mobaxterm.mobatek.net/. [Consulta: 20- Abr- 2022].

[29] "Sipeed MaixHub – sipeed AI 模型平台", *Maixhub.com*. (2022). [En línea]. Recuperado de https://www.maixhub.com/modelInfo?modelId=14. [Consulta: 16- Abr- 2022].

[30] "下载站 - Sipeed", *Cn.dl.sipeed.com*. (2022). [En línea]. Recuperado de https://cn.dl.sipeed.com/. [Consulta: 17- Abr- 2022].

[31] "Online UUID Generator Tool", *Uuidgenerator.net*. (2022). [En línea]. Recuperado de https://www.uuidgenerator.net/version4. [Consulta: 19- Abr- 2022].

[32] "Comience | | DNS público Google Developers", *Google Developers.* (2022). [En línea]. Recuperado de https://developers.google.com/speed/public-dns/docs/using. [Consulta: 19- Abr- 2022].

[33] "A case for Sipeed Maixduino by komix", *Thingiverse.com*. (2022). [En línea]. Recuperado de https://www.thingiverse.com/thing:3768964. [Consulta: 28- Abr- 2022].

[34] "¿Puede alguien aclarar el concepto de caja de anclaje utilizado en Yolo? · Edición #568 · pjreddie/darknet", *GitHub*. (2022). [En línea]. Recuperado de https://github.com/pjreddie/darknet/issues/568. [Consulta: 17- Abr- 2022].

[35] "Guía de introducción a MQTT con ESP8266 y Raspberry Pi", *Programar fácil con Arduino*. (2022). [En línea]. Recuperado de https://programarfacil.com/esp8266/mqtt-esp8266-raspberry-pi/. [Consulta: 26- Abr- 2022].

[36] "What HACS can do". *HACS*. (2022). [En línea]. Recuperado de https://hacs.xyz/docs/faq/highlights. [Consulta: 20- Abr- 2022].

10 ANEXOS

ANEXO I. Manual de instalación y uso del sistema

I.1 Introducción

En el anexo siguiente, se indicará un resumen de como conectar los dispositivos que intervienen en el sistema para su correcta instalación.

I.2 Instalación de los dispositivos.

En el sistema intervienen dos dispositivos principales, el Maixduino y la Raspberry Pi.

El Maixduino con el programa desarrollado solo necesita ser conectado a una fuente de alimentación como una batería portátil de 5V o a un enchufe de igual

forma que realizamos la carga de un teléfono móvil. Una vez se alimenta el dispositivo ya se empezaría a ejecutar el programa y estaría disponible para su uso.

En cuanto a la Raspberry Pi, tendremos que conectarle el USB con la configuración que se ha indicado en el apartado **4.3.2** de la memoria, a continuación, conectaremos el cable ethernet al router, introducimos la tarjeta MicroSD en la ranura correspondiente, la cual contiene el software de Home Assistant y, por último, alimentamos la Raspberry Pi.

Procedemos a entrar en el servidor de Home Assistant para la configuración del actuador o actuadores. En este caso tomamos como ejemplo el enchufe inteligente Tapo que para su configuración se siguen los pasoso indicados en el apartado **5.6.4.**

Realizados los pasos descritos anteriormente, el sistema ya estaría listo para su uso.

I.3 Modo de empleo del sistema

El uso del dispositivo Maixduino es muy sencillo, una vez situado enfrente del paciente con discapacidad motora, este puede controlar una serie de actividades con ligeros movimientos de cabeza.

El paciente podrá accionar y desactivar los botones digitales que aparecen en la pantalla mediante el posicionamiento de la punta de la nariz en estos, basta con permanecer cuatro segundos hasta observar el cambio de color de los botones; el cambio de color de rojo a verde nos indicará que la acción domótica se ha activado y el cambio de verde a rojo nos indicará que la acción domótica se ha desactivado.

El paciente observará que la placa de Maixduino está detectando la nariz y que está posicionado correctamente en el botón gracias a una pequeña cruz que se deposita sobre la punta de la nariz y a la iluminación de los botones, ya que, una vez que se apunta a un botón, este permanece iluminado de un color hasta pasados los cuatro segundos, momento en el que el color cambia.

ANEXO II. Programa desarrollado en MaixPy IDE

II.1 Introducción

En el presente anexo, se indica como quemar el código en la placa de Maixduino, así como el código al completo cargado en la misma, en nuestro caso, este programa se podrá cargar en tantas placas Maixduino como se desee, permitiendo el uso de varios de estos dispositivos los cuales se conectan a un mismo servidor (Home Assistant), lo cual disminuye el coste, ya que solo necesitaríamos una Raspberry Pi a la que se podrían conectar una gran cantidad de dispositivos Maixduino.

II.2 Carga del programa en Maixduino

En este apartado se muestra como cargar el programa en la placa Maixduino, lo que va a permitir que, una vez alimentada la placa, se ejecute el código cargado en la misma directamente, sin necesidad de un ordenador. Para ello, introducimos una tarjeta microSD de mínimo 4 GB en la placa Maixduino, para evitar problemas de espacio en memoria y conectamos la placa al ordenador y al software Maixpy. A continuación, vamos a **Herramientas > Save open script to board (booty.py)** como se muestra en la **FIGURA 112**.

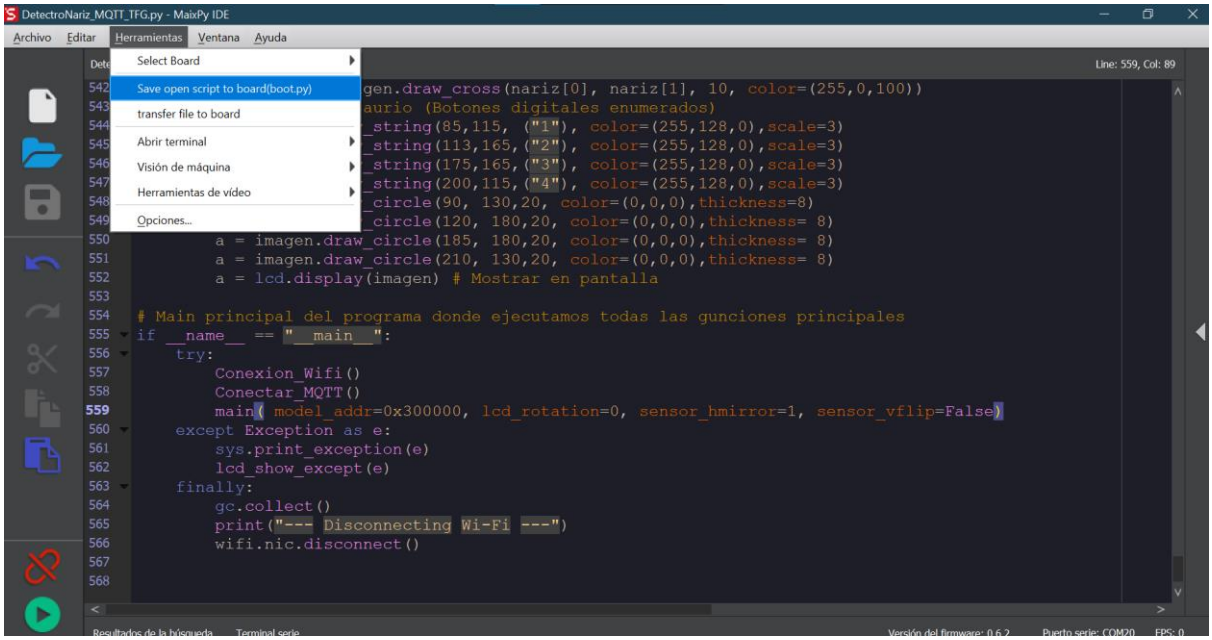

FIGURA 112.Quemar el código de Maixpy IDE a la placa Maixduino

II.3 Código cargado en Maixduino

A continuación, se muestra el diagrama de flujo correspondiente al proceso seguido por el programa desarrollado en Maixpy IDE (**FIGURA 113**).

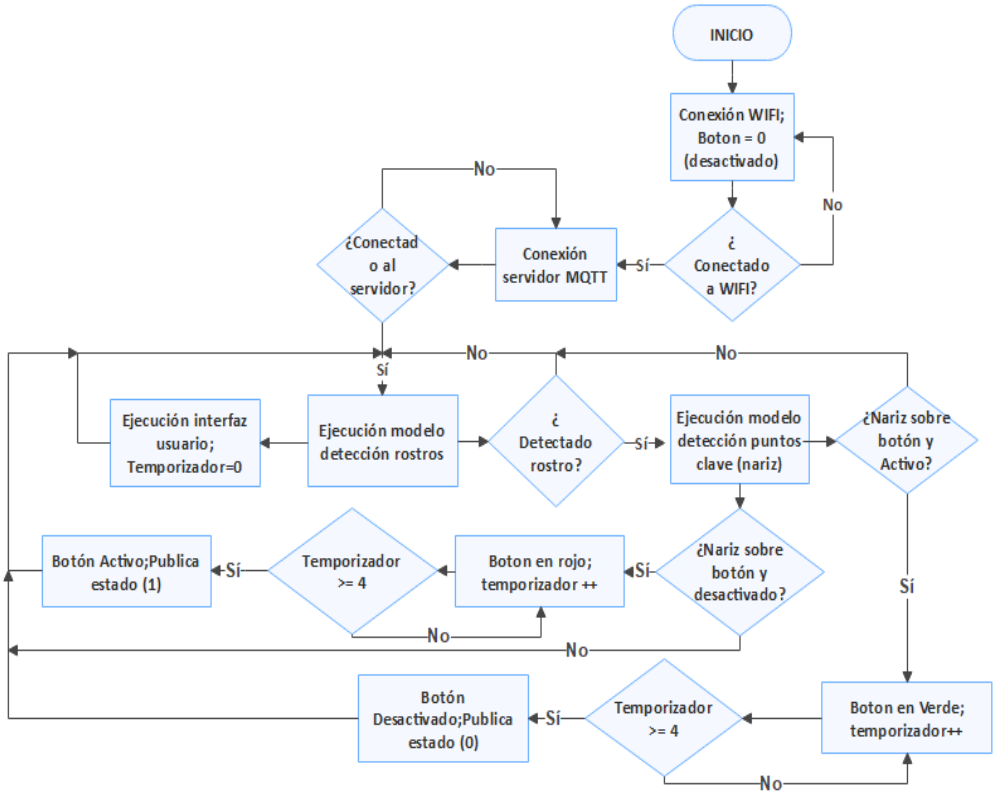

FIGURA 113.Diagrama de flujo del programa desarrollado en Maixpy IDE

El código desarrollado y cargado en la placa Maixduino y que permite cumplir todos nuestros objetivos es el que se muestra en la **TABLA 27**.

```
#######################################################################
#                                                                     #
#                                                                     #
#                                                                     #
#                    SMAIL BOUAOUDA - UVa - 2022                      #
#                                                                     #
#                                                                     #
#                                                                     #
#                        Placa MAIXDUINO                              #
#                                                                     #
#                                                                     #
#                                                                     #
# El código desarrollado permite realizar lo siguiente:              #
#                                                                     #
#                                                                     #
#                                                                     #
#    * Conexión a la red WIFI                                         #
#                                                                     #
```

```
#    * Comunicación con Home Assistant mediante protocolo MQTT  #
#                                                               #
#                                                               #
#    * Detección de rostro                                      #
#                                                               #
#    * Detección de nariz                                       #
#                                                               #
#    * Pantalla con interfaz de usuario con 4 botones digitales #
#                                                               #
#                                                               #
#                                                               #
#################################################################


#Importamos módulos
import sensor, image, lcd, time
import KPU as kpu
import gc, sys
import network
from Maix import GPIO
from fpioa_manager import fm
import usocket as socket
import ustruct as struct
from ubinascii import hexlify
import json

# Parámateros WIFI
SSID_WIFI = "Mecatronica1"
CONTRASE_WIFI = "*Disa.Eii*"

#Parámetros MQTT
SERVIDOR_MQTT     = "192.168.0.98"
PUERTO_MQTT       = 1883
USUARIO_MQTT   = "ISMQTT"
CONTRASE_MQTT   = "EMQTT"
IDENTIFICADOR_CLIENTE_MQTT = "MaixDuino"
MQTT_PUB_TOPIC_BOTON_1 = "MaixDuino/Boton/1"
MQTT_PUB_TOPIC_BOTON_2 = "MaixDuino/Boton/2"
MQTT_PUB_TOPIC_BOTON_3 = "MaixDuino/Boton/3"
MQTT_PUB_TOPIC_BOTON_4 = "MaixDuino/Boton/4"

# Variables globales
MQTT_Conectado = False
cliente = None
Boton_1 = 0
Boton_2 = 0
Boton_3 = 0
Boton_4 = 0
contador = 0
Tiempo_act_des = 4

# Clase MQTT
class MQTTException(Exception):
    pass

class MQTTClient:

    connect_status_rostros = ["Connection accepted", "Connection
refused, unacceptable protocol version",
```

```
        "Connection refused, identifer rejected", "Connection
refused, server unavailable",
        "Connection refused, bad user name or password",
"Connection refused, not authorized"]

    def __init__(self, cliente_id, server, port=0, user=None,
password=None, keepalive=0,
                 ssl=False, ssl_params={}):
        if port == 0:
            port = 8883 if ssl else 1883
        self.cliente_id = cliente_id
        self.sock = None
        self.server = server
        self.port = port
        self.ssl = ssl
        self.ssl_params = ssl_params
        self.pid = 0
        self.cb = None
        self.user = user
        self.pswd = password
        self.keepalive = keepalive
        self.lw_topic = None
        self.lw_msg = None
        self.lw_qos = 0
        self.lw_retain = False
        self.last_connect_status = None

    def _send_str(self, s):
        self.sock.write(struct.pack("!H", len(s)))
        self.sock.write(s)

    def _recv_len(self):
        n = 0
        sh = 0
        while 1:
            b = self.sock.read(1)[0]
            n |= (b & 0x7f) << sh
            if not b & 0x80:
                return n
            sh += 7

    def set_callback(self, f):
        self.cb = f

    def set_last_will(self, topic, msg, retain=False, qos=0):
        assert 0 <= qos <= 2
        assert topic
        self.lw_topic = topic
        self.lw_msg = msg
        self.lw_qos = qos
        self.lw_retain = retain

    def connect(self, clean_session=True):
        self.sock = socket.socket()
        addr = socket.getaddrinfo(self.server, self.port)[0][-1]
        self.sock.connect(addr)
        if self.ssl:
            import ussl
```

```
            self.sock = ussl.wrap_socket(self.sock,
**self.ssl_params)
        premsg = bytearray(b"\x10\0\0\0\0\0")
        msg = bytearray(b"\x04MQTT\x04\x02\0\0")

        sz = 10 + 2 + len(self.cliente_id)
        msg[6] = clean_session << 1
        if self.user is not None:
            sz += 2 + len(self.user) + 2 + len(self.pswd)
            msg[6] |= 0xC0
        if self.keepalive:
            assert self.keepalive < 65536
            msg[7] |= self.keepalive >> 8
            msg[8] |= self.keepalive & 0x00FF
        if self.lw_topic:
            sz += 2 + len(self.lw_topic) + 2 + len(self.lw_msg)
            msg[6] |= 0x4 | (self.lw_qos & 0x1) << 3 |
(self.lw_qos & 0x2) << 3
            msg[6] |= self.lw_retain << 5

        i = 1
        while sz > 0x7f:
            premsg[i] = (sz & 0x7f) | 0x80
            sz >>= 7
            i += 1
        premsg[i] = sz

        self.sock.write(premsg, i + 2)
        self.sock.write(msg)
        #print(hex(len(msg)), hexlify(msg, ":"))
        self._send_str(self.cliente_id)
        if self.lw_topic:
            self._send_str(self.lw_topic)
            self._send_str(self.lw_msg)
        if self.user is not None:
            self._send_str(self.user)
            self._send_str(self.pswd)
        resp = self.sock.read(4)
        assert resp[0] == 0x20 and resp[1] == 0x02
        self.last_connect_status = resp[3]
        if resp[3] != 0:
            raise MQTTException(str(resp[3]) + ' ' +
self.get_last_connect_status_message())
        return resp[2] & 1

    def disconnect(self):
        self.sock.write(b"\xe0\0")
        self.sock.close()

    def ping(self):
        self.sock.write(b"\xc0\0")

    def publish(self, topic, msg, retain=False, qos=0):
        pkt = bytearray(b"\x30\0\0\0")
        pkt[0] |= qos << 1 | retain
        sz = 2 + len(topic) + len(msg)
        if qos > 0:
            sz += 2
        assert sz < 2097152
```

```
        i = 1
        while sz > 0x7f:
            pkt[i] = (sz & 0x7f) | 0x80
            sz >>= 7
            i += 1
        pkt[i] = sz
        #print(hex(len(pkt)), hexlify(pkt, ":"))
        self.sock.write(pkt, i + 1)
        self._send_str(topic)
        if qos > 0:
            self.pid += 1
            pid = self.pid
            struct.pack_into("!H", pkt, 0, pid)
            self.sock.write(pkt, 2)
        self.sock.write(msg)
        if qos == 1:
            while 1:
                op = self.wait_msg()
                if op == 0x40:
                    sz = self.sock.read(1)
                    assert sz == b"\x02"
                    rcv_pid = self.sock.read(2)
                    rcv_pid = rcv_pid[0] << 8 | rcv_pid[1]
                    if pid == rcv_pid:
                        return
        elif qos == 2:
            assert 0

    def subscribe(self, topic, qos=0):
        assert self.cb is not None, "Subscribe callback is not
set"
        pkt = bytearray(b"\x82\0\0\0")
        self.pid += 1
        struct.pack_into("!BH", pkt, 1, 2 + 2 + len(topic) + 1,
self.pid)
        #print(hex(len(pkt)), hexlify(pkt, ":"))
        self.sock.write(pkt)
        self._send_str(topic)
        self.sock.write(qos.to_bytes(1, "little"))
        while 1:
            op = self.wait_msg()
            if op == 0x90:
                resp = self.sock.read(4)
                #print(resp)
                assert resp[1] == pkt[2] and resp[2] == pkt[3]
                if resp[3] == 0x80:
                    raise MQTTException(resp[3])
                return

    # Espere un solo mensaje MQTT entrante y procesarlo
    # Los mensajes suscritos se entregan a una devolución de
llamada
    # establecido por el método .set_callback(). Otro MQTT
(interno)
    # mensajes procesados internamente.
    def wait_msg(self):
        res = self.sock.read(1)
        self.sock.setblocking(True)
        if res is None:
```

```
            return None
        if res == b"":
            raise OSError(-1)
        if res == b"\xd0":  # PINGRESP
            sz = self.sock.read(1)[0]
            assert sz == 0
            return None
        op = res[0]
        if op & 0xf0 != 0x30:
            return op
        sz = self._recv_len()
        topic_len = self.sock.read(2)
        topic_len = (topic_len[0] << 8) | topic_len[1]
        topic = self.sock.read(topic_len)
        sz -= topic_len + 2
        if op & 6:
            pid = self.sock.read(2)
            pid = pid[0] << 8 | pid[1]
            sz -= 2
        msg = self.sock.read(sz)
        self.cb(topic, msg)
        if op & 6 == 2:
            pkt = bytearray(b"\x40\x02\0\0")
            struct.pack_into("!H", pkt, 2, pid)
            self.sock.write(pkt)
        elif op & 6 == 4:
            assert 0

    # Comprueba si un mensaje pendiente del servidor está
disponible
    # Si no, regresa inmediatamente con Ninguno.
    # De lo contrario,el mismo procesamiento que wait_msg.
    def check_msg(self):
        self.sock.setblocking(False)
        return self.wait_msg()

    def get_last_connect_status_message(self):
        if self.last_connect_status == None:
            return "(not defined)"
        elif self.last_connect_status in
range(len(self.connect_status_rostros)):
            return
self.connect_status_rostros[self.last_connect_status]
        else:
            return "UNKNOWN MQTT STATUS"

# Clase Wi-Fi para ESP32
class wifi():

    nic = None

    def reset(force=False, reply=5, is_hard=True):
        if force == False and __class__.isconnected():
            return True
        try:
            # Mapa IO para ESP32 en Maixduino
            fm.register(25,fm.fpioa.GPIOHS10)#cs
            fm.register(8,fm.fpioa.GPIOHS11)#rst
            fm.register(9,fm.fpioa.GPIOHS12)#rdy
```

```
            fm.register(28,fm.fpioa.GPIOHS13, force=True)#mosi
            fm.register(26,fm.fpioa.GPIOHS14, force=True)#miso
            fm.register(27,fm.fpioa.GPIOHS15, force=True)#sclk
            __class__.nic =
network.ESP32_SPI(cs=fm.fpioa.GPIOHS10,rst=fm.fpioa.GPIOHS11,rdy
=fm.fpioa.GPIOHS12,
mosi=fm.fpioa.GPIOHS13,miso=fm.fpioa.GPIOHS14,sclk=fm.fpioa.GPIO
HS15)
            print(" Version de firmware ESP32_SPI :",
__class__.nic.version())

        except Exception as e:
            print(e)
            return False
        return True

    def connect(ssid="wifi_name", pasw="pass_word"):
        if __class__.nic != None:
            return __class__.nic.connect(ssid, pasw)

    def ifconfig():
        if __class__.nic != None:
            return __class__.nic.ifconfig()

    def isconnected():
        if __class__.nic != None:
            return __class__.nic.isconnected()
        return False

# Conectarse al servidor MQTT
def Conexion_MQTT(cliente):
    global MQTT_Conectado
    if not MQTT_Conectado:
        try:
            cliente.connect()
            MQTT_Conectado = True
            print("MQTT conectado {}".format(SERVIDOR_MQTT))
        except Exception as e:
            MQTT_Conectado = False
            print("Fallo en la conexion MQTT: {}".format(e))
    else:
        print("MQTT Conectado")

#Desconectarse del servidor MQTT
def Desconexion_MQTT(cliente):
    global MQTT_Conectado
    if MQTT_Conectado:
        try:
            cliente.disconnect()
        except:
            pass
    MQTT_Conectado = False

#Conectarse a la red WIFI
def Conexion_Wifi():
    while not wifi.isconnected():
        print("--- Reset ESP32 ---")
        wifi.reset()
        print("--- Conectando a Wi-Fi ---")
```

```
        try:
            wifi.connect(SSID_WIFI, CONTRASE_WIFI)
        except Exception as e:
            print("Conexion fallida: {}".format(e))

    print("Wi-Fi conectado a:", wifi.isconnected(),
wifi.ifconfig())

    # Ping google.com
    ping_host = "google.com"
    print("--- Ping {}: {} ms ---".format(ping_host,
wifi.nic.ping(ping_host)))

# Paso de los parámateros para la conexión al servidor MQTT
def Conectar_MQTT():
    global cliente, MQTT_Conectado
    cliente = MQTTClient(cliente_id=IDENTIFICADOR_CLIENTE_MQTT,
server=SERVIDOR_MQTT, port=PUERTO_MQTT,
                         user=USUARIO_MQTT,
password=CONTRASE_MQTT)
    while not MQTT_Conectado and wifi.isconnected():
        print("--- Conectando al broker MQTT  {} ---
".format(SERVIDOR_MQTT))
        Conexion_MQTT(cliente)
        if not MQTT_Conectado:
            time.sleep(1)

#Publicar mensajes al servidor MQTT
def Publicar_MQTT():
    global cliente,
MQTT_Conectado,Boton_1,Boton_2,Boton_3,Boton_4
    if not wifi.isconnected():
        if MQTT_Conectado:
            Desconexion_MQTT(cliente)
            MQTT_Conectado = False
        Conexion_Wifi()
    Intentos = 3 #Intentos para conectarse al servidor
    while Intentos > 0:
        if not MQTT_Conectado:
            Conexion_MQTT(cliente)
        if MQTT_Conectado:
            try:
                #Una publicación por tópico
                cliente.publish(MQTT_PUB_TOPIC_BOTON_1,
json.dumps(Boton_1))
                cliente.publish(MQTT_PUB_TOPIC_BOTON_2,
json.dumps(Boton_2))
                cliente.publish(MQTT_PUB_TOPIC_BOTON_3,
json.dumps(Boton_3))
                cliente.publish(MQTT_PUB_TOPIC_BOTON_4,
json.dumps(Boton_4))
                print("Mensaje publicado")
                break
            except Exception as e:
                print("MQTT publicacion fallida")
                Desconexion_MQTT(cliente)
        else:
            print("MQTT no conectado, publicacion fallida")
        Intentos = Intentos - 1
```

```
        if Intentos > 0:
            print("Se intentara conectar {}
mas".format(Intentos))

# Características de la pantalla cuando se produce la
interrupción del programa
def lcd_show_except(e):
    import uio
    err_str = uio.StringIO()
    sys.print_exception(e, err_str)
    err_str = err_str.getvalue()
    imagen = image.Image(size=(224,224))
    imagen.draw_string(0, 10, err_str, scale=1,
color=(0xff,0x00,0x00))
    lcd.display(imagen)

# Ejecución de la detección de la nariz y la interfaz de usuario
def main(model_addr=0x300000, lcd_rotation=0,
sensor_hmirror=False, sensor_vflip=False):

 global
Light_Estate,contador,Boton_Llamada,Boton_1,Boton_2,Boton_3,Boto
n_4

#SENSORES
    sensor.reset()
    sensor.set_pixformat(sensor.RGB565)
    sensor.set_framesize(sensor.QVGA)
    sensor.set_hmirror(sensor_hmirror)
    sensor.set_vflip(sensor_vflip)
    sensor.run(1)
#LCD
    #lcd.init(type=1)
    lcd.init(freq=15000000)
    lcd.rotation(lcd_rotation)
    lcd.clear(lcd.WHITE)

    task_fd = kpu.load(model_addr)
    task_ld = kpu.load(0x400000)
    anchor = (1.889, 2.5245, 2.9465, 3.94056, 3.99987, 5.3658,
5.155437, 6.92275, 6.718375, 9.01025) #Ancla para detección de
rostros
    IMDR = kpu.init_yolo2(task_fd, 0.5, 0.3, 5, anchor)
#Inicializar el modelo de detección de rostros

    while(1): # Bucle principal
        imagen = sensor.snapshot() #Obtener una foto de la
cámara
        rostro = kpu.run_yolo2(task_fd, imagen) # Ejecute el
modelo de detección de rostros para obtener la posición de las
coordenadas del rostro
        if rostro: # Si se detecta un rostro
            for i in rostro: # Cuadro de coordenadas iterativo
                # Cortar la cara y cambiar el tamaño a 128x128
                #a = imagen.draw_rectangle(i.rect()) # Mostrar
el marco de la cara en la pantalla
                face_cut=imagen.cut(i.x(),i.y(),i.w(),i.h()) #
Recorta parte de la imagen de la cara para face_cut
```

```
                face_cut_128=face_cut.resize(128,128) # Escale
la imagen de la cara recortada a 128 * 128 píxeles
                IMDR =face_cut_128.pix_to_ai() # Convierta la
imagen detectada al formato aceptado por kpu
                # Marcas de la cara
                fmap = kpu.forward(task_ld, face_cut_128) #
Ejecute el modelo de detección de puntos clave del rostro
                plist=fmap[:] # Obtenga resultados clave de
predicción
                nariz=(i.x()+int(plist[4]*i.w()),
i.y()+int(plist[5]*i.h())) #Calcular la posición de la nariz
                if (Boton_1 == 0 and (nariz[0]>70 and
nariz[0]<110) and (nariz[1]>110 and nariz[1]<150)):
                    contador=contador+1
                    tiempo=contador/6
                    if (tiempo <= Tiempo_act_des):
                        IMDR = imagen.draw_circle(90, 130,20,
color=(255,0,0),thickness=12)
                    if (tiempo >= Tiempo_act_des):
                        Boton_1 = 1
                        contador = 0
                        Publicar_MQTT()
                        print('Boton 1: ON {}'.format(Boton_1))

                elif (Boton_1 == 1 and (nariz[0]>70 and
nariz[0]<110) and (nariz[1]>110 and nariz[1]<150)):
                    contador=contador+1
                    tiempo=contador/6
                    if (tiempo <= Tiempo_act_des ):
                        IMDR = imagen.draw_circle(90, 130,20,
color=(0,255,0),thickness=12)

                    if (tiempo >= Tiempo_act_des ):
                        Boton_1 = 0
                        contador = 0
                        Publicar_MQTT()
                        print('Boton 1: OFF {}'.format(Boton_1))


                elif (Boton_2 == 0 and (nariz[0]>100 and
nariz[0]<140) and (nariz[1]>160 and nariz[1]<200)):
                    contador=contador+1
                    tiempo=contador/6
                    if (tiempo <= Tiempo_act_des ):
                        IMDR = imagen.draw_circle(120, 180,20,
color=(255,0,0),thickness=12)

                    if (tiempo >= Tiempo_act_des):
                        Boton_2 = 1
                        contador = 0
                        Publicar_MQTT()
                        print('Boton 2: ON {}'.format(Boton_2))

                elif (Boton_2 == 1 and (nariz[0]>100 and
nariz[0]<140) and (nariz[1]>160 and nariz[1]<200)):
                    contador=contador+1
                    tiempo=contador/6
                    if (tiempo <= Tiempo_act_des ):
```

```
                        IMDR = imagen.draw_circle(120, 180,20,
color=(0,255,0),thickness=12)

                    if (tiempo >= Tiempo_act_des ):
                        Boton_2 = 0
                        contador = 0
                        Publicar_MQTT()
                        print('Boton 2: OFF {}'.format(Boton_2))


                elif (Boton_3 == 0 and (nariz[0]>170 and
nariz[0]<210) and (nariz[1]>160 and nariz[1]<200)):
                    contador=contador+1
                    tiempo=contador/6
                    if (tiempo <= Tiempo_act_des ):
                        IMDR = imagen.draw_circle(185, 180,20,
color=(255,0,0),thickness=12)

                    if (tiempo >= Tiempo_act_des ):
                        Boton_3 = 1
                        contador = 0
                        Publicar_MQTT()
                        print('Boton 3: ON {}'.format(Boton_3))

                elif (Boton_3 == 1 and (nariz[0]>170 and
nariz[0]<210) and (nariz[1]>160 and nariz[1]<200)):
                        contador=contador+1
                        tiempo=contador/6
                        if (tiempo <= Tiempo_act_des ):
                            IMDR = imagen.draw_circle(185,
180,20, color=(0,255,0),thickness=12)

                        if (tiempo >= Tiempo_act_des ):
                            Boton_3 = 0
                            contador = 0
                            Publicar_MQTT()
                            print('Boton 3: OFF
{}'.format(Boton_3))


                elif (Boton_4 == 0 and (nariz[0]>190 and
nariz[0]<230) and (nariz[1]>110 and nariz[1]<150)):
                    contador=contador+1
                    tiempo=contador/6
                    if (tiempo <= Tiempo_act_des ):
                        IMDR = imagen.draw_circle(210, 130,20,
color=(255,0,0),thickness=12)

                    if (tiempo >= Tiempo_act_des):
                        Boton_4 = 1
                        contador = 0
                        Publicar_MQTT()
                        print('Boton 4: ON {}'.format(Boton_4))

                elif (Boton_4 == 1 and (nariz[0]>190 and
nariz[0]<230) and (nariz[1]>110 and nariz[1]<150)):
                        contador=contador+1
                        tiempo=contador/6
                        if (tiempo <= Tiempo_act_des):
```

```
                                IMDR = imagen.draw_circle(210,
130,20, color=(0,255,0),thickness=12)

                            if (tiempo >= Tiempo_act_des):
                                Boton_4 = 0
                                contador = 0
                                Publicar_MQTT()
                                print('Boton 4: OFF
{}'.format(Boton_4))


                    else:
                        contador=0
                    IMDR = imagen.draw_cross(nariz[0], nariz[1], 10,
color=(255,0,100))
            #Interfaz de usaurio (Botones digitales enumerados)
            IMDR = imagen.draw_string(85,115, ("1"),
color=(255,128,0),scale=3)
            IMDR = imagen.draw_string(113,165,("2"),
color=(255,128,0),scale=3)
            IMDR = imagen.draw_string(175,165,("3"),
color=(255,128,0),scale=3)
            IMDR = imagen.draw_string(200,115,("4"),
color=(255,128,0),scale=3)
            IMDR = imagen.draw_circle(90, 130,20,
color=(0,0,0),thickness=8)
            IMDR = imagen.draw_circle(120, 180,20,
color=(0,0,0),thickness= 8)
            IMDR = imagen.draw_circle(185, 180,20,
color=(0,0,0),thickness= 8)
            IMDR = imagen.draw_circle(210, 130,20,
color=(0,0,0),thickness= 8)
            IMDR = lcd.display(imagen) # Mostrar en pantalla



# Main principal del programa donde ejecutamos todas las
funciones principales
if __name__ == "__main__":
    try:
        Conexion_Wifi()
        Conectar_MQTT()
        main( model_addr=0x300000, lcd_rotation=0,
sensor_hmirror=1, sensor_vflip=False)

    except Exception as e:
        sys.print_exception(e)
        lcd_show_except(e)
    finally:
        gc.collect()
        print("--- Disconnecting Wi-Fi ---")
        wifi.nic.disconnect()
```

TABLA 27. Código desarrollado y cargado en la placa de Maixduino