Universidad de Valladolid


UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

# Desarrollo de una Aplicación Android para el Control de Entornos Inteligentes mediante el Reconocimiento Visual de Gestos Faciales

Autor:

Blanco Bezos, Adrián Santiago

Tutores:

Gómez García-Bermejo, Jaime

Gómez Ramos, Raúl

Valladolid, Julio 2024

# AGRADECIMIENTOS

Querría expresar mi más sincero agradecimiento a todas las personas que me han acompañado y apoyado a lo largo de mis años de estudio universitario.

A mis padres y a mi hermana por su apoyo incondicional. Siempre me han ayudado y soportado aun cuando menos lo merecía.

Querría también agradecer a mis tutores de TFG por haberme ayudado siempre que he tenido dudas o consultas sobre el desarrollo del proyecto.

# RESUMEN

El presente proyecto consiste en el desarrollo de una aplicación destinada a dispositivos Android para permitir el control de dispositivos inteligentes a través de la realización de ciertos gestos faciales.

Esta aplicación está pensada para ayudar a personas con problemas de movilidad a reducir su dependencia, permitiéndoles gestionar dispositivos que previamente no podrían controlar sin la ayuda de un asistente personal.

La aplicación se basa en la detección de una serie de puntos faciales del usuario por medio de visión artificial, para posteriormente calcular la distancia entre ciertos puntos detectados. Una vez calculada esa distancia, se la compara con un umbral de detección, de modo que si la distancia es mayor o menor que ese umbral se considera que se ha realizado el gesto facial o no.

Al haberse detectado la realización de ciertos gestos faciales, se envían mensajes a dispositivos inteligentes para que respondan en consecuencia y actúen así sobre el entorno del usuario de la manera deseada, permitiendo así el control de sus alrededores simplemente con gestos faciales.

Adicionalmente, se realizará el acondicionamiento de un robot de asistencia a la alimentación para poder accionarlo mediante los mensajes enviados desde la aplicación en lugar de empleando sus interruptores originales, permitiendo a personas que no puedan mover sus extremidades superiores comer con comodidad.

# PALABRAS CLAVE

Visión artificial, Internet de las cosas (IoT), Android, Kotlin, ML Kit, Face Mesh, MQTT, Bluetooth, ESP32, Robot OBI

# ABSTRACT

This project consists of the development of an application for Android devices which enables its users to control Smart devices through the performance of certain facial gestures.

The objective of the app is to help people with mobility issues reduce their dependece, allowing them to manage devices that they previously could not control without the help of a personal assistant.

This application is based on the detection of a series of facial landmarks using computer vision to calculate the distance between certain detected points. Once this distance is known, it is compared with a threshold, so if the distance is greater than that threshold, it is considered that the facial gesture has been made.

After detecting the performance of certain facial gestures, messages are sent to Smart devices for them to act accordingly and modify the user's environment in the way that the user intended, allowing them to control their surroundings simply with facial gestures.

Additionally, a feeding assintance robot will be adapted to be controlled through the messages sent from the application instead of using its original switches, allowing people who cannot move their upper limbs to eat comfortably.

# KEYWORDS

Computer Vision, Internet of Things (IoT), Android, Kotlin, ML Kit, Face Mesh, MQTT, Bluetooth, ESP32, OBI Robot

# Índice de Contenidos

# Índice de Figuras

# Índice de Tablas

# 1. INTRODUCCIÓN, MOTIVACIÓN Y OBJETIVOS

## 1.1 Introducción

En el presente proyecto se aborda el desarrollo de una aplicación destinada a dispositivos Android cuya función consiste en permitir al usuario controlar dispositivos inteligentes a través de la realización de ciertos gestos faciales. También se realizará la modificación del sistema de accionamiento de un robot de asistencia a la alimentación para poder ser controlado desde la aplicación desarrollada.

Los campos principales que involucra este proyecto son la programación de aplicaciones Android, la visión artificial y el Internet de las cosas (IoT). Todas estas áreas han visto grandes avances en los últimos años, y es que, con la proliferación de los dispositivos móviles, de dispositivos inteligentes y de la automatización en todo tipo de procesos, se ha transformado enormemente la manera con la que podemos interactuar con nuestro entorno.

Estos mismos avances han hecho posible el control remoto a través de aplicaciones móviles de elementos de domótica como luces, electrodomésticos, sistemas de calefacción y de ventilación o sistemas de seguridad, entre otros. Sin embargo, para personas que padecen problemas de movilidad muy severos, la accesibilidad a estas nuevas tecnologías sigue siendo un desafío considerable.

## 1.2 Motivación del proyecto

Las personas con movilidad reducida enfrentan dificultades en su vida cotidiana que afectan a su independencia y su calidad de vida. Para ellos, ciertas tareas diarias que podrían resultar de los más simples para la mayoría pueden llegar a ser sumamente difíciles de realizar sin asistencia, como encender o apagar las luces de su hogar, subir o bajar persianas o incluso llevarse una cuchara a la boca para poder comer. Así, surge la necesidad de soluciones tecnológicas, innovadoras y accesibles que permitan a estas personas controlar de forma

autónoma ciertos elementos de su entorno que no podrían manipular de forma convencional.

Para conocer más en detalle la realidad de esta situación, se han analizado tres encuestas sobre el número y situación de millones de discapacitados en España.

El Instituto Nacional de Estadística (INE) ha publicado una encuesta de discapacidad, autonomía personal y situaciones de dependencia en 2020 [1], en la que se refleja que hay 4,38 millones de personas en España con discapacidad. De entre ellas, la más común son problemas de movilidad, lo que también deriva en dificultades en la realización de tareas domésticas. Según los datos mostrados en la encuesta, los problemas de movilidad afectan a 54 de cada 1000 españoles **(figura 1)**, lo que supone un total de 2,44 millones de personas con discapacidad por problemas de movilidad en nuestro país.

**Tipo de discapacidad de personas de seis y más años por sexo**
Tasas por mil habitantes

| | Total | Hombres | Mujeres |
|---|---|---|---|
| Total | 97,0 | 81,2 | 112,0 |
| Movilidad | 54,0 | 38,9 | 68,5 |
| Vida doméstica | 45,1 | 31,8 | 57,8 |
| Autocuidado | 30,6 | 22,9 | 38,0 |
| Audición | 27,6 | 24,1 | 31,0 |
| Visión | 23,6 | 18,4 | 28,6 |
| Comunicación | 21,3 | 18,7 | 23,7 |
| Aprendizaje | 15,8 | 13,4 | 18,2 |
| Interacciones y relaciones personales | 13,6 | 13,5 | 13,8 |

Nota: Una misma persona puede estar en más de una categoría de discapacidad

*Figura 1. Datos sobre los tipos de discapacidad más comunes en España en 2020 (Fuente: [1])*

Consultando otra encuesta del INE publicada en 2023 [2], en este caso sobre discapacidades en residencias, se puede observar que Castilla y León es la comunidad en la que hay mayor tasa de personas con discapacidad en residencias **(figura 2)**.

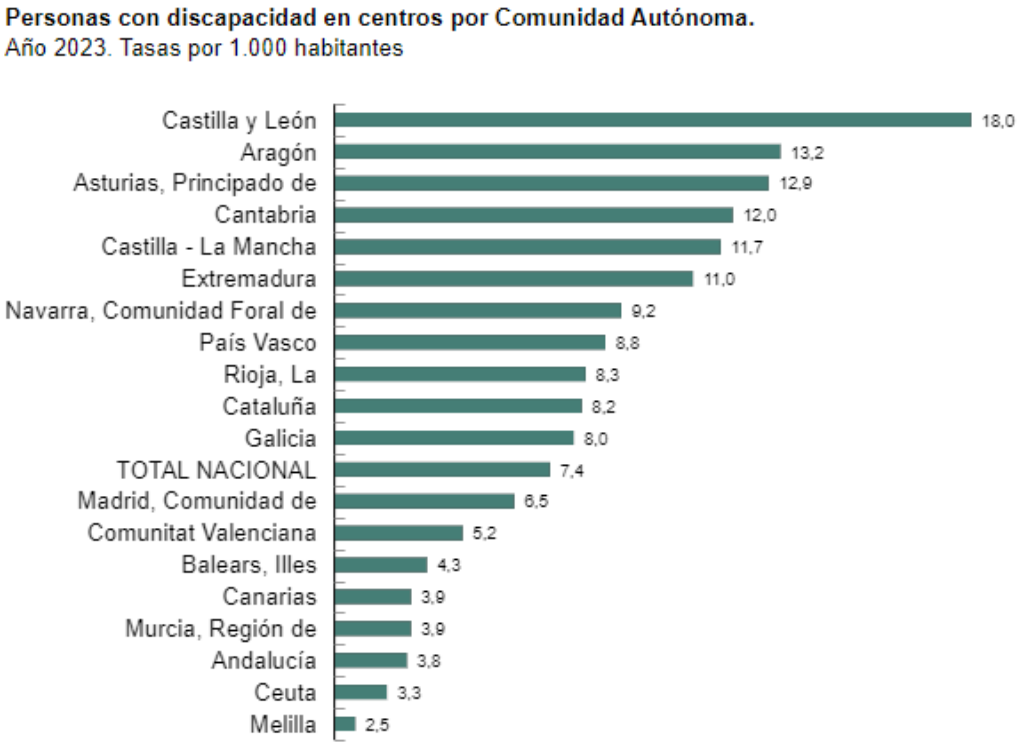



*Figura 2. Gráfico sobre personas con discapacidad en centros por comunidad (Fuente: [2])*

Esto refleja que nuestra comunidad es una de las más afectadas en lo que a personas con discapacidad y necesidad de asistencia se refiere, de modo que este proyecto podría ayudar a miles de habitantes de nuestra propia comunidad autónoma a sobrellevar las dificultades que enfrentan en la realización de tareas cotidianas.

Por si esto fuera poco, otra encuesta del INE sobre discapacidad, en este caso la del 2008 [3], muestra que el número de personas con discapacidad en 2008 era de 3,85 millones comparados a los 4,38 millones de 2020, lo que significa que ha aumentado en más de medio el número de personas con discapacidad en nuestro país en menos de 15 años, haciendo aún más necesario y urgente el desarrollo de proyectos que se basen en aumentar la accesibilidad de estas personas a las nuevas tecnologías para mejorar su calidad de vida y reducir su dependencia.

Según la Organización Mundial de la Salud (OMS) [4], el número de personas con discapacidad ha crecido en las últimas décadas y se espera que siga creciendo al aumentar la población mundial y al aumentar la esperanza de vida promedio. Esto implica que este proyecto podría impactar positivamente a la vida de aún más personas en un futuro.

## 1.3 Objetivos

El objetivo general de este proyecto es diseñar y desarrollar una aplicación que permita a usuarios con problemas de movilidad muy severos controlar su entorno de forma cómoda y eficiente, esto es, minimizando la realización de movimientos que les puedan suponer dolor o molestia y a la vez procurando que el control sea intuitivo y rápido.

Además, hay que tener en cuenta que cada caso médico difiere del resto, por lo que unos gestos sencillos de realizar pueden resultar problemáticos para otros usuarios. Por ello, también habrá que permitir cierta personalización del conjunto de movimientos a detectar y sus efectos en los dispositivos inteligentes que se estén controlando.

De este modo, podemos fijar una serie de subobjetivos a cumplir con el proyecto.

- Diseño de una interfaz sencilla e intuitiva que cumpla con las necesidades de usuarios con casos diversos de discapacidad motora

- Desarrollo de un sistema de detección de gestos faciales mediante el uso de herramientas de visión artificial que identifique e interprete correctamente los gestos faciales realizadas por el usuario

- Implementación de una representación visual que permita al usuario de la aplicación conocer que los gestos faciales han sido identificados correctamente

- Establecimiento de una conexión fiable entre el dispositivo Android en el que se ejecutará la aplicación y los diversos elementos inteligentes que se encuentren en el entorno a controlar

- Permitir al usuario elegir qué gesto realiza cada acción disponible para que el control sea tan cómodo y accesible como sea posible

- Optimización del código desarrollado para que su funcionamiento sea fluido y de bajo coste energético para la batería del dispositivo Android

- Validar el funcionamiento de la aplicación a través de experimentación, obteniendo comentarios de realimentación que permitan seguir mejorando y ampliando la aplicación

Estos objetivos para el proyecto tendrán también un impacto positivo en la sociedad de acuerdo con los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030. Este proyecto contribuye directamente a los ODS número 3 y 10:

“Salud y Bienestar” y “Reducción de las Desigualdades”, respectivamente. En el caso de Salud y Bienestar, el proyecto facilitará un entorno más adaptado y cómodo para personas con enfermedades y discapacidades severas, mientras que para el objetivo “Reducción de las Desigualdades”, se pretende mejorar la autonomía e independencia en la realización de tareas cotidianas de personas en riesgo de exclusión social como pueden ser personas con discapacidades o con problemas de movilidad graves, lo que es fundamental para promover la igualdad de oportunidades y su inclusión en la sociedad.

## 1.4 Estructura de la memoria

Una vez concluido este capítulo introductorio, y para documentar el proyecto de forma clara y organizada, se siguen un conjunto de capítulos diferenciados entre sí que aborden las diferentes partes del trabajo. A continuación, se describe la estructura de la memoria.

- Capítulo 2: El segundo capítulo tratará una serie de conceptos fundamentales para la comprensión del proyecto realizado. Se explicarán la definición y las bases conceptuales de la visión artificial, especialmente en el contexto del reconocimiento de gestos faciales. También se abordarán los diferentes tipos de enfermedades y discapacidades que sufren los usuarios objetivo de la aplicación para entender las necesidades específicas de accesibilidad de las que requiere la aplicación a desarrollar. Además, se introducirá el concepto de internet de las cosas y su aplicación en el control de entornos inteligentes, lo que incluye los protocolos de comunicación a emplear (MQTT y Bluetooth). También se estudiarán los robots de asistencia a la alimentación y los microcontroladores ESP32, al emplear uno de cada en la prueba del funcionamiento del protocolo Bluetooth programado. Finalmente, se tratará el desarrollo de aplicaciones para dispositivos Android empleando el entorno Android Studio y el lenguaje Kotlin.
- Capítulo 3: En el tercer capítulo se analizará el estado actual de la accesibilidad a dispositivos móviles, al control de entornos inteligentes y a la alimentación para personas con discapacidad. Así, se estudiarán diferentes aplicaciones y dispositivos que faciliten la interacción entre usuario y dispositivos, haciendo especial hincapié en aquellos que empleen visión artificial para conseguirlo.
- Capítulo 4: En el cuarto capítulo se recoge información sobre las herramientas y tecnologías utilizadas a lo largo del desarrollo del proyecto, tales como el entorno de desarrollo, el lenguaje de programación empleado o un programa de monitorización del servidor que permite la comunicación por protocolo MQTT.

- Capítulo 5: En el capítulo 5 se explicará en detalle el proceso de desarrollo del código que compone la aplicación Android final, abordando cada parte del código para ofrecer una explicación detalla del funcionamiento del programa. De esta forma, se explicará la implementación de la cámara del dispositivo, de las librerías ML Kit y Eclipse Paho, la determinación de gestos faciales a partir de la posición de los puntos faciales detectados y el manejo y establecimiento de la comunicación MQTT y de la comunicación Bluetooth.
- Capítulo 6: El sexto capítulo comprende la experimentación realizada para probar el funcionamiento de la aplicación y los resultados de dicha experimentación. Este proceso incluirá estudiar cuál es la localización óptima del dispositivo para la correcta detección de puntos faciales, probar el encendido y apagado de dispositivos inteligentes y documentar pruebas realizadas con usuarios. Por último, se discutirán los resultados obtenidos y se extraerán conclusiones sobre el funcionamiento final de la aplicación desarrollada.
- Capítulo 7: Se realizará el estudio económico del proyecto, analizando los recursos empleados, incluyendo los costes de personal, equipos y software.
- Capítulo 8: Resumen de las conclusiones del proyecto. Se destacarán los logros alcanzados y también las posibles limitaciones encontradas. Finalmente, se propondrán futuras líneas de trabajo para mejorar y perfeccionar la aplicación.
- Capítulo 9: El noveno capítulo incluirá las referencias bibliográficas estudiadas a lo largo del desarrollo del proyecto.
- Capítulo 10: Por último, el décimo capítulo comprenderá los anexos, que proporcionarán información adicional relevante, como materiales complementarios que faciliten la comprensión del proyecto.

# 2. CONCEPTOS CLAVE Y FUNDAMENTOS TEÓRICOS

El correcto desarrollo de una aplicación Android para el control de dispositivos inteligentes mediante gestos faciales, especialmente diseñada para personas con movilidad muy severa, requiere la comprensión de una serie de conceptos tecnológicos. Este capítulo proporciona una visión general de dichos conceptos clave.

## 2.1 Visión Artificial

### 2.1.1 Definición de Visión Artificial

La visión artificial hace referencia a un grupo de tecnologías que permiten adquirir, procesar, analizar y comprender imágenes del mundo real con el fin de producir información numérica o simbólica que pueda ser gestionada por un sistema informático. [5]

El funcionamiento de la visión artificial se logra a través del entrenamiento de máquinas para realizar el procesado y análisis de las imágenes. Este entrenamiento se basa en la tecnología *Machine Learning* (aprendizaje automático en español) y, más concretamente, en redes neuronales convolucionales (CNN). [6]

Machine Learning (ML) es el uso de modelos algorítmicos para que un ordenador identifique patrones en un conjunto de datos [7]. Esto también permite al ordenador realizar predicciones basándose en esos patrones detectados. Se distinguen 3 tipos de aprendizaje dentro del ML: supervisado (se entrena a la máquina con datos previamente etiquetados para que después pueda determinar cuándo una imagen tiene algo en común con las que se han usado en su entrenamiento), no supervisado (la máquina no es entrenada para detectar un tipo específico de dato, sino en agrupar ejemplos similares) y por refuerzo (la máquina aprende por prueba y error hasta encontrar la manera más adecuada de realizar una tarea).

Las CNNs son una arquitectura de red neuronal (modelo computacional cuya toma de decisiones está inspirada en el funcionamiento del cerebro humano) que se agrupa dentro del aprendizaje automático y que se diferencian de otras redes neuronales en que están optimizadas para realizar tareas de visión artificial. Una CNN asigna etiquetas a conjuntos cercanos de píxeles de una imagen y después

realiza predicciones basadas en cálculos matriciales usando esas etiquetas. Así, con un amplio conjunto de datos etiquetados se logra que un ordenador pueda analizar las imágenes por sí mismo, en lugar de tener un programa por cada una de las imágenes a analizar. [8]

También cabe aclarar que el correcto funcionamiento de un sistema de visión artificial no solo depende del algoritmo de detección, sino también de una adecuada iluminación, de la disposición del objeto a detectar respecto a la cámara para que se sitúe dentro del rango de detección, de un procesador suficientemente potente para analizar las imágenes obtenidas y de las conexiones que existan entre el sistema de visión y los sistemas a controlar. [9]

El área en el que es más común el uso de sistemas de visión artificial es el industrial, ya que permiten monitorear estaciones de trabajo automatizadas, controlar inventarios, determinar parámetros de calidad (tales como la presencia de desperfectos, residuos o contaminantes), o participar en la producción de piezas muy complejas y precisas. Sin embargo, también son empleados en un sinfín de campos muy diversos, como en la conducción autónoma de vehículos, en sistemas de videovigilancia y seguridad, en el ámbito médico para ayudar en el diagnóstico de enfermedades, clasificación de objetos **(figura 3)**, etc.


*Figura 3. Detección y clasificación de objetos mediante visión artificial (Fuente: [10])*

En este trabajo, se profundizará en la visión artificial aplicada al reconocimiento de gestos faciales, al ser el método que usará el usuario de la aplicación a desarrollar para enviar órdenes a dispositivos inteligentes.

### 2.1.2 Visión Artificial aplicada al reconocimiento de gestos faciales

El reconocimiento de gestos faciales es una subárea de la visión artificial que se encarga de identificar y analizar movimientos y expresiones del rostro humano. La detección de gestos como tal se consigue mediante la localización de puntos clave (*landmarks*) de la cara. Una vez determinada la posición de dichos puntos, se extraen características geométricas de los mismos, como distancias, ángulos y proporciones entre ellos. Estas características geométricas se comparan con umbrales para determinar si se ha realizado un gesto facial. Es entonces cuando la ocurrencia de uno de esos gestos se puede asignar a algún comando específico con el que controlar dispositivos, lo que es crucial en este proyecto al permitir al usuario la interacción con su entorno sin necesidad de movimiento físico.

La detección de puntos clave del rostro ha evolucionado significativamente a lo largo de las últimas décadas, impulsada por avances en el aprendizaje automático.

Los primeros esfuerzos en la detección de landmarks faciales se remontan a la década de 1970. Por ejemplo, en 1973 Takeo Kanade desarrolló un sistema de detección de más de 30 puntos faciales para la identificación de individuos [11] mediante la detección de bordes y contornos de una imagen binarizada.

En la década de los 90s se comenzó a utilizar algoritmos basados en plantillas, como el ASM(*Active Shape Model*) o el AAM(*Active Appearance Model*) [12], ambos métodos funcionan ajustando iterativamente un modelo hasta minimizar la diferencia entre los puntos del modelo y los puntos de la cara. También en los 90s empezó el desarrollo de sistemas de detección de rostros mediante redes neuronales convolucionales [13], si bien el análisis de puntos clave de la cara se daría a lo largo de la década de los 2000 [14].

A lo largo de la década de 2010, se mejoró enormemente la detección por CNNs al desarrollarse redes profundas (caracterizadas por un mayor número de capas), también llamadas *Deep Convolutional Neural Networks* (DCNN), que alcanzaron una precisión y robustez sin precedentes [15].

Por último, recientes mejoras de eficiencia en redes neuronales convolucionales han permitido llevar la detección de puntos faciales a dispositivos móviles [16].

Esta aplicación de la visión artificial es empleada en múltiples campos, como la atención al cliente y publicidad (por ejemplo, interpretando las emociones y reacciones de un cliente al trato que ha recibido o a la publicidad que le ha sido mostrada), la seguridad (al identificar a sospechosos y facilitar la verificación de la identidad) [17], el estudio de la comunicación verbal (para determinar las palabras dichas por alguien al estudiar el movimiento de sus labios) [18], la salud (monitoreando el estado emocional de pacientes, detectando signos tempranos

de enfermedades, tales como ictus o accidente cardiovascular, para su correcto diagnóstico, o analizando el resultado de operaciones en la cara [19]), o la interacción hombre-máquina.

El reconocimiento de gestos faciales mediante visión artificial enfrenta ciertos desafíos, como la gran variedad de expresiones faciales que puede realizar el rostro humano, las oclusiones parciales de la cara en ciertas condiciones o la variabilidad en las condiciones de iluminación. Sin embargo, es un área capaz de transformar la forma en la que interactuamos con el mundo a nuestro alrededor, y a medida que la tecnología detrás de este reconocimiento continúe su avance, como mejorando el rendimiento de las CNNs [20], se mejorará su precisión y robustez y se empleará en aplicaciones aún más innovadoras y beneficiosas para la sociedad.

A día de hoy, existen numerosas librerías que permiten la detección de landmarks faciales de forma rápida y sencilla de implementar, como dlib [21] **(figura 4)**, Mediapipe o ML Kit.

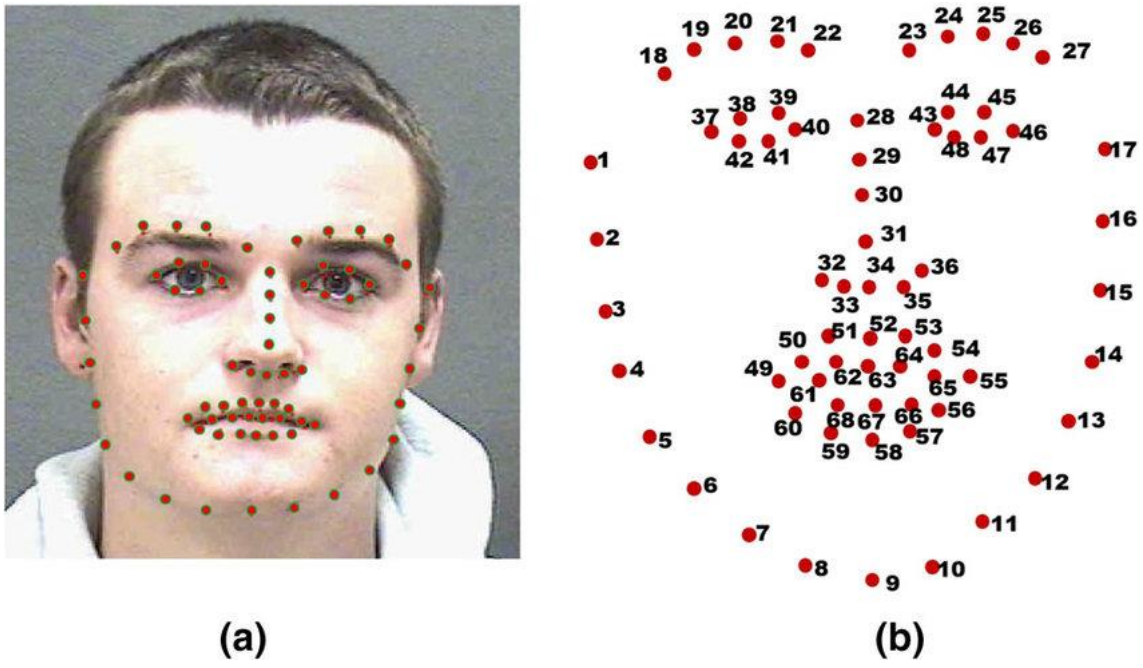



*Figura 4. Identificación de puntos del rostro usando Dlib (a) y posición y numeración de los puntos detectados (b) (Fuente: [22])*

El desarrollo de un sistema de visión artificial para el reconocimiento de gestos faciales [23] implica varias etapas. A continuación, se detalla un enfoque estructurado para dicho proceso.

1) Adquisición y preparación de datos

   Se usarán bases de datos que contengan imágenes etiquetadas con puntos clave del rostro. En caso de no disponer de suficientes datos, se puede recurrir a la anotación manual para etiquetar las imágenes.

2) Procesamiento

   Se estandariza el tamaño y resolución de las imágenes para garantizar la coherencia. Tras ello, se pueden aplicar rotaciones, traslaciones y escalados a las imágenes para disponer así de un conjunto con mayor variabilidad, lo que dará robustez al sistema de detección al permitirlo detectar landmarks incluso al tener la cabeza inclinada o parcialmente oculta.

3) Selección y entrenamiento del modelo

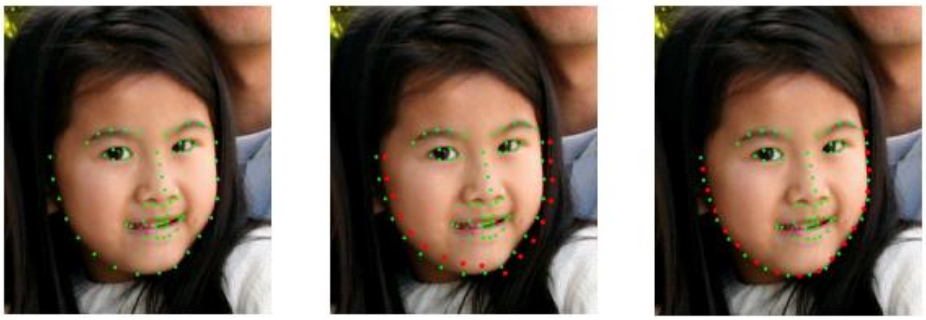
   Existen diferentes arquitecturas de redes neuronales convolucionales que están pensadas para la detección de puntos del rostro. Tras elegir el modelo, se divide el conjunto de datos en subconjuntos de entrenamiento y validación.

4) Implementación

   El primer paso de la implementación consiste en la detección del rostro en la imagen, ya sea mediante Haar Cascade (método clásico y rápido) o por CNNs (método más preciso) [24]. Una vez detectada la cara, se aplica detección de landmarks por el CNN escogido.

5) Evaluación

   Es posible cuantificar la precisión de los landmarks predichos calculando la distancia de error entre el punto predicho y el anotado inicialmente **(figura 5)**. También es posible determinar la robustez del sistema bajo diferentes condiciones de iluminación, diferentes expresiones faciales y diferentes posiciones de la cara.



Ground truth  Error in normal direction  Error in tangent direction

*Figura 5. Error entre los puntos predichos y los anotados en detección de landmarks faciales (Fuente: [25])*

6) Optimización

   Experimentar con diferentes configuraciones de hiperparámetros (parámetros ajustables que permiten controlar el proceso de entrenamiento del modelo, como número de capas y número de nodos [26]) hasta ajustarlos de forma que mejore el rendimiento del modelo.

7) Despliegue

   Integrar el sistema en aplicaciones finales, como sistemas de reconocimiento facial, de análisis de emociones, o interfaces de usuario basadas en gestos faciales.

## 2.2 Discapacidades y Enfermedades: Beneficiarios de la Aplicación

Existe una gran variedad de discapacidades y enfermedades cuyos pacientes podrían beneficiarse significativamente de la aplicación que se desarrolla en este trabajo, ya que al permitir el control de dispositivos inteligentes mediante gestos faciales se evita la realización de acciones físicas que podrían resultar molestas, dolorosas o incluso imposibles de realizar para ciertas personas.

La aplicación busca ayudar a personas que no puedan realizar acciones físicas básicas en sus extremidades superiores y que padezcan problemas del habla (ya que en caso de poder hablar sin dificultad ya existen asistentes virtuales que reconocen órdenes por voz y pueden actuar cumpliendo esos comandos, como Alexa, Siri o Google Assistant).

Así, este proyecto puede resultar en una mejora en la calidad de vida e independencia diaria de personas con movilidad reducida y problemas para hablar.

### 2.2.1 Tipos de discapacidades y enfermedades objetivo

Los pacientes de discapacidades motoras (especialmente las que causan tetraplejia y paraplejia superior) podrían verse beneficiados por el uso de la aplicación, al pretender ésta eliminar la necesidad de movimiento.

Las discapacidades motoras afectan a la capacidad de una persona para moverse y realizar acciones físicas. Pueden ser resultado de diversas

condiciones médicas como parálisis cerebral, esclerosis lateral amiotrófica, esclerosis múltiple, distrofia muscular, lesiones de médula espinal u otras enfermedades neurodegenerativas.

Parálisis cerebral [27]: Es un grupo de trastornos neurológicos permanentes que afectan al desarrollo del movimiento y la postura, causando limitación de la actividad, que se atribuyen a perturbaciones no progresivas que ocurrieron en el desarrollo del cerebro fetal o infantil. Estos trastornos dificultan el movimiento corporal y la coordinación muscular. Los pacientes de esta dolencia pueden experimentar una amplia gama de problemas motores, desde debilidad muscular en una extremidad hasta problemas severos que afecten a todos los movimientos del cuerpo.

Esclerosis Lateral Amiotrófica (ELA) [28]: La ELA es una enfermedad neurodegenerativa progresiva que afecta a las neuronas motoras del cerebro y la médula espinal. Los pacientes de ELA experimentan una pérdida gradual de la función muscular, lo que eventualmente conduce a la parálisis del cuerpo.

Esclerosis Múltiple [29]: Es una enfermedad crónica que afecta al sistema nervioso central, específicamente al cerebro y a la médula espinal. Los síntomas pueden variar ampliamente, incluyendo problemas de coordinación, debilidad, fatiga y dificultades en la movilidad. En etapas avanzadas, también produce trastornos del habla.

Distrofia Muscular [30]: Las distrofias musculares son un grupo de enfermedades genéticas que causan debilidad muscular progresiva y pérdida de masa muscular. Con el paso del tiempo, estas condiciones pueden llevar a una discapacidad severa.

Lesiones de médula espinal [31]: Las lesiones medulares, causadas por traumas como accidentes automovilísticos, caídas o lesiones deportivas, pueden resultar en una pérdida parcial o total de la función sensorial y motora por debajo del nivel de la lesión. La gravedad de la discapacidad depende de la ubicación y severidad de la lesión.

Otras enfermedades neurodegenerativas: Enfermedades como el Parkinson [32] o la atrofia muscular espinal [33] también pueden llevar a una pérdida significativa de la movilidad y de la capacidad para hablar.

También debe considerarse otros casos médicos, como personas que hayan sufrido amputaciones en ambas extremidades superiores y que sufran de disartria (dificultad para mover o controlar los músculos que se usan para hablar).

### 2.2.2 Medidas para facilitar la accesibilidad en la aplicación

Teniendo en cuenta la cantidad y variedad de enfermedades y discapacidades cuyos pacientes se verían beneficiados por la aplicación Android a desarrollar, ésta debería incorporar una serie de medidas de accesibilidad que garanticen su usabilidad y utilidad para todos los usuarios, independientemente de sus limitaciones físicas [34]. A continuación, se describen las principales medidas que deben implementarse para asegurar su accesibilidad y funcionalidad óptima.

- Interfaz de usuario (UI) intuitiva y configurable: Una interfaz que sea fácilmente comprensible por el usuario y que pueda ser personalizada por él mismo es esencial para que se pueda navegar y utilizar la aplicación con facilidad. Esto incluye un diseño simple y claro, emplear colores con alto contraste para diferenciar las distintas partes de la interfaz, el uso de tipografía legible y medidas para permitir a los usuarios personalizar la disposición de los elementos de la UI para adaptarse a sus necesidades específicas.
- Reconocimiento de puntos faciales avanzado y preciso: El reconocimiento facial que realice la aplicación debe ser preciso y confiable para permitir un control eficiente de los dispositivos inteligentes. Se debe poder detectar y distinguir gestos faciales aún en condiciones de iluminación variables y también se debe contemplar la personalización de gestos para adaptar el uso de la aplicación a las capacidades individuales de cada usuario.
- Dar información al usuario en tiempo real: Para que los usuarios sepan qué gestos han sido reconocidos y qué acciones se están realizando, se debe proporcionar información en tiempo real al usuario, tal como señales visuales fácilmente comprensibles.

## 2.3 Internet de las Cosas (IoT) y entornos inteligentes

### 2.3.1 Conceptos básicos

El internet de las cosas (IoT por sus siglas en inglés) se refiere a la interconexión de dispositivos y objetos a través de una red (ya sea Internet o una red privada), lo que permite a estos dispositivos generar, enviar y recibir datos [35, 36]. Esta tecnología está revolucionando numerosos sectores, desde la domótica hasta la salud, proporcionando una mayor capacidad de control y automatización de tareas. Se estima que existen en torno a 17000 millones de dispositivos conectados a Internet en 2024 [37, 38] y según esas mismas estimaciones, se

espera que a finales de esta década haya cerca de 30000 millones, más de 3 veces la población mundial.

Los dispositivos IoT suelen estar equipados con sensores, actuadores y software que les permitan interactuar y comunicarse con otros dispositivos y sistemas. Estos dispositivos recopilan datos del entorno y, a través de Internet, envían esa información a otros dispositivos y servidores para su procesamiento y análisis.

Los elementos clave que suelen componer un ecosistema IoT **(figura 6)** incluyen:

- Sensores: recopilan datos del entorno, como humedad, temperatura, movimiento o presión.

- Actuadores: realizan acciones físicas en respuesta a comandos (como encender luces o ajustar un termostato).

- Conectividad: los dispositivos IoT usan diversas tecnologías de comunicación como Bluetooth, Wi-Fi, Zigbee, LoRaWan, NB-IoT, etc [39].

- Plataformas de IoT: gestionan los datos recopilados por los dispositivos IoT y proporcionan interfaces para que los usuarios puedan interactuar con el sistema. Ejemplos: AWS IoT, Google Cloud IoT y Microsoft Azure IoT.

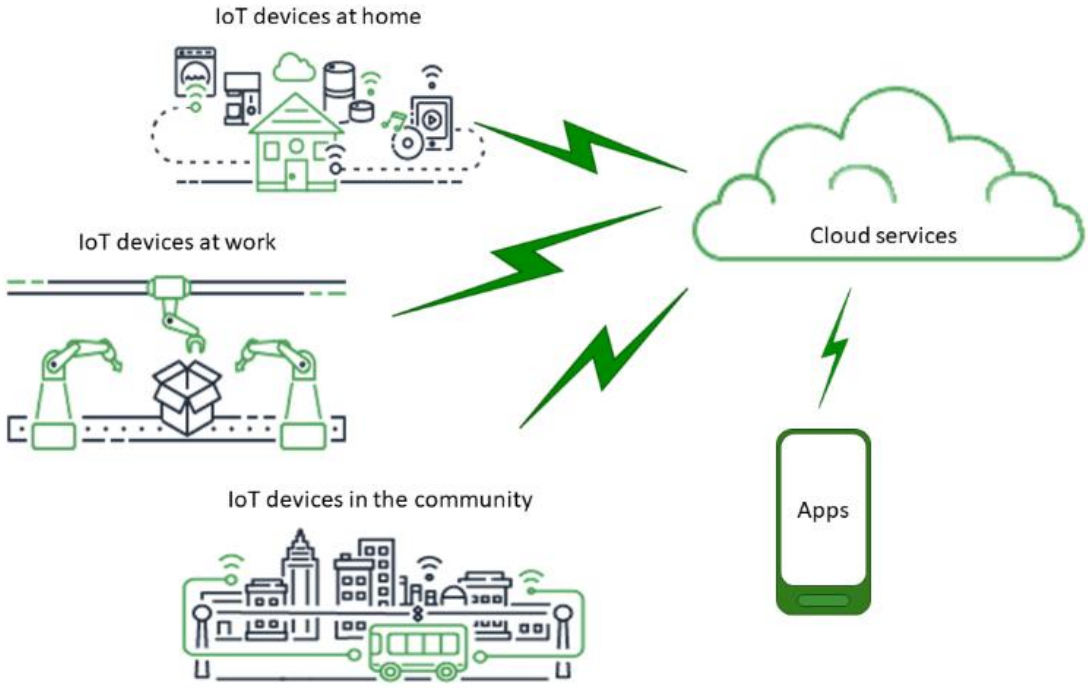



*Figura 6. Representación de comunicación entre aplicación y dispositivos IoT (Fuente: [40])*

### 2.3.2 Control de entornos mediante IoT

El control de entornos mediante IoT implica la utilización de dispositivos conectados para gestionar y automatizar diferentes aspectos del entorno físico, como iluminación, temperatura, seguridad o entretenimiento. Esto lo hace perfecto para el presente proyecto, ya que permite a personas con problemas de movilidad interactuar con su entorno de manera más sencilla y efectiva.

Así, podremos controlar los elementos IoT instalados en el hogar o alrededores del usuario a través de comandos enviados desde la aplicación. El proceso que seguirá la ejecución de la aplicación se puede desglosar en los pasos mostrados a continuación.

1. Detección de landmarks: la aplicación Android usará la cámara del dispositivo móvil para captar imágenes en tiempo real del usuario y reconocerá los puntos clave de su cara.

2. Interpretación de los gestos detectados: en función de la distancia entre los puntos detectados, se determinará si se ha realizado un gesto.

3. Envío de comandos en respuesta a los gestos: al detectarse un gesto específico, se envían comandos a los dispositivos IoT correspondientes a través de protocolo MQTT o Bluetooth.

4. Ejecución de acciones por parte de los dispositivos IoT: los dispositivos IoT reciben los comandos enviados desde la aplicación y ejecutan la orden dada, como encender una luz o levantar una persiana.

### 2.3.3 Ejemplos en el mundo real

El uso de IoT para el control de entornos tiene múltiples aplicaciones prácticas en el mundo real, algunas de las cuales incluyen:

- Hogares inteligentes **(figura 7)**: múltiples empresas han desarrollado sistemas de iluminación, termostatos, cámaras de seguridad, enchufes, sistemas de ventilación y hasta mirillas inteligentes que pueden ser controlados a distancia a través de aplicaciones móviles o asistentes de voz. De esta forma se pueden programar y automatizar tareas como el encendido y apagado de luces, el ajuste de temperatura del hogar, la monitorización de las cámaras de seguridad o hasta el encendido de dispositivos convencionales que estén enchufados a enchufes inteligentes.

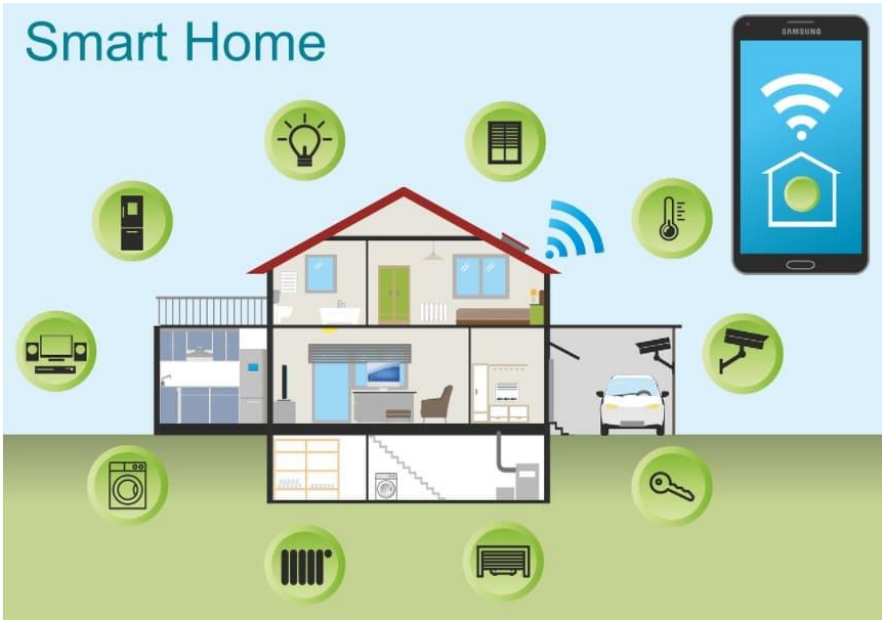



*Figura 7. Representación de Smart home con elementos IoT controlados desde un móvil (Fuente: [41])*

- Salud y bienestar: Dispositivos como pulseras de actividad y relojes inteligentes son capaces de recopilar datos sobre la salud del usuario, como frecuencia cardíaca, patrones de sueño, registro de actividad física o medición de oxígeno en sangre [42].

- Ciudades inteligentes: Varias ciudades están implementando sensores IoT para monitorizar y gestionar infraestructuras urbanas como el tráfico, alumbrado público y la calidad del aire. Estos sistemas pueden mejorar la eficiencia energética, reducir la congestión de tráfico y aumentar la calidad de vida de sus ciudadanos [43].

Sus aplicaciones en el mundo real demuestran que IoT ofrece un vasto potencial para mejorar la calidad de vida de las personas, especialmente aquellas con limitaciones de movilidad. La integración de IoT con aplicaciones móviles que utilizan gestos faciales para enviar órdenes proporciona una solución innovadora y accesible para el control de entornos, permitiendo a los usuarios interactuar con su entorno de manera más autónoma y cómoda.

## 2.4 Protocolos de Comunicación: MQTT y Bluetooth

La comunicación entre la aplicación y los dispositivos a controlar se dará a través de uno de los dos protocolos siguientes: MQTT o Bluetooth. Por ello, se explica

el funcionamiento de cada uno de ellos y el por qué son adecuados para nuestro caso.

### 2.4.1 Funcionamiento de los protocolos

#### *2.4.1.1 Funcionamiento de MQTT*

MQTT (Message Queuing Telemetry Transport) es un protocolo de mensajería ligera diseñado para ser utilizado en situaciones donde el ancho de banda es limitado y la confiabilidad de la red es baja. Es ideal para aplicaciones IoT debido a su eficiencia y bajo consumo de energía. MQTT sigue un modelo de comunicación publish/suscribe (publicación/suscripción), en la que los dispositivos pueden publicar mensajes en temas específicos o suscribirse a esos temas para recibir mensajes [44]. Esto significa que todos los dispositivos suscritos a un mismo tópico reciben el mismo mensaje, proveniente del dispositivo que publique en ese tópico concreto.

Se suele comparar la forma en la que funciona la publicación y suscripción MQTT con el funcionamiento de un periódico. Al igual que un periodista desconoce la cantidad e identidad de los lectores del artículo que publique, también lo desconoce el dispositivo que publique un mensaje en un tópico.

Los elementos básicos de la comunicación MQTT son:

- Broker: Es el servidor central que recibe todos los mensajes de los clientes y los distribuye a los suscriptores correspondientes.

- Cliente: Puede ser cualquier dispositivo que publique mensajes o que se suscriba a tópicos para recibir mensajes. El funcionamiento publish/suscribe funciona de modo que un cliente no necesita conocer la existencia de otros clientes.

- Tópicos o Temas (Topics): Son categorías o etiquetas a las que los clientes se suscriben para recibir mensajes.

- Mensajes: Contienen el contenido que se envía a través de los tópicos. Los mensajes MQTT se caracterizan por ser ligeros, teniendo una cabecera de mensaje de tamaño fijo 2 bytes y mensaje de como máximo 256MB, y por tener diferentes niveles de calidad de servicio (QoS).

Existen 3 niveles de calidad de servicio, lo que permite a los diseñadores de la red decidir entre maximizar la transmisión de datos o maximizar fiabilidad [45, 46].

- QoS0 (Como máximo una vez) **(figura 8)**: Cada mensaje publicado se envía a un suscriptor una única vez, sin confirmación de que lo haya

recibido. En cuanto un mensaje ha sido enviado, este nivel de QoS asume que la entrega se ha completado. Esto implica que el servidor no almacena el mensaje en ningún momento.


*Quality of Service level 0: delivery at most once*

*Figura 8. Esquema del funcionamiento del QoS0 (Fuente: [45])*

- QoS1 (Al menos una vez) **(figura 9)**: Al intentar entregar un mensaje a un suscriptor, se espera una confirmación por parte del mismo, y en caso de no recibirla dentro de un intervalo de tiempo determinado, se envía el mensaje de nuevo. Como resultado, un receptor puede recibir y procesar varias veces el mismo mensaje. El mensaje es almacenado en el servidor hasta que se reciba la confirmación de entrega, tras ello, se elimina del servidor.


*Quality of Service level 1: delivery at least once*

*Figura 9. Esquema del funcionamiento del QoS1 (Fuente: [45])*

- QoS2 (Exactamente una vez) **(figura 10)**: Este nivel de calidad del servicio se encarga de que cada mensaje se reciba exactamente una sola vez por cada suscriptor del tópico mediante un protocolo de 4 pasos. Al igual que en el nivel 1, al haber enviado un mensaje se espera a que se reciba una confirmación de entrega, y si no se recibe, se reenvía el mensaje. Cuando el emisor reciba la confirmación de entrega, éste comunicará al receptor que puede comenzar el procesado del mensaje que ha recibido. Una vez el emisor recibe la confirmación de procesado por parte del receptor, se elimina el mensaje almacenado.

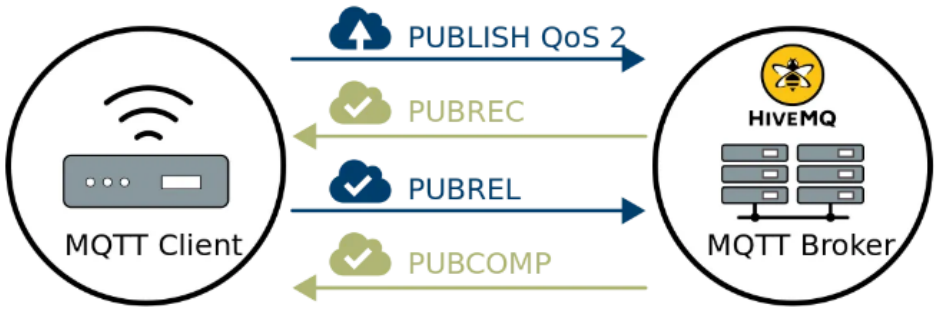



*Figura 10. Esquema del funcionamiento del QoS2 (Fuente: [45])*

#### 2.4.1.2 Funcionamiento de Bluetooth

Bluetooth es una tecnología de comunicación inalámbrica de corto alcance, diseñada para reemplazar los cables entre dispositivos electrónicos. Utiliza ondas de radio en la banda ISM de 2.4 GHz para intercambiar datos entre dispositivos. Este rango de frecuencias proporciona hasta 79 canales distintos que los dispositivos pueden usar para comunicarse. La forma en la que los dispositivos que se están comunicando emplean estos canales es saltando de uno a otro buscando constantemente la mayor calidad de señal posible, es decir, la menor interferencia. Además de la versión clásica de Bluetooth, también existe Bluetooth Low Energy (BLE), que se caracteriza por su menor gasto energético, aunque dispone de menos canales [47].

Una ventaja de Bluetooth respecto a otras tecnologías como Wi-Fi es que no requiere de la instalación de infraestructura como routers o APs (Access Points), ya que cada dispositivo se conecta directamente a otro.

Para poder comunicarse a través de Bluetooth, los dispositivos que formarán parte de esa conexión primero deben emparejarse, lo que incluye autenticación e intercambio de claves para garantizar una comunicación segura [48].

Si bien el rango efectivo de Bluetooth se sitúa en torno a los 10 metros y se puede ver afectado por obstáculos como paredes o muebles, sigue siendo una buena opción para el control de dispositivos en un entorno cercano. Tanto es así que, a día de hoy, Bluetooth es utilizado en una amplia variedad de aplicaciones y productos, incluyendo teléfonos, ratones, teclados, y hasta en el ámbito médico [49]. Otros usos incluyen:

- Audio inalámbrico: Los auriculares y altavoces Bluetooth han reemplazado en gran medida a sus contrapartes con cables, ofreciendo mayor movilidad.
- Dispositivos médicos: Monitores de frecuencia cardíaca, glucómetros y otros dispositivos de salud pueden utilizar Bluetooth para transmitir datos a móviles y ordenadores [50].
- Automóviles: Los sistemas de manos libres y la transmisión de música desde teléfonos móviles a sistemas de audio de automóviles son algunas de las aplicaciones más comunes de Bluetooth.
- Domótica: En el ámbito del hogar inteligente, Bluetooth se usa para controlar luces, cerraduras, termostatos y otros dispositivos.

### 2.4.2 Ventajas del uso de MQTT y Bluetooth en este proyecto

#### *2.4.2.1 Ventajas de usar MQTT*

A continuación, se enumeran algunas de las ventajas de la utilización de la comunicación MQTT en la aplicación Android a desarrollar [51].

1. Eficiencia en el ancho de banda: MQTT es ideal para situaciones en las que el ancho de banda es limitado, lo que es crucial para dispositivos IoT que pueden operar en redes móviles o conexiones de baja velocidad. Esto asegura que la aplicación pueda funcionar de forma fluida incluso con restricciones de red.
2. Bajo consumo de energía: La naturaleza ligera del protocolo MQTT permite que los dispositivos mantengan un bajo consumo de energía, lo que es beneficioso para la longevidad de dispositivos móviles y sensores IoT que se utilicen en el proyecto.
3. La existencia de los niveles de calidad del servicio (QoS): Permiten decidir qué tan fiable queremos que sea la comunicación MQTT.
4. Infraestructura: Si bien la comunicación MQTT requiere de un bróker que gestione la publicación de mensajes en tópicos, no es necesario ninguna conexión física entre dispositivos que requiera de elementos como cables, lo que facilita la instalación en el hogar del usuario.

5. Fiabilidad: Gracias a su arquitectura basada en el uso de un bróker, MQTT proporciona un método robusto para manejar comunicaciones entre múltiples dispositivos. Esto es crucial para procurar que los comandos gestuales lleguen consistentemente a los dispositivos controlados.

También se debe tener considerar que MQTT cuenta con algunos inconvenientes, como tener que gestionar la conexión dentro de la aplicación Android mediante la especificación por parte del usuario de la dirección del servidor MQTT, el puerto, usuario y contraseña para poder comunicarnos correctamente con el bróker.

#### *2.4.2.2 Ventajas de usar Bluetooth*

Al igual que con MQTT, son varios los motivos para considerar el uso de Bluetooth como protocolo de comunicación en la aplicación [52].

1. Conexión rápida y estable: Este aspecto será útil para el control en tiempo real de los dispositivos inteligentes, ya que proporcionará una respuesta rápida a los gestos faciales realizados.
2. Infraestructura: Tampoco requiere de ningún cable que haya que instalar para transmitir la información, abaratando y simplificando la puesta en marcha de la instalación.
3. Comunicación inalámbrica: Aunque el rango de acción de Bluetooth sea de unos pocos metros, se pueden controlar elementos inteligentes situados en habitaciones contiguas o en otras plantas, al poder comunicarse incluso con paredes de por medio.
4. Coste y popularidad: Los dispositivos con conectividad Bluetooth llevan años dominando el mercado, lo que hace que haya una gran variedad de ellos y que su precio no sea muy alto. De igual forma, muchos dispositivos Bluetooth se caracterizan por tener dimensiones pequeñas, lo que evita tener que ocupar mucho espacio del hogar del usuario.

Por todos estos motivos, el uso de Bluetooth es beneficioso para el proyecto, pero cuenta con las desventajas de que tiene menor velocidad de transmisión de datos que otras tecnologías como Wi-Fi, no es recomendable para la transmisión de grandes cantidades de información, un dispositivo Bluetooth solo se puede conectar con otros 7 dispositivos como máximo y en caso de querer transmitir entre dos dispositivos muy alejados se pueden enfrentar problemas de conexión.

En conclusión, tanto MQTT como Bluetooth ofrecen ventajas significativas para el desarrollo de la aplicación de control de entornos inteligentes mediante gestos faciales. Mientras que MQTT asegura una comunicación eficiente y fiable a

través de diferentes condiciones de red, Bluetooth proporciona una conexión de baja latencia y segura para el control directo de dispositivos cercanos. La combinación de ambos protocolos permite desarrollar una solución robusta y versátil que responde a las necesidades específicas de los usuarios con problemas de movilidad.

## 2.5 Robots de alimentación asistida

Para mostrar la variedad de dispositivos controlables desde la aplicación, para probar la comunicación Bluetooth implementada y para permitir a los usuarios objetivo de la aplicación mayor comodidad e independencia a la hora de comer, se controlará un robot de asistencia en la alimentación OBI [53].

Por ello, en este apartado se explicará la funcionalidad general de este tipo de dispositivos y, posteriormente, se presentará el caso específico del robot de alimentación asistida OBI.

### 2.5.1 Funcionalidad de los Robots de Asistencia a la Alimentación

Los robots de asistencia a la alimentación o robots de alimentación asistida están diseñados para ayudar a personas con discapacidades motoras severas a alimentarse de forma autónoma. Debido a diversas condiciones médicas como las estudiadas en el apartado de discapacidades y enfermedades, hay personas que pueden tener dificultades a la hora de manejar utensilios de comida, por lo que encuentran en este tipo de dispositivos una solución que mejora su independencia y su calidad de vida [54].

Estos robots están diseñados teniendo en cuenta la situación médica de sus usuarios, por lo que tienen una serie de funcionalidades y propiedades características.

- Manipulación de utensilios **(figura 11)**: Los robots de asistencia a la alimentación consisten en un brazo robótico que se encarga de sostener y manejar un utensilio de comida como un tenedor o una cuchara, llevando los alimentos desde el plato hasta la boca del usuario.

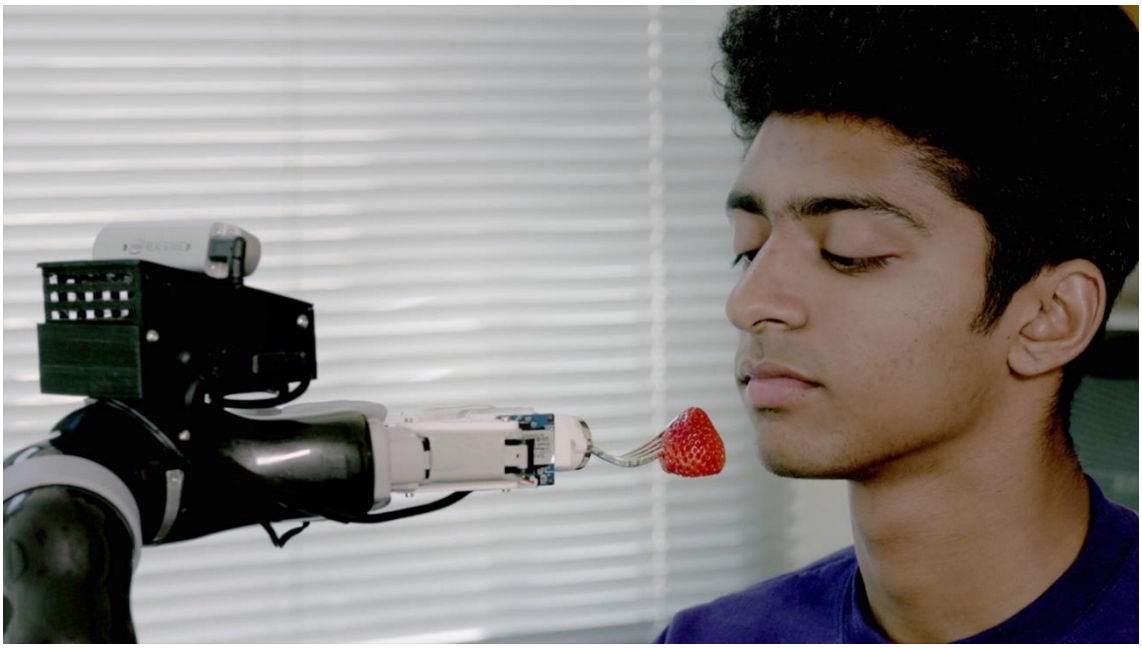


*Figura 11. Ejemplo de robot de asistencia a la alimentación con tenedor (Fuente: [55])*

- Seguridad y precisión en sus movimientos: El hecho de que sean robots destinados a estar situados en las inmediaciones de personas conlleva que su diseño se realice con la seguridad de los usuarios como prioridad. Así, muchos de estos robots cuentan con mecanismos para evitar golpear a quien los use, por ejemplo, deteniéndose en caso de que uno de los motores del brazo robótico detecte una carga demasiado elevada, lo que significaría que está colisionando con algo o alguien [56].

- Facilidad de uso: La mayoría de los robots de alimentación asistida están ideados para ser fáciles y cómodos de usar. Esto incluye la posibilidad de elegir la posición final del robot y accionamientos sencillos como interruptores en el caso del OBI o sistemas de visión artificial que acerquen la cuchara a la posición exacta de la boca del usuario en cada instante [57].

Debido a todas estas funcionalidades, estos robots tienen un impacto significativo en la mejora de la calidad de vida de personas con discapacidades motoras. Otorgan una mayor autonomía y comodidad al usuario por poder realizar una actividad tan fundamental como comer de forma independiente.

### 2.5.2 Robot OBI

El robot OBI **(figura 12)**, al igual que el resto de robots de alimentación asistida, se basa en un brazo robótico que sujeta un utensilio para comer, en su caso, una cuchara.

A diferencia de otros robots, tiene incorporada una base en la que se pueden colocar hasta 4 platos diferentes, para que el usuario pueda elegir qué comer en cada instante. Esto ofrece una aún mayor autonomía al usuario, ya que no debe preocuparse de tener algún asistente que esté cambiando su plato constantemente.

*Figura 12. Robot OBI (Fuente: [53])*

Tiene un bajo peso y un diseño compacto y portátil, lo que facilita su transporte y uso en diferentes entornos, ya sea en una casa, en un hospital o en una residencia. Esto permite a los usuarios llevar consigo al robot y mantener su autonomía a la hora de comer en diversos contextos.

El brazo robótico OBI lleva la cuchara hasta una posición establecida previamente por el usuario o por su asistente pulsando el botón situado en el lateral del brazo y llevando el extremo del brazo robótico hasta la posición deseada. Esto significa que puede ser configurado para acercar la cuchara hasta una posición situada delante de la boca del usuario, lo que haría que los movimientos del cuello del usuario no debieran ser muy bruscos o dolorosos.

Su encendido se realiza al pulsar el botón situado junto a la base del brazo, y cuenta con 2 acciones básicas, cambiar el plato del que la cuchara situada en el extremo del brazo cogerá la comida y la acción de acercar la comida a la boca con la cuchara. Estas 2 acciones se activan pulsando unos interruptores

conectados al robot a través de las conexiones Jack situadas en la parte inferior del lateral de la base **(figura 13)**.

*Figura 13. Interruptores que activan las entradas Jack del OBI (Fuente: [58])*

Estos interruptores deben ser pulsados por el usuario para que el OBI realice las acciones deseadas, por lo que personas con la imposibilidad de mover sus brazos no podrán usar el robot de esta forma.

Es por este motivo que parte del proyecto estará destinado a sustituir dichos pulsadores por una solución basada en la aplicación desarrollada que permita a un mayor rango de personas emplear el robot OBI para mejorar su calidad de vida a la hora de alimentarse, pudiendo controlar el robot mediante gestos faciales de la misma forma que se controlaría el entorno del usuario.

La sustitución de los interruptores manuales por un control desde la aplicación requerirá el uso de algún microcontrolador que pueda recibir datos desde el dispositivo Android para activar alguno de sus pines digitales como respuesta a los gestos faciales realizados por el usuario. Por este motivo, se estudiará a continuación las bases de los microcontroladores y se profundizará en el escogido para llevar a cabo esta tarea.

## 2.6 Microcontroladores

### 2.6.1 Definición y funciones

Un microcontrolador **(figura 14) (figura 15)** es un circuito electrónico que contiene procesador, memoria y periféricos de entrada/salida programables [59]. Se diferencia de un microprocesador en los elementos que componen a cada uno, ya que un microprocesador es solamente una CPU (Central Processing Unit), que consiste en una ALU (unidad aritmético-lógica), registros y una unidad de control, mientras que un microcontrolador contiene de por sí a un microprocesador, además de la memoria en la que almacenar programas, datos y configuraciones, y a los puertos de entrada/salida [60]. Esto hace que a efectos prácticos se traten de ordenadores en un único circuito integrado.


*Figura 14. Microcontrolador ATMEGA328P (Fuente: [61])*

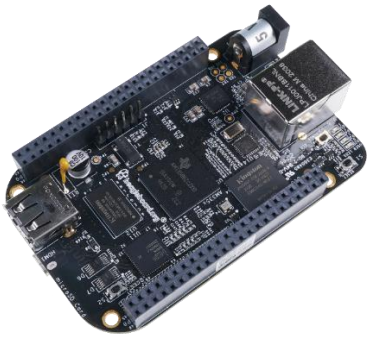


*Figura 15. Microcontrolador BeagleBone Black (Fuente: [62])*

Los microcontroladores se utilizan en una amplia variedad de campos e industrias, como en automoción, robótica, ofimática, dispositivos médicos, domótica, etc. Sus principales funciones consisten en la automatización, control y gestión de procesos, como controlar la inyección de combustible en un motor de combustión en el caso de la automoción o monitorizar y controlar sistemas domésticos como luces, persianas o termostatos en el caso de la domótica.

Existen microcontroladores dispuestos en un único encapsulado (también llamados MCU, Microcontroller Unit), como el ATMEGA visto en la figura 14, pero también existen microcontroladores SoC (System on Chip) [63,64], que están formados por una placa con los elementos que lo conforman dispuestos sobre ella **(figura 16)**, como el BeagleBone mostrado en la figura 15 o la serie de microcontroladores ESP32 [65].

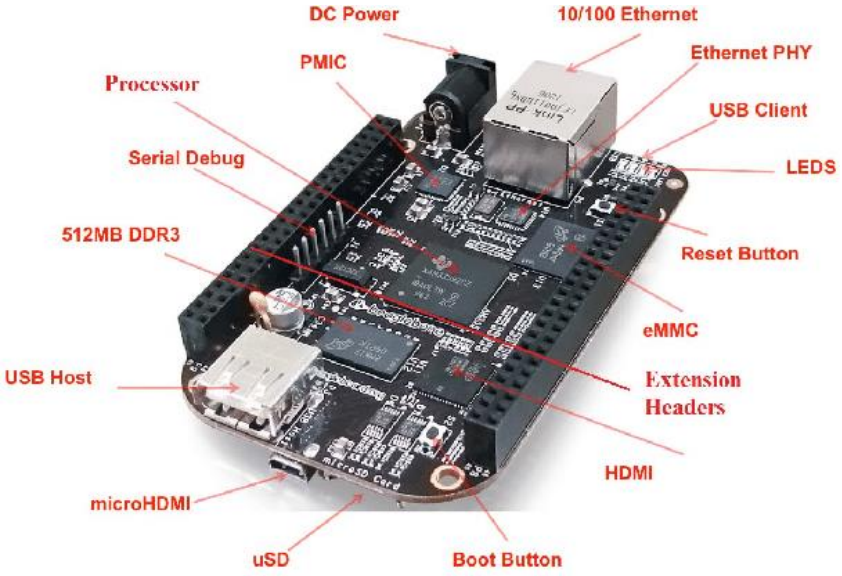



*Figura 16. Partes del microcontrolador SoC BleagleBone Black (Fuente: [66])*

Esta diferenciación es muy importante, ya que un MCU en un único encapsulado solo puede proporcionar periféricos muy específicos, mientras que un microcontrolador SoC puede tener implementados periféricos que le permitan comunicarse vía Wi-Fi o Bluetooth, lo que es de sumo interés para este proyecto al permitir comunicar la aplicación con el microcontrolador que sustituirá a los pulsadores del robot OBI por Bluetooth.

Además de permitir comunicación Bluetooth, las diferencias entre microcontrolador encapsulado MCU y microcontrolador sistema en chip SoC [67] se abordan en la **tabla 1**.

| Tipo de microcontrolador | MCU | SoC |
|---|---|---|
| Memoria | Memoria muy pequeña, suele ser de unos pocos KB | El diseño en placa permite incluir mucha más memoria, puede |

| | | |
|---|---|---|
| | | ser de unos cuantos MB o incluso GB |
| Periféricos | Numero pequeño de periféricos no específicos y más limitados | Mayor número de periféricos y más específicos |
| Sistema operativo | Sin sistema operativo | Puede incorporar sistema operativo, como Linux en BeagleBone |
| Uso | Aplicaciones de control de baja complejidad | Aplicaciones con mayor número de requisitos y mayor complejidad |

*Tabla 1. Diferencias entre microcontroladores MCU y SoC*

Con la finalidad de no confundir conceptos, cabe explicar que existen también placas de desarrollo (development board) basadas en microcontroladores [68], las cuales están pensadas principalmente para simplificar el aprendizaje del desarrollo y programación de proyectos electrónicos, uno de los ejemplos más famosos es el Arduino UNO **(figura 17)**, basado en el microcontrolador ATMEGA328P.

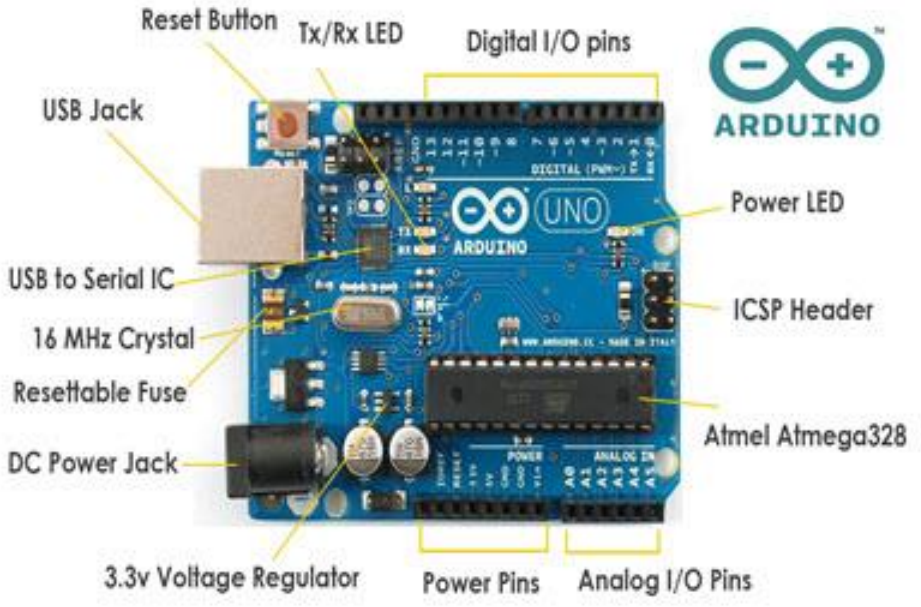



*Figura 17. Placa Arduino UNO basada en microcontrolador ATMEGA 328P (Fuente: [69])*

Estas placas de desarrollo difieren de los microcontroladores SoC en el hecho de que las placas Arduino incluyen un microcontrolador MCU, mientras que los microcontroladores SoC son microcontroladores como tal, incluyen un microprocesador, memoria y periféricos, solo que dispuestos sobre una placa en lugar de en un solo encapsulado.

### 2.6.2 ESP32

ESP32 **(figura 18)** es una familia de microcontroladores SoC desarrollados por Espressif Systems. Se basan en la familia de microcontroladores ESP8266 desarrollados por la misma empresa, con el añadido de más y mejores características, como la inclusión de comunicación Wi-Fi ,Bluetooth y BLE (Bluetooth Low Energy) [70], lo que los hace ideales para aplicaciones de IoT.

*Figura 18. ESP32 (Fuente: [71])*

Se caracterizan por su bajo consumo energético, bajo coste, ser ligero y por tener una cantidad de memoria y capacidad de cálculo considerables. Su conectividad Wi-Fi permite crear un servidor o una red para que otros dispositivos puedan conectarse a ella, y su capacidad Bluetooth también hace posible la comunicación inalámbrica en distancias cortas con otros dispositivos.

Su microprocesador es un Tensilica Xtensa LX6 de dos núcleos con frecuencia de 240MHz, cuenta con 34 pines de entrada/salida programables, convertidores analógico-digitales (ADC) y digitales-analógicos (DAC), LED de indicación de alimentación y un puerto Micro USB para poder alimentar y cargar los programas en la placa [72].

Todas estas características le otorgan robustez, versatilidad y eficiencia en una gran variedad de aplicaciones, y su bajo consumo hace que sea muy utilizado en proyectos de Internet de las Cosas aplicados a un sinfín de ámbitos, como

automatización de procesos industriales, dispositivos médicos, robótica, electrónica de consumo o domótica.

Su programación se puede adaptar a diversos proyectos y situaciones al permitir ser programada empleando Arduino C/C++, Espressif IDF o Micropython [73].

Todos estos motivos son la razón por la que será utilizado en este proyecto, al permitir la transmisión de comandos desde la aplicación hasta el robot mediante su conectividad Bluetooth, y por garantizar una rápida y precisa ejecución de las órdenes recibidas gracias a sus capacidades de procesamiento, lo que es esencial para el adecuado control del robot.

## 2.7 Desarrollo de aplicaciones Android

El desarrollo de aplicaciones Android implica el uso de una variedad de herramientas y tecnologías para crear aplicaciones móviles funcionales y eficientes. En este apartado, se exploran las bases fundamentales para el desarrollo de aplicaciones Android, así como la implementación de librerías específicas para el reconocimiento de puntos faciales y la comunicación mediante MQTT y Bluetooth.

### 2.7.1 Definición de Android

El primer concepto que es necesario comprender para poder profundizar en el desarrollo de aplicaciones Android es el de Android como tal.

Android es un sistema operativo orientado a dispositivos móviles como teléfonos, tablets o relojes inteligentes que está basado en el núcleo Linux [74, 75]. Es uno de los sistemas operativos más populares del mundo, gracias en gran medida a que permite programar e incluir el sistema operativo en cualquier dispositivo sin tener que pagar por hacerlo. Esto ha llevado a muchos fabricantes a emplearlo en sus productos, ya que reduce el coste de producción y desarrollo de los dispositivos que comercializan. Además, su arquitectura es modular y configurable, lo que permite a fabricantes modificar el sistema operativo para adaptarlo a su hardware.

Su popularidad y el bajo coste de algunos de los dispositivos que implementan Android hacen que sea el sistema operativo idóneo en el que desarrollar la aplicación, al permitir a muchos usuarios potenciales descargar la aplicación sin

tener que adquirir un nuevo dispositivo, y, en caso de tener que adquirirlo, podría resultar más barato que comprar móviles o tablets con otro sistema operativo.

Esta versatilidad lo ha hecho común entre tipos de dispositivos muy diversos, como televisores inteligentes, relojes inteligentes, interfaces en automóviles, tablets, telefónos móviles y dispositivos IoT [76].

Su primera versión fue lanzada en 2008 y, desde entonces, ha recibido múltiples mejoras y actualizaciones, siendo la versión actual Android 14.

### 2.7.2 Elementos clave para el desarrollo de aplicaciones Android

Una vez entendido qué es Android, cabe recordar que para desarrollar aplicaciones destinadas a dispositivos Android, se requiere de un entorno de desarrollo integrado (IDE) adecuado y un conocimiento de los lenguajes de programación compatibles. Las herramientas más comunes incluyen:

- Android Studio [77]: Es el IDE oficial para el desarrollo de aplicaciones Android. Proporciona un entorno completo para escribir, depurar, probar y empaquetar aplicaciones Android. Android Studio está basado en IntelliJ IDEA y ofrece un editor de código avanzado, emuladores de dispositivos Android, herramientas de depuración, y soporte para el diseño de interfaces gráficas mediante XML.
- Lenguaje de programación Kotlin [78, 79]: Tanto Kotlin como Java pueden ser empleados en el desarrollo de aplicaciones Android, pero Kotlin es el lenguaje que está adquiriendo mayor relevancia debido a su compatibilidad con Java y a ser más conciso y seguro. Esto significa que Kotlin permite escribir código más limpio y menos propenso a errores, lo que facilita el desarrollo y mantenimiento de aplicaciones Android.
- SDK de Android [80]: El Software Development Kit (SDK) de Android proporciona las bibliotecas y herramientas necesarias para desarrollar aplicaciones para el sistema operativo Android. Incluye APIs para acceder a funcionalidades del dispositivo como la cámara, el GPS, y sensores incorporados.
- Gradle [81]: Gradle es un sistema de compilación para Android que permite automatizar y facilitar el proceso de construcción de aplicaciones.
- Android Jetpack [82]: Es un conjunto de componentes, herramientas y guías diseñadas para facilitar el desarrollo de aplicaciones Android. Incluye librerías como “activity”, “databinding”, “fragment” o “navigation” (las cuales se volverán a tratar en el apartado de desarrollo de la aplicación), que ayudan a crear aplicaciones más robustas y eficientes.

### 2.7.3 Librerías de reconocimiento de puntos faciales para Android

Para implementar el control de dispositivos mediante gestos faciales, es necesario integrar librerías de reconocimiento facial que puedan identificar y localizar puntos específicos en el rostro del usuario. Entre las librerías más utilizadas se encuentran:

- Google ML Kit [83, 84]: ML Kit de Google proporciona una API de reconocimiento facial que permite detectar y rastrear puntos clave del rostro en tiempo real. Esta API es fácil de integrar y gracias a los puntos detectados se puede determinar la apertura o cierre de ojos, la realización de sonrisas o de muchos otros gestos faciales, lo que es crucial para establecer controles con dichos gestos. Devuelve una malla conformada por 468 landmarks de la cara **(figura 19)**.

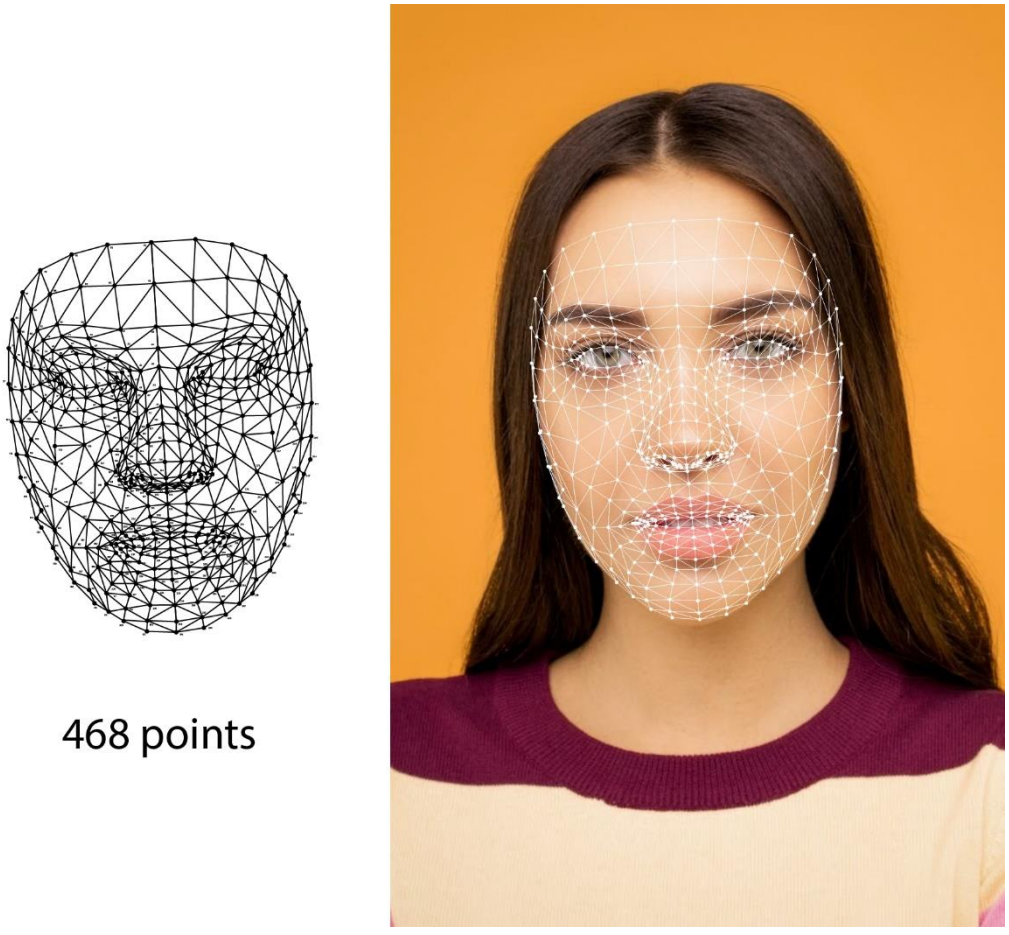



*Figura 19. Malla devuelta por la API de detección de malla facial de ML Kit (Fuente: [84])*

- OpenCV [85]: OpenCV (Open Source Computer Vision Library) es una biblioteca de visión artificial de código abierto que incluye herramientas para el reconocimiento facial. Aunque puede requerir de más configuración que ML Kit, OpenCV es altamente personalizable y es potente, ya que permite el procesamiento de imágenes y la implementación de algoritmos de reconocimiento facial avanzados.
- Mediapipe [86]: Mediapipe es una solución multiplataforma desarrollada por Google que cuenta con múltiples aplicaciones de visión artificial, entre ellas una especializada en la detección de una malla de puntos faciales. Dicha malla está conformada por 478 puntos **(figura 20)**, los mismos 468 que ML Kit más 4 puntos de cada iris y las 2 pupilas.

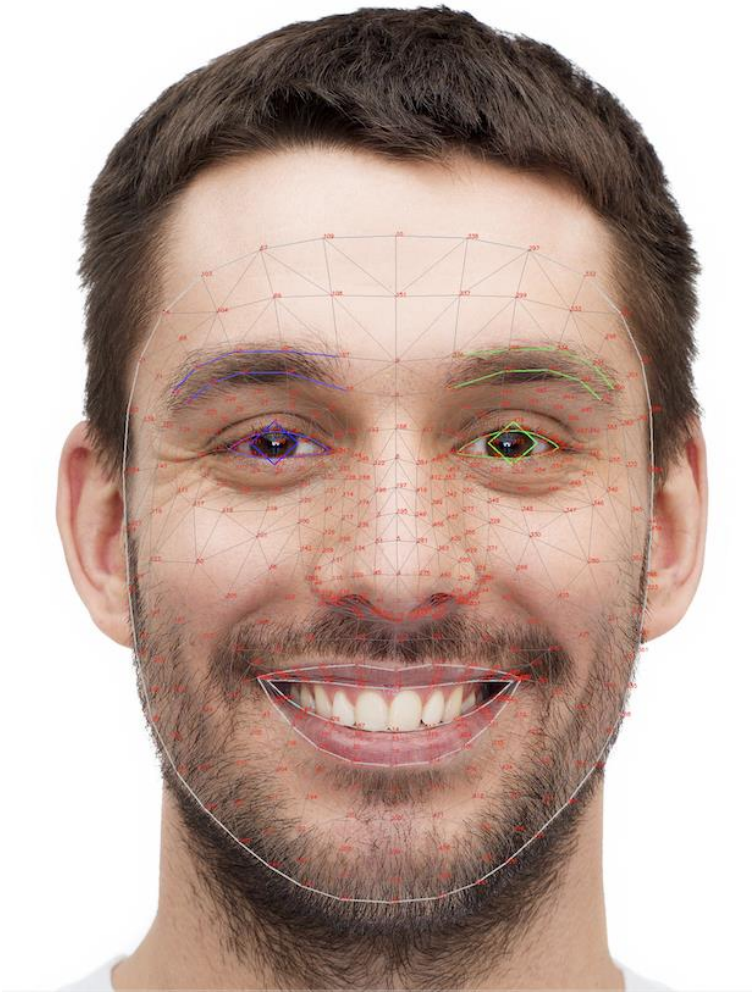


*Figura 20. Malla devuelta por la API de malla facial de Mediapipe (Fuente: [86])*

- Dlib [22, 87, 88]: Dlib es una librería de aprendizaje automático y de herramientas de visión artificial que incluye una implementación eficiente del algoritmo de detección de puntos faciales. Dlib es capaz de rastrear 68 puntos del rostro.

### 2.7.4 Librerías de comunicación MQTT y Bluetooth para Android

Para controlar dispositivos inteligentes, es crucial establecer una comunicación eficiente entre la aplicación Android y los dispositivos. Dos tecnologías comunes para esto son MQTT y Bluetooth.

MQTT: Para implementar MQTT en una aplicación Android, se pueden utilizar librerías como Paho MQTT de Eclipse [89], que ofrece un cliente MQTT robusto y fácil de integrar. MQTT permite la comunicación a través de un modelo de publicación-suscripción, lo que facilita el control de múltiples dispositivos mediante un solo emisor del mensaje.

Bluetooth: Android proporciona APIs integradas para gestionar conexiones Bluetooth [90], permitiendo el escaneo de dispositivos, la conexión y el intercambio de datos. Esto facilita la implementación de la comunicación Bluetooth en la aplicación al no requerir el uso de librerías externas.

# 3. ESTADO DEL DESARROLLO DE APLICACIONES DE CONTROL DE ENTORNOS Y DE ASISTENCIA A LA ALIMENTACIÓN

A lo largo de este capítulo analizaremos diferentes aplicaciones de concepto similar a la que se desarrolla en el presente trabajo y estudios que traten la detección específica de gestos faciales o el control de entornos inteligentes mediante acciones detectadas por sistemas de visión artificial.

## 3.1 Aplicaciones Similares en el Mercado

El desarrollo de aplicaciones que permiten el control de dispositivos inteligentes mediante el reconocimiento de gestos faciales no es un campo en el que haya una gran inversión económica ni un gran número de proyectos en desarrollo por el reducido número de usuarios potenciales, pero sí existen unos pocos proyectos prometedores, motivados en gran parte por la búsqueda de soluciones inclusivas para personas con discapacidades. A continuación, se muestran algunas de las aplicaciones y proyectos más relevantes en esta área.

### 3.1.1 EVA Facial Mouse

Esta aplicación permite controlar un dispositivo Android mediante movimientos de cabeza, los cuales son capturados por la cámara frontal del dispositivo controlado.

EVA Facial Mouse [91, 92] está especialmente ideado para personas con limitaciones motoras en sus extremidades superiores, ya que permite el acceso a la mayoría de las funcionalidades del dispositivo sin necesidad de usar las manos. Su funcionamiento se basa en un ratón que se desplaza por la pantalla siguiendo los movimientos de cabeza realizados por el usuario **(figura 21)**, y, al detener la cabeza durante unos segundos, se realiza la acción de hacer clic en el lugar hasta el que se haya llevado el ratón. Esto significa que el uso de la aplicación requiere del movimiento del cuello para poder manejar correctamente un móvil o Tablet, lo que hace que personas con parálisis en el cuello no puedan realizar el control deseado. Además, la aplicación está pensada para el control

de un dispositivo Android, pero no para el control directo de entornos inteligentes. Para esta tarea, serían necesarias aplicaciones externas instaladas en el móvil en el que se esté usando EVA Facial Mouse.

*Figura 21. Usuario controlando una Tablet mediante EVA Facial Mouse (Fuente: [93])*

### 3.1.2 Switch Access

Switch Access [94] es una aplicación desarrollada por Google e ideada para facilitar a personas con problemas de movilidad en sus muñecas y manos el control de sus dispositivos Android. Funciona colocando un interruptor externo que el usuario pulsará para realizar las acciones que deseé realizar **(figura 22)**. Un recuadro marca acciones en la pantalla de forma cíclica y, al pulsar el interruptor, se ejecuta la acción resaltada en ese instante. Esta aplicación requiere del movimiento de al menos un brazo para poder usarse, por lo que no está orientada al mismo perfil de discapacidad que la aplicación que se pretende desarrollar.


*Figura 22. Representación del uso de un móvil mediante los interruptores requeridos por Switch Access (Fuente: [95])*

### 3.1.3 Voice Access

Aunque se basa en el control mediante la voz, Voice Access [96] es una herramienta de accesibilidad de Android desarrollada por Google que permite a los usuarios con discapacidades motoras interactuar con sus dispositivos Android de forma más cómoda, ya que la aplicación permite realizar acciones como desplazarse, seleccionar y editar texto mediante comandos de voz **(figura 23)**.

Esta aplicación solo puede ser usada por personas con capacidad del habla, por lo que tampoco está orientada al mismo tipo de usuario que la aplicación desarrollada en este trabajo.

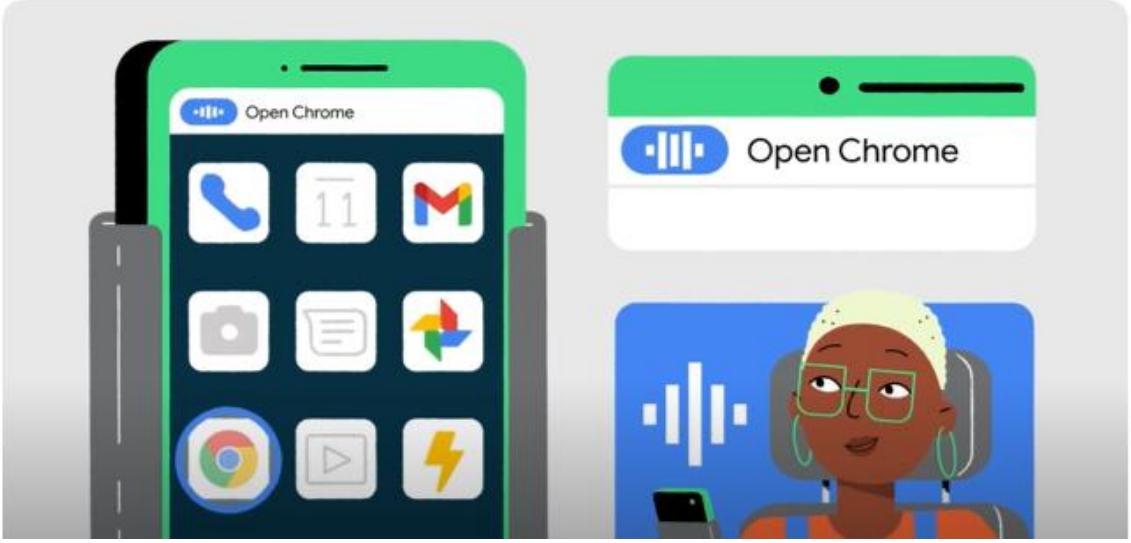



*Figura 23. Control de dispositivo mediante comandos de voz con Voice Access (Fuente: [97])*

### 3.1.4 Project Activate

Project Activate [98] también ha sido desarrollada por Google, y en este caso sí está pensada para personas con problemas de movilidad y de habla simultáneos. Permite realizar llamadas, decir una frase, reproducir audios y enviar mensajes de texto con la realización de gestos faciales **(figura 24)**. También permite al usuario elegir los gestos a realizar y ajustar la sensibilidad para que el uso de la aplicación resulte cómodo y fiable.

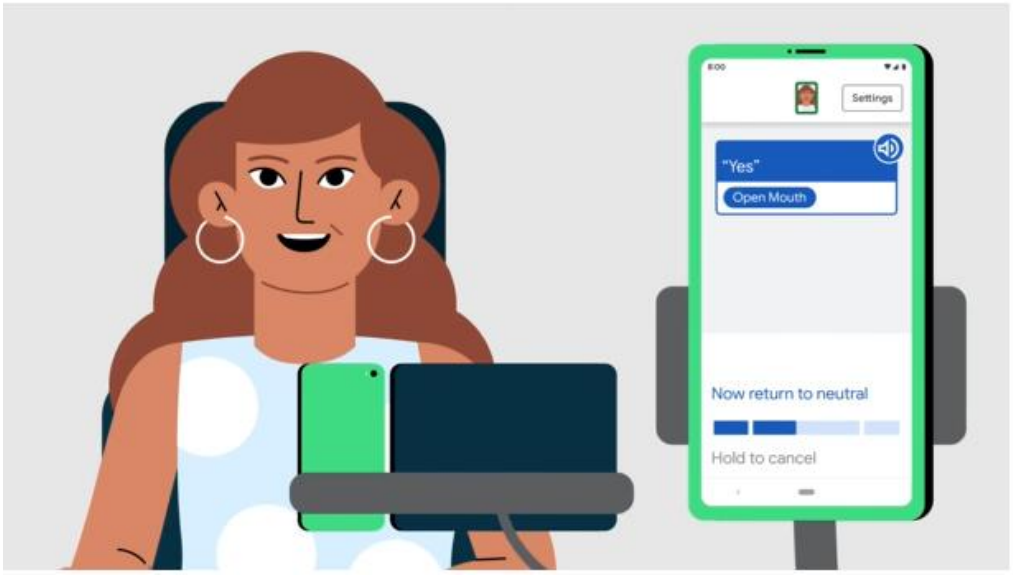



*Figura 24. Envío de mensaje de texto mediante apertura de la boca con Project Activate (Fuente: [99])*

El concepto base de esta aplicación es similar a la que se desarrollará, pero se diferencia en que está pensada para la realización de acciones básicas de mensajería móvil, como la reproducción de mensajes de audio o el envío de mensajes de texto, pero no permite controlar entornos inteligentes. Es decir, está diseñada para poder ser usada por usuarios con las mismas discapacidades que nuestra aplicación, pero no para realizar las mismas acciones.

### 3.1.5 Dwell Click

Dwell Click [100] permite controlar un dispositivo Android mediante elementos externos como ratones, joysticks o un dispositivo de seguimiento de la cabeza. En caso de usar tales dispositivos, la aplicación funcionaría de modo que el usuario mueve su cabeza hasta la posición en la que quiere clicar y se mantiene inmóvil durante unos segundos para realizar la acción de clicar como tal **(figura 25)**. Esto significa que sigue requiriendo el movimiento de la cabeza, una acción que la aplicación que se desarrollará pretende evitar, y también puede provocar un click indeseado si mantenemos la cabeza en reposo. Por tanto, la navegación por las interfaces del dispositivo puede resultar compleja, no se ajusta a las mismas necesidades de accesibilidad que este trabajo y requiere de la instalación de aplicaciones externas para poder controlar entornos inteligentes.


*Figura 25. Interfaz de Dwell Click mostrando mediante la intersección de líneas el lugar en el que se clicará al mantener la cabeza en reposo (Fuente: [100])*

### 3.1.6 Jabberwocky

Jabberwocky [101] es una aplicación compatible con Android que detecta movimientos de la cabeza y gestos faciales para interactuar con el dispositivo. Está diseñada para que la apertura y cierre de la boca equivalga a un click, de modo que una apertura y cierre rápidos suponen una acción equivalente a tocar la pantalla, mientras que abrir la boca, mover la cabeza y cerrarla en otro punto equivale a arrastrar el dedo por la pantalla **(figura 26)**. Esto permite interactuar con el dispositivo de forma tan completa como se haría con las manos, pero difiere de nuestro objetivo por tener que mover el cuello y por no estar enfocada en el control de entornos inteligentes.


*Figura 26. Uso de Jabberwocky para interactuar con un móvil Android (Fuente: [101])*

### 3.1.7 Open Sesame

Open Sesame [102], al igual que Jabberwocky, permite a las personas controlar sus dispositivos móviles mediante movimientos de cabeza y gestos faciales. Open Sesame requiere de activación por voz y utiliza la cámara frontal del dispositivo para rastrear los movimientos de la cabeza y los gestos faciales, proporcionando una navegación por el móvil adaptada para usuarios con movilidad reducida **(figura 27)**. Aunque esta aplicación permite el uso de todas las aplicaciones de un dispositivo móvil, sigue requiriendo del uso de voz y del movimiento de la cabeza del usuario, lo que excluye a personas con problemas del habla y de movilidad en el cuello.

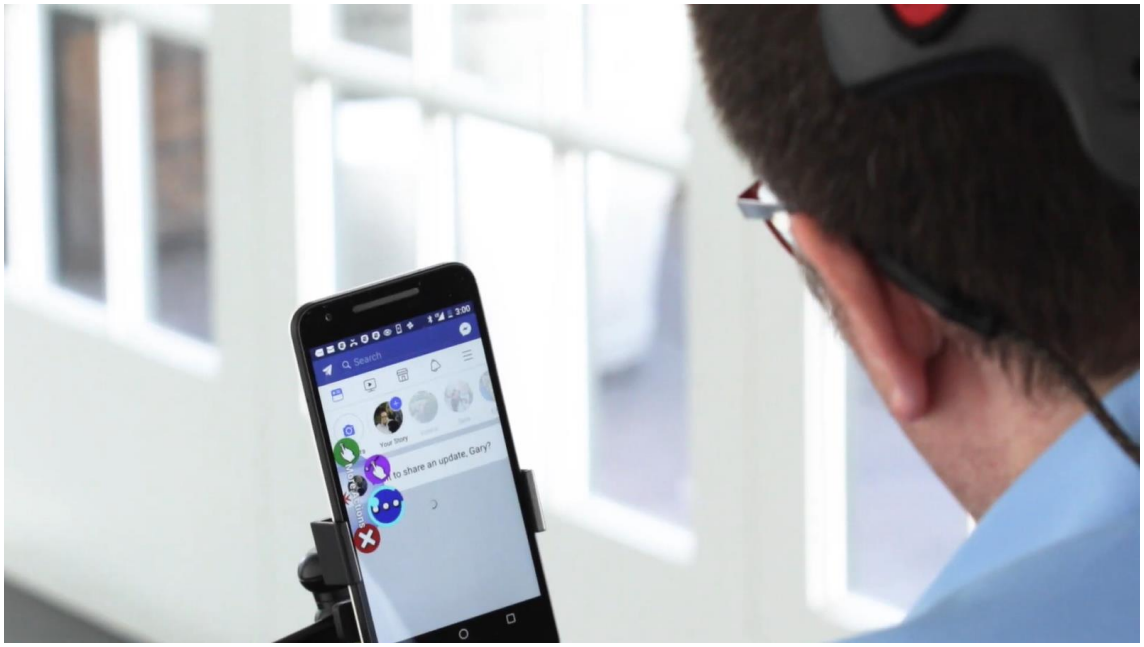


*Figura 27. Uso de Open Sesame para navegar por un teléfono móvil (Fuente: [103])*

### 3.1.8 Conclusión de analizar apps similares

Analizando las aplicaciones de accesibilidad para personas con problemas de movilidad y habla que se han abordado en este punto, se puede llegar a la conclusión de que si bien ya existen aplicaciones en el mercado orientadas a permitir controlar dispositivos móviles para personas con problemas de movilidad muy severos, solo Project Activate contempla la imposibilidad de mover el cuello, pero está destinada a mensajería móvil, no al control específico de dispositivos del entorno del usuario.

Por ello, nuestra aplicación atiende a una necesidad de accesibilidad no contemplada por ninguna otra alternativa en el mercado a día de hoy y se centra en darle la oportunidad al usuario del control de una serie de dispositivos específicos que ninguna otra aplicación considera.

## 3.2 Robots de Asistencia a la Alimentación Similares en el Mercado

En este punto se analizarán robots de alimentación asistida similares al robot OBI que se empleará en el proyecto para comprender el estado actual del desarrollo de dichos robots.

### 3.2.1 MySpoon

MySpoon [104, 105, 106] **(figura 28)** es un robot de alimentación asistida desarrollado por la empresa japonesa SECOM, que, al igual que el resto de robots de asistencia a la alimentación, consiste en un barco robótico que pretende ayudar a personas que no pueden comer ni beber sin asistencia externa.

*Figura 28. Robot MySpoon (Fuente: [104])*

Al igual que OBI, transporta el alimento desde una de las 4 bandejas situadas en su base hasta la boca del usuario.

Cuenta con 3 modos de funcionamiento: manual, semiautomático y automático.

El modo manual consiste en el control del brazo robótico a través de la palanca de un mando, la cual deberá ser accionada por algún asistente, lo que no reduce la dependencia del usuario a la hora de comer. El modo semiautomático funciona de modo que el usuario elige de qué bandeja cogerá comida el robot, y, por último, en el modo automático el robot elige por sí mismo de qué compartimento cogerá la comida.

### 3.2.2 Bestic

Bestic [107] **(figura 29)** es un robot de asistencia a la alimentación desarrollado en Suecia que fue lanzado al mercado en 2011. Es un brazo robótico con una cuchara en su extremo que recoge comida del plato situado en su base para acercarla a la boca del usuario.

*Figura 29. Robot Bestic (Fuente: [107])*

Este robot se caracteriza por su reducido peso, de poco más de 2 kilogramos, y puede ser controlado mediante botones, joystick o mediante un mando para un control más avanzado.

### 3.3.3 Mealtime Partner

Mealtime Partner [108] **(figura 30)** es un robot de asistencia a la alimentación que acerca la comida desde un plato hasta la boca del usuario.

Puede ser controlado de forma automática, cogiendo comida al ritmo al que coma el usuario, o de forma manual a través de la activación de dos interruptores, uno para cambio de plato y otro para acercar la comida del plato escogido con la cuchara.

*Figura 30. Robot Mealtime Partner (Fuente: [108])*

Resalta su diseño, ya que, a diferencia de los robots de alimentación asistida vistos hasta ahora, no dispone de los platos en su base, sino que están suspendidos en la parte delantera del robot, rotando para situar el plato deseado justo debajo de la cuchara, para que sea desde ahí desde donde parta la cuchara con la comida hasta la posición de la boca del usuario.

Su precio es de 7 870$.

## 3.3 Investigaciones relevantes en el campo

En los últimos años, se han realizado varios estudios e investigaciones que han contribuido significativamente al control de dispositivos inteligentes mediante

gestos faciales. Si bien estas investigaciones se centran en aspectos concretos como el reconocimiento de gestos faciales por visión artificial, el control de dispositivos inteligentes por detección de gestos y la interpretación de los gestos faciales detectados, sus conclusiones y hallazgos pueden ayudar en el desarrollo de la aplicación. Algunos de los trabajos más destacados incluyen:

- Facial Gesture Recognition in Patients with Facial Palsy [109]: Este trabajo trata el estudio de la identificación de gestos faciales en los rostros de pacientes con parálisis facial, lo que puede ayudar a elegir qué gestos se deberían considerar a la hora de desarrollar el código de la aplicación para que a pacientes con parálisis facial les resulte cómodo usarla.

- Smart Home Automation-Based Hand Gesture Recognition Using Feature Fusion and Recurrent Neural Network (Bayan Ibrahim Alabdullah) [110]: En este trabajo, se estudia el uso de visión artificial para detectar gestos realizados con las manos del usuario en lugar de con la cara, pero también con la finalidad de controlar un entorno inteligente (en este caso la domótica de una casa) en función de esos gestos. De forma conceptual, este estudio es similar al trabajo a realizar, pero se basa en la detección de una serie de movimientos que los usuarios objetivos de la aplicación que se desarrollará en este trabajo no pueden realizar.

- Smart Home Management System with Face Recognition Based on ArcFace Model in Deep Convolutional Neural Network (Thai-Viet Dang) [111]: Trabajo muy similar al anterior, en el que se controlan dispositivos IoT de un hogar mediante el reconocimiento de gestos realizados con la mano, con el añadido de la utilización de un sistema de reconocimiento facial previo a la realización de gestos manuales para autenticar la identidad del usuario.

- Real-Time Intelligent Facial Expression Recognition System [112]: Desarrollo de un sistema de visión artificial que interpreta cómo se siente el usuario en función de las expresiones faciales detectadas. Este proyecto muestra otro uso de la detección de gestos faciales mediante visión artificial.

Estas investigaciones proporcionan otro punto de vista sobre cómo funciona el control de dispositivos inteligentes mediante la detección de gestos, pero también sobre usos de la detección de dichos gestos faciales. Además, en ellos se refleja la necesidad de un enfoque multidisciplinario que combine conocimientos en visión artificial y en interacción hombre-máquina para poder desarrollar proyectos similares.

El estado del arte en el desarrollo de aplicaciones de control de dispositivos mediante gestos faciales, si bien es un campo muy concreto y poco estudiado, muestra un progreso significativo, con varias soluciones similares ya disponibles en el mercado y ciertas investigaciones académicas que respaldan su desarrollo.

La aplicación a desarrollar se apoyará en estos avances para proporcionar una herramienta accesible y eficiente, dirigida a mejorar la calidad de vida de personas con discapacidades motrices.

# 4. HERRAMIENTAS Y TECNOLOGÍAS DE DESARROLLO

En este apartado se detallarán las herramientas y tecnologías utilizadas en el desarrollo de la aplicación Android. Se cubrirán tres aspectos fundamentales: la instalación y configuración de Android Studio, el uso de Android Studio para el desarrollo de la aplicación y la utilización de MQTT Explorer para monitorizar la comunicación con dispositivos inteligentes.

## 4.1 Instalación y Configuración de Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android. Es una herramienta completamente gratuita que proporciona todas las funcionalidades necesarias para crear, probar y desplegar aplicaciones Android.

**Instalación**

Para instalar Android Studio [113], se debieron seguir estos pasos:

- Descarga: Se visitó la página oficial de Android Studio y se descargó el instalador correspondiente al sistema operativo (existen versiones para Windows, macOS o Linux).

- Instalación en Windows: Se ejecutó el archivo .exe descargado y se siguieron las instrucciones del asistente de instalación.

- Configuración inicial: Una vez instalado, se abrió Android Studio. Se inició el asistente de configuración, que permite una configuración estándar o una personalizada. Se seleccionó la estándar y el asistente descargó e instaló las últimas versiones del SDK de Android y otros componentes necesarios.

**Configuración**

- Configuración de la vista del IDE: En el menú de Android Studio, el apartado llamado View permite modificar la posición y visibilidad de la mayoría de elementos de la interfaz del entorno de desarrollo, haciendo la programación más cómoda y eficiente para programadores que no quieran tener una gran cantidad de información simultánea en la pantalla.

- Emulador de Android **(figura 31)**: Se puede configurar un dispositivo virtual para probar la aplicación desde dentro del propio entorno de desarrollo y sin necesidad de un dispositivo físico real. Sin embargo, este emulador no permite utilizar cámara, para eso hace falta un dispositivo Android físico, por lo que no es una herramienta a considerar en el proyecto.

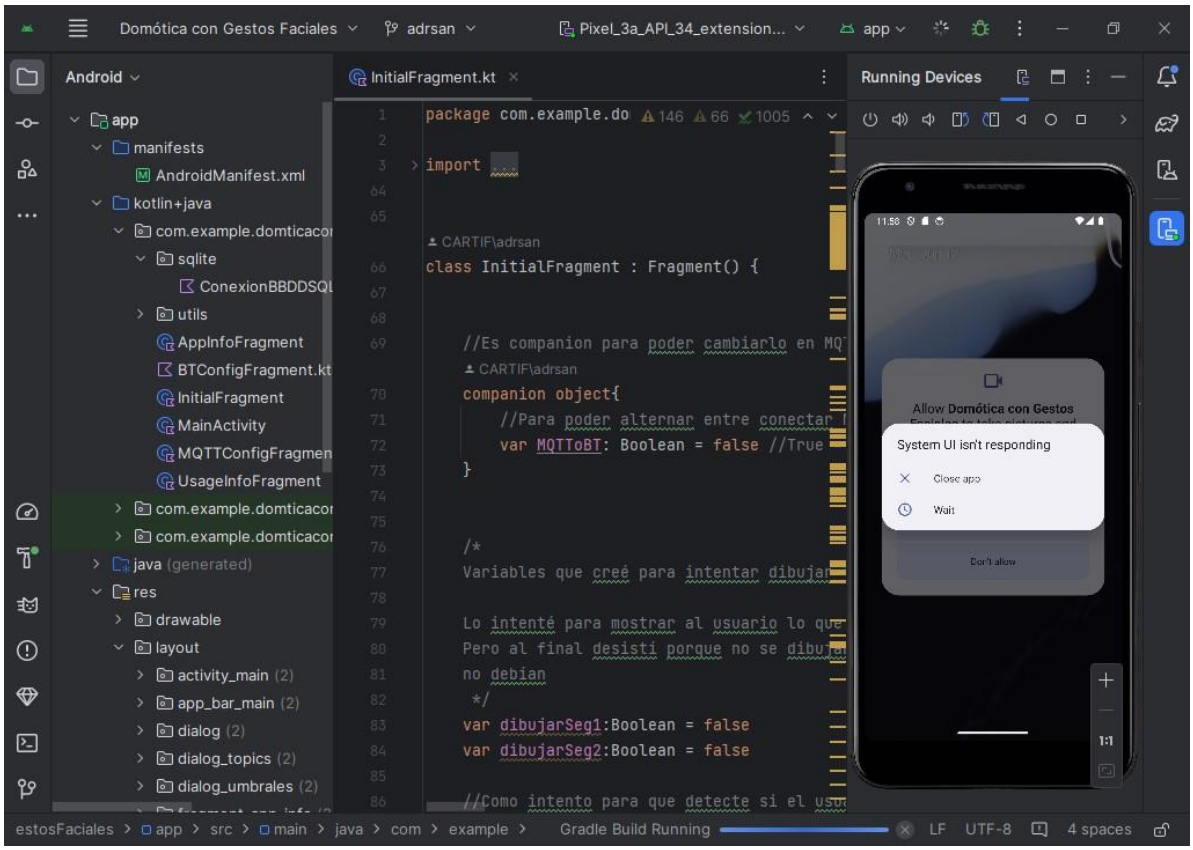



*Figura 31. Vista del IDE Android Studio con el emulador Android situado en el lateral derecho de la pantalla*

- Plugins: Se pueden instalar plugins o extensiones adicionales desde el menú de Android Studio, siguiendo la ruta File > Settings > Plugins, para extender las funcionalidades del IDE, por ejemplo, mejorando la productividad, la eficiencia y las capacidades de desarrollo, o instalando herramientas de diseño.

- Configuraciones de ejecución y depuración [114]: Android Studio permite modificar la forma en la que se ejecuta y depura el programa desarrollado desde Run > Edit Configurations. Desde esas opciones, se puede configurar las opciones de instalación, de inicio y de prueba de la aplicación. Sin embargo, no será necesario modificar estas opciones para el proyecto, las opciones por defecto serán suficientes.

- Dependencias y librerías: Como parte de la configuración de un proyecto, la herramienta de compilación Gradle, la cual se mencionó en los elementos clave

de Android Studio, permite incluir bibliotecas y módulos externos que se vayan a emplear en el programa.

## 4.2 Uso de Android Studio

Dado que Android Studio es un entorno de desarrollo muy completo y con muchas funcionalidades, se irá explicando la utilidad y modo de uso de las principales para comprender cómo abordar el desarrollo de la aplicación Android.

La primera elección a tomar en Android Studio al querer empezar a desarrollar una aplicación es la de seleccionar una plantilla **(figura 32)**, es decir, escoger un código básico a partir del cual comenzar a desarrollar la app. También existe la opción de una plantilla vacía, la que en la figura 32 se denomina “no activity”.

Existen plantillas predefinidas para dispositivos móviles, pero también para Smart Watches, televisores o incluso vehículos.

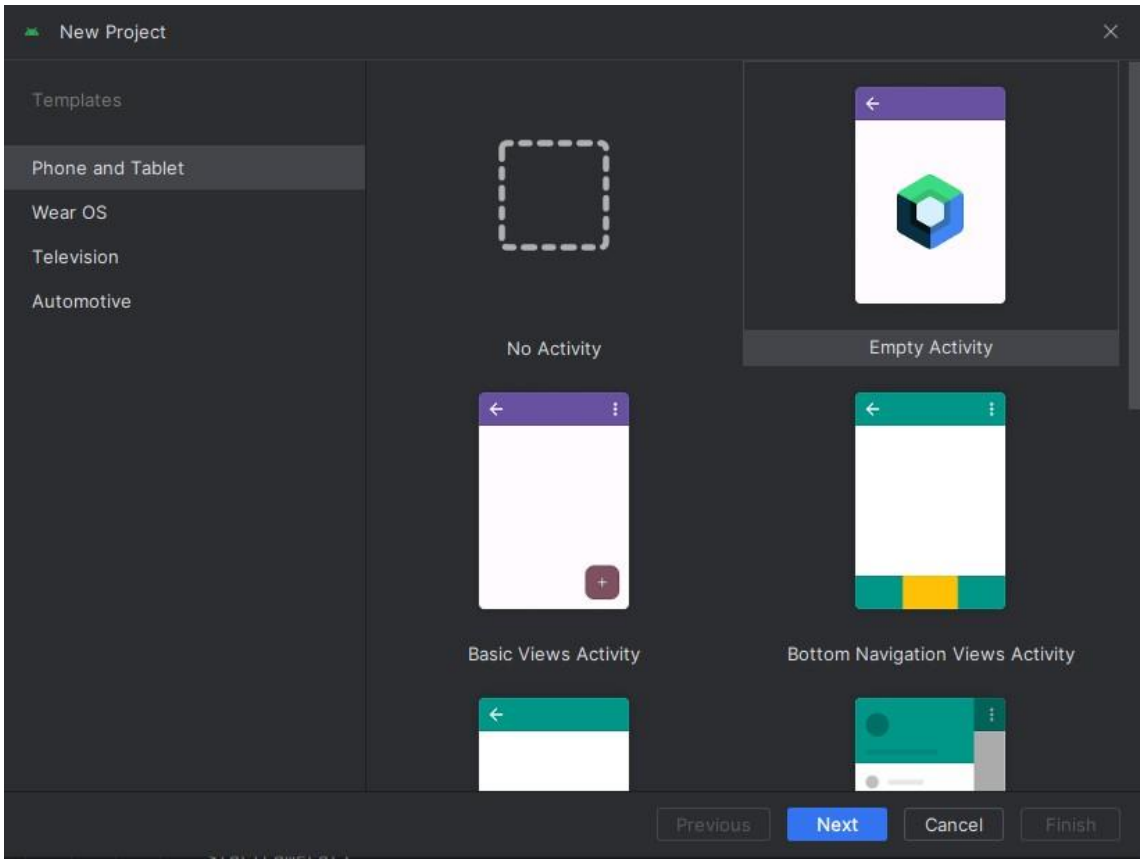



*Figura 32. Plantillas disponibles al crear un nuevo proyecto en Android Studio*

Una vez seleccionada la plantilla, la siguiente pantalla que muestra Android Studio es una en la que se debe dar nombre al proyecto, escoger una dirección

de guardado, elegir si programar en Kotlin o en Java y decidir la versión de Android a partir de cuál la aplicación será compatible con todos los dispositivos **(figura 33)**.

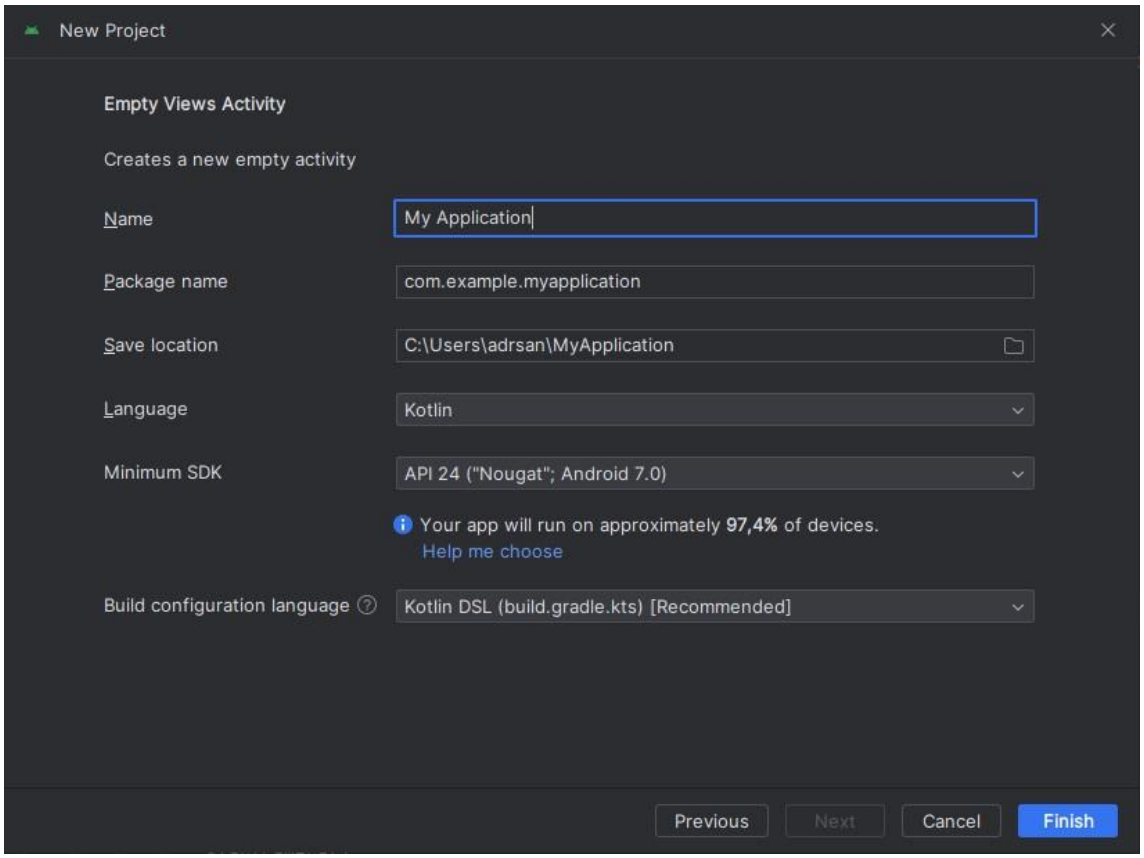



*Figura 33. Elección de nombre, lenguaje de programación y versión de Android en Android Studio*

Una vez se ha pulsado el botón Finish de la esquina inferior izquierda, se llega a la vista de proyecto como tal **(figura 34)**, en el que se escribirá el código de la aplicación.

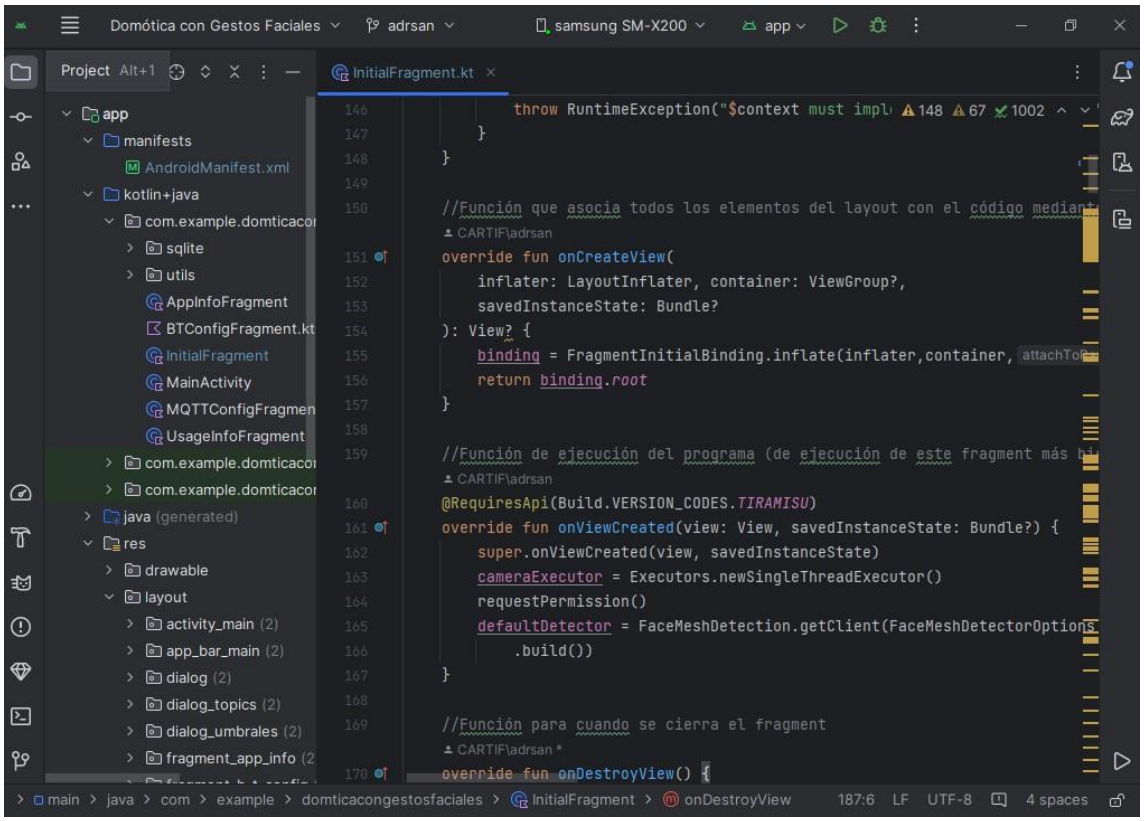



*Figura 34. Vista principal de Android Studio*

Esta interfaz es compleja y cuenta con muchas partes, por lo que se explicarán solamente las más importantes, útiles y recurrentes [115, 116].

1. Barra superior **(figura 35)**:


*Figura 35. Barra superior de Android Studio*

En la barra superior de la interfaz y yendo de izquierda a derecha encontramos:

- Icono de Android: No tiene función, solo indica que el entorno de desarrollo es Android Studio.
- Menú principal: Representado por las 4 barras blancas horizontales. Incluye una gran variedad de opciones respectivas a la gestión de archivos **(figura 36)**, a la gestión de la interfaz, de la compilación, de las herramientas o de la navegación entre partes del IDE.

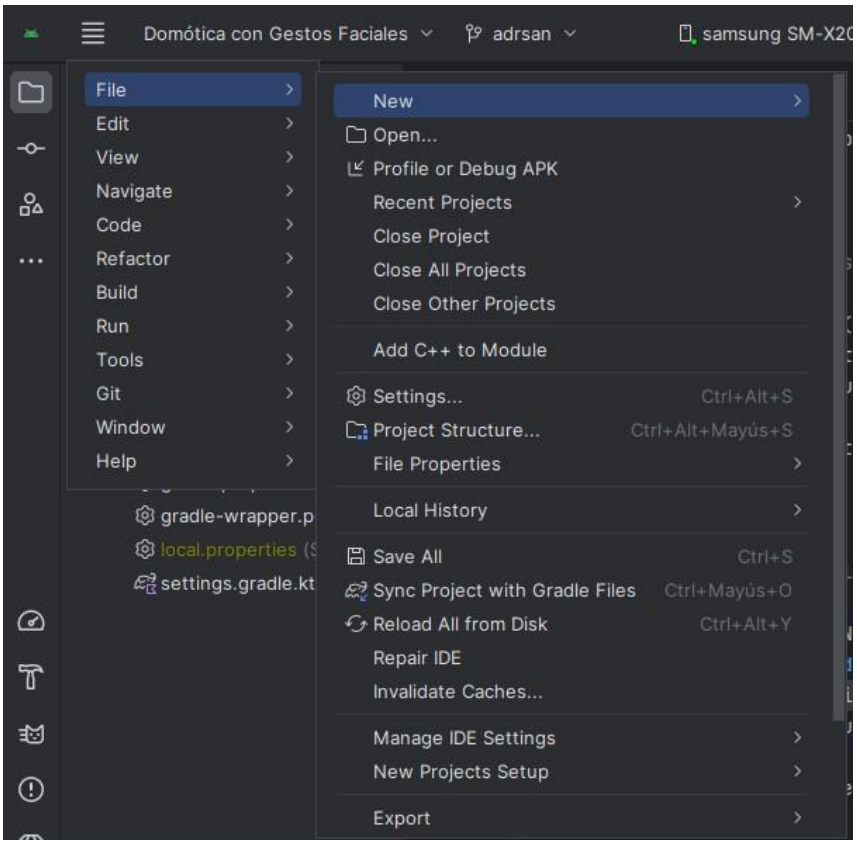
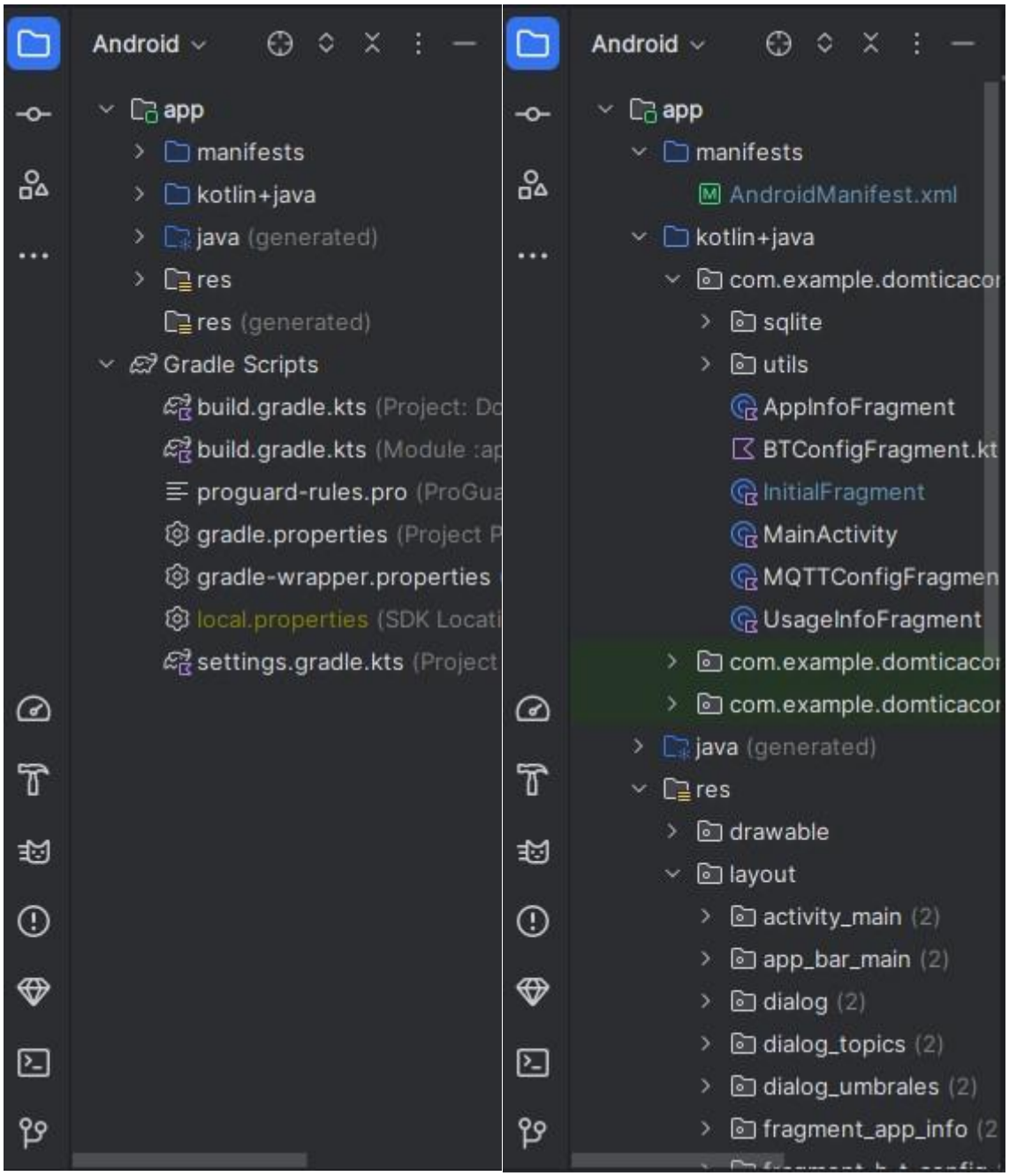



*Figura 36. Menú principal desplegado*

- Navegación entre proyectos: Muestra el nombre del proyecto actual y, en caso de clicar en él, despliega una ventana con la opción de crear un nuevo proyecto y con los proyectos creados anteriormente en caso de que el programador desee abrir alguno de ellos.
- Git Branch: Git es un sistema de control de versiones, lo que permite a desarrolladores trabajar en diferentes versiones de un mismo proyecto de forma simultánea. También permite realizar un seguimiento de las modificaciones realizadas en el código.
- Selector de dispositivos: Permite elegir el dispositivo en el que se ejecutará la aplicación. Puede ser el emulador Android interno o un dispositivo Android externo conectado al PC por USB.
- Configuración de ejecución: El icono de Android con la palabra “app” al lado indica la configuración actual de ejecución y compilación.
- Icono de ejecución: El triángulo verde ejecuta la app al ser clicado.
- Icono de debug: Se emplea para ejecutar el código con interrupciones para detectar de manera más eficiente posibles errores.
- Más opciones: Los tres puntos en vertical indican más opciones relativas a la edición y gestión de configuraciones.

2. Vista de los archivos del proyecto **(figuras 37 y 38)**: Se sitúa en el lateral izquierdo de la interfaz y representa la estructura de la app [77].




*Figuras 37 (izquierda) y 38 (derecha). Archivos del proyecto (por carpetas y desglosados, respectivamente)*

La pestaña superior en la que se lee Android indica la forma en la que se visualizan los archivos, y hay varias opciones entre las que elegir. Se puede seleccionar como “Project” para ver los archivos estructurados de la forma en la que se verían al exportar el proyecto. De igual manera, existen otras formas de visualizar los archivos del proyecto, como por paquetes o viendo solo los

archivos que están siendo modificados en ese momento (o lo que es lo mismo, los archivos abiertos).

Todos los proyectos creados en Android Studio contienen uno o más módulos con archivos de código y con archivos de recursos, y la vista “Android” organiza los archivos por módulos.

Cada módulo tiene 3 carpetas en su interior, las que se ven en la figura 37:

- manifests, que contiene el archivo AndroidManifest (visible en la figura 38), en el cual se declaran las actividades, servicios, permisos necesarios y funciones de la app [117].
- java, que contiene los archivos de código fuente (visible en la figura 38).
- res, que contiene los recursos de la aplicación codificados en XML y que incluyen interfaces, icono de la app, colores e imágenes.

Además de los módulos de la app, también se encuentran los archivos de compilación, almacenados en “Gradle Scripts” (visible en la figura 37).

3. Barra de herramientas (Toolbar): Incluye múltiples acciones y opciones **(figura 39)**.

*Figura 39. Barra de herramientas situada en el lateral izquierdo de la UI de Android Studio*

Las opciones presentes en esta toolbar ordenadas de arriba abajo son:

- Profiler [118]: ofrece herramientas para la optimización de la ejecución de la aplicación.
- Build: muestra por pantalla mensajes relacionados con el proceso de compilación y construcción del programa.

- Logcat [119]: herramienta de depuración que muestra mensajes recibidos desde el dispositivo Android en el que se está probando la aplicación en tiempo real **(figura 40)**. Esto es sumamente importante para corregir el funcionamiento de la aplicación en caso de que no funcione como se deseaba.

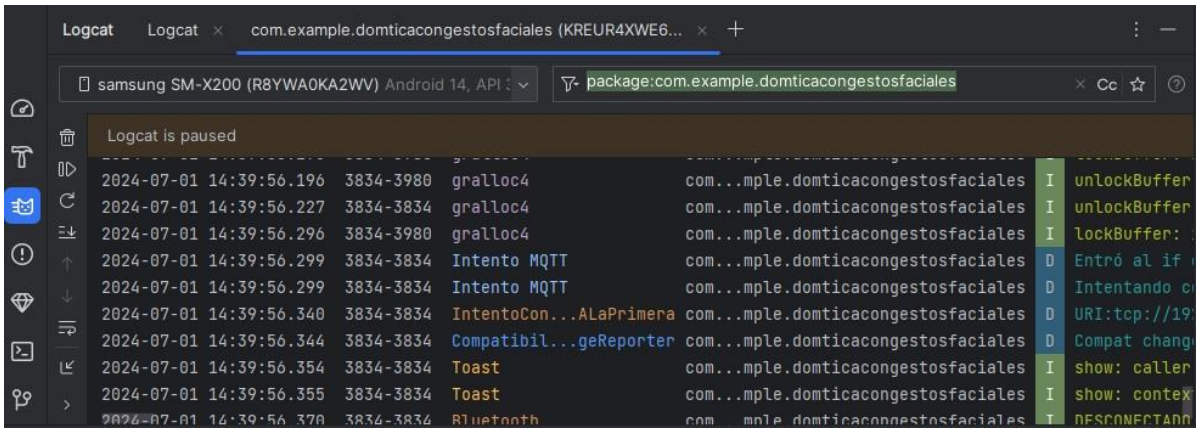



*Figura 40. Mensajes recibidos desde el dispositivo mostrados en el Logcat*

- Problems: identifica problemas en el código fuente y en la configuración del proyecto. Enumera errores y advertencias formando una lista.
- App Quality Insights [120]: proporciona información sobre la calidad de la aplicación, esto es, da estadísticas sobre fallos que experimentan los usuarios, problemas de estabilidad o las versiones de la aplicación más afectadas por los fallos.
- Terminal: proporciona acceso a una línea de comandos de terminal para realizar diversas tareas, como acciones relacionadas con Git o con comandos de Gradle para ejecutar pruebas de compilación.
- Version Control: herramienta integrada con Git para el control de versiones de la aplicación.

4. Editor de código **(figura 41)**: Situado en el centro de la pantalla, es donde se escribe y modifica el código que compone la aplicación.

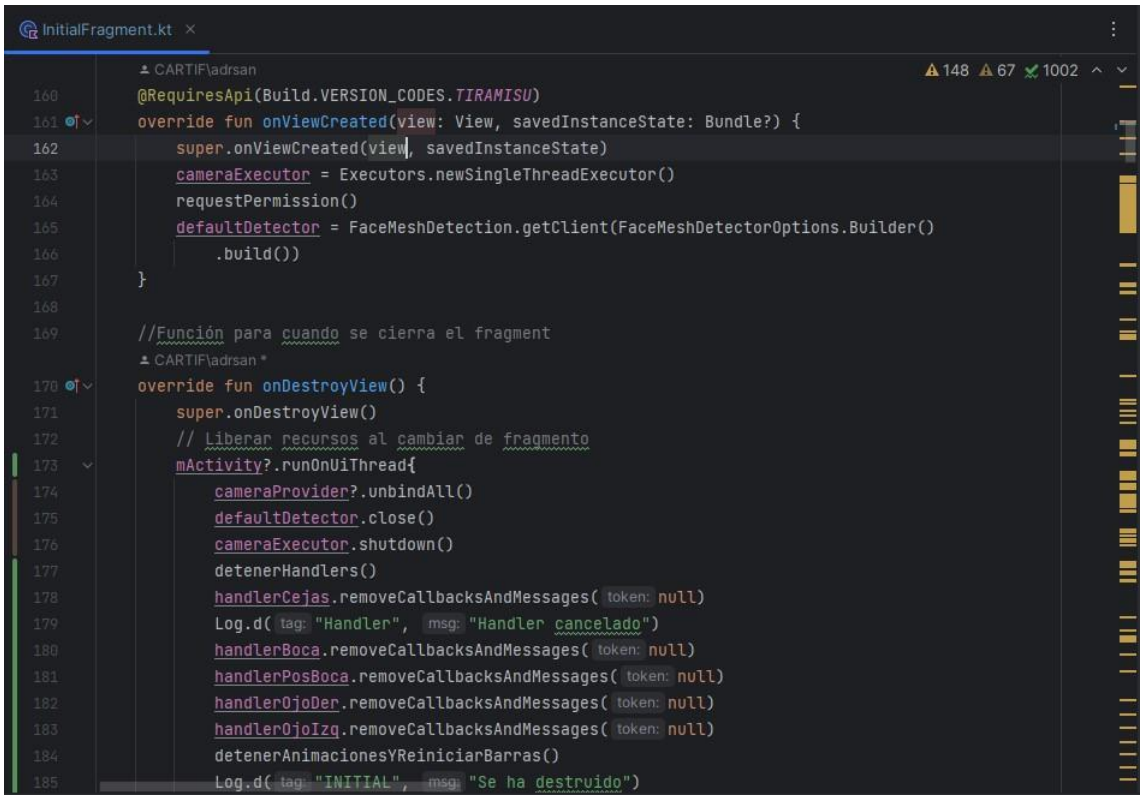



*Figura 41. Editor de código de Android Studio*

El editor de código tiene varias funciones para facilitar el desarrollo de aplicaciones, como el resaltado de sintaxis (los colores que se pueden ver en la figura 41, indicando distinto tipo de elemento, como funciones, nombres de variables, etc.), el autocompletado (de forma que el IDE te sugiere opciones para acabar una línea de código antes de que se termine de escribir como tal) o herramientas de refactorización, de modo que resulte sencillo cambiar el nombre de una variable de forma simultánea en todo el código en lugar de ir línea a línea.

También se dispone de múltiples atajos de teclado [121] para acciones comunes como copiar (Ctrl+C), cortar (Ctrl+X) o pegar (Ctrl+V) código, deshacer (Ctrl+Z) acciones, buscar (Ctrl+F) en todo el código y un gran número de acciones más avanzadas.

Cabe destacar que el editor proporciona también una vista de diseño visual para el desarrollo y edición de código XML **(figura 42)**, es decir, de las interfaces de la aplicación Android. Esto hace más sencillo e intuitivo el desarrollo de interfaces al poder ver al instante los cambios provocados en lugar de tener que esperar a ejecutar en el dispositivo.

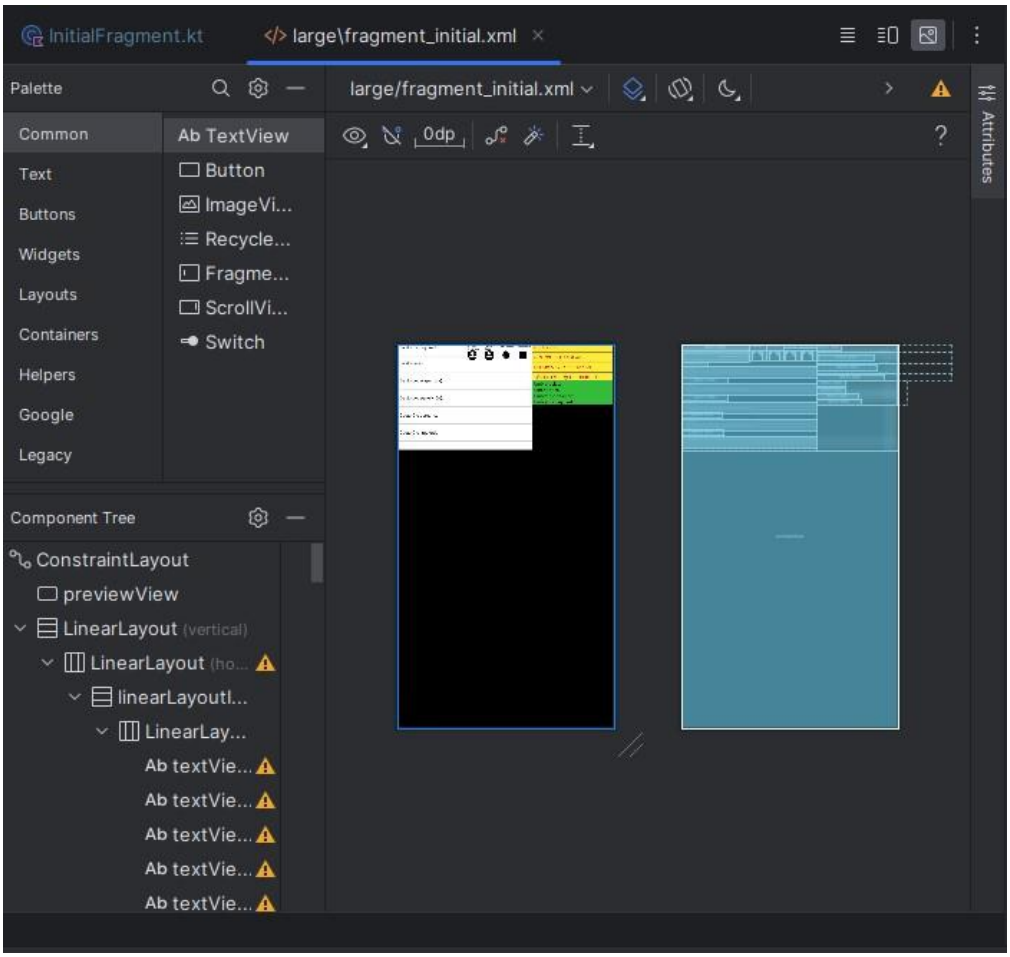



*Figura 42. Editor visual de interfaces de Android Studio*

## 4.3 Uso de MQTT Explorer

MQTT Explorer es una herramienta que permite conectarse a un servidor MQTT y visualizar los mensajes que se envían y reciben desde dicho servidor [122]. Esto la hace muy útil para comprobar el correcto funcionamiento de la comunicación MQTT de la aplicación.

Para utilizar MQTT Explorer, primero se descargó e instaló la última versión del software disponible para Windows, si bien también hay versiones para macOS y Linux.

Una vez descargado, se abre MQTT Explorer y se configura una nueva conexión **(figura 43)**, introduciendo la dirección IP o el nombre del host del servidor MQTT, además del puerto y las credenciales de acceso (usuario y contraseña) si es que es necesario.

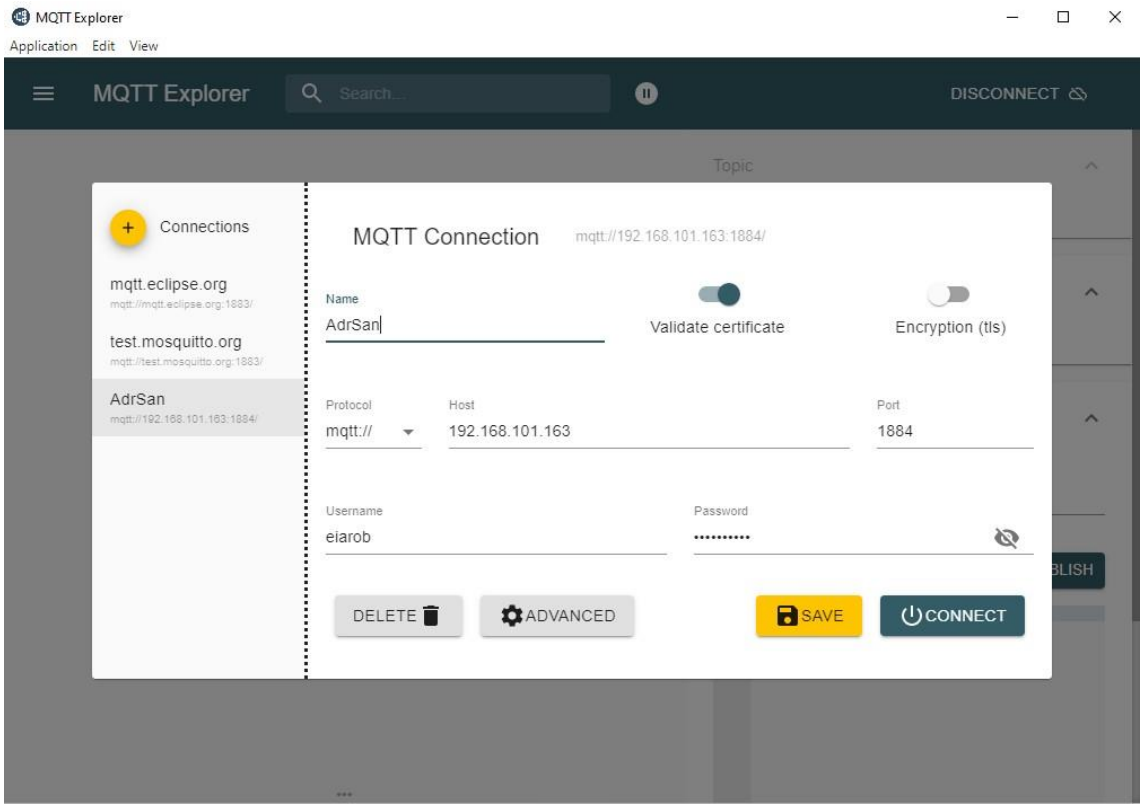



*Figura 43. Pantalla de conexión de MQTT Explorer*

Tras ello, se hace clic en “CONNECT” para establecer la conexión.

Una vez se ha conectado con el servidor MQTT, se pueden monitorear los diferentes tópicos. Así, aparecerán por pantalla los mensajes que se publiquen en cada tópico del servidor **(figura 44)**.

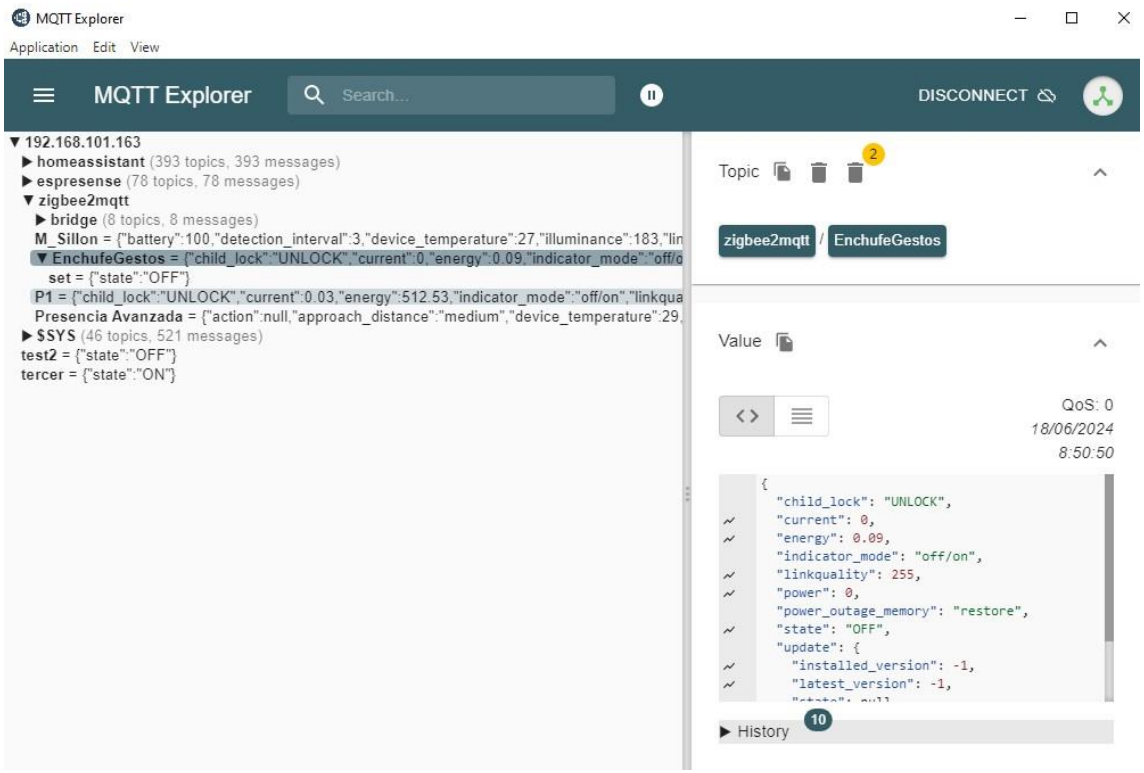



*Figura 44. Vista de los mensajes publicados en MQTT Explorer*

También se pueden publicar mensajes en cualquier tópico para comprobar si el dispositivo de prueba los recibe. En el panel de publicación, se debe escribir el nombre del tópico en el que se quiera publicar, y tras ello, se introduce el mensaje y se pulsa “PUBLISH”.

De este modo, MQTT Explorer permitirá depurar problemas de comunicación MQTT de la aplicación y asegurarse de que los dispositivos inteligentes reciben los mensajes correctamente al poder ver en tiempo real si los mensajes enviados desde la aplicación llegan el servidor y si los mensajes enviados son los esperados.

# 5. DESARROLLO DE LA APLICACIÓN ANDROID

En esta parte del trabajo se detallará el proceso de desarrollo de la aplicación Android con la finalidad de cumplir los objetivos establecidos al comienzo de este proyecto.

El código será escrito empleando lenguaje Kotlin, ya que, como se mencionó, es más conciso, popular y seguro que Java.

Se analizará en detalle el código desarrollado para el Back-end (datos y lógica interna del programa) y del Frtont-end (interfaz que ve el usuario) [123] de la app.

## 5.1 Dispositivos empleados en el desarrollo de la aplicación

Antes de profundizar en el código, se enumerarán los distintos dispositivos utilizados a lo largo del desarrollo de la aplicación Android y se mencionarán sus principales características.

- Tablet Samsung Galaxy Tab A8 [124]

  Fue el dispositivo que más se empleó a lo largo del desarrollo de la aplicación. Cada cambio en la lógica de funcionamiento del código fue ejecutado primero en esta Tablet para comprobar su viabilidad.

  Las características del dispositivo más importantes para el funcionamiento de la aplicación son la resolución de la cámara, para saber qué tan precisa será la detección de puntos faciales, y el procesador del dispositivo para comprobar si la ejecución de la aplicación será fluida. Cuenta con una cámara delantera (la que apunta hacia el usuario) de 5 megapíxeles y procesador de 8 núcleos que trabaja a 2 GHz.

  Aunque viene de fábrica con Android 11, fue actualizado a la versión 14.

  En la **figura 45** se ve su aspecto.

*Figura 45. Tablet Samsung Galaxy Tab A8 (Fuente: [124])*

- Móvil OPPO A58 [125]

También se empleó este teléfono móvil **(figura 46)** en las pruebas de la aplicación para comprobar que el funcionamiento del programa era fluido en dispositivos con procesadores menos potentes y que la interfaz para móviles era adecuada y se ajustaba al menor tamaño de pantalla sin situar elementos como iconos o la vista de la cámara colocándolos fuera de la pantalla.

Su cámara frontal, la que utiliza la aplicación, es de 8 megapíxeles, y su procesador es un Helio G85 [126], un procesador de 8 núcleos en los que 6 trabajan a 1.8GHz y los otros 2 restantes a 2GHz.

Viene de fábrica con Android 13 y no ha sido actualizado.


*Figura 46. OPPO A58 (Fuente: [127])*

- Enchufe inteligente FDTEK [128] **(figura 47)**

Fue el dispositivo con el que se probaba la comunicación MQTT de la aplicación.

A lo largo de las pruebas, se configuró el servidor, puerto, usuario, contraseña y tópico de forma que fuera este enchufe inteligente el que recibiera los mensajes enviados desde el dispositivo Android.

Usa protocolo de comunicación Zigbee 3.0 y tiene una distancia de comunicación inalámbrica de 55 metros.


*Figura 47. Smart Switch FDTEK (Fuente: [128])*

- Lámpara de escritorio TETRIAL [129] **(figura 48)**

Lámpara enchufada al Smart Switch FDTEK. Sujetaba la bombilla que encendían las señales MQTT al activar y desactivar el enchufe inteligente.

*Figura 48. Lámpara TERTIAL (Fuente: [129])*

- Bombilla LED E27 [130]

Bombilla que se encendía al recibir el enchufe los ON u OFF que se enviaban desde la aplicación mediante protocolo MQTT **(figura 49)**.

*Figura 49. Bombilla LED E27 Solhetta (Fuente: [131])*

- Microcontrolador ESP32 DFRobot FireBeetle 4.0 [132]

Incorpora el procesador dual-core ESP-WROOM-32 y cuenta con conectividad Wi-Fi y Bluetooth.

Puede ser alimentado por USB o por una batería externa de 3.7V.

A continuación, se puede ver su aspecto en la **figura 50**.


*Figura 50. ESP32 DFRobot FireBeetle (Fuente: [132])*

Como se ve en su esquema de conexiones **(figura 51)**, cuenta con 5 pines analógicos (en el lado derecho superior del esquema), 10 digitales (lado izquierdo superior), 1 pin de tierra (GND), 1 salida de 3.3V para alimentar otros componentes, 1 interfaz UART (TXD y RXD) [133], otra I2C (SDA y SCL) [134] y otra SPI (MISO y MOSI) [135] para comunicación serie entre dispositivos, múltiples pines de entrada/salida generales (GPIO) y un pin de enable (EN).

Wi-Fi y Bluetooth están integrados en el microcontrolador mediante una antena en la propia placa.

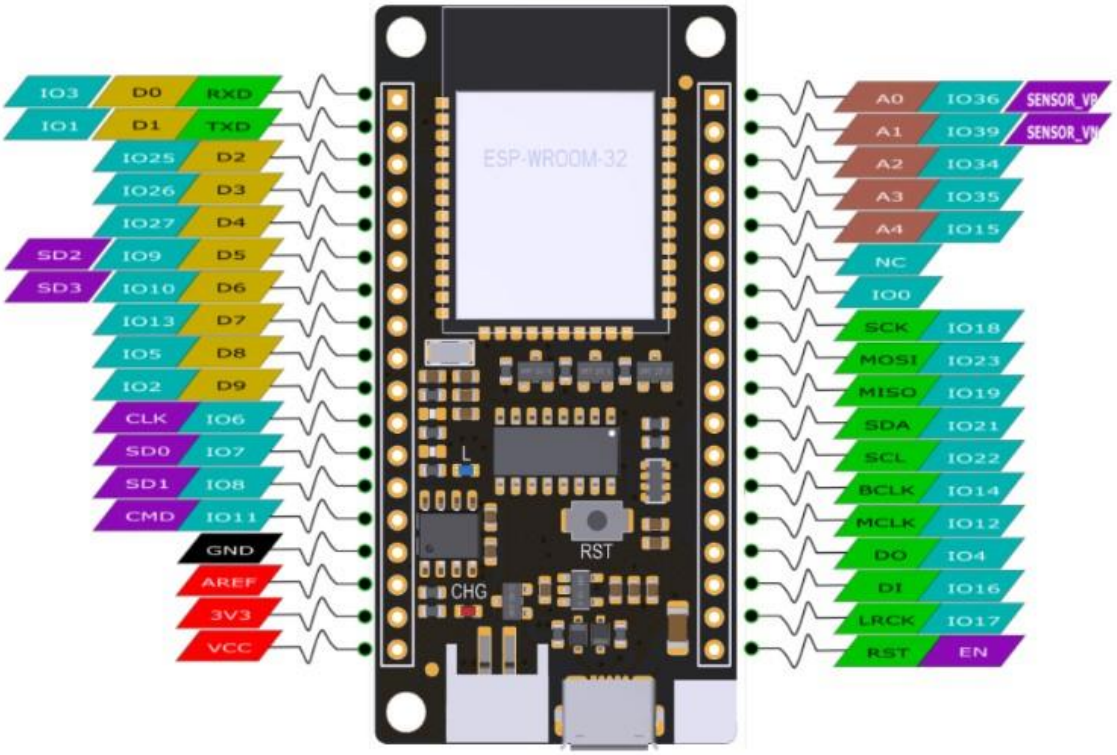



*Figura 51. Esquema de pines del ESP32 DFRobot FireBeetle (Fuente: [132])*

- Placa propia con 2 resistencias y transistores **(figuras 52 y 53)**

El objetivo de la placa es activar las 2 entradas Jack del robot OBI a partir de la señal en los pines de salida del ESP32. Cuando esté activa la comunicación Bluetooth de la aplicación y el dispositivo Android esté vinculado al microcontrolador ESP32, se enviarán señales desde la app al ESP32 al detectar ciertos gestos realizados. Después, el microcontrolador activará pines digitales en respuesta al mensaje recibido desde la app gracias al código subido al micro. Por último, la placa diseñada se encarga de cerrar el mismo circuito que se cerraba al accionar los pulsadores originales.

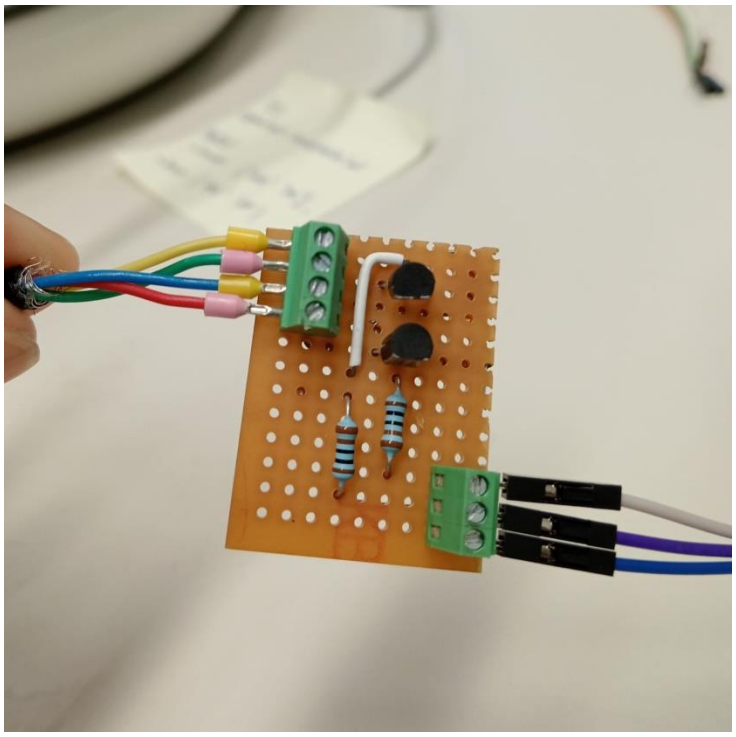


*Figura 52. Parte superior de la placa*

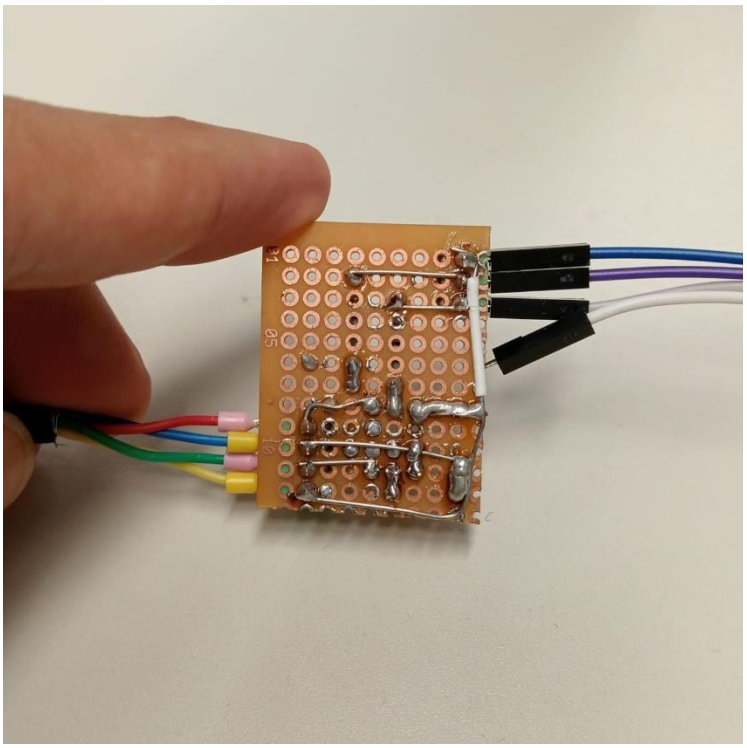


*Figura 53. Parte inferior de la placa*

Se consigue el funcionamiento esperado gracias a los 2 transistores BJT NPN BC547 [136], visibles en la figura 52. Cada uno funciona como un interruptor

al hacerle pasar corriente entre el colector y el emisor controlando la corriente base **(figura 54)**. Así, al activar un pin del ESP32, le llega corriente a la base del transistor y cierra el circuito como lo hacían los interruptores.

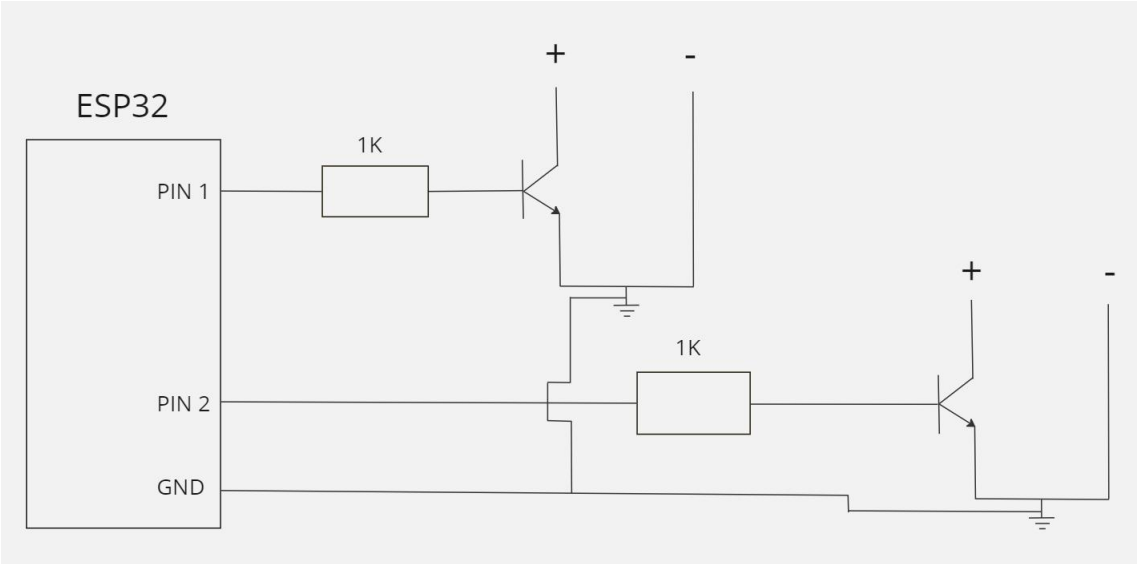



*Figura 54. Esquema eléctrico del circuito de la placa*

Los pines de salida de la ESP32 ofrecen 3.3V, de modo que la corriente de la base del transistor será *Ib = (Vesp32-Vbe)/1k = (3.3-0.7)/1000 = 2.6mA*. Esta corriente de base permitirá controlar el transistor para que circule una corriente colestor-emisor.

- Robot OBI [53]:

  Visible en la figura 12, es un robot diseñado para asistir en la alimentación de individuos con dificultades en el movimiento.

  Empleando el microcontrolador ESP32 y la placa diseñada, se lograrán sustituir sus interruptores originales por la realización de gestos faciales.

Una vez vistos todos los elementos a emplear, se ha realizado un esquema muy simple sobre el funcionamiento de dichos dispositivos en el proyecto **(figura 55)**.

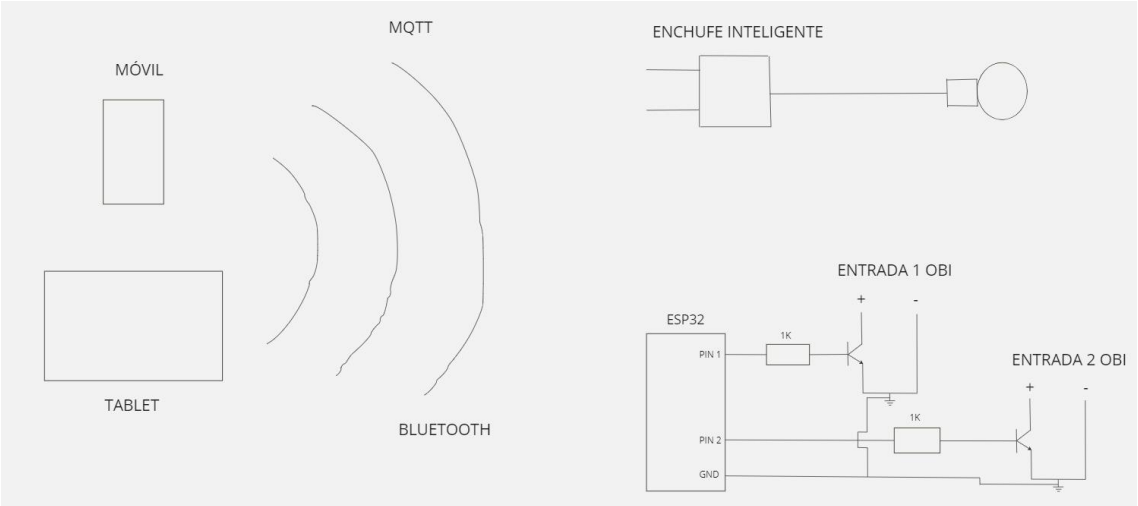



*Figura 55. Representación del proyecto*

## 5.2 Organización del código de la aplicación

La aplicación está ideada para que el control de entornos pueda ser gestionado por un usuario que no pueda interactuar directamente con la pantalla del dispositivo, pero es necesario configurar primero la comunicación MQTT y/o la comunicación Bluetooth. Esta configuración requiere de la introducción de números y cadenas de texto en la aplicación en caso de MQTT y de seleccionar el dispositivo vinculado en caso de Bluetooth, para lo que hará falta que un asistente establezca dichas comunicaciones la primera vez que se ejecute la aplicación en un dispositivo. Una vez conectado, el usuario con problemas de movilidad ya podrá controlar su entorno cómodamente y no debería ser necesario que el asistente tenga que volver a interactuar con la aplicación para la configuración de conexiones, aunque se cierre y se vuelva a abrir la aplicación.

Para poder llevar a cabo este planteamiento, la estructura de la aplicación debería disponer de distintos apartados de forma que el asistente pueda navegar entre ellos para configurar el uso de la aplicación a gusto del usuario con discapacidad y que este pueda después controlar los dispositivos inteligentes de su entorno sin tener que navegar por las distintas partes de la aplicación. Además, será necesario el uso de alguna herramienta de almacenamiento de datos para no tener que introducir los parámetros propios de la comunicación MQTT ni tener que seleccionar el dispositivo Bluetooth con el que se quiera comunicar en cada ejecución de la aplicación. Las partes en las que se dividirá el código de la aplicación son:

- Principal, o la vista de la cámara: Como se estableció en los objetivos del proyecto, se debe implementar algún tipo de representación visual que permita al usuario conocer qué gestos faciales se han identificado y qué acciones se están realizando. Para ello, la mejor forma de informar al usuario es mostrando la vista en tiempo real de la cámara del dispositivo, para que el usuario pueda ver y entender la posición y gesticulación actual de su cara, además de estableciendo iconos o dibujos que indiquen al usuario qué interacciones están sucediendo, como envíos de mensajes o en qué tópico MQTT se está publicando.

- Configuración de la comunicación MQTT: Otro de los objetivos del proyecto es el establecimiento de una conexión fiable entre el dispositivo Android y los elementos inteligentes, y para el establecimiento de la conexión MQTT es necesario la introducción de dirección, puerto, usuario, contraseña y tópicos en los que publicar. El usuario final de la aplicación no puede introducir dichos parámetros por sí mismo, y la interfaz para introducir y visualizar texto ocuparía mucha superficie de la vista de la cámara, por lo que se hace necesario que se establezca la comunicación en otra pantalla dentro de la aplicación que sea usada por el asistente.

- Configuración de la comunicación Bluetooth: Al igual que con MQTT, hará falta alguna interfaz que permita al asistente establecer una conexión mediante Bluetooth con algún dispositivo con el que interactuar. Para no ocultar la vista de la cámara con interfaces que el usuario no usará directamente, es mejor opción el establecer la comunicación en una parte de la aplicación separada de la vista de la cámara.

- Ajustes: Se permitirá la personalización del control de dispositivos mediante medidas como seleccionar qué acción realiza cada gesto para que el uso de la aplicación sea más cómodo al repetir menos veces los gestos que puedan resultar más molestos de hacer. También se podrá elegir qué elementos de la interfaz ocultar o cuales mostrar.
  Todas estas acciones requieren de interactuar con la pantalla, por lo que será un apartado de la app con la que interactuará el asistente y no el usuario, y se programará en una pantalla diferente a la de la cámara.

- Información sobre el uso de la app (Manual de usuario): Es conveniente que dentro de la propia aplicación exista una parte en la que se indique cómo funciona la app. Se puede informar al asistente sobre qué debería hacer para establecer la comunicación y sobre cómo configurar la app a

gusto del usuario. Para ello, se puede añadir otra pantalla que haga las veces de manual para facilitar la labor del asistente.
Sin embargo, no tendría mucho sentido que esa información se mostrara por pantalla en todo momento aún cuando el usuario ya sabe cómo usar la aplicación.

Adicionalmente, el asistente debe poder pasar de una pantalla a otra cuando así lo requiera o lo desee, de modo que se implementará algún elemento que posibilite la navegación entre las distintas partes de la aplicación para hacer posible este proceso, como por ejemplo un menú.

También se implementarán herramientas como una base de datos para el almacenaje de servidor, puerto, usuario, contraseña, tópicos o la dirección MAC del dispositivo Bluetooth con el que se estableció conexión la última vez. Esto hará posible que, a partir de la primera ejecución de la aplicación, el usuario pueda utilizar la aplicación directamente, sin necesitar asistencia para configurar la app en ningún momento.

Una vez se ha definido la organización conceptual de la aplicación, es necesario comprender cómo llevar a la práctica esta estructura.

Android incluye una serie de elementos que hacen posible una estructuración de la aplicación en la que se pueda navegar entre diferentes pantallas a través de un menú.

Estos elementos son:

- **Activity**: Una actividad, o “Activity” [137, 138, 139] es el componente base de una aplicación Android. Representa una única pantalla con un interfaz en la que el usuario puede interactuar. Son las Activities las que gestionan la interacción del usuario con la pantalla, como respondiendo a los toques o pulsaciones de botones. Las Activities tienen un ciclo de vida gestionado por el sistema operativo y el programador puede manejar cómo se debe comportar la Activity cuando su pantalla se inicia, se pausa, se renauda o se destruye. Las Activities pueden interactuar con otros componentes del sistema, lo que permite a la aplicación realizar operaciones en segundo plano.

- **Fragment**: Un fragmento, o “Fragment” [139, 140] es un componente modular de la interfaz de usuario de una actividad, lo que significa que forma parte de una Activity. Los Fragment permiten unas interfaces más flexibles y reutilizables. Tienen un ciclo de vida propio, pudiendo parar o finalizar sin que lo haya hecho la Activity que los contiene. Sin embargo,

su ciclo de vida sí está asociado al Activity, lo que quiere decir que si finaliza la actividad, ellos también.
Todo esto significa que los Fragment permiten cambiar dinámicamente partes de la interfaz de usuario dentro de un Activity sin necesidad de reemplazar toda la pantalla, lo que será útil en la aplicación al poder cambiar de parte del programa manteniendo el mismo menú.

- **App bar**: Una app bar [141] es un elemento de interfaz que proporciona una barra de herramientas en el borde superior de la pantalla **(figura 56)**. Se usa para ofrecer una navegación simple y coherente. Suele incluir un botón para abrir un menú en el que haya opciones más avanzadas. Ayuda a mantener una interfaz consistente a lo largo de la ejecución del programa, aunque se cambie de pantalla.

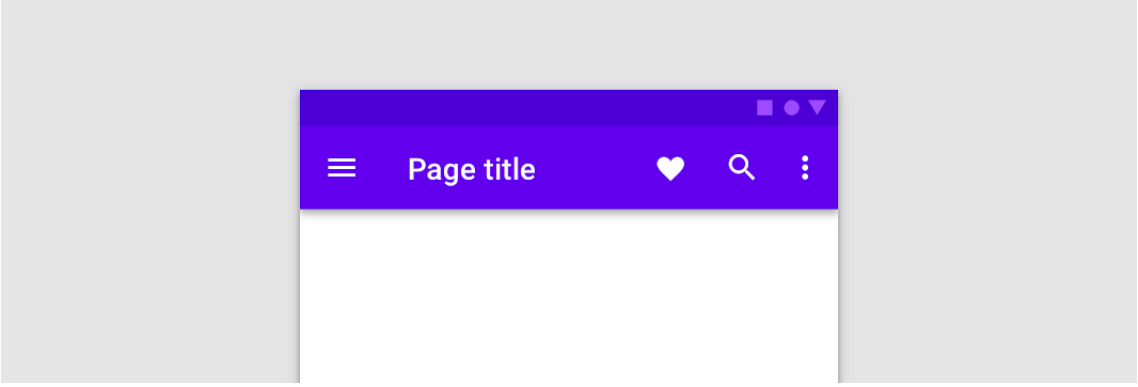



*Figura 56. Ejemplo de app bar (Fuente: [141])*

- **Navigation Drawer** [142]: Es un elemento de la interfaz Android que muestra un panel que desliza desde un lateral de la pantalla para revelar opciones de navegación de la aplicación **(figura 57)**.

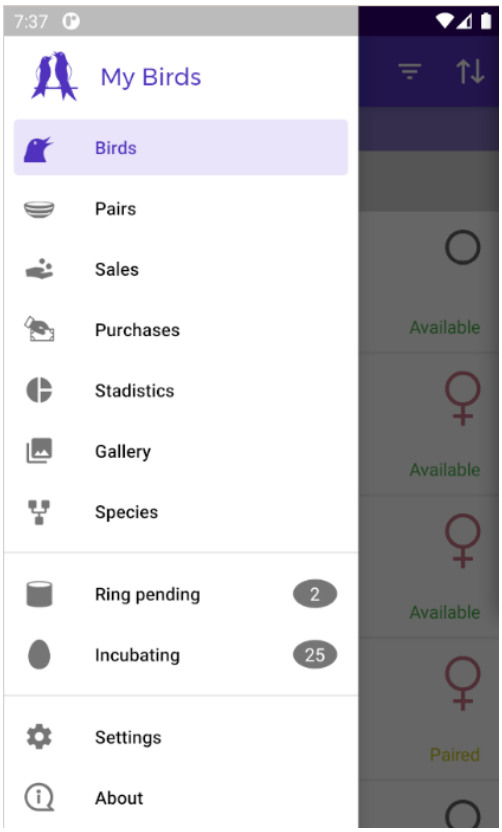



*Figura 57. Ejemplo de Navigation Drawer (Fuente: [143])*

Las aplicaciones Android pueden estar compuestas por múltiples Activities, pero la navegación entre ellas difiere de la navegación entre Fragments.

Para pasar de una Activity a otra se emplea un "Intent" [144], una operación en programación Android que provoca un cambio de contexto, es decir, se vuelve a crear la interfaz de usuario y se destruye la Activity anterior. Además, la creación de una nueva Activity consume más recursos del dispositivo que el crear un nuevo Fragment. Esto hace que no sea recomendable ejecutar muchos cambios de Activity.

Para navegar entre fragmentos, en cambio, se emplea el mecanismo llamado "FragmentManager" [145], que permite reemplazar, eliminar y realizar varias operaciones con Fragments en Android sin cambiar de contexto. Este método es más eficiente en cuanto al uso de recursos se refiere que los Intents.

Las diferencias entre Activity y Fragment [146] se enumeran en la **tabla 2**.

| ACTIVITY | FRAGMENT |
|---|---|
| Una actividad presenta una interfaz en la que el usuario puede interactuar con el dispositivo Android | Un fragmento es una parte de la actividad, también presenta una interfaz |
| La actividad es independiente del fragmento, puede existir sin ninguno de ellos | El fragmento no puede existir por sí solo, necesita una actividad que lo contenga |
| La actividad tiene ciclo de vida propio | El ciclo de vida del fragmento depende de la actividad, si esta termina, el fragmento también |
| La actividad es un elemento más pesado, requiere de más recursos para su creación | El fragmento es un elemento más ligero, está ideado para que su creación, gestión y destrucción sean más eficientes |
| Administrar la navegación entre actividades es un proceso complejo | La navegación entre fragmentos es más simple y rápida |

*Tabla 2. Diferencias entre Activity y Fragment*

Debido a las diferencias de rendimiento entre los Intent y el uso de FragmentManager, para conseguir implementar la estructura de aplicación deseada, la solución más eficiente y que ofrece una navegación más sencilla es el establecer una única Activity principal que actúe como contenedor de Fragments y que maneje el Navigation Drawer [147], de forma que el asistente podrá cambiar de pantallas de forma eficiente y rápida **(figura 58)**.

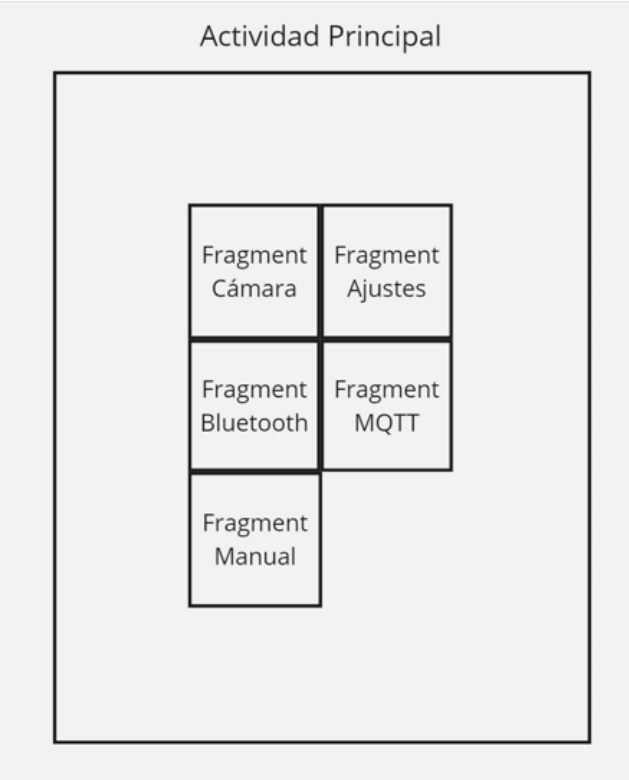



*Figura 58. Estructura de la aplicación representada con Activities y Fragments*

Así, la aplicación consistirá de una sola actividad que albergue varios fragmentos, entre los que se navega con un menú NavigationDrawer al que se puede acceder clicando en el app bar superior.

También existen en Android herramientas para almacenar datos entre distintas ejecuciones de la aplicación, como SharedPreferencies, además de otras externas, como SQLite. Se emplearán dichas herramientas para agilizar el uso de la aplicación, evitando tener que configurarlo todo otra vez en cada ejecución.

- **SharedPreferences** [148, 149]: Permite almacenar datos pequeños entre sesiones de la aplicación, de forma que los datos se conservan incluso si la aplicación se cierra o se reinicia. Su implementación es sencilla y está pensada para datos simples como enteros, cadenas de texto o booleanos, pero no es tan recomendable su uso para almacenar gran cantidad de datos o datos que sean muy complejos. También es destacable el hecho de que los datos almacenados en SharedPreferences no están cifrados, por lo que tampoco es recomendable utilizarlo para almacenar datos críticos o sensibles.

- **SQLite** [150, 151]: Es un sistema de gestión de bases de datos que tiene integración con Android y que permite a aplicaciones almacenar y gestionar datos estructurados. Se caracteriza por no requerir conexión a Internet, pudiendo crear una base de datos en el almacenamiento del dispositivo sin necesidad de servidores externos. Tiene funciones para consultar, insertar, actualizar o eliminar datos dentro de las tablas creadas en la base de datos.

Las características de cada una de estas herramientas hacen que una buena opción para el desarrollo de la aplicación sea usar ambas, almacenando datos sencillos, como booleanos que indiquen qué configuración ha establecido el usuario en los ajustes en SharedPreferences y datos estructurados, como por ejemplo el conjunto de datos necesarios para establecer la comunicación MQTT en SQLite.

## 5.3 Desarrollo de la Interfaz de Usuario de la Aplicación

En este apartado se tratarán las interfaces desarrolladas, explicando los elementos incluidos en ellas y justificando su localización y diseño para hacer intuitivo el uso de la aplicación.

Cabe destacar que el desarrollo de interfaces ha sido parejo con el desarrollo de la lógica detrás de la aplicación, no se ha diseñado toda la interfaz desde cero y después se ha programado su uso. Solo se explica al comienzo de esta parte del trabajo para poder comprender más fácilmente el código que se mostrará más adelante al ya haber visto el aspecto e intención de uso de la interfaz.

El desarrollo de interfaces en Android Studio se realiza en código XML en el editor que se puede ver en la figura 42. Existe la opción de o bien escribir el código directamente, o bien arrastrar elementos al recuadro que representa la interfaz y colocarlos ahí, lo que genera automáticamente el código, pero suele resultar en interfaces descuadradas y con elementos situados en posiciones que no son las más adecuadas. Por ello, se desarrolló todo el código XML a mano.

Dado que la aplicación está pensada para poder ser usada tanto en tablets como en móviles, se han creado 2 archivos XML por cada interfaz, una versión para pantallas más pequeñas y otra para más grandes.

- **Interfaz común (Interfaz de la actividad)**: Como se mencionó en el apartado anterior, la aplicación se basará en una actividad principal que

albergue distintos fragmentos. Esto significa que todos los fragmentos tendrán una parte común de la interfaz, y esa parte común será la app bar superior y el menú desplegable lateral **(figuras 59, 60 y 61)**.

La vista de dichos elementos desde el editor de interfaces de Android Studio se ve en la figura 59.

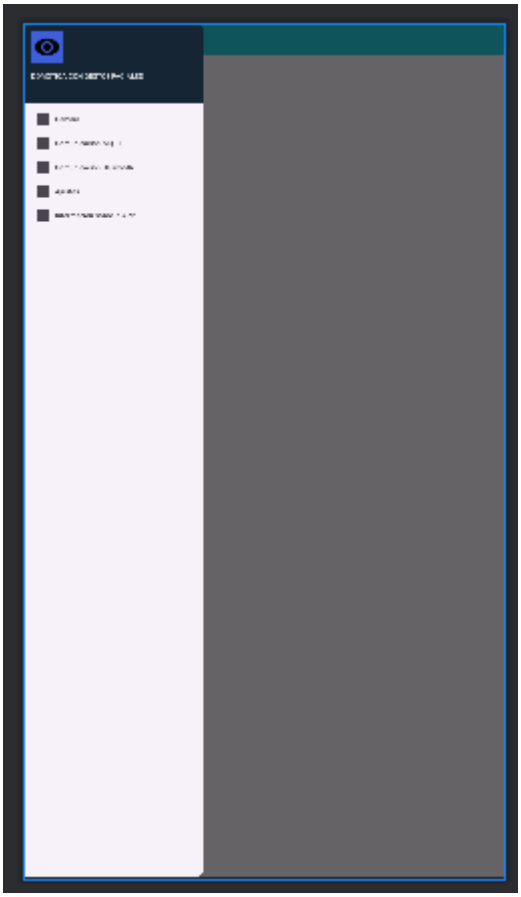


*Figura 59. Interfaz común a todos los fragmentos vista en Android Studio*

La vista desde la aplicación para versión de Tablet se ve en la figura 60 y desde la versión de móvil, en la figura 61.

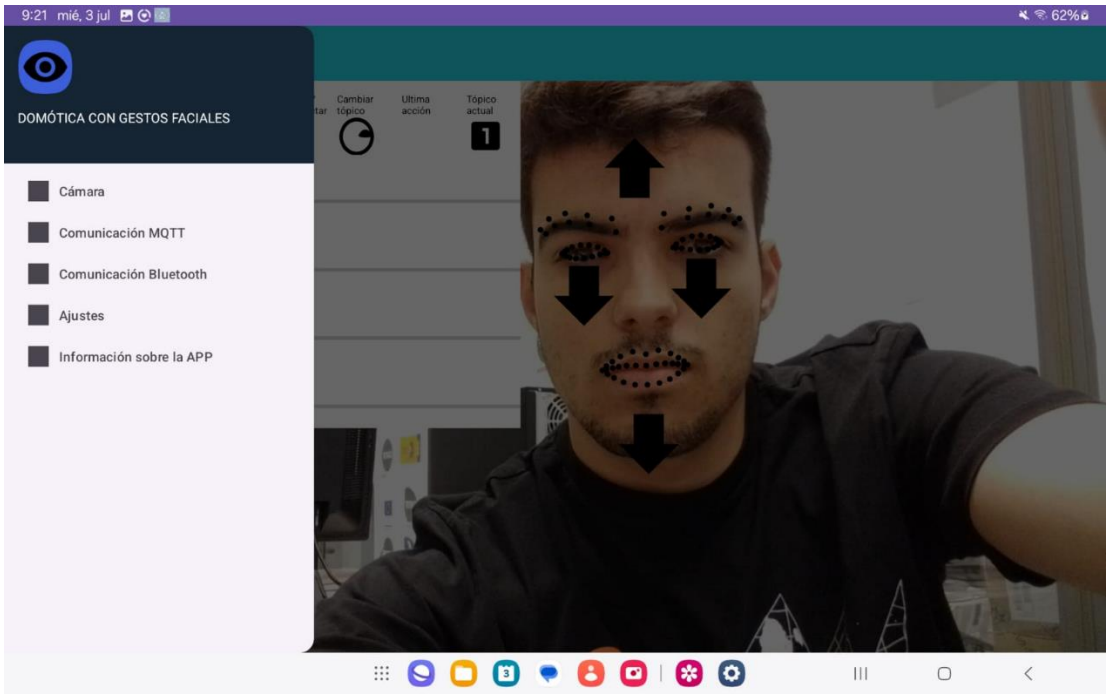



*Figura 60. Interfaz común a todos los fragmentos vista en la aplicación en la Tablet*

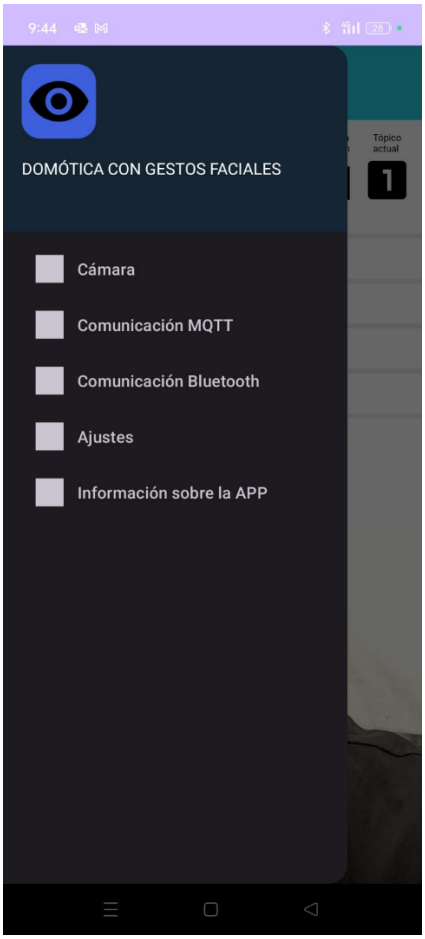



*Figura 61. Interfaz común a todos los fragmentos vista en la aplicación en móvil*

El menú lateral se despliega al pulsar en la app bar superior y accederá solo el asistente, al no requerir el usuario entrar en ningún fragment que no sea el de la cámara.

- **<u>Interfaz del fragment de la vista de la cámara</u>**: Se ideó de modo que el usuario pueda ver su rostro y las indicaciones de estado de la aplicación y de acciones que se están realizando de forma simultánea **(figuras 62 y 63)**.

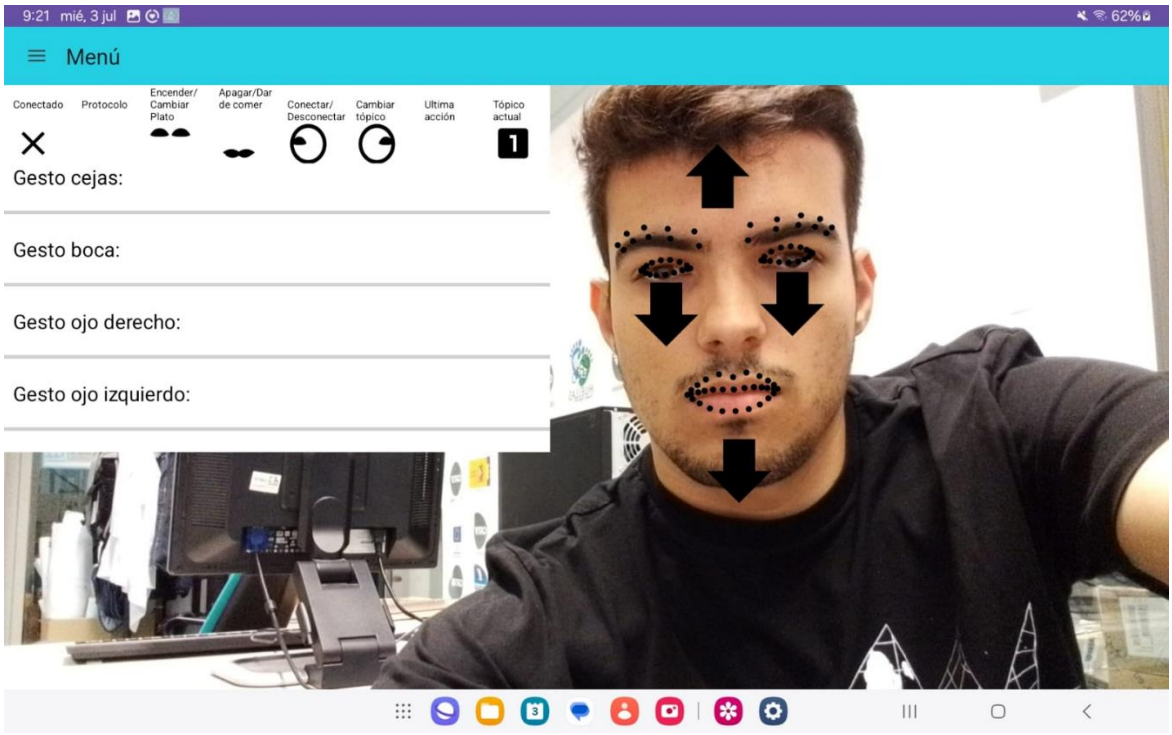



*Figura 62. Interfaz de la vista de la cámara para Tablet*


*Figura 63. Interfaz de la vista de la cámara para móvil*

Esta interfaz cuenta con 8 iconos para indicar al usuario información sobre la configuración actual, que de izquierda a derecha simbolizan:

1. Si se ha establecido conexión ya sea por Bluetooth o por MQTT.
2. El protocolo de comunicación que se está usando (si no se está conectado siquiera no sale ningún icono).
3. El gesto configurado para encender dispositivos por MQTT o para ordenar al OBI cambiar de plato por Bluetooth.
4. El gesto configurado para apagar dispositivos por MQTT o para ordenar al OBI dar de comer por Bluetooth.
5. El gesto configurado para conectarse ya sea por MQTT o por Bluetooth en caso de no estar conectado ya, y para desconectarse en caso de estar conectado.
6. El gesto configurado para cambiar el tópico de MQTT en el que se publica.
7. La última acción realizada (el último mensaje enviado).
8. El tópico actual de MQTT en el que se publica.

Además de los iconos, se han establecido 4 barras de progreso, las cuales indican durante cuanto tiempo se ha realizado un gesto. Se han puesto barras de progreso porque los mensajes se enviarán solamente después de mantener un gesto durante 3 segundos consecutivos para evitar enviar mensajes inintencionadamente, por ejemplo, al abrir la boca para masticar o al parpadear. Así, cuando estas barras se llenen es cuando sucederá la acción asignada al gesto.

Como se ve en la figura 63, existen indicaciones que se dibujan sobre la cara del usuario, como las flechas que marcan cómo realizar gestos (subiendo cejas, abriendo la boca o cerrando un ojo) y los puntos que marcan la localización de los landmarks faciales detectados.

- **Interfaz del fragment de la comunicación MQTT**: Diseñada para introducir los valores necesarios para establecer conexión con el servidor **(figuras 64 y 65)**.

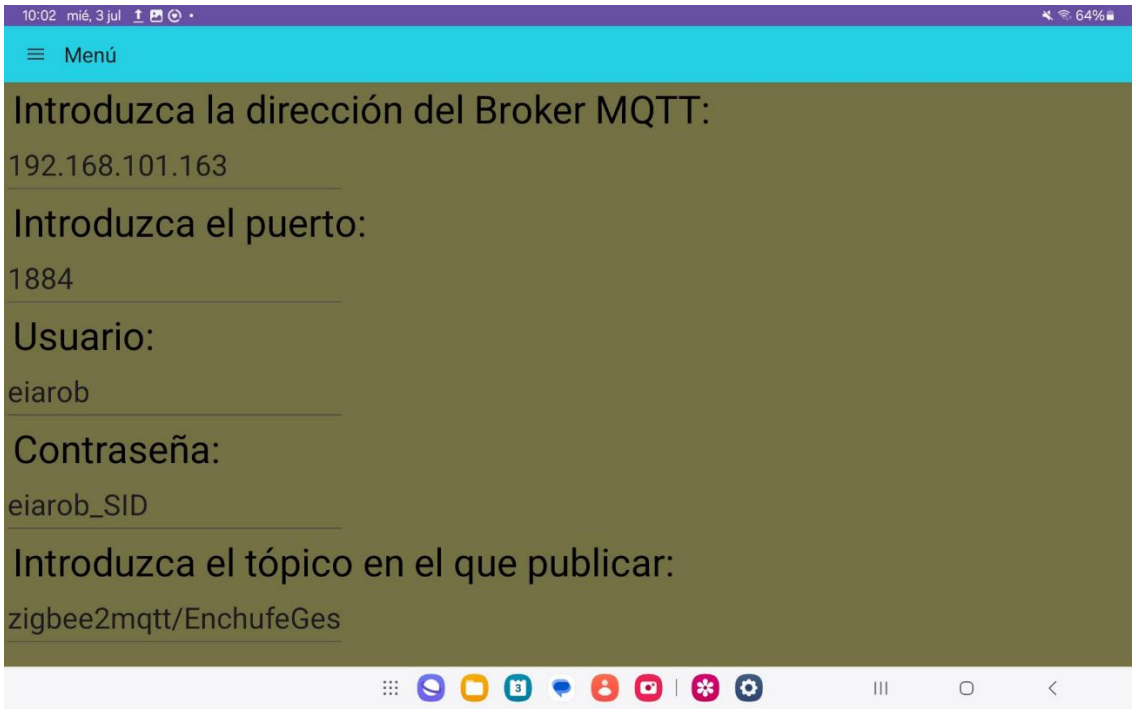



*Figura 64. Parte superior de la interfaz para la comunicación MQTT*

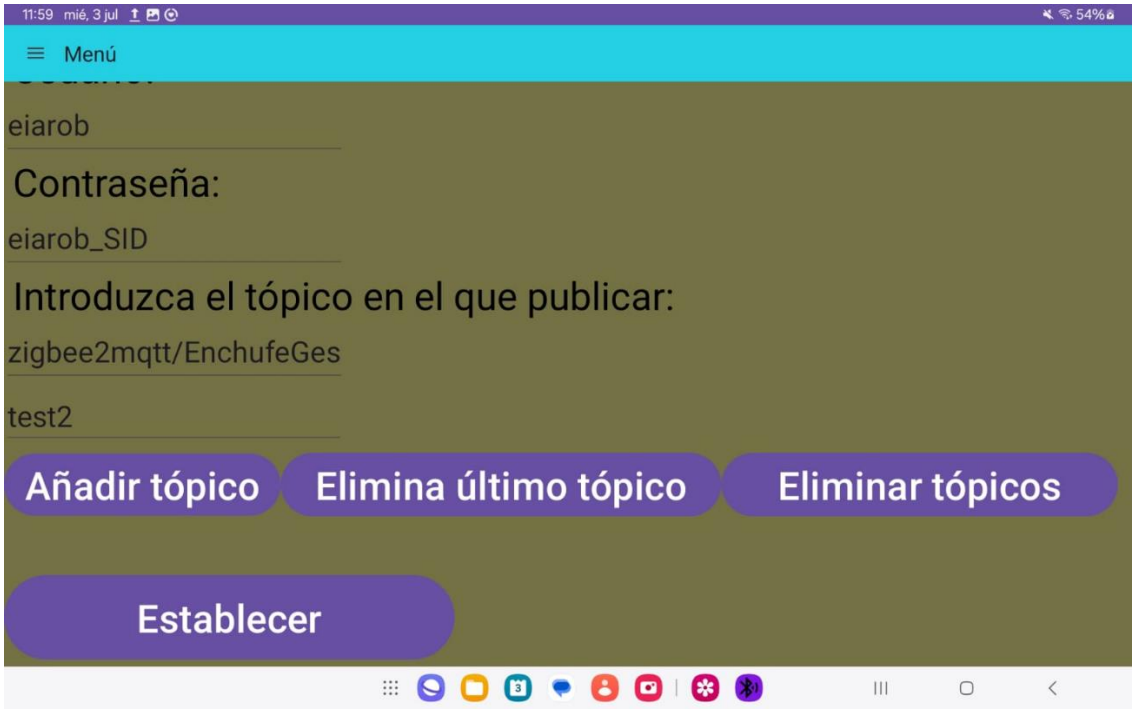



*Figura 65. Parte inferior de la interfaz para la comunicación MQTT*

Cuenta con varios “textViews” y “editTexts”, las nomenclaturas de Android para texto que el usuario no puede editar y para los apartados en los que el usuario puede introducir texto, respectivamente.

Se han puesto 2 capturas ya que esta interfaz no podría ser lo suficientemente corta como para entrar en la pantalla por todos los parámetros a introducir, así

que se hizo una interfaz deslizable, es lo que en Android se denomina “ScrollView” [152], esto permite subir o bajar la vista arrastrando el dedo por la pantalla.

Al bajar la pantalla se llega al apartado de tópicos, en el cual se configura una pila vertical de tópicos en la que unos botones permiten añadir un nuevo editText en la parte inferior en el que escribir el nombre de un nuevo tópico, eliminar el último tópico (el situado más abajo) y eliminar todos los tópicos.

Por último, para poder establecer la comunicación MQTT por primera vez, es necesario pulsar el botón “Establecer”.

La interfaz de móvil es muy similar, solo cambia el tamaño de los textos, hechos más pequeños para entrar en la pantalla del móvil, por eso no se muestra en ninguna figura.

- **Interfaz del fragment de la comunicación Bluetooth**: La conexión por Bluetooth no requiere la introducción de parámetros por parte del asistente, pero sigue necesitando la activación del Bluetooth del dispositivo, la vinculación con el dispositivo con el que comunicarse y el lanzamiento de petición de conexión.

Para poder realizar todas estas acciones, se ha añadido la interfaz que se ve en la **figura 66**.

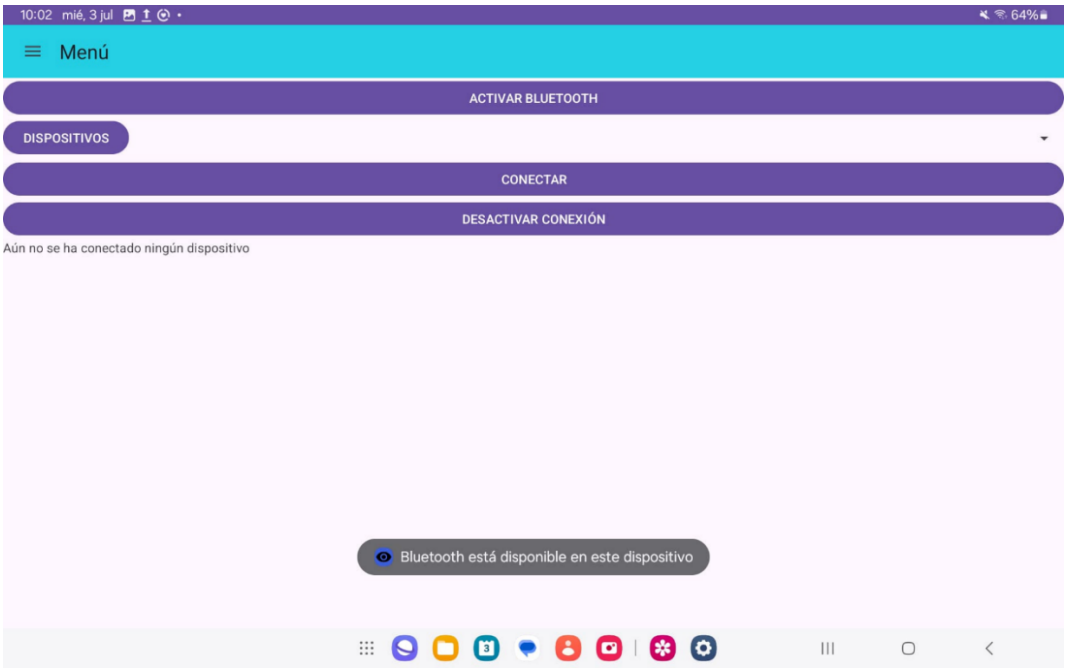



*Figura 66. Interfaz del fragment de la comunicación Bluetooth*

El botón situado en el extremo superior activará la comunicación Bluetooth del dispositivo en el que se ejecuta la app en caso de que no estuviera ya activo. El segundo botón mostrará a su derecha los dispositivos vinculados para seleccionar con cuál se quiere comunicar. El botón siguiente (en el que se lee “Conectar”) es el que establece la conexión como tal, y el último botón es para cortar dicha conexión.

- **Interfaz del fragment de ajustes**: La mayoría de las opciones de ajustes determinarán qué elementos se ven o no se ven en la interfaz de usuario de la vista de la cámara, por lo que, en la lógica del programa, serán variables booleanas. Para cambiar el valor de una variable booleana, que solo puede ser o verdadero o falso, le bastará al asistente con pulsar un botón, por lo que la interfaz de los ajustes se basará en una serie de botones **(figuras 67, 68 y 69)**.

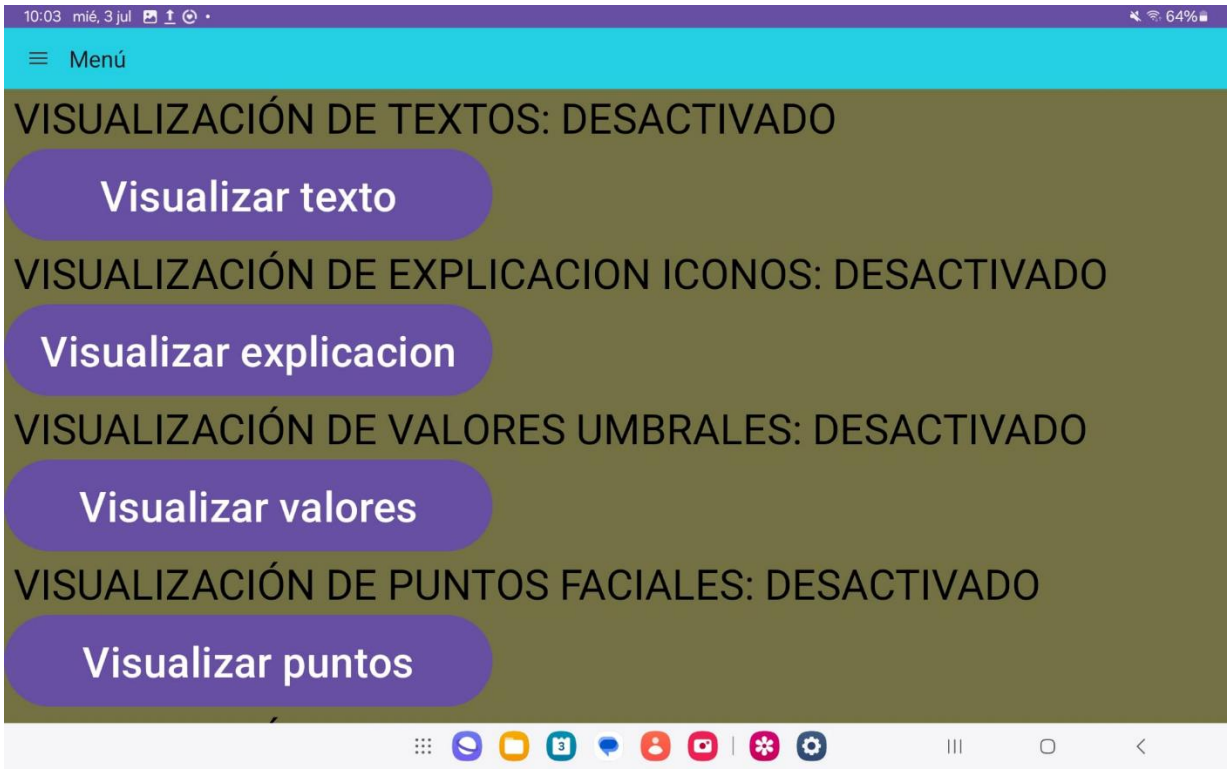



*Figura 67. Primera parte de la interfaz del fragment de ajustes*

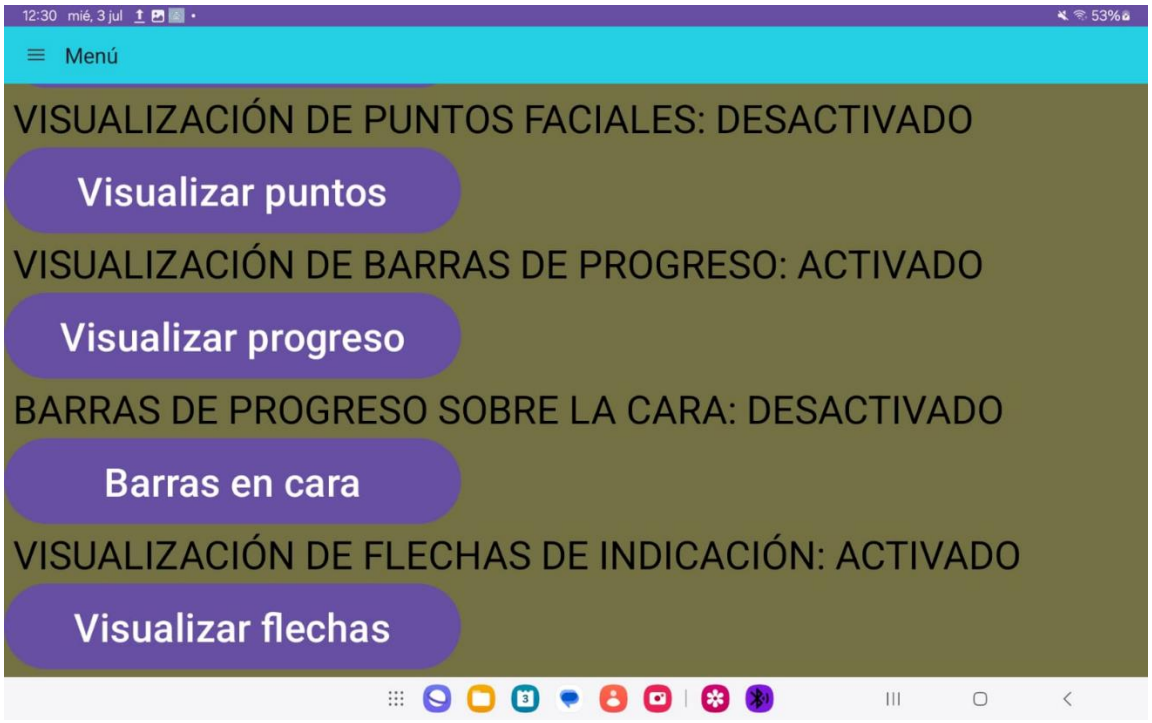



*Figura 68. Segunda parte de la interfaz del fragment de ajustes*

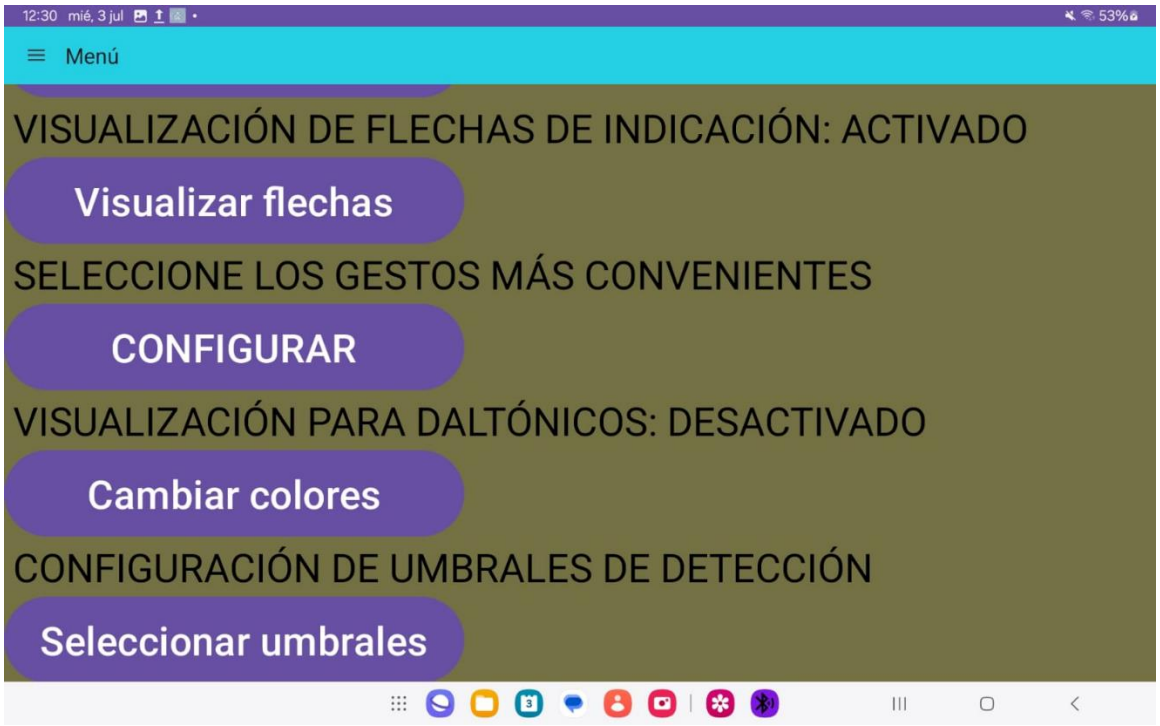



*Figura 69. Tercera parte de la interfaz del fragment de ajustes*

Al igual que en el fragment de comunicación MQTT, una sola vista de pantalla se queda pequeña para la interfaz, por eso, también se configuró como un ScrollView (de ahí que todo el interfaz ocupe 3 imágenes).

Cada botón (a excepción del de selección de gestos y el de configuración de los umbrales) cambia el valor de una variable booleana cuando es pulsado. Así, si por ejemplo se desea dejar de visualizar las flechas que se sitúan sobre la cara del usuario en la vista de la cámara, el asistente pulsará el botón de visualización de flechas y se pondrá en “false”, lo que hará que se dejen de ver en la cámara.

Existen 2 excepciones: el botón de configuración de gestos, que asocia cada gesto con cada acción, y el botón de configuración de umbrales, que permite ajustar la sensibilidad de la detección de gestos, de modo que si la realización de un gesto, como abrir la boca, le resulta incómodo al usuario, se puede ajustar el valor umbral de distancia que debe abrirla para que se detecte el gesto como realizado.

Estas 2 configuraciones no se pueden gestionar simplemente pulsando un botón por ser más complejas que el resto, por lo que se han creado 2 ventanas emergentes, llamadas “dialogs” [153] en Android, que se mostrarán al pulsar esos botones **(figuras 70 y 71)**.

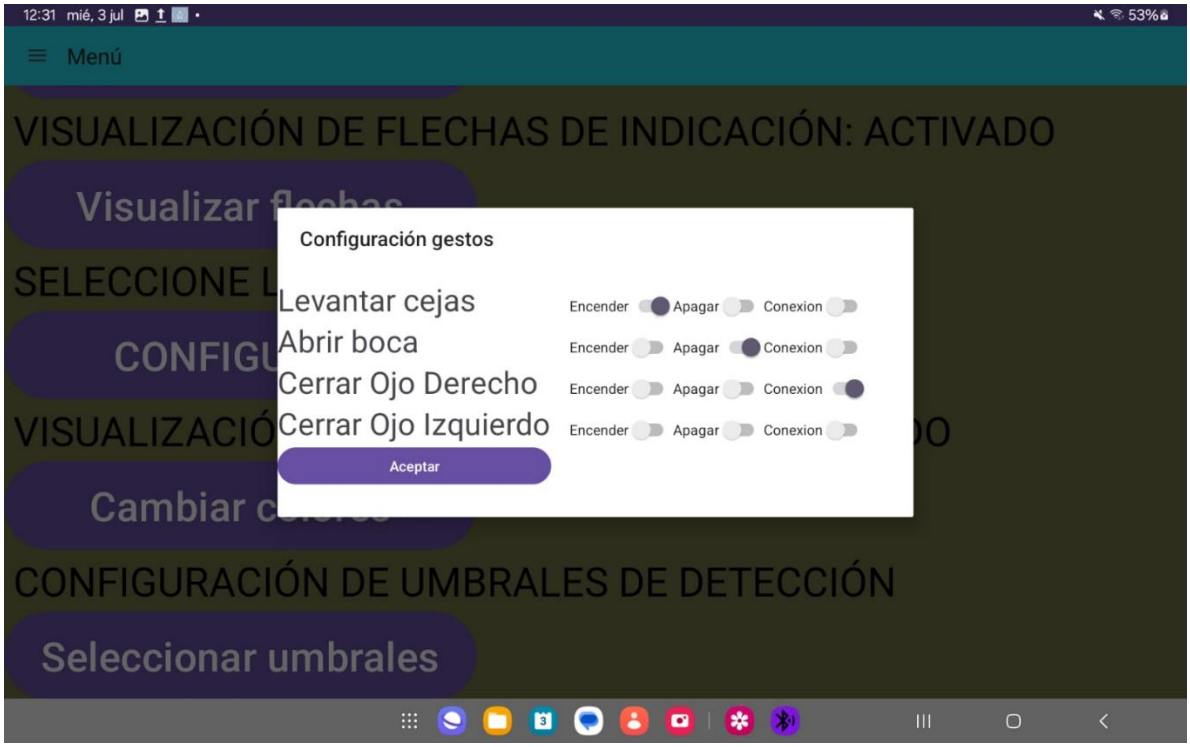



*Figura 70. Ventana de configuración de gestos*

El diseño de esta ventana emergente consiste en switches, o interruptores, que el asistente puede accionar para seleccionar qué acción se asocia a cada gesto. El nombre del gesto con el que comienza cada fila será el gesto configurado y el switch activado (la lógica del programa se asegurará de que solamente 1 de cada fila esté activo en cada momento) determinará la acción que realizará este gesto.

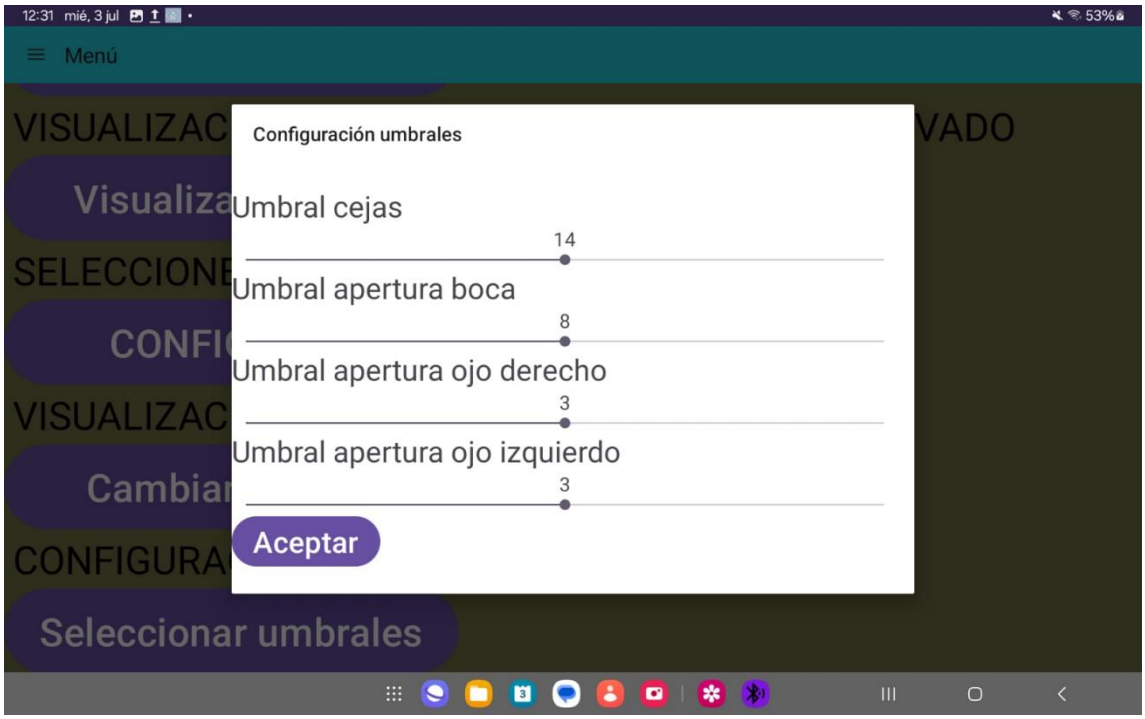



*Figura 71. Ventana de configuración de umbrales de detección*

Esta ventana muestra una serie de barras deslizantes, o "*seekBars*" como las denomina Android, que pueden ser arrastradas hacia la izquierda o hacia la derecha por el asistente para establecer el porcentaje del tamaño de la cara que se deben separar los dos puntos considerados en el cálculo de la distancia a partir del cual se considerará el gesto como realizado.

Se ha establecido el porcentaje del tamaño de la cara como criterio para que la distancia entre la cara del usuario y la pantalla no afectara a la detección de gestos. Esto asegura que tanto si el usuario está muy cerca de la pantalla como si está muy alejado, se detectarán correctamente los gestos realizados.

- **Interfaz del fragment de información sobre la aplicación**: La interfaz de este fragmento será la más sencilla, al no requerir ningún elemento interactivo como botones o barras de progreso. Su diseño serán simplemente textos en los que se darán instrucciones para usar la aplicación **(figura 72)**.

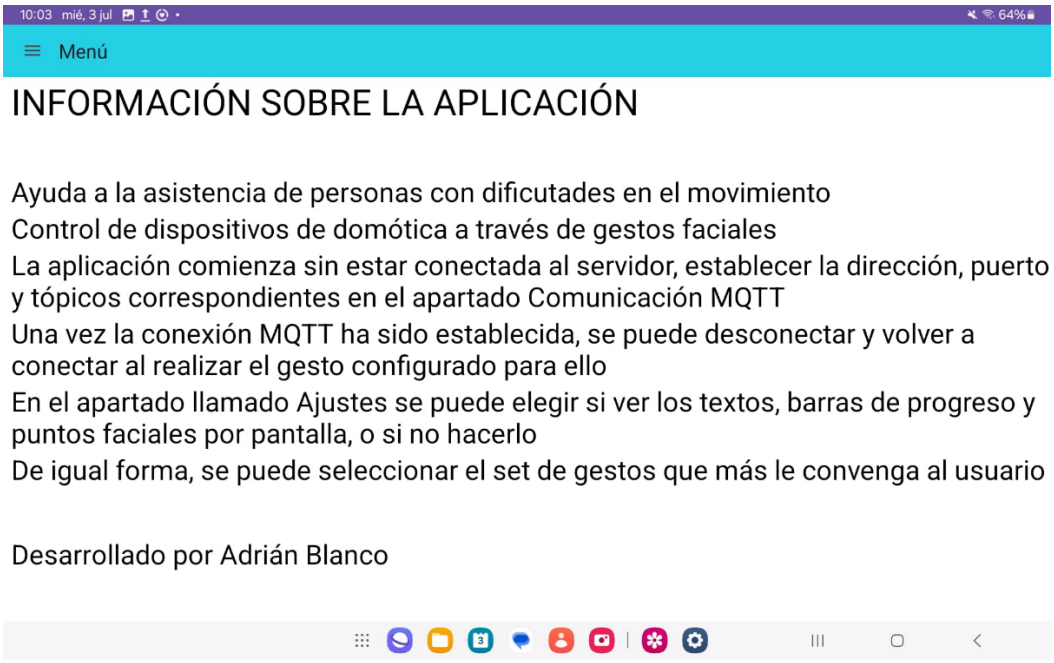



*Figura 72. Interfaz del fragment de información sobre la aplicación*

## 5.4 Implementación de la actividad principal y menú

Antes de comenzar la explicación cabe decir que en todos los ficheros de código en Android hay que importar una serie de paquetes para poder usar ciertas funciones, pero no es código que defina el funcionamiento de la aplicación. Por ese motivo, ese código para importar no será incluido en las explicaciones. También puede ser necesaria la implementación de librerías externas incluyéndolas en los archivos de construcción de la aplicación, como build.gradle.

Una vez se han visto las interfaces finales del programa, se puede comenzar el análisis del código, y el primer paso en el desarrollo de la aplicación fue implementar la actividad que almacena a los distintos fragmentos. También al comienzo se incluyeron los permisos a usar en el AndroidManifest y las librerías a implementar en el archivo de construcción de la aplicación build.gradle, pero ese código se tratará en su debido momento.

El código que se ejecuta al crear la “MainActivity” es mostrado en la **tabla 3**.

```
//Creación de la actividad
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    //Llamada a la función permisos para solcitar permisos para bases de datos y Bluetooth
```

```
    permisos()

    //Asignación de la app bar de la interfaz con la variable toolbar
    val toolbar:Toolbar = findViewById(R.id.toolbar_main)
    setSupportActionBar(toolbar)

    // Cargar el fragment inicial debajo del Toolbar
    supportFragmentManager.beginTransaction()
        .replace(R.id.fragment_container, InitialFragment())
        .commit()

    //Se inicializa y configura la vista de fragmentos
    drawer = findViewById(R.id.drawer_layout)

    toggle = ActionBarDrawerToggle(this, drawer, toolbar, R.string.navigation_drawer_open,
R.string.navigation_drawer_close)
    drawer.addDrawerListener(toggle)

    supportActionBar?.setDisplayHomeAsUpEnabled(true)
    supportActionBar?.setHomeButtonEnabled(true)

    //Se configura navigationView para manejar la navegación con el menú
    val navigationView: NavigationView = findViewById(R.id.nav_view)
    navigationView.setNavigationItemSelectedListener(this)


    //Inicialización de Base de datos
    conexionBBDDSQLite = ConexionBBDDSQLite(this)

    //Se crean las tablas de usuario y de topicos
    conexionBBDDSQLite!!.createTable_t_usuarios()
    conexionBBDDSQLite!!.createTable_t_topics()

    //Si la tabla está vacía (como al crearla), se introduce un usuario vacío
    if(conexionBBDDSQLite!!.check_t_usuarios()==0){
        val usuarios = llenarUsuarios()
        if(usuarios.isNotEmpty()){
            usuarios.forEach { usuario ->
                conexionBBDDSQLite!!.insert_t_usuarios(usuario.dns, usuario.puerto,
usuario.nombre, usuario.contrasena)
            }
        }
    }

    if(conexionBBDDSQLite!!.check_t_topics()==0){
        if(llenarTopics().size!=0){
            llenarTopics().forEach {
                conexionBBDDSQLite!!.insert_t_topics(it)
            }
        }
    }
}
```

*Tabla 3. Actividad principal de la aplicación*

La primera instrucción realizada al crear la actividad es el establecimiento de la interfaz de dicha actividad con “setContentView”. Tras ello, se solicitan los permisos necesarios para la ejecución de la aplicación al invocar a la función “permisos()”, la cual se mostrará en la tabla 6. También se configura el Toolbar que se mostrará en la parte superior de la pantalla.

Hasta ahora se ha cargado una interfaz con una appbar pero debajo de la misma no hay nada, por ello y para poder usar la aplicación sin entrar a más fragmentos, se usa un FragmentManager que hace que se vea el fragmento de cámara nada más arrancar la aplicación, el llamado “InitialFragment”.

Se configura la navegación del menú lateral y se inicializa la base de datos, en la que se crean 2 tablas, una para almacenar usuarios (con los parámetros para la comunicación MQTT) y otra en la que almacenar el nombre de los tópicos. En caso de que dichas tablas estén vacías (lo que ocurre al crearlas), se inserta un usuario con campos vacíos para luego poder modificarlos más fácilmente en la tabla de usuarios y un tópico de prueba en la tabla de tópicos.

Estas funciones de inserción de usuario y tópico se ven en la **tabla 4**.

```
//He creado la "estructura" MeterUsuario para introducir las diferentes columnas de la
tabla de usuarios
data class MeterUsuario(
   val dns: String,
   val puerto: Int,
   val nombre: String,
   val contrasena: String
)
   //Funciones de bases de datos para empezar con el usuario y un tópico ya en la DB
   fun llenarUsuarios(): ArrayList<MeterUsuario>{
      var arrayUsuarios: ArrayList<MeterUsuario> = ArrayList()
      arrayUsuarios.add(MeterUsuario("", 0, "", ""))
      return arrayUsuarios
   }

   fun llenarTopics(): ArrayList<String>{
      var arrayTopics: ArrayList<String> = ArrayList()
      arrayTopics.add("TopicPrueba")
      return arrayTopics
   }
```

*Tabla 4. Inserción de usuario y tópico en tablas de la base de datos*

Tras la creación de la actividad como tal, se debe especificar una función que permita la navegación entre fragmentos al clicar en el menú (**tabla 5**).

```
//Esta es la función que permitirá navegar entre fragmentos con el menú
override fun onNavigationItemSelected(item: MenuItem): Boolean {
   when(item.itemId) {
      //Cada uno de las 5 opciones del menú lateral
      R.id.nav_item_one -> {
```

```
            // Cargar el fragment de la cámara
            supportFragmentManager.beginTransaction()
                .replace(R.id.fragment_container, InitialFragment())
                .addToBackStack(null)
                .commit()
        }
        R.id.nav_item_two -> {
            // Cargar el fragment de configuración MQTT
            supportFragmentManager.beginTransaction()
                .replace(R.id.fragment_container, MQTTConfigFragment())
                .addToBackStack(null)
                .commit()
        }
        R.id.nav_item_three -> {
            // Cargar el fragment de configuración Bluetooth
            supportFragmentManager.beginTransaction()
                .replace(R.id.fragment_container, BTConfigFragment())
                .addToBackStack(null)
                .commit()
        }
        R.id.nav_item_four -> {
            // Cargar el fragment de ajustes
            supportFragmentManager.beginTransaction()
                .replace(R.id.fragment_container, UsageInfoFragment())
                .addToBackStack(null)
                .commit()
        }

        R.id.nav_item_five -> {
            // Cargar el fragment de información sobre la APP
            supportFragmentManager.beginTransaction()
                .replace(R.id.fragment_container, AppInfoFragment())
                .addToBackStack(null)
                .commit()
        }
    }
    drawer.closeDrawer(GravityCompat.START)
    return true
}
```

*Tabla 5. Navegación entre fragmentos al seleccionar una opción del menú*

"R.id.nav_item_one" identifica al recuadro de la interfaz en el que se ve la primera opción del menú, y al clicar sobre ella se invoca a" supportFragmentManager" para que cambie la interfaz actualmente mostrada por pantalla. Lo mismo sucede para el resto de opciones.

Ese es todo el código de funcionamiento de la aplicación principal, consiste en la creación de su interfaz, la petición de permisos necesarios para la ejecución del programa, la creación de las tablas de la base de datos, la generación del fragmento de cámara debajo de la barra superior y la gestión de la navegación a través del menú lateral con el cambio del fragmento visible debajo de la barra.

En cuanto a la petición de los permisos, el usuario debe permitir que la aplicación tenga acceso a funcionalidades del dispositivo que por motivos de seguridad no se conceden directamente. La petición se hace con la función mostrada en la **tabla 6**.

```
private fun permisos() {

    if (ContextCompat.checkSelfPermission(
            this,
            Manifest.permission.READ_SMS
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        ActivityCompat.requestPermissions(
            this, arrayOf(
                Manifest.permission.BLUETOOTH,
                Manifest.permission.BLUETOOTH_CONNECT,
                Manifest.permission.BLUETOOTH_ADMIN,
                Manifest.permission.BLUETOOTH_SCAN,
                Manifest.permission.BLUETOOTH_ADVERTISE,
                Manifest.permission.BLUETOOTH_PRIVILEGED,
                Manifest.permission.MANAGE_DEVICE_POLICY_BLUETOOTH,
                Manifest.permission.MANAGE_EXTERNAL_STORAGE,
                Manifest.permission.READ_EXTERNAL_STORAGE,
                Manifest.permission.WRITE_EXTERNAL_STORAGE
                ),
            1
        )
    } else {
        Log.d("Bluetooth", "Todos los permisos concedidos ")
    }

    if (Build.VERSION.SDK_INT >= 30) {
        if (!Environment.isExternalStorageManager()) {
            try {
                var uri = Uri.parse("package:" + packageName)

                var intent =
Intent(Settings.ACTION_MANAGE_APP_ALL_FILES_ACCESS_PERMISSION, uri)
                startActivity(intent);
            } catch (ex: Exception) {
                var intent = Intent();
                intent.setAction(Settings.ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION)
                startActivity(intent);
            }
        }
    }
}
```

*Tabla 6. Función de solicitud de permisos*

Los permisos también deben incluirse en el archivo Manifest para que la aplicación pueda usar esas funcionalidades.

## 5.5 Implementación del uso de la cámara del dispositivo

Ya explicada la actividad que gestiona los cambios de fragmento, se puede comenzar la explicación del código del fragmento de la cámara.

Se ve el código que se ejecuta al crear el fragmento y la función que se ejecuta al destruirlo en la **tabla 7**.

```
//Función que asocia todos los elementos del layout con el código mediante binding
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    binding = FragmentInitialBinding.inflate(inflater,container,false)
    return binding.root
}

//Función de ejecución del programa (de ejecución de este fragment más bien)
@RequiresApi(Build.VERSION_CODES.TIRAMISU)
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    cameraExecutor = Executors.newSingleThreadExecutor()
    requestPermission()
    defaultDetector = FaceMeshDetection.getClient(FaceMeshDetectorOptions.Builder()
        .build())
}

//Función para cuando se cierra el fragment
override fun onDestroyView() {
    super.onDestroyView()
    // Liberar recursos al cambiar de fragmento
    cameraProvider?.unbindAll()
    defaultDetector.close()
    cameraExecutor.shutdown()
}
```

*Tabla 7. Funciones de ciclo de vida del fragmento de la cámara*

Al igual que en el MainActivity, la primera acción es crear la interfaz en el "onCreate" (función que se ejecuta para crear el fragmento). Una vez iniciado, en el "onViewCreated", se crea un servicio de ejecución de la cámara y el detector de malla de puntos faciales.

En caso de destruirse el fragmento (ya sea por cambiar a otro o por cerrar la aplicación), se liberan recursos al cerrar la cámara y el detector en la función "onDestroyView".

Aunque se haya iniciado el fragmento y los servicios de cámara y detección de malla facial, se debe otorgar acceso a la cámara para poder usarla **(tabla 8)**.

```
//Para pedir el permiso de acceso a cámara la primera vez que se usa la aplicación
@RequiresApi(Build.VERSION_CODES.TIRAMISU)
private fun requestPermission(){
   requestPermissionIfMissing{granted->
      if(granted)
         startCamera()
      else
         Toast.makeText(requireContext(),"Por favor, acepta la solicitud",
Toast.LENGTH_LONG).show()
   }
}

//Para volver a pedir permiso si el usuario dice que no
private fun requestPermissionIfMissing(onResult:((Boolean)->Unit)){
   if(ContextCompat.checkSelfPermission(requireContext(),Manifest.permission.CAMERA)
== PackageManager.PERMISSION_GRANTED)
      onResult(true)
   else
      registerForActivityResult(ActivityResultContracts.RequestPermission()){
         onResult(it)
      }.launch(Manifest.permission.CAMERA)
}
```

*Tabla 8. Solicitud de permisos de acceso a cámara*

Una vez se ha concedido acceso a la cámara, se usa la siguiente función en la **tabla 9** para usarla.

```
//Para usar la cámara del dispositivo
@RequiresApi(Build.VERSION_CODES.TIRAMISU)
private fun startCamera(){
   if(!isAdded){
      return
   }
   val processCameraProvider = ProcessCameraProvider.getInstance(requireContext())
   processCameraProvider.addListener({
      try {
         if(!isAdded){
            return@addListener
         }
         val cameraProvider = processCameraProvider.get()
         val previewUseCase = buildPreviewUseCase()
         val imageAnalysisUseCase = buildImageAnalysisUseCase()
         cameraProvider.unbindAll()
         cameraProvider.bindToLifecycle(this,
CameraSelector.DEFAULT_FRONT_CAMERA,previewUseCase, imageAnalysisUseCase)
      } catch (e: Exception) {
      }
   },ContextCompat.getMainExecutor(requireContext()))
}
```

```
//Para poder mostrar por pantalla la vista de la cámara, y para usar esa vista al dibujar
private fun buildPreviewUseCase(): Preview {
   return
Preview.Builder().build().also{it.setSurfaceProvider(binding.previewView.surfaceProvider
)}
}

//Función en la que hago todos los cálculos y detección de gestos
@RequiresApi(Build.VERSION_CODES.TIRAMISU)
private fun buildImageAnalysisUseCase(): ImageAnalysis {
```

*Tabla 9. Función de uso de la cámara y de análisis de la imagen*

La función “startCamera”, como indica su nombre, se encarga de comenzar el uso de la cámara. Lo hace mediante su llamada a “CameraProvider.bindToLifecycle”, en la que se puede ver cómo se selecciona la cámara frontal del dispositivo (la que apunta hacia el usuario) con “CameraSelector.DEFAULT_FRONT_CAMERA”.

La función “buildImageAnalysisUseCase” que se ve al final de la tabla 9 es sumamente importante, por tratarse de la función en la que se realiza todo el análisis de la imagen, incluyendo la detección de malla facial y cálculos de distancias entre los puntos de dicha malla, a partir de los cuales se determina la realización de gestos. Su código se analizará más adelante.

## 5.6 Implementación de ML Kit para el reconocimiento de puntos faciales

Para la detección de malla facial, se decidió emplear la herramienta Face Mesh de la librería ML Kit. Si bien la librería Mediapipe también está desarrollada por Google, es compatible con Android y detecta la posición de las pupilas, su uso no está optimizado para la integración en dispositivos móviles [154], lo que puede causar problemas de rendimiento que afecten a la experiencia de uso. A este hecho hay que añadir que ML Kit es más fácil de implementar en Android que Mediapipe, lo que proporciona un código más entendible y fácil de mantener y expandir.

Fue necesario incluir ML Kit en la aplicación a través de los archivos de Gradle **(tabla 10)**.

```
//Para face mesh de MLKit
implementation("com.google.mlkit:face-mesh-detection:16.0.0-beta1")
```

*Tabla 10. Implementación de la librería ML Kit en el archivo build.gradle*

Ya incorporada la librería, se crea una instancia de detector de malla facial en el onViewCreated del fragmento de la cámara, lo cual se mostró en la tabla 7.

Dentro de la función de análisis de imágenes de la tabla 9 se ejecuta el detector de malla facial **(tabla 11)**.

```
//Cálculos con los resultados del detector de malla facial
defaultDetector.process(inputImage)
   .addOnSuccessListener { result ->
      if(result.isEmpty()){
         Log.d("FaceMesh", "No se han detectado caras")

         //Para dejar de dibujar la cara
         mActivity?.runOnUiThread {
            val overlayView = binding.overlayView
            overlayView.background = null
         }

         //Para detener los gestos detectados en el último frame
         cejas = 1 // Valor por defecto para cejas
         boca = 1  // Valor por defecto para boca
         posBoca = 1 // Valor por defecto para posición de boca
         ojoDer = 1 // Valor por defecto para ojo derecho
         ojoIzq = 1 // Valor por defecto para ojo izquierdo

         //Para detener las animaciones de las barras de progreso
         detenerAnimacionesYReiniciarBarras()

         //Para evitar que se hagan acciones aún no habiendo cara
         detenerHandlers()

      } else {
         for (faceMesh in result) {

            //Rectángulo alrededor de la cara (no lo uso)
            val bounds: Rect = faceMesh.boundingBox

            //Para vaciar los arrays de posiciones entre una ejecución y otra
            posicionesX.clear()
            posicionesY.clear()

            //En esta variable almaceno todos los puntos detectados
            val faceMeshpoints = faceMesh.allPoints
            //Para usarlo en el dibujado de las flechas
            val positionFlechas = faceMeshpoints[10].position
            val positionFlechas2 = faceMeshpoints[152].position
            val dy = positionFlechas.y-positionFlechas2.y
            val tamano = -dy*0.01


            //Llamada a la función que mete las coordenadas X e Y de cada punto en dos
arrays distintos
            obtenerPosiciones(faceMeshpoints, posicionesX, posicionesY)
```

*Tabla 11. Gestión de resultados del detector de malla facial*

La instrucción "defaultDetector.process(inputImage).addOnSuccessListener" devuelve el resultado de la detección de mallas faciales. En caso de no haber detectado ninguna, se muestra por Logcat un mensaje informando sobre ello y se evita la realización de acciones asociadas a gestos cortando la ejecución de las herramientas "handlers" [155], que son aquellas que se aseguran de no enviar mensajes a menos que no se haya realizado un gesto durante 3 segundos consecutivos.

En caso de sí haber localizado mallas faciales, el detector devuelve todos los 468 puntos que detecta ML Kit, los que se muestran en la figura 19, a través de "faceMesh.allPoints". Dichos puntos están ordenados en un array, con el índice de cada punto siendo igual que en la herramienta Face Mesh de Mediapipe, salvo por los puntos del 468 al 478, los respectivos a las pupilas, ya que ML Kit no los puede detectar. Se usa el mismo sistema de numeración que el que se muestra en la figura 20, por ser ML Kit y Mediapipe ambos desarrollados por Google.

Siguiendo ese sistema de numeración, el punto 10 está situado en el extremo superior de la cara y el punto 152 en el extremo inferior. Calculando la distancia entre las coordenadas Y de esos 2 puntos, se obtiene el tamaño vertical de la cara detectada, lo que se empleará en la determinación de gestos al ser los valores umbrales porcentajes del tamaño de la cara.

Para facilitar cálculos con las coordenadas de los puntos de la malla, se usa la función de la **tabla 12** para almacenar las coordenadas X en un array y las coordenadas Y en otro.

```
//Función que introduce las coordenadas de los puntos faciales a dos arrays
fun obtenerPosiciones(faceMeshpoints: MutableList<FaceMeshPoint>, posicionesX:
MutableList<Float>, posicionesY: MutableList<Float>){
   for ((index,faceMeshpoint) in faceMeshpoints.withIndex()) {
      val position = faceMeshpoint.position
      posicionesX.add(position.x)
      posicionesY.add(position.y)
   }
}
```

*Tabla 12. Almacenamiento de todas las posiciones en listas propias*

## 5.7 Determinación de gestos realizados a partir de la posición de los puntos faciales detectados

Una vez se dispone de la localización de los puntos de la malla, se deben calcular las distancias entre los puntos situados en posiciones de la cara en los que intervengan los gestos faciales a detectar y puntos que no se vean afectados por dicho gesto. Es decir, si por ejemplo se quiere analizar la realización del gesto de subir las cejas, se debe calcular la distancia entre un punto como el borde superior de la ceja, que se ve desplazado por el gesto, y otro como un punto situado en la mejilla del usuario, que permanece en la misma posición.

La fusión empleada para el cálculo de dichas distancias se ve en la **tabla 13**.

```
//Función que usa las coordenadas X e Y de ciertos puntos faciales para saber distancias
entre cejas y ojos, por ejemplo
fun calcularDistancia(operacion: Int, posicionesX: MutableList<Float>, posicionesY:
MutableList<Float>): Float{

  var dx: Float
  var dy: Float

  when(operacion){
    0-> {
      dx = posicionesX[152]-posicionesX[10]
      dy = posicionesY[152]-posicionesY[10]
    }
    1-> {
      dx = posicionesX[145]-posicionesX[52]
      dy = posicionesY[145]-posicionesY[52]
    }
    2-> {
      dx = posicionesX[14]-posicionesX[13]
      dy = posicionesY[14]-posicionesY[13]
    }
    3-> {
      dx = posicionesX[146]-posicionesX[132]
      dy = posicionesY[146]-posicionesY[132]
    }
    4-> {
      dx = posicionesX[375]-posicionesX[361]
      dy = posicionesY[375]-posicionesY[361]
    }
    5->{
      dx = posicionesX[145]-posicionesX[159]
      dy = posicionesY[145]-posicionesY[159]
    }
    else-> {
      dx = posicionesX[374]-posicionesX[386]
      dy = posicionesY[374]-posicionesY[386]
    }
  }
```

```
    var distanciaExt = sqrt((dx * dx) + (dy * dy))

    return distanciaExt
}
```

*Tabla 13. Cálculo de distancias entre puntos clave de la malla facial*

La instrucción “when” de Kotlin funciona de forma similar a la instrucción “switch” de muchos lenguajes de programación, como C++, de modo que si por ejemplo la variable operacion tiene valor 2, se accede al apartado 2. Se divide la función en 6 apartados para poder invocarla después para cada distancia a tener en cuenta para detectar gestos **(tabla 14)**.

```
//Llamadas a la función calcularDistancia, con el resultado determinaremos si se realizan, o no, gestos
var distanciaExt = calcularDistancia(0, posicionesX, posicionesY)
var distanciaCejas = calcularDistancia(1, posicionesX, posicionesY)
var distanciaBoca = calcularDistancia(2, posicionesX, posicionesY)
var distanciaBocaDer = calcularDistancia(3, posicionesX, posicionesY)
var distanciaBocaIzq = calcularDistancia(4, posicionesX, posicionesY)
var distanciaOjoDer = calcularDistancia(5, posicionesX, posicionesY)
var distanciaOjoIzq = calcularDistancia(6, posicionesX, posicionesY)
```

*Tabla 14. Invocaciones de la función calcularDistancia*

La primera distancia es aquella entre los extremos superior e inferior de la cara (por eso en la tabla 13 el “when 0” usa los puntos 10 y 152 en el cálculo).

Para “when 1”, el punto 52 se sitúa en la ceja derecha y el 145 en el párpado inferior, de modo que, al subir las cejas, esta distancia aumentará. Lo mismo sucede para el resto de distancias, que la distancia entre los puntos considerados varía al realizar el gesto y esa variación es la que se usa para determinar el gesto.

En el caso del levantamiento de cejas y de la apertura de la boca, la distancia entre puntos debe ser mayor que el umbral, pero en el cierre de los ojos se tiene en cuenta la distancia entre párpados, por lo que se realiza el cierre de ojos cuando esa distancia sea menor que el umbral.

En la **tabla 15** se muestra una de las comparaciones de las distancias calculadas con los umbrales de detección, además de las acciones que se toman tras la comparación, tanto en caso de que no supere el umbral como si sí lo supera.

```
if((distanciaGesto/distanciaExt)>(UsageInfoFragment.umbralCejas*0.01)){
            seRealiza = 0
            if(seRealiza==0){
                handlerCejas.postDelayed(runnableCejasSubidas, 3000)
            } else {
                handlerCejas.removeCallbacks(runnableCejasSubidas)
            }
        } else{
```

```
            seRealiza = 1
            handlerCejas.removeCallbacksAndMessages(null)
            binding.progressBarCejas.progress = 0
            ObjectAnimator.ofInt(binding.progressBarCejas,"progress", 500)
                .setDuration(3000)
                .start()
            binding.progressBarCejas2.progress = 0
            ObjectAnimator.ofInt(binding.progressBarCejas2,"progress", 500)
                .setDuration(3000)
                .start()
            binding.textView6.text = "Gesto cejas:"
            binding.textViewProgressBarCejas2.text = "Gesto cejas:"
        }
```

*Tabla 15. Comparación de distancias entre puntos con umbrales de detección*

En caso de que la distancia no sea mayor que la del umbral, se elimina la publicación de mensajes y se paran las barras de progreso asociadas a ese gesto.

En caso de que lo sea, se ejecutará el conjunto de instrucciones llamado "runnableCejasSubidas", que se ve en la **tabla 16**.

```
val runnableCejasSubidas = Runnable {
            binding.textView6.text = "Cejas levantadas durante 5 segundos consecutivos"
            binding.textViewProgressBarCejas2.text = "Cejas levantadas durante 5 segundos
consecutivos"
            transcurridoDesdeEnviado = System.currentTimeMillis() - ultimoEnviado
            transcurridoDesdeConexion = System.currentTimeMillis() - ultimaConexion
            transcurridoDesdeBT = System.currentTimeMillis() - ultimoBT
            transcurridoDesdeCambio = System.currentTimeMillis() - ultimoCambio
            if((MQTTConfigFragment.conectado || BTConfigFragment.m_isConnected) &&
UsageInfoFragment.cejasEncender){
                if(MQTTConfigFragment.conectado && transcurridoDesdeEnviado >= 2000){
                    cejasObject.put("state","ON")
                    publish(MQTTConfigFragment.publishTopic, cejasObject.toString())
                    lastPublishedMessage = "EnviadoON"
                    ultimoEnviado = System.currentTimeMillis()
                }
                if(BTConfigFragment.m_isConnected && transcurridoDesdeBT >= 5000){
                    sendCommand("A")
                    lastPublishedMessage = "Cambio Plato"
                    ultimoBT = System.currentTimeMillis()
                }
            }
            else if((MQTTConfigFragment.conectado || BTConfigFragment.m_isConnected)
&& UsageInfoFragment.cejasApagar){
                if(MQTTConfigFragment.conectado && transcurridoDesdeEnviado >= 2000){
                    cejasObject.put("state","OFF")
                    publish(MQTTConfigFragment.publishTopic, cejasObject.toString())
                    lastPublishedMessage = "EnviadoOFF"
```

```
                    ultimoEnviado = System.currentTimeMillis()
                }
                if(BTConfigFragment.m_isConnected && transcurridoDesdeBT >= 5000){
                    sendCommand("B")
                    lastPublishedMessage = "Dar de Comer"
                    ultimoBT = System.currentTimeMillis()
                }
            }
                //Intento de conectar MQTT con gestos
            else if(MQTTConfigFragment.conectado && transcurridoDesdeConexion >=
5000 && UsageInfoFragment.cejasConexion){
                MQTTConfigFragment().desconexionMQTT()
                Toast.makeText(requireContext(),"Desconectado de
MQTT",Toast.LENGTH_SHORT).show()
                ultimaConexion = System.currentTimeMillis()
            }
            else if(BTConfigFragment.m_isConnected && transcurridoDesdeConexion
>=5000 && UsageInfoFragment.cejasConexion){
                BTConfigFragment().disconnect()
                Toast.makeText(requireContext(),"Desconectado de
Bluetooth",Toast.LENGTH_SHORT).show()
                ultimaConexion = System.currentTimeMillis()
            }
            else if(!MQTTConfigFragment.conectado && !BTConfigFragment.m_isConnected
&& transcurridoDesdeConexion >= 5000 && UsageInfoFragment.cejasConexion){
                Log.d("Intento MQTT", "Entró al if de conexión con gesto")
                if (!MQTToBT) { //Si lo último conectado fue BT
                    MQTToBT = true //Porque se intentó acceder a MQTT, aunque no se haya
metido direcciones se pone para que el siguiente sea Bluetooth

                    //Para ver si puedo conectarme sin entrar en MQTTConfigFragment
                    MQTTConfigFragment.direccion =
ConexionBBDDSQLite(requireContext()).getDnsById(1).toString()
                    MQTTConfigFragment.puerto =
ConexionBBDDSQLite(requireContext()).getPuertoById(1).toString()
                    MQTTConfigFragment.serverUri =
"tcp://${MQTTConfigFragment.direccion}:${MQTTConfigFragment.puerto}"
                    MQTTConfigFragment.usuarioMQTT =
ConexionBBDDSQLite(requireContext()).getUserNameById(1).toString()
                    MQTTConfigFragment.contrasenaMQTT =
ConexionBBDDSQLite(requireContext()).getContrasenaById(1).toString()
                    Log.d("IntentoConexionMQTT", "URI:${MQTTConfigFragment.serverUri}")
                    Toast.makeText(requireContext(),"Intentando conexión
MQTT",Toast.LENGTH_SHORT).show()


MQTTConfigFragment().conexionMQTT(requireContext(),MQTTConfigFragment.serverUr
i) { exitosa ->
                        if(exitosa){
                            val handler = Handler(Looper.getMainLooper())
                            handler.post {
                                Toast.makeText(requireContext(), "Conexión MQTT exitosa",
Toast.LENGTH_LONG).show()
                            }
```

```
                    } else {
                        val handler = Handler(Looper.getMainLooper())
                        handler.post {
                            Toast.makeText(requireContext(), "Conexión MQTT fallida, vuelve a
intentarlo", Toast.LENGTH_LONG).show()
                        }
                    }
                }
            } else {
                MQTToBT = false
                val BTpreferences =
requireContext().getSharedPreferences("BTPreferences", Context.MODE_PRIVATE)
                BTConfigFragment.m_address =
BTpreferences.getString("direccionBTGuardada", "").toString()
                Log.d("Bluetooth", "DireccionBT obtenida del preferences:
${BTConfigFragment.m_address}")
                Toast.makeText(requireContext(),"Intentando conexión
Bluetooth",Toast.LENGTH_SHORT).show()
                if(BTConfigFragment.m_address!=""){
                    var mBtAdapter =
(requireContext().getSystemService(Context.BLUETOOTH_SERVICE) as
BluetoothManager).adapter
                    val device: BluetoothDevice =
mBtAdapter.getRemoteDevice(BTConfigFragment.m_address)
                    if (ActivityCompat.checkSelfPermission(requireContext(),
Manifest.permission.BLUETOOTH_CONNECT) != PackageManager.PERMISSION_GRANTED)
{
                        ActivityCompat.requestPermissions(
                            requireActivity(), arrayOf(
                                Manifest.permission.BLUETOOTH,
                                Manifest.permission.BLUETOOTH_CONNECT,
                                Manifest.permission.BLUETOOTH_ADMIN,
                                Manifest.permission.BLUETOOTH_SCAN,
                                Manifest.permission.BLUETOOTH_ADVERTISE,
                                Manifest.permission.BLUETOOTH_PRIVILEGED,
                                Manifest.permission.MANAGE_DEVICE_POLICY_BLUETOOTH,
                            ), 1)
                    }
                    connectToDevice(device)
                } else {
                    Toast.makeText(requireContext(),"No hay dirección de dispositivo BT
registrada",Toast.LENGTH_SHORT).show()
                }
            }
            ultimaConexion = System.currentTimeMillis()
        }
        else if(transcurridoDesdeCambio>=5000 &&
UsageInfoFragment.cejasCambiar){
            if(topicoActual<MQTTConfigFragment.numeroTopics){
                topicoActual=topicoActual+1
            } else {
                topicoActual=1
            }
            ultimoCambio = System.currentTimeMillis()
```

```
        }
    }
```

*Tabla 16. Código que se ejecuta al haberse cumplido la realización de un gesto*

Es un código extenso, pero eso se debe al hecho de que debe considerar todas las posibles configuraciones de gestos. Es decir, el gesto de levantar las cejas podría ser el que envíe un mensaje de encendido/cambio de plato, el de apagado/dar de comer, el de conexión/desconexión o el que cambie de tópico, y, dentro de esas 4 configuraciones posibles, se debe tener en cuenta que si por ejemplo no está conectado y subir las cejas es la acción de conexión/desconexión, enviará el mensaje de conexión, pero si ya estaba conectado enviará el mensaje de desconexión. Esta situación da lugar a un gran número de condicionales dentro del código a ejecutar al realizarse un gesto, motivo por el que es tan largo.

El mismo código se empleará en caso de realizarse cualquier otro gesto, solo que adaptando las variables como “cejasEncender” (que se configura en los ajustes y representa que el gesto de levantar cejas está asociado al encendido por MQTT o al cambio de plato del robot OBI por Bluetooth) por “bocaEncender”, por ejemplo.

A lo largo de los puntos 5.8 y 5.9 se profundizará más en el código de la tabla 16 para explicar el manejo de la comunicación MQTT y Bluetooth.

## 5.8 Configuración y manejo de la conexión MQTT

De igual forma que ocurría con la detección de puntos faciales y ML Kit, es necesario incluir una librería externa en el archivo build.gradle, que gestiona la construcción de la app, para poder implementar una comunicación por MQTT, y la librería es Eclipse Paho [89] **(tabla17)**.

```
//Para MQTT
implementation("androidx.legacy:legacy-support-v4:1.0.0")
implementation("com.github.hannesa2:paho.mqtt.android:4.3.beta1")
```

*Tabla 17. Implementación de la librería Eclipse Paho en el build.gradle*

Una vez implemetada la librería Paho, se debe establecer la conexión MQTT, para ello se usa la función de la **tabla 18**.

```
fun conexionMQTT(context: Context, serverUri:String, callback:(Boolean)->Unit) {

    try {
```

```
        //Esto lo añado para que no se conecte 2 veces y me salgan dobles mensajes en el
logcat
        desconexionMQTT()

        mqttClient = MqttAndroidClient(context, serverUri, clientId)
        mqttClient!!.setCallback(object : MqttCallbackExtended {


            /*** Entrega completa de mensajes al broker ****/
            override fun connectComplete(reconnect: Boolean, serverURI: String) {
                if (reconnect) {
                    Log.d("Reconexion", "Reconectado")
                } else {
                    Log.d("Reconexion", "No reconectado")
                }
            }

            /*** Perdió la conexión con el broker. ****/
            override fun connectionLost(cause: Throwable?) {
                Log.d("ERROR","Conexión perdida")
            }


            /**** Recibir nuevos mensajes del Broker ***/
            override fun messageArrived(topic: String, message: MqttMessage) {
                Log.d("LLEGA EL MENSAJE DEL BROKER","MENSAJE RECIBIDO")
                Log.d("MENSAJE:",message.toString())

                requireActivity().runOnUiThread {
                    binding.textView2.text = "Mensaje recibido: ${message.toString()}"
                }

            }

            /*** Se llama cuando se ha completado la entrega de un mensaje y se han recibido
todas las confirmaciones ***/
            override fun deliveryComplete(token: IMqttDeliveryToken) {
                Log.d("Éxito","MENSAJE ENVIADO")
            }
        })

        /**** Opciones de conexión MQTT ***/
        val mqttConnectOptions = MqttConnectOptions()
        mqttConnectOptions.isAutomaticReconnect = true
        mqttConnectOptions.isCleanSession = false
        mqttConnectOptions.keepAliveInterval = 50
        mqttConnectOptions.setUserName(usuarioMQTT)
        mqttConnectOptions.setPassword(contrasenaMQTT.toCharArray())

        //Desconectar primero BT
        BTConfigFragment().disconnect() //Para desconectar BT
        Log.d("Bluetooth","Desconectado")
```

```
        /*** Conexión con el Broker ****/
        mqttClient!!.connect(mqttConnectOptions, null, object : IMqttActionListener {

            /*** Este método se invoca cuando una acción se ha completado correctamente.***/
            @RequiresApi(Build.VERSION_CODES.TIRAMISU)
            override fun onSuccess(asyncActionToken: IMqttToken) {
                /*** Se crea la suscripción ***/

                Log.d("MQTT","Éxito, Conectado")


                //Para la lógica de conexiones
                conectado = true
                InitialFragment.MQTToBT = true

                callback(true)
            }

            /**** Este método se invoca cuando se produce un error en una acción. ****/
            override fun onFailure(asyncActionToken: IMqttToken?, exception: Throwable?) {
                Log.d("Error MQTT","No conectado")
                Log.d("Error MQTT",exception.toString())
                conectado = false
                callback(false)
            }
        })

    } catch (e: MqttException) {
        Log.e(TAG, "Error excepción la clase MqttExceptiom.")
        callback(false)
    } catch (e: Exception) {
        Log.e(TAG, "Error en la conexión MQTT.")
        callback(false)
    }
}
```

*Tabla 18. Función de conexión para MQTT*

Primero se desconecta la comunicación actual MQTT, en caso de que la conexión que se quiere establecer se trate de una actualización de una anterior al haber añadido más tópicos o al haber cambiado el usuario, por ejemplo.

También se desconecta la comunicación Bluetooth en caso de que existiera para evitar situaciones complejas e indeseadas al controlar 2 dispositivos diferentes mediante 2 protocolos de comunicación distintos.

La función usa los parámetros introducidos en las bases de datos a través de los editText de la interfaz que se vio en la figura 65 para crear el cliente MQTT, en cuyo interior se definen una serie de funciones que se ejecutan en eventos como al llegar un mensaje (messageArrived) o al perder la conexión (conectionLost). Posteriormente, se intenta conectar al servidor MQTT con “mqttClient!!.connect”,

y en caso de haberse conectado correctamente, devuelve un valor booleano true, en caso contrario, un false.

Esta función es invocada o bien al pulsar el botón de establecer conexión en la interfaz del fragmento de MQTT o al realizar el gesto configurado para conectarse al servidor MQTT.

Una vez conectado, y para enviar un mensaje de encendido, se accede al siguiente código (**tabla 19**), el cual ya era visible en la tabla 16, pero aquí será explicado en mayor profundidad.

```
transcurridoDesdeEnviado = System.currentTimeMillis() - ultimoEnviado
transcurridoDesdeConexion = System.currentTimeMillis() - ultimaConexion
transcurridoDesdeBT = System.currentTimeMillis() - ultimoBT
transcurridoDesdeCambio = System.currentTimeMillis() - ultimoCambio
if((MQTTConfigFragment.conectado || BTConfigFragment.m_isConnected) &&
UsageInfoFragment.cejasEncender){
   if(MQTTConfigFragment.conectado && transcurridoDesdeEnviado >= 2000){
      cejasObject.put("state","ON")
      publish(MQTTConfigFragment.publishTopic, cejasObject.toString())
      lastPublishedMessage = "EnviadoON"
      ultimoEnviado = System.currentTimeMillis()
   }
```

*Tabla 19. Envío de mensajes de encendido por MQTT*

Dentro de ese código que se ejecuta al completar un gesto durante 3 segundos, lo primero que se hace es actualizar la variable de tiempo transcurrido que se emplea para no enviar una gran cantidad de mensajes al servidor MQTT al hacer los gestos de encendido y apagado a la vez, lo que podría resultar perjudicial para los dispositivos controlados. Así, para enviar un mensaje de encendido por MQTT al realizar un gesto, como por ejemplo levantar las cejas, primero es necesario que:

- Haya conexión con el servidor MQTT

- Que el gesto de levantar las cejas esté asociado al envío del mensaje de encendido

- Que haya transcurrido un tiempo mínimo entre mensajes para no enviar muchos mensajes de encendido y apagado muy seguidos en caso de mantener ambos gestos realizados

Una vez se cumplen todas estas condiciones, se realiza el envío con la publicación en el tópico actual del mensaje, y se actualiza la variable que almacena el instante de tiempo en el que se produjo el último envío de mensaje.

Se observa también en el código de la tabla 19 que el mensaje publicado se denomina “cejasObject” y se debe convertir a cadenas de caracteres (string) cuando es enviado. Ese mensaje está en formato JSON [156], lo que significa

que está conformado por una pareja de elementos, uno el nombre de una variable y el otro el valor de dicha variable, escritos de forma {nombre:valor}. Se usa este formato por ser el que entienden un gran número de dispositivos inteligentes IoT, como el enchufe que se usó en el desarrollo.

## 5.9 Configuración y manejo de la conexión Bluetooth

En el caso de los mensajes transmitidos usando protocolo MQTT, los dispositivos inteligentes controlados vienen configurados de fábrica para interpretar los mensajes recibidos. Por ejemplo, el enchufe inteligente usado en el desarrollo deja pasar corriente para encender el dispositivo enchufado si recibe un mensaje JSON cuyo valor sea “ON”, y si el mensaje recibido es un OFF, corta el paso de dicha corriente.

Sin embargo, en el caso del microcontrolador ESP32 usado para la comunicación Bluetooth, será necesario desarrollar un programa que se encargue de interpretar los mensajes recibidos desde el dispositivo Android. Así, esta comunicación requerirá del desarrollo de código tanto para el emisor como para el receptor del mensaje.

### 5.9.1 Configuración y envío de mensajes Bluetooth desde la app

A diferencia de la comunicación MQTT, la comunicación Bluetooth no requiere de librerías externas para ser programada en Android, de modo que no será necesario modificar el archivo build.gradle para usar Bluetooth en la aplicación. Sin embargo, sí que es necesario añadir en el archivo AndroidManifest, que es el que gestiona los permisos a usar, código que especifique qué permisos de Bluetooth se van a utilizar **(tabla 20)**.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH_ADVERTISE" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

*Tabla 20. Permisos necesarios para el uso de Bluetooth en la aplicación*

Su función es darle acceso a la aplicación a funcionalidades como poder rastrear dispositivos Bluetooth cercanos (con el permiso SCAN), ser detectable para otros dispositivos (con ADVERTISE) y poder comunicarse con dispositivos vinculados (con CONNECT). Para poder ser utilizado en dispositivos con versiones de Android anteriores a la 11, son necesarios los permisos ADMIN, COARSE_LOCATION y FINE_LOCATION [157].

Una vez se tiene acceso a las funcionalidades Bluetooth del dispositivo, se puede establecer conexión con aquellos dispositivos que estén vinculados por Bluetooth con el que ejecuta la app. Es decir, primero se debe vincular, de forma externa a la aplicación, los dispositivos a controlar, ya que por motivos de seguridad Android no permite vincular dispositivos desde dentro de una aplicación.

De esta forma, primero se escoge el dispositivo con el que conectarse de entre los vinculados. Para ello, se usa el código que se ejecuta al pulsar el botón en el que se lee “Dispositivos” en la interfaz del fragmento de la configuración Bluetooth, visible en la figura 66. Dicho código se muestra en la **tabla 21**.

```
//Boton dispositivos emparejados
idBtnDispBT.setOnClickListener {
    if (mBtAdapter.isEnabled) {

        val pairedDevices: Set<BluetoothDevice>? = mBtAdapter?.bondedDevices
        mAddressDevices!!.clear()
        mNameDevices!!.clear()

        pairedDevices?.forEach { device ->
            val deviceName = device.name
            val deviceHardwareAddress = device.address // MAC address
            mAddressDevices!!.add(deviceHardwareAddress)
            //.......... EN ESTE PUNTO GUARDO LOS NOMBRE A MOSTRARSE EN EL COMBO BOX
            mNameDevices!!.add(deviceName)
        }

        //ACTUALIZO LOS DISPOSITIVOS
        idSpinDisp.setAdapter(mNameDevices)
    } else {
        val noDevices = "Ningun dispositivo pudo ser emparejado"
        mAddressDevices!!.add(noDevices)
        mNameDevices!!.add(noDevices)
        Toast.makeText(requireContext(), "Primero vincule un dispositivo bluetooth",
Toast.LENGTH_LONG).show()
    }
}
```

*Tabla 21. Código para mostrar los dispositivos vinculados*

En este código, se toma el nombre y la dirección MAC de cada dispositivo vinculado para incluirlos en una lista de la que el usuario pueda seleccionar con cuál conectarse. Dicha lista se identifica como “idSpinDisp” en el código y consiste en lo que Android denomina un *spinner*, una lista desplegable. Tras seleccionar el dispositivo desde la lista, se pulsa el botón que dice “Conectar” en la figura 66.

El código desarrollado para establecer la conexión Bluetooth al pulsar el botón de conectar se ve en la **tabla 22**.

```
idBtnConect.setOnClickListener {
   try {
      if ((m_bluetoothSocket == null || !m_isConnected) &&
idSpinDisp.selectedItemPosition!=-1) {
         val IntValSpin = idSpinDisp.selectedItemPosition
         m_address = mAddressDevices!!.getItem(IntValSpin).toString()
         Toast.makeText(requireContext(),m_address,Toast.LENGTH_LONG).show()
         // Cancelar la búsqueda de dispositivos para no ralentizar la conexión
         mBtAdapter?.cancelDiscovery()
         val device: BluetoothDevice = mBtAdapter.getRemoteDevice(m_address)
         m_bluetoothSocket =
device.createInsecureRfcommSocketToServiceRecord(m_myUUID)
         m_bluetoothSocket!!.connect()


         MQTTConfigFragment().desconexionMQTT()
         Log.i("MQTT", "DESCONECTADO")
         m_isConnected = true
         InitialFragment.MQTToBT = false
         Log.i("Bluetooth", "CONECTADO")

         //Para poder luego usarla en el InitialFragment
         Log.d("Bluetooth", "Direccion guardada en el preferences: ${m_address}")
         BTpreferences.edit().putString("direccionBTGuardada", m_address).apply()


         val textView = binding.textView
         textView.text = "Estableciendo conexión..."

         Log.d("Bluetooth", "UUID:${m_myUUID}")
         Log.d("Bluetooth", "Direccion:${m_address}")

         Toast.makeText(requireContext(),"CONEXION
EXITOSA",Toast.LENGTH_LONG).show()
         Log.i("Bluetooth", "CONEXION EXITOSA")

         //Para volver al fragment_initial en lugar de con un Intent
         val fragmentManager = requireActivity().supportFragmentManager
         val transaction = fragmentManager.beginTransaction()
         transaction.replace(R.id.fragment_container, InitialFragment())
         transaction.addToBackStack(null) // Si deseas agregar la transacción a la pila de
retroceso
         transaction.commit()
      } else {
         if(m_isConnected)
            Toast.makeText(requireContext(),"YA ESTÁ
CONECTADO",Toast.LENGTH_LONG).show()
         if(idSpinDisp.selectedItemPosition==-1)
            Toast.makeText(requireContext(),"NO HAY DISPOSITIVO
SELECCIONADO",Toast.LENGTH_LONG).show()
         Log.i("Bluetooth", "O YA ESTÁ CONECTADO, O NO HAY DISPOSITIVO
SELECCIONADO")
      }
   } catch (e: IOException) {
```

```
        //connectSuccess = false
        e.printStackTrace()
        Toast.makeText(requireContext(),"ERROR DE
CONEXION",Toast.LENGTH_LONG).show()
        Log.i("Bluetooth", e.toString())
    }
}
```

*Tabla 22. Código ejecutado al pulsar el botón de conexión en el fragmento de comunicación Bluetooth*

La conexión Bluetooth se realiza tomando como parámetro la dirección MAC del dispositivo vinculado con el que se conectará, y creando un socket (uno de los extremos de la comunicación) [158] que se usa para conectarse mediante la instrucción “m_BluetoothSocket!!.connect()”.

De igual forma que al establecer comunicación MQTT se desconectaba la comunicación Bluetooth, en caso de conectarse mediante Bluetooth, se ejecuta código para la desconexión de MQTT para no controlar dispositivos diferentes a la vez de forma indeseada.

La conexión se puede realizar o bien en el fragmento destinado a ello o en la vista de la cámara al realizar el gesto de conexión (misma situación para MQTT). El código ejecutado al realizar el gesto de conexión se ve en la **tabla 23**, y forma parte de lo visto en la tabla 16. Es el que define cómo se gestiona la conexión con 2 protocolos posibles.

```
else if(!MQTTConfigFragment.conectado && !BTConfigFragment.m_isConnected &&
transcurridoDesdeConexion >= 5000 && UsageInfoFragment.cejasConexion){
    Log.d("Intento MQTT", "Entró al if de conexión con gesto")
    if (!MQTToBT) { //Si lo último conectado fue BT
        MQTToBT = true //Porque se intentó acceder a MQTT, aunque no se haya metido
direcciones se pone para que el siguiente sea Bluetooth

        //Para ver si puedo conectarme sin entrar en MQTTConfigFragment
        MQTTConfigFragment.direccion =
ConexionBBDDSQLite(requireContext()).getDnsById(1).toString()
        MQTTConfigFragment.puerto =
ConexionBBDDSQLite(requireContext()).getPuertoById(1).toString()
        MQTTConfigFragment.serverUri =
"tcp://${MQTTConfigFragment.direccion}:${MQTTConfigFragment.puerto}"
        MQTTConfigFragment.usuarioMQTT =
ConexionBBDDSQLite(requireContext()).getUserNameById(1).toString()
        MQTTConfigFragment.contrasenaMQTT =
ConexionBBDDSQLite(requireContext()).getContrasenaById(1).toString()
        Log.d("IntentoConexionMQTT", "URI:${MQTTConfigFragment.serverUri}")
        Toast.makeText(requireContext(),"Intentando conexión
MQTT",Toast.LENGTH_SHORT).show()


MQTTConfigFragment().conexionMQTT(requireContext(),MQTTConfigFragment.serverUr
```

```
i) { exitosa ->
        if(exitosa){
            val handler = Handler(Looper.getMainLooper())
            handler.post {
                Toast.makeText(requireContext(), "Conexión MQTT exitosa",
Toast.LENGTH_LONG).show()
            }
        } else {
            val handler = Handler(Looper.getMainLooper())
            handler.post {
                Toast.makeText(requireContext(), "Conexión MQTT fallida, vuelve a intentarlo",
Toast.LENGTH_LONG).show()
            }
        }
    }
  } else {
    MQTToBT = false
    val BTpreferences = requireContext().getSharedPreferences("BTPreferences",
Context.MODE_PRIVATE)
    BTConfigFragment.m_address = BTpreferences.getString("direccionBTGuardada",
"").toString()
    Log.d("Bluetooth", "DireccionBT obtenida del preferences:
${BTConfigFragment.m_address}")
    Toast.makeText(requireContext(),"Intentando conexión
Bluetooth",Toast.LENGTH_SHORT).show()
    if(BTConfigFragment.m_address!=""){
      var mBtAdapter =
(requireContext().getSystemService(Context.BLUETOOTH_SERVICE) as
BluetoothManager).adapter
      val device: BluetoothDevice =
mBtAdapter.getRemoteDevice(BTConfigFragment.m_address)
      if (ActivityCompat.checkSelfPermission(requireContext(),
Manifest.permission.BLUETOOTH_CONNECT) != PackageManager.PERMISSION_GRANTED)
{
        ActivityCompat.requestPermissions(
          requireActivity(), arrayOf(
            Manifest.permission.BLUETOOTH,
            Manifest.permission.BLUETOOTH_CONNECT,
            Manifest.permission.BLUETOOTH_ADMIN,
            Manifest.permission.BLUETOOTH_SCAN,
            Manifest.permission.BLUETOOTH_ADVERTISE,
            Manifest.permission.BLUETOOTH_PRIVILEGED,
            Manifest.permission.MANAGE_DEVICE_POLICY_BLUETOOTH,
          ), 1)
      }
      connectToDevice(device)
    } else {
      Toast.makeText(requireContext(),"No hay dirección de dispositivo BT
registrada",Toast.LENGTH_SHORT).show()
    }
  }
  ultimaConexion = System.currentTimeMillis()
}
```

*Tabla 23. Código ejecutado al realizar el gesto de conexión*

Se decide si se conecta a MQTT o a BT al hacer el gesto en función del último protocolo usado. Se consigue gracias a la variable booleana llamada MQTToBT, la cual toma el valor true al haber intentado la conexión por MQTT y toma false al haber intentado la conexión por Bluetooth. Si la última conexión fue por MQTT y se vuelve a ejecutar el gesto de conexión, se intentará por Bluetooth, y si se intentó por Bluetooth la última vez, se intentará por MQTT.

Así, la realización del gesto de conexión/desconexión provoca el siguiente ciclo:

CONEXIÓN POR MQTT -> DESCONEXIÓN -> CONEXIÓN POR BLUETOOTH -> DESCONEXIÓN -> CONEXIÓN POR MQTT

En el código de la tabla 23 se aprecia también que cada conexión usa las funciones vistas en sus respectivos apartados. Para MQTT se usa la función "conexionQTT" vista en la tabla 18 tras tomar los datos necesarios desde la base de datos de usuario y tópicos, y para Bluetooth se usa la llamada a la función "connectToDevice" que toma como parámetro la dirección MAC del dispositivo Bluetooth con el que se conectó la última vez a través de SharedPreferences.

La función connectToDevice básicamente comprueba los permisos Bluetooth (en caso de que se ejecute directamente desde gesto sin entrar en el fragment) y establece la conexión mediante la instrucción "m_BluetoothSocket!!.connect()" vista en la tabla 22. También cierra el socket en caso de acabar la comunicación. El código de connectToDevice se ve en la **tabla 24**.

```
private fun connectToDevice(device: BluetoothDevice) {
   CoroutineScope(Dispatchers.IO).launch {
      try {
         if (ActivityCompat.checkSelfPermission(requireContext(),
Manifest.permission.BLUETOOTH_CONNECT) != PackageManager.PERMISSION_GRANTED)
{
            ActivityCompat.requestPermissions(
               requireActivity(), arrayOf(
                  Manifest.permission.BLUETOOTH,
                  Manifest.permission.BLUETOOTH_CONNECT,
                  Manifest.permission.BLUETOOTH_ADMIN,
                  Manifest.permission.BLUETOOTH_SCAN,
                  Manifest.permission.BLUETOOTH_ADVERTISE,
                  Manifest.permission.BLUETOOTH_PRIVILEGED,
                  Manifest.permission.MANAGE_DEVICE_POLICY_BLUETOOTH,
               ),
               1
            )
         }
         BTConfigFragment.m_bluetoothSocket =
device.createInsecureRfcommSocketToServiceRecord(BTConfigFragment.m_myUUID) //
BTConfigFragment.m_bluetoothSocket =
device.createRfcommSocketToServiceRecord(BTConfigFragment.m_myUUID)
```

```
            BTConfigFragment.m_bluetoothSocket!!.connect()
            BTConfigFragment.m_isConnected=true
            withContext(Dispatchers.Main) {
                Toast.makeText(requireContext(), "Conexión Bluetooth exitosa",
Toast.LENGTH_SHORT).show()
            }
            Log.d("Bluetooth", "UUID:${BTConfigFragment.m_myUUID}")
            Log.d("Bluetooth", "Direccion:${BTConfigFragment.m_address}")
        } catch (e: IOException) {
            Log.e("Bluetooth", "Error al conectar con el dispositivo Bluetooth", e)
            withContext(Dispatchers.Main) {
                Toast.makeText(requireContext(), "Error al conectar con el dispositivo
Bluetooth", Toast.LENGTH_SHORT).show()
            }
        } finally {
            if (BTConfigFragment.m_bluetoothSocket != null &&
!BTConfigFragment.m_bluetoothSocket!!.isConnected) {
                try {
                    BTConfigFragment.m_bluetoothSocket!!.close()
                } catch (e: IOException) {
                    Log.e("Bluetooth", "Error al cerrar el socket Bluetooth: ", e)
                }
            }
        }
    }
}
```

*Tabla 24. Función llamada al establecer conexión Bluetooth con el gesto*

Una vez visto el código para el establecimiento de la conexión Bluetooth, se analiza el que se encarga del envío de mensajes **(tabla 25)**.

```
if((MQTTConfigFragment.conectado || BTConfigFragment.m_isConnected) &&
UsageInfoFragment.cejasEncender){
    if(BTConfigFragment.m_isConnected && transcurridoDesdeBT >= 5000){
        sendCommand("A")
        lastPublishedMessage = "Cambio Plato"
        ultimoBT = System.currentTimeMillis()
    }
}
else if((MQTTConfigFragment.conectado || BTConfigFragment.m_isConnected) &&
UsageInfoFragment.cejasApagar){
    if(BTConfigFragment.m_isConnected && transcurridoDesdeBT >= 5000){
        sendCommand("B")
        lastPublishedMessage = "Dar de Comer"
        ultimoBT = System.currentTimeMillis()
    }
}
```

*Tabla 25. Código correspondiente al envío de mensajes por Bluetooth*

Este código forma parte del que se ejecuta al realizar un gesto, el que es visible en la tabla 16. Se encarga de invocar a la función sendCommand **(tabla 26)** para

llevar a cabo el envío de mensajes cuando se realiza o bien el gesto de encendido o el de apagado y se está conectado a un dispositivo mediante Bluetooth.

Se aprecia en el código de la tabla 25 que los mensajes transmitidos mediante Bluetooth consisten simplemente en un carácter, y este puede ser A o B. Esto se debe a que solo queremos transmitir dos posibles órdenes al microcontrolador. De ese modo, esos 2 caracteres serán recibidos e interpretados por el código que se desarrollará para la ESP32.

```
//Para enviar mensajes por Bluetooth
private fun sendCommand(input: String) {
   if (BTConfigFragment.m_bluetoothSocket != null) {
      try{
         BTConfigFragment.m_bluetoothSocket!!.outputStream.write(input.toByteArray())
      } catch(e: IOException) {
         e.printStackTrace()
      }
   }
}
```

*Tabla 26. Función que envía mensajes por Bluetooth*

Como se puede ver, “sendCommand” es una función muy sencilla que simplemente usa la instrucción “m_bluetoothSocket!!.outputStream.write()” para emitir el mensaje deseado.

### 5.9.2 Programación del ESP32 para recibir e interpretar los mensajes Bluetooth

Ya se ha visto todo el código destinado a la gestión de la comunicación que se emplea en la app, por lo que tan solo queda comprender cómo actúa el microcontrolador al recibir los mensajes enviados desde el dispositivo Android.

Para gestionar la forma en la que el ESP32 recibe y responde ante los mensajes Bluetooth, se desarrolló un programa empleando PlatformIO, una extensión del entorno de desarrollo Visual Studio Code especializada en la programación de microcontroladores.

El funcionamiento deseado del microcontrolador consiste en la recepción de los mensajes, y posteriormente en la activación de un pin digital distinto para cada uno de los dos caracteres recibidos. Este funcionamiento se logra mediante el código de la **tabla 27**.

```
#include <Arduino.h>
#include "BluetoothSerial.h"
```

```
#define RX_QUEUE_SIZE 4000
BluetoothSerial ESPBT;

char entrante = ' ';

const int ledPin1 = 25;
const int ledPin2 = 26;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  ESPBT.begin("ESP32_CARTIF");
  Serial.print("Prueba de que sale texto por el monitor");
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(500);
  if(ESPBT.available())
  {
    //Recibe una letra (un char) del móvil, una en función del botón
pulsado en la aplicación
      entrante = ESPBT.read();
      if(entrante == 'A'){
        Serial.print("Cambiar plato");
        digitalWrite(ledPin1, HIGH);
        delay(1000);
        digitalWrite(ledPin1, LOW);
        }
      else if(entrante == 'B'){
        Serial.print("Dar de comer");
        digitalWrite(ledPin2, HIGH);
        delay(1000);
        digitalWrite(ledPin2, LOW);
        }
      Serial.print("\n");
  }
}
```

*Tabla 27. Código cargado en el ESP32 para la recepción de mensajes Bluetooth*

La activación durante un segundo del pin 25 al recibir un carácter A equivale a la acción de pulsar el interruptor de cambiar el plato del robot OBI durante un segundo, ya que la salida de ese pin digital está conectada en la placa

desarrollada con el transistor que da corriente a la entrada Jack del OBI que gestiona el cambio de plato.

Sucede lo mismo con el pin 26, el carácter B y la acción de dar de comer.

Con esto, ya se ha visto todo el código desarrollado que se encarga de la comunicación.

# 6. EXPERIMENTACIÓN Y RESULTADOS

En este apartado del proyecto se realizarán ensayos para comprobar el correcto funcionamiento de la aplicación desarrollada, demostrando que permite el control de entornos inteligentes mediante la realización de gestos faciales, y que la adaptación del sistema de control del robot OBI para no depender de pulsadores puede satisfacer las necesidades de personas con problemas de movilidad en los brazos.

## 6.1 Ubicación óptima del dispositivo Android

Debido a la necesidad de la detección de puntos faciales en el rostro del usuario, existe un límite de distancia a partir del cual la herramienta Face Mesh para la detección de dichos puntos deja de funcionar correctamente al no localizar la cara del usuario. Según la página oficial de Google para la herramienta de malla facial de ML Kit [84], esa distancia es de 2 metros, lo que permite el uso de la aplicación situando el dispositivo Android en un lugar cercano al usuario, para así no sufrir problemas de detección.

De igual forma, la localización dentro de ese rango de distancia que mejor permite la detección de gestos es justo enfrente de la cara del usuario, si bien existe un rango de ángulos en los que, aunque la cara se vea inclinada al mirar al frente, se sigue detectando sin problema los gestos realizados. Se muestran imágenes del ángulo a partir del cual se deja de detectar correctamente la realización de estos en las **figuras 73 y 74**:

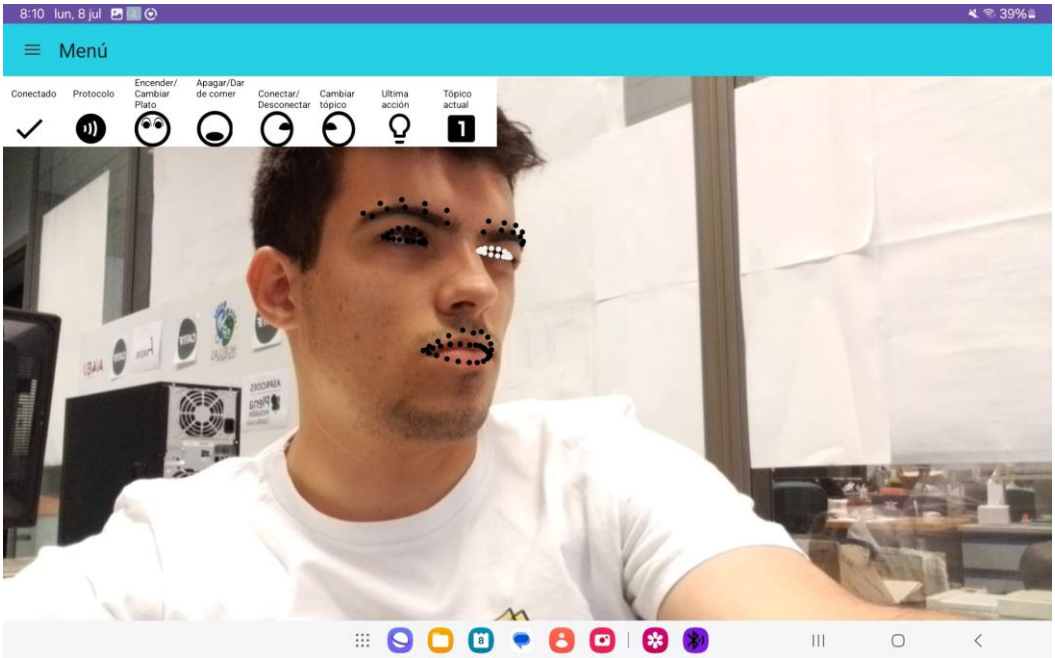


*Figura 73. Ángulo horizontal en el que aún se detectan correctamente gestos*

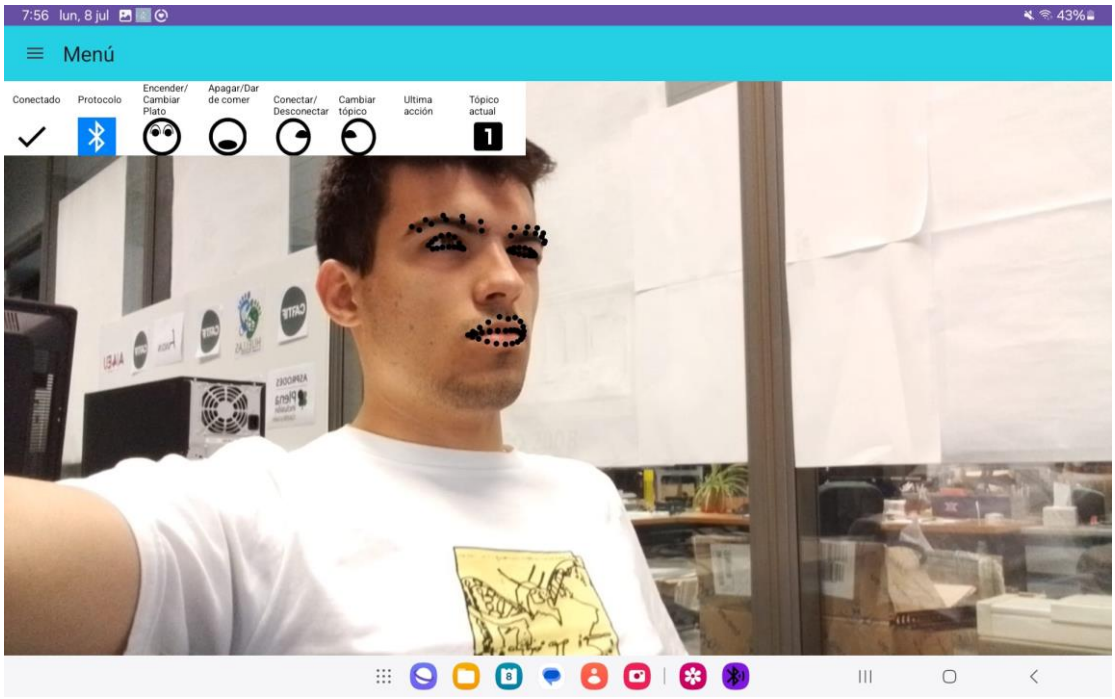



*Figura 74. Ángulo horizontal a partir del cual falla el reconocimiento de gestos*

Este ángulo es de aproximadamente 45 grados desde la dirección hacia la que mira la cara del usuario.

Esta combinación de rango de distancias y rango de ángulos en los que se puede situar el móvil o Tablet a usar permite una gran variedad de localizaciones posibles en caso de disponer de una mesa o superficie horizontal similar cercana al usuario en la que colocar el dispositivo Android.

Otra condición a tener en cuenta en la correcta detección de gestos es la iluminación, que puede provocar que la detección de puntos faciales falle al no lograr detectar el rostro del usuario. Se analizan imágenes con distintas condiciones de luminosidad para comprobar visualmente una estimación del rango de luminosidad en el que se puede usar la aplicación **(figuras 75 y 76)**.

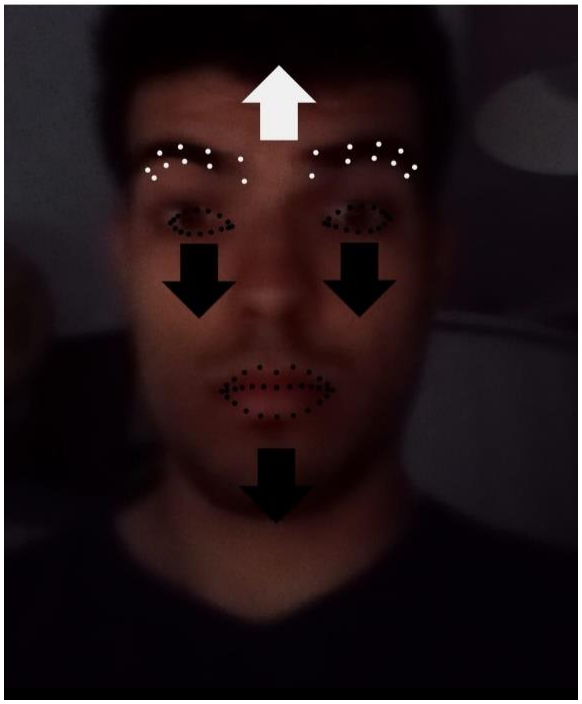


*Figura 75. Detección del rostro aún con baja iluminación*

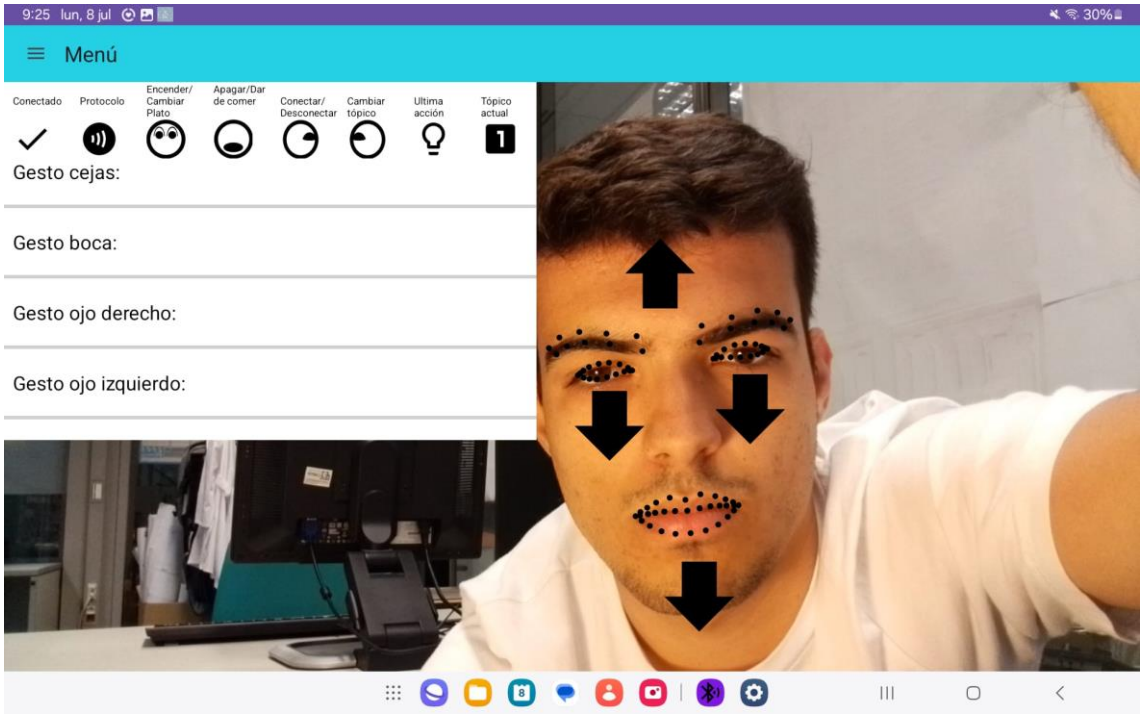



*Figura 76. Detección del rostro aún con alta iluminación*

Como se ve en las figuras, la iluminación ambiental no supondrá un problema para la detección de puntos faciales aunque se trate de una oscuridad casi absoluta o de una luz cegadora siempre que la cara del usuario sea distinguible.

Así, serán múltiples las situaciones y posiciones del dispositivo Android en las que la detección de gestos faciales será posible, permitiendo diferentes opciones de posicionamiento que se podrían adaptar a las necesidades del usuario con problemas de movilidad.

## 6.2 Ensayo del funcionamiento de la aplicación con el encendido y apagado de un enchufe inteligente

Se realizó un ensayo para comprobar la gestión de dispositivos inteligentes a través de la aplicación al realizar gestos faciales. Para dicho ensayo se utilizó el enchufe inteligente mostrado en la figura 47, el cual recibiría mensajes de ON o de OFF a través de MQTT para dejar pasar corriente o para no dejarla pasar, respectivamente, encendiendo así la bombilla vista en la figura 49.

Se configuró el levantar las cejas como encendido y el gesto de abrir la boca como apagado. El resultado del ensayo se ve en las figuras **77 y 78**.

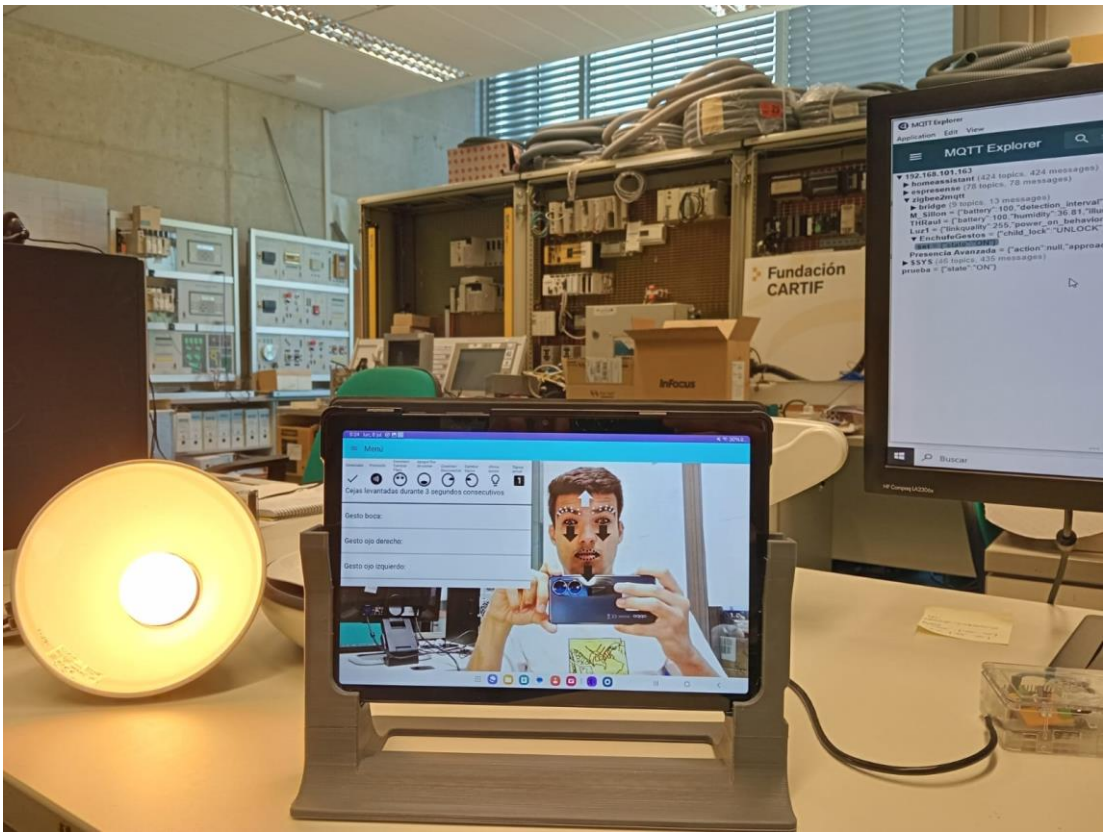



*Figura 77. Encendido de la lámpara al enviar un ON al enchufe*

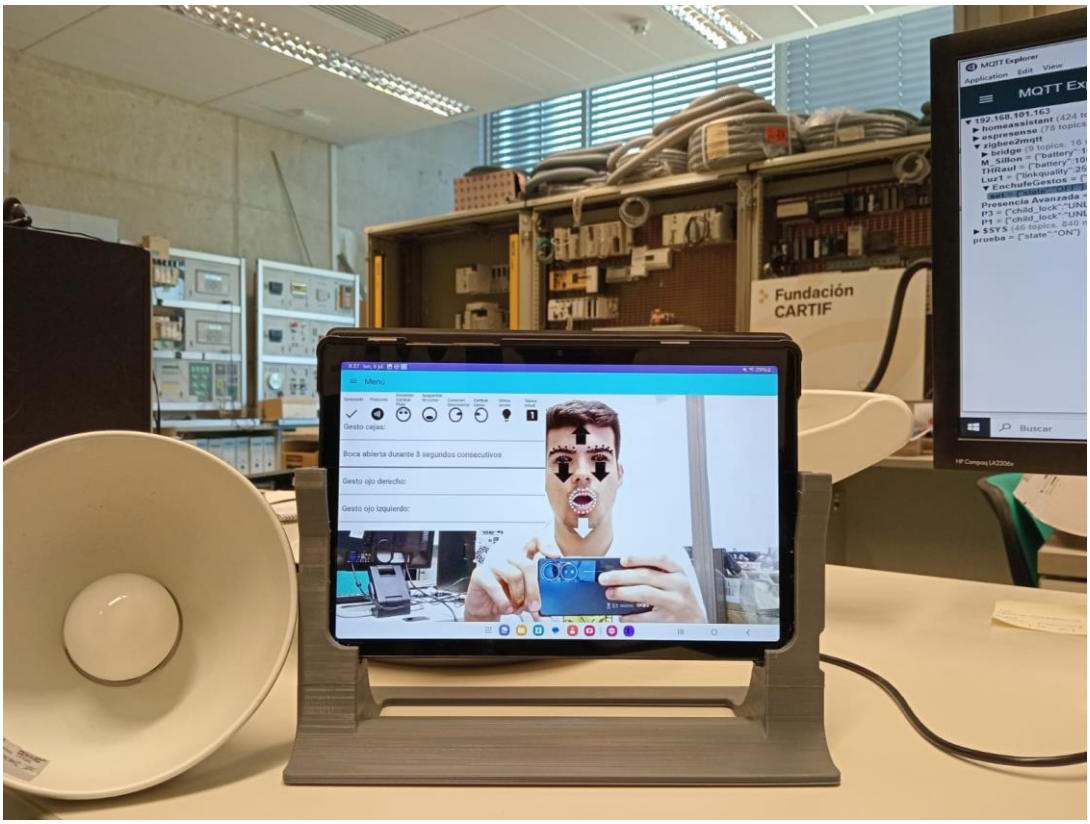



*Figura 78. Apagado de la lámpara al enviar un OFF al enchufe*

El funcionamiento del control de dispositivos inteligentes a través de MQTT al detectar gestos faciales usando la aplicación desarrollada fue el esperado.

## 6.3 Ensayo del funcionamiento de la aplicación con el control del robot OBI a través del ESP32

También se comprobó el funcionamiento de la comunicación Bluetooth con el ESP32 y de la placa diseñada con el robot OBI, de modo que al recibir el microcontrolador una de las 2 señales programadas por Bluetooth, activa el pin digital programado y el transistor conectado a ese pin cierra el mismo circuito que cerraban los pulsadores originales, activando el robot.

Los gestos tenían la misma configuración que en la prueba del enchufe, por lo que el levantamiento de cejas da la orden de cambio de plato y la apertura de la boca da la de dar de comer. El control del OBI se aprecia en las **figuras 79, 80 y 81**:


*Figura 79. Robot OBI en una posición anterior al control*

*Figura 80. Robot OBI tras recibir una orden de cambio de plato*


*Figura 81. Robot OBI tras recibir una orden de dar de comer*

El control del robot OBI a través de los gestos realizados fue el esperado, lo que demuestra la correcta comunicación mediante los mensajes Bluetooth programados entre el dispositivo Android y el microcontrolador ESP32 y el correcto funcionamiento de la placa diseñada a la hora de sustituir los pulsadores.

## 6.4 Experimentación con Usuarios

Para poder analizar si la aplicación desarrollada es de fácil comprensión para otros usuarios, si su interfaz es intuitiva y si su uso es cómodo y fiable, se ha pedido la opinión de usuarios externos al desarrollo del proyecto. La opinión de estos usuarios se ha recogido en una encuesta basada en el modelo Almere [159], encuesta orientada a obtener la opinión de personas de la tercera edad sobre la inclusión de robots de asistencia social. Las preguntas formuladas a los usuarios que dieron su opinión eran un total de 28, 22 de las cuales se agrupaban en 10 campos (ansiedad, actitud hacia la tecnología, condiciones facilitadoras, intención de uso, adaptabilidad percibida, entretenimiento percibido, facilidad de

uso percibida, sociabilidad percibida, utilidad percibida y confianza) y otras 6 pidiendo una valoración general de la aplicación.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | Hombre | | | | | |
| | | Mujer | | | | | |
| | | Edad | | | | | |
| | | | | | | | |
| | | Puntúa el cuestionario en escala de 1 a 5 | | | | | |
| | | 5: completamente de acuerdo, 4: relativamente de acuerdo, 3: neutral, 2: relativamente en desacuerdo, | | | | | |
| | | 1: completamente en desacuerdo | | | | | |
| | | | 5 | 4 | 3 | 2 | 1 |
| Ansiedad (ANX) | 1 | Si usara la aplicación, tendría miedo a equivocarme | | | | | |
| | 2 | Si usara la aplicación, tendría miedo de romper algo | | | | | |
| Actitud hacia la Tecnología (ATT) | 3 | Creo que es una buena idea utilizar la aplicación | | | | | |
| Condiciones Facilitadoras (FC) | 4 | Sé como utilizar la aplicación | | | | | |
| Intención de Uso (ITU) | 5 | Utilizaría la aplicación otra vez | | | | | |
| | 6 | Utilizaría la aplicación a menudo | | | | | |
| Perceived Adaptability (PAD) | 7 | Creo que la aplicación puede adaptarse a lo que necesito a menudo | | | | | |
| | 8 | Creo que la aplicación puede adaptarse a lo que necesito en un momento particular | | | | | |
| | 9 | Creo que la aplicación puede ayudarme cuando sea necesario | | | | | |
| Entretenimiento Percibido (PENJ) | 10 | La aplicación me parece fascinante | | | | | |
| | 11 | La aplicación me parece agradable | | | | | |
| | 12 | La aplicación me parece aburrida | | | | | |
| Facilidad de uso percibida (PEOU) | 13 | Creo que sabría rápidamente como utilizar la aplicación | | | | | |
| | 15 | Encuentro la aplicación fácil de usar | | | | | |
| | 16 | Creo que podría utilizar la aplicación sin ayuda | | | | | |
| | 17 | Creo que podría utilizar la aplicación si alguien me ayudara | | | | | |
| | 18 | Creo que podría utilizar la aplicación si me enseñaran a manejarla durante un tiempo | | | | | |
| Sociabilidad Percibida (PS) | 19 | Encuentro agradable la interfaz de la aplicación | | | | | |
| Utilidad Percibida (PU) | 20 | Creo que la aplicación podría serme útil | | | | | |
| | 21 | Me vendría bien tener la aplicación en mi dispositivo | | | | | |
| Confianza | 22 | La aplicación transmite confianza | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | **Valora la aplicación de acuerdo a los siguientes aspectos (5 mejor valorado, 0 peor valoración)** | | | | | |
| | 1 | Reconocimiento de gestos | | | | | |
| | 2 | Control de entornos inteligentes | | | | | |
| | 3 | Control del robot de alimentación asistida | | | | | |
| | 4 | Facilidad de uso | | | | | |
| | 5 | Diseño de la interfaz de usuario | | | | | |
| | 6 | Valoración General | | | | | |

*Figura 82. Plantilla de la encuesta sobre la aplicación desarrollada*

A cada usuario se le situó enfrente del dispositivo Android en el que se usaba la aplicación, junto con la lámpara usada en el desarrollo y junto con el robot OBI conectado a la placa desarrollada. Se les explicó que la aplicación estaba destinada al control de entornos inteligentes mediante la realización de gestos faciales y se les dejó usarla para probar su funcionamiento.

Se consultó a un total de 6 usuarios, cuyas opiniones se resumen en la **tabla 28**. En dicha tabla, cada una de las abreviaciones de la primera columna representa uno de los 10 campos en los que se agrupaban las preguntas. En cuanto al sistema de calificación, un 5 significa “completamente de acuerdo” y un 1 equivale a “completamente en desacuerdo”.

| Usuario | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Edad | 23 | 23 | 24 | 48 | 22 | 23 |
| Género | Hombre | Hombre | Mujer | Hombre | Hombre | Mujer |
| ANX | 1 | 1 | 1 | 2 | 2 | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| ATT | 5 | 5 | 5 | 5 | 5 | 5 |
| FC | 5 | 5 | 4 | 2 | 5 | 4 |
| ITU | 5 | 5 | 5 | 5 | 5 | 4 |
| PAD | 4 | 5 | 5 | 3 | 3 | 4 |
| PENJ | 4 | 5 | 5 | 3 | 5 | 3 |
| PEOU | 4 | 5 | 5 | 4 | 4 | 4 |
| PS | 4 | 4 | 5 | 2 | 4 | 3 |
| PU | 5 | 5 | 5 | 3 | 4 | 3 |
| Trust | 5 | 5 | 5 | 5 | 5 | 4 |
| Gestos reconocidos | 4 | 5 | 5 | 4 | 4 | 4 |
| Control de entornos | 5 | 5 | 5 | 5 | 5 | 5 |
| Control del robot OBI | 5 | 5 | 5 | 4 | 5 | 5 |
| Facilidad de uso | 4 | 5 | 5 | 3 | 5 | 4 |
| Diseño de la interfaz | 4 | 4 | 5 | 3 | 4 | 3 |
| Valoración general | 4 | 5 | 5 | 4 | 5 | 4 |

*Tabla 28. Resultados de la encuesta*

Siguiendo las opiniones expresadas por los usuarios en la encuesta, ninguno siente miedo de usar la app de forma indeseada o de romper algo al hacerlo. De igual forma, todos opinaron que la aplicación puede resultar sumamente útil.

La facilidad de uso de la aplicación fue un apartado con opiniones enfrentadas. Mientras algunos usuarios opinaron que era una aplicación que ya sabían usar al poco tiempo de probarlo, otros consideraron que su uso era difícil de entender, principalmente por no entender lo que significaba la interfaz. Este problema se puede atribuir al desafío que supone mostrar información útil en la interfaz mientras se la mantiene pequeña para convivir con la vista de la cámara en una

pantalla de dimensiones reducidas. Sin embargo, todos afirmaron que la volverían a usar si así lo requieren.

También todos los usuarios encuestados consideraron que, si se les enseñara a utilizar la aplicación, podrían usarla sin problemas. En un caso de uso real, el usuario podría ser formado por el asistente que instale y configure la aplicación en el uso de la misma, para así poder aumentar su independencia y su calidad de vida.

En cuanto a las funcionalidades de la aplicación, hubo consenso generalizado en torno al buen desempeño de la detección de gestos y del control de entornos. El único apartado de opinión sobre características de la aplicación en el que las opiniones no fueron totalmente positivas fue en el diseño de la interfaz de usuario, principalmente por la gran cantidad de información en pantalla entre iconos, barras de progreso y elementos de indicación dibujados sobre la cara del usuario. Sin embargo, esta interfaz es configurable pudiendo ocultar ciertos elementos en el apartado de ajustes.

La valoración general de los usuarios consultados fue muy positiva y demuestra la utilidad del programa, su accesibilidad y su buen desempeño. En lo referente al control del enchufe de prueba y del robot OBI no hubo ningún momento en el que se activaran o desactivaran de forma inintencionada, por lo que se puede concluir que el uso de la aplicación es muy fiable.

# 7. ESTUDIO ECONÓMICO

Se realizará un estudio del proyecto desde el punto de vista económico, estimando el coste que ha supuesto su realización.

Una vez comprobada la viabilidad técnica del proyecto, se debe comprobar su viabilidad económica. Se analizarán costes directos e indirectos para determinar el coste total y así concluir si supone una ventaja económica respecto a otras posibles soluciones que podrían desarrollarse.

## 7.1 Costes Directos

### 7.1.1 Costes de Personal

Para el cálculo del coste de mano de obra en el desarrollo del proyecto, se ha considerado al alumno como ingeniero junior para tener una referencia de salario medio, el cual es de 25 000€ brutos anuales [160], o de 12,82€ por hora. Teniendo en cuenta que el tiempo empleado en el desarrollo de la aplicación ha sido de 510 horas, se puede realizar el cálculo mostrado.

$$Salario\ bruto\ recibido = 12{,}82€ * 510h = 6\ 538{,}2€$$

Hace falta sumar la cotización en la Seguridad Social que debe pagar la empresa por tener contratado al trabajador, lo que es un 33,4% del salario bruto del empleado [161].

$$Coste\ de\ personal = 6\ 538{,}2€ * 0{,}334 + 6\ 538{,}2€ = \ 8\ 721{,}96€$$

### 7.1.2 Costes de Equipos y Software

Algunos de los recursos empleados en el desarrollo del proyecto fueron enumerados en el apartado 5.1, pero ni se estudiaron sus costes ni se incluyeron elementos como el ordenador en el que desarrolló el código ni su sistema operativo, por ejemplo.

Por ello, se analizan a continuación los costes de los equipos empleados **(tabla 29)**:

| MATERIAL | IMPORTE | CANTIDAD | TOTAL |
|---|---|---|---|
| Ordenador HP Compaq 8200 | 151,95€ | 1 | 151,95€ |
| Tab A8 | 172,73€ | 1 | 172,73€ |
| Oppo A58 | 138,84€ | 1 | 138,84€ |
| Enchufe FDTEK | 19,98€ | 1 | 19,98€ |
| Lámpara TERTIAL | 9,99€ | 1 | 9,99€ |
| Bombilla Solhetta | 9,99€ | 1 | 9,99€ |
| Robot OBI | 8 625€ | 1 | 8 625€ |
| ESP32 FireBeetle | 8,90€ | 1 | 8,90€ |
| Transistores BC547 | 0,51€ | 2 | 1,02€ |
| Resistencias 1K | 0,13€ | 2 | 0,26€ |

*Tabla 29. Coste del Hardware empleado*

Se analiza también el coste del Software empleado a lo largo del desarrollo **(tabla 30)**:

| MATERIAL | IMPORTE |
|---|---|
| Windows 10 Pro | 260€ |
| Android Studio | 0€ |
| Visual Studio Code | 0€ |
| PlatformIO | 0€ |
| MQTT Explorer | 0€ |

*Tabla 30. Coste del Software empleado*

Esto hace que el coste total de equipos se sitúe en los 9 138,66€ y el de software en 260€.

La suma de ambos da un coste de 9 398,66€.

### 7.1.3 Total de Costes Directos

Los costes directos totales son la suma de los costes de personal y de equipos y software:

$$Coste\ Directo\ Total = 8\,721{,}96€ + 9\,398{,}66€ = 18\,120{,}62€$$

## 7.2 Costes Indirectos

Los costes indirectos están asociados a procesos relacionados con el proyecto, pero que no forman parte directamente del mismo, como el desplazamiento hasta el puesto de trabajo o la electricidad consumida.

| CONCEPTO | COSTE |
|---|---|
| Desplazamiento | 150€ |
| Electricidad | 50€ |

| | |
|---|---|
| Total Indirectos | 200€ |

## 7.3 Coste Total

El coste total consiste en la suma de costes directos e indirectos:

$$Coste\ Total\ del\ Proyecto = 18\ 120{,}62€ + 200€ = 18\ 320{,}62€$$

# 8. CONCLUSIONES Y LÍNEAS FUTURAS

## 8.1 Conclusiones

En este trabajo se ha desarrollado una aplicación Android para el control de entornos inteligentes buscando alcanzar una serie de objetivos definidos al comienzo del mismo.

Debido a la naturaleza del proyecto, que ha buscado satisfacer las necesidades del día a día de personas en condiciones de salud muy delicadas, se ha hecho especial énfasis en reducir todo lo posible los movimientos de las extremidades, en establecer un sistema de control intuitivo y sencillo de usar y en la posibilidad de configurar exhaustivamente el uso de la aplicación para hacerla tan accesible como fuera posible.

A través de la comprensión del estado actual de aplicaciones con conceptos similares, de la explicación de la organización y estructura de la aplicación, del análisis del código desarrollado para la comunicación con dispositivos al detectar la realización de gestos faciales y de la experimentación con la aplicación final, es seguro decir que el proyecto ha cumplido con sus objetivos iniciales.

- Se diseñó una interfaz sencilla e intuitiva que no requiere de interacción directa con la pantalla del dispositivo para el uso de la aplicación, cumpliendo con las necesidades de usuarios con discapacidades motoras
- Se implementó una herramienta de detección de puntos faciales con los que se puede identificar cuando se hace un gesto determinado.
- Se incluyeron en la interfaz elementos de representación visual sobre el rostro del usuario para que pueda saber si se está detectando el gesto realizado.
- Se implementaron 2 protocolos de comunicación para el control de dispositivos inteligentes que permiten tanto el control simultáneo y avanzado de múltiples dispositivos en el caso de MQTT como el control inalámbrico sin necesidad de conexión a Internet en el caso de Bluetooth
- Se establecieron herramientas en la aplicación para la personalización de gestos y de umbrales de detección de los mismos para hacer accesible el uso de la app
- Se consiguió una ejecución fluida y sin problemas de rendimiento
- Se experimentó con elementos físicos reales y con usuarios ajenos al proyecto para demostrar su funcionamiento

No solo se demostró el funcionamiento con usuarios, sino que también cabe destacar la buena recepción de la aplicación entre ellos. Los usuarios consultados dieron una valoración de 4/5 en el peor de los casos al funcionamiento de la aplicación.

Gracias a la realización de estos objetivos, también se apoya el cumplimiento de los ODS de la Agenda 2030 expuestos en el apartado inicial de objetivos. El proyecto ayuda a la inclusión social de personas con discapacidad al mejorar su autonomía, reduce desigualdades al brindarles nuevas posibilidades de interacción con su entorno y supone una herramienta importante para la mejora de su bienestar.

De este modo, la aplicación desarrollada supone una herramienta innovadora y práctica destinada a mejorar la calidad de vida y autonomía de un gran número de personas.

## 8.2 Trabajos Futuros

La aplicación desarrollada podría ser mejorada, añadiendo aún más funcionalidades o mejorando las ya implementadas. Algunos ejemplos son:

- Usar Mediapipe para detectar pupilas: Si en un futuro se optimiza Mediapipe para su uso en Android, o si bien los procesadores de dispositivos móviles mejoran tanto sus capacidades que la implementación de Mediapipe no suponga pérdida de fluidez, se podría cambiar la librería de detección de gestos faciales para poder usar la dirección en la que mira el usuario como parte del control de entornos.

- Trasladar la aplicación a otros sistemas operativos: El código desarrollado está escrito en Kotlin, lenguaje específico del desarrollo Android, pero una vez el código ha sido finalizado, la comprensión del proyecto obtenida a lo largo de su desarrollo hace más sencillo volver a programar la aplicación pero orientada a otros dispositivos móviles, como los de Apple, que usan sistema operativo ios.

- Traducir al inglés o a más idiomas: Toda la interfaz de la aplicación está en castellano, pero podría ser fácilmente traducida a otros idiomas y se podría permitir al asistente cambiar dicho idioma en los ajustes de la app.

- Configurar un gesto de emergencia que avise a un cuidador o a los servicios de emergencia en caso de por ejemplo, sufrir un ictus: Los accidentes cardiovasculares o ictus son una de las dolencias objetivo de la aplicación, y la detección de las mismas es reconocible por la parálisis en la mitad de la cara. Se podría implementar una detección de dicha situación para advertir a los servicios sanitarios.

# 9. REFERENCIAS BIBLIOGRÁFICAS

[1] Instituto Nacional de Estadística (INE). (2020). *Encuesta de Discapacidad, Autonomía personal y situaciones de Dependencia (EDAD) de 2020*.

https://www.ine.es/prensa/edad_2020_p.pdf

Consultado por última vez el 17 de Mayo de 2024

[2] Instituto Nacional de Estadística (INE). (2023). *Encuesta de Discapacidad, Autonomía personal y situaciones de Dependencia para población residente en centros (EDAD de Centros) de 2023.*

https://www.ine.es/dyngs/Prensa/EDAD2023.htm

Consultado por última vez el 17 de Mayo de 2024

[3] Instituto Nacional de Estadística (INE). (2008). *Encuesta de Discapacidad, Autonomía personal y situaciones de Dependencia (EDAD) de 2008*.

https://www.ine.es/revistas/cifraine/1009.pdf

Consultado por última vez el 21 de Mayo de 2024

[4] Organización Mundial de la Salud (OMS). (2011). *Informe sobre discapacidades a nivel mundial de 2011.*

https://www.who.int/teams/noncommunicable-diseases/sensory-functions-disability-and-rehabilitation/world-report-on-disability

Consultado por última vez el 25 de Mayo de 2024

[5] Fuentes, F. (14 de diciembre de 2023). *¿Qué es, cómo funciona y dónde se aplica la visión artificial?.* Arsys

https://www.arsys.es/blog/vision-artificial

Consultado por última vez el 28 de Mayo de 2024

[6] IBM. (s.f.). *¿Qué es la visión artificial?*

https://www.ibm.com/es-es/topics/computer-vision#:~:text=La%20visi%C3%B3n%20artificial%20es%20un,en%20funci%C3%B3n%20de%20esa%20informaci%C3%B3n.

Consultado por última vez el 1 de Junio de 2024

[7] BBVA. (8 de noviembre de 2019). *Machine Learning, ¿qué es y cómo funciona?*

https://www.bbva.com/es/innovacion/machine-learning-que-es-y-como-funciona/

Consultado por última vez el 4 de Junio de 2024

[8] Telefónica. (24 de octubre de 2023). *Redes Neuronales Convolucionales: qué son, tipos y aplicaciones*

https://www.telefonica.com/es/sala-comunicacion/blog/redes-neuronales-convolucionales-que-son-tipos-aplicaciones/

Consultado por última vez el 4 de Junio de 2024

[9] Iberdrola. (s.f.). *¿Qué es la visión artificial y cuáles son sus aplicaciones?*

https://www.iberdrola.com/innovacion/vision-artificial

Consultado por última vez el 6 de Junio de 2024

[10] Mihajlovic, I. (25 de abril de 2019). *Everything You Ever Wanted To Know About Computer Vision*. Medium

https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e

Consultado por última vez el 9 de Junio de 2024

[11] Kanade, T. (1974). *Picture Processing System By Computer Complex And Recognition Of Human Faces*. Universidad de Kioto

https://www.ri.cmu.edu/pub_files/pub3/kanade_takeo_1973_1/kanade_takeo_1973_1.pdf

Consultado por última vez el 11 de Junio de 2024

[12] Cootes, T. F., Edwards, G. J., & Taylor, C. J. (1999). *Comparing Active Shape Models with Active Appearance Models*. Bmvc (Vol. 99, No. 1, p.173-182)

https://www.researchgate.net/publication/221259802_Comparing_Active_Shape_Models_with_Active_Appearance_Models

Consultado por última vez el 11 de Junio de 2024

[13] Rowley, H. A., Baluja, S., & Kanade, T. (1998). *Neural Network-Based Face Detection*. IEEE Transactions on pattern análisis and machine intelligence, 20(1), 23-28

https://www.ri.cmu.edu/pub_files/pub1/rowley_henry_1996_3/rowley_henry_1996_3.pdf

Consultado por última vez el 10 de Junio de 2024

[14] Duffner, S., & Garcia, C. (2005). *A connexionist approach for robust and precise facial feature detection in complex scenes*. ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing And Analysis, 2005. (p.316-321). IEEE

https://www.researchgate.net/publication/4181165_A_connexionist_approach_for_robust_and_precise_facial_feature_detection_in_complex_scenes

Consultado por última vez el 11 de Junio de 2024

[15] Sun, Y., Wang, X., Tang, X. (2013). *Deep Convolutional Network Cascade for Facial Point Detection*. Proceedings of the IEEE conference on computer vision and pattern recognition (p. 3476-3483)

https://www.researchgate.net/publication/261259065_Deep_Convolutional_Network_Cascade_for_Facial_Point_Detection

Consultado por última vez el 11 de Junio

[16] Howard, A. G., Zhu, M., Chen, B., Kaleninchenko, D., Wang, W., Weyand, T., & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*

https://arxiv.org/pdf/1704.04861

Consultado por última vez el 11 de Junio

[17] Villanueva, U. R., Delión, J. C. G., & Larroca, F. P. (2022). *Reconocimiento de expresiones faciales y características personales como herramienta para identificar personas en un sistema de transporte público*. Ingeniería Industrial, 216-277

https://revistas.ulima.edu.pe/index.php/Ingenieria_industrial/article/view/5811/5662

Consultado por última vez el 10 de Junio de 2024

[18] Das, S. P., Talukdar, A. K., & Sarma, K. K. (2015). *Sign Language Recognition using Facial Expression*. Procedia Computer Science, 58, 210-216

https://www.sciencedirect.com/science/article/pii/S1877050915021675

Consultado por última vez el 10 de Junio de 2024

[19] Bahçeci Şimşek, I., & Şirolu, C. (2021). *Analysis of surgical outcome after upper eyelid surgery by computer visión algorithm using fase and facial landmark detection*. Graefe's Archive for Clinical and Experimental Ophtalmology, 259(10). 3119-3125

https://link.springer.com/article/10.1007/s00417-021-05219-8

Consultado por última vez el 11 de Junio de 2024

[20] Zhang, Z., Luo, P., Loy, C. C., & Tang, X. (2014). *Facial Landmark Detection by Deep Multi-task Learning*. Computer Vision – ECCV 2014: 13th European

Coference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13 (p.94-108). Springer International Publishing

https://personal.ie.cuhk.edu.hk/~ccloy/files/eccv_2014_deepfacealign.pdf

Consultado por última vez el 11 de Junio de 2024

[21] Rosenbrock, A. (2017). *Facial landmarks with dlib, OpenCV and Python*

https://pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/

Consultado por última vez el 11 de Junio de 2024

[22] Elmahmudi, A., & Ugail, H. (2021). *A framework for facial age progression and regression using exemplar fase templates*. The visual computer, 37(7), 2023-2038

https://www.researchgate.net/publication/343699139_A_framework_for_facial_age_progression_and_regression_using_exemplar_face_templates

Consultado por última vez el 11 de Junio de 2024

[23] Guo, X., Li, S., Yu, J., Zhang, J., Ma, J., Ma, L., & Ling, H. (2019). *PFLD: A Practical Facial Landmark Detector*

https://arxiv.org/pdf/1902.10859v2

Consultado por última vez el 11 de Junio de 2024

[24] Asmara, R. A., Ridwan, M., & Budiprasetyo, G. (2021). *Haar Cascade and convolutional Neural Network Face Detection in Client-Side for Cloud Computing Face Recognition*. 2021 International Conference on Electrical and Information Technology (IEIT) (p.1-5). IEEE

https://ieeexplore.ieee.org/document/9587388

Consultado por última vez el 11 de Junio de 2024

[25] Huang, Y., Yang, H., Li, C., Kim, J., & Wei, F. (2021). *ADNet: Leveraging Error-Bias Towards Normal Direction in Face Alignment*. Proceedings of the IEEE/CVF International conference on computer visión (p. 3080-3090)

https://openaccess.thecvf.com/content/ICCV2021/papers/Huang_ADNet_Leveraging_Error-Bias_Towards_Normal_Direction_in_Face_Alignment_ICCV_2021_paper.pdf

Consultado por última vez el 11 de Junio de 2024

[26] Microsoft. (25 de junio de 2024). *Ajuste de hiperparámetros de un modelo*

https://learn.microsoft.com/es-es/azure/machine-learning/how-to-tune-hyperparameters?view=azureml-api-2

Consultado por última vez el 29 de Junio de 2024

[27] National Institute of Health (NIH). (4 de marzo de 2024). *Parálisis cerebral*

https://espanol.ninds.nih.gov/es/trastornos/paralisis-cerebral

Consultado por última vez el 12 de Junio de 2024

[28] Camacho, A., Esteban, J., & Paradas, C. (2018). *Informe de la Fundación Del Cerebro sobre el impacto social de la esclerosis lateral amiotrófica y las enfermedades neuromusculares*. Neurología, 33(1), 35-46

https://www.sciencedirect.com/science/article/pii/S0213485315000341

Consultado por última vez el 12 de Junio de 2024

[29] Pérez Litago, U. (2021). *Trastornos del habla en personas con esclerosis múltiple y su repercusión en la calidad de vida*

https://www.sciencedirect.com/science/article/pii/S0214460323000049

Consultado por última vez el 12 de Junio de 2024

[30] Cammarata-Scalisi, F., Camacho, N., Alvarado, J., & Lacruz-Rengel, M. A. (2008). Distrofia muscular de Duchenne, presentación clínica. *Revista chilena de pediatría*, 79(5), 495-501

https://www.scielo.cl/scielo.php?pid=s0370-41062008000500007&script=sci_arttext

Consultado por última vez el 12 de Junio de 2024

[31] Clínica Mayo. (30 de diciembre de 2023). *Lesiones de la médula espinal*

https://www.mayoclinic.org/es/diseases-conditions/spinal-cord-injury/symptoms-causes/syc-20377890

Consultado por última vez el 12 de Junio de 2024

[32] Martínez-Fernández, R., Gasca-Salas, C., Sánchez-Ferro, Á., & Obeso, J. Á. (2016*).* Actualización en la enfermedad de Parkinson. *Revista Médica Clínica Las Condes*, 27(3), 363-379

https://www.sciencedirect.com/science/article/pii/S0716864016300372

Consultado por última vez el 12 de Junio de 2024

[33] Romero, M. D. M. G., & Pascual, S. I. P. (2022). *Atrofia muscular espinal*

https://www.aeped.es/sites/default/files/documentos/19.pdf

Consultado por última vez el 12 de Junio de 2024

[34] Potal de Administración Electrónica del Gobierno de España. (2023) *Guía de accesibilidad de aplicaciones móviles*

https://administracionelectronica.gob.es/pae_Home/pae_Estrategias/pae_Accesibilidad/pae_documentacion/pae_eInclusion_Accesibilidad_de_apps.html

Consultado por última vez el 12 de Junio de 2024

[35] Mouha, R. A. R. A. (2021). Internet of Things (IoT). *Journal of Data Analysis and Information Processing*, 9(02), 77

https://www.scirp.org/journal/paperinformation?paperid=108574

Consultado por última vez el 12 de Junio de 2024

[36] Gracia, M. (s.f.). *IoT-Internet of Things*. Deloitte

https://www2.deloitte.com/es/es/pages/technology/articles/IoT-internet-of-things.html

Consultado por última vez el 12 de Junio de 2024

[37] Duarte, F. (19 de febrero de 2024). *Number of IoT Devices (2024)*. Exploding Topics

https://explodingtopics.com/blog/number-of-iot-devices

Consultado por última vez el 12 de Junio de 2024

[38] Sinha, S. (24 de mayo de 2023). *State of IoT 2023: Number of connected IoT devices growing 16% to 16.7 billion globally*. IOT Analytics

https://iot-analytics.com/number-connected-iot-devices/

Consultado por última vez el 12 de Junio de 2024

[39] DIGI. (s.f.). *¿Cómo se comunican los dispositivos de IoT?*

https://es.digi.com/blog/post/how-do-iot-devices-communicate

Consultado por última vez el 12 de Junio de 2024

[40] Amazon. (s.f.). *¿Qué es IoT (Internet de las cosas)?*

https://aws.amazon.com/es/what-is/iot/#:~:text=The%20term%20IoT%2C%20or%20Internet,as%20between%20the%20devices%20themselves.

Consultado por última vez el 12 de Junio de 2024

[41] SensorGO. (26 de agosto de 2021). *Smart Home: Descubre los beneficios de una casa inteligente*

https://sensorgo.mx/smart-home/

Consultado por última vez el 13 de Junio de 2024

[42] Apple. (s.f.). *Apple Watch Series 9 - Especificaciones técnicas*

https://support.apple.com/es-la/111833

Consultado por última vez el 13 de Junio de 2024

[43] Gómez, J. E., Marcillo, F. R., Triana, F. L., Gallo, V. T., Oviedo, B. W., & Hernández, V. L. (2017). *IoT for Environmental Variables in Urban Areas*. Procedia computer science, 109, 67-74

https://www.sciencedirect.com/science/article/pii/S1877050917309535

Consultado por última vez el 13 de Junio de 2024

[44] Paessler. (s.f.). *IT Explained: MQTT*

https://www.paessler.com/es/it-explained/mqtt#:~:text=MQTT%20son%20las%20siglas%20de,cuanto%20al%20ancho%20de%20banda.

Consultado por última vez el 13 de Junio de 2024

[45] HiveMQ. (20 de febrero de 2024). *What is MQTT Quality of Service (Qos) 0,1 & 2?*

https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

Consultado por última vez el 13 de Junio de 2024

[46] IBM. (31 de enero de 2024). *Calidades de servicio proporcionadas por un cliente de MQTT*

https://www.ibm.com/docs/es/ibm-mq/9.1?topic=concepts-qualities-service-provided-by-mqtt-client

Consultado por última vez el 13 de Junio de 2024

[47] Bluetooth. (s.f.). *Bluetooth Technology Overview*

https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

Consultado por última vez el 14 de Junio de 2024

[48] Intel. (s.f.). *¿Cómo funciona la tecnología Bluetooth?*

https://www.intel.la/content/www/xl/es/products/docs/wireless/how-does-bluetooth-work.html

Consultado por última vez el 14 de Junio de 2024

[49] IONOS. (1 de diciembre de 2022). *¿Qué es Bluetooth?*

https://www.ionos.es/digitalguide/servidores/know-how/que-es-bluetooth/

Consultado por última vez el 14 de Junio de 2024

[50] Dusun. (7 de junio de 2023). *Cómo la puerta de enlace Bluetooth ayuda al desarrollo de la atención médica de IoT y construye hospitales inteligentes*

https://www.dusuniot.com/es/blog/how-are-bluetooth-based-devices-used-in-healthcare/

Consultado por última vez el 14 de Junio de 2024

[51] Amar, M. (30 de noviembre de 2023). *A Deep dive into the pros and cons of using MQTT for IoT Applications*. Ellenex

https://www.ellenex.com/post/a-deep-dive-into-the-pros-and-cons-of-using-mqtt-for-iot-applications#:~:text=Section%201%3A%20Core%20Advantages%20of%20MQTT&text=MQTT%20is%20designed%20to%20use,battery%20power%20are%20often%20constrained.

Consultado por última vez el 14 de Junio de 2024

[52] Javatpoint. (s.f.). *Advantages and Disadvantages of Bluetooth*

https://www.javatpoint.com/advantages-and-disadvantages-of-bluetooth

Consultado por última vez el 14 de Junio de 2024

[53] OBI. (s.f.). *Meet OBI*

https://meetobi.com/

Consultado por última vez el 27 de Junio de 2024

[54] Park, D., Hoshi, Y., Mahajan, H. P., Kim, H. K., Erickson, Z., Rogers, W. A., & Kemp, C. C. (2020). Active robot-assisted feeding with a general-purpose mobile manipulator: Design, evaluation, and lessons learned. *Robotics and Autonomous Systems*, 124, 103344

https://www.researchgate.net/publication/336993304_Active_robot-assisted_feeding_with_a_general-purpose_mobile_manipulator_Design_evaluation_and_lessons_learned

Consultado por última vez el 27 de Junio de 2024

[55] Furness, D. (13 de marzo de 2019). *This robot is built to feed dinner to people who can't feed themselves*. Digitaltrends

https://www.digitaltrends.com/cool-tech/robot-feeds-dinner-mobility-impaired-people/

Consultado por última vez el 27 de Junio de 2024

[56] OBI. (s.f.). *Instructions For Use*

https://meetobi.com/wp-content/uploads/2023/06/IFD-600-003_INSTRUCTIONS-FOR-USE_R3.3.1.pdf

Consultado por última vez el 27 de Junio de 2024

[57] Personal Robotics. (s.f.). *Robot Assisted Feeding*

https://robotfeeding.io/

Consultado por última vez el 27 de Junio de 2024

[58] Control Bionics. (14 de julio de 2022). *Meet OBI: The Revolutionary Adaptive Dining Robot*

https://www.controlbionics.com/2022/meet-obi-the-revolutionary-adaptive-dining-robot/

Consultado por última vez el 28 de Junio de 2024

[59] Lutkevich, B. (noviembre de 2019). *Microcontroller (MCU)*. TechTarget

https://www.techtarget.com/iotagenda/definition/microcontroller

Consultado por última vez el 28 de Junio de 2024

[60] Marmolejo, R. (s.f.). *Microcontrolador – qué es y para qué sirve*. HETPRO

https://hetpro-store.com/TUTORIALES/microcontrolador/

Consultado por última vez el 28 de Junio de 2024

[61] Sigma Electrónica. (s.f.). *ATMEGA 328P*

https://www.sigmaelectronica.net/producto/atmega328p-pu/

Consultado por última vez el 28 de Junio de 2024

[62] BeagleBoard. (s.f). *What is a BeagleBone Black?*

https://www.beagleboard.org/boards/beaglebone-black

Consultado por última vez el 29 de Junio de 2024

[63] Vega, F. (s.f). *¿Qué es un system on a chip?*. Platzi

https://platzi.com/clases/1098-ingenieria/6552-que-es-un-system-on-a-chip/

Consultado por última vez el 29 de Junio de 2024

[64] GeeksForGeeks. (20 de mayo de 2024). *Diferencia entre MCU y SoC*

https://www.geeksforgeeks.org/difference-between-mcu-and-soc/

Consultado por última vez el 29 de Junio de 2024

[65] Cameron, N. (2023). *ESP32 Microcontroller.* ESP32 Formats and Communication. Maker Innovation Series

https://link.springer.com/chapter/10.1007/978-1-4842-9376-8_1#:~:text=The%20ESP32%20microcontroller%20includes%20two,watch%20with%20touchscreen%20and%20GPS.

Consultado por última vez el 29 de Junio de 2024

[66] Nayyar, A. (2015). *A Review of Beaglebone Smart Boards's-A Linux/Android Powered Low Cost Development Platform Base don ARM Technology*. 2015 9th Conference on Future Generation Communication and Networking (FGCN)

https://www.researchgate.net/publication/304412094_A_Review_of_Beaglebone_Smart_Board's-A_LinuxAndroid_Powered_Low_Cost_Development_Platform_Based_on_ARM_Technology

Consultado por última vez el 29 de Junio de 2024

[67] Element14. (17 de noviembre de 2020). *The Difference Between SoCs and MCUs*

https://community.element14.com/technologies/embedded/b/blog/posts/the-difference-between-socs-and-mcus

Consultado por última vez el 29 de Junio de 2024

[68] TRBL Services. (1 de junio de 2021). *Sistemas embebidos y sus características*

https://trbl-services.eu/blog-sistema-embebido-caracteristicas/

Consultado por última vez el 29 de Junio de 2024

[69] ELPROCUS. (s.f.). *Different Microcontroller Boards and their Applications*

https://www.elprocus.com/different-types-of-microcontroller-boards/

Consultado por última vez el 28 de Junio de 2024

[70] Guerra, J. (s.f.). *ESP32: WiFi y Bluetooth en un solo chip*. ProgramarFacil

https://programarfacil.com/esp8266/esp32/

Consultado por última vez el 29 de Junio de 2024

[71] AranaCorp. (18 de febrero de 2024). *Descripción general del microcontrolador ESP32*

https://www.aranacorp.com/es/descripcion-general-del-microcontrolador-nodemcu-esp32/

Consultado por última vez el 29 de Junio de 2024

[72] Espressif Systems. (s.f.). *ESP32 Series Datasheet*

https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Consultado por última vez el 29 de Junio de 2024

[73] Random Nerd Tutorials. (s.f). *Getting Started with the ESP32 Development Board*

https://randomnerdtutorials.com/getting-started-with-esp32/#esp32-programming-environments

Consultado por última vez el 29 de Junio de 2024

[74] Basumallick, C. (19 de marzo de 2024). *What is Android OS? History, Features, Versions, and Benefits*. SpiceWorks

https://www.spiceworks.com/tech/tech-general/articles/android-os/

Consultado por última vez el 17 de Junio de 2024

[75] Mixon, E. (s.f.). *Definition of Android OS*. TechTarget

https://www.techtarget.com/searchmobilecomputing/definition/Android-OS

Consultado por última vez el 17 de Junio de 2024

[76] García, D. (21 de abril de 2021). *Android Things, la versión de Android para el Internet de las Cosas*. La Vanguardia

https://www.lavanguardia.com/andro4all/otros-android/android-things-developer-preview-6

Consultado por última vez el 17 de Junio de 2024

[77] Android. (s.f.). *Introducción a Android Studio*

https://developer.android.com/studio/intro?hl=es-419

Consultado por última vez el 18 de Junio de 2024

[78] Android. (s.f.). *Desarrolla apps para Android con Kotlin*

https://developer.android.com/kotlin?hl=es-419

Consultado por última vez el 18 de Junio de 2024

[79] Kumar, A. (24 de marzo de 2024). *Advantages and Disadvantages of Kotlin Programming Language*. LinkedIn

https://www.linkedin.com/pulse/advantages-disadvantages-kotlin-amit-kumar-lzoff#:~:text=Overall%2C%20Kotlin's%20advantages%2C%20such%20as,risk%20of%20common%20programming%20errors.

Consultado por última vez el 18 de Junio de 2024

[80] Android. (s.f.). *Notas de la versión de las Herramientas de la plataforma del SDK*

https://developer.android.com/tools/releases/platform-tools?hl=es-419

Consultado por última vez el 18 de Junio de 2024

[81] Gradle. (s.f.). *Gradle Build Tool*

https://gradle.org/

Consultado por última vez el 18 de Junio de 2024

[82] Android. (s.f.). *Android Jetpack*

https://developer.android.com/jetpack?hl=es-419

Consultado por última vez el 18 de Junio de 2024

[83] Google. (s.f.). *ML Kit*

https://developers.google.com/ml-kit?hl=es-419

Consultado por última vez el 18 de Junio de 2024

[84] ML Kit. (s.f.). *Detección de malla facial*

https://developers.google.com/ml-kit/vision/face-mesh-detection?hl=es-419

Consultado por última vez el 18 de Junio de 2024

[85] OpenCV. (s.f.). *Face landmark detection in an image*

https://docs.opencv.org/4.x/d2/d42/tutorial_face_landmark_detection_in_an_image.html

Consultado por última vez el 18 de Junio de 2024

[86] Google for Developers. (s.f.). *Guía de detección de puntos de referencia faciales*

https://ai.google.dev/edge/mediapipe/solutions/vision/face_landmarker?hl=es-419

Consultado por última vez el 18 de Junio de 2024

[87] Neupane, R. (14 de enero de 2024). *Facial Landmark Detection*. Medium

https://medium.com/@RiwajNeupane/facial-landmark-detection-a6b3e29eac5b

Consultado por última vez el 18 de Junio de 2024

[88] Anzalone, L. (26 de febrero de 2019). *Setting-up Dlib and OpenCV for Android*. Medium

https://luca-anzalone.medium.com/setting-up-dlib-and-opencv-for-android-3efdbfcf9e7f

Consultado por última vez el 18 de Junio de 2024

[89] Eclipse. (s.f.). *Paho*

https://eclipse.dev/paho/

Consultado por última vez el 19 de Junio de 2024

[90] Google for Developers. (s.f.). *Introducción general a Bluetooth*

https://developer.android.com/develop/connectivity/bluetooth?hl=es-419

Consultado por última vez el 19 de Junio de 2024

[91] Servicio de Información sobre Discapacidad. (s.f.). *EVA Facial Mouse*

https://sid-inico.usal.es/recursos_internet/eva-facial-mouse/

Consultado por última vez el 20 de Junio de 2024

[92] Orientatech. (noviembre de 2020). *EVA Facial Mouse PRO*

https://orientatech.es/eva-facial-mouse-pro/

Consultado por última vez el 20 de Junio de 2024

[93] Wikinclusion. (2 de diciembre de 2021). *EVA Facial Mouse PRO*

https://wikinclusion.org/index.php/EVA_FACIAL_MOUSE_PRO

Consultado por última vez el 20 de Junio de 2024

[94] Android Accessibility Help. (s.f.). *Control your Android device with Switch Access*

https://support.google.com/accessibility/android/answer/6122836?hl=en

Consultado por última vez el 20 de Junio de 2024

[95] Linder, B. (6 de junio de 2017). *Switch Access in Google Android O lets you control pones from a physical switch*. Liliputing

https://liliputing.com/switch-access-google-android-o-lets-control-phones-physical-switch/

Consultado por última vez el 20 de Junio de 2024

[96] Google. (s.f.). *Empezar a utilizar los comandos de voz de Voice Access*

https://support.google.com/accessibility/android/answer/6151848?hl=es&ref_topic=6151842&sjid=12628465443511621 05-EU

Consultado por última vez el 30 de Junio de 2024

[97] Android. [Android] (3 de diciembre de 2020). *Navigate your pone by speaking aloud with Voice Access on Android* [Video]. Youtube

https://www.youtube.com/watch?v=3RyjUwzXWTs

Consultado por última vez el 30 de Junio de 2024

[98] Android. (s.f.). *Primeros pasos con Project Activate*

https://support.google.com/accessibility/android/answer/11348365?hl=es

Consultado por última vez el 30 de Junio de 2024

[99] Android. [Android] (23 de septiembre de 2021). *Communicate using facial gestures and preset actions* [Video]. Youtube

https://www.youtube.com/watch?v=ItyAEbCIjCU

Consultado por última vez el 30 de Junio de 2024

[100] TIC Salut Social. (27 de noviembre de 2019). *Dwell Click*

https://ticsalutsocial.cat/es/app/dwell-click/

Consultado por última vez el 30 de Junio de 2024

[101] Spinal Cord Team. (18 de enero de 2020). *Using a Smartphone Touch-Free with Jabberwocky*

https://www.spinalcord.com/blog/using-a-smartphone-touch-free-with-jabberwocky#:~:text=For%20supported%20devices%20on%20Android,and%20all%20existing%20Android%20apps.

Consultado por última vez el 30 de Junio de 2024

[102] MP, R. (23 de octubre de 2022). *Aberete Sesamo, una app de control por visión para Android*. Accesibles

https://www.accesibles.org/abrete-sesamo-una-app-de-control-por-vision-para-android/

Consultado por última vez el 30 de Junio de 2024

[103] Sesame Enable. [Sesame Enable] (12 de noviembre de 2017). *Open Sesame!* [Video]. Youtube

https://www.youtube.com/watch?v=EWC6iis--kY

Consultado por última vez el 30 de Junio de 2024

[104] Grupo ADD. (2020). Robot My Spoon

https://grupoadd.es/el-brazo-robot-my-spoon#:~:text=El%20robot%20Robotarm%20My%20Spoon,ni%20beber%20sin%20necesitar%20ayuda.

Consultado por última vez el 30 de Junio de 2024

[105] Robotic-Lab. (s.f.). *My Spoon, Robot que te ayuda a comer*

https://www.robotic-lab.com/blog/2007/08/22/my-spoon-robot-que-ayuda-a-comer/

Consultado por última vez el 30 de Junio de 2024

[106] Kids Web Japan. (octubre de 2009). *Now There's a Robot to Help Physically Challenged People Eat*

https://web-japan.org/kidsweb/hitech/helperrobot/index.html

Consultado por última vez el 30 de Junio de 2024

[107] Adaptado. (15 de octubre de 2015). *Bestic: Robot de Asistencia para Comer*

https://www.adaptado.es/bestic-robot-de-asistencia-para-comer/

Consultado por última vez el 30 de Junio de 2024

[108] Mealtime Partners Inc. (18 de julio de 2020). *The Mealtime Partner Dining System Description*

https://www.mealtimepartners.com/dining/mealtime-partner-dining-device.htm

Consultado por última vez el 30 de Junio de 2024

[109] Parra-Dominguez, G. S., Sanchez-Yanez, R. E., & Garcia-Capulin, C. H. (31 de marzo de 2022). *Towards Facial Gesture Recognition in Photographs of Patients with Facial Palsy*

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9031481/

Consultado por última vez el 1 de Julio de 2024

[110] Alabdullah, B. I. (23 de septiembre de 2023). *Smart Home Automation-Based Hand Gesture Recognition Using Feature Fusion and Recurrent Neural Network*

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10490576/

Consultado por última vez el 1 de Julio de 2024

[111] Dang, T. (noviembre de 2022). *Smart Home Management System With Face Recognition Based On ArcFace Model in Deep Convolutional Neural Network*

https://www.researchgate.net/publication/365362063_Smart_home_Management_System_with_Face_Recognition_based_on_ArcFace_model_in_Deep_Convolutional_Neural_Network

Consultado por última vez el 1 de Julio de 2024

[112] Tanweer, M. R. (diciembre de 2022). *Real-Time Intelligent Facial Expression Recognition System*

https://www.researchgate.net/publication/366354691_Real-Time_Intelligent_Facial_Expression_Recognition_System

Consultado por última vez el 1 de Julio de 2024

[113] Ramírez, I. (18 de mayo de 2021). *Cómo instalar Android Studio en tu PC en cinco sencillos pasos*. Xataka

https://www.xatakandroid.com/tutoriales/como-instalar-android-studio-tu-pc-cinco-sencillos-pasos

Consultado por última vez el 1 de Julio de 2024

[114] Android Developers. (s.f.). *Cómo crear y editar configuraciones de ejecución y depuración*

https://developer.android.com/studio/run/rundebugconfig?hl=es-419

Consultado por última vez el 1 de Julio de 2024

[115] Android Developers. (s.f.). *Get to know the Android Studio UI*

https://developer.android.com/studio/intro/user-interface

Consultado por última vez el 1 de Julio de 2024

[116] Android Developers. [Android Developers] (22 de noviembre de 2023). *New UI for Android Studio* [Video]. Youtube

https://www.youtube.com/watch?v=K1TTzkToDyE

Consultado por última vez el 1 de Julio de 2024

[117] Android Developers. (s.f.). *Descripción general del manifiesto de la app*

https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419

Consultado por última vez el 1 de Julio de 2024

[118] Android Developers. (s.f.). *Profile your app performance*

https://developer.android.com/studio/profile

Consultado por última vez el 1 de Julio de 2024

[119] Android Developers. (s.f.). *View logs with Logcat*

https://developer.android.com/studio/debug/logcat

Consultado por última vez el 1 de Julio de 2024

[120] Hakigura, T. (27 de enero de 2023). *See Crashlytics issue reports directly in Android Studio with App Quality Insights*. Medium

https://medium.com/androiddevelopers/see-crashlytics-issue-reports-directly-in-android-studio-with-app-quality-insights-db0ff27454f0

Consultado por última vez el 1 de Julio de 2024

[121] Ridge, B. V. (14 de noviembre de 2023). *Configuración Inicial de Android Studio: Guía paso a paso para principiantes*. MBlog

https://www.mediummultimedia.com/apps/como-configurar-android-studio-por-primera-vez/

Consultado por última vez el 1 de Julio de 2024

[122] Nordquist, T. (s.f.). *MQTT Explorer*

https://mqtt-explorer.com/

Consultado por última vez el 1 de Julio de 2024

[123] Amazon Web Services (AWS). (s.f.). *¿Cuál es la diferencia entre e front end y el back end en el desarrollo de aplicaciones?*

https://aws.amazon.com/es/compare/the-difference-between-frontend-and-backend/#:~:text=El%20front%20end%20es%20aquello,permiten%20que%20la%20aplicaci%C3%B3n%20funcione.

Consultado por última vez el 2 de Julio de 2024

[124] García, J. (16 de noviembre de 2021). *Samsung Galaxy Tab A8 (2021): La nueva Tablet de Samsung es más grande, más potente y más enfocada a ver contenido.* Xataka

https://www.xataka.com/tablets/samsung-galaxy-tab-a8-2021-caracteristicas-precio-ficha-tecnica

Consultado por última vez el 2 de Julio de 2024

[125] Ramírez, I. (24 de julio de 2023). *OPPO A58 4G: un móvil de gama media con pantalla Full HD+, Helio G88 y mucha batería*. Xataka

https://www.xatakamovil.com/oppo/oppo-a58-4g-caracteristicas-precio-ficha-tecnica

Consultado por última vez el 2 de Julio de 2024

[126] Fernández, S. (30 de abril de 2020). *Así es el procesador MediaTek Helio G85 al mando del nuevo Xiaomi Redmi Note 9*. Xataka

https://www.xatakamovil.com/procesadores/asi-procesador-mediatek-helio-g85-al-mando-nuevo-xiaomi-redmi-note-9

Consultado por última vez el 2 de Julio de 2024

[127] PCComponentes. (s.f.). *OPPO A58 6/128GB Negro Libre*

https://www.pccomponentes.com/oppo-a58-6-128gb-negro-libre

Consultado por última vez el 2 de Julio de 2024

[128] Amazon. (s.f.). *FDTEK Enchufe Inteligente WiFi con monitor de consumo de energía y control de voz Smart Home Plug, compatible con Alexa y Google Home Outlet*

https://www.amazon.es/FDTEK-Enchufe-Inteligente-ZigBee/dp/B0BKL4NLPG?th=1

Consultado por última vez el 2 de Julio de 2024

[129] IKEA. (s.f.). *Lámpara TERTIAL*

https://www.ikea.com/es/es/p/tertial-lampara-flexo-trabajo-beige-20507729/#content

Consultado por última vez el 2 de Julio de 2024

[130] Lámpara Directa. (s.f.). *Bombillas LED E27*

https://www.lamparadirecta.es/bombillas-led/e27

Consultado por última vez el 2 de Julio de 2024

[131] IKEA. (s.f.). *Bombilla Solhetta*

https://www.ikea.com/es/es/p/solhetta-bombilla-led-e27-470-lumenes-forma-globo-blanco-opalo-60564138/

Consultado por última vez el 2 de Julio de 2024

[132] DFRobot. (s.f.). *FireBeetle ESP32 IoT Microcontroller*

https://www.dfrobot.com/product-1590.html

Consultado por última vez el 2 de Julio de 2024

[133] Rohde & Schwarz. (s.f.). *Qué es UART*

https://www.rohde-schwarz.com/es/productos/test-y-medida/essentials-test-equipment/digital-oscilloscopes/que-es-uart_254524.html#:~:text=UART%20(universal%20asynchronous%20receiver%20%2F%20transmitter,y%20recibir%20en%20ambas%20direcciones

Consultado por última vez el 2 de Julio de 2024

[134] Aprendiendo Arduino. (9 de julio de 2017). *I2C*

https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/

Consultado por última vez el 2 de Julio de 2024

[135] Random Nerd Tutorials. (s.f.). *ESP32 SPI Communication*

https://randomnerdtutorials.com/esp32-spi-communication-arduino/

Consultado por última vez el 2 de Julio de 2024

[136] MOUSER. (s.f.). *C547B Datasheets*

https://www.mouser.es/c/ds/semiconductors/?q=c547b

Consultado por última vez el 2 de Julio de 2024

[137] Android Developers. (s.f.). *Activity*

https://developer.android.com/reference/kotlin/android/app/Activity

Consultado por última vez el 2 de Julio de 2024

[138] Android Developers. (s.f.). *Introducción a las actividades*

https://developer.android.com/guide/components/activities/intro-activities?hl=es-419

Consultado por última vez el 2 de Julio de 2024

[139] Morales, K. (2 de enero de 2024). *Activity vs Fragment*. Medium

https://medium.com/@kevinhomorales/activity-vs-fragment-162fe98992e6

Consultado por última vez el 2 de Julio de 2024

[140] Android Developers. (s.f.). *Fragments*

https://developer.android.com/guide/fragments?hl=es-419

Consultado por última vez el 2 de Julio de 2024

[141] Material Design. (s.f.). *App bars*

https://m2.material.io/components/app-bars-top#anatomy

Consultado por última vez el 2 de Julio de 2024

[142] Android Developers. (s.f.). *Panel lateral de navegación*

https://developer.android.com/develop/ui/compose/components/drawer?hl=es-419

Consultado por última vez el 2 de Julio de 2024

[143] Danielme. (10 de junio de 2023). *Diseño Android: Menú Lateral Navigation Drawer*

https://danielme.com/2018/12/19/diseno-android-menu-lateral-con-navigation-drawer/

Consultado por última vez el 2 de Julio de 2024

[144] Android Developers. (s.f.). *Intent*

https://developer.android.com/reference/android/content/Intent

Consultado por última vez el 2 de Julio de 2024

[145] Android Developers. (s.f.). *Administrador de fragmentos*

https://developer.android.com/guide/fragments/fragmentmanager?hl=es-419

Consultado por última vez el 2 de Julio de 2024

[146] GeeksForGeeks. (10 de marzo de 2021). *Diferencia entre un fragmento y una actividad en Android*

https://www.geeksforgeeks.org/difference-between-a-fragment-and-an-activity-in-android/

Consultado por última vez el 3 de Julio de 2024

[147] Necanli, B. (21 de octubre de 2023). *Single Activity vs Multiple Activities Architecture*. Medium

https://medium.com/appcent/a-single-activity-vs-multiple-activities-architecture-96a23b783036

Consultado por última vez el 2 de Julio de 2024

[148] Android Developers. (s.f.). *SharedPreferences*

https://developer.android.com/reference/android/content/SharedPreferences

Consultado por última vez el 3 de Julio de 2024

[149] Android Developers. (s.f.). *Cómo guardar datos simples con SharedPreferences*

https://developer.android.com/training/data-storage/shared-preferences?hl=es-419

Consultado por última vez el 3 de Julio de 2024

[150] SQLite. (s.f.). *What is SQLite?*

https://www.sqlite.org/

Consultado por última vez el 3 de Julio de 2024

[151] Android Developers. (s.f.). *Cómo guardar datos con SQLite*

https://developer.android.com/training/data-storage/sqlite?hl=es-419

Consultado por última vez el 3 de Julio de 2024

[152] Develou. (s.f.). *Desplazar Contenido Con El Scrollview En Android*

https://www.develou.com/scrollview-en-android/

Consultado por última vez el 3 de Julio de 2024

[153] Javatpoint. (s.f.). Kotlin Android AlertDialog

https://www.javatpoint.com/kotlin-android-alertdialog

Consultado por última vez el 3 de Julio de 2024

[154] QuickPose. (s.f.). *Mediapipe VS ML Kit: A Comparison of Pose Estimation Tools*

https://quickpose.ai/faqs/mediapipe-vs-ml-kit/

Consultado por última vez el 4 de Julio de 2024

[155] Android Developers. (s.f.). *Handler*

https://developer.android.com/reference/android/os/Handler

Consultado por última vez el 3 de Julio de 2024

[156] McLibre. (17 de julio de 2020). *Qué es JSON*

https://www.mclibre.org/consultar/informatica/lecciones/formato-json.html

Consultado por última vez el 6 de Julio de 2024

[157] Android Developers. (s.f.). *Permisos de Bluetooth*

https://developer.android.com/develop/connectivity/bluetooth/bt-permissions?hl=es-419

Consultado por última vez el 5 de Julio de 2024

[158] KeepCoding. (23 de abril de 2024). ¿Qué es un socket?

https://keepcoding.io/blog/que-es-un-socket/

Consultado por última vez el 6 de Julio de 2024

[159] Heerink M., Kröse B., Evers V. (2010) Assesing Acceptance of Assistive Social Agent Technology by Older Adults: the Almere Model. *International Journal of Social Robotics*. 2, 361-375.

https://doi.org/10.1007/s12369-010-0068-5

Consultado por última vez el 7 de julio de 2024

[160] Talent. (s.f.). *Salario medio para Ingeniero Junior en España 2024*

https://es.talent.com/salary?job=ingeniero+junior#:~:text=%C2%BFCu%C3%A1nto%20gana%20un%20Ingeniero%20junior%20en%20Espa%C3%B1a%3F&text=El%20salario%20ingeniero%20junior%20promedio,hasta%20%E2%82%AC%2031.895%20al%20a%C3%B1o.

Consultado por última vez el 8 de Julio de 2024

[161] Personio. (s.f.). *El coste de los trabajadores para tu empresa: ¿qué es y cómo calcularlo?*

https://www.personio.es/glosario/coste-de-trabajador-para-empresa/

Consultado por última vez el 9 de Julio de 2024

# 10. ANEXO

Se han incluido tanto el código completo de la aplicación como documentos de interés como detalles técnicos del robot de alimentación OBI o del microcontrolador ESP32 en una carpeta de anexos externa este documento.