Memoria del proyecto

App para la navegación basada en comandos de voz del robot iRobot Create 3

Trabajo de Fin de Grado de Ingeniería Informática

VNiVERSiDAD D SALAMANCA

Julio de 2023

Autor:

Jorge Sánchez Rubio

Tutores:

Francisco Javier Blanco Rodríguez

Belén Curto Diego

Certificado de los tutores TFG

Dña. Belén Curto Diego, profesora del Departamento de Informática y Automática de la Universidad de Salamanca y

D. Fco Javier Blanco Rodríguez, profesor del Departamento de Informática y Automática de la Universidad de Salamanca,

HACEN CONSTAR:

Que el trabajo titulado "App para la navegación basada en comandos de voz del robot iRobot Create 3", que se presenta, ha sido realizado por Jorge Sánchez Rubio, con DNI 44412321B y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Ingeniería Informática en esta Universidad.

Salamanca, 5 de julio de 2023

CURTO DIEGO MARIA BELEN - 07855759V
Firmado digitalmente por CURTO DIEGO MARIA BELEN - 07855759V
Fecha: 2023.07.05 14:09:01 +02'00'

Fdo.: Belén Curto Diego

BLANCO RODRIGUEZ FRANCISCO JAVIER - 07965219C
Firmado digitalmente por BLANCO RODRIGUEZ FRANCISCO JAVIER - 07965219C
Fecha: 2023.07.05 18:01:56 +02'00'

Fdo.: Fco Javier Blanco Rodríguez

Resumen

En los últimos años, los Sistemas de Control por Voz y la Inteligencia Artificial han revolucionado la forma que tenemos de interactuar con los distintos dispositivos electrónicos que se utilizan en la vida cotidiana.

Desde dispositivos inteligentes como móviles, altavoces, robots e incluso electrodomésticos y coches, gracias al creciente desarrollo de la Inteligencia Artificial y la integración de Sistemas de Control por Voz, muchas de las tareas cotidianas se han vuelto cada vez más sencillas y accesibles. Los fabricantes de este tipo de dispositivos inteligentes quieren que el usuario interactúe con ellos, por lo que apuestan por sistemas de control por voz como Google Assistant, Siri (Apple) o Alexa (Amazon), dotados de Inteligencia Artificial [1] y que se integran en dispositivos electrónicos de manera que los usuarios puedan darles órdenes para controlarlos y realizar tareas.

iRobot Create 3 es un robot educativo basado en el robot aspiradora Roomba e ideado para profesores, estudiantes y desarrolladores. Incorpora un completo conjunto de sensores y actuadores que permite desarrollar y probar los algoritmos para la navegación de robots que se hayan diseñado con el fin de que realice una tarea. Su software está basado en ROS2 (*Robot Operating System*), que es un conjunto de bibliotecas y herramientas software para construir aplicaciones para robots [2] .

El objetivo de este Trabajo de Fin de Grado es desarrollar una aplicación Android para que el robot iRobot Create 3 realice movimientos por medio de comandos de voz dictados por el usuario. A través de una aplicación Android se recogen las órdenes de voz dictadas por el usuario, siguiendo un conjunto de comandos preestablecido, y se envían a un servidor de ROS2 en un ordenador para que sean ejecutadas posteriormente por el robot. Tanto el móvil como el robot sirven como interfaz HMI (Human-Machine Interface) de forma que el usuario recibe una realimentación por parte del robot y de la aplicación móvil para saber si la orden dada por el usuario se ha recibido y entendido correctamente o de si se ha producido algún error.

Palabras clave: comandos de voz, robot, IA, ROS2, Android, Create 3

Abstract

In the last few years, Voice Control Systems and Artificial Intelligence have revolutionised the way we interact with the different electronic devices used in the daily life.

From intelligent devices as smartphones, speakers, robots and even home appliances and cars; thanks to the growing development of AI and the integration of Voice Control Systems, many of the daily tasks we do have become increasingly simple and accessible. The manufacturers of this type of smart devices want the user to interact with them, so they opt for Voice Control Systems as Google Assistant, Siri (Apple) or Alexa (Amazon), equipped with AI and integrated inside electronic devices so users can give them orders to control them and perform tasks.

iRobot Create 3 is an educational robot based on the robot vacuum Roomba, and designed for teachers, students and developers. It encompasses a complete set of sensors and actuators that allows algorithms to be developed and tested for the navigation of robots that have been designed to perform a task. Its software is based on ROS2 (Robot Operating System), which is a set of libraries and software tools to build applications for robots.

The goal of this Final Degree Project is the development of an Android app for the iRobot Create 3 in order to carry out movements through voice commands dictated by the user. Through an Android app, the voice orders dictated by the user are collected following a set of pre-established commands and are sent to a ROS2 server on a computer to be subsequently executed by the robot. Both the smartphone and the robot serve as an HMI (Human-Machine-Interface) so that the user receives feedback from the robot and the smartphone's app to find out if the order has been received and understood correctly or if some error has occurred.

Key words: voice commands, robot, AI, ROS2, Android, Create 3

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Actualmente los Sistemas de Control por Voz junto con la Inteligencia Artificial han revolucionado la forma que tenemos de interactuar con los distintos dispositivos electrónicos que nos rodean en la vida cotidiana. Estos sistemas son una herramienta muy útil que permite a los usuarios controlar dispositivos con el simple uso de la voz. Por ello se les conoce como asistentes virtuales.

Hoy en día, Sistemas de Control por Voz dotados de Inteligencia Artificial como Google Assistant, Siri (Apple), o Alexa (Amazon), entre otros, se integran en dispositivos como relojes, móviles, altavoces, electrodomésticos, televisores e incluso coches. Esto permite una comunicación natural y directa con los dispositivos y hace que muchas tareas se hayan vuelto cada vez más sencillas y accesibles, evitando el uso de interfaces de usuario complejas y comandos enrevesados [3].

A continuación, se presentan ejemplos de la aplicación del control por voz en dispositivos, tanto en el ámbito comercial como académico, algunos de los cuales están muy relacionados con el proyecto a desarrollar.

- **Altavoces inteligentes.** Entre sus funciones destacan responder a dudas del usuario o ayudar con tareas, recordatorios, reproducir música, así como manejar la domótica de la casa, conectándose a diferentes dispositivos, como puede ser una bombilla, y gestionándolos como si fuera un centro de mando [4] . También ofrecen prestaciones en cuanto a la seguridad en el hogar, como el control de cámaras o la simulación de presencia de personas en casa. Entre los disponibles en el mercado, algunos de los más conocidos son Google Home, que cuenta con el Asistente de Google, Amazon Echo, con su asistente Alexa, o Apple Homepad, que apuesta por el asistente Siri.

Figura 1: Altavoz inteligente Amazon Echo Dot

- **Reloj Inteligente o Smartwatch.** Existen muchos relojes inteligentes que ofrecen soporte para los principales asistentes de voz mencionados anteriormente. Entre sus ventajas destacan la activación en todo momento del asistente, lo que se conoce como escucha activa. Por ejemplo, si el usuario necesita tomar nota de algo en un momento dado, basta con decirle al asistente de voz que apunte lo que el usuario diga [5]. También pueden ofrecer la posibilidad de realizar o recibir llamadas y activar otras funcionalidades que el reloj ofrezca. Algunos de los más conocidos en el mercado son los siguientes: Apple Watch Series 6 (Siri), Amazfit GTR 2 (Alexa), Samsung Galaxy Watch3 (Bixby).

Figura 2: Apple Watch Series 6

- **Robots aspiradora.** Muchas empresas creadoras de robots aspiradores inteligentes, como iRobot, han integrado los Sistemas de Control por Voz para el control de sus productos por parte de los usuarios. La empresa iRobot, que es la fabricante del robot con el que se trabaja en este proyecto, ofrece la aplicación móvil iRobot Home para que los usuarios finales manejen los Roomba. Estos robots tienen

su propio sistema operativo iRobot OS [6] . Esta aplicación permite, entre otras funciones, controlar la limpieza a través de comandos de voz, añadiendo accesos directos en la aplicación al asistente de voz del usuario, como puede ser Alexa, Siri o el Asistente de Google [7] . La disponibilidad de funciones varía según el modelo del robot y proporciona al usuario una experiencia de limpieza a su medida, teniendo en cuenta sus necesidades.

Figura 3: Robot Aspirador Roomba

- **RemoteBot.** Es una aplicación cliente-servidor que permite controlar los robots del proyecto “Programando con robots y Software libre” mediante dispositivos móviles Android de la Facultad de Informática de la Universidad Nacional de La Plata (Argentina) [8] .

- **Hans (Robot controlado por voz).** Este proyecto de la Universidad Rey Juan Carlos [9] consiste en un robot capaz de realizar determinadas acciones, las cuales son ordenadas mediante voz. El robot realiza una acción una vez se dice su nombre como palabra clave. Cuando recibe uno de los 8 comandos que entiende, enciende dos diodos LED para la confirmación, lo que sirve como realimentación al usuario. Se caracteriza por usar un módulo de reconocimiento de voz (VR3) que es el encargado de almacenar los comandos, captarlos a través de la voz y de transmitirlos.

El objetivo de este Trabajo de Fin de Grado es desarrollar una aplicación para que el robot iRobot Create 3 realice los movimientos que el usuario desee a través de un conjunto de comandos de voz preestablecido. Mediante una aplicación Android que se ejecuta en un teléfono móvil se recogerán y se interpretarán los comandos de voz del usuario. Posteriormente, estos comandos se enviarán a un servidor de ROS2 en un ordenador para que sean ejecutadas por el robot. ROS (Robot Operating System) es un conjunto de bibliotecas y herramientas software para construir aplicaciones para robots y se emplea para establecer la comunicación con el robot, así como para que el robot ejecute los comandos de movimiento que le da el usuario y recoger el estado de los sensores. Se explicará en detalle qué es ROS y cómo funciona en un apartado propio en la sección de conceptos teóricos.

La finalidad de esta memoria es explicar los aspectos del desarrollo del sistema. Estos son los apartados en los que se estructura:

- **Objetivos del proyecto:** explicación de los objetivos que se persiguen con la realización del proyecto.
- **Conceptos teóricos:** explicación de los conceptos teóricos necesarios para la comprensión y desarrollo del proyecto.
- **Técnicas y herramientas:** documentación sobre las técnicas metodológicas y herramientas de desarrollo utilizadas para llevar a cabo el proyecto.
- **Aspectos relevantes del desarrollo:** descripción de los aspectos más importantes del desarrollo, diseño e implementación.
- **Conclusiones:** descripción de las conclusiones que se obtienen tras el desarrollo del proyecto.
- **Líneas de trabajo futuras:** líneas de trabajo futuras o mejoras propuestas para la continuación del proyecto.
- **Referencias:** citas empleadas en el desarrollo del proyecto.

A esta memoria se incluye el siguiente conjunto de anexos que constituye la documentación técnica del proyecto:

- **Anexo I. Planificación temporal**
- **Anexo II. Especificación de requisitos software**
- **Anexo III. Especificación de diseño**
- **Anexo IV. Documentación técnica de programación**
- **Anexo V. Manual de Usuario**

2. OBJETIVOS DEL PROYECTO

En este apartado se detallan los objetivos del sistema y personales a cumplir en el desarrollo del proyecto.

2.1. Objetivos del sistema

Los principales objetivos del sistema son el diseño, desarrollo e implementación de una aplicación para el control por voz del iRobot Create 3.

- **Definición de un diccionario de órdenes de movimiento** en lenguaje natural para que el robot realice una tarea.
- **Definición de un protocolo de interacción** entre el usuario y el robot.
- **Reconocimiento de órdenes:** el sistema debe ser capaz de reconocer por voz un conjunto de órdenes de movimiento preestablecido utilizando un terminal móvil.
- **Envío de órdenes:** el sistema debe ser capaz de enviar las órdenes a un ordenador/servidor para su gestión.
- **Gestión de órdenes:** el sistema debe ser capaz de gestionar las órdenes de manera que puedan ser ejecutadas por el robot para su movimiento.
- **Realimentación al usuario:** el sistema debe ser capaz de proporcionar al usuario una realimentación al procesar una orden para informarlo de si se ha reconocido la orden, si es la correcta o si no se ha reconocido.
- **Ejecución de órdenes:** el sistema debe ser capaz de que el robot ejecute correctamente las órdenes dictadas por el usuario.
- **Compatibilidad del sistema:** el sistema debe ser compatible con una amplia gama de dispositivos móviles Android.
- **Escalabilidad:** el sistema debe ser escalable de manera que se pueda adaptar adecuadamente al futuro crecimiento de la aplicación.
- **Usabilidad:** el sistema debe proporcionar al usuario una interacción sencilla, intuitiva y una realimentación para que este pueda saber qué está ocurriendo.

2.2. Objetivos personales

En este apartado se explican los motivos personales que me han llevado a realizar este Trabajo de Fin de Grado. El mundo de la robótica, la inteligencia artificial y el desarrollo de aplicaciones es algo por lo que siempre he sentido interés. De manera que estos son los principales objetivos personales que me propuse:

- Comprender cómo es de verdad el desarrollo de un proyecto de estas características, poniendo en práctica todos los conocimientos aprendidos en el Grado de Ingeniería Informática.
- Aprender a desarrollar aplicaciones Android. Comprender su SDK, su entorno de desarrollo Android Studio y el lenguaje de programación Python, que es muy utilizado hoy en día sobre todo en el desarrollo de sistemas robóticos y la Inteligencia Artificial.
- Conseguir un producto que sea útil y fácil de usar para el usuario, de manera que se sienta satisfecho y que le proporcione una realimentación adecuada.
- Mejorar mi capacidad para buscar soluciones a los problemas que vayan surgiendo durante el desarrollo del proyecto, así como la mejora de mi capacidad de búsqueda de recursos imprescindibles.

3. CONCEPTOS TEÓRICOS

En este apartado se describen los principales conceptos teóricos que se han empleado y que pueden servir de ayuda al lector para la comprensión del desarrollo del proyecto.

3.1. Aplicación móvil nativa

Una aplicación nativa es aquella que se desarrolla para un sistema operativo específico y, por lo tanto, tiene un lenguaje de programación específico. En el caso de este proyecto el SO elegido para la aplicación sobre el dispositivo móvil ha sido Android con el lenguaje Java [10] .

Entre algunas de las ventajas de este tipo de aplicaciones se encuentran la rapidez, una mejor experiencia con el usuario, mayor rendimiento y funcionamiento sin conexión a internet.

Entre algunas de las desventajas se encuentran el coste elevado de tener que realizar la aplicación para cada sistema operativo, una mayor complejidad a la hora de desarrollar y, por lo tanto, mayor tiempo de desarrollo.

La principal diferencia entre una aplicación nativa y una aplicación híbrida (Figura 4) es que las aplicaciones híbridas se adaptan a cualquier plataforma porque están desarrolladas con un patrón de diseño responsivo. De manera que, al desarrollar la aplicación (por ejemplo, web), esta se adapta a cualquier dispositivo, sin importar el tamaño o la resolución de este. Además, en una aplicación híbrida se emplean lenguajes web como HTML, CSS o JavaScript que actúan en una capa adicional entre la aplicación y el sistema operativo.

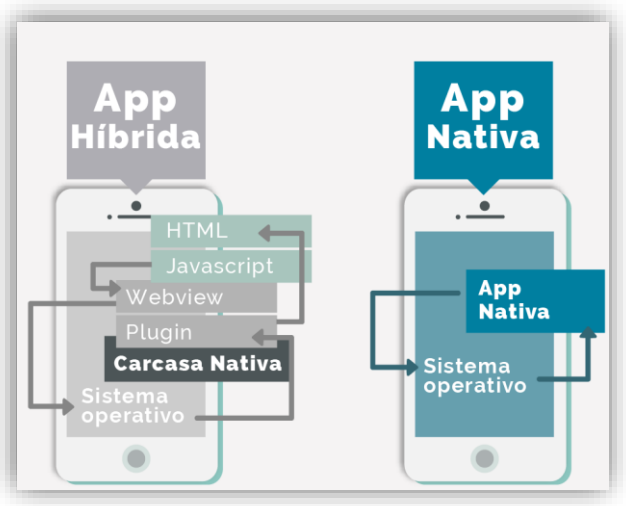

Figura 4: Diferencia entre aplicación nativa e híbrida

3.2. Tecnologías de voz

Las tecnologías de voz (en inglés *Voice Tech*) hacen referencia a todos esos dispositivos electrónicos que se ponen en funcionamiento a través de la voz gracias a la Inteligencia Artificial. En este aspecto, una de las características más interesantes y de mayor uso de la IA es el llamado Procesamiento de Lenguaje Natural (PLN), que consiste en la interacción de las máquinas con los usuarios con un lenguaje cada vez más humano [11]. El PNL utiliza algoritmos de Machine Learning en el texto y el habla. De esta forma, el llamado *Speech to Text* o proceso de reconocimiento de voz da lugar a resultados, como en este caso, son las tecnologías de voz.

Voice User Interface (Interfaz de Usuario de Voz): Siri, Cortana, Alexa, Ok Google … Todos ellos son ejemplos de interfaces de usuario de voz, también conocidos como asistentes de voz o asistentes virtuales. Son el resultado de la implementación del PNL y la mejora del proceso de reconocimiento de voz en IA. Sencillamente consiste en la interacción del usuario y la máquina para darle órdenes a esta última empleando únicamente su voz.

Entre los principales beneficios que se pueden encontrar en el uso de las tecnologías de voz se encuentra la simplificación de todo tipo de tareas tanto diarias como empresariales. Las personas tienen total control sobre los dispositivos utilizando esta tecnología, pudiendo tener las manos libres. Además,

ofrece una mejor experiencia de usuario, evitando interfaces gráficas complejas. Es más cómoda, rápida, sutil y muy eficiente.

Como ya se ha mencionado en la introducción, las tecnologías de voz se encuentran en un gran número de dispositivos de carácter inteligente como altavoces, relojes, televisiones, móviles e incluso coches.

Figura 5: Tecnologías de voz

3.3. ROS2

Este apartado describe detalladamente lo que es ROS y ROS2, su origen y su funcionamiento. Para ello se ha seguido la documentación oficial de ROS como también el trabajo llevado a cabo por el Grupo de Trabajo de Innovación de la AER (Asociación Española de Robótica) [12].

3.3.1. Introducción a ROS

ROS (*Robot Operating System*) surge en un momento en el que el sector de la robótica se encontraba dominado por sistemas cerrados, donde cada robot disponía de su propio sistema operativo y lenguaje de programación. Esto suponía desventajas, como código no reutilizable en otros robots, el continuo aprendizaje de APIs nuevas y, por lo tanto, la obstaculización en los avances en el campo académico. Había que dedicar mucho tiempo y esfuerzo en la puesta en marcha del hardware, lo que dejaba poco tiempo para la investigación y desarrollo en el campo de la robótica.

Sus creadores, dos estudiantes de doctorado de la Universidad de Stanford, Keenan Wyrobek y Eric Berger tenían la visión de hacer de ROS el “Linux de la robótica”.

La primera versión de ROS surge en 2010 y tras los primeros años de desarrollo, la OSRF (Fundación de Robótica de Código Abierto en inglés) se esfuerza en crear ROS2 como un conjunto paralelo de paquetes que pudieran instalarse junto a ROS 1 (ROS original) e interactuar con él.

Con los años, ROS se considera un framework de facto para el desarrollo e investigación de tecnologías robóticas, además de una comunidad global de código abierto formada por ingenieros, desarrolladores y aficionados que contribuyen a su mejora y accesibilidad.

3.3.2. Evolución a ROS2 y Middleware DDS

ROS se diseñó y desarrolló inicialmente dentro de un contexto muy específico y, a través de los años, han ido surgiendo nuevos requerimientos. Para poder implementar más adecuadamente estos requerimientos y dar un mejor soporte sobre todo al mundo industrial y sus necesidades se decide dar el salto a la implementación de ROS2.

ROS2 nace y se apoya en ROS. Muchos de los conceptos principales de ROS, como nodos, *topics*, servicios, o acciones, siguen presentes en ROS2. Sin embargo, ROS2 no es una versión nueva de ROS. El código desarrollado en ROS no es compatible directamente con ROS2 y, en general, no funcionará en ROS2. Por ello, para facilitar la transición de uno a otro, se proporcionan una serie de recursos, entre los que se encuentran la guía de migración de ROS, el nodo *ros1_bridge* que actúa como intermediario de mensajes, o la librería de Python *rospy2*.

El principal cambio que ha impulsado la implementación de ROS2 ha sido la adopción de DDS/RTPS (*Data Distribution Service/ Real Time Publish Subscribe*) [13] como capa de comunicaciones, es decir, como su middleware.

Se trata de un estándar de comunicación definido por el OMG (*Object Manager Group*) que tiene un gran uso en aplicaciones críticas, como sistemas de vuelo y financieros. DDS proporciona características avanzadas de red, como

descubrimiento, serialización y transporte de publicación-suscripción. También, ofrece diferentes implementaciones entre las cuales ROS2 soporta Fast DDS y Cyclone DDS.

A través de DDS como middleware de ROS2 se proporciona un sistema descentralizado y un sistema distribuido de descubrimiento de nodos, lo cual es muy útil a la hora de construir aplicaciones ROS2 que se comuniquen con un sistema robótico que también tenga DDS como su middleware. El sistema distribuido de descubrimiento que implementa DDS registra y anuncia la presencia de nodos en el dominio DDS, permitiendo que dos nodos DDS se puedan comunicar entre ellos sin la necesidad de otro tipo de herramientas [14] y haciendo al sistema más tolerante a fallos y flexible.

Para que dos nodos en ROS2 se encuentren en este contexto deben pertenecer al mismo dominio DDS (espacio lógico en el que se ejecutan las aplicaciones de ROS2) que viene dado por la variable de entorno `ROS_DOMAIN_ID` [15]. Si cada nodo tiene un dominio distinto entonces no se podrán comunicar. Además, a cada nodo en ROS2 se le asigna un nombre único que lo hace identificable en el dominio DDS gracias al sistema de nombres del que dispone ROS2. Esto último se aplica igual para los *topics*, servicios y acciones.

En la Tabla 1 se resumen algunos de los cambios más significativos que hay entre ROS y ROS2.

ROS	ROS2
Capa de comunicación basada en un middleware ad-hoc y TCP como capa de transporte.	Capa de comunicación basada en el estándar DDS, soportando distintas implementaciones de este.
Comunicación centralizada, ya que el Master es responsable de establecer las comunicaciones, lo que lo convierte en un posible punto único de fallo.	DDS permite tener un sistema completamente descentralizado y que el descubrimiento de nodos se realice de forma distribuida.
Sin posibilidad de establecer Calidad del Servicio (QoS, del inglés Quality of Service).	Apoyándose en DDS, ofrece distintas configuraciones de QoS; además es posible tener un mayor control modificando la configuración de la propia implementación de DDS.
Sin soporte multi-robot, principalmente debido a la limitación de la existencia de un único Máster, lo que dificulta la gestión de múltiples sistemas robóticos que trabajen en redes independientes.	La capa de comunicación basada en DDS permite un sistema totalmente distribuido, permitiendo gestionar múltiples redes.
Oficialmente sólo es soportado en Ubuntu, aunque en las últimas versiones también hay paquetes binarios para Windows.	Multiplataforma, el sistema de integración continua que se encarga de verificar y generar los paquetes binarios de ROS2. Soporta Ubuntu, Windows y Mac OS X. Además, gracias al uso de DDS-XRCE (DDS for Extremely Resource Constrained Environments) también es posible utilizar micro-ROS61 para utilizar ROS2 sobre microcontroladores.

Tabla 1: Diferencias entre ROS y ROS2

3.3.3. Descripción y características principales de ROS2

ROS2 [16] es un conjunto de bibliotecas que forman un marco de trabajo (*framework*) de software libre para construir, ejecutar y administrar aplicaciones para robots y sistemas robóticos, proporcionando un conjunto estándar de configuración y una variedad de herramientas para trabajar ya integradas que simplifican la tarea de crear un comportamiento robusto y complejo del sistema.

Concretamente, la distribución de ROS2 que se ha utilizado en este trabajo es Humble; disponible para Ubuntu 22.04, que es la versión de Linux con la que se ha trabajado y cuya instalación puede hacerse fácilmente siguiendo los pasos en la documentación sobre el iRobot Create 3 [17]. Todas estas herramientas y bibliotecas tienen como objetivo simplificar la tarea de crear un comportamiento complejo y potente en una variedad de plataformas robóticas.

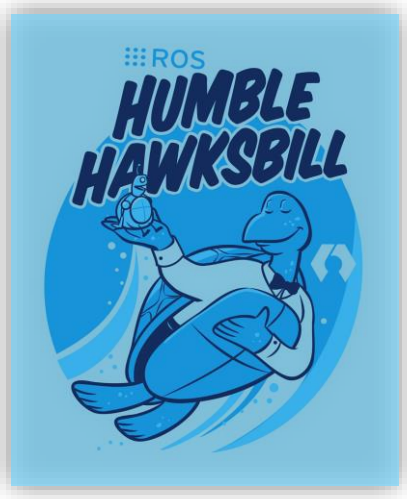

Figura 6: ROS2 Humble

ROS2 se caracteriza por ser multiplataforma y, por lo tanto, compatible para los diferentes sistemas operativos. Además, tiene soporte para múltiples lenguajes de programación, como C++ o Python. Este último es el que se ha decidido emplear para desarrollar el servidor de ROS2 que gestiona las órdenes recibidas.

ROS2 está basado en una arquitectura de grafos con una estructura descentralizada, en donde los procesos se representan mediante nodos que se conectan mediante *topics*. En la Figura 7 aparece un ejemplo de cómo es un

grafo de ROS2 con todos sus elementos procesando información al mismo tiempo.

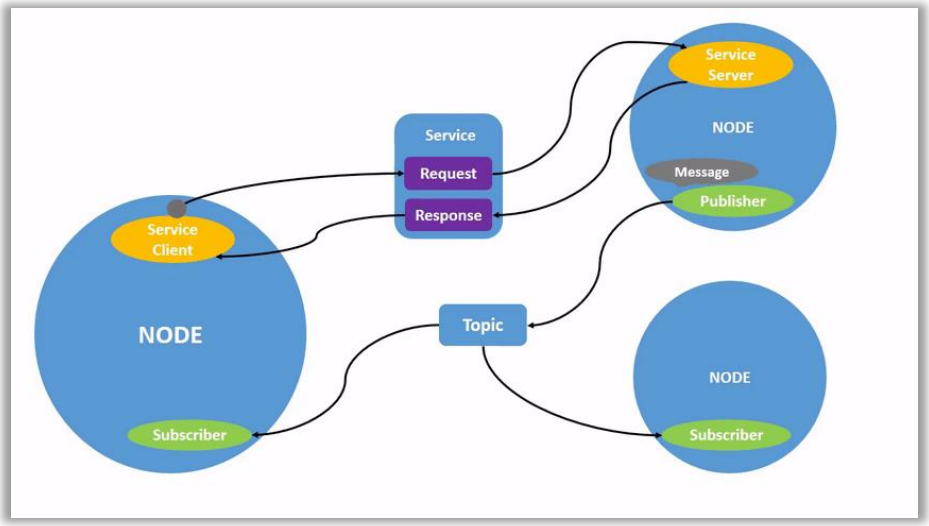

Figura 7: Ejemplo de un grafo de ROS2

Los conceptos principales (entidades) de ROS2 son los siguientes:

- **Nodo.** Un nodo [18] en ROS2 es un proceso individual cuya ejecución es independiente. Cada nodo debe ser único, se comunica con otros nodos a través de mensajes, servicios o acciones y pueden ejecutarse en diferentes máquinas. Un sistema robótico completo está compuesto por múltiples nodos trabajando entre sí, enviándose mensajes a través de los *topics*.
- **Topics.** Los *topics* [19] son una de las formas que tienen los nodos de comunicarse entre ellos de manera asíncrona mediante la publicación de un mensaje en un *topic*, de forma que otro nodo al suscribirse a ese *topic* obtiene el mensaje. Se basan en el modelo publicador-suscriptor.
- **Mensajes.** Los mensajes son estructuras de datos utilizadas para el intercambio de información entre nodos. Algunos son generados por los servicios [19] .
- **Servicios.** Los servicios son una forma de comunicación síncrona entre nodos, permitiéndolos enviar una solicitud y recibir una respuesta [20]. Otra diferencia importante es que los *topics* permiten a los nodos suscribirse a flujos de datos y obtener actualizaciones

continuas, mientras que los servicios sólo proporcionan información cuando son expresamente llamados por un cliente [21] .

- **Acciones.** Las acciones son un tipo de comunicación en ROS2 y consisten en tres partes: un objetivo (*goal*), una realimentación (*feedback*) y un resultado. Se basan en *topics* y servicios y su funcionalidad es similar a la de los servicios, a diferencia de que las acciones pueden cancelarse y proporcionan una respuesta constante [22] .
- **ActionClient.** Un cliente de acción de basa en el envío de una o más metas (una acción que se quiere ejecutar). El encargado de proporcionar una acción es el servidor de acción (*Action Server*).

La comunicación con robot en este proyecto es uno de los aspectos más importantes y es por eso por lo que se ha elegido establecerla mediante el uso de ROS2.

En ROS2 se pueden definir esquemas publicador-suscriptor, de manera que cuando un publicador, por ejemplo, publica un mensaje sobre un *topic* específico, los suscriptores que existan pueden suscribirse a ese *topic* y leer el mensaje publicado.

3.4. iRobot Create 3

El robot empleado para el desarrollo del proyecto es un robot educativo basado en el robot aspiradora Roomba e ideado por la empresa iRobot. Abarca un completo conjunto de sensores y actuadores, que permite desarrollar y probar los algoritmos para la navegación de robots que se hayan diseñado con el fin de realizar tareas. Su software está basado en ROS2. En la Figura 8 se muestra el robot junto a su estación de carga.

Figura 8: iRobot Create y su estación de carga

A continuación, se describe el robot tanto a nivel hardware [24] como software [23] .

Componentes hardware: sensores y actuadores

La parte frontal dispone de un paragolpes (*multi-zone bumper*) con siete sensores infrarrojos de proximidad usados para la detección de obstáculos.

En la parte superior se encuentran tres botones que pueden ser manipulados por el usuario. El botón de encendido cuenta con un anillo de seis luces LED RGB que indican el estado del robot (cargando, inactivo, en punto de acceso, actualizando *firmware*, etc.). Si se pulsan simultáneamente los dos botones que se encuentran a ambos lados del botón de encendido se activa el punto de acceso del robot.

Junto a estos tres botones se encuentra también el sensor de acoplamiento (*docking sensor*) para la detección de la estación de carga de la

batería (*charging station*). En la parte inferior se encuentran cuatro sensores que detectan desniveles (*cliff sensors*), como pueden ser unas escaleras.

A ambos lados de la rueda caster (Figura 9) se encuentran los contactos de carga de la batería. Además, las ruedas principales disponen de interruptores para detectar si el robot ha sido levantado.

Figura 9: Parte inferior del iRobot Create

Las dos ruedas principales que generan el movimiento también disponen de sensores de corriente y encoders. El sensor óptico para la odometría visual, como se muestra en la Figura 10, se encuentra a la derecha del contacto de carga derecho.

Figura 10: Situación de la placa adaptadora del iRobot Create 3

El robot dispone también de una IMU (Inertial Measurement Unit) que se utiliza junto con el sensor óptico de odometría y los encoders de las ruedas para realizar una estimación de la posición del robot (odometría) mediante la fusión de los datos proporcionados por los tres tipos de sensores.

En lo que respecta a los actuadores del robot, en la parte de abajo delantera, en el centro, hay colocada una pequeña rueda giratoria (caster), que sirve como punto de apoyo. Por defecto, el centro de gravedad del robot está por delante del eje central. En la parte de abajo central, a ambos lados, se encuentran en un eje las dos ruedas de dirección, que generan el movimiento del robot gracias al motor que dispone cada una.

Comunicaciones y software

En cuanto a las comunicaciones, el robot tiene dos conexiones eléctricas que proporciona una placa adaptadora (Figura 10 y Figura 11) para la conexión con dispositivos externos a través de Bluetooth o USB-C 2.

También puede configurarse como punto de acceso y proporcionar su propia red para que otros dispositivos se conecten a él. Además, puede conectarse a una determinada red inalámbrica local. El conector USB-C proporciona una conexión de host USB 2.0 en el robot para alimentar las conexiones *downstream* [25].

En la Figura 11 se muestra la placa adaptadora del iRobot Create 3.

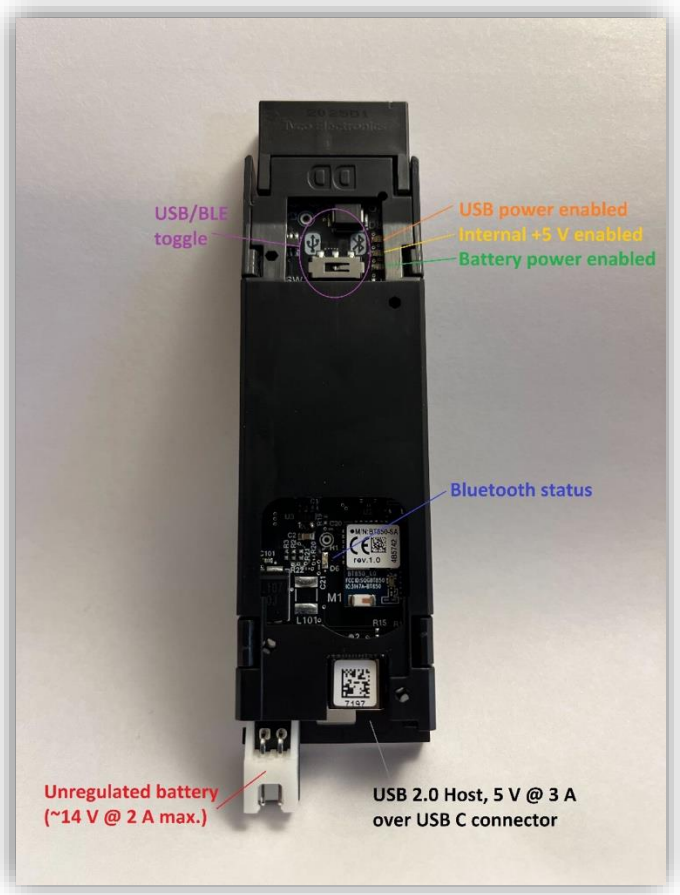

Figura 11: Placa adaptadora del iRobot Create 3

El software del robot se basa en ROS2 y, como tal, presenta todas sus APIs orientadas al usuario a través de entidades ROS2 (*topics*, acciones y servicios) ya explicadas con anterioridad. Además, al basarse en ROS2 implementa un middleware de comunicación DDS, ofreciendo las implementaciones Fast RTPS o Cyclone DDS. En el caso de la versión Humble de ROS2 que se utiliza en el proyecto, Fast RTPS es la que se recomienda en la documentación oficial [26] .

A través de DDS como middleware se proporciona un sistema descentralizado y distribuido para el descubrimiento de nodos que permite que el robot pueda comunicarse con cualquier aplicación ROS2 que tenga la misma versión DDS como middleware.

El sistema operativo del robot utiliza los mensajes estándar de ROS2 cuando están disponibles e implementa los mensajes propios del paquete `irobot_create_msgs` para el acceso a los datos que no están definidos en

los mensajes estándar. Cabe destacar que, además, el sistema de coordenadas de ROS2 es centrado en el robot y dextrógiro, con el eje x apuntando hacia adelante, el eje z hacia arriba y el eje y hacia la izquierda [27].

Por último, es importante comentar en este apartado el servidor web de configuración del robot.

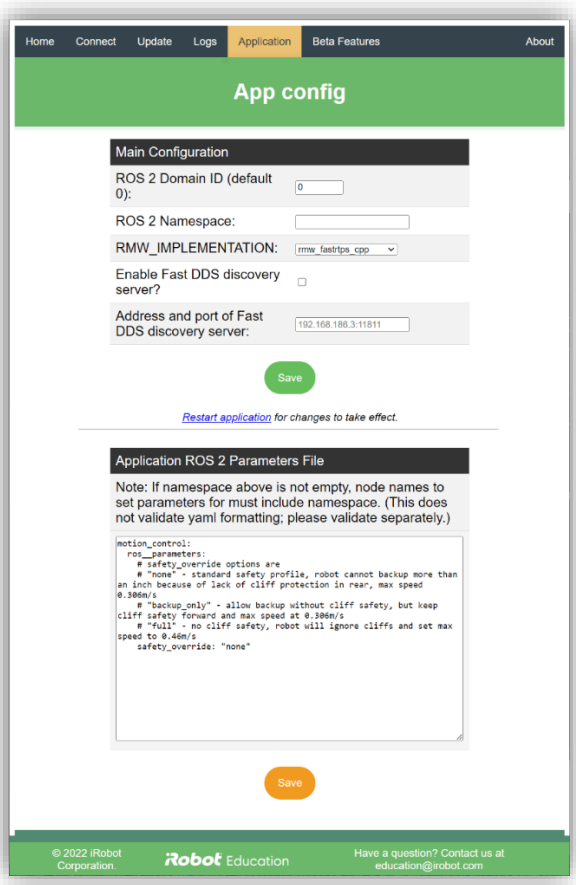

Figura 12: Sección de Application para la configuración del servidor web

iRobot Create 3 ejecuta un servidor web que permite al usuario modificar los ajustes de configuración del robot. Esto es un proceso separado de su aplicación ROS2, ya que se ejecuta independientemente del estado de esta [28] a través del protocolo HTTP.

El usuario puede conectarse al servidor web mediante la dirección IP del robot seguida de "*:8080*" en un navegador web (el puerto HTTP es el 80) cuando el robot está en modo punto de acceso a través de su interfaz *wlan1*. La dirección IP del robot es 192.168.10.1 en su interfaz *wlan1*. También se puede conectar a través de las interfaces *usb0* o *wlan0*.

El servidor ofrece diferentes secciones como *Home*, donde se puede identificar al robot y ver su número de versión; *Update*, donde se actualiza el firmware con el que trabaja el robot; o la sección *Connect* (Figura 13) para la conexión del robot; y la sección *Application* (Figura 12), donde se encuentra la configuración principal.

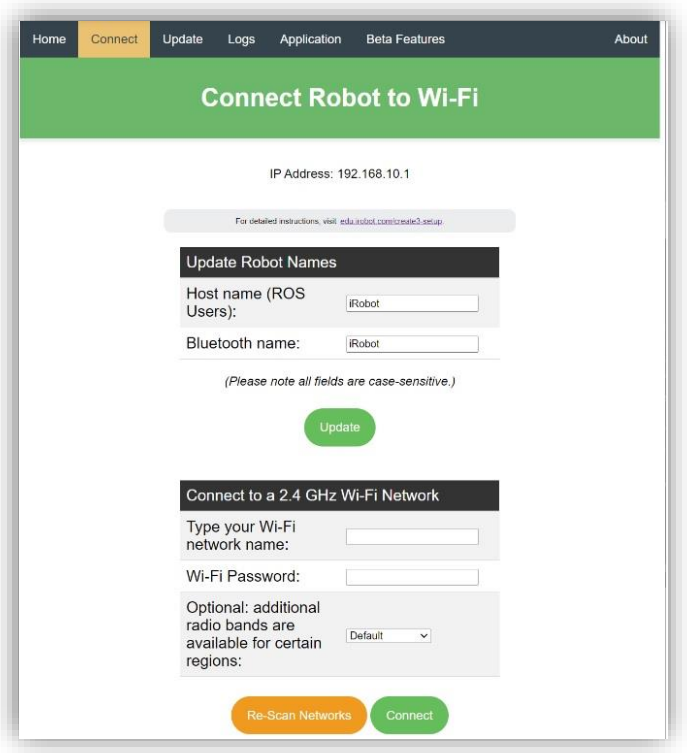

Figura 13: Sección de conexión a Wi-Fi del servidor web

4. TÉCNICAS Y HERRAMIENTAS

En este apartado se describen las técnicas metodológicas y herramientas que se han utilizado para el desarrollo del proyecto en lo que respecta a la parte de la aplicación móvil que actúa como cliente y el servidor de ROS2. También se realiza la descripción del robot y otras herramientas empleadas para la elaboración de la documentación técnica.

4.1. Android Studio

Android Studio [29] es el Entorno de Desarrollo Integrado (IDE) oficial que se usa en el desarrollo de apps para Android y es por ello por lo que se ha decidido utilizarlo.

Figura 14: Android Studio

Otras características importantes que han llevado al uso de este IDE son, entre otras:

- Su sistema de compilación flexible basado en Gradle [30] , ejecutado en una herramienta de depuración integrada.
- Un emulador rápido y cargado de funciones que permite al desarrollador probar la aplicación en un entorno seguro, sin correr riesgos.
- Un entorno unificado de desarrollo para todos los dispositivos Android.
- Edición en vivo para la actualización de los elementos de la interfaz gráfica de usuario de la aplicación que se va a desarrollar.
- Compatibilidad integrada con *Google Cloud Platform* [31] , que facilita la integración con *Google Cloud Messaging*.

El lenguaje de programación que se ha utilizado para la parte lógica de la aplicación es Java, puesto que es el lenguaje nativo de Android, y XML para la interfaz gráfica de usuario. Ambos son bien conocidos, ya que son objeto de estudio en el Grado y son fáciles de manejar.

4.1.1. Android SDK (Software Development Kit)

Se trata de un conjunto de paquetes o herramientas de desarrollo proporcionadas por Google [32] para la plataforma Android que permite crear aplicaciones sin la necesidad de tener una gran experiencia en el campo.

Entre sus paquetes [33] se encuentran los siguientes:

- **Android SDK Build-Tools:** herramientas de compilación del SDK de Android.
- **Android Emulator:** emulador de Android.
- **Android SDK Platform-Tools:** herramientas de la plataforma del SDK de Android.
- **Android SDK Tools:** herramientas del SDK de Android.

4.2. Biblioteca RCLPY

Esta biblioteca proporciona la API canónica de Python para la interacción y uso de ROS2 desde Python de manera nativa. Se basa en la biblioteca RCL (*ROS Client Library*) [34] , que es la implementación en C++ del middleware de comunicación de ROS2 [35] . Esta biblioteca es necesaria para el desarrollo del proyecto, ya que permite a los desarrolladores crear, modificar, y controlar nodos, así como suscribirse o publicar *topics* y proporcionar soporte para el uso de servicios y acciones.

4.3. Biblioteca Lottie

Lottie [36], [37] es una biblioteca multiplataforma para iOS, macOS, tvOS, Android y Web que representa de forma nativa animaciones y arte basados en vectores en tiempo real con un código mínimo. Lottie carga y renderiza animaciones y vectores exportados en formato JSON.

4.4. Herramientas CASE

Las herramientas CASE (*Computer Aided Software Engineering* o Ingeniería de Software Asistida por Computador) son un conjunto de aplicaciones informáticas usadas para automatizar actividades del ciclo de vida de desarrollo de software [38] .

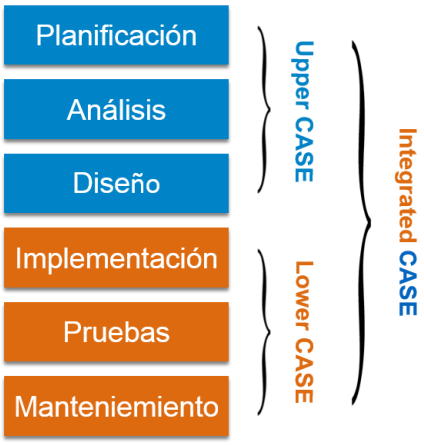

Figura 15: Herramientas CASE

4.4.1. Visual Paradigm

Es una herramienta UML CASE [39] compatible con UML y la notación de modelado de procesos. Además del soporte de modelado, proporciona capacidades de ingeniería de código y generación de informes. Útil para trabajar con SCRUM y la metodología ágil.

Figura 16: Visual Paradigm

4.5. UML

UML o *Unified Modeling Language* [40] es un lenguaje de modelado visual común y semántica y sintácticamente rico para la arquitectura, diseño e implementación de sistemas software complejos. Los diagramas UML describen los límites, la estructura y el comportamiento del sistema y los objetos que contiene.

Figura 17: UML

5. ASPECTOS RELEVANTES DEL DESARROLLO

En este apartado se explican los aspectos más importantes relacionados con el desarrollo del proyecto.

5.1. Ciclo de vida

Este apartado resume el Anexo I – Planificación del proyecto.

Scrum es un marco de gestión de proyectos de metodología ágil y está especialmente indicado para proyectos en los que se necesita obtener resultados rápidos y donde los requisitos pueden estar continuamente cambiando o estar poco definidos [41]. Esta es la razón por la que se ha elegido seguir este proceso, ya que el proyecto a desarrollar se encuentra dentro de este ámbito mencionado.

El desarrollo del proyecto se ha dividido en una serie de sprints con sus respectivos hitos u objetivos, compuestos por historias de usuario. Cada historia de usuario está dividida a su vez en tareas a las que se le ha asignado un esfuerzo estimado.

El objetivo del primer sprint es implementar un reconocimiento de voz funcional. Con el sprint 1 se inicia el desarrollo del proyecto. Comienza con una primera reunión con los tutores y la conceptualización del proyecto. Se realiza una configuración inicial del entorno de desarrollo de la aplicación móvil y la implementación de la captación de voz. Teniendo en cuenta el esfuerzo estimado y el trabajo final realizado se elabora un primer diagrama *Burn-Down Chart* (Figura 18) a través del cual se muestra la evolución del sprint real respecto al ideal. Se observa que conforme avanzan los días la cantidad de trabajo va disminuyendo hasta acabar un día antes de lo esperado, a pesar de que algunos días hay un estancamiento debido a imprevistos.

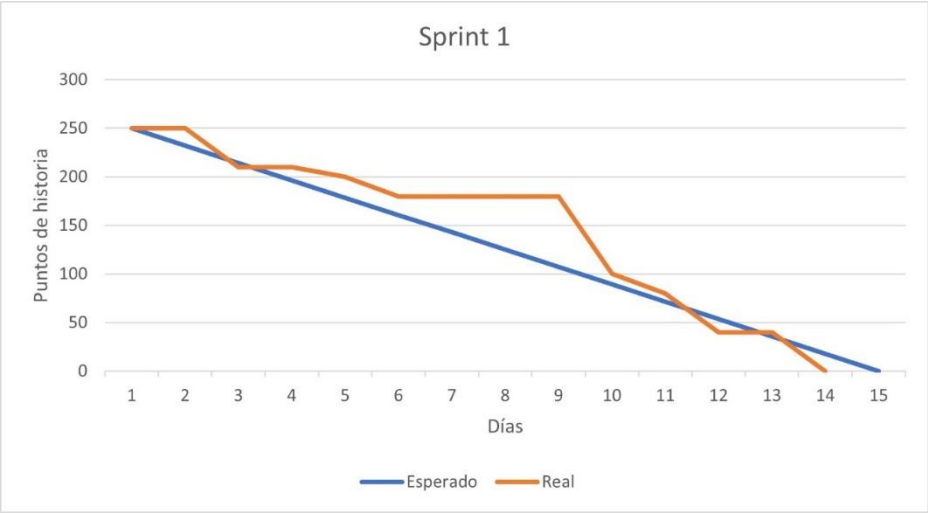

Figura 18: Burn-Down Chart Sprint 1

Una vez finaliza el primer sprint (todas las tareas se han completado), se pasa a la elaboración del siguiente sprint. El objetivo del segundo sprint es establecer la comunicación entre la aplicación y el servidor. Para ello, se procede a definir y establecer el protocolo de comunicación TCP/IP para el envío de las órdenes de voz desde la aplicación utilizando sockets. A su vez, comienza la fase de documentación y configuración de ROS2 en Linux para la creación del servidor de ROS2 con el fin de recibir en este último las órdenes de voz enviadas desde la aplicación. Por último, se realizan pruebas para comprobar el correcto funcionamiento y solucionar posibles errores. En este caso se logó finalizar el sprint en el tiempo establecido como se observa en el siguiente *Burn-Down Chart* (Figura 19).

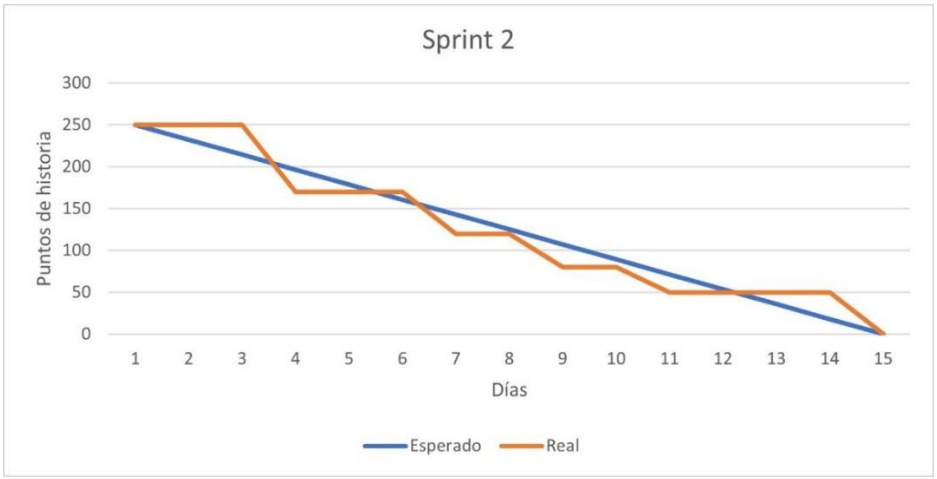

Figura 19: Burn-Down Chart Sprint 2

El objetivo del tercer sprint es controlar el robot y recibir la realimentación en la app. Para ello, se procede a implementar en el servidor de ROS2 la lógica de control del robot. Primero, es necesario documentarse sobre las APIs de ROS2 y el robot tanto a nivel hardware como software. Una vez implementada la lógica, se procede a modificar la interfaz gráfica para añadir la posibilidad de mostrar el estado de la orden. También se configura el servidor de ROS2 para que el robot muestre una realimentación visual al usuario sobre la orden dada. Para el sprint 3, como se observa en el siguiente *Burn-Down Chart* (Figura 20), la estimación de esfuerzo fue menos acertada, por lo que fue necesario emplear un día más para la finalización de todas las tareas. Hubo más problemas e imprevistos de los que se esperaban y muchos días de los trabajados no se pudo avanzar.

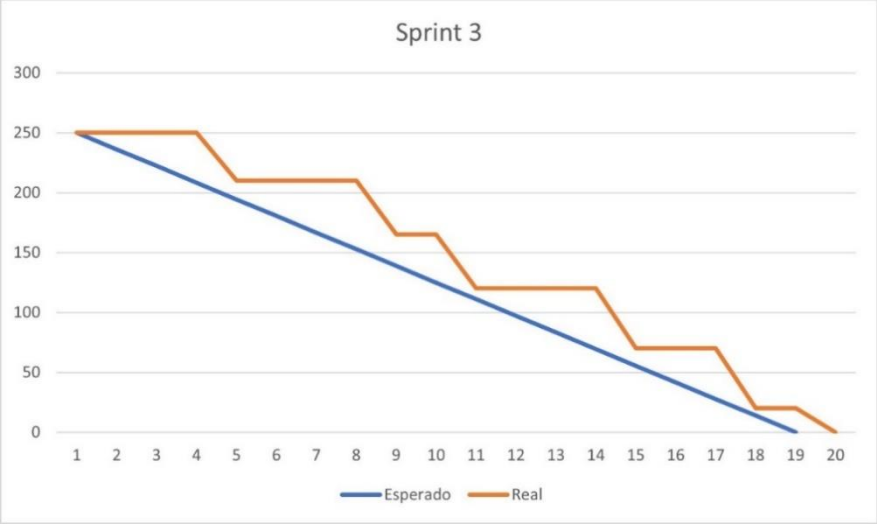

Figura 20: Burn-Down Chart Sprint 3

El objetivo del cuarto y último sprint se centra en la realización de pruebas de funcionamiento del sistema completo y la depuración de posibles errores; seguido de la implementación de ciertas mejoras y ajustes tanto en la interfaz gráfica de usuario como en la lógica de control del robot en caso de ser necesario. En este último sprint se pudieron terminar todas las tareas finalmente a tiempo como se observa en el siguiente *Burn-Down Chart* (Figura 21).

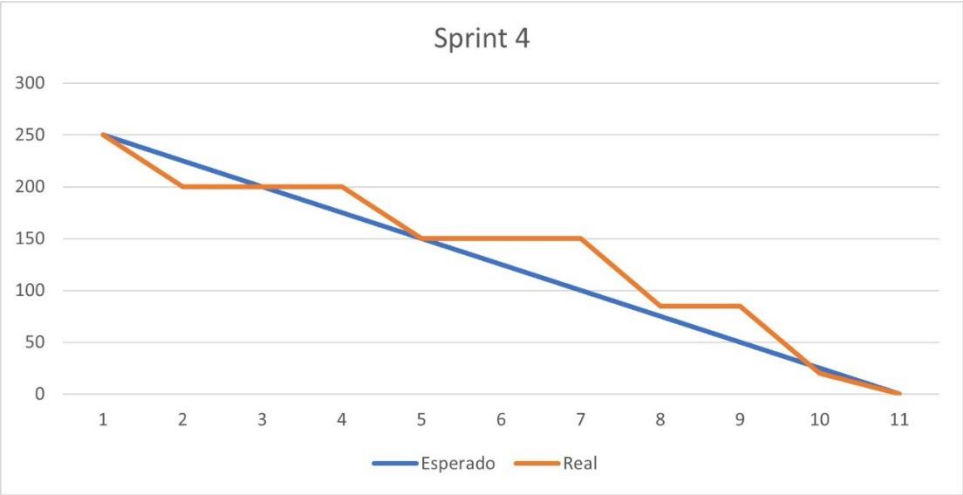

Figura 21: Burn-Down Chart Sprint 4

5.2. Desarrollo del software

Este apartado resume los aspectos más importantes de los anexos II – Especificación de requisitos software y III – Especificación de diseño.

5.2.1. Especificación de requisitos software

En este Anexo II se capturan todos los aspectos relacionados con el software del sistema.

Para la documentación de este anexo se ha empleado el libro de Ingeniería del Software: Un Enfoque Práctico, de Roger Pressman, así como los apuntes de las asignaturas de Ingeniería del Software del Grado. Además, se ha seguido la estructura de Durán y Bernárdez.

Se recogen los participantes del proyecto, los objetivos que debe cumplir el sistema y los requisitos del sistema tanto funcionales como no funcionales. En la Figura 22 se puede observar la captación de los casos de uso del sistema.

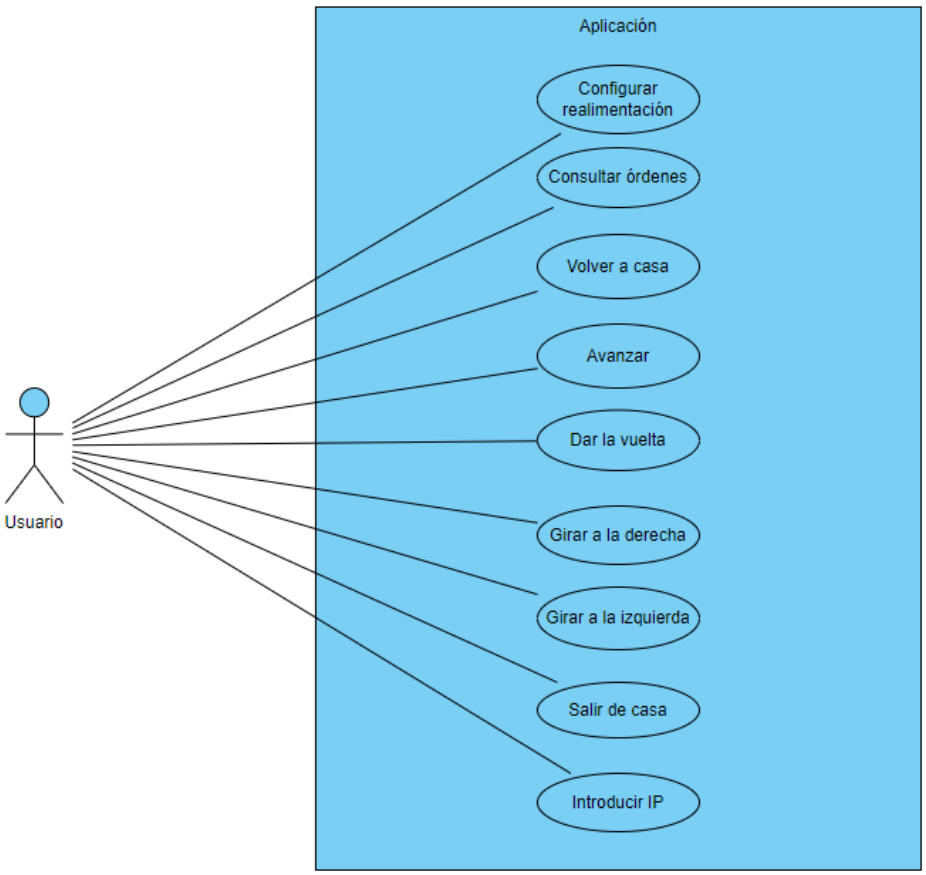

Figura 22: Diagrama de casos de uso – Requisitos del sistema

Tras la obtención de los requisitos se construye un modelo del dominio del problema. Se puede observar en la Figura 23 cómo el sistema en un primer momento va a estar formado por los paquetes *Aplicación* (que se refiere a la app en el dispositivo móvil Android) y *Servidor de ROS2* unidos, ya que se comunican entre ellos. Cada uno de ellos constituye una clase con sus respectivos atributos y métodos necesarios para el funcionamiento que se ha ideado para el sistema. El paquete Aplicación tiene cuatro métodos que son: `escucharOrden`, para recoger el comando de voz por el micro del terminal móvil; `enviarOrden`, para transmitir el comando de voz del usuario al servidor de ROS; `recibirRealimentación` para recibir el estado de la orden dada y mostrar el resultado por la interfaz gráfica; `insertarIP` para que el usuario inserte en la aplicación la IP del ordenador que ejecuta el servidor de ROS2.

El paquete Servidor ROS2 representa el servidor de ROS2 que se ejecuta en un ordenador con Sistema Operativo Linux (distribución Ubuntu 22.04) y tiene tres métodos: `recibirOrden`, para recibir la orden procedente de la aplicación a través del protocolo TCP/IP; `enviarOrden`, para ejecutar la clase NodoROS2 y enviar la orden de movimiento al robot; `enviarRealimentación`, para enviar de vuelta a la aplicación una respuesta sobre el estado de la orden dada.

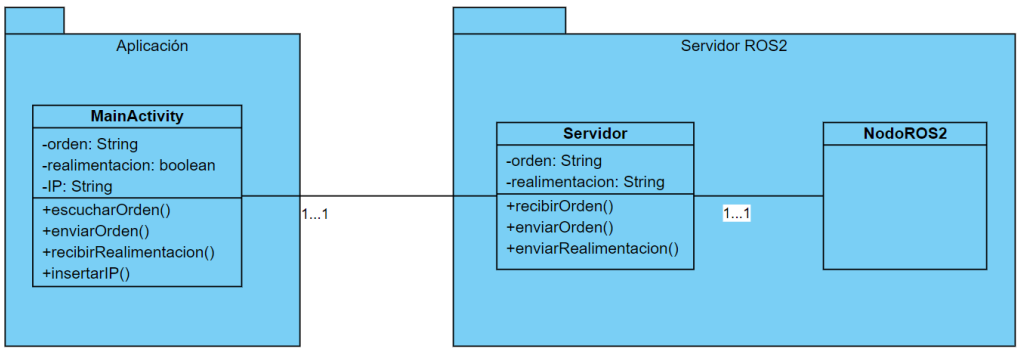

Figura 23: Diagrama de clases

Por último, se especifica una vista de interacción en la que se definen los diagramas de secuencia relacionados con los casos de uso. Se muestra un ejemplo con el diagrama de secuencia más representativo del proyecto (Figura 24), que es el envío de una orden de movimiento.

El usuario envía una orden, en este caso la de avanzar, a través de la aplicación utilizando su interfaz gráfica (InterfazApp). Internamente en la lógica de la aplicación (SistemaApp) se procesa la orden para enviarla posteriormente al servidor de ROS2 que, al recibirla, se muestra en la interfaz del servidor (ServidorROS2) para que seguidamente en la lógica del servidor se procese enviándose hasta el robot (iRobot Create 3) y se ejecute por medio de su Sistema Operativo.

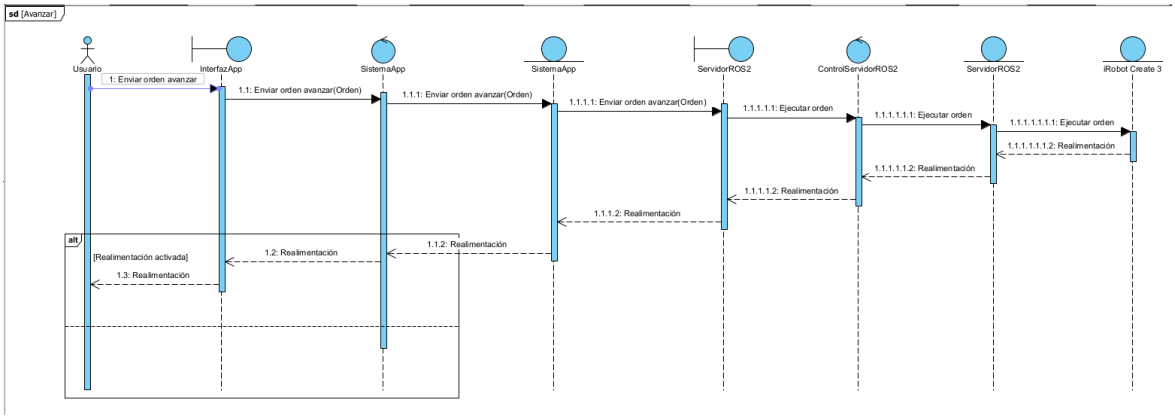

Figura 24: Diagrama de secuencia Avanzar

5.2.2. Especificación de diseño

Este apartado resume los aspectos más relevantes del Anexo III – Especificación de diseño.

En la Figura 25 se muestra un esquema de la arquitectura del sistema completo.

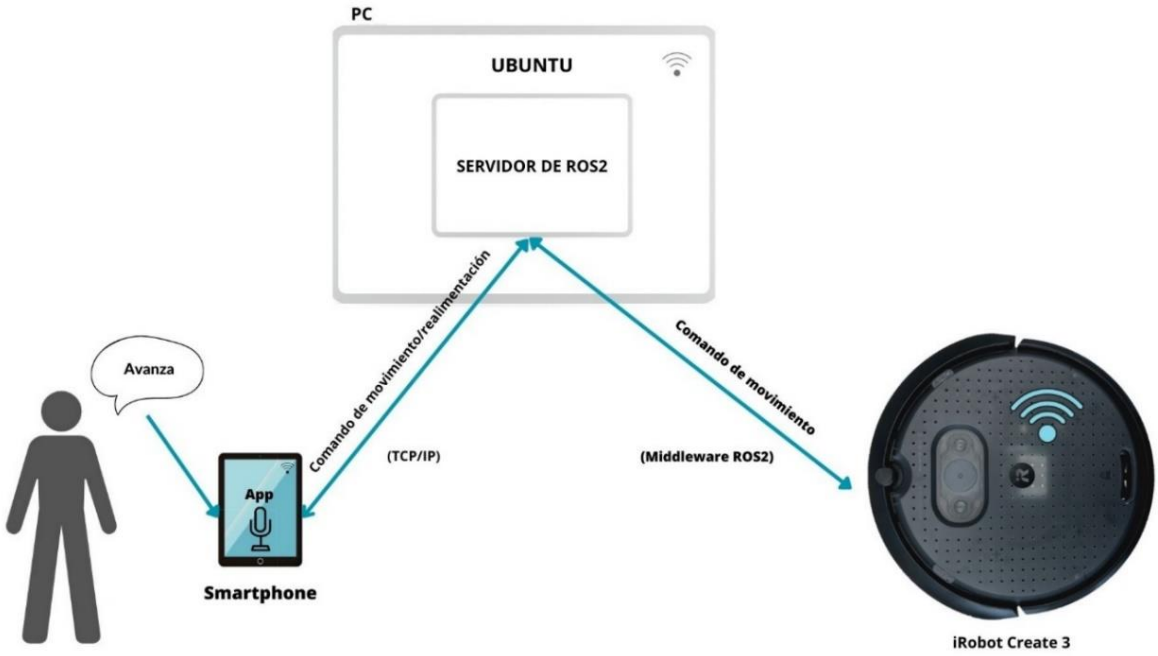

Figura 25: Arquitectura del sistema

El sistema diseñado se caracteriza por tres componentes que se comunican entre ellos a través de la misma red de área local: un smartphone, un ordenador-servidor y el robot Create 3. La aplicación en el smartphone envía, a través del protocolo TCP/IP, la orden de voz captada del usuario al servidor de ROS2 en un ordenador con sistema operativo Linux (distribución Ubuntu 22.04), donde se ejecuta un nodo. Es necesario que la aplicación conozca la IP del ordenador que ejecuta el servidor de ROS2 para poder realizar el envío de la orden. En función de la orden que se reciba en el servidor, el nodo de ROS 2 envía al robot Create 3 el comando de movimiento que se requiere por medio del middleware de ROS2. Finalmente, el robot ejecuta la orden que el usuario ha solicitado.

El sistema completo que concierne al proyecto se basa en una arquitectura Cliente-Servidor, cuya comunicación se establece a través del protocolo TCP/IP.

En la parte del cliente se tiene la aplicación Android en un dispositivo móvil que recoge y envía las órdenes de voz del usuario al servidor de ROS2. Para la arquitectura de la aplicación móvil se emplea el patrón Modelo-Vista-Controlador cuyo propósito es separar los datos, la interfaz de usuario y la lógica de negocio en diferentes capas relacionadas entre ellas.

El servidor, un programa escrito en Python, ejecuta desde su clase principal una clase secundaria que representa un nodo de ROS2. Se encarga de recibir la información de la aplicación en el smartphone, enviar (a través del nodo de ROS2) las órdenes al robot para que este se mueva y devolver a la aplicación información de realimentación. El nodo de ROS2 (la clase secundaria) está escrito también en Python y contiene las instrucciones necesarias para que el robot reciba las órdenes que tiene que hacer y las ejecute.

Es imprescindible que tanto ordenador/servidor como el terminal móvil donde se ejecuta la aplicación y robot se encuentren en la misma red de área local. Esto es debido a cómo es el funcionamiento del protocolo TCP/IP entre la aplicación y el servidor de ROS2, y el funcionamiento del middleware de ROS2 para la comunicación entre el servidor y el robot.

5.2.3. Modelo de diseño

En este apartado se muestra el diagrama de clases final del sistema Figura 26, con sus respectivos atributos y operaciones que se implementan. Estos se explican más adelante en los esquemas generales de la aplicación y del servidor.

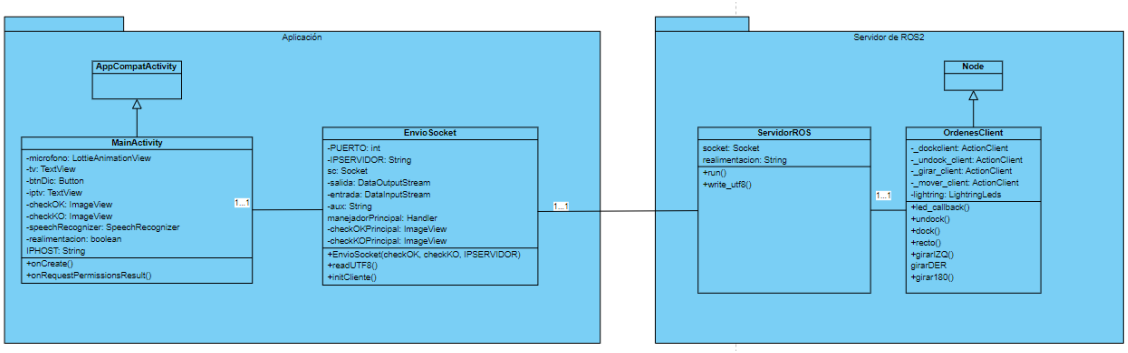

Figura 26: Diagrama de clases final

5.3. Diseño de la interfaz de usuario

El diseño de la interfaz de usuario se explica más en detalle también en el Anexo III.

La interfaz de la aplicación es la vía que tiene el usuario de comunicarse con el robot para que este ejecute las órdenes deseadas. En el diseño ha primado que la interfaz sea simple e intuitiva, de manera que el usuario no tenga problemas a la hora de comprender su funcionamiento. El diseño de la interfaz gráfica de usuario tiene lugar en los ficheros layout de extensión XML que se encuentran dentro de la carpeta de la aplicación.

A lo largo del desarrollo de la aplicación móvil se han realizado pruebas con usuarios, por lo que el diseño se ha ido modificando con el objetivo principal de que la aplicación facilite la interacción del usuario con el robot para su navegación.

En la Figura 27 y la Figura 28, se muestra el diseño de la interfaz gráfica de la aplicación móvil que maneja el usuario. Se aplica un color verde de los botones y el fondo blanco con la intención de ofrecer al usuario una sensación de seguridad y confianza respecto al uso de la aplicación. Cada uno de los botones ofrece una funcionalidad distinta entre las que se encuentra el

diccionario de órdenes, elegir o no la realimentación, insertar la dirección IP del ordenador que ejecuta el servidor e iniciar el reconocimiento de voz.

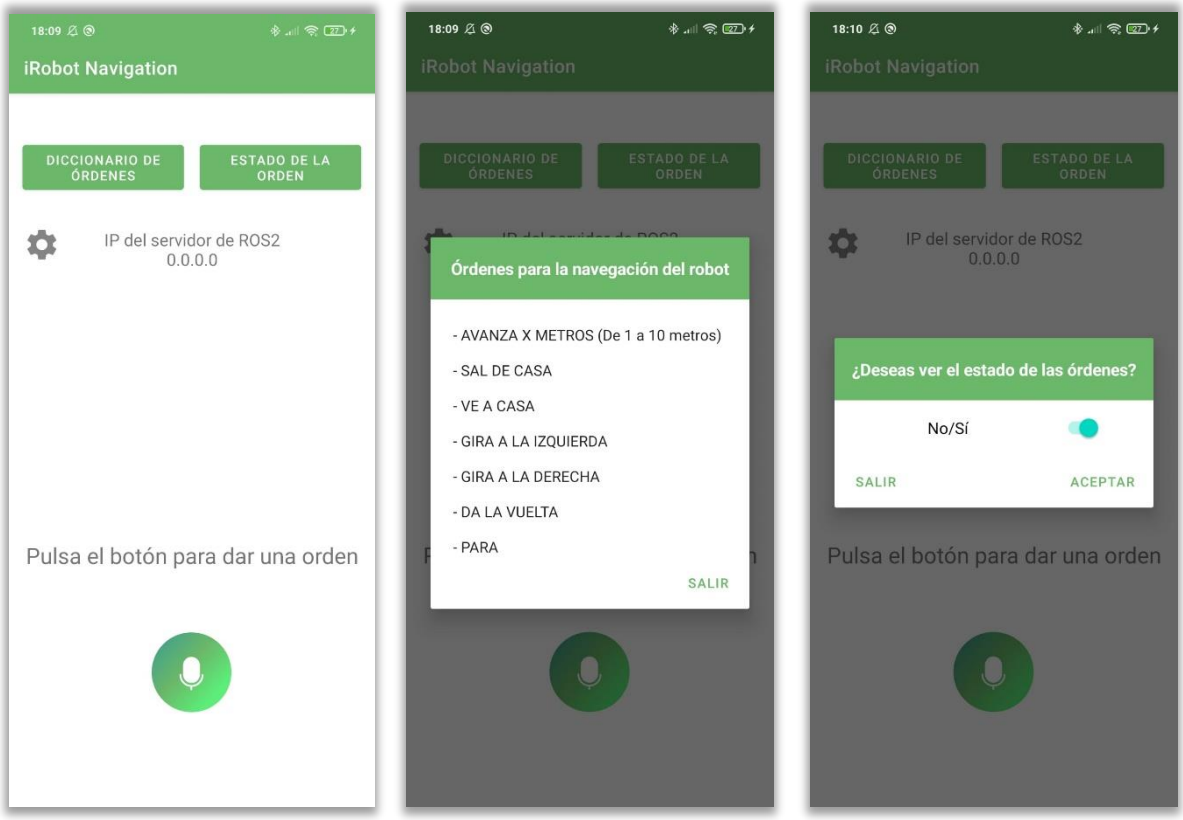

Figura 27: Diseño de la interfaz gráfica

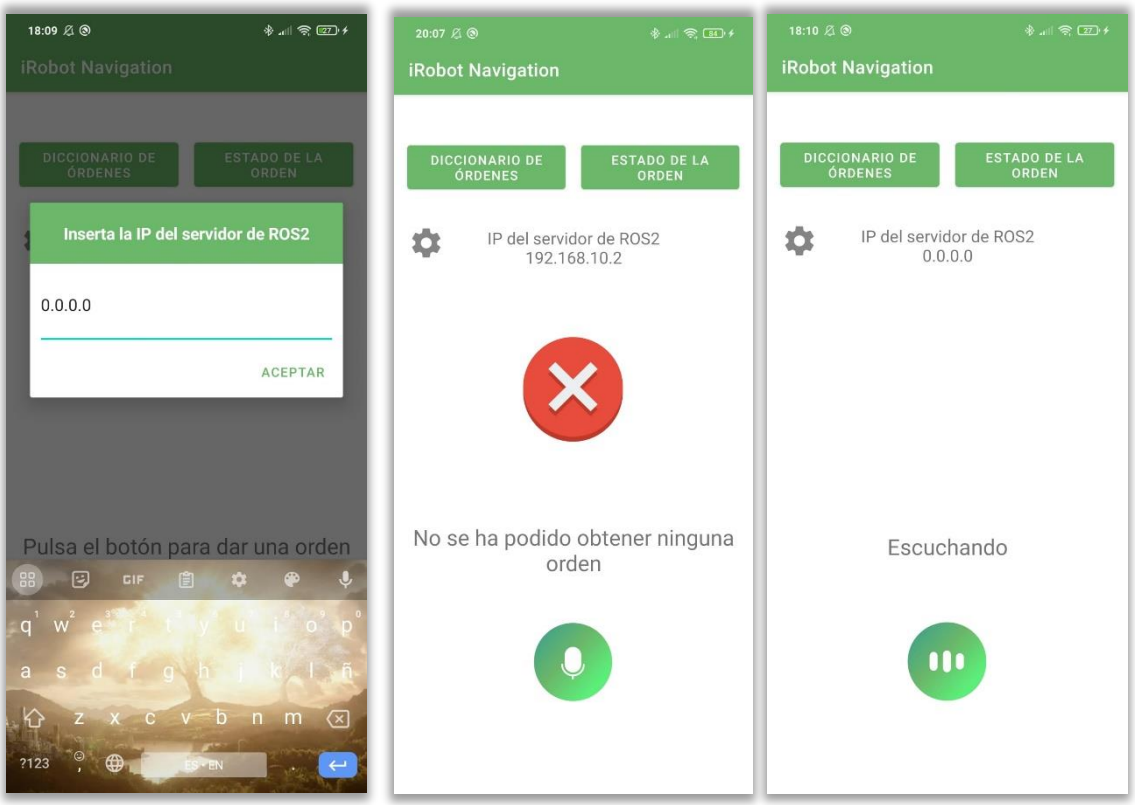

Figura 28: Diseño de la interfaz gráfica 2

5.4. Diseño del servidor de ROS2

Se diseña el servidor de ROS2 como encargado de recibir la orden de la aplicación y, en función de cuál sea, enviarla al robot para que se ejecute el movimiento y enviar el resultado de la orden (la realimentación) de vuelta a la aplicación.

Como modelo de comunicación entre la aplicación y el servidor se ha utilizado la arquitectura Cliente-Servidor, que es un modelo síncrono. Se ha establecido como modelo de comunicación entre el servidor y el robot un modelo asíncrono, ya que ROS2 está basado en la publicación y suscripción a sus entidades.

El servidor ejecuta un nodo de ROS2 como clase secundaria, encargada de la comunicación con el robot a través de la publicación-suscripción a las entidades de ROS2

Respecto a las entidades de ROS2 que se utilizan en el nodo de este servidor para poder ejecutar los movimientos del robot iRobot Create 3 se han empleado las siguientes:

- Topics
- Acciones
- Mensajes
- Servicios
- ActionClient

Se emplean las instancias de *ActionClient*, una para cada acción que va a realizar el robot, especificando el ***topic*** que se quiere utilizar para el movimiento del robot. A través de los ActionClient se envía como **mensaje** la **acción** que se quiere ejecutar a través del *topic* específico. La acción es ejecutada por el servidor de acción correspondiente (**servicios**).

Para otras acciones del robot, como el cambio de colores del anillo LED o la recogida de datos de los sensores infrarrojos se emplea el sistema de publicación-suscripción a los *topics* correspondientes. La publicación se realiza para ejecutar el cambio de color en el robot y la suscripción para que el robot envíe los datos al servidor.

5.5. Aspectos relevantes de la aplicación en smartphone

5.5.1. Esquema general de la aplicación

Cuando se inicia la aplicación en la clase `MainActivity`, que es el hilo principal, se establece el diseño de la interfaz de usuario que se muestra por pantalla al ejecutarse la actividad. Se realiza también la comprobación de permisos de grabación de audio. Posteriormente, se crea el objeto `SpeechRecognizer` para tener acceso al servicio de reconocimiento de voz y se crea un objeto Intent para indicar al sistema que se desea iniciar una actividad de reconocimiento de voz. Se configuran también los parámetros del reconocimiento de voz.

A través del uso de `View.OnClickListener` (objeto de escucha de eventos de entrada), se manejan los eventos relacionados con la activación de los objetos de la interfaz gráfica por interacción del usuario. Por medio de la interfaz `RecognitionListener` se definen los eventos relacionados con el reconocimiento de voz, entre los que destacan el método `endOfSpeech()`, que detiene el reconocimiento de voz cuando se detecta que ha finalizado el habla, y el método `onResults()`, que recoge como resultado la orden de movimiento que dicta el usuario captada por el reconocimiento de voz.

Cuando el usuario pulsa sobre el objeto micrófono de la interfaz, se capta este evento y se ejecuta el método `onClick()` referente al objeto, que inicia el reconocimiento de voz. Cuando se capta el evento de finalización del habla, se ejecuta consecuentemente el método `onResults()` de la interfaz RecognitionListener de manera que la orden se almacena en una variable como cadena de caracteres y se manda al método `initCliente()` de la clase EnvioSocket, que ejecuta el hilo secundario de la aplicación, donde se envía al servidor de ROS2 por medio de un socket utilizando el protocolo TCP/IP.

Cuando el servidor envía de vuelta la realimentación, esta se recibe en el hilo secundario como cadena de caracteres, se modifica su formato para que sea compatible con Java y se llama a un manejador para poder ejecutar operaciones en el hilo principal y mostrar la realimentación en la interfaz gráfica de la aplicación.

5.5.2. Reconocimiento de voz

A continuación, se van a describir los componentes/clases utilizados para establecer el reconocimiento de los comandos de voz en la aplicación.

Clase SpeechRecognizer

Dentro del paquete `android.speech`, Android proporciona la clase `SpeechRecognizer`. Esta clase proporciona acceso al servicio de reconocimiento de voz de Android utilizando los Servicios de Voz de Google [42]. En nuestro proyecto se utiliza para recoger las órdenes de voz del usuario. La implementación de esta API [43] es probable que transmita audio a servidores remotos para realizar el reconocimiento de voz. No está diseñada para usarse para el reconocimiento continuo de voz, ya que con este planteamiento se consumiría una cantidad significativa de batería y ancho de banda. Esta clase debe ser invocada desde el hilo principal de la aplicación, y para poderse usar han de estar activos los permisos de grabación de audio mencionados anteriormente.

La creación de la instancia del objeto `SpeechRecognizer` (Figura 29) se realiza a través del método `createSpeechRecognizer()`. Para ello, primero se comprueba si el reconocimiento está habilitado mediante el método `isRecognitionAvailable()`.

```
//Creación del objeto speech recognizer
if(SpeechRecognizer.isRecognitionAvailable( context: this)){
    speechRecognizer = SpeechRecognizer.createSpeechRecognizer( context: this);
}
else{
    Toast.makeText( context: this,  text: "Reconocimiento no habilitado", Toast.LENGTH_SHORT).show();
}
```

Figura 29: Creación de la instancia de SpeechRecognizer

Interfaz RecognitionListener

`RecognitionListener` es una interfaz que define una serie de métodos que son llamados durante el reconocimiento de voz y que se establece en un objeto `SpeechRecognizer` para manejar los eventos relacionados con el reconocimiento de voz. De manera que el `RecognitionListener` recibe notificaciones del `SpeechRecognizer` cuando los eventos relacionados con el

reconocimiento ocurren [44] . Se establece un `RecognitionListener` en un `SpeechRecognizer` utilizando el método `setRecognitionListener()` y con la creación de este aparecen por defecto una serie de métodos utilizados para manejar los eventos del reconocimiento de voz. El método más importante aquí es `onResults()` (Figura 30), que se ejecuta cuando se obtienen los resultados del reconocimiento de voz. Los resultados se recogen en un `ArrayList` de `Strings` de manera que al aplicarle el método `get()` la orden del usuario se almacena en una variable de tipo `String`.

```
@Override
public void onResults(Bundle resultado) {

    ArrayList<String> data = resultado.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);
    String envio = data.get(0);
    tv.setText(data.get(0));

    cliente.initCliente(envio, realimentacion);

}
```

Figura 30: Método onResults()

5.5.3. Objeto Intent

Un `Intent` [45] es un objeto de mensajería que se utiliza para solicitar una acción/facilidad a otros componentes de la aplicación. De esta manera la aplicación se comunica con ellos.

Cuando se crea el objeto `Intent`, hay que establecer el tipo de acción que va a comenzar Intent. En este caso se utiliza `ACTION_RECOGNIZE_SPEECH` para indicar al sistema Android que se desea iniciar una actividad de reconocimiento de voz. Una vez el `Intent` es creado se pueden configurar los parámetros del reconocimiento de voz (Figura 31) mediante el uso del método `putExtra()` [46] [47], para definir el modelo de lenguaje (el conjunto de reglas y patrones lingüísticos predefinidos para reconocer y entender el habla) y el idioma que se va a reconocer.

```
//Creación del objeto Intent
final Intent speechRecognizerIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);

//Configuración de los parámetros del reconocimiento de voz
speechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
speechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, value: "es-ES");
```

Figura 31: Creación del Intent y configuración del reconocimiento de voz

5.5.4. Objetos de escucha de eventos de entrada – View.OnClickListener

En Android existen diferentes formas para interceptar eventos desde una interacción con el usuario en una aplicación [48]. El enfoque consiste en capturar los eventos desde el objeto de vista específico con el que interactúa el usuario; es decir, si el usuario pulsa sobre algún componente de la interfaz de usuario de la aplicación, este evento se recoge en una acción determinada.

Un objeto de escucha de eventos es una interfaz de la clase `View` que contiene un solo método de devolución de llamada. El framework de Android llamará a estos métodos cuando la vista con la que se haya registrado el objeto de escucha se active por la interacción del usuario con el elemento de la interfaz de usuario. Concretamente en el proyecto se utiliza la interfaz `View.OnClickListener` para gestionar los eventos de interacción del usuario.

El método `setOnClickListener` [49] [50] está proporcionado por la interfaz `View.OnClickListener` que se usa, y aplicado a un componente de la vista registra una llamada de retorno (listener) que se invocará cuando se haga clic en el contexto del componente de la vista de la aplicación como puede ser un `Button`, `ImageView`, `TextView` ... Este método se utiliza, entre otros, en el componente de la interfaz `LottieAnimationView` micrófono (Figura 32) para que inicie la escucha una vez es pulsado.

```
microfono.setOnClickListener(view -> {
        microfono.playAnimation();
        checkOK.setVisibility(View.INVISIBLE);
        checkKO.setVisibility(View.INVISIBLE);
        //Se empieza a escuchar, proporcionamos un Intent al recognizer
        speechRecognizer.startListening(speechRecognizerIntent);
});
```

Figura 32: Objeto de escucha

5.5.5. Comunicación con el servidor de ROS2

Protocolo utilizado

Para la comunicación de la aplicación con el servidor de ROS2 se ha decidido emplear el protocolo TCP/IP, ya que proporciona unas garantías de entrega y fiabilidad. En un primer momento se llegó a considerar el uso del protocolo UDP, que se utiliza a menudo en aplicaciones de tiempo real, como streaming. Sin embargo, pese a que UDP pueda ser más rápido que TCP/IP debido a la falta de conexión, no garantiza la entrega de los datos.

Hilo secundario para el envío de órdenes

En Android está prohibida la ejecución de operaciones de red en el hilo principal de las aplicaciones por razones de seguridad. Cuando se inicia una aplicación, el sistema crea un hilo de ejecución denominado 'principal'. Este hilo es el más importante, porque está a cargo de distribuir eventos a los widgets correspondientes de la interfaz de usuario, incluidos los eventos de dibujo.

Cuando la aplicación realiza un trabajo intensivo, como en este caso son las operaciones de envío y recepción de datos a través de la red, el hilo principal de la aplicación puede generar un rendimiento deficiente, llegando a bloquear toda la interfaz de usuario. Cuando se bloquea este hilo principal no se pueden distribuir los eventos, dando al usuario la sensación de que la aplicación no responde [51].

Por esta razón, el envío de las órdenes al servidor de ROS2 desde la aplicación se realiza en un hilo secundario, que se ejecuta en la clase EnvioSocket mediante el uso de un objeto `Executor` [52] . En la Figura 33 se muestra la creación de este objeto. Mediante el uso de su método `execute()` se ejecuta la tarea definida del hilo secundario.

```
1 usage
public void initCliente(String datos, Boolean realimentacion) {

    Executor executor = Executors.newSingleThreadExecutor();
    executor.execute(() -> {
```

Figura 33: Objeto Executor

La tarea del hilo secundario se basa en la creación de un objeto `Socket` [53] al que se le pasa como argumentos el puerto y la dirección IP donde se mandan las órdenes que el usuario dicta por voz.

Además, en este hilo secundario también se recibe un mensaje del servidor de ROS2, en función de si todo ha ido bien o ha ocurrido algún problema.

Se emplean dos objetos de tipo `DataInputStream` [54] y `DataOutputStream` [55] para leer y escribir respectivamente en la aplicación tipos de datos primitivos de Java (Figura 34).

```
sc = new Socket(HOST, PUERTO);

entrada = new DataInputStream(sc.getInputStream());
salida = new DataOutputStream(sc.getOutputStream());

salida.writeUTF(datos);     //Enviamos mensaje
aux = readUTF8(entrada);    //Recibimos mensaje
```

Figura 34: Socket

Hilo secundario para visualizar la realimentación

Este hilo secundario, además de enviar/recibir los datos desde/hacia el servidor de ROS2, se encarga de mostrar en la interfaz gráfica de la aplicación la información de realimentación que se recibe del robot a través del servidor de ROS2. El problema reside en que sólo el hilo principal puede modificar componentes de la vista de la aplicación y, por lo tanto, al estar en el hilo secundario, es necesario emplear algún mecanismo para lograr este objetivo. Para ello se ha utilizado un manejador de tipo `Handler` [56] que se pasa al constructor de la clase `EnvioSocket` desde la clase principal.

A través del manejador Handler [57] se da soporte a la gestión del trabajo entre dos hilos. De esta manera, se transfiere trabajo al hilo principal desde el hilo secundario. Se utiliza el método `post()` del `Handler` para enviar una tarea concreta al hilo principal, definida en un objeto de tipo `Runnable`. El hilo principal de la actividad de la aplicación ejecuta esta tarea, de forma que se consigue mostrar al usuario en la interfaz gráfica la información de realimentación que ofrece el robot.

El objeto `Runnable` ejecuta el método `run()`, donde se define la tarea a realizar por el hilo principal. En la Figura 35 se muestra un ejemplo del código en el caso de que el usuario haya activado la realimentación. Se pone como visible uno de los dos `ImageView` de la vista en función de si la orden se ejecutó correctamente o si hubo algún problema.

```
if(realimentacion){
    if (aux.contains("OK")) {
        manejadorPrincipal.post(new Runnable() {
            @Override
            public void run() {
                checkKOPrincipal.setVisibility(View.INVISIBLE);
                checkOKPrincipal.setVisibility(View.VISIBLE);
            }
        });
    } else {
```

Figura 35: Uso de Handler para el hilo principal

Problemas de equivalencia de formato

Los datos se envían y reciben como cadena de caracteres [58] y para ello es necesario tratarlos, pero existe un problema.

Estos métodos de envío y recepción de datos en Java no tienen una equivalencia en Python, que es el lenguaje de programación en el que está escrito el servidor de ROS2, y por lo tanto esto da lugar a una entrega y recepción de los datos con un formato diferente. Para solventarlo se ha seguido una solución propuesta en un foro de Internet y en donde se desarrolla con mayor profundidad el problema [59].

La solución consiste en abandonar los métodos propios que ofrece Java , que son `readUTF()` y `writeUTF()` y reemplazarlos por una versión propia que es la que se ha seguido, con el fin de que los datos en ambos lados tengan el formato correcto para poder manejarlos como se desea.

A continuación, se muestra en la Figura 36 el método `readUTF8()` que sustituye al original, y que se encarga de recibir y tratar el dato del servidor de manera que tenga el formato adecuado para poder utilizarse.

```
1 usage
public String readUTF8(DataInput in) throws IOException {
    int length = in.readInt();
    byte [] encoded = new byte[length];
    in.readFully(encoded);
    return new String(encoded, StandardCharsets.UTF_8);
}
```

Figura 36: Método readUTF8()

En el caso de este proyecto no ha sido necesario utilizar una versión propia para el método `writeUTF()`, ya que se podía acceder correctamente a los datos en el lado del servidor. Sin embargo, sí ha sido necesario también establecer nuevos métodos de envío por la parte del servidor, que se explicarán en el apartado que corresponde.

5.5.6. Datos de la aplicación

Como datos, la aplicación móvil almacena la orden que recoge el sistema de reconocimiento de voz y la realimentación recibida desde el servidor de ROS2. Todos en forma de cadena de caracteres en variables de tipo `String`.

La aplicación también almacena como dato la opción que elige el usuario para ver o no por la pantalla la realimentación recibida desde el servidor y la IP del ordenador que ejecuta el servidor de ROS2. Estas dos variables se almacenan a través del uso de un objeto `SharedPreferences`, que establece un archivo de configuración, donde se almacena un elemento clave-valor con el valor de estas variables. Se utiliza para que la aplicación tenga almacenados estos datos en todo momento, aunque se cierre, ya que de otra forma se perderían y el usuario tendría que volverlos a introducir de nuevo, algo que puede resultar molesto para el usuario. En la Figura 37, se muestra la definición del objeto `SharedPreferences`.

```
SharedPreferences sp = getApplicationContext().getSharedPreferences( s: "ArchivoConfig", Context.MODE_PRIVATE);
realimentacion = sp.getBoolean( s: "ArchivoConfig",  b: false);
```

Figura 37: Objeto SharedPreferences

5.5.7. Dependencias de compilación

El sistema de compilación de Gradle de Android Studio [60] facilita al desarrollador incluir archivos binarios externos u otras librerías en la compilación que funcionan como dependencias. Estas dependencias pueden estar ubicadas en el equipo o en un repositorio remoto. Cualquier dependencia transitiva que se declare también se incluirá.

En la sección de dependencias `dependencies` del archivo build.gradle a nivel de módulo, además de las que vienen por defecto con la creación de un proyecto en Android Studio, se ha incluido la biblioteca Lottie [61]. Esta biblioteca, de código libre desarrollada por la empresa Airbnb, permite la reproducción de animaciones utilizando archivos JSON en dispositivos. En este proyecto, esta biblioteca permite utilizar un archivo JSON como animación de un micrófono, que se reproduce en la interfaz cuando el usuario pulsa el micro para dar una orden. La siguiente Figura 38 muestra la sección de dependencias en la aplicación Android.

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.9.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'com.airbnb.android:lottie:6.0.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
}
```

Figura 38: Sección de dependencias en aplicación Android

5.5.8. Declaración de permisos

El archivo `AndroidManifest.xml` de la carpeta manifest del proyecto software describe la información esencial de la aplicación para las herramientas de creación de Android, el sistema operativo Android y Google Play. En este archivo de manifiesto [62] se declaran, entre otros, los permisos que necesita la aplicación para acceder a las partes protegidas del sistema o a otras aplicaciones.

En nuestro proyecto, se han añadido los permisos de acceso a Internet y la grabación de audio para la captación de voz y transcripción de voz a texto por medio de los Servicios de Voz de Google que Android utiliza. En la Figura 39 se muestra la parte del código de este archivo en la que se declaran estos permisos.

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.INTERNET" />
```

Figura 39: Declaración de permisos en Android

5.6. Aspectos relevantes del servidor de ROS2

5.6.1. Esquema general del servidor de ROS2

Cuando se inicia el servidor, este se mantiene en escucha continua esperando a recibir, a través de todas las interfaces de la red de área local a la que está conectado el ordenador que lo ejecuta y el puerto 8080, los datos que se envían desde la aplicación móvil por el usuario.

Cuando el servidor recibe una orden en la clase principal `EnvioSocket`, procede a comprobar si está dentro de las órdenes que puede ejecutar el robot. Si no es así, se llama a la clase `OrdenesClient`, que es el nodo de ROS2, y se realiza una publicación al *topic* `/cmd_lightring` para poner en rojo el anillo LED del robot y se envía a la aplicación la realimentación de que no se ha podido ejecutar la orden. En cambio, si la orden está dentro de las preestablecidas, se procede a llamar a la función correspondiente al movimiento que se pretende de la clase `EnvioSocket`. En esta función se realiza la petición al servidor de acción correspondiente utilizando la instancia del cliente de acción, se espera por el servidor de acción y si se encuentra, se procede a cambiar el color del anillo LED del robot a verde momentáneamente y a enviar la acción al servidor de acción del robot para que la ejecute.

Posteriormente, cuando la función del nodo del servidor retorna, se procede a enviar, ya desde la clase principal `EnvioSocket`, la realimentación de que la orden se ha ejecutado correctamente.

En otra terminal diferente a la que se ejecuta el servidor, que es un paquete de ROS2, se ejecuta otro paquete llamado `Infrarrojos` cuya función es la de mostrar los datos de los sensores infrarrojos del robot por pantalla para determinar cuándo se detectan obstáculos. Se ha implementado en otro paquete independiente para poder visualizar los datos de los infrarrojos en todo momento.

5.6.2. Comunicación con la aplicación

La comunicación con la aplicación se realiza a través de la creación de un socket al que se le asocian las interfaces de red y el puerto 8080 para después habilitarlo con el fin de aceptar conexiones entrantes y recibir la información. Se muestra en la siguiente figura (Figura 40).

```
def __init__(self):

    self.sc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.sc.bind(("", PUERTO))
    self.sc.listen()

    print("Servidor iniciado. Esperando órdenes ...")
```

Figura 40: Creación y habilitación del socket

Como ya se ha mencionado anteriormente, al no existir una equivalencia entre Python y Java para el envío y recepción de datos a través de la red, se crea la función `write_utf8()` para solucionar este problema y poder devolver una realimentación a la aplicación. Esta función se muestra en la siguiente figura (Figura 41).

```
def write_utf8(self, s: str, sock: socket.socket):
    encoded = s.encode(encoding='utf-8')
    sock.sendall(struct.pack('>i', len(encoded)))
    sock.sendall(encoded)
```

Figura 41: Función write_utf8()

5.6.3. Comunicación con el robot y ejecución de comandos

El uso de ROS2 en este proyecto se basa en la publicación y suscripción a los *topics* que ofrece el robot iRobot Create 3 así como la ejecución de los clientes de acción para que los servidores de acción del robot lleven a cabo los movimientos deseados. Entre las diferentes acciones que el robot proporciona, algunas son: `/audio_note_sequence`, para hacer sonar frecuencias de sonido determinadas en el robot; `/dock drive_distance`, para que el robot avance en línea recta una determinada distancia especificada en metros; `/rotate_angle`, para que el robot gire los grados que se especifiquen; `/undock`, para que el robot salga de la estación de carga; `/wall_follow`, para que el robot avance siguiendo una pared.

En este trabajo se han utilizado publicadores y suscriptores sobre un *topic* en el nodo de ROS2 creado (la clase `OrdenesClient` ejecutada por la clase principal `EnvioSocket`), para cambiar el color de las luces del anillo LED del robot y para recibir información de los sensores infrarrojos respectivamente. Cuando se realiza una publicación a un *topic* del robot, el robot la recibe al

suscribirse a ese *topic* y procede a ejecutar el mensaje; como ocurre cuando se quiere cambiar el color del anillo LED. En este caso, para cambiar el color del anillo LED se realiza una publicación al *topic* `/cmd_lightring` del robot, el robot la recibe al suscribirse a ese *topic* y procede a ejecutar la orden de cambio de color. En la Figura 42, se puede observar la creación del publicador en el servidor de ROS2 que hemos desarrollado para modificar el color en los LEDs del robot.

```
self.lights_publisher = self.create_publisher(LightringLeds, '/cmd_lightring', 10)
self.lightring = LightringLeds()    #Definición de los mensajes del anillo LED
self.lightring.override_system = True #Esto nos permite cambiar el color de las luces
```

Figura 42: Publicador para el anillo LED del robot

Cuando se realiza una suscripción a un *topic* del robot, el robot la recibe y procede a publicar la información sobre el *topic*; como ocurre cuando se quiere recibir la información sobre los sensores infrarrojos. En este caso, se realiza una suscripción al *topic* `/ir_intensity` del robot, el robot la recibe y procede a publicar la publicación sobre ese *topic*.

De la misma forma, a través del uso de los clientes de acción, se envían metas (acciones que se quiere que el robot ejecute) que son recibidas por los servidores de acción del robot. Una vez recibidas, el sistema operativo del robot procede a ejecutarlas para conseguir el movimiento deseado del robot. En la Figura 43 se puede observar la creación de los clientes de acción en el servidor para los comandos implementados de “Salir de casa”, “Ir a casa”, “Girar un cierto ángulo” y “Avanzar en línea recta una determinada distancia”.

```
def __init__(self):
    super().__init__('ordenes_client')
    self._dock_client = ActionClient(self, Dock, 'dock')
    self._undock_client = ActionClient(self, Undock, 'undock')
    self._girar_client = ActionClient(self, RotateAngle, '/rotate_angle')
    self._mover_client = ActionClient(self, DriveDistance, '/drive_distance')
```

Figura 43: Creación de los clientes de acción

La comunicación entre el servidor de ROS2 y el robot es posible gracias al middleware de ROS2 descrito en apartados anteriores. Ya que el robot está

basado en ROS2 y el servidor también, es posible mandarle instrucciones para su navegación, recepción de datos, etc.

En la Figura 44 se muestra el ejemplo de petición del servidor al robot para volver a la estación de carga a través del envío de una meta utilizando un cliente de acción.

```
#Ve a casa
def dock(self):
    goal_msg = Dock.Goal()

    if not self._dock_client.wait_for_server(timeout_sec=5.0):
        raise Exception("No se ha encontrado el servidor de acción")

    self.led_callback(r=0, g=255, b=0)
    instFuture = self._dock_client.send_goal_async(goal_msg)
    self.led_callback(r=255, g=255, b=255)

    global FLAG_CASA
    FLAG_CASA = True

    return instFuture
```

Figura 44: Petición para ir a la estación de carga

En este caso, se utiliza el cliente de acción `Dock`. Los clientes de acción proporcionan el método `Goal()` que devuelve como mensaje la acción correspondiente deseada, que en este caso es la de volver a la estación de carga. La instancia del cliente de acción `_dock_client` se crea en la función `__init__()` del nodo de ROS2 (la clase `OrdenesClient`). Primero, se utiliza para esperar por el servidor de acción que le corresponde, de manera que si no se encuentra (no hay comunicación con el robot), se produce una excepción. Si el servidor de acción se encuentra, entonces se hace una llamada a la función `led_callback()`del nodo, donde se realiza una publicación al *topic* `/cmd_lightring` para poner en verde el anillo LED del robot. Posteriormente, se llama a la función `send_goal_async()` pasándole como parámetro el mensaje recogido anteriormente con el método `Goal()` (es decir, la acción deseada). De esta manera se realiza el envío de la acción al robot por medio del middleware de ROS2 para que se ejecute internamente en el software del robot.

5.6.4. Paquete del servidor de ROS2

El servidor constituye un paquete de ROS2, que es la unidad de organización del código.

Cuando se crea el paquete, se crean por defecto una serie de archivos que son imprescindibles para su funcionamiento. Entre los que se encuentran:

- package.xml
- resource/servidorROS2
- setup.cfg
- setup.py
- servidorROS2

Se definen dos clases. La clase `ServidorROS` es la clase principal que establece la conexión con la aplicación y se encarga de recibir los datos, tratarlos y, dependiendo de la orden de movimiento que se haya recibido, llamar a la función que corresponda de la clase secundaria `OrdenesClient` para ejecutar el movimiento del robot. Estas clases son las que se pueden observar en el diagrama de clases final del modelo de diseño de la Figura 26.

5.6.5. Paquete Infrarrojos

El paquete de ROS2 `Infrarrojos` se ejecuta simultáneamente en otra terminal del ordenador junto con el servidor, para mostrar los datos de los sensores infrarrojos del robot de forma independiente y determinar si este ha encontrado algún obstáculo durante el camino. Este paquete ejecuta un código empleando ROS2 para realizar la suscripción al *topic* `/ir_Intensity` de manera que el robot publique los datos de los sensores infrarrojos y detectar así la presencia cercana de obstáculos. Los datos se representan a través de la terminal del ordenador.

5.6.6. Dependencias

El archivo `package.xml` contiene metadatos sobre el paquete que constituye el servidor de ROS2 y en él se definen las dependencias con otros paquetes/bibliotecas necesarios.

5.6.7. Bibliotecas

Las siguientes bibliotecas que se han empleado para el desarrollo del servidor son imprescindibles para el uso de ROS2 y la comunicación con la aplicación y el robot iRobot Create 3.

- **socket**: proporciona funciones y clases imprescindibles para la comunicación mediante el uso de sockets entre el servidor y la aplicación.
- **rclpy**: es la biblioteca de ROS2 para la programación en Python. Imprescindible para la creación de nodos y la publicación y suscripción a *topics* y llamadas a servicios del robot. Establece el contexto de ROS2 para poder realizar operaciones.
- **irobot_create_msgs**: permite crear, enviar y recibir mensajes y acciones específicas del robot.

5.7. Comunicación global del sistema

Es indispensable que tanto dispositivo móvil que ejecuta la aplicación Android, como servidor de ROS2 y robot se encuentren todos en la misma red ya que de otra forma no es posible establecer la comunicación debido a las características de comunicación explicadas anteriormente (TCP/IP y middleware de ROS2).

Para que el robot esté conectado a la misma red que el resto, es necesario acceder primero a su servidor web, conectando el ordenador al punto de acceso Wi-Fi que proporciona. Una vez en el servidor web del robot, hay que acceder a la sección Connect y seleccionar la red.

Para la comprobación de que servidor y robot se reconocen basta con hacer un '`ros2 topic list`' en la terminal del ordenador donde se ejecuta el servidor.

5.8. Configuración del robot iRobot Create 3

Para que el robot esté conectado a la misma red que el resto, es necesario acceder primero a su servidor web conectando el ordenador al punto de acceso Wi-Fi que proporciona. Una vez en el servidor web del robot, hay que acceder a la sección *Connect* y seleccionar la red.

Para la comprobación de que servidor y robot se reconocen basta con hacer un '`ros2 topic list`' en la terminal del ordenador donde se ejecuta el servidor.

6. CONCLUSIONES

Las conclusiones a las que se han llegado tras la realización y cumplimiento de los objetivos de este Trabajo de Fin de Grado son las siguientes:

- Se ha desarrollado una aplicación para un smartphone mediante el uso de Android Studio con una interfaz HMI por voz y bajo el marco de trabajo ROS2 para controlar un robot.
- Se ha conseguido establecer una comunicación entre la aplicación móvil de Android y el robot iRobot Create por medio del servidor que ejecuta un nodo de ROS2.
- Se ha conseguido que el robot ejecute una serie de movimientos básicos a través de la aplicación: salir de la estación de carga y volver a ella, girar a la derecha e izquierda un determinado ángulo, avanzar en línea recta una distancia entre uno y diez metros. En el servidor se ven reflejados los mensajes de depuración cuando se realiza el envío y recepción de las órdenes.
- Se ha conseguido recibir en la aplicación una realimentación para saber si la orden se ha ejecutado correctamente o no.
- Se han puesto en práctica conocimientos y conceptos estudiados en el Grado, como la arquitectura Cliente-Servidor. Además, han sido indispensables los conocimientos sobre redes necesarios para establecer las comunicaciones entre la aplicación móvil y el servidor de ROS2. También se ha puesto en práctica la metodología ágil utilizando Scrum, así como las técnicas necesarias de Ingeniería del Software para el desarrollo del proyecto.
- El uso de ROS en general para el control de robots es una herramienta muy potente hoy en día pues supone un importante avance para establecer una manera de trabajar común en el campo de la robótica. ROS sigue avanzando y sufriendo modificaciones constantes. Se ha evolucionado de las primeras versiones a ROS2.
- Las principales dificultades que han surgido durante el desarrollo del proyecto han estado ligadas a la documentación de ROS2 y concretamente al uso de este marco de trabajo para el robot iRobot

Create 3. Esto se ha debido a que la información necesaria está muy dispersa y no muy bien documentada ni detallada. Entender el funcionamiento y la manera de trabajar con ROS2 para este robot en concreto, ha supuesto un enorme esfuerzo y ha llevado más tiempo de lo esperado en un primer momento.

- También se ha investigado y trabajado con una biblioteca de reconocimiento de voz, que abre las puertas a una interesante vía de comunicación hombre-máquina en multitud de aplicaciones.
- En un primer momento se planteó que el usuario interaccionara con la aplicación a través de un dispositivo Tablet fijado en el robot únicamente con mensajes de voz, sin utilizar la pantalla táctil. Este planteamiento se descartó porque era necesario utilizar una tecnología de reconocimiento continuo de voz, con un consumo de energía que suponía que la batería de la Tablet se agotara en poco tiempo y a la dificultad de crear un reconocimiento continuo de voz como puede ser los ya conocidos Siri u OkGoogle. Finalmente, se optó por un reconocimiento de voz a petición del usuario, cuando pulsa el icono del micrófono que se encuentra representado en la aplicación del smartphone. De esta forma se consigue ahorrar energía eléctrica.

En el siguiente enlace se puede ver un vídeo demostración del funcionamiento de este proyecto.

Hay que constatar que en el momento de la grabación aún no se había implementado la interfaz gráfica de la aplicación en su totalidad y por lo tanto falta el botón para cambiar la IP del ordenador que ejecuta el servidor de ROS2. Esto es un cambio menor y no varía en la consecución de los objetivos del proyecto conforme a la apariencia final de la aplicación.

https://drive.google.com/file/d/1eq0R31WJp-rpmNweefGrNuqtKqgxmeKX/view

También se ofrece un enlace a otro vídeo demostración de las aplicaciones que puede tener el proyecto. En este vídeo se muestra cómo al robot se le pueden fijar cargas para el transporte de objetos y los lleve donde uno quiere.

https://drive.google.com/file/d/1S6TYPcokkdxjQmLGYJGuw8TjEG0qXkPr/view

7. LÍNEAS DE TRABAJO FUTURAS

Este proyecto ha supuesto iniciar un nuevo campo de trabajo con ROS2, por lo que existirán multitud de líneas de trabajo futuras, entre las que destacamos:

- Añadir la posibilidad de enviar varios comandos de movimiento en una sola orden y que se ejecuten uno por uno en el orden establecido hasta completar una determinada tarea. En la versión actual, los comandos de movimiento se envían individualmente.
- Mejorar la información que se realimenta en la aplicación del smartphone. Por ejemplo, si el robot se queda parado en algún lugar, que envíe un mensaje para que se muestre en la aplicación y el usuario tome las decisiones oportunas.
- Respecto a ROS2 sería muy útil añadir instrucciones más complejas, como evitar un obstáculo, traspasar una puerta, etc.
- Añadir la creación de mapas del entorno donde se encuentra el robot para poder ejecutar instrucciones más complejas como ir a un lugar concreto.
- Enviar información sobre los datos de los sensores del robot a la aplicación del smartphone para conocer su estado.

8. REFERENCIAS

[1] «Los Robots y los Sistemas de Control por Voz protagonizan los hogares inteligentes en CES 2017 • CASADOMO». https://www.casadomo.com/2017/01/13/robots-sistemas-control-voz-protagonizan-hogares-inteligentes-ces-2017 (accedido 11 de mayo de 2023).

[2] «Create® 3 Docs». https://iroboteducation.github.io/create3_docs/ (accedido 11 de mayo de 2023).

[3] «¿Qué son los comandos de voz? | NFON ES». https://www.nfon.com/es/get-started/cloud-telephony/lexicon/base-de-conocimiento-destacar/comandos-de-voz (accedido 30 de mayo de 2023).

[4]? «¿Es bueno instalar un altavoz inteligente en casa? - Blog Prosegur». https://blog.prosegur.es/altavoz-inteligente/ (accedido 30 de mayo de 2023).

[5] «Smartwatch con asistente de voz: los mejores modelos». https://topesdegama.com/listas/wearables/mejores-smartwatch-asistente-voz (accedido 30 de mayo de 2023).

[6] «iRobot OS | iRobot®». https://www.irobot.es/es_ES/irobot-os.html (accedido 26 de junio de 2023).

[7] «iRobot® Home App | iRobot®». https://www.irobot.es/es_ES/irobot-home-app.html (accedido 17 de mayo de 2023).

[8] C. Queiruga, C. B. Tzancoff, y F. López, «RemoteBot: una Aplicación que Combina Robots y Dispositivos Móviles».

[9] «Hans (Robot controlado por voz) - Proyectos con Arduino.» https://blogs.etsii.urjc.es/dseytr/hans-robot-controlado-por-voz/ (accedido 30 de mayo de 2023).

[10] «App nativa: ventajas e inconvenientes I Glosario de App Marketing». https://actualizatec.com/blog/app-nativa/ (accedido 18 de mayo de 2023).

[11] «Voice Tech: la tecnología por voz». https://immune.institute/blog/voice-tech-la-tecnologia-por-voz/ (accedido 30 de mayo de 2023).

[12] R. Suárez *et al.*, «Robot Operating System (ROS)», Accedido: 26 de junio de 2023. [En línea]. Disponible en: http://wiki.ros.org/catkin

[13] «Middleware Config - Create® 3 Docs». https://iroboteducation.github.io/create3_docs/setup/xml-config/#fast-dds (accedido 31 de mayo de 2023).

[14] «ROS on DDS». https://design.ros2.org/articles/ros_on_dds.html (accedido 31 de mayo de 2023).

[15] «Multi-Robot Setup - Create® 3 Docs». https://iroboteducation.github.io/create3_docs/setup/multi-robot/#ros-2-domain-ids (accedido 27 de junio de 2023).

[16] «ROS 2 Documentation — ROS 2 Documentation: Humble documentation». https://docs.ros.org/en/humble/index.html (accedido 20 de mayo de 2023).

[17] «Humble on Ubuntu 22 - Create® 3 Docs». https://iroboteducation.github.io/create3_docs/setup/ubuntu2204/#before-you-start (accedido 20 de mayo de 2023).

[18] «Understanding nodes — ROS 2 Documentation: Humble documentation». https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html (accedido 20 de mayo de 2023).

[19] «4 Nodos, Topics y Mensajes. Turtlesim – ROS TUTORIAL. Tutorial ROS en español». http://rostutorial.com/4-nodos-topics-y-mensajes-turtlesim/ (accedido 20 de mayo de 2023).

[20] «es/ROS/Tutoriales/ComprendiendoServiciosParametros - ROS Wiki». http://wiki.ros.org/es/ROS/Tutoriales/ComprendiendoServiciosParametros (accedido 21 de mayo de 2023).

[21] «Understanding services — ROS 2 Documentation: Humble documentation». https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-

Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html (accedido 21 de mayo de 2023).

[22] «Understanding actions — ROS 2 Documentation: Humble documentation». https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html (accedido 21 de mayo de 2023).

[23] «A Buildable, Mobile Robotics Platform Create ® 3 Educational Robot iRobot's Foundation for Mobile Robot Development».

[24] «Overview - Create® 3 Docs». https://iroboteducation.github.io/create3_docs/hw/overview/ (accedido 22 de mayo de 2023).

[25] «Adapter Board - Create® 3 Docs». https://iroboteducation.github.io/create3_docs/hw/adapter/ (accedido 22 de mayo de 2023).

[26] «Humble on Ubuntu 22 - Create® 3 Docs». https://iroboteducation.github.io/create3_docs/setup/ubuntu2204/ (accedido 31 de mayo de 2023).

[27] «ROS 2 Interface - Create® 3 Docs». https://iroboteducation.github.io/create3_docs/api/ros2/#ros-2-coordinate-system (accedido 26 de junio de 2023).

[28] «Overview - Create® 3 Docs». https://iroboteducation.github.io/create3_docs/webserver/overview/ (accedido 22 de mayo de 2023).

[29] «Introducción a Android Studio | Android Studio | Android Developers». https://developer.android.com/studio/intro?hl=es-419 (accedido 19 de mayo de 2023).

[30] «Configure your build | Android Studio | Android Developers». https://developer.android.com/build (accedido 19 de mayo de 2023).

[31] «Servicios de cloud computing | Google Cloud». https://cloud.google.com/?hl=es (accedido 19 de mayo de 2023).

[32] «Qué es Android SDK y cómo empezar a usarlo». https://code.tutsplus.com/es/tutorials/the-android-sdk-tutorial--cms-34623 (accedido 19 de mayo de 2023).

[33] «Command-line tools | Android Studio | Android Developers». https://developer.android.com/tools (accedido 19 de mayo de 2023).

[34] «About ROS 2 client libraries — ROS 2 Documentation: Humble documentation». https://docs.ros.org/en/humble/Concepts/About-ROS-2-Client-Libraries.html (accedido 22 de mayo de 2023).

[35] «About ROS 2 middleware implementations — ROS 2 Documentation: Humble documentation». https://docs.ros.org/en/humble/Concepts/About-Middleware-Implementations.html (accedido 22 de mayo de 2023).

[36] «GitHub - airbnb/lottie-ios: An iOS library to natively render After Effects vector animations». https://github.com/airbnb/lottie-ios (accedido 4 de julio de 2023).

[37] «LottieFiles: Download Free lightweight animations for website & apps.» https://lottiefiles.com/ (accedido 4 de julio de 2023).

[38] «Software - CASE Herramientas». https://www.tutorialspoint.com/es/software_engineering/case_tools_overview.htm (accedido 23 de mayo de 2023).

[39] «Ideal Modeling & Diagramming Tool for Agile Team Collaboration». https://www.visual-paradigm.com/ (accedido 31 de mayo de 2023).

[40] «¿Qué es el lenguaje unificado de modelado (UML)? | Lucidchart». https://www.lucidchart.com/pages/es/que-es-el-lenguaje-unificado-de-modelado-uml (accedido 23 de mayo de 2023).

[41] «Qué es SCRUM – Proyectos Ágiles». https://proyectosagiles.org/que-es-scrum/ (accedido 5 de junio de 2023).

[42] «Servicios de voz de Google - Aplicaciones en Google Play». https://play.google.com/store/apps/details?id=com.google.android.tts&hl=es&gl=US (accedido 12 de junio de 2023).

[43] «SpeechRecognizer | Android Developers». https://developer.android.com/reference/android/speech/SpeechRecognizer (accedido 4 de junio de 2023).

[44] «RecognitionListener | Android Developers». https://developer.android.com/reference/android/speech/RecognitionListener (accedido 13 de junio de 2023).

[45] «Intents y filtros de intents | Desarrolladores de Android | Android Developers». https://developer.android.com/guide/components/intents-filters?hl=es-419 (accedido 12 de junio de 2023).

[46] «How to set the language in speech recognition on android? - Stack Overflow». https://stackoverflow.com/questions/10538791/how-to-set-the-language-in-speech-recognition-on-android (accedido 13 de junio de 2023).

[47] «Example usage for android.speech RecognizerIntent EXTRA_CALLING_PACKAGE». http://www.java2s.com/example/java-api/android/speech/recognizerintent/extra_calling_package-0.html (accedido 13 de junio de 2023).

[48] «Descripción general de eventos de entrada | Desarrolladores de Android | Android Developers». https://developer.android.com/guide/topics/ui/ui-events?hl=es-419 (accedido 13 de junio de 2023).

[49] «View | Android Developers». https://developer.android.com/reference/android/view/View#setOnClickListener(android.view.View.OnClickListener) (accedido 13 de junio de 2023).

[50] «Método setOnClickListener en Android». https://keepcoding.io/blog/metodo-setonclicklistener-en-android/ (accedido 13 de junio de 2023).

[51] «Descripción general de los procesos y subprocesos | Desarrolladores de Android | Android Developers». https://developer.android.com/guide/components/processes-and-threads?hl=es-419 (accedido 15 de junio de 2023).

[52] «Executor | Android Developers». https://developer.android.com/reference/java/util/concurrent/Executor (accedido 16 de junio de 2023).

[53] «Socket | Android Developers». https://developer.android.com/reference/java/net/Socket (accedido 16 de junio de 2023).

[54] «DataInputStream | Android Developers». https://developer.android.com/reference/java/io/DataInputStream (accedido 16 de junio de 2023).

[55] «DataOutputStream | Android Developers». https://developer.android.com/reference/java/io/DataOutputStream (accedido 16 de junio de 2023).

[56] «Handler | Android Developers». https://developer.android.com/reference/android/os/Handler (accedido 16 de junio de 2023).

[57] «Handlers – Aplicaciones Móviles Multimedia». https://umhandroid.momrach.es/handlers/ (accedido 16 de junio de 2023).

[58] «android - How to get String from DataInputStream in Java? - Stack Overflow». https://stackoverflow.com/questions/27827803/how-to-get-string-from-datainputstream-in-java (accedido 16 de junio de 2023).

[59] «Python – Create a python server to send data to Android app using socket – iTecNote». https://itecnote.com/tecnote/python-create-a-python-server-to-send-data-to-android-app-using-socket/ (accedido 16 de junio de 2023).

[60] «Add build dependencies | Android Studio | Android Developers». https://developer.android.com/build/dependencies (accedido 8 de junio de 2023).

[61] «Featured animations from our community». https://lottiefiles.com/featured?utm_medium=web&utm_source=navigation-featured (accedido 9 de junio de 2023).

[62] «Descripción general del manifiesto de una app | Desarrolladores de Android | Android Developers». https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419 (accedido 9 de junio de 2023).