Anexo III: Especificación de diseño

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

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN 3
2. DISEÑO DE DATOS 3
3. DISEÑO ARQUITECTÓNICO 4
3.1. Arquitectura Cliente-Servidor 4
3.2. Patrón Modelo-Vista-Controlador (MVC) 6
4. MODELO DE DISEÑO 8
5. DISEÑO E IMPLEMENTACIÓN DE LA INTERFAZ DE USUARIO 9
6. DISEÑO DEL SERVIDOR DE ROS2 19
7. REFERENCIAS 21

Índice de figuras

Figura 1: Esquema de la arquitectura Cliente-Servidor 4
Figura 2: MVC 6
Figura 3: Diagrama de clases 8
Figura 4: Logo de la empresa iRobot 9
Figura 5: Diseño de la interfaz gráfica 10
Figura 6: Orden ejecutada con éxito 11
Figura 7: Ejemplo de ImageView de la aplicación 12
Figura 8: Declaración del componente TextView 12
Figura 9: Componente LottieAnimationView 13
Figura 10: Configuración de la IP del servidor 14
Figura 11: Diccionario de órdenes 15
Figura 12: Estado de la orden 16
Figura 13: Componente Button 16
Figura 14: Componente SwitchCompat 17
Figura 15: Asociación de los objetos a la vista 18
Figura 16: Dependencias de package.xml 19
Figura 17: setup.py 19

1. INTRODUCCIÓN

Una vez finalizada la especificación de requisitos se procede a llevar a cabo la especificación de diseño del sistema donde se explica el diseño de datos, el modelo utilizado MVC, la arquitectura del sistema, el diseño de la interfaz de la aplicación y el diseño del servidor de ROS2.

2. DISEÑO DE DATOS

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 en forma de cadena de caracteres en variables de tipo `String`.

La aplicación también almacena como dato booleano la opción que elige el usuario para ver o no por la pantalla la realimentación recibida desde el servidor, además de la IP del ordenador que ejecuta el servidor de ROS2. Ambas como cadena de caracteres.

Estas dos variables se almacenan en un archivo de configuración como elemento clave-valor con el valor de estas variables. La aplicación tiene 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.

En la parte del servidor de ROS2 también se almacena como dato en forma de cadena de caracteres en variable de tipo `String` la orden recibida desde la aplicación para su posterior uso y, además, se emplea una variable de tipo `String` para enviar la realimentación a la aplicación, si se ha ejecutado o no la orden.

3. DISEÑO ARQUITECTÓNICO

A continuación, se explica la arquitectura general del sistema completo, así como el patrón arquitectónico utilizado en la aplicación Android.

3.1. Arquitectura Cliente-Servidor

El sistema completo que constituye el proyecto se basa en una arquitectura Cliente-Servidor cuya comunicación se establece a través del protocolo TCP/IP y el middleware de ROS2. 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.

A continuación, se muestra un esquema (Figura 1) de la arquitectura del sistema completo del proyecto que sirve como modelo de despliegue.

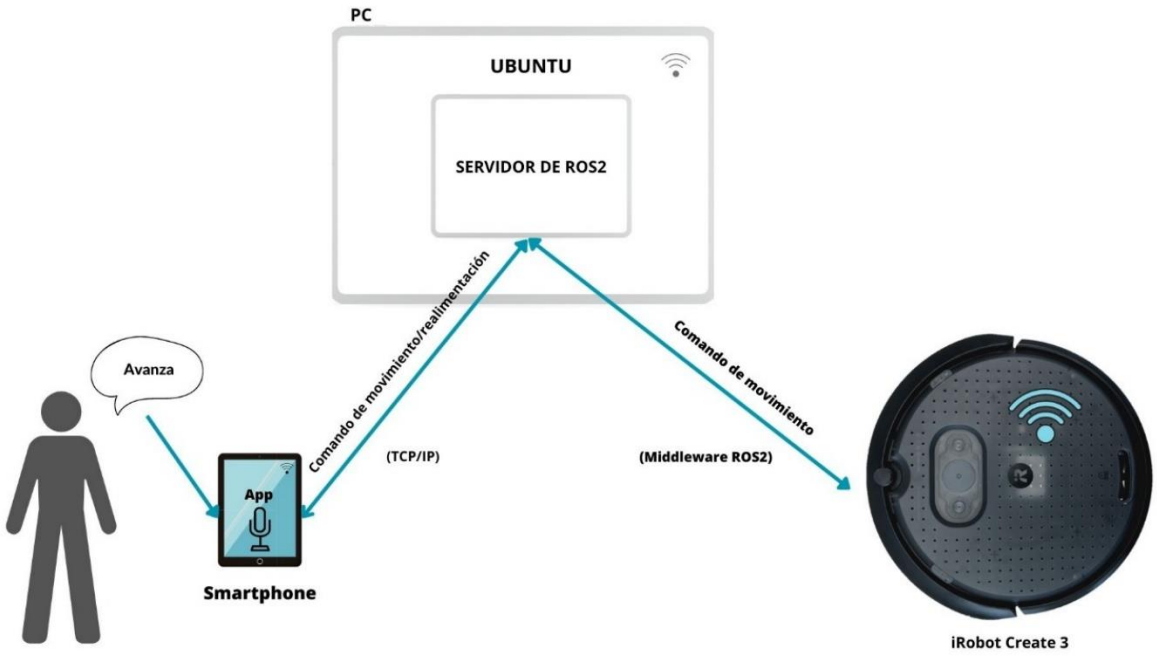

Figura 1: Esquema de la arquitectura Cliente-Servidor

En la parte del cliente se tiene la aplicación Android en un dispositivo móvil (*smartphone*) que se encarga de recoger los comandos de movimiento del robot que dicta el usuario. 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 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.

Cuando se ejecuta el nodo de ROS2 en el servidor, este se comunica con el robot gracias al middleware de ROS2. Este middleware utiliza DDS (*Data Distribution Service*) para el descubrimiento de nodos a través de una lista de nombres únicos para los *topics*, clientes de acción y servicios que ofrece el robot, de manera que el robot recibe los comandos de movimiento procedentes del nodo y los ejecuta.

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.

3.2. Patrón Modelo-Vista-Controlador (MVC)

En Android se utiliza el patrón de arquitectura Modelo Vista Controlador [2] para la construcción de aplicaciones. Se encarga de tratar la solución para separar la interfaz de usuario de la lógica de negocio y los datos.

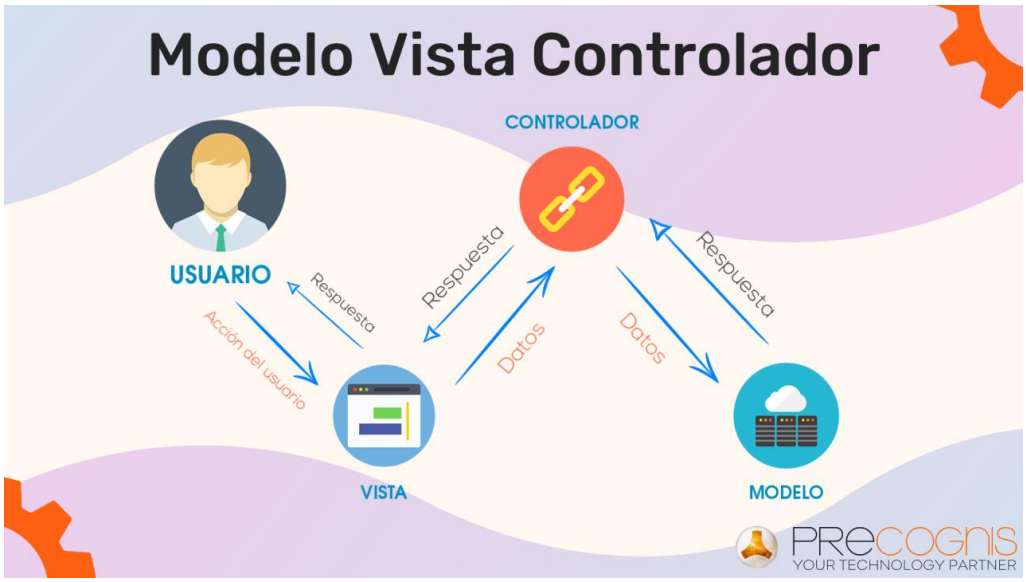

Figura 2: MVC

- **Modelo**: es la capa encargada del tratamiento de los datos (la lógica de la aplicación), responsable de manejar la conexión con la base de datos o APIs.
- **Vista**: es la capa de la interfaz de usuario, constituye la representación del modelo. En Android son los layouts y constituye la parte del lenguaje XML. Se comunica con el controlador.
- **Controlador**: es la capa que se corresponde con la funcionalidad de la aplicación como puede ser el código de hacer clic y realizar alguna acción.

La ventaja del uso del MVC reside en la facilidad de la portabilidad de la aplicación y la facilidad de mantenimiento al estar separados funcionalidad, interfaz y datos. Como ventaja adicional, las vistas están sincronizadas y los cambios de estas junto con los controles se realizan en tiempo de ejecución.

Como inconveniente hay que destacar el incremento de la complejidad o la íntima conexión entre la vista y el controlador unido al posible alto acoplamiento de las vistas y los controladores con respecto al modelo.

En este proyecto la Vista de la aplicación está diseñada en XML y está asociada a la actividad principal (el Controlador) de la clase MainActivity para gestionarla. La asociación se realiza a través del método setContentView() de la actividad principal.

La parte del Modelo de la aplicación se encarga de almacenar como datos las órdenes del usuario, la IP del ordenador que ejecuta el servidor de ROS2, así como la realimentación recibida y la configuración de esta última (si está activada o no).

4. MODELO DE DISEÑO

En este apartado se muestra el diagrama de clases del sistema (Figura 3) con sus respectivos atributos y operaciones que se implementan. Cada clase y sus respectivas operaciones se explican en detalle en el Anexo IV – Documentación técnica de programación.

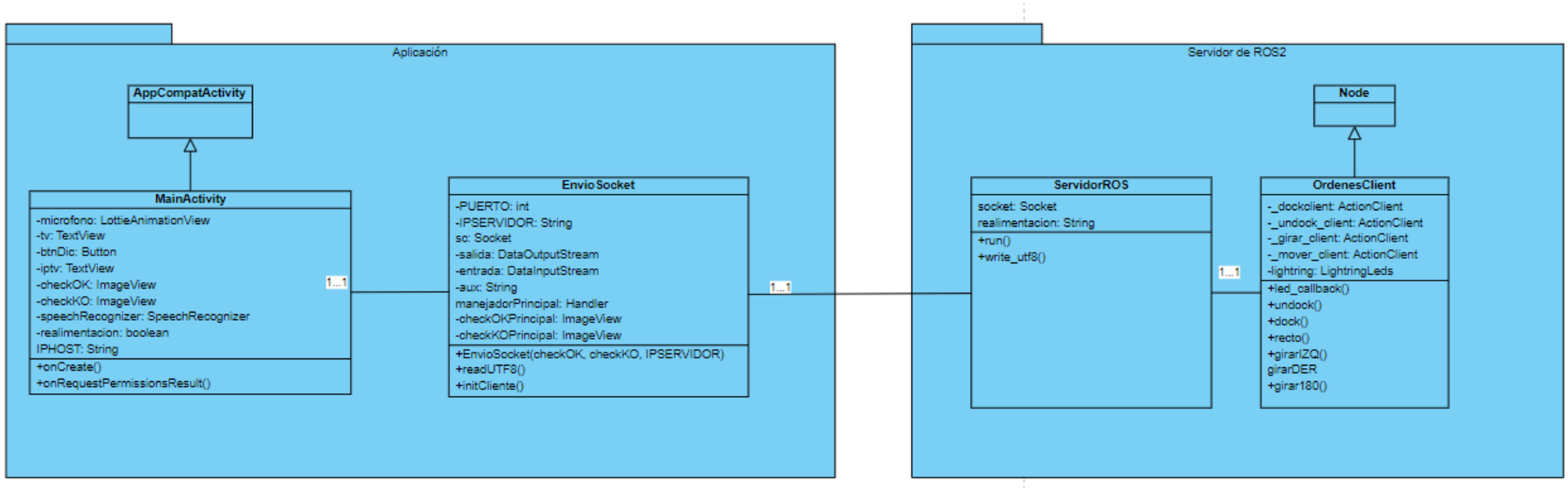

Figura 3: Diagrama de clases

5. DISEÑO E IMPLEMENTACIÓN DE LA INTERFAZ DE USUARIO

La interfaz de la aplicación es una de las partes principales del proyecto ya que es la manera que tiene el usuario de comunicarse con el robot para que este ejecute los movimientos deseados.

Se ha realizado una interfaz sencilla e intuitiva para cualquier persona, de manera que no se tenga problemas a la hora de comprender su funcionamiento.

A lo largo del desarrollo de la aplicación móvil se han realizado pruebas y consultas a los usuarios para conocer las necesidades básicas que debe cumplir la aplicación para poder conseguir el objetivo principal, que es la comunicación con el robot. Entre estas consultas se encuentran:

- Cómo debería el usuario enviar su orden por voz.
- Cómo puede saber el usuario las órdenes configuradas.
- Cuál es la información que necesita ver el usuario y cómo debería estar distribuida junto con los botones en la pantalla de la aplicación.

En función de las necesidades se ha ido desarrollando la interfaz de la aplicación hasta su estado final.

Para la interfaz gráfica de la aplicación se han escogido tonos de color verde claro para los botones y la barra del nombre de la aplicación y títulos haciendo una pequeña referencia al logo de la empresa iRobot (Figura 4), que es la creadora del robot iRobot Create 3 con el que se trabaja en este proyecto.

Figura 4: Logo de la empresa iRobot

Además, el contraste del color verde de los botones y el fondo blanco tiene la intención de ofrecer al usuario una sensación de seguridad y confianza respecto al uso de la aplicación.

El diseño de la interfaz gráfica de usuario (Figura 5) tiene lugar en los ficheros layout de extensión XML que se encuentran dentro de la carpeta del proyecto de la aplicación en Android Studio.

Figura 5: Diseño de la interfaz gráfica

En el caso de este proyecto, en el archivo `activity_main.xml` se definen todos los componentes principales de la interfaz gráfica de la aplicación, así como los elementos de realimentación para que el usuario conozca el estado de la orden que se ha mandado al robot. Cada componente tiene un identificador y sus respectivos parámetros.

Se tienen dos `ImageView` [3] declarados que muestran una imagen para cada estado de la orden (si la orden se ha ejecutado con éxito o no). Se declaran para cada elemento de este tipo una serie de restricciones y parámetros con el fin de que cada ImageView esté correctamente integrado en la aplicación. Para mostrar la imagen se accede a la carpeta `drawable`, que es la carpeta donde se almacenan las imágenes a mostrar en la aplicación.

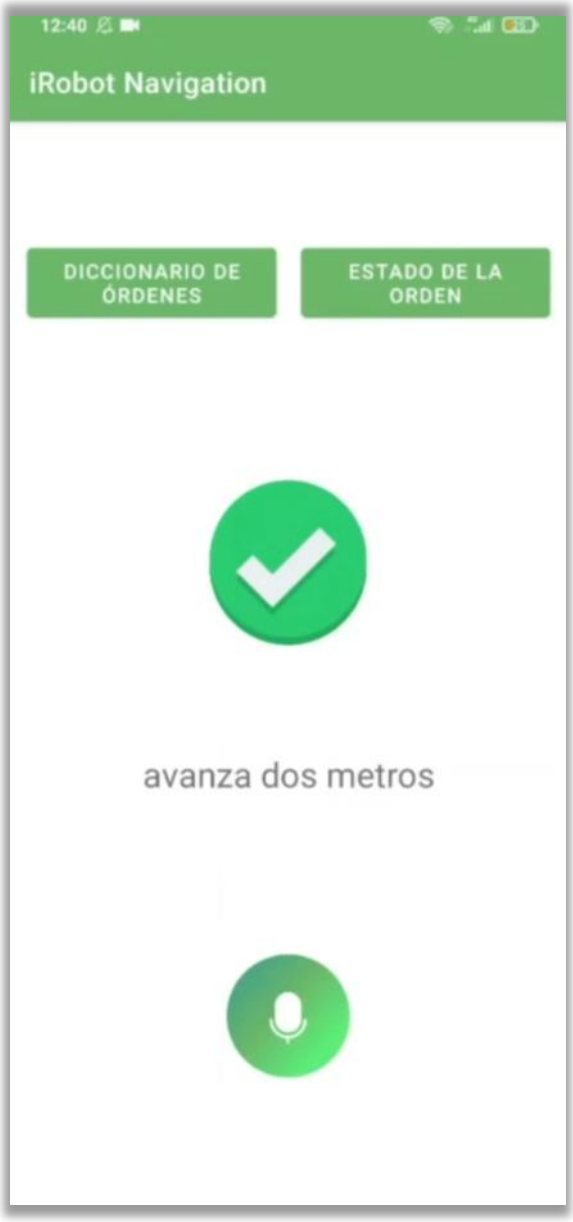

Figura 6: Orden ejecutada con éxito

En Figura 7 podemos observar un ejemplo de la declaración de uno de los `ImageView` presentes en la aplicación.

```
<ImageView
    android:id="@+id/imageCheck"
    android:layout_width="132dp"
    android:layout_height="131dp"
    android:src="@drawable/check"
    android:contentDescription="IRC3"
    android:visibility="invisible"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.375" />
```

Figura 7: Ejemplo de ImageView de la aplicación

Se dispone también de un `TextView` [4] para mostrar por pantalla la orden que dicte el usuario y otro para mostrar la IP del servidor. Mediante la creación de una instancia de este componente en la actividad principal podemos utilizar el campo `setText` para que se muestre el texto deseado.

```
<TextView
    android:id="@+id/textV"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:height="55sp"
    android:gravity="center"
    android:textSize="24sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.599"
    tools:layout_editor_absoluteX="0dp" />
```

Figura 8: Declaración del componente TextView

Se emplea un componente del tipo `LottieAnimationView` [5], que es una vista específica de la biblioteca Lottie en Android (Figura 9). Esta vista proporciona una animación de un micrófono cuando es pulsada por el usuario. Para la animación se utiliza un archivo JSON que se encuentra en la carpeta

`raw` denominado `irobotmic.json`. Este componente se emplea además en la actividad principal para activar el reconocimiento de voz.

```
<com.airbnb.lottie.LottieAnimationView
    android:id="@+id/lottieMic"
    android:layout_width="110dp"
    android:layout_height="99dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintDimensionRatio="1:1"
    app:layout_constraintHorizontal_bias="0.498"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.857"
    app:layout_constraintWidth_percent="0.4"
    app:lottie_autoPlay="false"
    app:lottie_loop="false"
    app:lottie_rawRes="@raw/irobotmic" />
```

Figura 9: Componente LottieAnimationView

Se tiene también un componente de tipo `ImageView` con forma de tuerca para indicar al usuario que, si se pulsa, se accede a la configuración para insertar la IP del ordenador que ejecuta el servidor de ROS2. Se puede ver en la siguiente figura la ventana emergente al pulsar este botón. Al pulsar aceptar, la aplicación almacenará como dato String la dirección IP introducida por el usuario y mostrará un pequeño mensaje avisando de que se ha guardado la IP. Se muestra en la Figura 10.

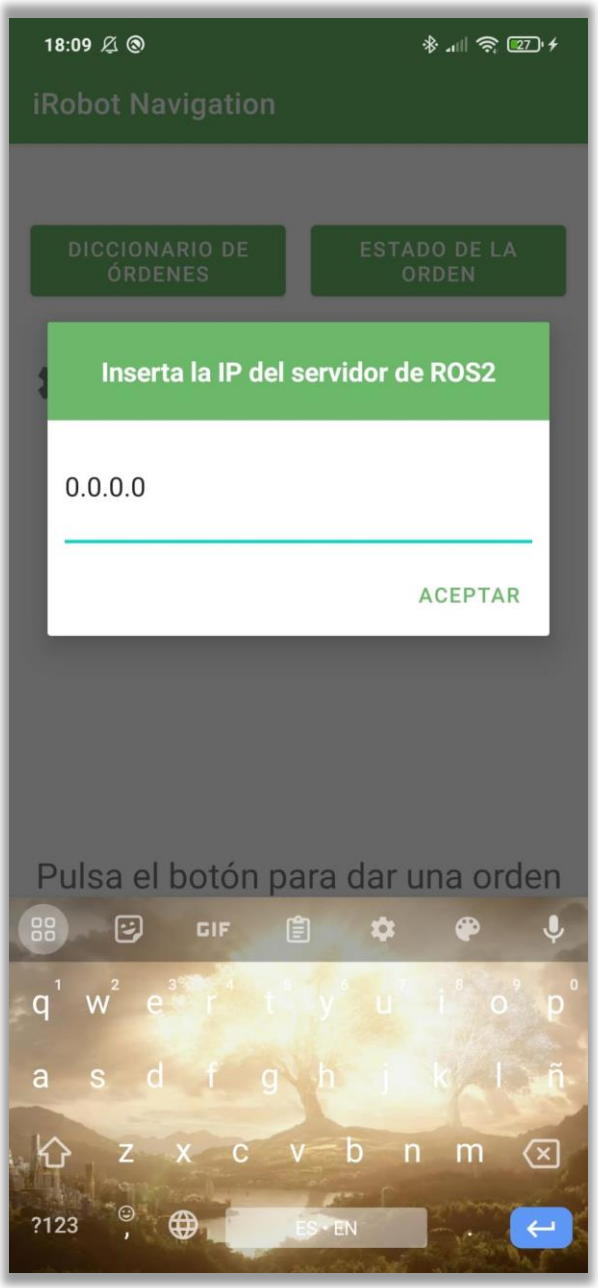

Figura 10: Configuración de la IP del servidor

Por último, se dispone de dos componentes de tipo `Button` [6] . El botón ‘Diccionario de órdenes’, se utiliza para mostrar al usuario un diccionario de las órdenes que el robot entiende y que puede ejecutar. Se muestra en la Figura 11.

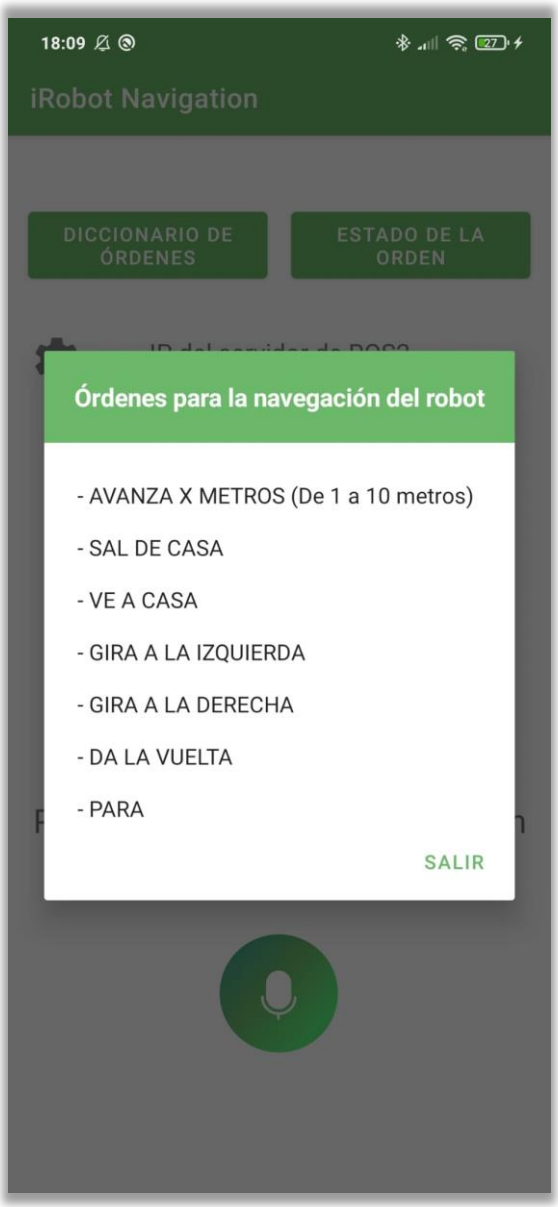

Figura 11: Diccionario de órdenes

El botón ‘Estado de la orden’, se utiliza para abrir un cuadro de diálogo para que el usuario decida si quiere recibir la realimentación (el estado de la orden mandada) en la aplicación a través de un botón `Switch`. Se puede observar en la Figura 12.

Figura 12: Estado de la orden

En la Figura 13 se presenta una definición de uno de los dos botones de la aplicación.

```
<Button
    android:id="@+id/btnConfig"
    android:layout_width="173sp"
    android:layout_height="60sp"
    android:text="Estado de la orden"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.932"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.116"
    tools:ignore="MissingConstraints" />
```

Figura 13: Componente Button

En el archivo `config_realimentación.xml` que se encuentra en la carpeta `layout` define un componente `SwitchCompat` [7] [8] que es utilizado en el cuadro de diálogo que surge cuando el usuario pulsa el botón de configuración. Sirve para establecer si el usuario quiere o no ver el estado de la orden dada en la aplicación. Por lo tanto, puede tomar dos valores, verdadero o falso. En la siguiente figura se muestra la definición del componente.

```
<androidx.appcompat.widget.SwitchCompat
    android:id="@+id/switchConfig"
    android:layout_width="229dp"
    android:layout_height="59dp"
    android:text="          No/Sí"
    android:textSize="17sp"

    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Figura 14: Componente SwitchCompat

Cuando se inicia la aplicación en el `MainActivity` a través del método `onCreate()`, se llama al método `setContentView()` [9] con el que se establece el contenido de la actividad principal en una vista explícita, es decir, establece el diseño de la interfaz de usuario que se muestra por pantalla al ejecutarse la actividad.

A través del parámetro que se pasa a este método (`R.layout.activity_main`), se especifica que el archivo `activity_main.xml` contenido en la carpeta `layout` es el que se usará como diseño. Siendo `R` una clase que se genera automáticamente por Android Studio que contiene referencias a todos los recursos definidos del proyecto.

```
setContentView(R.layout.activity_main);     //Es

microfono = findViewById(R.id.lottieMic);   //Asc
tv = findViewById(R.id.textV);
tv.setText("Pulsa el botón para dar una orden");
btnDic = findViewById(R.id.btnDiccionario);
btnConfig = findViewById(R.id.btnConfig);
checkOK = findViewById(R.id.imageCheck);
checkKO = findViewById(R.id.imageBadCheck);
```

Figura 15: Asociación de los objetos a la vista

Como se observar también en la figura anterior, posteriormente a la ejecución de este método se asocian las vistas (componentes) que se han explicado en el apartado `activity_main.xml` a los objetos creados para el manejo y configuración de dichas vistas. A través del método `findViewById()` se recogen las vistas por medio de su identificador `id` para almacenarlas en un objeto del tipo concreto de vista que sea.

6. DISEÑO DEL SERVIDOR DE ROS2

En este apartado se explica el diseño que se ha llevado a cabo en la construcción del servidor de ROS2 en un ordenador con Sistema Operativo Linux (distribución Ubuntu 22.04).

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. Son los siguientes:

- **package.xml** : contiene metadatos sobre el paquete. También se definen las dependencias con otros paquetes como son el paquete `'irobot_create_msgs'`, `'geometry_msgs'` o `'rclpy'` necesarios para la implementación del servidor y las instrucciones de movimiento del robot.

```
<exec_depend>rclpy</exec_depend>
<exec_depend>std_msgs</exec_depend>

<depend>irobot_create_msgs</depend>
<depend>geometry_msgs</depend>
```

Figura 16: Dependencias de package.xml

- **resource/servidorROS2** : archivo de marcadores para el paquete.
- **setup.cfg** : necesario al tener ejecutables, para que el comando 'ros2 run' pueda encontrarlos y ejecutarlos.
- **setup.py** : contiene instrucciones sobre cómo instalar el paquete. Cuando se instala, se define el punto de entrada '`irc3`' asociado a la función '`main`' del archivo '`irobot.py`' del paquete '`servidorROS`'.

```
entry_points={
    'console_scripts': [
        'irc3 = servidorROS.irobot:main',
    ],
},
```

Figura 17: setup.py

- **servidorROS2** : es un directorio con el mismo nombre que el paquete usado por las herramientas de ROS2 para encontrar el paquete. Contiene el archivo `__init__.py` y el archivo `irobot.py` que contiene el código del servidor.

El archivo `irobot.py` representa el código fuente del servidor. Está escrito en Python y tiene como objetivo recibir las órdenes que el usuario envía desde la aplicación móvil para posteriormente ejecutar en un nodo de ROS2 los comandos de movimiento del robot en función de la orden recibida, así como enviar una realimentación del estado de la orden a la aplicación.

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.

La clase `OrdenesClient` representa el nodo de ROS2. Cuando se recibe una orden, se ejecuta una función determinada que corresponde con el movimiento que ha pedido el usuario. El nodo de ROS2 se encarga de enviar al robot la orden para que este la ejecute.

Hay que mencionar que, además del paquete del servidor de ROS2, se utiliza un paquete adicional que ejecuta un nodo de ROS2 cuyo objetivo es mostrar por terminal los datos de los sensores infrarrojos del robot y determinar si el robot ha encontrado algún obstáculo durante la ejecución de sus órdenes.

7. REFERENCIAS

[1] «Cómo guardar datos simples con SharedPreferences | Android Developers». https://developer.android.com/training/data-storage/shared-preferences?hl=es-419#java (accedido 14 de junio de 2023).

[2] M. N. Moreno García y F. J. García Peñalvo, «TEMA 3 Patrones INGENIERÍA DEL SOFTWARE II 3º Grado en Ingeniería Informática».

[3] «ImageView | Android Developers». https://developer.android.com/reference/android/widget/ImageView (accedido 10 de junio de 2023).

[4] «TextView | Android Developers». https://developer.android.com/reference/android/widget/TextView (accedido 10 de junio de 2023).

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

[6] «Botones | Desarrolladores de Android | Android Developers». https://developer.android.com/guide/topics/ui/controls/button?hl=es-419 (accedido 10 de junio de 2023).

[7] «Switch En Android - Develou». https://www.develou.com/switch-en-android/ (accedido 10 de junio de 2023).

[8] «SwitchCompat | Android Developers». https://developer.android.com/reference/androidx/appcompat/widget/SwitchCompat (accedido 10 de junio de 2023).

[9] «Activity | Android Developers». https://developer.android.com/reference/android/app/Activity#setContentView(android.view.View) (accedido 11 de junio de 2023).