Universidad Carlos III de Madrid

Escuela Politécnica Superior

Grado en Ingeniería Telemática

Proyecto Fin de Grado

NAO Sports

Sistema de teleoperación mediante Android para jugar con el robot NAO

Autor: Jorge Alcolea Coronel

Directora: Nerea Luis Mingueza

Co-director: Moisés Martínez Muñoz

Agradecimientos

A Nerea y Moisés, por la ayuda prestada y por tener paciencia.

A mi familia, por su apoyo inconsciente.

A Espe, porque sin ella no hubiera sido posible.

A Jaime, Sandra, Santi y Jou porque forman parte de esto.

A mis resto, por estar ahí para desconectar.

Por último, a Lee Carvallo.

Resumen

En este trabajo, se presenta el desarrollo de un sistema de teleoperación para el robot NAO, mediante una aplicación para móviles basada en Android. Ésta consiste en un *framework* de juego y competición, para uno o dos jugadores; que podrán controlar un robot, de forma individual, a través de sus dispositivos móviles. La aplicación ha sido desarrollada con el fin de poder participar en diferentes juegos, mediante el control del robot NAO; disponiendo de un contador de tiempo de juego, marcadores para contabilizar los puntos y una base de datos para almacenar las puntuaciones. Para el desarrollo del proyecto se ha utilizado el IDE de desarrollo Android Studio, los software Choregraphe Suite y WeBots for NAO para la conexión y simulación de los robots, el SDK de desarrollo JNaoQi, que nos ofrece un conjunto de métodos para conectarse y controlar los robots y el robot humanoide NAO, el cual se ha utilizado para probar el correcto funcionamiento del sistema.

Abstract

This document describes the development of a teleoperation system for NAO robot by using a mobile application for Android phones. The app consists on a game and a competition framework for one or more players, who can control individually the robot with their mobile phones. The app has been developed with the aim of participating in different games by controlling the NAO robot, providing a time-keeper, scoreboards and a database to keep the scores updated. In order to develop this project, different software tools has been used, including: Android Studio IDE, for developing the app; Choregraphe and WeBots for NAO for robot's connection and simulation; SDK JNaoQi that offers a combination of different libraries to connect and control the NAO robot, and finally, the humanoid robot NAO, used to test the proper functionalities of the system in real environments.

Índice General

Índice de figuras

Índice de tablas

Capítulo 1: Introducción

1.1 Descripción del problema

Desde hace más de 60 años, los robots se han ido introduciendo poco a poco en la vida de los seres humanos. Éstos fueron utilizados inicialmente para realizar tareas repetitivas y peligrosas en fábricas pero, actualmente, se han convertido en piezas fundamentales en multitud de áreas, como el de la exploración espacial o la medicina. En los últimos años, los robots han sido introducidos en nuestras viviendas. Primero como mascotas y juguetes que reaccionaban a estímulos con comportamientos predefinidos [37], después como herramientas que ayudan a realizar las tareas del hogar [38]. Debido a esto, en los últimos años han aparecido en el mercado varios robots programables, que ponen al alcance, de cualquier persona que tenga interés, una serie de herramientas que permiten diseñar su comportamiento. Se pueden encontrar kits de construcción de robots de todo tipo de complejidad, o robots ya construidos como el robot NAO [32] desarrollado por Aldebaran Robotics. Estos robots están siendo utilizados en universidades para labores de investigación o en proyectos con fines terapéuticos y educativos con niños. Un ejemplo lo podemos encontrar en el proyecto NAOTherapist [28], desarrollado por el Grupo de Planificación y Aprendizaje (PLG) de la Universidad Carlos III de Madrid, donde el robot actúa como un terapeuta para los pacientes. Estos participan en sesiones en las que tienen que repetir los movimientos realizados por el robot que evalúa las posturas que adoptan y las corrige.

Por otro lado, los teléfonos móviles se han convertido en un dispositivo indispensable para la mayoría de la población. El auge de este tipo de tecnología tiene un punto de inflexión a partir de la aparición de los *smartphones* en la década 2000-2010. Esta nueva generación de teléfonos se caracteriza por ser más parecidos a pequeños ordenadores, que a simples aparatos utilizados para comunicarnos. Los *smartphones* ofrecen distintas herramientas -tanto a nivel de *software* como de *hardware*- que permiten crear aplicaciones de cualquier índole. Entre las aplicaciones más utilizadas por los usuarios nos encontramos los juegos [29], que aprovechan las distintas características de los *smartphones* para ofrecer distintas experiencias al usuario. Uno de los casos con más notoriedad de los últimos meses es el de Pokemon Go [30], que poco después de su lanzamiento se convirtió en una de las aplicaciones más descargadas de todos los tiempos. Parte de su éxito reside en la utilización de la realidad aumentada como componente principal del juego,

ya que la aplicación interactúa con la realidad, utilizando la cámara del dispositivo. Hace que los usuarios tengan que salir a la calle e interactuar con el entorno para conseguir sus objetivos.

El problema a resolver en este trabajo consiste en la realización de una aplicación para móviles Android, mediante la cual podamos jugar a través del robot NAO. Para resolver este problema es necesario cumplir dos objetivos principales:

- Diseñar e implementar una interfaz que se comunique con el robot. Esta interfaz debe permitir al usuario controlar al robot para que este realice a una serie de acciones y pueda interactuar con el entorno.
- Habrá que definir un marco de juego, ya que debe contar con unos objetivos a cumplir y la posibilidad de competir con otros usuarios. Una posible aproximación sería, debido al carácter general del problema a resolver, contar con la presencia de un árbitro para dotar al juego de una dinámica competitiva. El árbitro sería el encargado de definir los objetivos y de interactuar con los usuarios. Para ello, debe contar con un sistema mediante el cual poder modificar el estado de una partida en juego.

1.2 Motivación

La única condición es que tratara sobre el desarrollo de una aplicación Android. Esta condición venía motivada tras haber cursado una asignatura sobre Android en el último año, y la había disfrutado tanto que decidí intentar enfocar mi carrera laboral hacia el sector del desarrollo de aplicaciones móviles. Cabe destacar que desde hace unos años es un sector en alza y que tiene impacto en la vida cotidiana de muchas personas.

Entre todos los trabajos ofertados que involucraban el desarrollo de una aplicación Android, el trabajar con un robot me llamó la atención desde el principio. A pesar de no haber estudiado ninguna asignatura de robótica, era una rama de la ingeniería que me interesaba, en especial debido a que engloba otras muchas ramas en las que sí estoy formado como la electrónica, la programación o las comunicaciones.

El factor determinante para la elección de este trabajo es la versatilidad que tiene el sistema. Además de jugar con un robot o competir contra otros usuarios, también puede ser utilizado en modo cooperativo, con dos robots realizando acciones para ayudarse mutuamente a conseguir objetivos comunes. Las distintas posibilidades de utilización de la aplicación sólo están

limitadas por dos factores: (1) el nivel de control sobre el robot que nos proporcione el sistema de teleoperación; y (2) la imaginación de los usuarios.

1.3 Objetivos del trabajo

El objetivo principal de este trabajo consiste en el desarrollo de una aplicación para dispositivos Android que permita controlar el robot NAO con el fin de poder interactuar y competir con otros usuarios a juegos tradicionales del mundo real. Con el fin de llevar a término este trabajo, hemos dividido el objetivo principal de este trabajo en diferentes tareas, las cuales son descritas de forma más detallada a continuación:

1. **Análisis del funcionamiento del robot NAO**

   Debido a que nuestra aplicación pretende proveer de un marco con el que poder practicar distintos juegos con el robot NAO, es necesario familiarizarse con la mecánica del robot y las características estructurales y funcionales que ofrece. Esto supone obtener conocimiento acerca del funcionamiento de los actuadores y las limitaciones de movimiento, estudiar los distintos sensores y dispositivos de entrada/salida y decidir cuáles pueden ser utilizados para mejorar la experiencia de juego.

2. **Estudio del sistema operativo Android**

   La aplicación será desarrollada para dispositivos móviles que utilicen el sistema operativo Android, que es un sistema operativo basado en Linux, utilizado por el 65.9% de los dispositivos móviles del mundo [31]. Será necesario realizar un estudio en profundidad de sus funcionalidades, con el fin de definir qué elementos utilizar para la creación de interfaces y para la arquitectura de la aplicación.

   Los dispositivos móviles nos ofrecen una serie de componentes *hardware* que también tendrán que ser estudiados con el fin de averiguar si pueden ser incluidos en la aplicación a desarrollar. Multitud de sensores -como el acelerómetro y el giroscopio- y componentes -como el micrófono y la cámara- pueden ser controlados mediante librerías que nos proporciona Android.

3. **Estudio del SDK JNaoQi**

   NaoQi es el *framework* utilizado para programar el robot. Está disponible para los lenguajes C++ y Python y permite crear módulos para modelar el comportamiento sobre el

robot. Este *framework* nos ofrece una serie de librerías en distintos lenguajes para comunicarnos con el robot NAO. Debido a que la aplicación que se va a desarrollar debe poder ser ejecutada en un dispositivo Android (cuyo lenguaje nativo para el desarrollo de aplicación es Java) se utilizará la librería para Java JNaoQi, que permite conectarnos al robot y acceder a los distintos módulos de comportamiento necesarios: movimiento, visión, reconocimiento de objetos, etc.

4. **Diseño del sistema de juego**

   Puesto que nuestra aplicación consistirá en un sistema de competición, para uno o varios jugadores, es necesario definir unas reglas y elementos comunes a las distintas modalidades de juego, que doten a nuestra aplicación de la jugabilidad necesaria y proporcionen al usuario un entorno de juego sencillo y funcional. Para ello, será necesario mostrar un marcador y desarrollar un sistema, ajeno al usuario, que permita actualizarlo según éste vaya cumpliendo objetivos. Las partidas deben tener un final, el cual dependerá de los objetivos conseguidos por el usuario o del tiempo máximo de juego que sea establecido inicialmente.

5. **Diseño de la estructura de la aplicación**

   Una vez definido el funcionamiento básico, las características del sistema operativo Android y la estructura de nuestro sistema de juego, resulta indispensable definir una interfaz intuitiva y visualmente agradable para maximizar la experiencia del usuario. Hay que definir el número de actividades (pantallas) de la aplicación y la funcionalidad de cada una y podremos definir el flujo de ejecución, que consiste en definir cómo interactúan unas actividades con otras.

6. **Experimentación**

   En esta tarea será necesario elegir un proceso, con el fin de poder probar el correcto funcionamiento de la aplicación. Para ello se definirán un conjunto de pequeños juegos o tareas que deberán realizar varios usuarios utilizando la aplicación. Una vez realizadas las pruebas, los usuarios deberán rellenar un cuestionario. Los resultados obtenidos de todas las encuestas permitirán evaluar el grado de funcionalidad de la aplicación, así como la experiencia de usuario.

7. **Elaborar la documentación del trabajo**

   Esta tarea consiste en el desarrollo de la memoria final que describe todas las fases del desarrollo del proyecto, así como el funcionamiento final del mismo. Aunque se presente como el último de los objetivos, la documentación se ha confeccionado a base de notas y apuntes tomados a lo largo de la elaboración del trabajo; dejando para el final el apartado referente a las conclusiones y la maquetación.

1.4 Estructura del documento

El documento está dividido en varios capítulos.

1. **Introducción:**

   En el primer capítulo se encuentra la introducción a este documento que nos permite presentar el contexto en el cual se enmarca este trabajo.

2. **Estado del arte:**

   En el segundo capítulo se encuentra el marco teórico investigado en la realización del trabajo.

3. **Descripción del sistema:**

   En el tercer capítulo encontramos el análisis y el diseño del sistema desarrollado.

4. **Experimentación:**

   En el cuarto capítulo se describen las pruebas que se han llevado a cabo para probar el funcionamiento del sistema.

5. **Gestión del proyecto:**

   En el quinto capítulo encontramos la información referente a la planificación de las distintas fases de desarrollo del proyecto.

6. **Conclusiones y trabajos futuros:**

   En el sexto capítulo se presentan las conclusiones obtenidas al terminar este trabajo, tanto las generales como las extraídas de cada uno de los objetivos.

7. **Anexos:**

   En el último capítulo se presentan los anexos necesarios para el correcto uso de la aplicación.

Capítulo 2: Estado del Arte

2.1 Historia de la robótica

La robótica surgió como el resultado de la combinación de diversas ramas de la ingeniería, como la mecánica y la electrónica. Trata sobre el diseño y la construcción de robots. El término robótica fue usado por primera vez en 1942 en el relato de ciencia ficción 'Liar!' de Isaac Asimov [1], aunque éste no era consciente de que estaba acuñando el término cuando lo utilizó por primera vez, ya que asumió que era la palabra que se utilizaba para denominar la ciencia relacionada con los robots [2].

La palabra robot fue acuñada mucho antes, en 1920, por el escritor checo Karel Čapek en su obra de teatro *Rossum's Universal Robots* [3], y proviene de la palabra eslava *robota*, cuya traducción es mano de obra. En dicha obra, se definía a los robots como criaturas creadas artificialmente y que eran utilizados por los humanos para realizar tareas peligrosas y trabajos pesados.

Los primeros entes que pueden ser considerados como robots, son los autómatas. En la Grecia del siglo IV a.c, el filósofo y matemático Arquitas de Tarento construyó el que está considerado primer autómata mecánico de la historia*, la paloma voladora (Ilustración 1)*, que estaba propulsado por vapor. El cuerpo era de madera y fue uno de los primeros estudios sobre el vuelo de los pájaros [4].

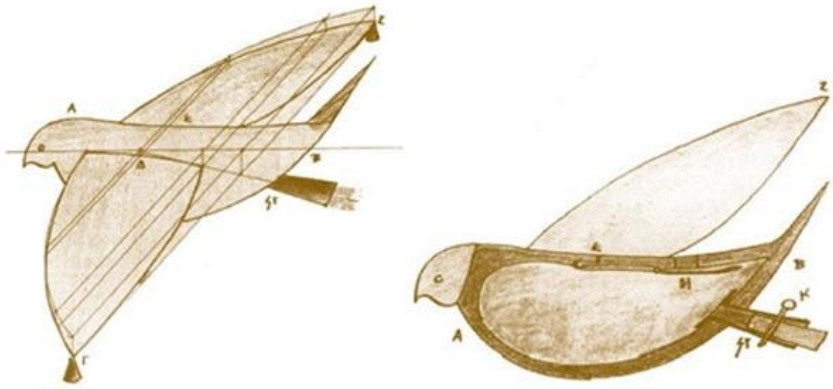

Ilustración 1 - Paloma voladora de Archytas

En el siglo I a.C, el ingeniero y matemático Herón de Alejandría construyó un autómata totalmente mecánico que representaba una obra de teatro, formado por máquinas simples operadas mediante una rueda dentada. Pero la mayoría de los escritos y diseños de Herón se perdieron.

Ismail al-Jazari fue un inventor musulmán que vivió en el siglo XII. Es conocido por escribir el libro *The Book of Knowledge of Ingenious Mechanical Devices* [5], donde describía y daba instrucciones para la construcción de más de cien ingenios mecánicos. Además, diseñó y construyó varios autómatas propulsados por la fuerza del agua, entre ellos un autómata musical (Ilustración 2), que consistía en un barco con cuatro músicos programados para tocar distintos ritmos y melodías [6].

Ilustración 2 - Barco musical de Al-Jazari

Uno de los primeros diseños de un robot con forma humana, de los que tenemos constancia, perteneció a Leonardo Da Vinci, y data de 1495. En los cuadernos de Da Vinci, que fueron encontrados en 1950, se encuentran detallados diseños sobre un caballero que era capaz de mantenerse de pie, sentado y mover independientemente sus brazos. El robot era operado a través de una serie de correas y cables. El diseño parece estar basado en su estudio anatómico *El hombre de Vitruvio* [7] y no existen pruebas de que llegara a construirlo [8].

Durante el siglo XVIII, varios inventores realizaron contribuciones reseñables. Jacques de Vaucanson construyó tres autómatas, siendo el tercero un pato mecánico (Ilustración 3, izquierda), con la aparente habilidad de comer grano y metabolizarlo [9]. El relojero suizo Pierre Jaquet-Droz trabajó para la realeza europea construyendo tres muñecas (Ilustración 3, centro)

con funciones distintas: una podía escribir, otra tocar música y la última dibujar [10]. Por último, en 1801, Joseph Jacquard construyó un telar automático (Ilustración 3, derecha) que era controlado con tarjetas perforadas, cabe reseñar que más de un siglo después las tarjetas perforadas fueron utilizadas como método de entrada de datos para las primeras computadoras [11].

Ilustración 3 - El pato de Jacques de Vaucanson (1), muñecas de Pierre Jaquet-Droz (2) y el telar de Joseph Jacquard (3)

La llegada de la revolución industrial supuso uno de los periodos con mayor impacto de transformación en la sociedad. Constituyó un paso desde la economía rural, basada principalmente en la agricultura, a una economía industrializada y mecanizada. Esta transición supuso pasar de la producción a mano, a la producción mecanizada; siendo, no otro sino este motivo, por el que surgieron los primeros autómatas que sustituyeron a las personas en la realización de tareas repetitivas y complejas. Algunos ejemplos se observan con la llegada de los telares automáticos, como el de Joseph Jacquard [11], que supusieron un impacto directo en la industria textil, dominante en términos de empleo en esa época.

Los avances tecnológicos y científicos que se produjeron a principios de la segunda mitad del siglo XX supusieron un punto de inflexión en el campo de la robótica:

- La implementación práctica del transistor revolucionó el campo de la electrónica [12]. Concebido en 1926 por Julius Lilienfeld; fue un grupo de físicos de los laboratorios Bell los que ganaron el Nobel de Física en 1956 por este logro.
- Los avances en el campo de la informática hicieron posible empezar a utilizar programas de ordenador para controlar el movimiento de los robots.
- La conferencia de Dartmouth en 1956 fue el germen de la investigación sobre inteligencia artificial [13].

En 1956, el inventor norteamericano George Devol fundó *Unimation*, la primera empresa en producir un robot para uso industrial. El nombre del robot era *Unimate* (Ilustración 4), y fue inventado por Norman Heroux. En 1961 fue instalado en una cadena de montaje de General Motors en Nueva Jersey. Su función era transportar piezas fundidas y soldarlas a la carrocería de los coches, una tarea muy peligrosa para los trabajadores [14].

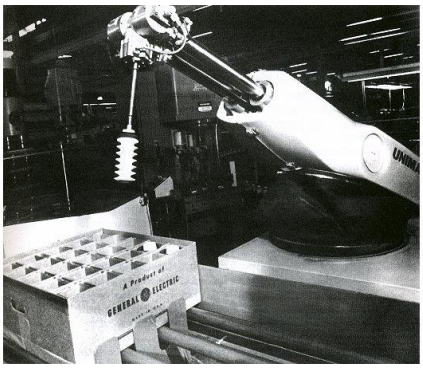

Ilustración 4 - Unimate en la cadena de montaje de General Motors

En 1969, un estudiante que trabajaba en el Laboratorio de Inteligencia Artificial de la Universidad de *Stanford* llamado Victor Scheinman [15] diseño *Stanford Arm*, el primer brazo robótico controlado por ordenador. El diseño de este robot se convirtió en un estándar y sigue influenciando el diseño de los brazos robóticos a día de hoy. Años después, en 1977, trabajando para Unimate, Victor Scheinman creo PUMA (Ilustración 5), un brazo robótico basado en los diseños originales del *Stanford Arm* destinado a tareas de montaje. Este brazo robótico fue comercializado hasta 1990 .

Ilustración 5 - Unimate PUMA

En 1981, Takeo Kanade construye el *Direct Drive Arm.* Era un brazo mecánico que tenía los motores instalados directamente en las articulaciones del brazo [16], esta mejora hacía que

pudieran moverse con mucha más libertad y rapidez que los brazos robóticos diseñados hasta la fecha. Debido a su éxito, Kanade empezó a trabajar en una segunda versión llamada *Direct Drive Arm II,* un brazo robótico con seis grados de libertad, que incorporaba sensores de proximidad.

A finales de los años 80, el robot Genghis [17] fue creado por un estudiante del *Massachusetts Institute of Technology* llamado Rodney Brooks. Era un robot hexápodo en el que todas las piernas se movían de manera independiente. Pesaba aproximadamente un kilogramo. Fue construido con la filosofía de reducir costes de producción al ser un robot más pequeño y simple que los que se venían diseñando para enviar al espacio, de esta manera podrían producirse más robots y más rápido.

Dante I y Dante II [18] fueron desarrollados en *Carnegie Mellon University* en 1993 y 1994 respectivamente. Su misión era recolectar información en entornos con condiciones extremas, similares a las encontradas en otros planetas. El primero fue enviado a un cráter en la Antártida pero la misión fracasó al romperse la cuerda con que descendía. Dante II descendió dentro del volcán Spurr y la misión fue considerada un éxito.

En 1994 Marc Thorpe crea *Robot Wars*, la primera competición de lucha entre robots. En ella, dos robots se enfrentaban entre sí y el objetivo era destruir al rival. Se celebró anualmente durante cuatro años, después se convirtió en un programa para la televisión británica.

En 1996, el doctorando de MIT David Barrett construye un robot biomimético, *RoboTuna* (Ilustración 6), que trataba de reproducir la manera en los atunes nadaban y actuaban dentro del agua. El objetivo de esta investigación era buscar mejoras para los sistemas de propulsión de los AUVs (*Autonomous Underwater Vehicles*), que son robots acuáticos que no necesitan de un operario que los controle [19].

La competición internacional de robótica *RoboCup*, fundada por los investigadores Manuela M. Veloso y Peter Stone, fue realizada por primera vez en 1997, y desde entonces se realiza de manera anual siendo su objetivo promover la investigación en Inteligencia Artificial. En ella, unos robots autónomos juegan una competición de fútbol.

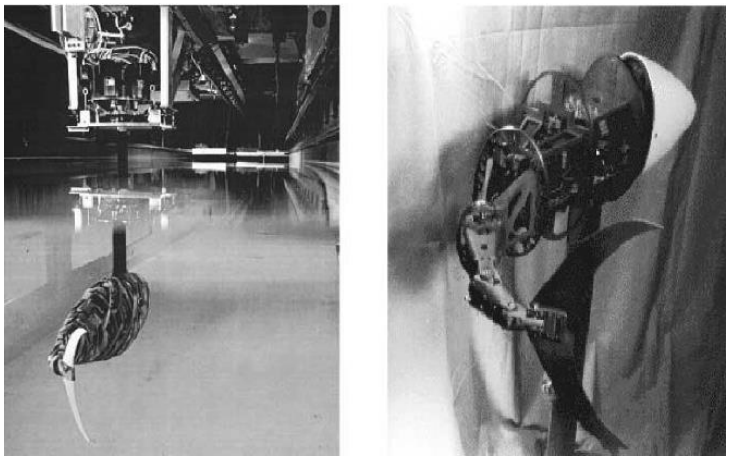

Ilustración 6 - RoboTuna en las instalaciones del MIT

Entre julio y septiembre de 1997 el *Rover Sojourner* recorre Marte, como parte de la Misión *Pathfinder*. El objetivo de esta misión era recoger datos para analizar la atmósfera, el clima y la composición de las rocas y el suelo marciano.

En 1998, varias empresas ya habían empezado a comercializar robots con fines educativos y de entretenimiento. Algunos ejemplos fueron el *Furby* de Tiger Electronics (Ilustración 7) y la mascota electrónica *AIBO* de Sony (Ilustración 8). Por otro lado, LEGO había empezado a comercializar kits de desarrollo de robots.

Ilustración 7 - Furby de Tiger Electronics

Ilustración 8 - AIBO de Sony

En el año 2000, la empresa japonesa Honda desarrolla el robot humanoide ASIMO. Fue diseñado con el objetivo de ayudar a personas con discapacidad motora, y entre sus habilidades encontramos la capacidad de reconocer posturas, gestos y objetos en movimiento, así como sonidos y caras que le dan le habilitan para interactuar con humanos.

La compañía iRobot, fundada por Rodney Brooks, lanza en 2002 el robot aspiradora Roomba. Se calcula desde su lanzamiento se han vendido más de 10 millones de unidades.

En 2008, después de cuatro años de desarrollo, fue lanzado a nivel académico el robot NAO (Ilustración 9), un robot humanoide desarrollado por Aldebaran Robotics. Es autónomo y programable, contando con un sistema operativo propio. Desde 2008 sustituye al robot AIBO de Sony en la competición RobotCup. NAO fue lanzado a disposición del público en 2011.

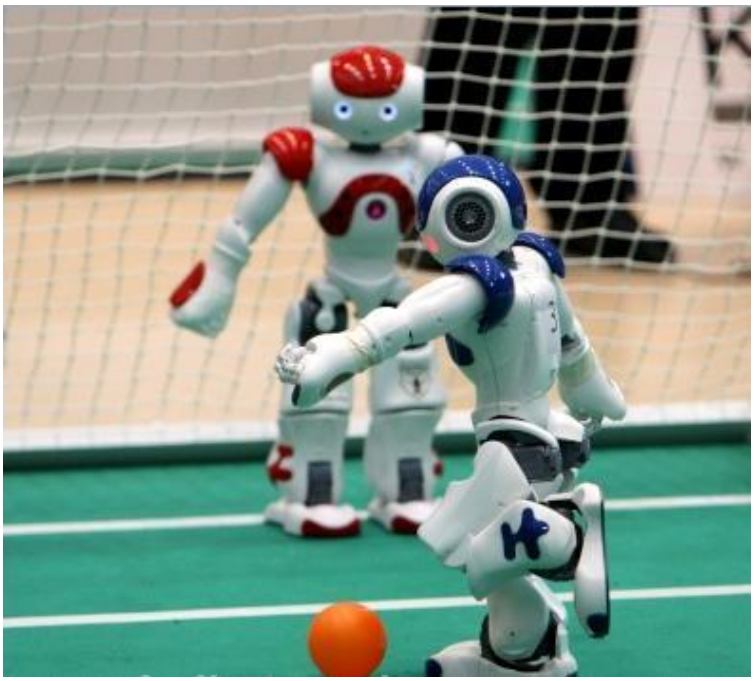

Ilustración 9 - Dos robots NAO compitiendo en la RoboCup

En 2012, el *rover* Curiosity aterrizó en la superficie de Marte. Es el más grande hasta la fecha de los vehículos de exploración espacial, enviados al planeta rojo. Sus objetivos son estudiar el clima y la geología del planeta, además de analizar si se pueden dar las condiciones necesarias para albergar vida. Los diseños del Curiosity servirán de base para el próximo *rover* que la NASA tiene pensado enviar a Marte en 2020.

2.2 Teleoperación

Se define la teleoperación como la acción de operar un sistema a distancia. Cualquier herramienta que extienda la acción mecánica de una persona más allá de su alcance es un teleoperador. En el campo de la robótica, la teleoperación es un medio para operar un robot utilizando la inteligencia humana, lo que requiere de una interfaz adecuada entre el humano y la máquina. La Ilustración 10, nos presenta un sistema de teleoperación mediante el cual un operador humano puede interactuar con un entorno remoto. Un sistema de teleoperación, normalmente, consiste en dos sistemas separados pero conectados, de tal forma, que permiten al operador humano controlar uno de ellos, llamado *maestro*, y generar acciones que son realizadas por el sistema remoto, llamado *esclavo*, sobre un entorno que no está al alcance del operador.

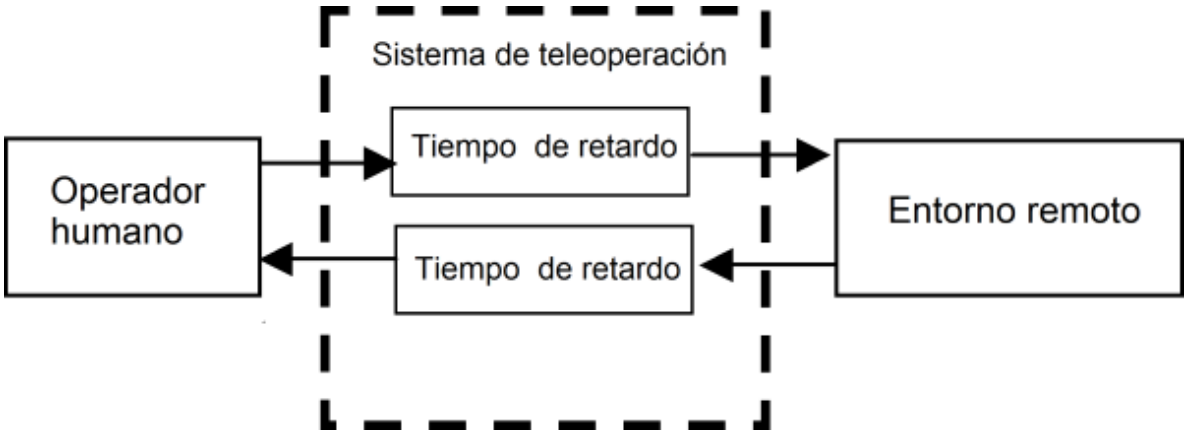

Ilustración 10 - Esquema básico de teleoperación

La función principal de un sistema de teleoperación es asistir al operador a realizar tareas complejas en entornos que puedan ser peligrosos o inaccesibles para él, como pueden ser plantas nucleares, las profundidades marinas o el espacio exterior [20]. Uno de los principales problemas que nos encontramos en la teleoperación es el *time delay (tiempo* de retardo), que existe en todos los sistemas electro-mecánicos. En muchos casos es imperceptible pero en otros puede hacer que el sistema sea inestable y hacer imposible la teleoperación. El problema del tiempo de retardo supone, a nivel técnico, uno de los retos a depurar en el futuro de la teleoperación.

2.2.1 Usos y aplicaciones

El primer sector científico que se acercó al campo de teleoperación fue el de la energía nuclear, debido a que era necesario tratar con elementos radiactivos, que son altamente nocivos para el ser humano [21]. Por este motivo, la compañía Central Research Laboratories fue contratada en 1945 para desarrollar un manipulador remoto para el Argonne National Laboratory. El encargo consistía en un mecanismo que se operase a través de una pared, para que el operario trabajase en condiciones óptimas. Fue en 1949, cuando Raymond Goertz presentó el primer teleoperador mecánico maestro/esclavo de la historia, el M1 [22], que se puede ver en la Ilustración 11. Este consistía en dos manipuladores prácticamente iguales, en los que los movimientos entre ambos se transmitían eje a eje de modo que ambos realizaban el mismo movimiento. En el extremo esclavo había una pinza que reproducía los movimientos realizados por el extremo maestro.

Ilustración 11 - Goertz manejando el teleoperador M1

Unos años después, el propio Goertz -que había continuado con la investigación de la teleoperación y desarrollando nuevos prototipos a partir del M1- introdujo motores en ambos manipuladores, y presentó el E1 (Ilustración 12), el primer sistema de teleoperación maestro-esclavo accionado por electricidad y con un sistema de control mediante servocontroles.

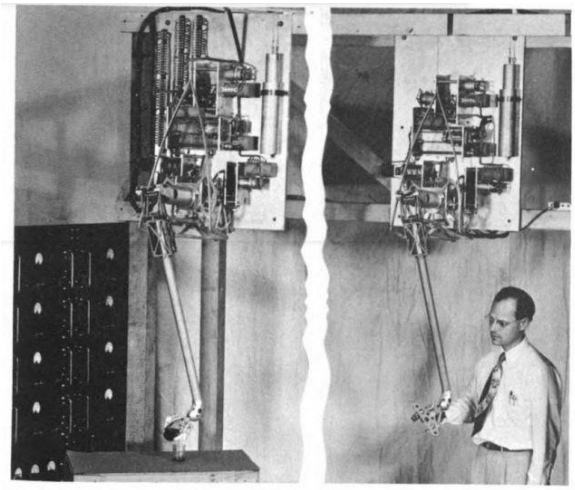

FIGURE 6.—The ANL Model E1 electric master slave. Used only for experimental purposes, this bilateral manipulator was developed in 1954. (Courtesy of Argonne National Laboratory.)

Ilustración 12 - Goertz manejando el teleoperador E1

En los últimos 60 años, la teleoperación ha avanzado conjuntamente con la informática y la robótica. Diferentes sistemas de teleoperación han sido desarrollados para permitir a los operadores humanos realizar sus tareas, de manera remota, en los entornos más extremos:

- La teleoperación se ha utilizado en aplicaciones espaciales. La mayoría de sistemas teleoperados enviados al espacio tenían controles relativamente sencillos pero fiables, con la habilidad de ser reprogramados en el espacio. Los primeros vehículos teleoperados mandados al espacio fueron los Lunujod 1 y 2 [23] (Ilustración 13), enviados a la luna por la Unión Soviética entre 1970 y 1973. Fueron diseñados por Alexander Leonovich Kemurdzhian, un pionero del programa de vuelos espaciales de la URSS. Contaban con paneles solares para recargar baterías y con varias cámaras para explorar el entorno; además de varios sensores como espectrómetro y telescopio de rayos-X, así como varias antenas para comunicarse con la Tierra. Hasta que el *Mars Pathfinder* aterrizó en Marte en 1998, eran los únicos robots teleoperados situados en un entorno extraterrestre. En 1993, la agencia alemana DLR demostró exitosamente el funcionamiento del primer telerobot, con el experimento *Rotex*. Este experimento demostró la habilidad de un ordenador para controlar un robot en el espacio, además de que la teleoperación espacial podía ser controlada desde la tierra a pesar del tiempo de retardo [24].

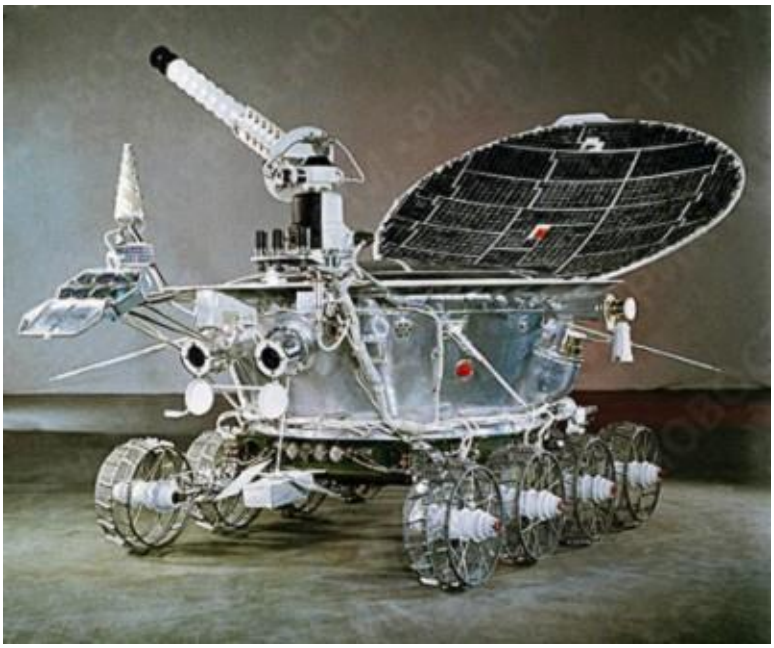

Ilustración 13 - Rover Lunojod 1

- En plantas y vertederos nucleares, la presencia de contenidos radiactivos hace que el entorno sea muy peligroso para los humanos y todas las tareas se realizan mediante teleoperación. Los sistemas varían desde teleoperadores similares a los desarrollados por Goertz, para manejar pequeñas cantidades de elementos radioactivos; hasta camiones con manipuladores hidráulicos que pueden cargar toneladas. Un ejemplo de estos últimos son los STR-1 [25] (Ilustración 14), desarrollados por la compañía TransMash y diseñados por Alexander Leonovich Kemurdzhian, el mismo responsable de los Lunojod-1 y 2 [23]. EL STR-1 será tristemente recordado por ser uno de los vehículos teleoperados que trabajaron en las tareas de sellado del reactor nuclear de Chernobyl. El diseño era muy parecido al de los mencionados Lunojod-1 y 2, con la diferencia de que contaba con un *bulldozer* en su parte delantera para empujar desechos radioactivos al interior del reactor. La teleoperación se realizaba mediante radiocontrol, el operario podía visualizar el terreno con la ayuda de dos cámaras, una apuntaba hacia delante y otra hacia atrás. La interfaz de teleoperación sólo contaba con las instrucciones de ir hacia delante, detrás, izquierda, derecha, parada y parada de emergencia.

Ilustración 14 - STR-1 en Chernobyl

- En las profundidades marinas la teleoperación se ha utilizado en exploraciones y plataformas, en la inspección y mantenimiento de taladros submarinos y en exploraciones históricas para encontrar e inspeccionar barcos hundidos. El vehículo teleoperado Jason [26], tiene la capacidad de sumergirse hasta 500 kilómetros. La teleoperación se realiza desde un barco, comunicándose con el sistema mediante un cable. Un prototipo de éste, Jason Jr, fue el utilizado para explorar los restos del Titanic (Ilustración 15).

Ilustración 15 - Jason Jr explorando los restos del Titanic en 1986

Hoy en día, los sistemas teleoperados son utilizados para multitud de aplicaciones, y se están convirtiendo en la forma habitual de realizar tareas, debido a la precisión que nos ofrece la utilización de máquinas para realizar tareas supervisadas por personas.

2.2.2 Tipos de interfaces

Las interfaces humano-máquina son uno de los elementos clave para poder teleoperar con éxito. Éstas deben proveer de las herramientas necesarias para que el operario pueda percibir el entorno remoto y pueda tomar decisiones. Deben ser diseñadas para minimizar el tiempo de adaptación del usuario. Es importante recalcar que el nivel de autonomía del robot no disminuye la importancia de la interfaz, ya que a pesar de que el teleoperador realice tareas de manera independiente, siempre tendrá que comunicar al operador qué es lo que hizo y cómo lo hizo. Con el tiempo, las interfaces serán usadas cada vez menos para control y más para monitorización y diagnosis. Existen cuatro tipos de interfaces: directa, multisensorial, control supervisado y novel. Los cuales se describen a continuación.

2.2.2.1 Directa

La interfaz directa es la interfaz clásica en la teleoperación. El operador controla el robot por medio de controladores de mano, como podría ser un *joystick* o una aplicación de móvil, y recibe la visión remota por medio de cámaras incorporadas en el robot. El control se realiza en primera persona ya que estamos viendo exactamente lo que ve el robot como si estuviéramos dentro de él, esta aproximación se conoce como *inside-out*. Las interfaces directas se utilizan cuando es imprescindible que el operador tome las decisiones en tiempo real, consiguiendo un nivel de telepresencia bajo pero significativo, ya que se tiene el control de todo el proceso. La condición óptima para utilizar esta interfaz es contando con una conexión con gran ancho de banda y mínimo tiempo de retardo o *delay.* El control directo, en presencia de retardos, es tedioso y por lo tanto propenso a que se cometan errores. Ejemplos de interfaz directa los podemos encontrar en los STR-1 que se utilizaron en las tareas de sellado del núcleo de Chernobyl. En la actualidad este tipo de interfaces se utilizan en sistemas para recorrer el alcantarillado, vehículos sumergibles y rescates en sitios inaccesibles como pueden ser minas derrumbadas.

2.2.2.2 Multisensorial

El control multisensorial implica un robot muy complejo, con multitud de sensores incorporados, de los que el operador puede recibir muchos tipos distintos de datos.

La información recibida por todos estos sensores se debe analizar y utilizar para resolver situaciones dinámicas. Las interfaces multisensoriales son útiles para aplicaciones que exigen acciones específicas, cuando es necesario elegir entre los modos de control y las pantallas de visualización, dependiendo de los requerimientos de la situación. Un ejemplo de interfaz multisensorial era la utilizada para operar el robot Dante II [18], que fue enviado al interior del volcán Spurr en Alaska para recopilar datos.

2.2.2.3 Control supervisado

Con el control supervisado, se divide el problema en una serie de tareas que el robot puede realizar por sí mismo. Para ello, debe tener cierto nivel de autonomía. Una vez que se le da el control al robot, el operador asume un rol de monitorización de los resultados. No todo el proceso se deja del lado del robot, ya que puede llegar a estados en los que no sepa qué hacer o por no haber realizado las tareas correctamente. En este caso, el teleoperador puede tomar el control para ayudar al robot a realizar las tareas. Como resultado de las estrategias de control supervisado, se reducen los datos de comunicación entre operador y máquina, siendo una aproximación perfecta para situaciones en las que se disponga de un bajo ancho de banda y el tiempo de retardo sea un factor a tener en cuenta. En este tipo de interfaces, hay que prestar atención en el diseño de la pantalla de control así como en el intercambio de información entre el operador y el robot.

2.2.2.4 Novel

Se conoce interfaz *novel* a cualquier método de entrada no convencional. Ocurre que, con el paso del tiempo, el uso de algunos de los métodos *novel* se normaliza y dejan de ser consideradas así. Por ejemplo, las interfaces basadas en aplicaciones web fueron consideradas *novel* con el auge de Internet. Otro ejemplo lo podemos encontrar en *Ghengis* (Ilustración 16), el robot con forma de insecto desarrollado por Rodney Brooks para el MIT. Las patas no estaban controladas por un sistema de control central, sino que cada una de ellas decidía donde posicionarse y que hacer dependiendo de las circunstancias [4].

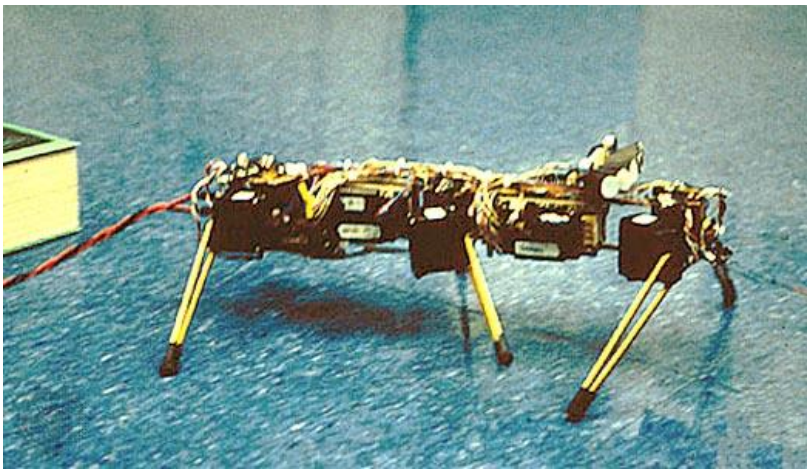

Ilustración 16 - Robot Genghis

2.2.3 Telepresencia

La telepresencia se refiere al ideal de que el operador tenga la suficiente información para que pueda sentir que está *físicamente* en el mismo espacio donde se encuentra el sistema teleoperado. Una combinación de cámara y monitor crea cierta sensación de presencia, pero normalmente se utilizan sistemas más sofisticados para poder llamarlo telepresencia. Los métodos típicos para crear esta sensación son la combinación de cámaras, efectos de sonidos, visión estereoscópica, realimentación de fuerzas y sentido del tacto. Para percibir una sensación de telepresencia máxima, todos los sentidos humanos deberían ser transmitidos desde el sistema teleoperado al operador. Los sentidos de la vista , el oído y el tacto son relativamente sencillos de transmitir, no ocurriendo lo mismo con el olfato y el gusto. Afortunadamente, estos sentidos son raramente importantes en la teleoperación de máquinas.

2.2.3.1 Vista

Los humanos percibimos más del 90% de la información de los que nos rodea por medio de la visión. Los sensores de visión humanos, los ojos, son sistemas muy complejos que nos permiten ver un campo de visión de 180 grados horizontalmente y 120 grados verticalmente. El área de enfoque es de sólo unos pocos grados, pero los movimientos y otros fenómenos son captados desde todo el área de visión. Es extremadamente difícil desarrollar un sistema de teleoperación que imite al ojo humano y que transmita al operador la misma cantidad de información que si se encontrara donde el sistema teleoperado.

En la mayoría de las configuraciones de realimentación visual (desde los simples monitores a complejos sistemas de telepresencia) el campo de vista se ve reducido por las tecnologías de la cámara o el monitor. También se ve afectada la percepción de distancias debido a la falta de vista en profundidad.

2.2.3.2 Oído

Es muy difícil crear la sensación de telepresencia sin sonido. Por ejemplo, cuando conducimos un vehículo, los sonidos que éste produce internamente y sobre el entorno son una información muy importante para nosotros con el fin de intuir si está averiado o tiene una rueda pinchada. Cuando teleoperamos un sistema ocurre lo mismo, los sonidos que recibimos nos puedan dar una información importante. Con la ventaja de que, en la teleoperación, la transmisión eléctrica del sonido nos permite modificar su intensidad y filtrar los sonidos que no nos interesen.

2.2.3.3 Tacto

Cuando palpamos algo, los sensores del tacto se activan. Estos sensores están situados por todo el cuerpo y son capaces de sentir el movimiento de las articulaciones, la tensión en los músculos y el contacto con la piel. En la teleoperación, la información referente al tacto la podemos dividir en dos clases:

- La realimentación de fuerza se refiere a la fuerza generada por un teleoperador, normalmente un manipulador. Ésta es enviada al operador para generar una respuesta real en acciones de agarre o manipulación. Cuando los manipuladores eran mecánicos, esta característica venía implementada de serie porque la fuerza del teleoperador era la empleada por el manipulador [22]. Con la implantación de los servos eléctricos e hidráulicos esta realimentación se perdió. En estos casos, la realimentación se genera artificialmente midiendo la fuerza generada sobre robot por medio de sensores y enviándola al operador. En tareas de manipulación, la realimentación de fuerzas es fundamental para tener una buena telepresencia.
- La realimentación háptica (información táctil) puede ser representada de varias maneras en la teleoperación [36]. El sentido del tacto del teleoperador puede ser realimentado a los dedos del operador. Como puede observarse en la Ilustración 17, el robot Hiro III simula el tacto de los objetos virtuales a través de una interfaz háptica. También podría ser

un caso de realimentación háptica la vibración de un vehículo teleoperado realimentada hacia a la piel de operador.

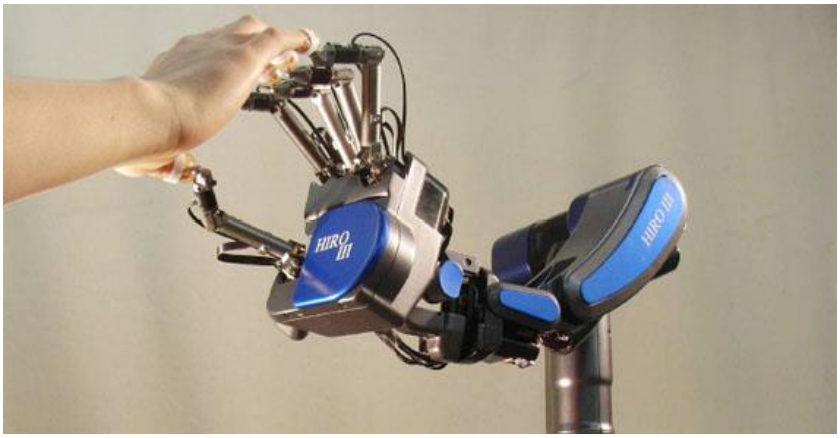

Ilustración 17 - El robot Hiro III presenta una interfaz háptica

2.3 Uso interactivo entre humanos y robots

Muchos objetos de nuestro día a día incorporan tecnologías robóticas. Por ejemplo, los sistemas de asistencia de aparcamiento y sistemas anti-colisión que incorporan los coches modernos son tipos de robots. Otro ejemplo son los robots domésticos (Ilustración 18) que hay en el mercado.

Ilustración 18 - Gato sentado sobre el robot limpiador Roomba

Poco a poco, los robots han ido introduciéndose en nuestras vidas sin darnos cuenta. Y no nos hemos dado cuenta porque a la mayoría de estos robots los seguimos viendo como una

herramienta, un aparato que realiza las tareas que le mandamos. Si pensamos de esta manera, estaremos equiparando la relación humano-robot a la de maestro-esclavo, limitando la capacidad de los sistemas robóticos a la habilidad del operador y a la calidad de la interfaz de usuario. Lo que nos propone Fong en [26] es un nuevo paradigma que nos aliente a pensar en una sinergia humano-robot, que permita a los robots aprender de los humanos y a los humanos pensar en los robots como compañeros en vez de herramientas.

El control colaborativo [26] es un sistema en el que los humanos y los robots trabajan juntos. El humano sirve como un recurso más para el robot, aportándole información igual que el resto de sus módulos. En particular, se le permite al robot hacerle preguntas al humano para obtener asistencia, así éste puede compensar la falta de autonomía del robot. De esta manera, si el robot tiene varias soluciones para un problema al realizar una tarea concreta, podría preguntar al humano cuál es la solución óptima. El control colaborativo se presenta como una nueva interfaz de teleoperación.

Las relaciones simbióticas [27] van un paso más allá del control colaborativo, presentan unos robots con mayor autonomía que no necesitan la supervisión de un humano, ya que no buscan la asistencia o confirmación para la realización de una tarea. En una relación simbiótica, el robot y el humano se benefician mutuamente, pidiendo y recibiendo ayuda para tareas que no pueden ser realizadas por la falta de capacidades, coordinando sus acciones sólo cuando necesitan ayuda.

2.4 Aplicaciones móviles para teleoperación de robots

Con la aparición del iPhone en 2007 y poco después del auge del sistema operativo Android, los móviles dejaron de ser dispositivos utilizados solamente para hablar por teléfono. Los *marketplace* de aplicaciones proveen una plataforma para la distribución, tanto gratuita como de pago, de todo tipo de aplicaciones que son desarrollados mediante *frameworks* que están a disposición de todo el mundo. Esto, unido a los distintos dispositivos periféricos y a los sensores incorporados en los dispositivos, hace de ellos los elementos ideales para utilizarlos como sistemas de teleoperación. En el caso del robot NAO [32], que ofrece herramientas de desarrollo para ser utilizados por distintas plataformas, existen varias aplicaciones en el mercado que nos permiten teleoperar el robot:

2.4.1 NAO para Android

- **NAO Robot Controller:** Desarrollada por Robin Bonnes como un trabajo de la universidad, es un sistema de teleoperación supervisado que consta de varias pantallas en las que podemos consultar la información de estado del robot (batería y temperatura), realizar capturas con la cámara del robot, hacer que se desplace y que realice acciones predefinidas como bailar. En las Ilustraciones 19 y 20 vemos dos capturas de pantalla de la interfaz.

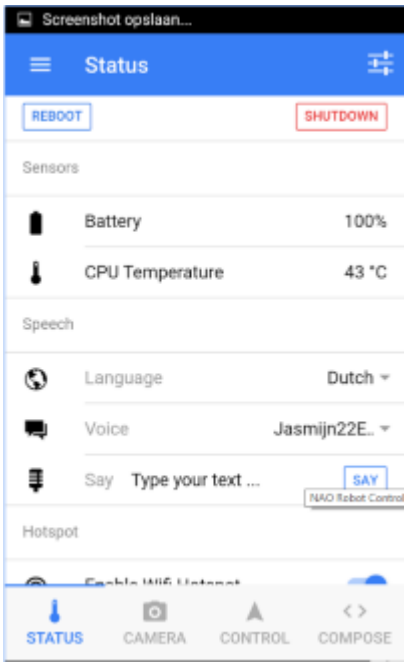

Ilustración 19 - Pantalla de estado de la aplicación Nao Robot Controller

Ilustración 20- Pantalla de control de movimiento de la aplicación Nao Robot Controller

- **NAO Communicator:** Esta aplicación, desarrollada por H. Eileres, tiene la funcionalidad de reconocer los robots conectados a la misma red que el dispositivo, pudiendo conectarse a ellos sin necesidad de introducir la dirección IP del robot. Igual que NAO Robot Controller, ofrece un sistema de teleoperación supervisado que nos permite cambiar la postura del robot y realizar ciertas acciones predefinidas, como ejecutar ciertas animaciones sobre el robot.
- **NaoController:** Desarrollada por Juan Domingo Gálvez como trabajo fin de grado para la Universidad Carlos III de Madrid. Se trata de una interfaz de control directa que te permite

seleccionar distintas articulaciones del robot y moverlas a través de *joysticks.* Además, puede recibir y mostrar la imagen de la cámara frontal del robot.

2.4.2 NAO para iOS

- **iControlNao:** Esta aplicación (que no está actualizada para las últimas versiones del robot) permite enviarle al robot órdenes para que realice distintos comportamientos predefinidos como sentarse, levantarse y moverse. También nos permite introducir un texto para que sea reproducido. Ha sido desarrollada por Klauss Engel. En la Ilustración 21 podemos ver la interfaz de la aplicación junto con el robot NAO.

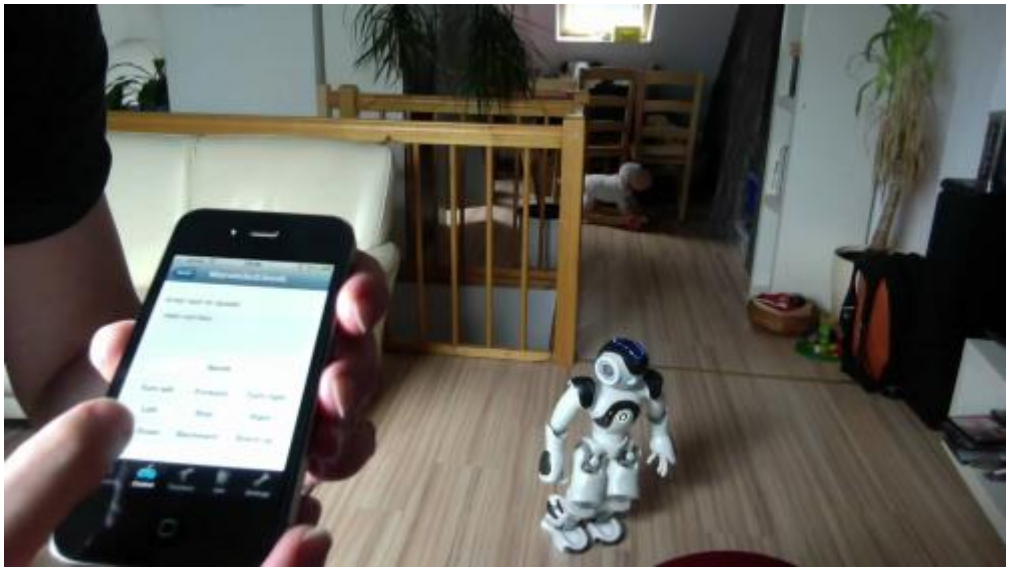

Ilustración 21 - Aplicación iControlNao

Capítulo 3: Descripción del sistema

A lo largo de este capítulo, describiremos en profundidad el análisis llevado a cabo para diseñar el sistema. Una vez presentado el análisis, se describirá el diseño del sistema, exponiendo de forma detallada su arquitectura mediante una descripción *Top-Bottom*. Partiendo de un diagrama estructural de alto nivel, terminando en un nivel de componentes que nos permita describir de forma detallada las funcionalidades del sistema.

3.1 Introducción

La finalidad del trabajo es la creación de una aplicación Android con la cual se pueda jugar con el robot NAO, para ello habrá que diseñar un sistema de teleoperación y un marco de juego.

El sistema de teleoperación forma parte de una aplicación interactiva con la que se podrán enviar al robot una serie de órdenes de bajo y alto nivel. Para poder jugar es necesario que la teleoperación se produzca en tiempo real, y teniendo en cuenta las características de los dispositivos móviles, el sistema contará con una **interfaz directa**. Las condiciones para la utilización de ésta interfaz son de alta calidad, porque el tiempo de retardo es mínimo; esto se debe a que la conexión con el robot se realiza a través de WiFi. El usuario podrá enviar acciones al robot utilizando distintas funcionalidades del dispositivo: (1) con la interfaz táctil puede hacer desplazarse al robot mediante unos controles de movimiento, hacerle hablar y que abra o cierre la mano derecha; (2) utilizándolo a modo de *joystick* puede controlar de manera precisa el movimiento de la cabeza y el brazo derecho. El operador recibirá realimentación visual de la imagen de la cámara frontal del robot que se mostrará en la pantalla del dispositivo, es el único tipo de realimentación, por parte del robot, que tiene; por lo que el nivel de telepresencia es **bajo**.

El marco de juego está compuesto por dos elementos: (1) un sistema de control del estado de la partida, que informe al jugador del tiempo de juego restante y de la puntuación obtenida mostrándolos en pantalla; y (2) un sistema que permita anunciar al jugador los objetivos de la partida y añadir nuevos retos a lo largo de la misma, actualizar la puntuación del jugador según se van consiguiendo los objetivos y, en partidas de dos jugadores, sincronizar los dispositivos de ambos.

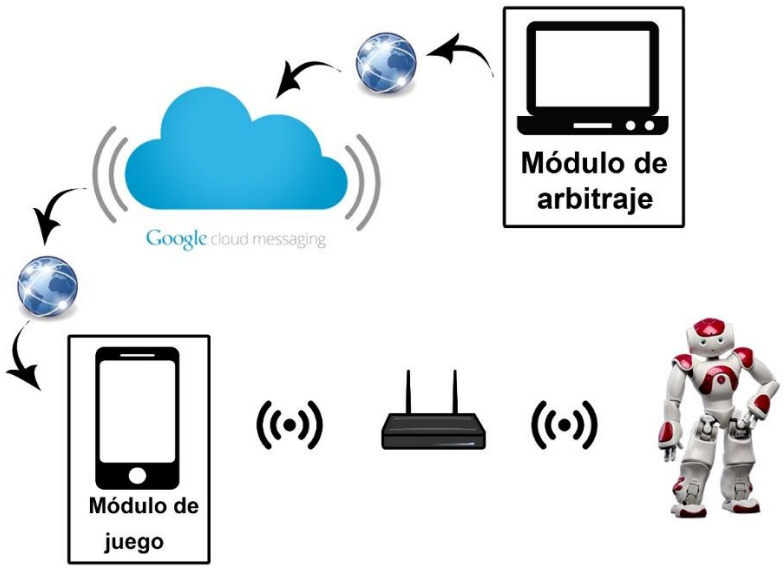

Ilustración 22 - Diagrama del framework

La ilustración 22 presenta una descripción de alto nivel de los diferentes dispositivos y aplicaciones que componen el sistema a desarrollar. El sistema de teleoperación y una parte de las funcionalidades del marco de juego están integrados en el módulo de juego, el resto de funcionalidades del marco de juego están integradas en el módulo de arbitraje. La comunicación entre ambos módulos se realiza a través del servicio *Google Cloud Messaging*. La combinación de los dos módulos conforma un *framework* de competición para jugar con el robot NAO.

3.2 Análisis del sistema

En esta sección se presenta el proceso de análisis realizado para identificar las características y requisitos del trabajo. Además, establecemos el marco impuesto por las restricciones de sistema y el entorno operacional.

3.2.1 Descripción de las características funcionales

El sistema desarrollado está formado por dos componentes, el módulo de juego y el de arbitraje. A continuación se describen las funcionalidades de cada uno de ellos:

El **módulo de juego** es una aplicación Java para Android. Éstas son sus funcionalidades:

- Proveer al usuario de una interfaz de configuración de la partida.
- Ser capaz de establecer una conexión con el robot.
- Proveer al usuario de una interfaz de juego.

- Ser capaz de recibir la imagen de la cámara del robot y mostrarla en pantalla.
- Permitir al usuario controlar al robot por medio de la pantalla táctil.
- Permitir al usuario controlar al robot utilizando el dispositivo como un *joystick*.
- Ser capaz de guardar los datos de las partidas jugadas por el usuario.
- Ser capaz de recibir los mensajes enviados desde el módulo de arbitraje.
- Contabilizar la duración de las partidas.
- Contabilizar los puntos obtenidos por el usuario.

El **módulo de arbitraje** es una aplicación Java. Éstas son sus funcionalidades:

- Ser capaz de enviar mensajes al módulo de juego.
- Proveer al usuario de una serie comandos con los que enviar los mensajes.

3.2.3 Restricciones del sistema

Restricciones *hardware*:

- El dispositivo móvil debe tener una tarjeta WiFi ya que es imprescindible para conectar con el robot.
- Debe ser táctil.
- Debe contar con un acelerómetro, que se utiliza para realizar algunos de los movimientos sobre el robot.
- Debe poder conectarse a Internet para recibir notificaciones *push* desde el servicio *Google Cloud Messaging*.
- El ordenador utilizado para ejecutar el módulo de arbitraje debe poder conectarse a Internet para enviar mensajes al servicio de *Google Cloud Messaging*, encargado de enviar notificaciones *push* a los dispositivos móviles.

Restricciones *software*:

- El SDK JNaoQi, necesario para comunicarse con el robot.
- Java es el lenguaje en el que se programan las aplicaciones Android.
- El dispositivo móvil utilizado debe utilizar Android como sistema operativo.
- La versión del sistema operativo debe ser igual o superior a Android 4.1 *JellyBean* para poder utilizar el servicio de *Google Cloud Messaging* y recibir *push*.

- El dispositivo móvil debe contar con la última versión del *API Google Services* para poder utilizar el servicio de *Google Cloud Messaging*.

3.2.4 Entorno operacional

En la Ilustración 23 podemos ver los componentes básicos del entorno operacional del sistema.

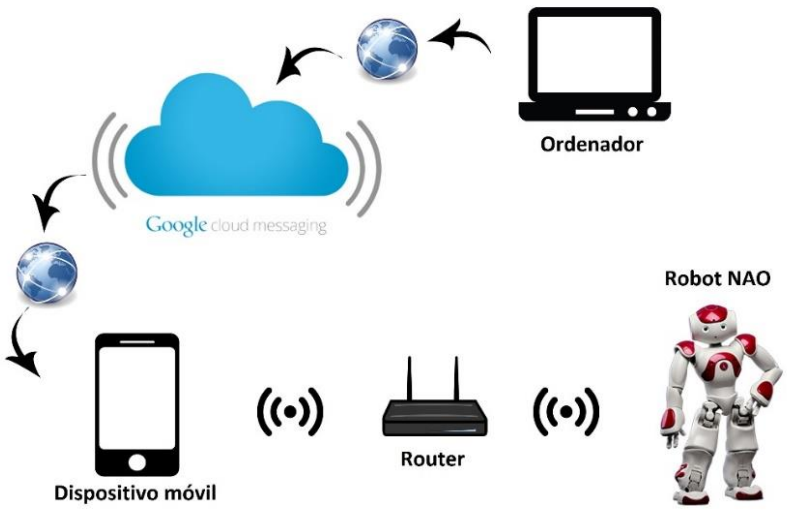

Ilustración 23 - Entorno operacional

3.2.4.1 Hardware

- **Robot NAO:** Robot humanoide fabricado por la empresa Aldebaran Robotics
- **Dispositivo móvil:** LG Nexus 5.
- **Ordenador**
- ***Router* inalámbrico**

3.2.4.2 Software

- **Android 6.0 *Marshmallow*:** Sistema operativo del dispositivo móvil.
- **SDK Android API 23 (Android 6.0):** Contiene el conjunto de herramientas de programación necesarias para desarrollar aplicaciones Android. Para poder usarlo es necesario tener instalado el SDK Java
- **JDK 1.7:** Contiene el conjunto de herramientas de programación necesarias para desarrollar aplicaciones Java.
- **SDK JNaoQi :** Librerías para java del *framework* NaoQi, contiene el conjunto de métodos necesarios para conectar y controlar al robot Nao mediante una aplicación java.
- **Android Studio 2.1:** es el entorno de programación para la plataforma Android.

- **Choreagraphe Suite 2.1.4:** Software de simulación y control del robot NAO.
- **Webots for NAO 8.4:** Entorno de simulación para robots.
- **Gradle 1.3.0:** Es un sistema de empaquetado de proyectos *software*. Es utilizado por *Android Studio* para construir y compilar los proyectos Android.

3.2.4.3 Comunicación entre dispositivos

- **Dispositivo móvil – Robot:** El dispositivo móvil y el robot se comunican a través de *WiFi*. Un *router* inalámbrico es el encargado de darnos esta funcionalidad.
- **Ordenador – Dispositivo móvil:** Los mensajes enviados desde el ordenador se mandan a través de Internet al servicio *Google Cloud Messaging*, este se encarga de enviar notificaciones *push* a los dispositivos indicados en el mensaje.

3.2.4.4 Robot Nao

El robot NAO es un robot humanoide desarrollado por Aldebaran Robotics. Se han fabricado distintas versiones a lo largo de los años, y cada una tiene modelos del robot que se diferencian en los grados de libertad con los que cuentan. Los grados de libertad o DOF (del inglés *Degrees Of Freedom*) hacen referencia al número de movimientos independientes que se pueden realizar. El robot utilizado para este trabajo es la última versión, V5, en particular el modelo H25, que cuenta 25 grado de libertad. Los motores y actuadores eléctricos situados en las distintas articulaciones del robot proveen al NAO de ese grado de libertad (Ilustración 24).

Además, cuenta con las siguientes características:

- Botón de encendido y apagado situado en el pecho
- Cuatro micrófonos para captar el sonido, situados en su cabeza y distribuidos de tal forma que puede detectar de qué dirección procede el sonido.
- Agrupaciones de *leds* distribuidos por su cuerpo y dos altavoces como dispositivos de comunicación.
- Distintos sensores: dos emisores y receptores sónar para detectar los obstáculos; un sensor inercial para estabilizarse y detectar una caída y dos sensores de presión situados en los pies para detectar colisiones.
- Una batería que le confiere una autonomía de unos 60 minutos de uso.
- Conectividad mediante Wifi y Ethernet.
- Un microprocesador ATOM 7530 de 1.6 GHz, 1GB de memoria RAM.

Dos cámaras de video VGA, con una resolución máxima de 1280x960 (Ilustraciones 25 y 26) y 30 fps. Tiene la capacidad de reconocer ciertos objetos como pelotas mediante el análisis de las imágenes de las cámaras.

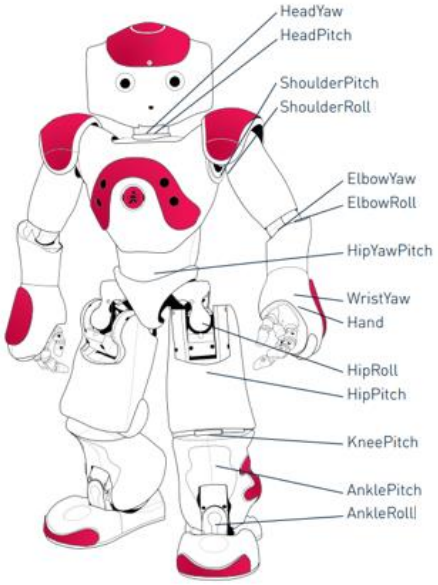

Ilustración 24 - Motores del NAO

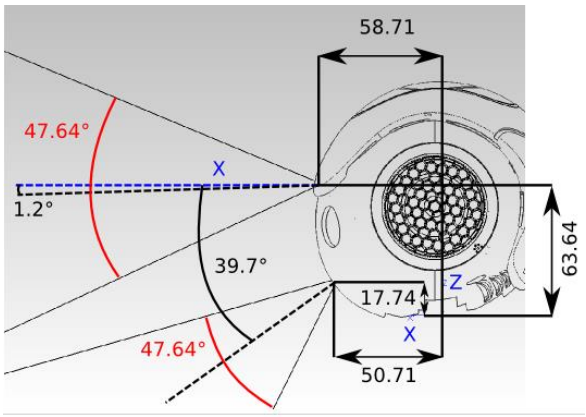

Ilustración 25 - Grados de visión vertical de las cámaras

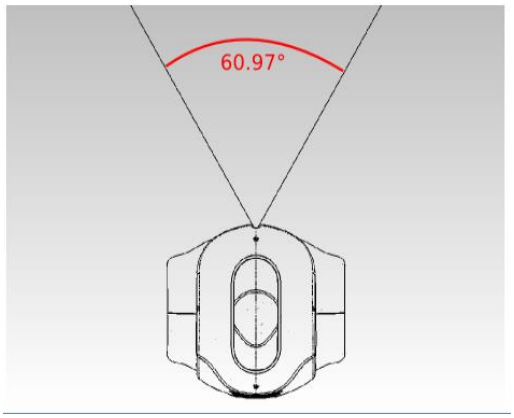

Ilustración 26 - Grados de visión horizontal de las cámaras

3.2.4.5 Android

Android es un sistema operativo basado en el *kernel* de Linux y, en un principio, diseñado para dispositivos con pantalla táctil como *smartphones* y *tablets*. La empresa Android Inc. lo empezó a desarrollar en 2003, pero en 2005 la empresa fue comprada por Google. La primera

versión fue lanzada al mercado en 2008 y actualmente Google acaba de presentar la versión 7.0. Con el tiempo, Android ha traspasado la barrera de los dispositivos móviles y es posible encontrarlo en televisores, neveras y coches. Se pasa a describir las funcionalidades de Android utilizadas en este trabajo:

- **Activity:** Son la representación visual e interactiva de una aplicación. Están formadas por dos partes: la parte lógica y la parte gráfica. La parte gráfica es un archivo XML, donde se diseña el aspecto de la interfaz de usuario, y la parte lógica es un archivo Java donde crea el código para manipular e interactuar con los elementos definidos en la parte gráfica. El ciclo de vida de una *Activity* se describe en la parte izquierda de la Ilustración (28).
- **Fragment:** Es una parte de la interfaz de usuario que puede añadirse o eliminarse de la interfaz de forma independiente al resto de elementos de un *Activity.* Esto permite dividir la interfaz en varias porciones de forma, que se puede diseñar diversas configuraciones de pantalla, dependiendo del tamaño y orientación, sin necesidad de duplicar código. El ciclo de vida de un *Fragment* se describe en la parte derecha de la Ilustración (28).

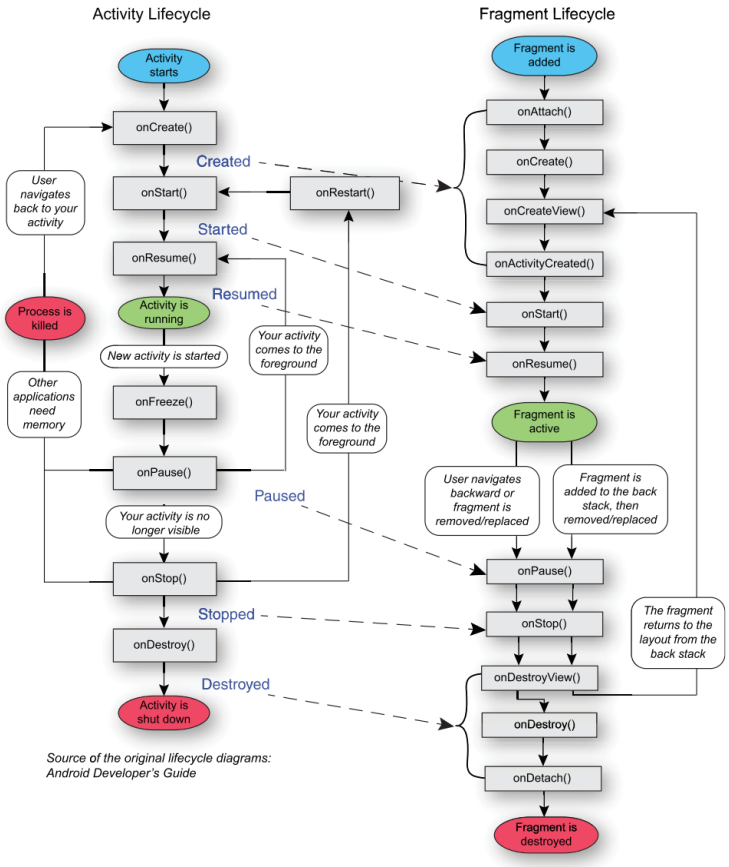

Ilustración 27 - Ciclo de vida de un Activity y un Fragment

- **Service:** Es una entidad que se ejecuta en segundo plano y con la que el usuario no interactúa. Son utilizados para realizar acciones de larga duración mientras las *activities* muestran información e interactúan con el usuario. Son de gran utilidad para gestionar tareas como sincronizar aplicaciones con servidores en internet, administrar las notificaciones del servicio GCM o monitorear información.
- **Acelerómetro:** Es el sensor que permite a Android detectar los movimientos del dispositivo. El acelerómetro genera eventos con los valores de aceleración registrados en cada uno de los ejes de coordenadas del dispositivo móvil (Ilustración 28). En este trabajo se utilizará para registrar los movimientos del usuario y transformarlos en los movimientos del robot.

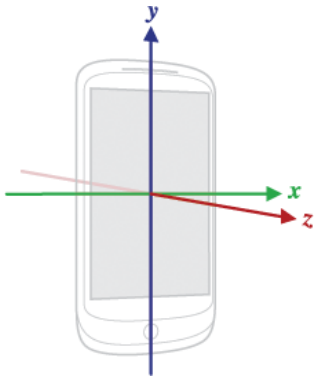

Ilustración 28 - Esquema de los ejes de coordenadas del dispositivo móvil

- **Servicio de mensajería GCM:** *Google Cloud Messaging* es un servicio gratuito desarrollado por Google que hace posible el envío de notificaciones desde un servidor a los dispositivos Android. Se puede destacar dos características:
    - No es necesario que la aplicación se encuentre ejecutándose para recibir notificaciones, el sistema despertará cuando se reciba la notificación.
    - GCM pasa la información directamente a la aplicación, tiene total control sobre ella.

  Para poder utilizar este servicio, es necesario registrar nuestra aplicación en los servidores del servicio GCM. Éste nos proporciona un *Server API Key* y un archivo de configuración. El archivo de configuración se añade al paquete de la aplicación móvil y será responsable de identificar al dispositivo. El *Server API Key* identificará los mensajes enviados desde el servidor. De esta manera el servicio de GCM sólo envía los mensajes a

los dispositivos que tenga un archivo de configuración relacionado con el *Server API Key* incluido en el mensaje.

- **SQLite:** Es un motor de bases de datos de código abierto que se caracteriza por su pequeño tamaño. A diferencia de otros gestores de bases de datos, SQLite tiene las siguientes ventajas:
    - No requiere el soporte de un servidor, implementa un conjunto de librerías que se encargan de la gestión.
    - No necesita configuración por lo que libera al programador de configuraciones de puertos, tamaños, ubicaciones, etc.
    - Usa un archivo para el esquema que le permite ahorrarse preocupaciones de seguridad; ya que los datos de las aplicaciones Android no pueden ser accedidos por contextos externos.

3.2.4.6 JNaoQi

NaoQi es el nombre *software* que ejecuta el robot y que lo controla. Para programar el NAO se utiliza el *NaoQi Framework.* Sus características principales son: (1) es multi-lenguaje, se pueden programar utilizando los lenguajes Python y C++; y (2) es multiplataforma, ya que se puede ejecutar en Windows, Linux o Mac. También ofrece una serie de *APIs* en otros lenguajes para poder utilizar las funciones de este *framework.* Ya que en este trabajo se desarrolla una aplicación Android que está basado en Java, utilizaremos la librería JNaoQi. La utilización de esta librería nos da las herramientas para lograr la teleoperación del robot. Se describen los servicios utilizados de esta librería:

- **Session:** Es el elemento clave para comunicarse con el robot. Gestiona la conexión con el robot y permite utilizar los servicios del robot a través de la red.
- **ALMotion:** Provee una serie de métodos para ayudarnos a mover al robot. Contiene comandos para controlar la rigidez de las articulaciones, para cambiar los ángulos de las articulaciones y funciones de alto nivel que nos permiten hacer andar al robot.
- **ALRobotPosture:** Se utiliza para cambiar la postura del robot.
- **ALTextToSpeech:** Permite hablar al robot.
- **ALMemory:** Da acceso a una memoria centralizada donde se guarda toda la información clave relativa a la configuración del *hardware* del robot. Más específicamente, nos da acceso al estado en que se encuentran los distintos sensores y actuadores del robot.

- **ALVideoDevice:** Es el encargado de proveer -de manera eficiente- las imágenes de los fuentes de video del robot.
- **ALTracker:** Permite al robot perseguir objetivos (pelotas, caras) con distintos medios (la cabeza o todo el cuerpo).

3.2.5 Especificación de casos de uso

Ilustración 29 - Diagrama de casos de uso

3.2.4.1 Descripción de los actores

Se puede observar en la Ilustración 29 que el sistema está compuesto por tres actores. El jugador y el robot están relacionados porque comparten varias acciones. Las acciones del árbitro añaden funcionalidad al sistema.

- Jugador: Este actor es el encargado de proveer la información al sistema.
- Robot: Este actor es el encargado de recoger la información enviada por el usuario y procesarla debidamente.
- Árbitro: Este actor es el encargado de establecer parámetros de juego en el sistema.

3.2.4.2 Descripción de los atributos de los casos de uso

Para la realización de la descripción textual de los distintos casos de uso, se han seleccionado una serie de atributos que describen cada uno de los casos de uso. A continuación se realiza una descripción del significado de cada uno de los atributos utilizados para la descripción de los casos de uso.

- Código: Identificación unívoca abreviada del caso de uso, se construye mediante CU seguido de un - y de tres dígitos. Por ejemplo CU-001.
- Nombre: Identificación extendida del caso de uso.
- Actores: Conjunto de entidades que interactúan con el caso de uso. El caso de uso representa una funcionalidad demandada por un actor.
- Descripción: Se realiza una descripción básica de la funcionalidad o funcionalidades del caso de uso.
- Precondiciones y poscondiciones: Se realiza una descripción de las condiciones que deben cumplirse para poder realizar una operación, y el estado en el que queda el sistema tras realizar una operación.
- Escenario: Se realiza una descripción básica de las acciones que se ejecutarán paso a paso en el caso de uso.

3.2.4.3 Descripción textual de los casos de uso

En este apartado se expone el análisis de cada uno de los casos de uso que aparecen en la Ilustración 29.

Código	**CU-001-1**
Nombre	Iniciar partida (modo individual).
Actores	Jugador.
Descripción	El jugador pulsa el botón de jugar en la interfaz de conexión.
Precondiciones	Se han configurado los parámetros de juego (CU-002), el modo de juego seleccionado es individual.
Poscondiciones	Se conecta con el robot (CU-003).
Escenario	1. El módulo de juego muestra la interfaz de conexión. 2. El jugador pulsa el botón "Play!". 3. La aplicación comprueba que se han configurado todos los parámetros de juego. 4. Se inicia el proceso de conexión con el robot.

Tabla 1– Caso de uso CU-001-1

Código	**CU-001-2**
Nombre	Iniciar partida (modo versus) .
Actores	Jugador, árbitro.
Descripción	El jugador pulsa el botón de jugar en la interfaz de conexión, se muestra por pantalla un icono de carga a la espera de la recepción del mensaje de inicio de juego por parte del árbitro (CU-004).
Precondiciones	Se han configurado los parámetros de juego (CU-002), el modo de juego seleccionado es versus.
Poscondiciones	En el momento que se reciba el mensaje de inicio de juego por parte del árbitro (CU-004), se conecta con el robot (CU-003) .
Escenario	1. El módulo de juego muestra la interfaz de conexión. 2. El jugador pulsa el botón "Play!". 3. La aplicación comprueba que se han configurado todos los parámetros de juego. 4. Se muestra una pantalla de carga a la espera de recibir el mensaje de comienzo de partida por parte del módulo de arbitraje. 5. Al recibir el mensaje, se inicia la conexión con el robot.

Tabla 2 – Caso de uso CU-001-2

Código	**CU-002**
Nombre	Configurar parámetros de juego.
Actores	Jugador.
Descripción	El jugador configura los parámetros obligatorios (nombre de usuario e IP del robot) en la interfaz de conexión para poder iniciar la partida.
Precondiciones	-
Poscondiciones	Se han rellenado correctamente los parámetros obligatorios.
Escenario	1. El módulo de juego muestra la interfaz de conexión. 2. El jugador rellena por medio del teclado los campos obligatorios (nombre e IP del robot), selecciona el tiempo de partida y el modo de juego. 3. Si el modo elegido es versus, el jugador rellena el campo obligatorio del nombre de su rival.

Tabla 3 – Caso de uso CU-002

Código	**CU-003**
Nombre	Conectar con el robot.
Actores	Jugador.
Descripción	Conectar el módulo de juego al robot para poder enviarle instrucciones para jugar la partida.
Precondiciones	La IP del robot introducida en la configuración de parámetros (CU-002) es correcta y el robot está preparado para realizar la conexión.
Poscondiciones	Conexión establecida con el robot, en el dispositivo se muestra la interfaz de juego.
Escenario	1. Se inicia la conexión con el robot desde la interfaz de conexión. 2. Si hay un error de conexión se muestra un mensaje. 3. Si la conexión se realiza correctamente, se muestra en pantalla la interfaz de juego.

Tabla 4 – Caso de uso CU-003

Código	**CU-004**
Nombre	Inicio de juego para dos jugadores.
Actores	Árbitro.
Descripción	El árbitro envía un mensaje de inicio de partida a los dispositivos móviles configurados en el módulo de arbitraje.
Precondiciones	En el módulo de arbitraje se han configurado los dispositivos móviles a los que llegarán los mensajes.
Poscondiciones	El mensaje se envía correctamente.
Escenario	1. El árbitro ejecuta el comando *"start"* en el módulo de arbitraje. 2. El módulo de arbitraje envía el mensaje al servicio *Google Cloud Messaging*.

Tabla 5 – Caso de uso CU-004

Código	**CU-005**
Nombre	Ver máximas puntuaciones.
Actores	Jugador.
Descripción	El jugador pulsa el botón de máximas puntuaciones.
Precondiciones	-
Poscondiciones	Se muestra una tabla una lista con las máximas puntuaciones guardadas en el dispositivo.
Escenario	1. El jugador pulsa el botón "*scores*" en las interfaces de conexión o juego. 2. La aplicación accede a la base de datos y recoge la lista de puntuaciones. 3. Se muestra una tabla con la lista de máximas puntuaciones.

Tabla 6 – Caso de uso CU-005

Código	**CU-006**
Nombre	Enviar imagen.
Actores	Robot.
Descripción	El robot envía fotogramas de la imagen que capta por su cámara frontal al módulo de juego.
Precondiciones	Partida iniciada y establecida conexión con el robot.
Poscondiciones	La imagen se envía correctamente.
Escenario	1. El módulo de juego envía una petición de imagen de la cámara frontal al robot cada 20 milisegundos. 2. El robot responde a la petición de imagen enviando un fotograma de su visión en la cámara frontal. 3. Cada vez que se recibe un fotograma en el módulo de juego se refresca la imagen en la interfaz de juego.

Tabla 7 – Caso de uso CU-006

Código	**CU-007**
Nombre	Mover piernas.
Actores	Jugador, Robot.
Descripción	El jugador pulsa los botones de desplazamiento en la interfaz de juego para que se envíe la acción de andar o girar al robot y que este realice la acción indicada.
Precondiciones	Partida iniciada y establecida conexión con el robot.
Poscondiciones	El robot se desplazará o girará en la dirección indicada.
Escenario	1. Se muestra la interfaz de juego. 2. El jugador pulsa cualquiera de los cuatro botones de desplazamiento. 3. El módulo de juego envía la petición de movimiento al robot. 4. El robot recibe la petición y empieza a andar en la dirección indicada.

Tabla 8 – Caso de uso CU-007

Código	**CU-008**
Nombre	Parar movimiento.
Actores	Jugador, robot.
Descripción	El jugador pulsa el botón de parar movimiento en la interfaz de juego para que se envíe la acción de parar al robot y que este realice la acción indicada. .
Precondiciones	Partida iniciada y establecida conexión con el robot.
Poscondiciones	El robot parará y dejará de moverse.
Escenario	1. Se muestra la interfaz de juego. 2. El jugador pulsa el botón de "*stop*". 3. El módulo de juego envía la petición al robot. 4. El robot recibe la petición y deja de realizar cualquier movimiento.

Tabla 9 – Caso de uso CU-008

Código	**CU-009**
Nombre	Mover cabeza.
Actores	Jugador, robot.
Descripción	El jugador pulsa el botón de mover cabeza en la interfaz de juego y usará el dispositivo a modo de *joystick* para enviar desplazamientos a las articulaciones del cuello del robot y este realizará los movimientos.
Precondiciones	Partida iniciada y establecida conexión con el robot.
Poscondiciones	El robot cambiará la posición de las articulaciones del cuello para mover la cabeza a la posición indicada.
Escenario	1. Se muestra la interfaz de juego. 2. El jugador pulsa el botón "cabeza" y este se activa. 3. El usuario utiliza el dispositivo como si fuera un *joystick* moviendo hacia los lados o arriba y abajo. 4. El módulo de juego capta los datos recibidos del acelerómetro y los convierte en movimientos que pueda interpretar el robot. 5. Se envía la lista de movimientos al robot. 6. El robot recibe la lista de movimientos y mueve la articulación del cuello de acuerdo con la lista recibida.

Tabla 10 – Caso de uso CU-009

Código	**CU-010**
Nombre	Mover brazo.
Actores	Jugador, robot.
Descripción	El jugador pulsa el botón de mover cabeza en la interfaz de juego y usará el dispositivo a modo de *joystick* para enviar desplazamientos a las articulaciones del hombro del robot y este realizará los movimientos.
Precondiciones	Partida iniciada y establecida conexión con el robot.
Poscondiciones	El robot cambiará la posición de las articulaciones del hombro para mover el brazo a la posición indicada.
Escenario	1. Se muestra la interfaz de juego. 2. El jugador pulsa el botón "brazo" y este se activa. 3. El usuario utiliza el dispositivo como si fuera un *joystick* moviendo hacia los lados o arriba y abajo. 4. El módulo de juego capta los datos recibidos del acelerómetro y los convierte en movimientos que pueda interpretar el robot. 5. Se envía la lista de movimientos al robot. 6. El robot recibe la lista de movimientos y mueve la articulación del brazo de acuerdo con la lista recibida.

Tabla 11 – Caso de uso CU-010

Código	**CU-011**
Nombre	Abrir/cerrar mano.
Actores	Jugador, robot.
Descripción	El jugador pulsa el botón de abrir/cerrar mano en la interfaz de juego para que se le envíe la acción de abrir/cerrar mano (dependiendo del estado en que estuviera de la mano de robot) al robot y este realice la acción indicada.
Precondiciones	Partida iniciada y establecida conexión con el robot.
Poscondiciones	El robot cambiará el estado de su mano.
Escenario	1. Se muestra la interfaz de juego. 2. El jugador pulsa el botón "mano". 3. El módulo de juego envía la petición al robot. 4. El robot recibe la petición y abre o cierra la mano dependiendo del estado en que estuviera.

Tabla 12 – Caso de uso CU-011

Código	**CU-012**
Nombre	Hablar.
Actores	Jugador, robot.
Descripción	El jugador pulsa el botón de hablar en la interfaz de juego y se abre un diálogo de texto donde introducir lo que quiera que diga el robot para que se le envíe la acción al robot y este realizará la acción indicada.
Precondiciones	Partida iniciada y establecida conexión con el robot.
Poscondiciones	El robot dirá el texto introducido por el usuario.
Escenario	1. Se muestra la interfaz de juego. 2. El jugador pulsa el botón “hablar”. 3. La interfaz de juego muestra un diálogo de texto. 4. El jugador introduce el texto. 5. El jugador pulsa el botón “aceptar” del diálogo de texto. 6. El módulo de juego envía la petición al robot. 7. El robot recibe la petición y reproduce el texto introducido por el usuario.

Tabla 13 – Caso de uso CU-012

Código	**CU-013**
Nombre	Detectar y perseguir pelota roja.
Actores	Jugador, robot.
Descripción	El jugador pulsa el botón de detectar pelota roja para que se le envíe la acción al robot y este realizará el comportamiento enviado.
Precondiciones	Partida iniciada y establecida conexión con el robot.
Poscondiciones	Si hay una pelota roja dentro del campo visual del robot, éste la reconocerá y se acercará hasta ella.
Escenario	1. Se muestra la interfaz de juego. 2. El jugador pulsa el botón “pelota roja”. 3. El módulo de juego envía la petición al robot. 4. El robot recibe la petición. 5. El robot activa un comportamiento mediante el cual podrá detectar una pelota roja si está dentro de su campo de visión.

Tabla 14 – Caso de uso CU-013

Código	**CU-014**
Nombre	Sentar.
Actores	Jugador, Robot.
Descripción	El jugador pulsa los botones el botón de sentar robot en la interfaz de juego para que se envíe la acción de sentarse al robot y que este realice la acción indicada.
Precondiciones	Partida iniciada y establecida conexión con el robot.
Poscondiciones	El robot se sentará.
Escenario	1. Se muestra la interfaz de juego. 2. El jugador pulsa el botón de sentarse. 3. El módulo de juego envía la petición de movimiento al robot. 4. El robot recibe la petición y se sienta.

Tabla 15 – Caso de uso CU-014

Código	**CU-015**
Nombre	Poner de pie.
Actores	Jugador, Robot.
Descripción	El jugador pulsa los botones el botón de poner de pie en la interfaz de juego para que se envíe la acción de sentarse al robot y que este realice la acción indicada.
Precondiciones	Partida iniciada y establecida conexión con el robot.
Poscondiciones	El robot se pondrá de pie.
Escenario	1. Se muestra la interfaz de juego. 2. El jugador pulsa el botón de poner de pie. 3. El módulo de juego envía la petición de movimiento al robot. 4. El robot recibe la petición y se pone de pie.

Tabla 16 – Caso de uso CU-015

Código	**CU-016**
Nombre	Abandonar partida.
Actores	Jugador.
Descripción	Se pulsa el botón de abandonar partida y la aplicación muestra la interfaz de fin de partida.
Precondiciones	Partida iniciada y establecida conexión con el robot.
Poscondiciones	Se para el movimiento del robot y se cierra la conexión con él. .
Escenario	1. Se muestra la interfaz de juego. 2. El jugador pulsa en el menú desplegable. 3. El jugador pulsa el botón “abandonar partida”. 4. Se muestra la interfaz de fin de juego.

Tabla 17 – Caso de uso CU-016

Código	**CU-017**
Nombre	Actualización de marcador.
Actores	Árbitro.
Descripción	El árbitro envía un mensaje de actualización de marcador con el resultado actual de la partida. .
Precondiciones	En el módulo de arbitraje se han configurado los dispositivos móviles a los que llegarán los mensajes.
Poscondiciones	El mensaje se envía correctamente.
Escenario	1. El árbitro ejecuta el comando *"score"* en el módulo de arbitraje. 2. El módulo de arbitraje envía el mensaje al servicio *Google Cloud Messaging*.

Tabla 18 – Caso de uso CU-017

Código	**CU-018**
Nombre	Enviar reto.
Actores	Árbitro.
Descripción	El árbitro envía un mensaje de reto a los dispositivos móviles configurados en el módulo de arbitraje.
Precondiciones	En el módulo de arbitraje se han configurado los dispositivos móviles a los que llegarán los mensajes.
Poscondiciones	El mensaje se envía correctamente.
Escenario	1. El árbitro ejecuta el comando *"challenge"* en el módulo de arbitraje. 2. El módulo de arbitraje envía el mensaje al servicio *Google Cloud Messaging*.

Tabla 19 – Caso de uso CU-018

3.2.5 Especificación de requisitos

En este apartado se presenta de especificación de requisitos, funcionales y no funcionales del sistema.

3.2.5.1 Descripción de los atributos de los requisitos

Para la realización de la descripción textual de los distintos requisitos que han sido identificados, se han seleccionado una serie de atributos que describen cada uno de los requisitos. A continuación se realiza una descripción del significado de cada uno de los atributos utilizados para su descripción:

- Código: Identificación unívoca abreviada del requisito, se construye mediante el código del requisito seguido de un - y de tres dígitos. Los requisitos serán divididos en funciones y no

funcionales y sus códigos son RF para los requisitos funciones y RNF para los requisitos no funcionales. Por ejemplo RF-001.

- Descripción: Se realiza una descripción básica del requisito que ha sido identificado.
- Fuente: Indica a través de qué fuente ha sido identificado el requisito. Normalmente, este valor se corresponderá con uno o varios códigos de los casos de uso.
- Necesidad: Determina el grado de implementación del requisito. Los valores que puede tomar este atributo son los siguientes:
    - Esencial: El requisito tiene que ser implementado.
    - Deseable: Es preferible implementar el requisito, pero no es obligatorio.
    - Opcional: El requisito se podrá implementar, pero no es importante ni obligatorio.
- Prioridad: Define la importancia del requisito, de forma que permita definir el orden en el cual será incluido en el proceso de diseño y el orden de implementación. Los valores que puede tomar este atributo son los siguientes:
    - Alta: El requisito debe ser implementado en las fases iniciales del desarrollo.
    - Media: El requisito debe ser implementado una vez que hayan sido implementados los requisitos de prioridad alta.
    - Baja: El requisito debe ser implementados en las fases finales del desarrollo. Estos requisitos no influyen en el correcto funcionamiento del sistema.
- Estabilidad: Define la estabilidad del requisitos durante la vida útil del *software*. Esto implica si el requisito podrá ser o no modificado durante el ciclo de vida. Los valores que puede tomar este atributo son los siguientes:
    - Estable: El requisito no puede variar durante el ciclo de vida del sistema.
    - Inestable: El requisito puede variar a lo largo de la ciclo de vida del sistema.
- Verificabilidad: Define el grado de verificabilidad de un requisito, es decir indica en qué grado es posible comprobar que el requisito se ha incorporado en el sistema desarrollado. Los valores que puede tomar este atributo son los siguientes:
    - Alta: Se puede verificar que el requisito ha sido implementado en el sistema. Este tipo de requisitos se corresponden con las funcionalidades básicas del sistema.

- Media: Se puede verificar que el requisito ha sido implementado en el sistema. Pero requiere de una comprobación compleja o del código fuente del sistema.
- Baja: Es difícil verificar si el requisito ha sido implementado en el sistema o en algunos casos no es posible.

3.2.5.2 Requisitos funcionales

En este apartado se presentan los requisitos funcionales del sistema desarrollado.

Código	**RF-001**	**Fuente**	Jugador
Descripción	Para acceder a la aplicación se hará a través de un icono en la lista de aplicaciones del dispositivo.		
Necesidad	Alta	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 20 – Requisito funcional RF-001

Código	**RF-002**	**Fuente**	Jugador
Descripción	Al entrar a la aplicación se debe presentar una interfaz de conexión donde configurar los parámetros de conexión y los parámetros de la partida.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 21 – Requisito funcional RF-002

Código	**RF-003**	**Fuente**	Jugador
Descripción	En la interfaz de conexión se debe mostrar un campo editable de texto obligatorio para introducir el nombre de usuario.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 22 – Requisito funcional RF-003

Código	**RF-004**	**Fuente**	Jugador
Descripción	En la interfaz de conexión se debe mostrar un campo obligatorio de texto editable para introducir la ip del robot.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 23 – Requisito funcional RF-004

Código	**RF-005**	**Fuente**	Jugador
Descripción	En la interfaz de conexión se debe mostrar un desplegable donde seleccionar el tiempo de duración de la partida.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 24 – Requisito funcional RF-005

Código	**RF-006**	**Fuente**	Jugador
Descripción	En la interfaz de conexión se debe mostrar un botón para acceder a la pantalla de máximas puntuaciones.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 25 – Requisito funcional RF-006

Código	**RF-007**	**Fuente**	Jugador
Descripción	En la interfaz de conexión se debe mostrar se debe mostrar un botón para seleccionar entre el modo individual y el modo versus.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 26 – Requisito funcional RF-007

Código	**RF-008**	**Fuente**	Jugador
Descripción	En la interfaz de conexión se debe mostrar se debe mostrar un campo obligatorio de texto editable para introducir el nombre del rival al seleccionar el modo versus.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 27 – Requisito funcional RF-008

Código	**RF-009**	**Fuente**	Jugador
Descripción	En la interfaz de conexión se debe mostrar se debe mostrar un botón de juego para iniciar la partida.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 28 – Requisito funcional RF-009

Código	**RF-010**	**Fuente**	Jugador
Descripción	Al pulsar el botón de juego se comprobará que todos los campos obligatorios han sido rellenados, mostrando un mensaje de error indicando que falta por rellenar.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 29 – Requisito funcional RF-010

Código	**RF-011**	**Fuente**	Robot
Descripción	Se debe mostrar un mensaje de conexión fallida si no se ha podido conectar con el robot al pulsar el botón de juego.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 30 – Requisito funcional RF-011

Código	**RF-012**	**Fuente**	Jugador
Descripción	La aplicación de contar con una base de datos donde almacenar las puntuaciones obtenidas en las partidas.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 31 – Requisito funcional RF-012

Código	**RF-013**	**Fuente**	Jugador, Robot
Descripción	Después de conectarse con el robot, la aplicación debe mostrar una interfaz de juego donde se presentan las opciones para controlar al robot.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 32 – Requisito funcional RF-013

Código	**RF-014**	**Fuente**	Jugador
Descripción	En la interfaz de juego se debe mostrar un contador de tiempo con cuenta regresiva en el que aparece el tiempo que queda de partida.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 33 – Requisito funcional RF-014

Código	**RF-015**	**Fuente**	Jugador
Descripción	En la interfaz de juego se debe mostrar un marcador con el nombre del jugador y los puntos que lleva en la partida. Si el modo de juego es versus, aparecerá también el nombre y los puntos de su rival.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 34 – Requisito funcional RF-015

Código	**RF-016**	**Fuente**	Robot
Descripción	En la interfaz de juego se debe mostrar la imagen recibida de la cámara frontal del robot.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 35 – Requisito funcional RF-016

Código	**RF-017**	**Fuente**	Jugador, Robot
Descripción	En la interfaz de juego se debe mostrar se debe mostrar un botón de avance que hace que el robot ande hacia adelante.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 36 – Requisito funcional RF-017

Código	**RF-018**	**Fuente**	Jugador, Robot
Descripción	En la interfaz de juego se debe mostrar un botón de retroceso que hace que el robot ande hacia atrás.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 37 – Requisito funcional RF-018

Código	**RF-019**	**Fuente**	Jugador, Robot
Descripción	En la interfaz de juego se debe mostrar un botón de giro hacia la izquierda que hace que el robot gire sobre sí mismo hacia la izquierda.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 38 – Requisito funcional RF-019

Código	**RF-020**	**Fuente**	Jugador, Robot
Descripción	En la interfaz de juego se debe mostrar un botón de giro hacia la derecha que hace que el robot gire sobre sí mismo hacia la derecha.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 39 – Requisito funcional RF-020

Código	**RF-021**	**Fuente**	Jugador, Robot
Descripción	En la interfaz de juego se debe mostrar un botón de parada que hace el robot pare de realizar cualquier movimiento.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 40 – Requisito funcional RF-021

Código	**RF-022**	**Fuente**	Jugador, Robot
Descripción	En la interfaz de juego se debe mostrar un botón de selección de brazo. Al pulsarlo se el robot moverá su brazo derecho según se mueva el dispositivo móvil.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 41 – Requisito funcional RF-022

Código	**RF-023**	**Fuente**	Jugador, Robot
Descripción	En la interfaz de juego se debe mostrar un botón de selección de cabeza. Al pulsarlo se el robot moverá su cabeza derecho según se mueva el dispositivo móvil.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 42 – Requisito funcional RF-023

Código	**RF-024**	**Fuente**	Jugador, Robot
Descripción	Al seleccionar los botones de movimiento de brazo o cabeza, se utilizará el acelerómetro para captar los movimientos realizados con el dispositivo móvil y transformarlos en movimientos del robot.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 43 – Requisito funcional RF-024

Código	**RF-025**	**Fuente**	Jugador, Robot
Descripción	En la interfaz de juego se debe mostrar un botón para abrir y cerrar la mano derecha. El robot abrirá o cerrará la mano al pulsarlo.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 44 – Requisito funcional RF-025

Código	**RF-026**	**Fuente**	Jugador, Robot
Descripción	En la interfaz de juego se debe mostrar un botón para hacer al hablar al robot.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 45 – Requisito funcional RF-026

Código	**RF-027**	**Fuente**	Jugador, Robot
Descripción	Al pulsar el botón de hablar se abrirá un diálogo de texto donde introducir el texto que reproducirá el robot. El diálogo tendrá dos botones, uno para descartar el mensaje y otro para que el robot reproduzca el texto introducido.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 46 – Requisito funcional RF-027

Código	**RF-028**	**Fuente**	Jugador, Robot
Descripción	En la interfaz de juego se debe mostrar un botón de seguimiento de pelota roja. Al pulsarlo el robot podrá reconocer y seguir una pelota roja que se encuentre dentro de su campo de visión.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 47 – Requisito funcional RF-028

Código	**RF-029**	**Fuente**	Jugador
Descripción	En la interfaz de juego se debe mostrar un desplegable con dos opciones: reconectar y abandonar partida.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 48 – Requisito funcional RF-029

Código	**RF-030**	**Fuente**	Jugador, Robot
Descripción	Al pulsar el botón reconectar en el menú desplegable se conectará con el robot si se ha perdido la conexión.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 49 – Requisito funcional RF-030

Código	**RF-031**	**Fuente**	Jugador, Robot
Descripción	Si se pierde la conexión, se mostrará un mensaje avisando de ello al pulsar cualquier botón de la interfaz de juego.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 50 – Requisito funcional RF-031

Código	**RF-032**	**Fuente**	Jugador, Robot
Descripción	Se mostrará un mensaje de error de conexión si se pierde la conexión y no es posible volver a conectar desde la interfaz de juego.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 51 – Requisito funcional RF-032

Código	**RF-033**	**Fuente**	Jugador
Descripción	Al abandonar la partida desde el menú desplegable se mostrará una pantalla de fin de juego.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 52 – Requisito funcional RF-033

Código	**RF-034**	**Fuente**	Jugador
Descripción	Cuando el contador de tiempo llegue a 0 se mostrará la pantalla de fin de juego. Esta pantalla mostrará el resultado obtenido en la partida.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 53 – Requisito funcional RF-034

Código	**RF-035**	**Fuente**	Jugador
Descripción	Se debe implementar un sistema para recibir notificaciones desde *Google Cloud messaging*.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 54 – Requisito funcional RF-035

Código	**RF-036**	**Fuente**	Árbitro
Descripción	Se debe actualizar el marcador con los datos recibidos al recibir una notificación de actualización de marcador.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 55 – Requisito funcional RF-036

Código	**RF-037**	**Fuente**	Árbitro
Descripción	Se debe mostrar un texto en pantalla con los datos recibidos al recibir una notificación de reto.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 56 – Requisito funcional RF-037

Código	**RF-038**	**Fuente**	Jugador
Descripción	En la interfaz de conexión se debe mostrar un símbolo de carga al configurar el modo versus y pulsar el botón de juego.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 57 – Requisito funcional RF-038

Código	**RF-039**	**Fuente**	Árbitro
Descripción	Se debe empezar la partida en el modo versus al recibir una notificación de comienzo de partida.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 58 – Requisito funcional RF-039

Código	**RF-040**	**Fuente**	Árbitro
Descripción	El módulo de arbitraje debe poder enviar mensajes al servicio de *Google Cloud Messaging*.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 59 – Requisito funcional RF-040

Código	**RF-041**	**Fuente**	Árbitro
Descripción	El módulo de arbitraje de poder enviar un mensaje de comienzo de partida.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 60 – Requisito funcional RF-041

Código	**RF-042**	**Fuente**	Árbitro
Descripción	El módulo de arbitraje de poder enviar un mensaje de actualización de marcador. Se podrán incluir uno o dos parámetros, dependiendo si la partida es en modo individual o modo versus, con las nuevas puntuaciones.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 61 – Requisito funcional RF-042

Código	**RF-043**	**Fuente**	Árbitro
Descripción	El módulo de arbitraje de poder enviar un mensaje de reto. Se podrá incluir como parámetro un texto con la descripción del reto.		
Necesidad	Esencial	**Prioridad**	Media
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 62 – Requisito funcional RF-043

3.2.5.3 Requisitos no funcionales

En este apartado se presentan los requisitos no funcionales del sistema desarrollado.

Código	**RNF-001**	**Fuente**	Módulo de juego
Descripción	Utilizar el robot NAO.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 63 – Requisito no funcional RNF-001

Código	**RNF-002**	**Fuente**	Módulo de juego
Descripción	La aplicación debe ser desarrollada en Android.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 64 – Requisito no funcional RNF-002

Código	**RNF-003**	**Fuente**	Módulo de juego
Descripción	La aplicación debe ser compatible con la versión Android 6.0 *Marshmallow*.		
Necesidad	Deseable	**Prioridad**	Alta
Estabilidad	Inestable	**Verificabilidad**	Alta

Tabla 65 – Requisito no funcional RNF-003

Código	**RNF-004**	**Fuente**	Módulo de juego
Descripción	El dispositivo móvil utilizado debe disponer de conexión WiFi.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 66 – Requisito no funcional RNF-004

Código	**RNF-005**	**Fuente**	Módulo de juego
Descripción	El dispositivo móvil utilizado debe tener conexión a Internet.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 67 – Requisito no funcional RNF-005

Código	**RNF-006**	**Fuente**	Módulo de juego
Descripción	El dispositivo móvil utilizado debe tener una pantalla táctil.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 68 – Requisito no funcional RNF-006

Código	**RNF-007**	**Fuente**	Módulo de juego
Descripción	El dispositivo móvil utilizado debe disponer de acelerómetro.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 69 – Requisito no funcional RNF-007

Código	**RNF-008**	**Fuente**	Módulo de juego
Descripción	La aplicación Android desarrollado debe ser multitarea.		
Necesidad	Esencial	**Prioridad**	Alta
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 70 – Requisito no funcional RNF-008

Código	**RNF-009**	**Fuente**	Módulo de arbitraje
Descripción	El módulo de arbitraje de ser desarrollado en java.		
Necesidad	Deseable	**Prioridad**	Media
Estabilidad	Inestable	**Verificabilidad**	Media

Tabla 71 – Requisito no funcional RNF-009

3.3 Diseño del sistema

En esta sección se presenta el diseño de la arquitectura del sistema, la sección tiene la siguiente estructura:

- En el apartado 3.3.1 se presenta el esquema de la arquitectura del sistema y se hace una introducción de los componentes que lo forman.
- En el apartado 3.3.2 se describe de forma general el sistema, para ello se presentan las distintas pantallas del módulo de juego y sus diagramas de flujo.
- En el apartado 3.3.3 se desgranan cada uno de los componentes que forman el sistema, explicando su funcionalidad e implementación.

3.3.1 Arquitectura del sistema

A continuación se va a describir de manera general el diseño que tiene la arquitectura global del sistema creado (Ilustración 30).

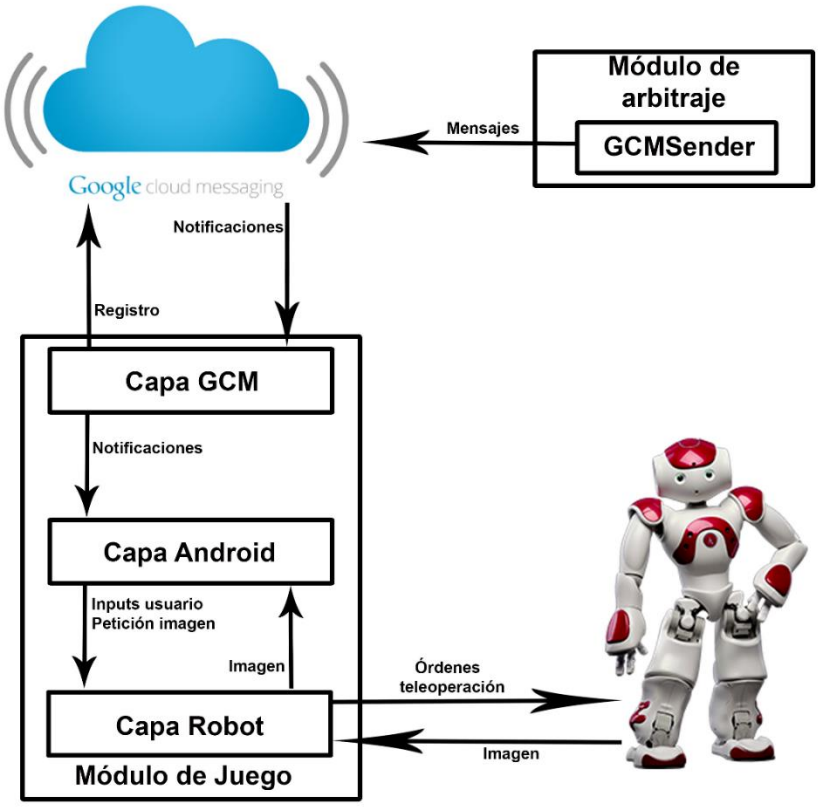

Ilustración 30 - Arquitectura del sistema

Módulo de juego

El módulo de juego está dividido en tres capas, cada una de ella nos proporciona unas funcionalidades distintas:

- **Capa Android:** Es la capa central de la aplicación que es la que interactúa con las otras dos capas de la aplicación. Esta capa se encarga de proporcionar las interfaces con la que el usuario interactúa (*Activities*) y una lógica para gestionar sus acciones. Además, controla el ciclo de vida de la aplicación y los datos de entrada recogidos por los sensores del dispositivo.
- **Capa Robot:** Es la capa encargada de interactuar con el robot. Contiene los mecanismos necesarios para conectarse y controlar al robot, así como la petición y recepción de imágenes. También es la encargada de almacenar los datos de la partida.
- **Capa GCM:** En esta capa encontramos toda la lógica relacionada con el servicio de *Google Cloud Messaging*. Se encarga de registrar al dispositivo y escuchar los eventos de notificación recibidos desde este servicio.

Módulo de arbitraje

El módulo de arbitraje es una aplicación que nos permite enviar varios tipos de mensajes al módulo de juego.

- **GCMSender:** Provee al árbitro una serie de comandos que aceptan unos parámetros con los que generar acciones para interactuar con el módulo de juego. Es el encargado de formatear en un JSON los datos introducidos por el árbitro, para enviarlos al servicio de *Google Cloud Messaging*; que es el que distribuye las notificaciones *push* a los dispositivos indicados por GCMSender.

3.3.2 Descripción general del sistema

En este apartado describiremos el funcionamiento de la aplicación, indicando el ciclo de vida de la misma. Para simplificar el diagrama de flujo, se ha divido en cuatro partes: el diagrama de flujo general de la aplicación; la segunda y la tercera son los diagramas de flujo de la fase de juego y la fase de fin de juego; por último, el diagrama del servicio de mensajería.

3.3.2.1 Diagrama de flujo general

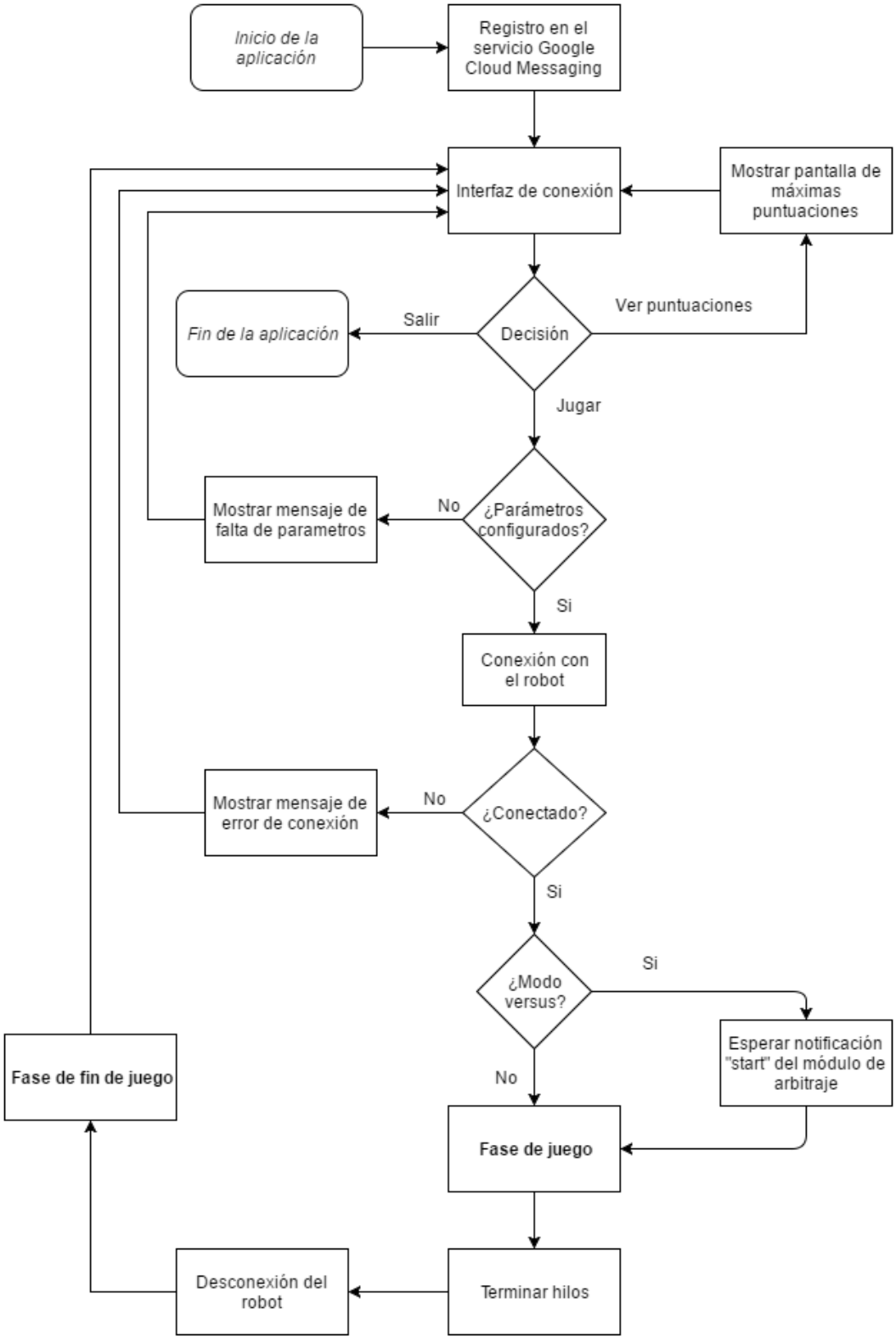

Ilustración 31 - Diagrama de flujo general

- **Inicio de la aplicación:** Se arranca la aplicación desde la lista de aplicaciones
- **Registro en el servicio de *Google Cloud Messaging*:** Se registra la aplicación el servicio GCM para obtener el *token* de identificación GCM.
- **Interfaz de conexión:** Como se puede observar en las Ilustraciones 32 y 33, se presenta al usuario varios campos para rellenar, el botón de modo de juego y tres opciones: (1) mostrar puntuaciones; (2) jugar una partida; y (3) salir de la aplicación.

Ilustración 32 – Interfaz de conexión (solo mode)

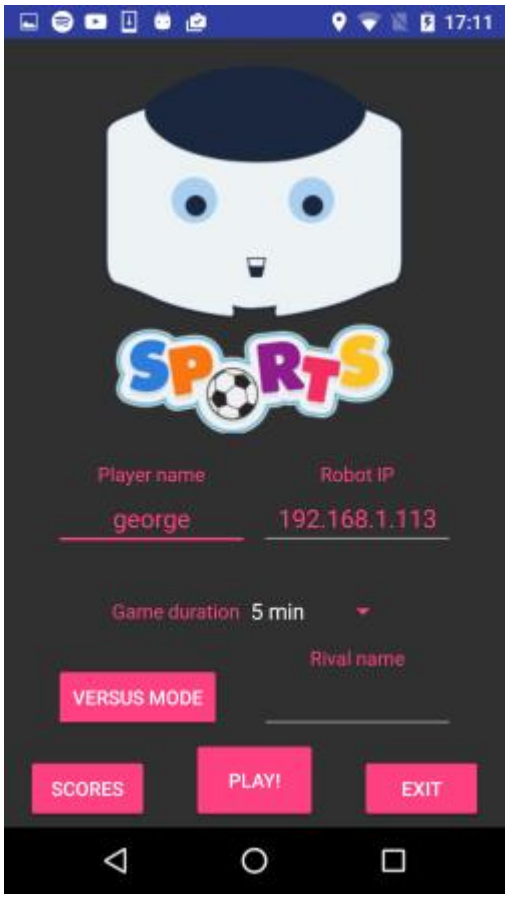

Ilustración 33 – Interfaz de conexión (versus mode)

- **Pantalla de máximas puntuaciones:** Se muestran las máximas puntuaciones de las partidas en modo individual (*solo mode*) guardadas en la base de datos de la aplicación. Se puede ver en la Ilustración 34.
- **Mostrar mensaje de falta de parámetros:** Como podemos ver en la Ilustración 35, si se pulsa el botón "*Play*" sin rellenar alguno de los parámetros obligatorios (nombre del jugador, IP del robot y el nombre del rival en modo *versus*), se muestra un mensaje de aviso.

Ilustración 34 - Pantalla de puntuaciones

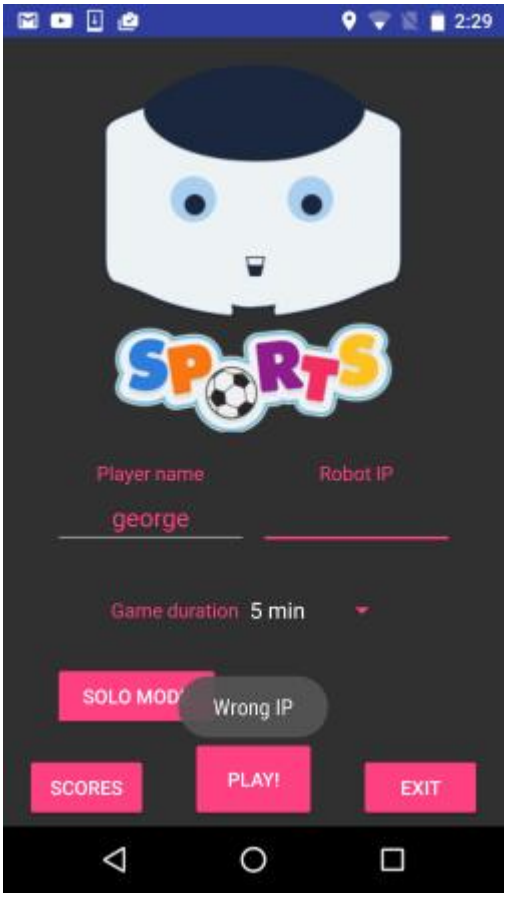

Ilustración 35 - Mensaje de falta de parámetros

- **Conexión con el robot:** Si la configuración de la partida es correcta, se intenta conectar con el robot. Una vez conectados, se mantiene una sesión que nos permitirá controlarlo.
- **Mostrar mensaje de error de conexión:** Si no se ha podido conectar con el robot, se muestra el mensaje que podemos ver en la Ilustración 36.
- **Esperar notificación *"start"* del módulo de arbitraje:** Si se configura una partida en modo *versus*, para empezar a jugar es necesario recibir una notificación por parte del módulo de arbitraje: para sincronizar la partida en los dispositivos de los dos jugadores. Para indicarlo se muestra un símbolo de carga a la espera de recibir la notificación, lo podemos ver en la Ilustración 37.

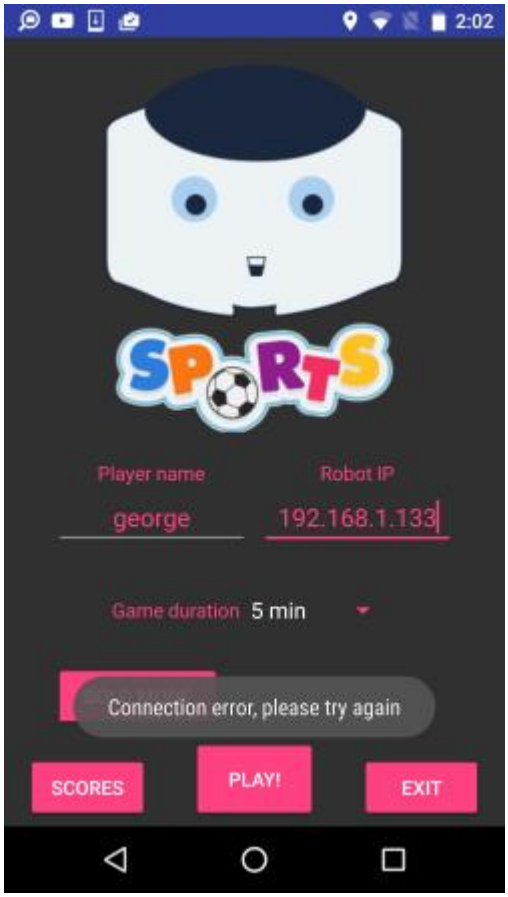

Ilustración 36 - Error de conexión

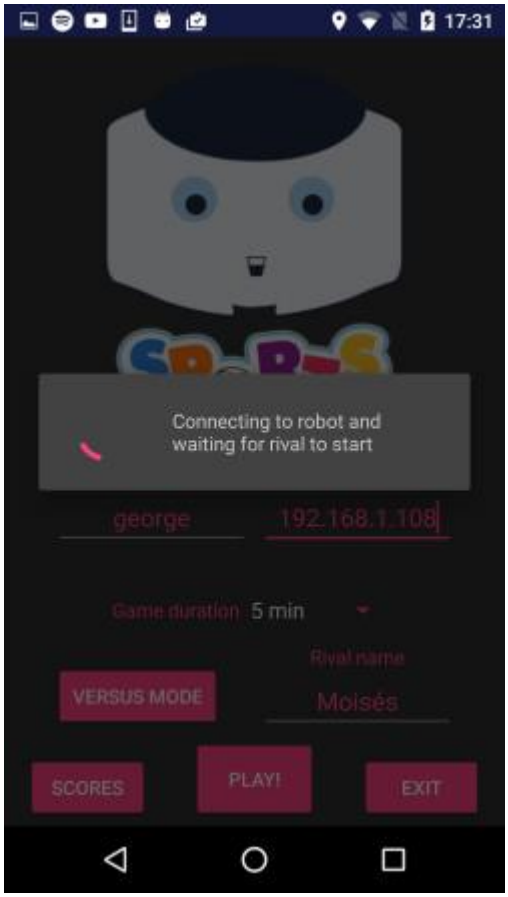

*Ilustración 37 - Esperando notificación "**start**" para jugar una partida en modo **versus***

- **Fase de juego:** La fase de juego es la parte más compleja de la aplicación porque en ella hay dos hilos extra ejecutándose simultáneamente al hilo principal (llamado *UI thread* en Android). Se explicará en detalle en la siguiente sección.
- **Terminación de los hilos:** Al finalizar la fase de juego, es necesario terminar los dos hilos extra que se abrieron para no consumir recursos. A partir de este punto continuamos la ejecución con el hilo principal (*UI Thread*).
- **Desconexión con el robot:** Después de finalizar la fase de juego se procede a cerrar la sesión que se mantenía abierta para dejar de consumir recursos.
- **Fase de fin de juego:** En esta parte de la aplicación se muestran los resultados obtenido durante la partida. Se explicará en detalle en la sección 3.3.2.3. Al cerrar esta pantalla se vuelve a la interfaz de conexión.

- **Fin de la aplicación:** Si pulsamos el botón "*Exit*" desde la interfaz de conexión cerramos la aplicación. También es posible salir si pulsamos el botón "atrás" del dispositivo.

3.3.2.2 Diagramas de flujo de la fase de juego

En Android, en el hilo principal (*UI Thread*) no se pueden realizar procesos que lo mantengan ocupado ya que este hilo es el encargado de interactuar con el usuario. Por eso, si el hilo principal se bloquea realizando una acción o esperando la respuesta durante más de tres segundos, el sistema operativo lanzará una excepción ANR (*Application Not Responding*) y pasará el control al usuario, quien tendrá que decidir si cerrar la aplicación o esperar a que la aplicación termine lo que está haciendo.

En la fase de juego se ejecutan varios procesos de forma simultánea:

1. Se tienen que controlar las acciones realizadas por el usuario, relativas a la teleoperación del robot.
2. En la interfaz de juego se muestran las imágenes recibidas desde la cámara del robot, esto conlleva petición, recepción y procesado constante de imágenes.
3. Las partidas tienen una duración, el tiempo restante se muestra en pantalla y tiene que actualizarse cada segundo. Además cuando el tiempo restante llegue a cero, se tiene que activar un procedimiento para terminar la partida.

Para que la aplicación sea capaz de ejecutar estos procesos sin bloquearse, es necesario crear dos hilos que, junto con el *hilo principal (UI Thread),* se repartan las tareas. A continuación se describe que es lo hace cada uno de ellos.

3.3.2.2.1 Hilo de juego (*UI Thread*)

Es el encargado de procesar las acciones del usuario. Las acciones introducidas por el usuario pueden ser de tres tipos:

- **Órdenes de teleoperación:** el usuario al utilizar los botones de control del robot en la interfaz de juego o utilizando el dispositivo móvil como un *joystick* realiza movimientos capturados por el acelerómetro que son transformados en movimientos de robot.
- **Reconectar con el robot:** Volver a conectar con el robot si se hubiera perdido la conexión
- **Abandonar partida:** El usuario decide abandonar la partida antes de que termine el tiempo de partida.

El diagrama de flujo del hilo de juego (Ilustración 38):

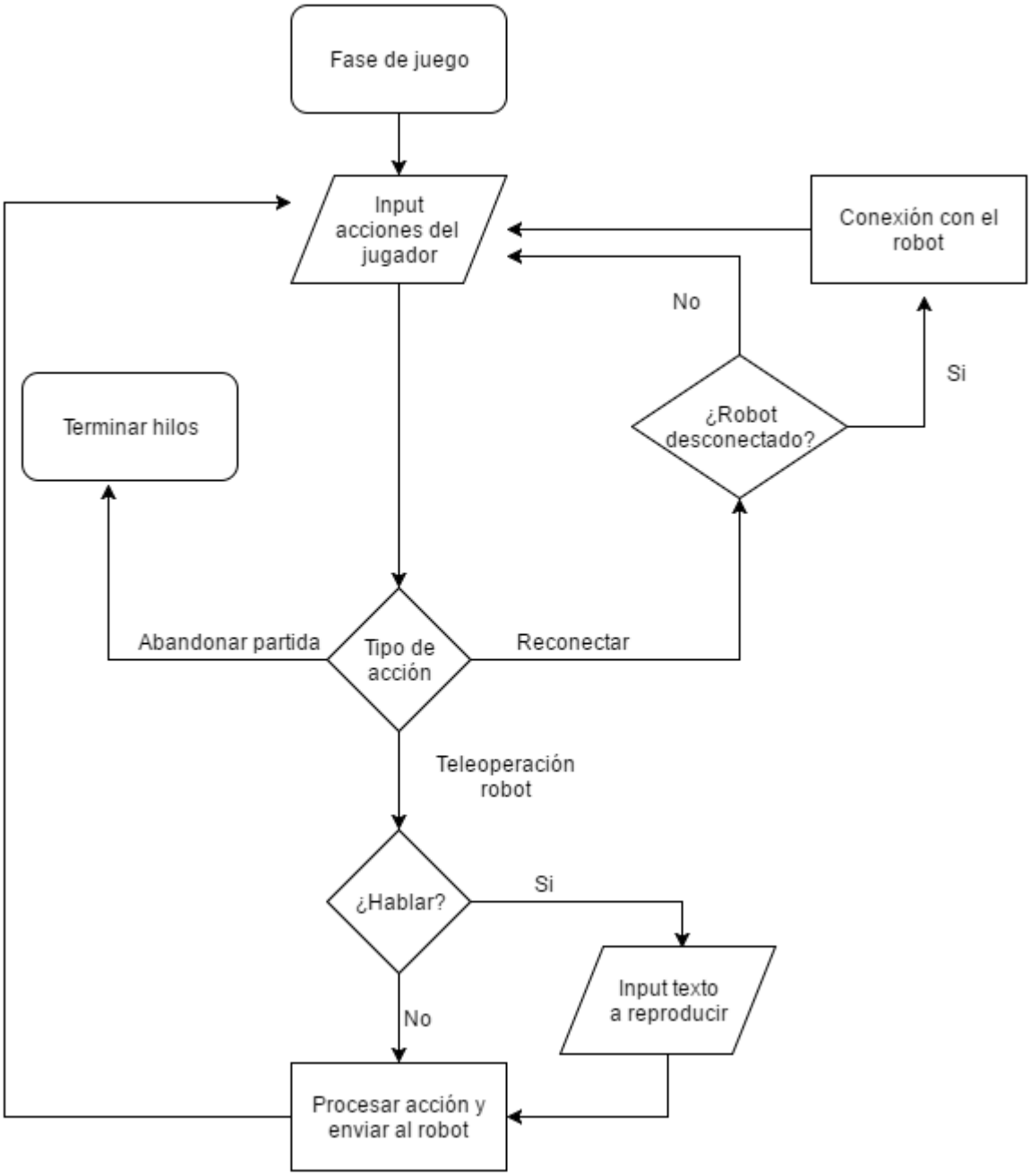

Ilustración 38- Diagrama de flujo del hilo de juego

- **Input acciones del jugador:** Este estado representa las distintas acciones que puede llevar a cabo el usuario interactuando con la interfaz de juego. Como podemos ver en las Ilustración 39, se presenta una interfaz con varios botones para controlar el movimiento del robot, además de las opciones de juego.

- **Procesar acción y enviar al robot:** Si el tipo de acción es una orden de teleoperación se envía al robot para que éste la realice. Si el usuario pulsa el botón "*talk*", se muestra un cuadro de texto para introducir el texto a reproducir por el robot (Ilustración 40).

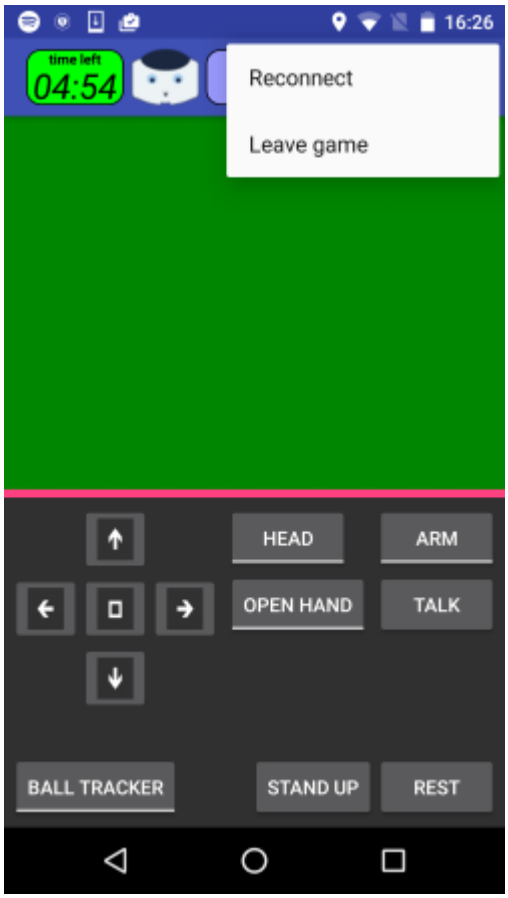

Ilustración 39 - Distintas acciones posibles para el usuario en la interfaz de juego

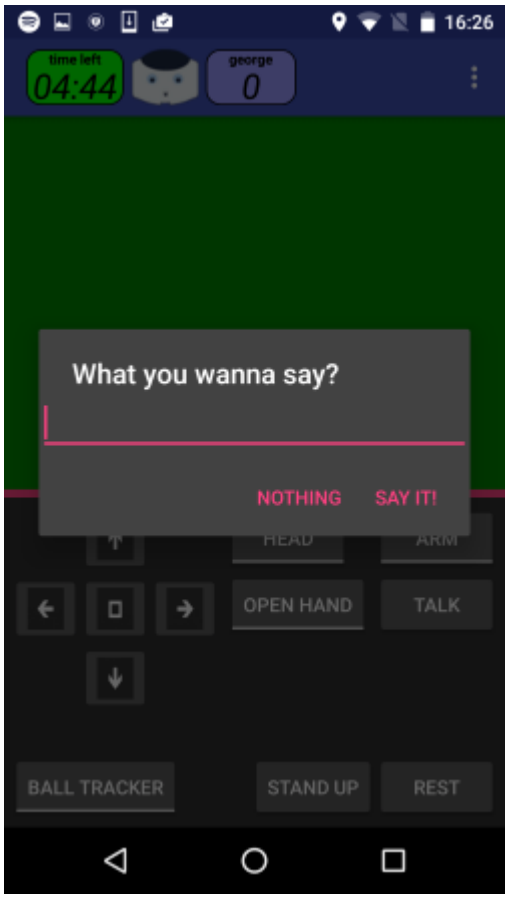

Ilustración 40 - Diálogo de texto para que lo reproduzca el robot

3.3.2.2.2 Hilo imagen

El *software* que tiene el robot no es capaz de hacer un *streaming* de video que se pueda capturar y reproducir en la aplicación. Sin embargo sí que nos permite hacerle peticiones para que nos mande fotogramas de la imagen capturada por una de sus cámaras; de manera que, si realizamos peticiones de manera continua y vamos refrescando la imagen que mostramos por pantalla, podemos conseguir una sensación de continuidad de la visión del robot. Cada uno de los fotogramas debe ser procesado ya que desde el robot recibimos un *array* de *bytes* que debemos

transformar en un mapa de *bits* para mostrarla por pantalla. El diagrama de flujo del hilo de la imagen lo podemos ver en la Ilustración 41.

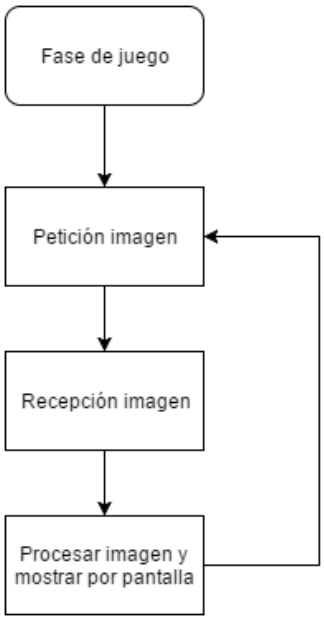

Ilustración 41- Diagrama de flujo del hilo imagen

Se puede ver un ejemplo de la imagen recibida desde el robot en la Ilustración 42.

Ilustración 42 - Recepción de imagen desde la cámara del robot

3.3.2.2.3 Hilo temporizador

El hilo temporizador se encarga de controlar el tiempo de duración de la partida: actualiza segundo a segundo el contador de tiempo de la interfaz de principal y se encarga de llevar al siguiente estado a la aplicación cuando el tiempo de juego se termina. En la Ilustración 43 podemos ver su diagrama de flujo.

Además, es el encargado de cambiar el color de fondo del contador de tiempo en la interfaz para avisar al usuario de que el tiempo se está agotando: si quedan más de tres minutos el color es verde, entre uno y tres minutos amarillo y si queda menos de un minuto el fondo es rojo (Ilustraciones 44, 45 y 46).

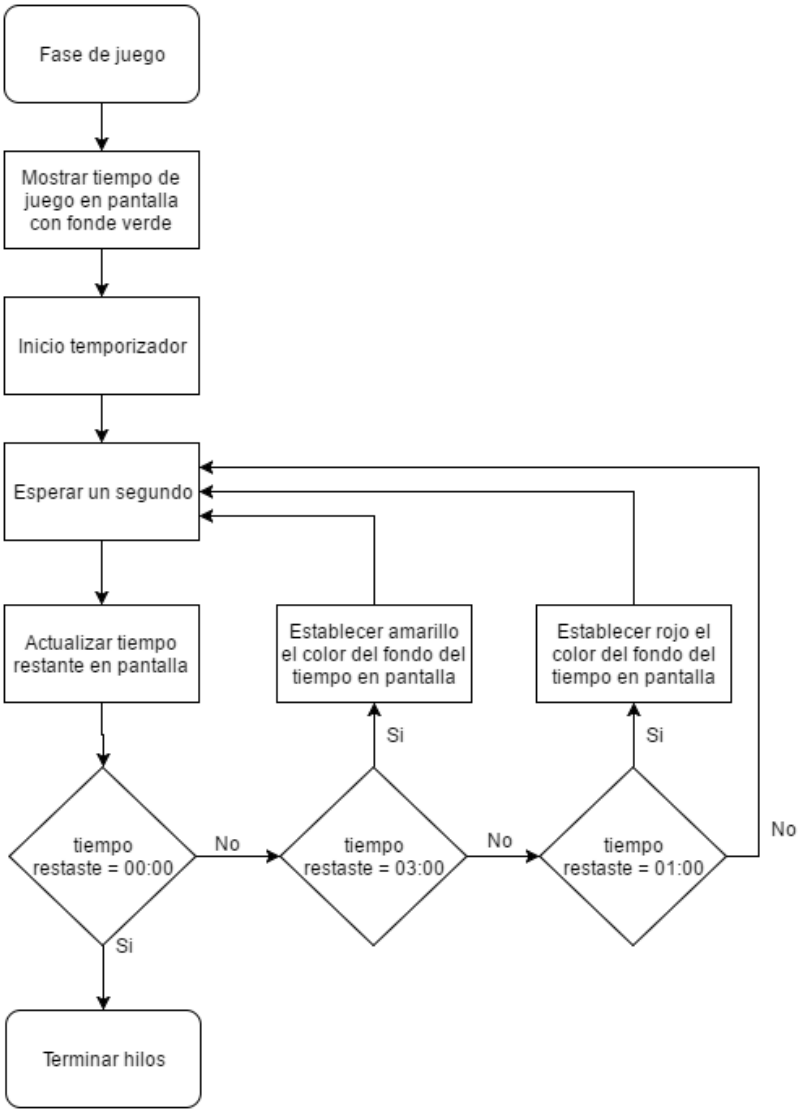

Ilustración 43- Diagrama de flujo de hilo temporizador

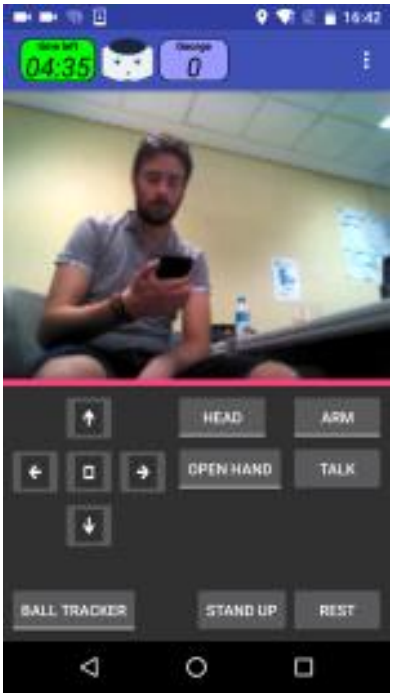

Ilustración 44 - Tiempo de juego verde

Ilustración 45 - Tiempo de juego amarillo

Ilustración 46 - Tiempo de juego rojo

Cuando el tiempo restante es cero, se cierran los hilos abiertos para la fase de juego y se continúa la ejecución en el hilo principal.

3.3.2.3 Diagrama de flujo de la fase de fin de juego

Al terminar una partida se llega a la fase de fin de juego, donde se muestra una pantalla que dependiendo del resultado de la partida y del modo de juego es diferente para cada caso. Tiene un caso particular en el modo individual, la puntuación obtenida por el usuario se guarda en base de datos. En la Ilustración 47 podemos ver el diagrama de flujo de la fase de fin de juego. En las Ilustraciones 48-50 vemos algunos ejemplos de las distintas pantallas de fin de juego. Al salir de la pantalla de fin de juego la aplicación vuelve a la interfaz de conexión.

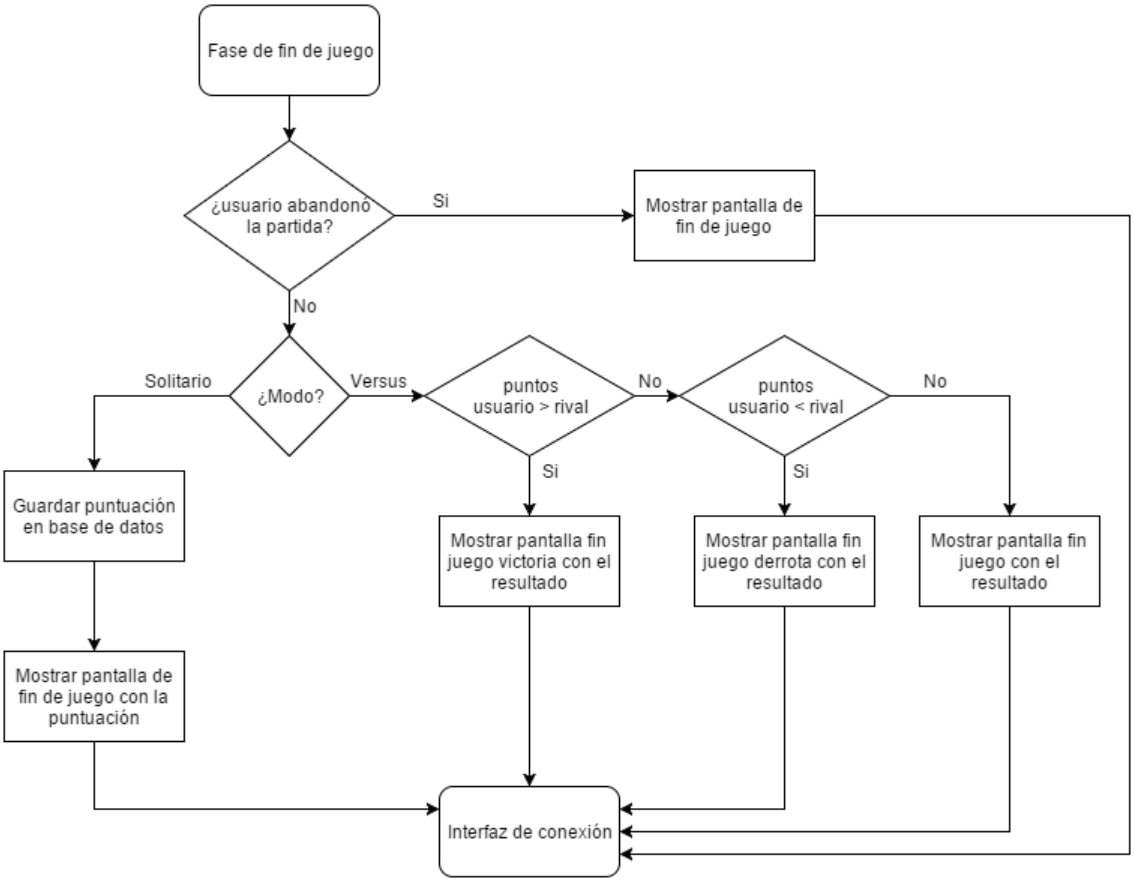

Ilustración 47 - Diagrama de flujo de fin de juego

Ilustración 48 - Pantalla fin de juego en modo individual

Ilustración 49 - Pantalla de fin de juego en modo versus victoria

Ilustración 50 - Pantalla de fin de juego en modo versus derrota

3.3.2.4 Diagrama de flujo del servicio de mensajería GCM

Al iniciar la aplicación, se inicia de manera paralela el servicio de mensajería GCM que está implementado como un *Service* de Android. En nuestra aplicación, es el encargado de escuchar por si el servicio de *Google Cloud Messaging* envía alguna notificación al dispositivo. En caso de recibir alguna notificación, el *Service* se dispara y la procesa. A continuación (Ilustración 51) podemos ver su diagrama de flujo:

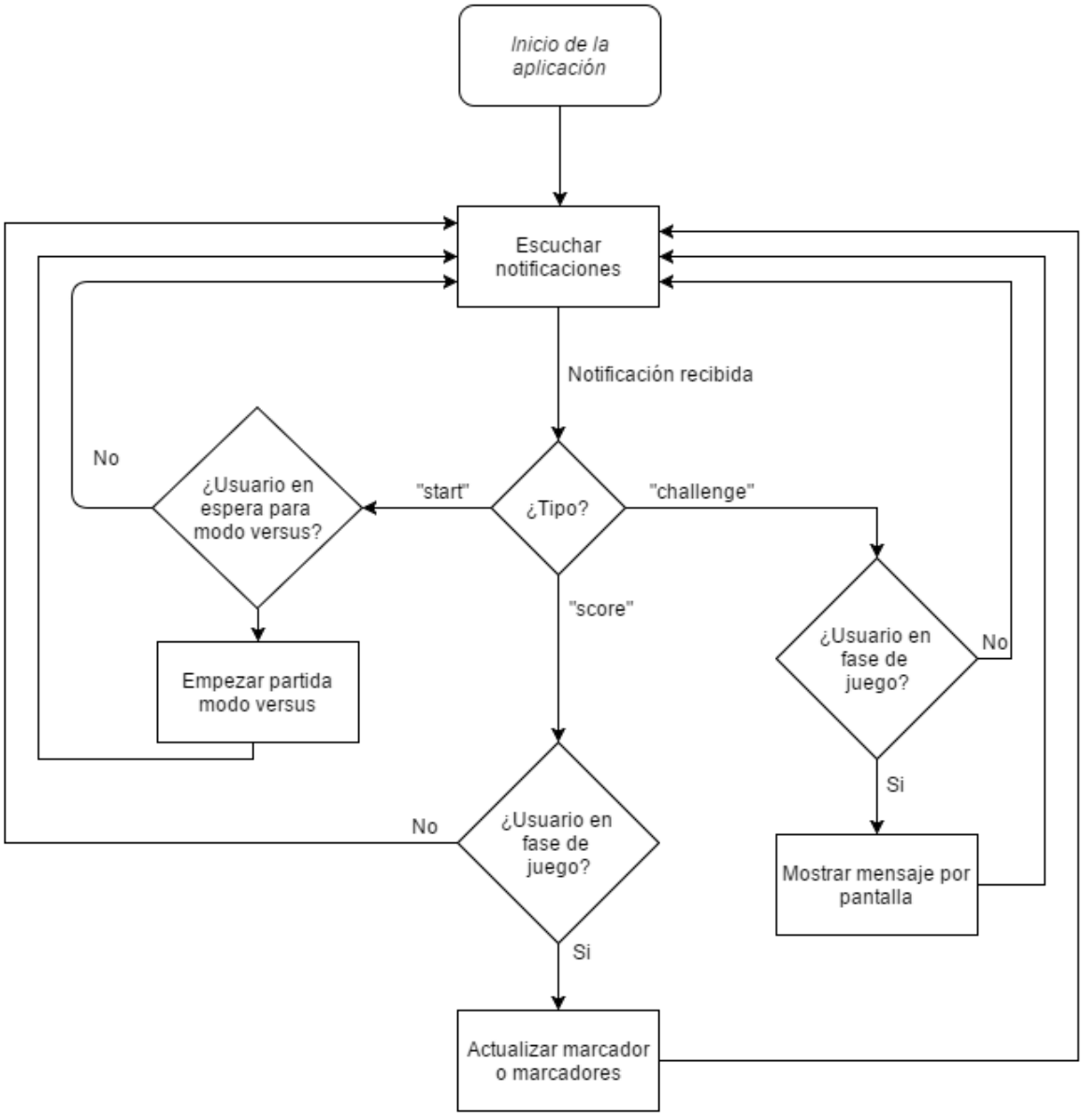

Ilustración 51 - Diagrama de flujo del servicio de mensajería GCM

El servicio de mensajería está corriendo en segundo plano durante todo el ciclo de vida de la aplicación. Cuando se recibe una notificación, se comprueba el tipo y se procesa. A continuación

describimos el resultado de procesar cada uno de los tipos de notificación. El formato de mensajes de las notificaciones se define más adelante, en la descripción de componentes del módulo de arbitraje.

- **Notificación *"start"*:** Una notificación de tipo *"start"* se utiliza para sincronizar el inicio de una partida entre dos dispositivos en modo *versus*. Esta notificación sólo se procesa cuando la aplicación se encuentra en espera para empezar una partida modo *versus* y se descarta si llega en cualquier otro momento. Al ser procesada, la aplicación pasa a la fase de juego.
- **Notificación *"score"*:** Este tipo de notificación sólo se procesa si la aplicación se encuentra en la fase de juego. Al ser procesada se actualiza el marcador de la partida en modo individual o los marcadores en modo *versus*. Además, se muestra en pantalla un mensaje notificando que el resultado se ha actualizado (Ilustración 52).

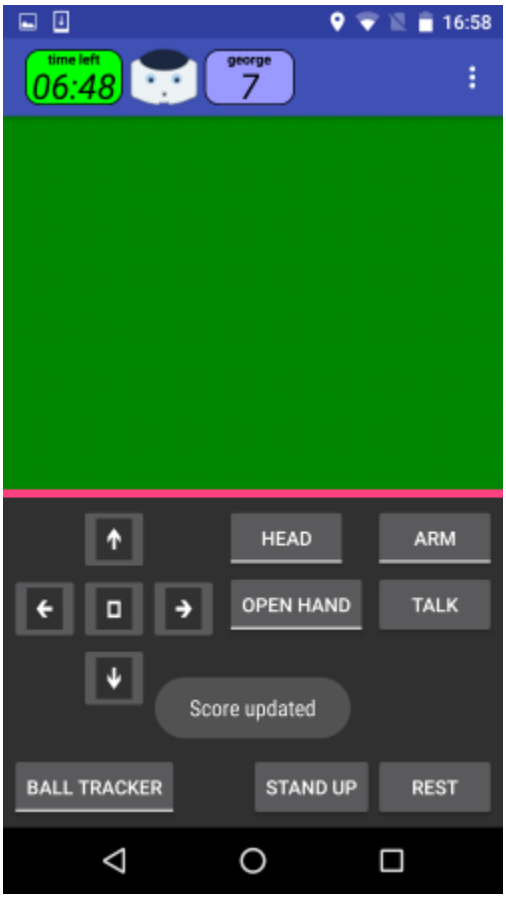

Ilustración 52 – Mensaje mostrado al recibir la notificación "score", el marcador se actualiza al recibirla

- **Notificación *"challenge"*:** La notificación *"challenge"*, al igual que *"score"*, sólo se procesa si la aplicación se encuentra en la fase de juego. Esta notificación se utiliza para añadir dinamismo al juego porque el árbitro puede mandar retos a los jugadores durante el transcurso de la partida. Al procesarla, se muestra por pantalla un mensaje que viene como parámetro de la notificación (Ilustración 53).

Ilustración 53 - Mensaje mostrado al recibir una notificación tipo "challenge"

3.3.3 Descripción de componentes

3.3.3.1 Módulo de juego

En la Ilustración 54 se muestra un diagrama de componentes que describe la estructura del módulo de juego. A continuación pasamos a describir cada uno de los componentes que lo forman.

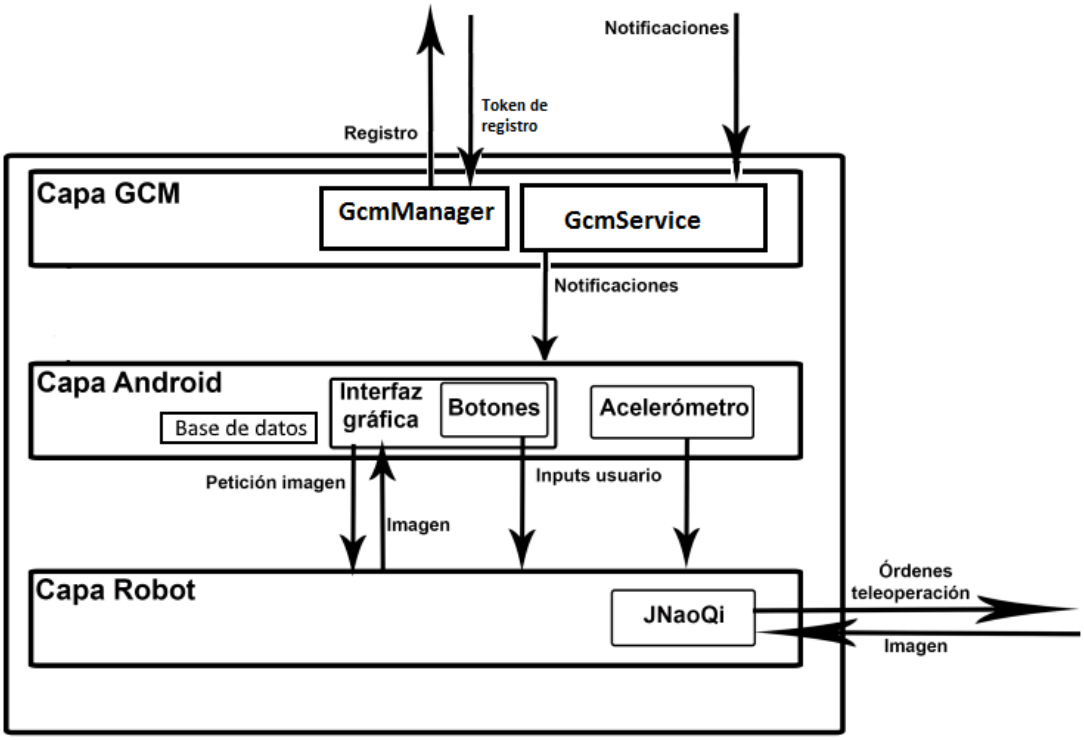

Ilustración 54 - Esquema de componentes del módulo de juego

3.3.3.1.1 Capa GCM

- **GcmManager:** Es una clase Java encargada de registrar al dispositivo móvil en el servicio de *Google Cloud Messaging* para poder recibir notificaciones. Si el registro se realiza correctamente, guarda el *token* de registro e inicia el *Service GcmService*.
- **GcmService:** Éste *Service* de Android utiliza las librerías del servicio de *Google Cloud Messaging* que proporciona el SDK de Android. En concreto hereda de la clase *GcmListenerService* que contiene el método *onMessageReceived(String from, Bundle*

data). Al heredar éste método, *GcmService* ya está preparado para recibir notificaciones *push* desde el servicio GCM. En los parámetros de entrada del método es donde está la información recibida en la notificación por lo que *GcmService* debe implementar la lógica para procesar los datos recibidos y notificar a la *Capa Android* de manera correspondiente dependiendo del tipo de notificación recibida.

3.3.3.1.2 Capa Android

- **Interfaz gráfica:** La interfaz gráfica está formada por dos *Activities*, y cada una de ellas tiene un *Fragment* asociado. El *Activity* que hace de punto de entrada de la aplicación es la interfaz de conexión (Ilustración 55) y desde ella se utilizan los métodos de la *Capa Robot* para realizar la conexión con el robot. Su *Fragment* asociado es la pantalla de puntuaciones (Ilustración 56), que se muestra al pulsar el botón *"Scores"*, éste *Fragment* se encarga de acceder a la base de datos para obtener la lista de puntuaciones y mostrarla.

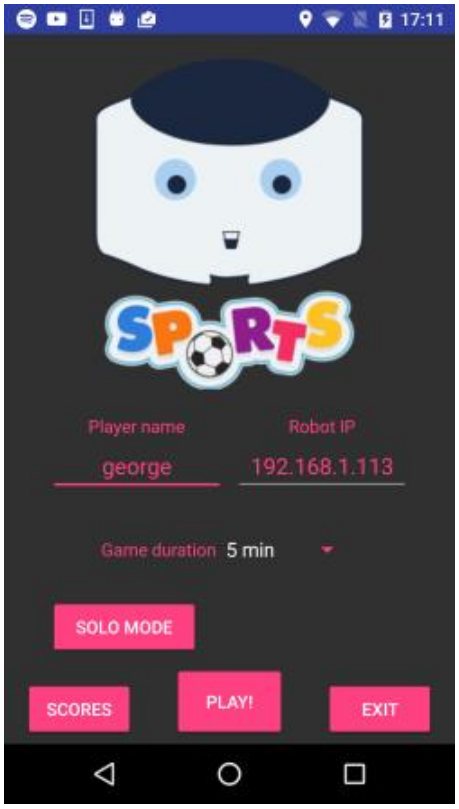

Ilustración 55 - Activity interfaz de conexión

Ilustración 56 - Fragment pantalla de puntuaciones

Una vez establecida la conexión se inicia el *Activity* de la interfaz de juego (Ilustración 57), donde el usuario interactúa con los distintos botones para controlar al robot, los *inputs* del usuario se traducen en la utilización de los métodos de teleoperación de la *Capa Robot*. Además, se muestran las imágenes recibidas desde su cámara frontal. El *Fragment* asociado a esta *Activity* es la pantalla de fin de juego (Ilustración 58), a la que sólo se accede cuando termina el tiempo de la partida o cuando el usuario abandona la misma. Se encarga de gestionar qué tipo de pantalla de fin de juego se debe mostrar (victoria, derrota, etc) y de guardar la puntuación en base de datos. Desde aquí la única opción posible es volver a la interfaz de conexión.

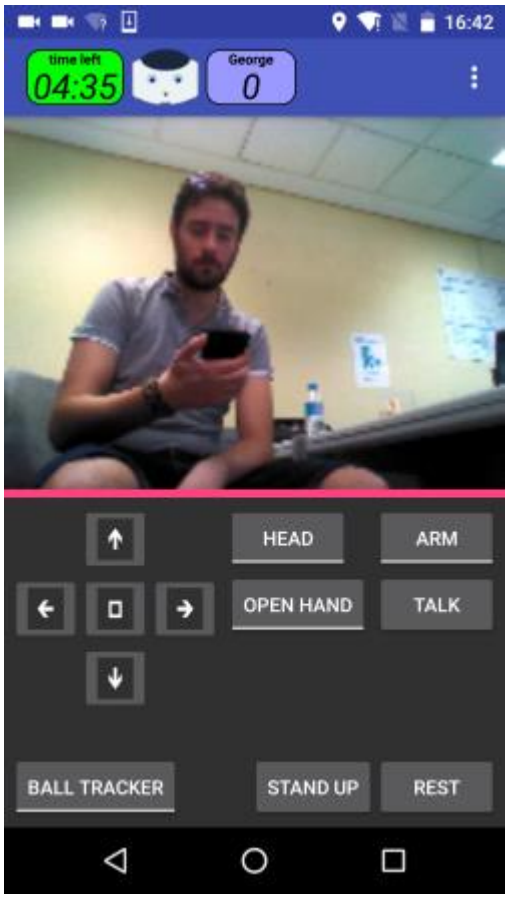

Ilustración 57 - Activity interfaz de juego

Ilustración 58 - Fragment pantalla fin de juego

- **Acelerómetro:** El acelerómetro es un sensor que genera eventos con los valores de aceleración registrados en cada uno de los ejes de coordenadas del dispositivo móvil (Ilustración 28). El objetivo es registrar estos eventos y traducir los movimientos realizados con el dispositivo móvil a movimientos de la cabeza y el brazo del robot. Para ello se

utilizarán solamente los valores de los ejes X e Y de los eventos lanzados por el acelerómetro. Para poder escuchar los eventos, el *Activity* de la interfaz de juego debe implementar la interfaz *SensorEventListener* que nos provee el SDK de Android. Esta interfaz contiene el método *onSensorChanged(SensorEvent e)* que debemos implementar en el *Activity*, y que es el que hará que recibamos los valores del acelerómetro.

Una vez que se escuchan los eventos, se debe establecer un umbral de movimiento que delimitará un estado de reposo del dispositivo. Esto es debido a que cualquier ligera variación de movimiento en el móvil produce eventos, por lo que se tienen que filtrar los valores más pequeños. Después de esto, lo que hay que hacer es guardar todos los valores que se generan entre estados de reposo y transformar el resultado en los movimientos del brazo y la cabeza del robot.

- **Base de datos:** Para guardar las puntuaciones de las partidas jugadas se utiliza un sistema de persistencia de datos. Este sistema es *SQLite*. Es una base de datos simple ya que solo cuenta con una tabla con dos campos: nombre de usuario y puntuación. Al terminar una partida en modo individual, se guarda la puntuación obtenida por el usuario y el nombre del jugador.

3.3.3.1.3 Capa Robot

La *Capa Robot* es una clase Java donde se encuentran los métodos de conexión y teleoperación del robot, para ello hace uso de las clases proporcionadas por la librería *JNaoQi*. Para enviar cualquier acción al robot es necesario iniciar una sesión con él, para ello se utiliza la clase *Session*. Esta sesión debe ser única ya que es el parámetro que se pasa a los distintos métodos relacionados con la teleoperación. Desde la *Capa Android* se accede a la sesión desde dos *Activities* distintas:

- La sesión con el robot se inicia desde la interfaz de conexión ya que si hay algún problema al conectar se muestra un mensaje de error.
- Una vez establecida la conexión con el robot y empezada la partida, la *Capa Android* utiliza la sesión desde la interfaz de juego para recibir la imagen y realizar la teleoperación.

Debido a que en Android no se pueden pasar objetos entre *Activities*, es necesario encontrar una manera persistir el objeto de la clase *Session* para pueda ser utilizada por las dos *Activities*. Para solucionar este problema, en el diseño de la capa robot se ha utilizado un patrón Singleton. Éste,

es un patrón de diseño utilizado para restringir la creación de objetos pertenecientes a una clase a un único objeto y su intención consiste en garantizar que una clase sólo tenga una instancia para proporcionar un punto global de acceso a ella. De esta manera, *Session* es un atributo de la *Capa Robot* al que se tiene acceso desde la Capa Android.

3.3.3.2 Módulo de arbitraje

En la Ilustración 59 se muestra un diagrama de componentes que describe la estructura del módulo de arbitraje. A continuación pasamos a describir cada uno de los componentes que lo forman.

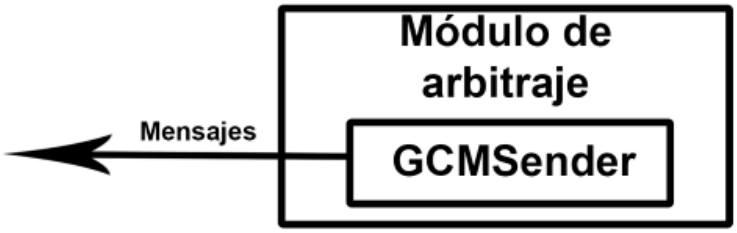

Ilustración 59 - Esquema de componentes de módulo de arbitraje

El módulo de arbitraje está compuesto por el GCMSender. Es una aplicación Java que pone a disposición del árbitro comandos distintos, y cada uno de estos comandos admiten distintos parámetros. Se ejecuta por línea de comandos y su función es formatear los datos introducidos por medio de los comandos en un *JSON* y enviarlos al servicio de *Google Cloud Messaging*.

- **Tipos de comandos y sus parámetros:**
  - ***"start"*:** Este comando se utiliza para dar comienzo a una partida entre dos jugadores. No admite parámetros.
  - ***"score"*:** Este comando se utiliza para actualizar el marcador de la partida en juego. Necesita el parámetro "playerScore" si el modo de la partida es individual. Si la partida es en modo *versus*, necesita los parámetros "playerScore" y "rivalScore".
    - **"playerScore":** El número de puntos con el que se actualizará el marcador del jugador uno.
    - **"rivalScore":** El número de puntos con el que se actualizará el marcador del jugador dos.

- ***"challenge"*:** Este comando se utiliza para enviar un reto al módulo de juego. Necesita el parámetro "*message*".
  - **"message":** El texto a mostrar en pantalla

- **Formato de los mensajes:** El formato utilizado para enviar los parámetros es *JSON*.
- **Conexión con *Google Cloud Messaging*:** Para enviar los datos al servicio de GCM es necesario abrir una conexión http con el servicio y realizar un petición POST con los datos que queremos enviar. Es importante incluir en la petición el *Server API Key* (como conseguir el *Server API Key* en el apartado 3.2.4.5) que relaciona al módulo de arbitraje con los dispositivos que ejecutan el módulo de juego.

Capítulo 4: Experimentación

4.1 Pruebas de teleoperación del robot

4.1.1 Conexión y ensayo de movimientos

En esta prueba se realiza la conexión con el robot desde el módulo de teleoperación y se comprueba que el robot responde a las acciones realizadas por el usuario. Una demostración de la prueba se puede ver en el siguiente enlace:

https://youtu.be/l4kJQa3geWg

4.1.2 Evasión de obstáculos

Este experimento consiste en controlar el robot para superar un recorrido de obstáculos y llegar hasta una marca en suelo que hace de meta, controlando el robot (Ilustración 60). El objetivo de esta prueba es comprobar si la aplicación se puede utilizar para teleoperar al robot desde otra habitación, guiándote solamente con la imagen recibida en el módulo de juego. Se puede ver una demostración de la prueba en el siguiente enlace:

https://youtu.be/U6q4hg3UPB0

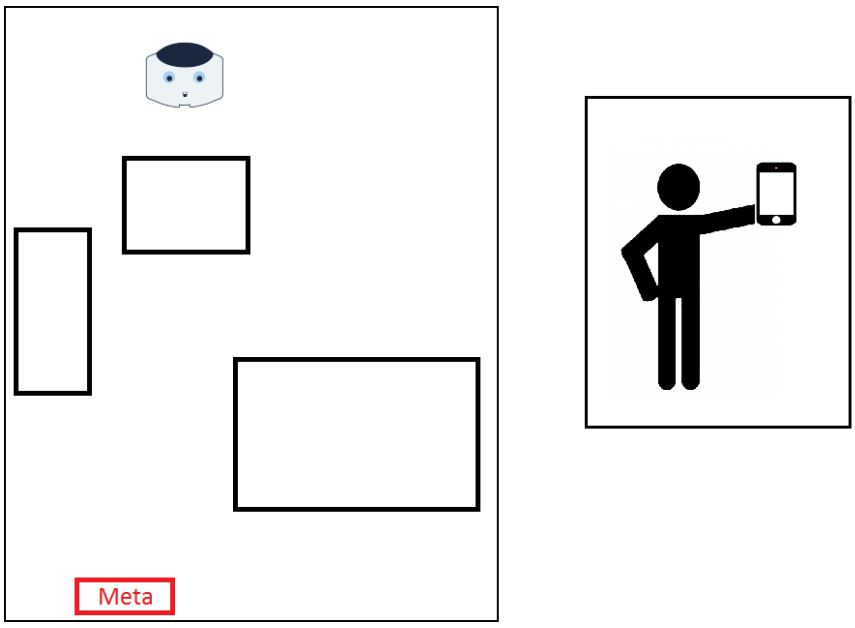

Ilustración 60 - Recorrido de obstáculos

4.1.3 Interactuar con el entorno

En esta prueba (Ilustración 61), el robot tendrá que realizar una serie de acciones:

1. Reconocer y acercarse de hasta una pelota que el usuario sostiene en su mano.
2. Pedirle la pelota al usuario y coger la pelota cuando éste se la dé.
3. Acercarse hasta una papelera y meter la pelota dentro.

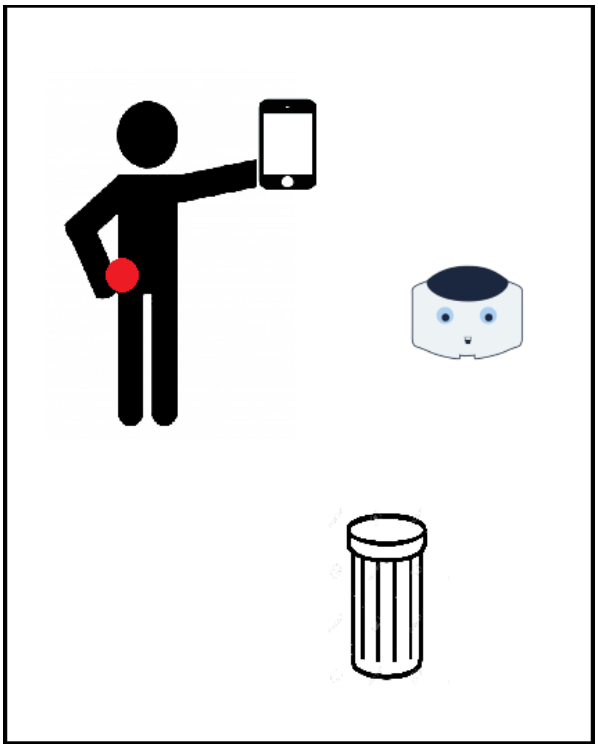

Ilustración 61 - Interactuar con el entorno

Se comprobará la capacidad de reconocer ciertos objetos del robot, en este caso una pelota. Además, se comprueba el movimiento de los brazos y las manos del robot. Una demostración de la prueba se puede ver en el siguiente enlace:

https://youtu.be/hVuk35QtQYk

4.2 Pruebas de sistema de juego

4.2.1 Recepción de notificaciones

En esta prueba se comprobará la recepción de notificaciones en módulo de juego. Para ello, se enviaran los tres tipos de notificaciones desde el módulo de arbitraje:

1. **Notificación *"start"*:** Al iniciar desde el módulo de juego una partida para dos jugadores se envía desde el módulo de arbitraje la notificación *"start"*. Al recibirla en el módulo de juego comenzará automáticamente la partida pasando a la interfaz de juego.
2. **Notificación *"score"*:** Durante el transcurso de una partida, al recibir esta notificación el módulo de juego actualiza el marcador con los datos contenidos en la notificación.
3. **Notificación *"challenge"*:** Durante el transcurso de la partida, al recibir esta notificación el módulo de juego muestra en pantalla el mensaje indicado por el árbitro.

Al realizar esta prueba, se comprueba que las notificaciones se reciben en cuanto el módulo de arbitraje envía el mensaje al servicio de *Google Cloud Messaging*. Se puede ver una demostración de la prueba se puede ver en el siguiente enlace:

https://youtu.be/YzPQWxkJST8

Nota: Ésta prueba se ha realizado conectando a un robot simulado, debido a que los robots del laboratorio están configurados para conectarse a un router que no tiene conexión a internet; que es imprescindible para la recepción de notificaciones. El envío de imagen está desactivado para mejorar la calidad del video.

4.3 Evaluación

En este apartado se expone cómo se realizó la experimentación del trabajo. Para ello, se contó con la ayuda de cuatro voluntarios que no tenían ninguna relación con el desarrollo del trabajo. Los voluntarios tuvieron que realizar varias pruebas sobre la aplicación que incluyen la configuración de partidas y la teleoperación del robot para realizar distintas acciones y jugar a juegos. Durante las pruebas, otro voluntario hacía de árbitro utilizando el módulo de arbitraje para validar las pruebas y asignar puntuaciones a los jugadores. Tras las pruebas, los usuarios tuvieron que rellenar un cuestionario que contiene preguntas de evaluación sobre la aplicación y los experimentos realizados. Los objetivos de la fase de experimentación son los siguientes:

- Probar el sistema en un entorno real, con usuarios externos.
- Comprobar las sensaciones de los usuarios acerca de la jugabilidad y usabilidad del sistema.
- Evaluar el grado de complejidad del sistema de teleoperación y el tiempo de adaptación que necesita un usuario para dominarlo.
- Conocer las opiniones de los usuarios sobre el diseño, funcionamiento y utilidad de la aplicación. Las opiniones recogidas podrán ser utilizadas como base para mejoras futuras.

Los usuarios que realizaron el experimento no estaban familiarizados con el uso de tecnologías relacionadas con la teleoperación, ni la robótica. Sí que estaban familiarizados con el uso de dispositivos móviles.

En todas las pruebas excepto en la tercera, los usuarios se encontraban en la misma habitación que el robot.

4.3.1 Cuestionario de evaluación

En el siguiente enlace encontramos el cuestionario *online* que tuvieron que rellenar los usuarios tras la realización de las pruebas:

- https://docs.google.com/forms/d/e/1FAIpQLScB-geRttOjnrAYMVwTOomFTbWPx1tzh4GHBvfvPOIFgoqe4w/viewform

4.3.2 Respuestas de los usuarios

Preguntas\Usuarios	1	2	3	4
Prueba 1: Configuración de una partida para un jugador				
¿Le ha parecido clara e intuitiva la interfaz de configuración de la partida?	Si	Si	Si	Si
¿Ha tenido algún problema para configurar e iniciar la partida?	No	No	No	No
Prueba 2: Familiarización con el movimiento de la cabeza y brazo				
¿Le ha costado cambiar entre los distintos tipos de movimiento?	No	No	No	No
¿El movimiento realizado por el robot respondía a las acciones realizadas?	Si	Si	No	Si
¿Le parece complicada la utilización de estos movimientos?	Si	Si	Si	Si
*Puntuación asignada por el árbitro**	*8*	*4*	*3*	*7*
Prueba 3: Llegar a la meta atravesando un recorrido de obstáculos manejando el robot desde otra habitación				
¿Las imágenes recibidas del robot se ven de manera continua y sin saltos?	Si	Si	Si	Si
¿Considera que las imágenes recibidas tienen una calidad aceptable?	Si	Si	No	Si
¿Ha conseguido superar el recorrido?	Si	Si	No	Si
¿El movimiento realizado por el robot respondía a las acciones realizadas?	Si	Si	No	Si
*Puntuación asignada por el árbitro**	*9*	*7*	*0*	*3*
Prueba 4: Hacer que el robot diga una frase				
¿Le ha costado encontrar la parte de la aplicación para poder hablar mediante el robot?	No	No	No	No
¿El robot ha pronunciado la frase introducida?	Si	Si	Si	Si

*Puntuación asignada por el árbitro**	*10*	*10*	*10*	*10*
Prueba 5: Jugar al baloncesto				
¿Ha conseguido coger la pelota?	Si	No	No	Si
¿Ha conseguido encestar la pelota?	Si	No	No	No
¿Le ha parecido útil la aplicación para este tipo de juegos?	Si	Si	Si	Si
*Puntuación asignada por el árbitro**	*10*	*0*	*0*	*5*

Tabla 72 - Respuestas de los usuarios a las preguntas de evaluación

**El rango de puntuaciones que podía asignar el árbitro se definieron antes del inicio de las pruebas. En este caso, se ha utilizado una escala de 0 a 10.*

4.3.3 Conclusiones de la evaluación de los usuarios

- Todos los usuarios han indicado que la interfaz de la configuración es muy intuitiva y sencilla. También que la configuración y conexión con el robot se realiza fácilmente. Ningún usuario encontró problemas al conectarse.
- Concluyeron que para conseguir el dominio de los movimientos relacionados con el acelerómetro (cabeza y brazo) es necesaria práctica. Todos los usuarios encontraron complicado controlar estos movimientos, pero estaban de acuerdo en que el robot respondía a los movimientos realizados. Después de unos 15 minutos de práctica, dos de los usuarios consiguieron dominarlos.
- Todos los usuarios estaban de acuerdo en que las imágenes se veían de manera continua y sin saltos en el dispositivo. Uno de ellos consideró que la calidad de la imagen era aceptable, el resto estaba de acuerdo en que era suficiente para teleoperar el robot.
- Tres de los cuatro usuarios consiguieron superar el recorrido sin chocar con ningún objeto. El tiempo medio para conseguir superar este recorrido fue elevado debido a las condiciones de la prueba, ya que se operaba desde otra habitación y era necesario utilizar el movimiento de la cabeza que todavía no estaba dominado.

- Todos los usuarios hicieron hablar al robot sin problemas. La opinión sobre la interfaz de juego también fue excelente respecto a usabilidad y sencillez.
- Sólo un usuario fue capaz de completarla la prueba del baloncesto. Esta es la prueba más difícil porque es necesario dominar el movimiento con el brazo.
- Durante la realización de las pruebas, se consiguió un ambiente de superación y de ganas de dominar los movimientos. Todos los usuarios quisieron volver a jugar una vez terminadas sus pruebas.
- Todos opinaron que el sistema era útil para jugar con el robot, y que tenía muchas posibilidades de uso para otro tipo de juegos, por ejemplo, uno de ellos sugirió que se podría utilizar para jugar al escondite con varios robots.

Capítulo 5: Gestión del proyecto

5.1 Descripción de las fases del proyecto

El trabajo se ha divido en varias fases, con el objetivo de agrupar las tareas en un orden de desarrollo lógico:

- **Análisis:** En la primera fase, el objetivo es establecer qué es lo que tiene que hacer el sistema para solucionar el problema propuesto. Se especificarán los casos de usos y los requisitos necesarios para el correcto funcionamiento del sistema. También se evalúa y se decide qué tecnologías utilizar para el diseño e implementación del proyecto.
- **Investigación:** Esta fase comprende el desarrollo del estado del arte, donde realizamos una investigación sobre los orígenes y evolución de las tecnologías presentes en el proyecto.
- **Aprendizaje:** Esta fase se dedica al aprendizaje y utilización las tecnologías elegidas: Java y Android. Además, se estudia el robot NAO y la integración del SDK JNaoQi.
- **Diseño:** Se diseña la arquitectura de la aplicación. Se identifican los distintos módulos que forman el sistema y funcionamiento de cada uno de ellos.
- **Implementación:** Durante esta fase, se desarrollan los módulos de juego y de arbitraje identificados en la fase de diseño. Durante el desarrollo, se realizan pruebas de los componentes de manera independiente utilizando entornos de simulación.
- **Pruebas generales del sistema:** Se realizan las pruebas del correcto funcionamiento de la aplicación en un entorno real. Además, el sistema será evaluado por usuarios externos.
- **Documentación:** En esta última fase, se redacta la memoria del trabajo en base a los apuntes obtenidos en las fases anteriores.

5.2 Planificación

La planificación de proyecto se hace en base a las fases definidas en el punto anterior. Cada una de estas fases se ha dividido en una serie de tareas. Cada una de estas tareas tiene un identificador, se indican el número de días necesitados para su finalización y las fechas que comprenden el inicio y el fin de la tarea. El diagrama de Gantt obtenido a partir de las fases de la planificación lo encontramos en la Ilustración 62.

Id	Tarea	Fecha inicio	Fecha fin	Duración (días)
	Análisis	**8/3/2016**	**27/3/2016**	**20**
1	Análisis de Android	8/3/2016	17/3/2016	10
2	Análisis del Robot NAO	18/3/2016	22/3/2016	5
3	Especificación caso de uso	23/3/2016	24/3/2016	2
4	Especificación de requisitos	25/3/2016	27/3/2016	3
	Investigación	**28/3/2016**	**10/4/2016**	**14**
5	Investigación robótica	28/3/2016	3/4/2016	7
6	Investigación teleoperación	4/4/2016	10/4/2016	7
	Aprendizaje	**11/4/2016**	**25/5/2016**	**45**
7	Aprendizaje Android	11/4/2016	5/5/2016	25
8	Aprendizaje robot NAO	6/5/2016	15/5/2016	10
9	Aprendizaje SDK JNaoQi	16/5/2016	25/5/2016	10
	Diseño	**26/5/2016**	**24/6/2016**	**30**
10	Diseño módulo de juego	26/5/2016	14/6/2016	20
11	Diseño módulo de arbitraje	15/6/2016	24/6/2016	10
	Implementación	**25/6/2016**	**14/8/2016**	**51**
12	Interfaz gráfica del módulo de juego	25/6/2016	9/7/2016	15
13	Lógica del módulo de juego	10/7/2016	29/7/2016	20
14	Desarrollo módulo de arbitraje	30/7/2016	6/8/2016	8
15	Prueba de los componentes	7/8/2016	9/8/2016	3
16	Corrección de errores encontrados en las pruebas	10/8/2016	14/8/2016	5
	Pruebas generales del sistema	**15/8/2016**	**29/8/2016**	**15**
17	Pruebas del sistema	15/8/2016	24/8/2016	10
18	Evaluación de los resultados	25/8/2016	29/8/2016	5
	Documentación	**30/8/2016**	**26/9/2016**	**28**
20	Elaboración de la memoria	30/8/2016	23/9/2016	25
21	Preparación de la presentación	24/9/2016	26/9/2016	3

Tabla 73 - Planificación del proyecto

Nº	Nombre de tarea	Comienzo	Fin	Duración-
	ANÁLISIS	**8/3/16**	**27/3/16**	**20 días**
1	Análisis de Android	8/3/16	17/3/16	10 días
2	Análisis del Robot NAO	18/3/16	22/3/16	5 días
3	Especificación caso de uso	23/3/16	24/3/16	2 días
4	Especificación de requisitos	25/3/16	27/3/16	3 días
	INVESTIGACIÓN	**28/3/16**	**10/4/16**	**14 días**
5	Investigación robótica	28/3/16	3/4/16	7 días
6	Investigación teleoperación	4/4/16	10/4/16	7 días
	APRENDIZAJE	**11/4/16**	**25/5/16**	**45 días**
7	Aprendizaje Android	11/4/16	5/5/16	25 días
8	Aprendizage robot NAO	6/5/16	15/5/16	10 días
9	Aprendizaje SDK JNaoQi	16/5/16	25/5/16	10 días
	DISEÑO	**26/5/16**	**24/6/16**	**30 días**
10	Diseño módulo de juego	26/5/16	14/6/16	20 días
11	Diseño módulo de arbitraje	15/6/16	24/6/16	10 días
	IMPLEMENTACIÓN	**25/6/16**	**14/8/16**	**51 días**
12	Interfaz gráfica del módulo de juego	25/6/16	9/7/16	15 días
13	Lógica del módulo de juego	10/7/16	29/7/16	20 días
14	Desarrollo módulo de arbitraje	30/7/16	6/8/16	8 días
15	Prueba de los componentes	7/8/16	9/8/16	3 días
16	Corrección de errores de las pruebas	10/8/16	14/8/16	5 días
	PRUEBAS GENERALES DEL SISTEMA	**15/8/16**	**29/8/16**	**15 días**
17	Pruebas del sistema	15/8/16	24/8/16	10 días
18	Evaluación de los resultados	25/8/16	29/8/16	5 días
	DOCUMENTACIÓN	**30/8/16**	**26/9/16**	**28 días**
19	Elaboración de la memoria	30/8/16	23/9/16	25 días
20	Preparación de la presentación	24/9/16	26/9/16	3 días

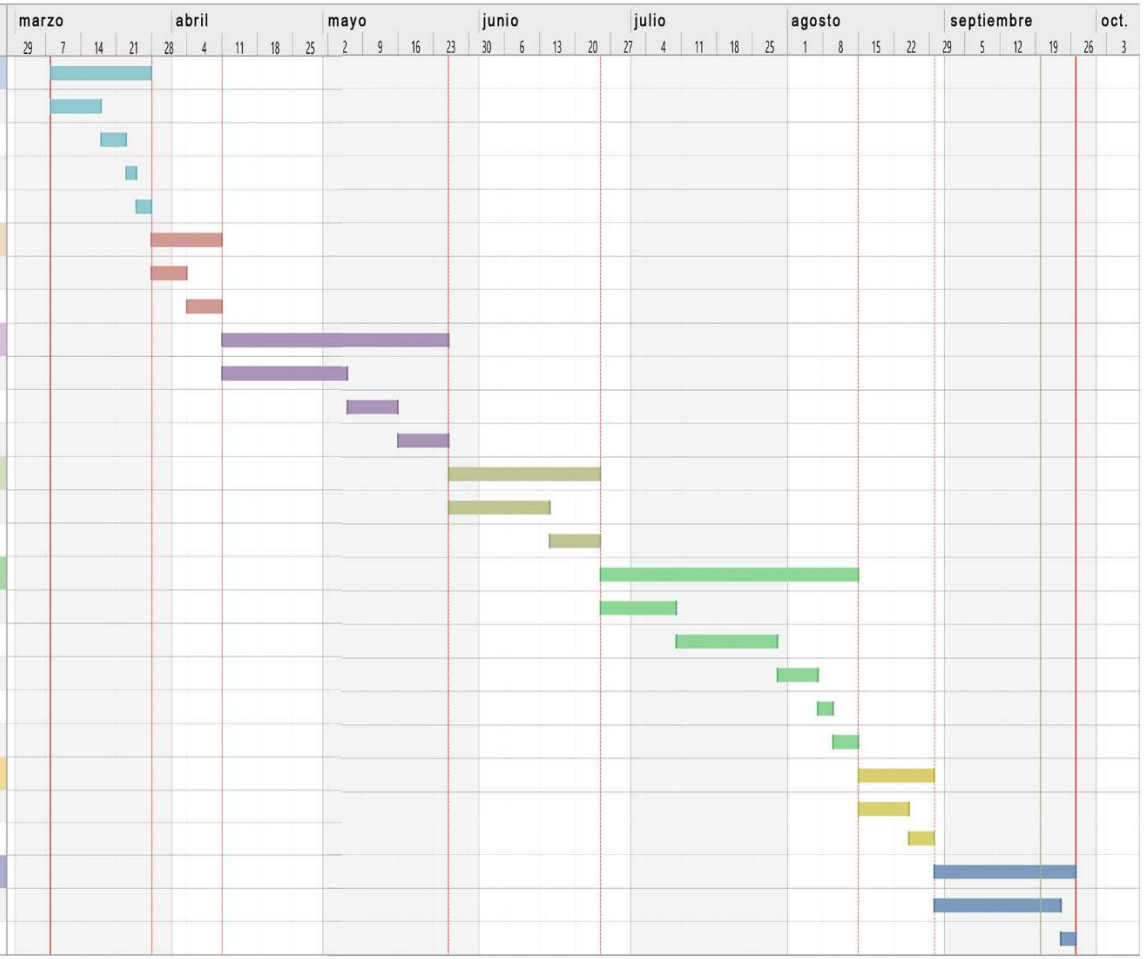

Ilustración 62 - Diagrama de Gantt del proyecto

5.3 Presupuesto

1. Autor:

Jorge Alcolea Coronel

2. Departamento:

Telemática

3. Descripción del proyecto:

- **Título:** Sistema de teleoperación mediante Android para jugar con el robot NAO
- **Duración:** 203 días (7 meses)
- **IVA:** 21%

4. Desglose de gastos:

Puesto	Precio/año	Meses trabajados	Salario final
Analista programador	21430,82€	4	7143,60€
Programador	18783,34€	5	7826.39€
Investigador	18972,57€	0.5	790.52€
		Total	**15760.51€**

Tabla 74 - Gastos de personal

Recurso	Precio	Dedicación (meses)	Periodo de deprecación (meses)	Amortización*
Portatil Samsung R512	500,00€	7	48	72.91€
LG Nexus 5	350,00€	7	48	51.04€
Robot NAO	8000,00€	6	48	1166.00€
			Total	**1289,95€**

Tabla 75 – Gastos de recursos hardware

Recurso	Precio	Dedicación (meses)	Periodo de deprecación (meses)	Amortización*
Webots for NAO	2300,00€	3	48	143,75€
Android Studio	0,00€	4	0	0,00€
Choregraphe Suite	0,00€	3	0	0,00€
SDK JNaoQi	0,00€	3	0	0,00€
JDK (Java Development Kit)	0,00€	4	0	0,00€
Ubuntu 14.04	0,00€	7	0	0,00€
			Total	**143,75€**

Tabla 76 - Gastos de recursos software

**Cálculo de amortización:*

$\frac{A}{B} \times C$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = Periodo de deprecación (48 meses)

C = Coste del equipo (sin IVA)

5. Resumen de costes

Resumen de costes	
Personal	15760,51€
Amortización	1433.70€
IVA	21%
Total	**20804.99€**

Tabla 77 - Resumen de costes

El presupuesto total de este trabajo asciende a la cantidad de VEINTE MIL OCHOCIENTOS CUATRO CON NOVENTA Y NUEVE EUROS (20804.99€).

Capítulo 6: Conclusiones y trabajos futuros

6.1 Conclusiones generales

Nunca había tenido que enfrentarme a un proyecto de esta magnitud y las sensaciones finales son buenas. La realización de este trabajo me ha permitido acercarme a las diferentes labores y funciones que puede desempeñar un ingeniero.

En primer lugar, me gustaría destacar las tareas de investigación realizadas, que me han hecho darme cuenta de la importancia que los campos de la robótica y la teleoperación han tenido sobre el ser humano. En el transcurso de la carrera no habíamos tenido que hacer ninguna investigación de este tipo. Ha sido necesario leer y sintetizar mucha información y artículos científicos muy interesantes. Además, me ha servido para mejorar mis estrategias de búsqueda de información.

La realización de este trabajo me ha permitido afianzar profundamente mis conocimientos sobre Android. El desarrollo de una aplicación desde cero ha supuesto un desafío, pasando de la organización de la estructura de la misma a la utilización de patrones de diseño para resolver los ciertos problemas.

Para terminar, ha sido un placer poder trabajar en un laboratorio de la universidad, donde me han ayudado y he sentido que hay un ambiente especial.

6.2 Conclusiones referentes a los objetivos

En este apartado se analizan las conclusiones obtenidas respecto a los objetivos marcados al inicio del proyecto. La definición de los objetivos se encuentra en el apartado 1.3.

1. **Análisis del funcionamiento del robot NAO**

   Se realizó un estudio de las funcionalidades y características del robot, centrándose en las que podrían aplicarse para la realización del trabajo. Este estudio sirvió para conocer en profundidad el robot NAO, que es uno de los robots programables más utilizados, y ayudó a decidir cuáles de sus funcionalidades se podrían aplicar al trabajo.

2. **Estudio del sistema operativo Android**

   He podido ampliar mis conocimientos sobre Android al tocar temas tan diversos como la creación de interfaces de usuarios, la comunicación con *Google Cloud Messaging*, la

utilización de sensores como el acelerómetro y la implementación de *Services*. Cabe destacar la gran cantidad de información que puedes encontrar sobre Android en la red y la calidad de la información de la plataforma para desarrolladores de Google. Los conocimientos adquiridos en este campo son de gran valía para el futuro.

3. **Estudio del SDK JNaoQi**

   Se han conseguido los conocimientos suficientes para integrar las librerías del SDK en la aplicación y crear las órdenes de teleoperación para poder controlar al robot.

4. **Diseño del sistema de juego**

   Fue una de las fases más complicadas que ya hubo que encontrar una solución para poder ofrecer un sistema de juego que pudiera ser utilizado en distintas situaciones. La solución propuesta con el sistema de arbitraje ha sido la clave para resolver el problema. Uno de los puntos fundamentales de este sistema de juego es que la imaginación de los usuarios al jugar con la aplicación es fundamental.

5. **Diseño de la estructura de la aplicación**

   Los conocimientos adquiridos en el estudio de Android han resultado decisivos para el diseño final de la estructura de la aplicación. Una vez se definía el sistema de juego, la estructura de la aplicación se fue modificando mientras se iba desarrollando la aplicación. El resultado final, con dos *Activities* y dos *Fragments,* la hacen una aplicación sencilla pero con mucha profundidad gracias a sus funcionalidades.

6. **Experimentación**

   La realización de este objetivo ha sido la parte más gratificante. Después de meses de investigación, diseño y desarrollo del sistema haciendo pruebas en el simulador, me ha dado la oportunidad de experimentar en el laboratorio, en un entorno real con un robot. Cabe destacar la precisión y fiabilidad del software de simulación *WeBots*, ya que las pruebas en el laboratorio por norma general han tenido resultado positivo debido a que ya que habían sido probadas en el simulador. Las pruebas realizadas con los usuarios fueron muy satisfactorias, ya que vieron mucho potencial en el proyecto y pasaron un buen rato realizándolas.

7. **Elaborar la documentación del proyecto**

   Gracias a las notas y apuntes tomadas a lo largo de las fases de análisis y diseño de la aplicación, la elaboración de la documentación ha sido una tarea sencilla pero a la que se ha dedicado gran parte de la duración del proyecto. Las tareas de investigación han sido las que más se han disfrutado en esta fase ya que no las había realizado nunca para un proyecto de esta envergadura.

6.3 Problemas encontrados

Los principales problemas vinieron por el SDK JNaoQi. No por su integración en el sistema, sino porque Aldebaran Robotics retiró hace tiempo de su página web las versiones de JNaoQi optimizadas para Android, por lo que hubo que buscar en Internet para poder descargar las librerías compiladas. Éstas se encontraron en proyectos de la plataforma de código GitHub [34], pero solo la versión 2.1.4 de JNaoQi. Ésta versión solo funciona en uno de los robots del laboratorio, ya que el otro es más antiguo y tiene instalada la versión 1.4.0, que no es compatible con la versión 2.1.4. Debido a este problema no se pudo realizar la evaluación utilizando dos robots, simultáneamente, en un entorno real.

6.4 Trabajos futuros

Se presenta una serie de mejoras que se podrían aplicar a los módulos del proyecto:

Módulo de juego

- Desarrollar un servicio de *login* para el módulo de juego, para ello habría que desarrollar una aplicación web sobre un servidor que gestione la lógica necesaria para hacer el *login* y una base de datos para la persistencia de los datos.
- Añadir un sistema de detección de los robots que estén conectados a la red Wifi.
- Permitir al usuario el cambio de cámara del robot para aumentar la telepresencia del sistema.
- Añadir nuevos movimientos a realizar con el robot. Esta mejora aumentaría las posibilidades del uso de la aplicación.
- Mejorar la precisión de los movimientos realizados que utilizan el acelerómetro.

- Mejorar la teleoperación mediante la utilización de los sensores no utilizados del robot. El sonar podría utilizarse para saber a qué distancia están los objetos frente al robot, lo que aumentaría la sensación de telepresencia.
- La visión de las imágenes podría mostrarse mediante un *streaming* de video realizado desde el robot. Esto mejoraría la calidad de la imagen y eliminaría los procesos de tratamiento de la imagen.
- Crear una sección en la aplicación en la que se propongan temáticas y juegos en los que utilizarla. Permitir a los usuarios que compartan sus ideas con la comunidad por medio de esta acción.

Módulo de arbitraje:

- Desarrollo de una interfaz gráfica para un manejo más intuitivo del módulo.

Capítulo 7: Anexos

A Repositorio de código del proyecto

Durante el desarrollo del proyecto se utilizó el software de control de versiones *git* [33], para dejar constancia de las funcionalidades que se iban desarrollando y para tener una copia de seguridad del código. En concreto se utilizó la plataforma GitHub [34]. En el siguiente enlace se encuentra repositorio con el código del sistema.

https://github.com/wasonte/NaoSports

Para descargar el código hace falta tener instalado el software *git*, que se puede descargar de manera gratuita desde su página oficial, y ejecutar el siguiente comando en un terminal:

>> git clone https://github.com/wasonte/NaoSports

A continuación, en la Ilustración 63, se puede ver la organización de las carpetas del repositorio de código.

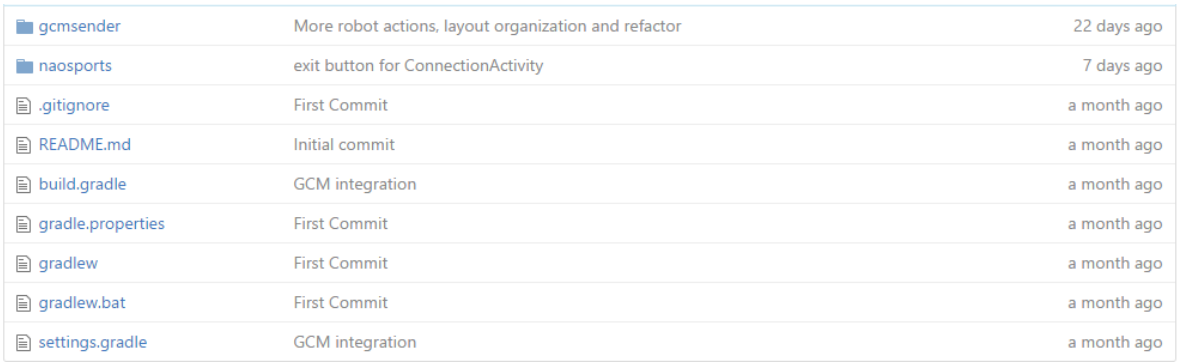

Ilustración 63 - Organización de las carpetas del proyecto

- En la carpeta 'gmcSender' se encuentra el código relativo al módulo de arbitraje.
- En la carpeta 'naosports' se encuentra el código relativo al módulo de juego.
- El resto de archivos que se ven en la Ilustración 63 son archivos de configuración del proyecto de Android.

B Manual de compilación e instalación del módulo de juego

Para compilar el módulo de juego, al tratarse de una aplicación Android, es necesario el *IDE Android Studio* [35], que se puede descargar de manera gratuita desde su página oficial.

1. Descargar el código del proyecto (instrucciones en el Anexo A).
2. Abrir *Android Studio* y seleccionar la opción "Abrir un proyecto existente de *Android Studio*" (Ilustración 64).

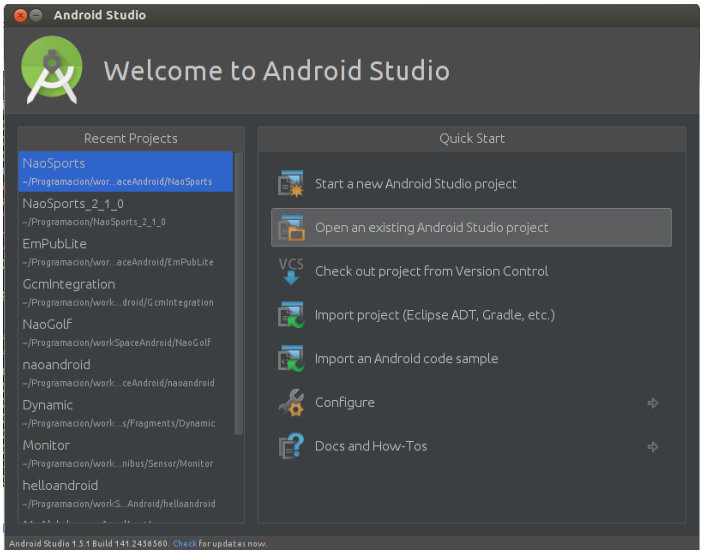

Ilustración 64 - Abrir proyecto existente Android Studio

3. Seleccionar el directorio donde se ha descargado el código del proyecto (Ilustración 65).

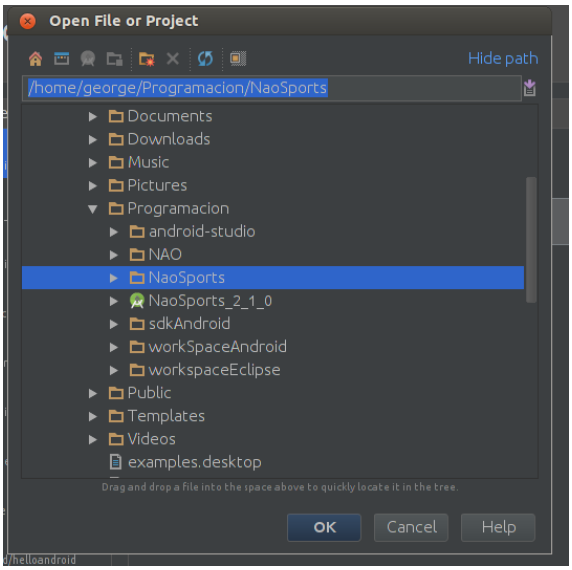

Ilustración 65 - Seleccionar directorio con el código

4. Una vez abierto el proyecto, pulsar el botón en la barra de tareas de *Android Studio* (Ilustración 66).

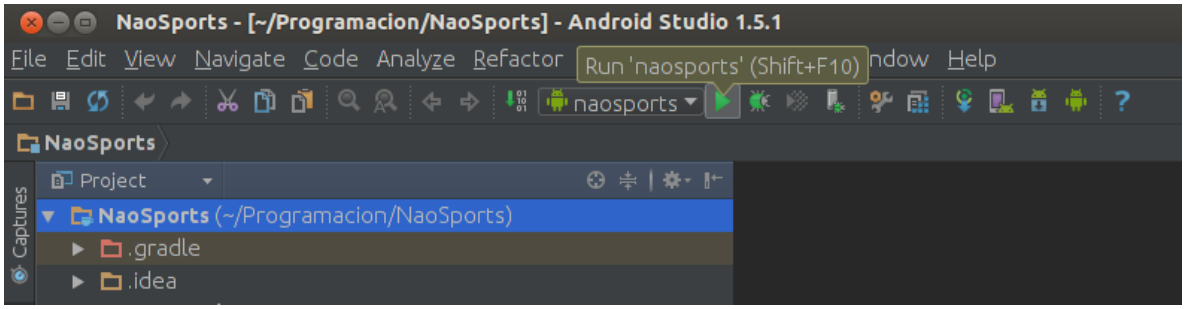

Ilustración 66 - Barra de tareas Android Studio

5. El proyecto se compilará.
6. Una vez compilado se instalará en el dispositivo que se seleccione. Puede ser un emulador o un dispositivo físico. Es recomendable utilizar un dispositivo físico para poder aprovechar al máximo las funcionalidades de la aplicación.

C Manual de uso del módulo de arbitraje

Para enviar mensajes desde el módulo de arbitraje, es necesario descargarse el código del proyecto (instrucciones en el Anexo A). Una vez descargado, hay que situarse en el directorio raíz (…/NaoSports) con un terminal y ejecutar un comando de los siguientes:

Comandos para modo individual:

- **Notificación *"score"*:**

Linux: gradlew run -Paction="score" -Pplayer=""
Windows: gradlew.bat run -Paction="score" -Pplayer=""
Ejemplo: gradlew run -Paction="score" -Pplayer="20"

- **Notificación *"challenge"*:**

Linux: gradlew run -Paction="challenge" -Pmessage=""
Windows: gradlew.bat run -Paction="challenge" -Pmessage=""
Ejemplo: gradlew run -Paction="challenge" -Pmessage="El primero en dar una vuelta sobre sí mismo gana 10 puntos!"

Comandos para modo versus:

- **Notificación *"start"*:**

Linux: gradlew run -Paction="start"
Windows: gradlew.bat run -Paction="start"

- **Notificación *"score"*:**

Linux: gradlew run -Paction="score" -Pplayer="" -Prival=""
Windows: gradlew.bat run -Paction="score" -Pplayer="" -Prival=""
Ejemplo: gradlew run -Paction="score" -Pplayer="5" -Prival="2"

- **Notificación *"challenge"*:**

Linux: gradlew run -Paction="challenge" -Pmessage=""
Windows: gradlew.bat run -Paction="challenge" -Pmessage=""
Ejemplo: gradlew run -Paction="challenge" -Pmessage="El primero en dar una vuelta sobre sí mismo gana 10 puntos!"

D Manual de usuario

Ilustración 67 - Manual de usuario

Glosario de términos

Android	Es un sistema operativo basado en el núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes y tablets; con el tiempo se empezó a integrar en otros dispositivos como relojes, televisores y automóviles.
Android Studio	Es un entorno de desarrollo integrado, de las siglas en inglés IDE, para la plataforma Android.
API	Interfaz de programación de aplicaciones. Se le denomina al conjunto de funciones y/o procedimientos ofrecidos por una biblioteca, que pueden ser usados por otro *software* como capa de abstracción.
Array	(Matriz) disposición sistemática de objetos/datos, por lo general en filas y columnas.
C++	Lenguaje de programación estáticamente escrito, multiparadigma, compilado y de uso general. Es considerado un lenguaje de nivel medio, ya que tiene características de lenguajes de alto y bajo nivel. Desarrollado por Bjarne Stroustrup en 1979 en los laboratorios Bell. Agregó características orientadas a objetos, como clases y otras mejoras sobre el lenguaje C.
Choregraphe Suite	*Software* de programación diseñado y desarrollado por la empresa Aldebaran Robotics. Permite a los usuarios crear y editar movimientos y comportamientos interactivos para el robot NAO.
Diagrama de Gantt	Herramienta gráfica que se utiliza para mostrar el tiempo previsto de dedicación de una o varias actividades o tareas, a lo largo de un periodo de tiempo.
Dirección IP	Dirección de protocolo de Internet, etiqueta asignada a cada dispositivo que participa en una red informática que utiliza el Protocolo

	de Internet para la comunicación. Tiene dos funciones principales: *host* o identificación de interfaz de red y dirección de localización.
Framework	Es una plataforma universal y reusable usada para crear aplicaciones, productos y soluciones. Incluyen programas de apoyo, compiladores, librerías de código, interfaces de programación de aplicaciones (APIs) y conjuntos de herramientas que reúnen juntas todos los componentes necesarios para permitir el desarrollo de un proyecto o solución.
Giroscopio	El giroscopio es un dispositivo mecánico o electrónico que sirve para medir, mantener o cambiar la orientación en el espacio de algún aparato o vehículo.
Hardware	La palabra *hardware* se refiere a las partes físicas tangibles de un sistema informático; sus componentes eléctricos, electrónicos, electromecánicos y mecánicos.
IDE	Siglas de *Integrated Development Environment*, es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de *software.*
Interfaz	Una herramienta y concepto que se refiere al punto de interacción entre componentes, y es aplicable en el nivel de hardware y software. Permite que los componentes de un sistema funcionen independientemente, y puedan comunicarse con otros componentes por medio de un sistema de entrada/salida y un protocolo asociado.
Java	Lenguaje de programación de alto nivel orientado a objetos. Su sintaxis deriva de C y C++ pero tiene un modelo de objetos más simple.
JellyBean	Versión 4.3 del sistema operativo Android.
Joystick	Es un dispositivo de entrada que consiste en un mando que los pivota sobre una base y transmite su ángulo o dirección al dispositivo que

	está controlando.
JSON	Acrónimo de *JavaScript Object Notation*, es un formato de texto ligero utilizado para el intercambio de datos.
JavaScript	Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.
Kernel	Es un *software* que constituye una parte fundamental del sistema operativo. Es el principal responsable de facilitar a los distintos programas acceso seguro al *hardware* y de gestionar los recursos.
Linux	Es un *kernel* de sistema operativo libre, que está basado en Unix.
Marketplace	Es un tipo de portal *e-commerce* que ofrece productos o servicios de distintos fabricantes y proveedores, donde las transacciones son procesadas por el gestor del portal.
e-commerce	Cuya traducción es comercio electrónico, o bien negocios por Internet o negocios online, consiste en la compra y venta de productos o de servicios a través de medios electrónicos, tales como Internet y otras redes informáticas.
Marshmallow	Versión 6.0 del sistema operativo Android.
Notificación push	Las notificaciones *push* son mensajes que se envían de forma directa a dispositivos móviles (*smartphones* y/o *tablets*) con sistema operativo iOS, Android, Blackberry y/o Windows Phone.
Patrón de diseño	Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo

	problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.
Python	Lenguaje de programación de alto nivel, cuya filosofía es la legibilidad. Es fuertemente tipado, multiplataforma y soporta orientación a objetos
Realidad aumentada	Es el término que se usa para definir una visión a través de un dispositivo tecnológico, directa o indirecta, de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta en tiempo real.
SDK	Es un conjunto de herramientas de desarrollo de *software* que le permite al programador o desarrollador de *software* crear aplicaciones para un sistema concreto.
Smartphone	Es un tipo de teléfono móvil construido sobre una plataforma informática móvil, con mayor capacidad de almacenar datos y realizar actividades, semejante a la de una minicomputadora, y con una mayor conectividad que un teléfono móvil convencional.
Software	Se conoce como *software* al equipo lógico o soporte lógico de un sistema informático, que comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos que son llamados *hardware*.
Streaming	Es la distribución digital de multimedia a través de una red, de manera que el usuario consume el producto (generalmente archivo de vídeo o audio) en paralelo mientras que se lo descarga.
Tablet	Es un ordenador portátil de mayor tamaño que un *smartphone*.

UI Thread	Es el hilo principal de ejecución de una aplicación Android.
WeBots for NAO	*Webots* es un software para simular robots móviles, ampliamente usado con fines educativos. El proyecto *Webots* fue iniciado en 1996 por el Dr. Oliver Michel en el instituto federal Suizo de Tecnología EPFL en Lausanne. Una de sus principales ventajas es que permite al usuario interactuar con el modelo durante la simulación.
WiFi	Es un mecanismo de conexión de dispositivos electrónicos de forma inalámbrica.

Bibliografía

[1] Isaac Asimov, *"Liar!" 1941*

[2] Liar! (short story), Wikipedia

[3] Karel Čapek, *"R.U.R (Rossum's Universal Robots)" 1920*

[4] The steam-powered pigeon of Archytas – the flying machine of antiquity, Ancient Origins

[5] Ismail al-Jazari, *"The Book of Knowledge of Ingenious Mechanical Devices" 1206*

[6] Prof. Gunalan Nadarajab, *"A Reading of al-Jazari's The Book of Knowledge of Ingenious Mechanical Devices" 2007*

[7] Leonardo Da Vinci, *"L'Uomo Vitruviano" alrededor de 1490*

[8] Michael E. Moran *"The da Vinci Robot". Journal of Endourology, 20(12): 986-990. January 2007*

[9] Jacques de Vaucanson, Wikipedia

[10] Pierre Jaquet-Droz, Wikipedia

[11] Joseph Jacquard, Wikipedia

[12] History of the transistor, Wikipedia

[13] Dartmouth Conferences, Wikipedia

[14] Unimate, Wikipedia

[15] Victor Scheinman, Wikipedia

[16] Haruhiko Asada y Takeo Kanade, "Design of direct-drive mechanical arms" 1981

[17] Rodney A. Brooks y Anita M. Flynn, "Fast, Cheap and Out of Control: A Robot Invasion of the Solar System" 1989

[18] John E. Bares & William "Red" Whittaker, "Dante I & II"

[19] Robotuna Project To Model Real Fish, The Tech

[20] Christopher Kitts, *"Development and Teleoperation of Robotic Vehicles" 2003*

[21] Remote manipulator, Wikipedia

[22] ElectroMechanical Manipulator, Cyberneticzoo

[23] Lunokhod programme, Wikipedia

[24] “Rotex (1993), Robot Technology Experiment on Spacelab D2-Mission”

[25] Los liquidadores robóticos de Chernóbil, Rusadas

[26] Jason, Ocean Explorer

[26] Fong y Thorpe, “Robot as partner: Vehicle Teleoperation with collaborative control” 2002

[27] Rosenthal y Veloso “Using Symbiotic Relationships with Humans to Help Robots Overcome Limitations” 2010

[28] NAOTherapist, UC3M

[29] The State Of Mobile Apps, The Nielsen Company

[30] Pokemon Go España

[31] Android vs iOS market share in 2015, Device Atlas

[32] Nao Robot, Aldebaran

[33] Git Official Page

[34] GitHub Inc.

[35] Android Studio

[36] Vincent Hayward ,*“Haptic interfaces and devices”*

[37] Furby, Hasbro

[38] Roomba, iRobot