UNIVERSIDADE DA CORUÑA

Facultad de Informática
Departamento de Computación

Trabajo de fin de Grado
de Ingeniería Informática

Desarrollo de subsistema de interacción humano-máquina para el ROBOBO 2.0

Autor: Llamas Luaces, Luis Felipe
Tutor: Bellas Bouza, Francisco Javier
Directores: Varela Fernández, Gervasio
Prieto García, Abraham

A Coruña, a 4 de septiembre de 2016.

Información general

Título del proyecto:	**"Desarrollo de subsistema de interacción humano-máquina para el ROBOBO 2.0 "**
Clase de proyecto:	Proyecto de desarrollo en investigación
Nombre del alumno:	Llamas Luaces, Luis Felipe
Nombre del tutor:	Bellas Bouza, Francisco Javier
Nombre de los directores:	Varela Fernández, Gervasio Prieto García, Abraham
Miembros del tribunal:	
Miembros suplentes:	
Fecha de lectura:	
Calificación:	

Dr. Bellas Bouza, Francisco Javier

CERTIFICA

Que la memoria titulada **"Desarrollo de subsistema de interacción humano-máquina para el ROBOBO 2.0"** ha sido realizada por Luis Felipe Llamas Luaces con D.N.I. 48113017-F bajo la dirección del Dr. Francisco Javier Bellas Bouza. La presente constituye la documentación que, con mi autorización, entrega el mencionado alumno para optar a la titulación de Ingeniería en Informática.

A Coruña, a 4 de septiembre de 2016.

Firmado:

A mis padres.

Agradecimientos

A mi familia y amigos por todo su apoyo.

A D. Francisco Bellas Bouza, Gervasio Varela Fernández y Abraham Prieto García, cuyos consejos y recomendaciones han sido imprescindibles en este proyecto, y sin los cuales no habría llegado a su fin.

Resumen

En este Trabajo de Fin de Grado (TFG) se desarrollará un subsistema de interacción humano robot para una plataforma robótica controlada desde un teléfono inteligente (smartphone) basado en Android, denominada ROBOBO.

ROBOBO es el resultado de la combinación de un smartphone con una plataforma motorizada desarrollado por el GII (Grupo Integrado de Ingeniería de la UDC) con el objetivo de conseguir un robot de bajo coste, con multitud de sensores y métodos de conexión inalámbrica.

El ROBOBO! Framework es el marco de trabajo empleado para el desarrollo de aplicaciones para el ROBOBO, y proporciona diferentes funcionalidades útiles a la hora del desarrollo, como la interfaz de control de la base motorizada, en forma de módulos.

El objetivo de este TFG es la expansión de dicho framework mediante el desarrollo de un subsistema de interacción entre humanos y el robot, que proporcione al programador diferentes posibilidades de interacción entre el robot y el usuario final.

Se desarrolló dicho subsistema de manera modular siguiendo una analogía con los que serían los sentidos del robot: vista, oído y tacto, voz y mensajería. Estos sentidos fueron condensados en cinco librerías diferentes que conforman el subsistema de interacción. Adicionalmente se desarrollaron varios ejemplos para demostrar el funcionamiento del subsistema.

Palabras clave:

- √ Robótica autónoma
- √ Robótica educativa
- √ Interacción humano robot
- √ Android

Índice general

Página

Índice de figuras

Figura **Página**

Capítulo 1
Introducción

1.1. Motivación

El Grupo Integrado de Ingeniería (GII) de la Universidade da Coruña (UDC) viene desarrollando una línea de investigación en Robótica y Cognición desde hace más de 15 años, que tiene como objetivo general desarrollar sistemas reales que puedan responder con el mayor grado de autonomía posible a las condiciones cambiantes del medio, sin tener que precisar de la ayuda de un operador o programador para establecer las nuevas estrategias necesarias para la consecución de la tarea encomendada. Esta línea de investigación tiene otra línea derivada que se denomina Mecanismos Cognitivos, en la cual se centra en el estudio e implementación de funcionalidades cognitivas de alto nivel en robots autónomos, utilizando los procesos cognitivos humanos como inspiración para desarrollar un mecanismo cognitivo completo para un robot.

Uno de los proyectos asociados a la línea de Mecanismos Cognitivos es DREAM [1] (Deferred Restructuring of Experience in Autonomous Machines), financiado por la Unión Europea, y que se centra en la integración de procesos cognitivos basados en el sueño dentro de una arquitectura cognitiva para robots autónomos. El objetivo de utilizar estos procesos reside en la capacidad de consolidar el aprendizaje adquirido durante la operación en "tiempo de vida"que se ha visto que existe en los procesos asociados al sueño en el cerebro humano. De esta forma, la experiencia adquirida por el robot se analiza y procesa en otra escala temporal que proporciona nuevas representaciones de más alto nivel.

Uno de los principales problemas a solucionar en el proyecto DREAM se centra en el aprendizaje a partir de la interacción con humanos. En este sentido, se

requieren numerosos datos de interacción en tiempo real en diversas situaciones para poder mejorar la arquitectura, y para lograrlos se ha diseñado la iniciativa «adopt a robot». Esta iniciativa se centra en proporcionar a los centros educativos un robot de bajo coste que contenga la arquitectura cognitiva desarrollada en el DREAM, de modo que los alumnos puedan interactuar con el robot libremente y se consiga gran variedad de datos para mejorar dicha arquitectura. Este robot está en desarrollo actualmente en el GII bajo el nombre de ROBOBO.

Para poder emplear el ROBOBO en el proyecto DREAM es necesario que el robot cuente con capacidades de interacción con humanos, por ello, el objetivo de este proyecto es el desarrollo del framework de interacción que dotará al ROBOBO de la capacidad de interactuar con las personas, específicamente centrado en niños.

El ROBOBO se basa en el uso de una plataforma motorizada y sensorizada, llamada ROB en el sistema, que se conecta a un Smartphone, llamado OBO. Este tipo de arquitectura presenta numerosas ventajas que resultaron interesantes a la hora de escogerlo:

- Bajo coste de la plataforma
- Gran capacidad de sensorización, cualquier smartphone de gama media cuenta con acelerómetros, giroscopios, gps, cámara y micrófono, y de comunicaciones, wifi, gsm y bluetooth.
- Robot fácilmente actualizable, mediante el cambio de smartphone, sin necesidad de adaptar el software antiguo.
- Gran disponibilidad de smartphones, todos los usuarios potenciales poseen uno, además Android, sistema operativo escogido para el sistema, cuenta con un 82.8 % de la cuota de mercado.

1.1.1. Robobo 1.0

La primera versión de la plataforma desarrollada por el GII fue el ROBOBO 1.0 (Figura 1.1), que sirvió como versión conceptual para comprobar la viabilidad del sistema plataforma + smartphone, y presentaba las siguientes características:

- 8 LEDs (diodos emisores de luz) RGB para interactuar con el usuario, que cambiaban de color con la proximidad de objetos.

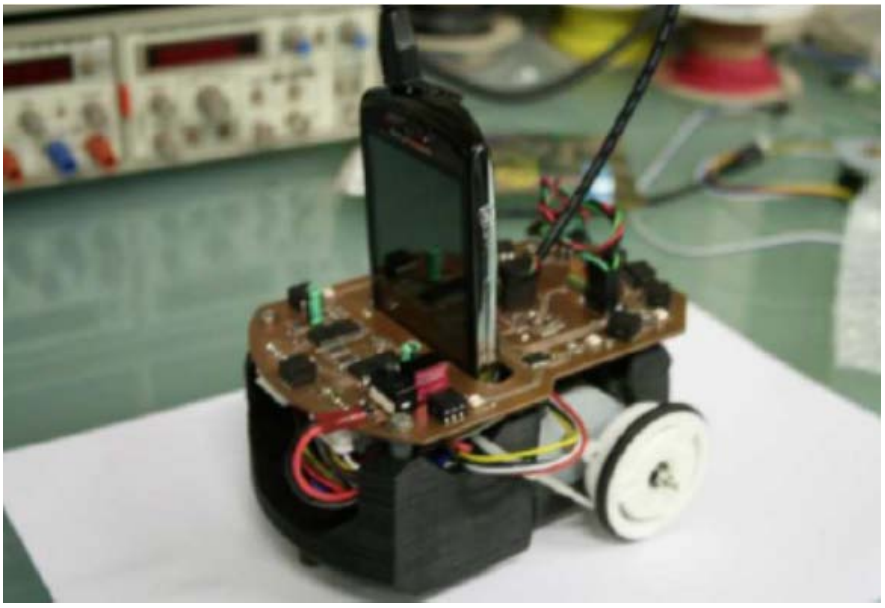

Figura 1.1: Robobo 1.0

- 9 sensores IR de proximidad para proporcionar capacidad de movimiento autónomo: 2 en la parte frontal, 4 en los laterales y los últimos tres en la parte trasera; de manera que proporcionaba una visión general del entorno.
- 2 motores paso a paso (convierten impulsos eléctricos en desplazamientos angulares discretos, es decir, pueden avanzar un ángulo concreto en función de la señal recibida) que aplicaban movimiento a las ruedas, con un paso de 1/8, con gran precisión de giro.
- Compatibilidad con dispositivos Android.
- Capacidad de interacción con otros dispositivos ROS.
- Conexión USB para la comunicación con el teléfono inteligente.

1.1.2. Robobo 2.0

La segunda versión, el ROBOBO 2.0 (figura 1.2), trata de solventar las carencias de la primera versión y añadir ciertas mejoras. Tras este rediseño, las características del robot pasaron a ser:

Figura 1.2: Robobo 2.0

- LEDs: Pasan a ser 9, 5 de ellos forman una matriz de comunicación en la parte delantera del robot para dar una información avanzada mediante colores y formas.
- Motores de las ruedas: Pasan a ser motores de corriente continua, menos voluminosos y más eficientes energéticamente. Cada motor cuenta con un encoder magnético, que transforma el movimiento del motor en pulsos eléctricos, que pueden ser interpretados por la parte software del robot.
- Comunicación: Se elimina la comunicación USB a favor de una conexión Bluetooth, lo cual permite utilizar diferentes smartphones con distintas posiciones de los puertos.
- Plataforma universal para smartphones: Se incluye un adaptador universal para teléfonos móviles, montado sobre una plataforma motorizada que permite el movimiento del smartphone sobre el chasis del robot. Esta plataforma tiene dos grados de libertad, rotación e inclinación.
- Sensores: Modificadas tanto el tipo, la cantidad y la posición. Se colocaron en la siguiente disposición (figura 1.3):

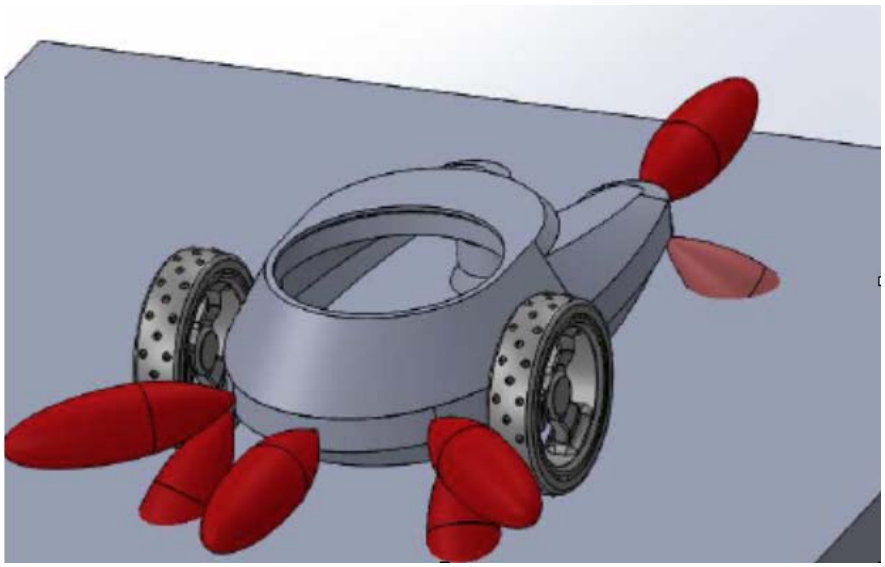

Figura 1.3: Disposición de los sensores en el ROBOBO 2.0

- 3 sensores para la detección de colisiones en la parte delantera; uno frontal y los otros dos cercanos a las ruedas, de tal forma que el ROBOBO no pudiera chocar, frontalmente con ninguna superficie.
- 2 sensores inclinados hacia abajo para la detección de caídas, en la parte delantera, colocados cerca de las ruedas, es decir, la zona con mayor peligro de caídas.
- 2 sensores de caídas y 2 de colisiones, de forma similar a los anteriores, en la zona anterior del robot.

- Unión sensores-microcontrolador: se han cambiado las resistencias pull-up anteriores por un multiplexor. Se consigue, así, un único elemento de muchas entradas y una única salida capaz de permitir la transmisión de una, y solo una, de las entradas hacia la salida.

- Microcontrolador: PIC32MX534F064H, de la familia de microcontroladores de 32 bits, económico, con alta capacidad de procesamiento, diversidad de interfaces de comunicación y multitud de puertos de entrada y salida; además de que su entorno de desarrollo y su compilador son gratuitos. Este microcontrolador se programa mediante el protocolo de comunicación I2C y logra el contacto entre sensores y microcontrolador mediante buses línea de reloj “SCL” y buses línea de datos “SDA”.

Esta versión del ROBOBO, aún en pleno desarrollo, es la que se utilizará en este trabajo de final de grado que se centrará en el desarrollo de librerías de interacción básica entre el robot y humanos, enfocándose específicamente en interacción por voz, táctil, sonido e imagen, dando lugar a módulos simples que más tarde podrán ser combinados en comportamientos interactivos más complejos. El desarrollo fundamental de este trabajo se realizará en Android, ya que inicialmente el ROBOBO solo soportará teléfonos con este sistema operativo.

1.2. Objetivos

El objetivo de este proyecto, como se comentó en la sección anterior, es el *desarrollo de un framework de interacción básica entre el ROBOBO y humanos.* Estas librerías serán desarrolladas en Android, sistema operativo que soporta el ROBOBO a día de hoy. Se desarrollarán 5 librerías Android, cada una enfocada a un tipo de interacción diferente con el robot:

- Librería de interacción por voz
- Librería de interacción por sonido
- Librería de interacción táctil
- Librería de interacción por imagen
- Librería de interacción mediante mensajes

Finalmente se implementarán una serie de ejemplos para demostrar el funcionamiento de las diferentes librerías de forma conjunta.

1.3. Estructura de la memoria

La memoria del trabajo está estructurada en 6 capítulos, siendo el primero de ellos esta introducción.

- Segundo capítulo: ***Fundamentos teóricos***, se habla sobre el ROBOBO! Framework, marco de desarrollo empleado para el desarrollo de los módulos que componen este trabajo, cómo interactúa con las diferentes partes del sistema ROBOBO y las tecnologías en las que se basa.

- Tercer capítulo: ***Antecedentes***, se hablará sobre el campo de la interacción humano robot y se expondrán ejemplos de los diferentes modos de interacción que surgen en los diferentes campos de la robótica.
- Cuarto capítulo: ***Fundamentos Tecnológicos*** se dará un repaso a las diferentes tecnologías empleadas a la hora del desarrollo de los módulos de interacción.
- Quinto capítulo: ***Desarrollo***, se elabora el desarrollo técnico del trabajo, el modelo conceptual, la arquitectura escogida, la metodología seguida y los detalles de implementación.
- Sexto capítulo: ***Resultados y pruebas***, se comentan los resultados del desarrollo y se exponen las aplicaciones de ejemplo desarrolladas utilizando los módulos de interacción. También se exponen los problemas conocidos aún sin solución.
- Séptimo capítulo: ***Conclusiones***, en este último capítulo se comenta el resultado general del trabajo así como el trabajo futuro a desarrollar.

1.4. Herramientas utilizadas

Para el desarrollo de las librerías y aplicaciones Android que conforman el presente trabajo, se ha empleado el entorno de desarrollo **Android Studio**, basado en el IntelliJ IDEA de Jetbrains. Actualmente Android Studio es el entorno de desarrollo oficial para Android.

Para la creación de los diagramas se ha empleado el software **StarUML**.

Una unidad del **robot ROBOBO 2.0** para realizar pruebas.

Un Smartphone **BQ Aquaris M5** para realizar pruebas.

Capítulo 2
Fundamentos teóricos

En este capítulo se introduce la base teórica sobre la que se desarrollará el trabajo, en concreto acerca del ROBOBO! Framework, un framework de desarrollo creado específicamente para el ROBOBO.

2.1. ROBOBO! Framework

El entorno ROBOBO está formado por la base robotizada ROB y el Smartphone OBO(figura 2.1) y su estructura puede verse en la figura 2.2. En dicha figura se puede observar cual es el software contenido en las diferentes partes del hardware.

El ROBOBO! Framework es el marco de trabajo empleado para desarrollar el software para el ROBOBO, y cuyas funcionalidades se expandirán mediante los módulos de interacción que se desarrollarán en este trabajo. El framework está enfocado al desarrollo de aplicaciones Android, sistema operativo del que se hablará en la sección 2.1.1, mediante el lenguaje de programación Java. El framework, como se ve en la figura 2.2 está contenido dentro del OBO.

Las *Android Native App* y *ROS Embedded App*, de la figura, representan la aplicación final que el usuario instalará en su teléfono móvil(OBO) y está construida sobre el framework, utilizando los diferentes módulos que este ofrece.

Dentro del framework se pueden encontrar diferentes módulos que proporcionan funcionalidades para el desarrollo de aplicaciones. Dos de ellos resultan especialmente importantes:

Figura 2.1: ROBOBO completo,Base robotizada(ROB) + smartphone(OBO)

ros-module:

El *ros-module*, proporciona una interfaz para pasar de acciones nativas del ROBOBO a conceptos propios de ROS, servicios o temas. Este modulo da acceso desde ROS a:

- Las funcionalidades del ROB
- Los módulos del ROBOBO
- La cámara del dispositivo
- Los datos de los sensores del Smartphone

Permite desarrollar tanto aplicaciones remotas(*ROS Remote App* en la figura), que no están alojadas en el smartphone pero hacen uso de los servicios o temas ejecutándose en el ROBOBO, como aplicaciones ROS nativas(*Ros Embedded App*).

Más adelante, en la sección 2.1.2, se hablará con mas detalle de ROS, pese a que no ha sido empleado para el desarrollo de este trabajo.

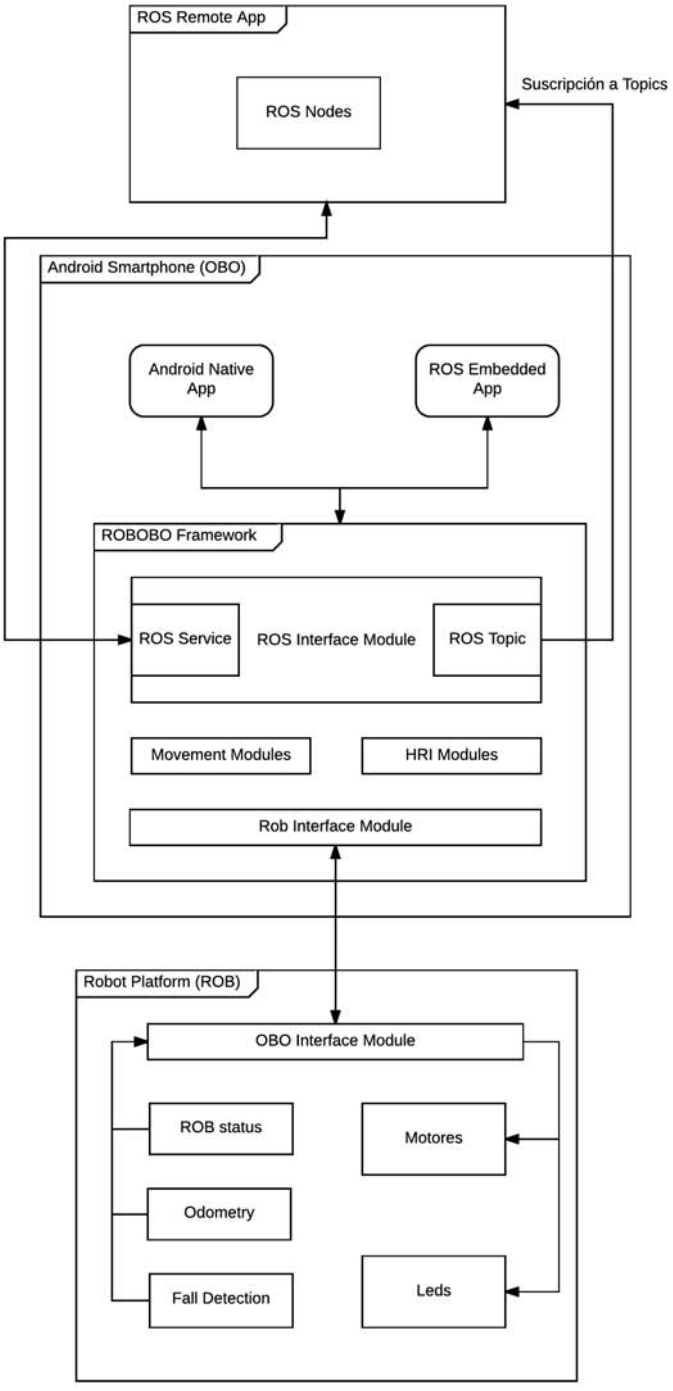

Figura 2.2: Entorno ROBOBO

rob-interface-module:

Este modulo nos permite obtener la interfaz de control de bajo nivel del robot, la clase *IRob*(figura 2.3) , mediante el uso de esta interfaz se puede:

Recibir mensajes periódicos del estado del ROB:

- Datos de los sensores de infrarrojos
- Información de detección de caídas
- Datos del movimiento de los motores
- Estado de la batería

Enviar comandos al ROB:

- Movimiento de los motores
- Cambio de colores de los LEDs
- Modificación de la periodicidad de los mensajes de estado
- Cambiar el modo del ROB

Existen módulos para facilitar el uso del robot y evitar tener que usar la clase IRob, un ejemplo es el *rob-movement-module*(figura 2.4) que proporciona una interfaz de alto nivel para manejar los movimientos del robot.

En la base ROB, *Robot Platform* en la figura 2.2, se encuentra un software escrito en C, que representa el firmware de ROB y se encarga de tres tareas diferentes:

- Gestionar la comunicación (vía Bluetooth) con el Smartphone a través de la obo-interface. Esta comunicación se hace a través de un protocolo binario muy reducido que permite al smartphone enviar comandos al ROB, que este ejecuta en cuanto los recibe. Además, el ROB envía periódicamente (con una periodicidad programada por el OBO) información de los sensores y dispositivos al OBO, para que desde el smartphone se pueda conocer, en tiempo real, el estado del ROB.
- Controlar los diferentes dispositivos hardware de los que dispone: sensores infrarrojos, motores y luces LED.

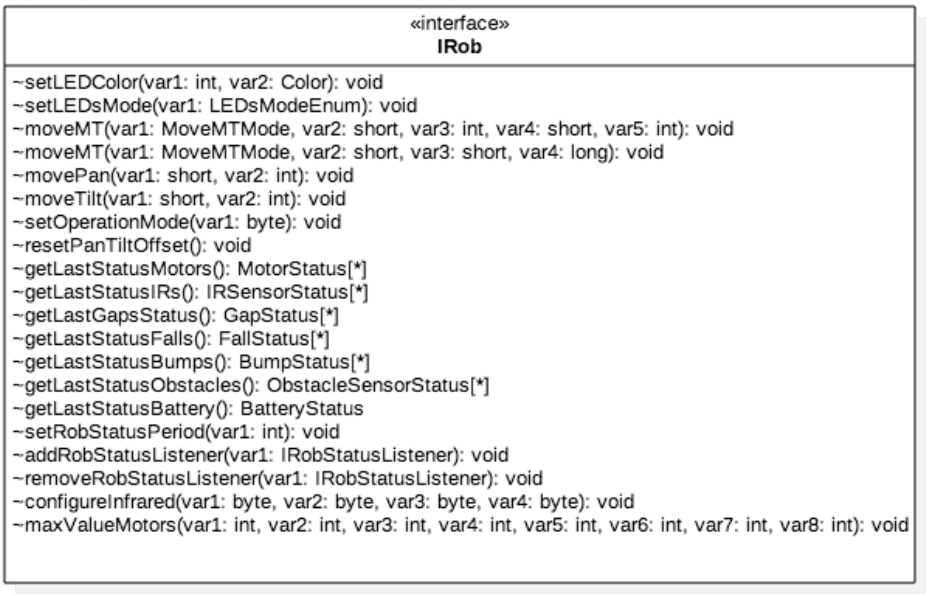

Figura 2.3: Clase IRob

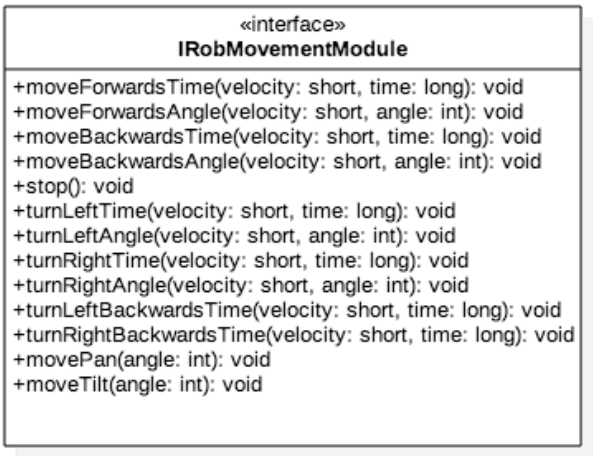

Figura 2.4: Clase IRobMovementModule

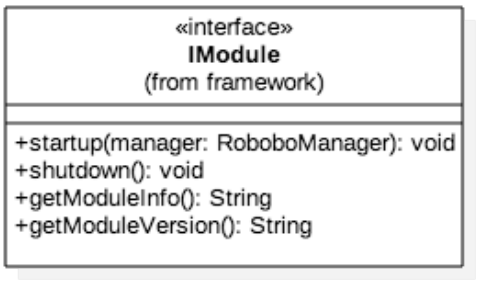

Figura 2.5: Interfaz IModule

- Ejecutar una serie de módulos de control para la detección de caídas o impactos contra objetos.

A la hora de desarrollar nuevas funcionalidades el ROBOBO! Framework sigue una filosofía modular, para ello proporciona al programador una interfaz común, *IModule*(figura 2.5), que todo módulo que vaya a residir dentro del framework debe implementar. Esta interfaz proporciona las cabeceras de los métodos a los que el framework llamará tanto al cargar cómo descargar los módulos.

2.1.1. Android

Android es un sistema operativo de código abierto desarrollado por Google enfocado a dispositivos móviles. Actualmente es el sistema operativo más extendido en cuanto a terminales móviles, ocupando un 82.8 % de la cuota de mercado en el segundo cuarto de 2015 (IDC, Aug 2015) y contando la tienda de aplicaciones de la plataforma, la Play Store, a Junio de 2016 con 2.2 millones de aplicaciones disponibles. Dados estos datos de extensión y con la capacidad de cómputo y sensorización de los teléfonos este sistema como base para el ROBOBO.

Android está construido sobre un núcleo Linux sobre el cual corre el *Android Runtime*(ART) o en versiones anteriores del sistema operativo la máquina virtual *Dalvik*, siendo estos los entornos de ejecución de las propias aplicaciones Android.

2.1.2. ROS

El Sistema Operativo Robótico [2] , por sus siglas en inglés ROS, es un framework libre de desarrollo de software para robots que provee los servicios que se podrían esperar de un sistema operativo, tales como la abstracción del hardware, la gestión de dispositivos a bajo nivel, implementación de funcionalidades comunes, paso de mensajes entre procesos y gestión de paquetes, sin embargo, ROS no es un sistema operativo y debe ejecutarse sobre un entorno Linux.

ROS cuenta con una amplia comunidad a lo largo del mundo que contribuye de manera activa al proyecto. El repositorio de software de ROS contiene multitud de paquetes que proporcionan funcionalidades de forma simple, de manera que el programador pueda centrarse en el desarrollo del control de su robot sin tener que preocuparse de implementar los algoritmos y técnicas más comunes.

La funcionalidad de utilizar ROS en el ROBOBO esta provista por el módulo del ROBOBO! Framework *ros-module* que emplea la librería **rosjava** para este cometido.

Capítulo 3

Antecedentes

En este capítulo se introduce el tema de la Interacción humano robot, y el diseño conceptual del sistema de interacción desarrollado para la plataforma ROBOBO.

3.1. Interacción humano robot

Desde finales de la década de los 90, la interacción entre humanos y robots ha cobrado importancia, creando un nuevo campo de investigación en la ciencia [3], la interacción humano-robot (HRI por sus siglas en inglés). El campo de la HRI busca entender, modelar y evaluar las diferentes modalidades de interacción entre las personas y los robots. La comunicación entre humanos y robots puede dividirse en dos categorías generales:

- Interacción remota
- Interacción próxima

La interacción remota suele ser referida como control supervisado o teleoperación, dependiendo de si el robot es autónomo con supervisión de un humano, que interviene en caso de necesidad, o si el robot es controlado por el humano directamente. Este tipo de interacción puede verse en robots de tipo industrial o en vehículos autónomos, cómo los llamados Drones del ejército.

La interacción próxima es aquella en la que el robot interactúa directamente con el humano, llegando incluso a haber interacción física. Este tipo de interacción incluye elementos emotivos y sociales, y se puede encontrar en, por ejemplo, los

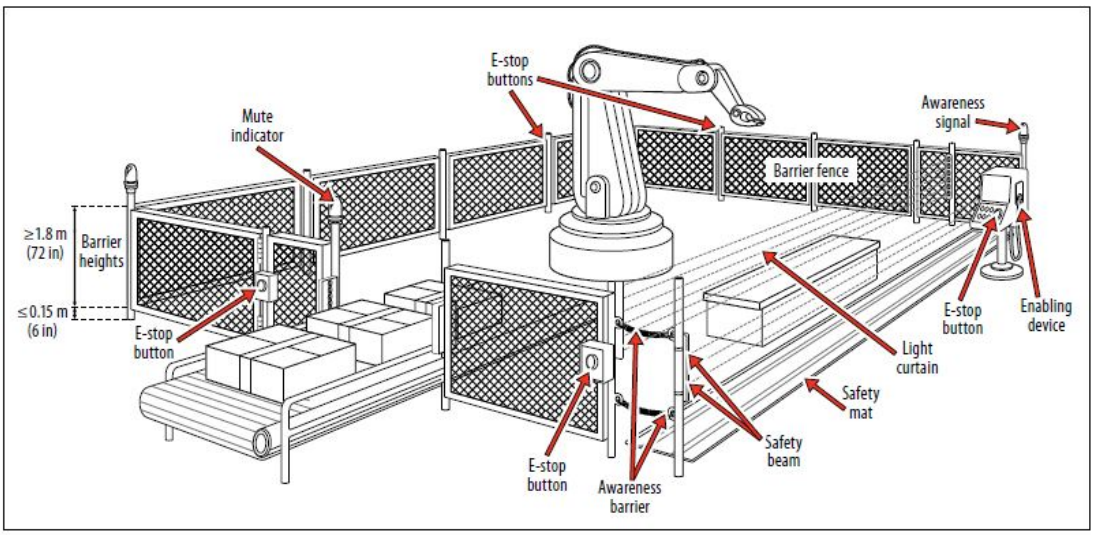

Figura 3.1: Esquema de una jaula de seguridad para un robot industrial

robots asistenciales o educativos. Este tipo de interacción es la que se tratará en este trabajo, en el cual se diseñarán diferentes sistemas de interacción para la plataforma ROBOBO.

3.1.1. HRI en la industria

Generalmente, en los procesos industriales, la interacción entre los robots y los operadores suele ser remota, los sistemas se programan para realizar una tarea y el humano solamente interviene en caso de necesidad, sin embargo pueden darse casos de interacción próxima con los robots. Uno de estos casos podría ser el aprendizaje de tareas mediante demostración, proceso mediante el cual, un operador humano realiza una tarea, por ejemplo moviendo manualmente el brazo de un robot, para que el controlador aprenda a realizar esa tarea. Este tipo de interacción permite que los robots aprendan comportamientos de alto nivel difícilmente programables.

Sin embargo a la hora de realizar tareas de forma cooperativa entre robots y humanos, la interacción cercana con robots industriales conlleva riesgos importantes, como pudo verse en el accidente del 2015 [4] en la planta de Volkswagen cerca de Kassel, Alemania, en el que un operario fue golpeado por un brazo industrial durante su instalación, resultando en la muerte del técnico. Para evitar esta clase de accidentes, se está buscando la consciencia del entorno en los mani-

puladores robóticos, para poder adaptar sus reacciones al contexto actual. Este tipo de consciencia no solo disminuye los riesgos de operación, sino que también disminuye los costes espaciales, ya que los robots no requerirían de jaulas de seguridad (figura 3.1), también la productividad se vería afectada positivamente, ya que tareas imposibles de realizar para un robot y para un humano individualmente, pueden llevarse a cabo mediante la llamada robótica cooperativa. En *Cooperative Tasks between Humans and Robots in Industrial Environments* [5] presentan un sistema de robótica cooperativa en el que un operador y un robot colaboran de manera cercana para llevar a cabo diferentes tareas, de manera que el robot realiza las tareas repetitivas y peligrosas, mientras que el humano lleva a cabo las tareas que requieren de cierta precisión o inteligencia con la que no cuenta el robot. En este sistema el operador lleva un traje de posicionamiento, que permite al robot conocer su posición, pudiendo así adaptar sus movimientos de manera que el humano no corra riesgos.

3.1.2. HRI en robots asistenciales

Uno de los campos en los que la interacción entre humanos y robots cobra mucha importancia es en el nicho de los robots asistenciales. Los robots asistenciales, también llamados de servicio, son definidos por la federación internacional de robótica cómo *Robots que operan de forma total o semiautónoma para realizar servicios útiles para el bienestar de humanos y equipamiento, excluyendo las operaciones de manufactura* [6]. En esta definición se diferencia entre dos tipos de robots asistenciales:

- Robots personales
- Robots profesionales

Los robots personales son aquellos que se utilizan para labores no comerciales, generalmente por personas sin perfil técnico. Por ejemplo, sillas de ruedas eléctricas, robots de asistencia de movilidad, o aspiradoras automáticas.

Los robots profesionales son aquellos utilizados para realizar tareas de asistencia en un entorno comercial, generalmente manejados y supervisados por un personal especializado. Por ejemplo robots de limpieza automatizados para zonas públicas, robots de mensajería en oficinas u hospitales, robots anti-incendios,robots quirúrgicos y de rehabilitación en hospitales o los robots terapéuticos.

Figura 3.2: Robot terapeutico Paro

En los robots terapéuticos se pueden encontrar múltiples formas de interacción, por ejemplo, el robot Paro [7] (figura 3.2) es un robot terapéutico con forma de bebé foca, utilizado con éxito en terapias contra la Demencia, que busca una interacción emocional con el paciente, para ello cuenta con cinco tipos de sensores diferentes, táctiles, auditivos, de temperatura, de luz y posturales. Los pacientes realizan una interacción con el robot cómo la que tendrían con un animal, y el robot responde acorde a los estímulos que recibe. Esto, en conjunto con la forma física del robot, más semejante a un animal de peluche que a una máquina, permite al paciente desarrollar emociones. El robot NAO [8](figura 3.3) es otro de los robots que se están empleando con éxito en tareas asistenciales, tanto de manera terapéutica, como robot de relaciones publicas o para tareas de educación. El robot cuenta con una amplia variedad de sensores y actuadores que le permiten interactuar de diversas formas con el usuario:

- Cámaras: Permiten reconocimiento de caras y procesado de imagen.
- Sensores táctiles y de presión: Permiten una interacción física con el robot.
- Altavoces: Permiten al robot producir diversos sonidos y hablar.
- Micrófonos: Permiten reconocer habla y ubicar el origen de los sonidos espacialmente.
- Sensores de distancia: Permiten detectar la distancia a los objetos.

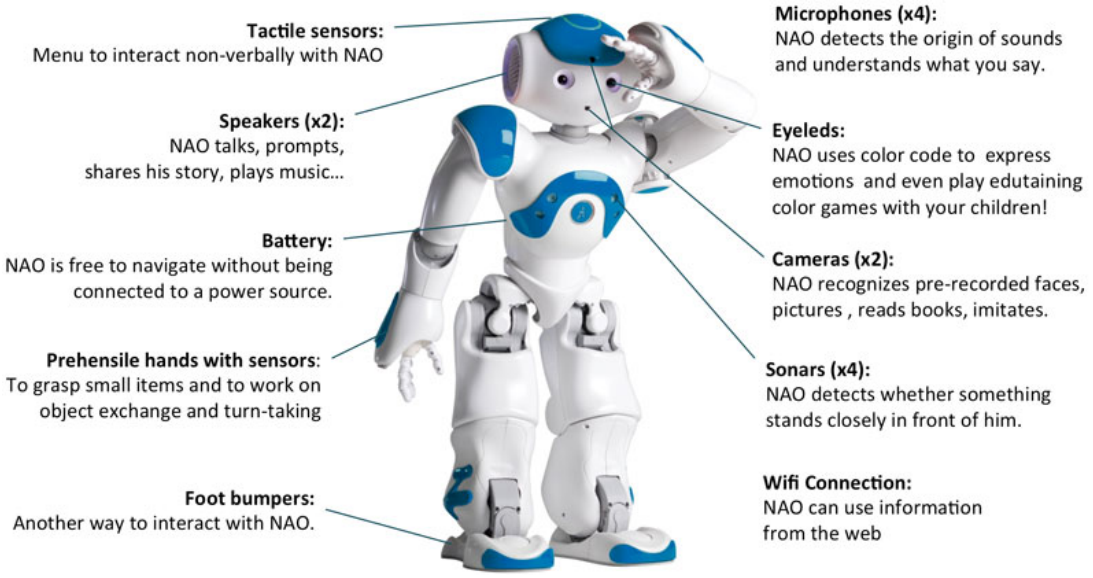

Figura 3.3: Esquema de características del robot NAO

- 25 Grados de libertad: Permiten al robot interactuar fisicamente con su entorno y realizar comunicación no verbal.
- Unidad de medición inercial: Permite detectar aceleraciones y giros.

En el caso de los robots médicos, los quirúrgicos especificamente, la interacción no suele pasar de la teleoperación del robot, es decir, el robot se controla como si fuera una extensión del cirujano. El robot más conocido en este campo es el sistema quirúrgico Da Vinci (figura 3.4), utilizado en cirugía de precisión. En este robot el cirujano controla los diferentes brazos del aparato desde una consola que cuenta con controles con feedback háptico, es decir, el operador no solamente mueve el brazo, sino que siente lo que hay al final del mismo, la presión ejercida y la resistencia al movimiento, dando al cirujano el tacto necesario para realizar las diferentes tareas que se realizan en una operación, como coser o cortar, de forma natural y con un alto grado de precisión, ya que permite escalar los movimientos del cirujano.

3.1.3. HRI en robots de entretenimiento

Se entiende por robots de entretenimiento aquellos cuya finalidad no es más que divertir al usuario. Dentro de esta categoría podríamos incluir a los robots

Figura 3.4: Sistema quirúrgico daVinci

mascota, que generalmente realizan una interacción de alto nivel con el usuario. Por ejemplo, uno de los robots más relevantes en el ámbito de los robots mascota es el desarrollado por Sony, Aibo [9] (figura 3.5) , cuya finalidad era comportarse como un perro. En este robot podemos encontrar múltiples tipos de interacción, interacción física en forma de movimientos perrunos, reconocimiento facial a través de cámaras, a través de caricias utilizando los sensores táctiles, y en las últimas versiones del robot mediante una matriz de leds situada en la cara del aparato, que permite poner diferentes expresiones en función del "humor"del robot. Mediante todas estas capacidades motoras y sensoriales, se puede interactuar con una unidad Aibo de manera semejante a la que se tendría con un perro real. Otro ejemplo de mascota robótica es el Bandai SmartPet (figura 3.6) , un robot pensado para utilizar junto un smartphone IPhone, que es colocado en la cabeza del robot y provee múltiples formas de interacción, reconocimiento de gestos mediante la cámara frontal, reconocimiento de sonido utilizando el micrófono y reconocimiento de gestos táctiles en la pantalla.

Figura 3.5: Robot mascota Aibo de Sony

Figura 3.6: Robot mascota Smartpet de Bandai

Figura 3.7: Beebot

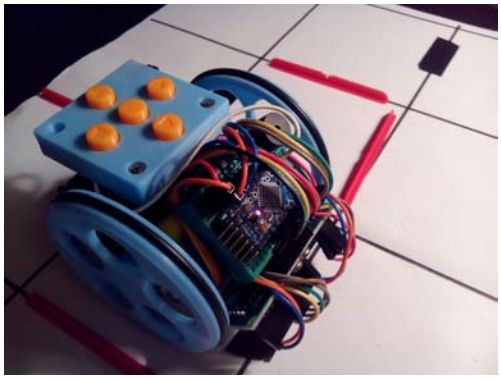

Figura 3.8: Escornabot

3.1.4. HRI en robots educativos

En los últimos años el uso de robots con fines educativos, si bien ya eran usados de manera educativa en enseñanza superior, ha tomado impulso en la educación primaria y secundaria, apareciendo múltiples robots enfocados a este tipo de mercado. La interacción con esta clase de robots puede darse de diferentes formas según el público objetivo, dependiendo principalmente del rango de edades del mismo.

Enfocados a la educación infantil nos podemos encontrar con robots cómo el BeeBot(figura 3.7) o su alternativa libre, el Escornabot(figura 3.8), el cometido de estos robots es introducir los niños a la programación, en forma de pensamiento

secuencial, y a la resolución de problemas. En esta clase de robots la interacción está limitada a la introducción de comandos en la botonera de la parte superior del robot, que serán traducidos a movimientos del robot posteriormente.

En la educación primaria y secundaria los robots utilizados ya adquieren una mayor complejidad y suelen emplearse para una introducción real a la programación, generalmente utilizando lenguajes gráficos de muy alto nivel como Scratch [10]. Uno de los robots más relevantes en este ámbito es el kit Mindstorms de Lego (figura 3.9) , que no solo permite programar el robot, sino también construirlo. El Lego Mindstorms cuenta con diferentes sensores y actuadores que ofrecen diversas maneras de interacción con el robot:

- Motores: Permiten el movimiento del robot de manera relativamente precisa
- Sensores de distancia: Permiten medir distancias y actuar en consecuencia a los datos medidos
- Sensores de luz ambiente: Miden la intensidad de la luz del entorno
- Sensores de color: Permiten detectar los colores básicos
- Unidad inercial: Permite medir giros en el plano horizontal

En la llamada educación especial la robótica también está siendo empleada con éxito, por ejemplo el robot NAO (figura 3.3), del que se habló en la sección anterior, se utiliza para educar a niños con trastornos de espectro autista. El éxito de este tipo de educación viene dada debido a que la interacción con el robot, al ser programada, resulta predecible para el alumno, creando un entorno estable y pautado que resulta óptimo en este tipo de trastornos. La interacción en este caso se da en forma de movimientos preprogramados y mediante los leds de los ojos del robot, que cambian de color según las emociones que intenta expresar el robot, lo cual ayuda al alumno a ejercitar uno de las mayores dificultades dadas por el autismo, la dificultad de establecer relaciones empáticas.

Entre otros ejemplos de robots educativos podemos encontrar el Zowi(figura 3.10) de la empresa española BQ. Zowi es un robot educativo enfocado a niños, cuanta con un sensor de ultrasonidos para medir distancias, un micrófono que le permite detectar sonidos, una pequeña matriz de leds a modo de boca, que permite comunicarse mediante expresiones faciales, y con cuatro servos posicionados

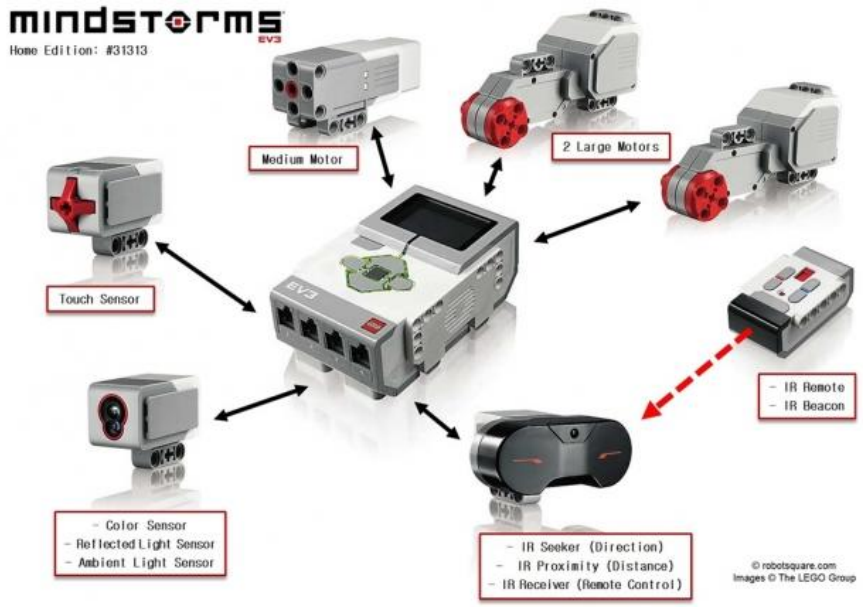

Figura 3.9: Kit básico Lego Mindstorms EV3

en forma de dos piernas que permiten moverse al robot. El robot viene con varios juegos para niños instalados de serie, y cuanta además con una aplicación de control para smartphones. Zowi puede ser empleado para enseñar programación a niños con un lenguaje gráfico similar a Scratch [10], Bitbloq [11].

Otro robot educativo es el Thymio(figura 3.11). Este robot cuenta con gran variedad de sensores, de distancia, acelerómetros, temperatura, control por infrarrojos, seguidores de lineas y micrófono, y con dos ruedas compatibles con las piezas de lego. La interacción con el usuario se realiza a través de los 39 leds RGB situados por todo el chasis, 5 teclas capacitivas que se encuentran en la parte superior del robot y los sensores citados anteriormente. El robot está pensado para enseñar programación a los niños de diferentes rangos de edad, para ello se pueden emplear 3 lenguajes diferentes de programación, Visual Programming Language (VPL) para los niños a partir de 6 años, Blockly, un lenguaje de programación gráfico semejante a Scratch desarrollado por Google, para niños a partir de 9 y programación textual mediante el software Aseba Studio para niños a partir de 12 años.

Dash and Dot (figura 3.12) son una pareja de robots educativos fabricados por Wonder Workshop enfocados a niños de poca edad. Dash cuenta con una plataforma motorizada equipada con sensores de distancia,«orejas» motorizadas,

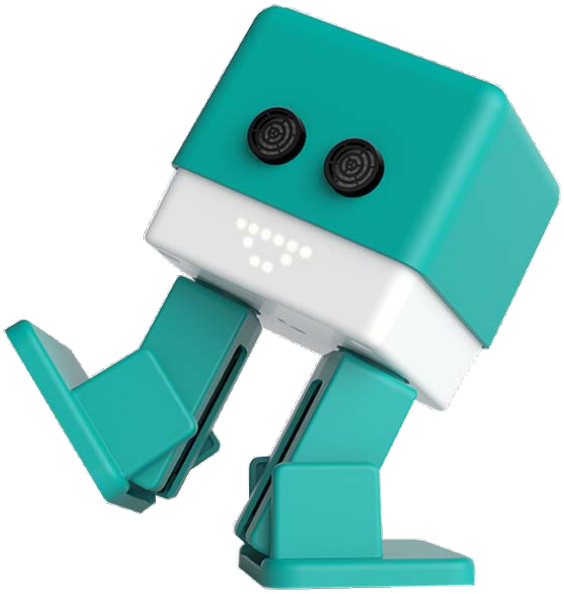

Figura 3.10: Robot Zowi de BQ

tres micrófonos, un altavoz, un «ojo circular» con 12 leds blancos, leds rgb en las «orejas» y «pecho». Dot en cambio, es una alternativa más económica que solo incluye la «cabeza» de Dash. Gracias a la gran cantidad de actuadores, Dash, y en menor medida Dot, cuentan con diversas formas de interacción con los niños, desde el movimiento del robot, habla a través del altavoz, gestos con los leds del ojo e incluso, mediante accesorios vendidos de forma separada, tocando un pequeño xilófono o lanzando pelotas con una pequeña catapulta, Dot además cuenta con un accesorio «orejas de conejo» que le permite suplir la falta de plataforma motorizada y moverse. Estos robots están pensados para enseñar programación a los niños, y cuentan con varias aplicaciones diseñadas para este cometido, *Path*, que permite a los niños dibujar la ruta que desean que tome el robot, *Wonder*, un lenguaje visual de programación y *Blockly*, el lenguaje de programación por bloques desarrollado por Google.

3.1.5. Conclusión

Cómo se pudo apreciar en las secciones anteriores de este capítulo, el campo de la interacción humano robot es amplio y crucial en la evolución de la robótica

Figura 3.11: Robot Thymio

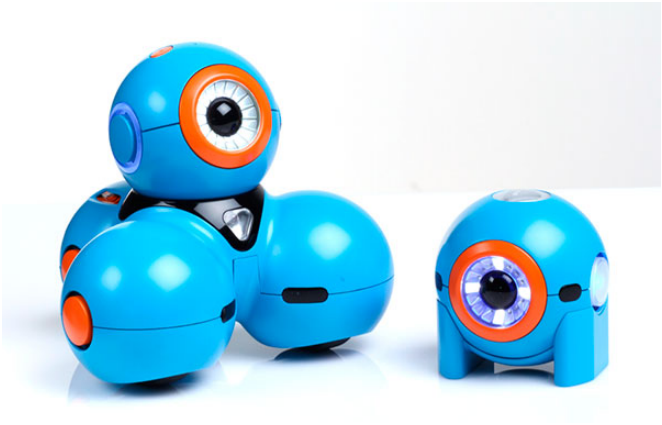

Figura 3.12: Robots Dash y Dot

por ello el desarrollo de nuevas formas de interacción permitirá el avance y la implantación de la misma en ámbitos generales de la sociedad, sin limitarse a nichos como puede ser la industria o la educación.

Siendo el ROBOBO un enfocado a la educación y a que los niños jueguen, tomaremos los ejemplos de robots mascota y robots educativos y las formas de interacción vistas en ellos como base a la hora de desarrollar el subsistema de interacción. Estas formas de interacción se separan de lo habitual ya que no deben simplemente ser una forma de comunicación con el aparato, sino que también deben entretener y mantener la atención de los niños.

Capítulo 4

Fundamentos tecnológicos

A La hora de desarrollar los módulos de interacción, para cubrir las limitaciones de la API de Android que aun siendo extensa no proporciona funcionalidades para todo, se han empleado una serie de librerías adicionales.

En este capítulo se explicarán brevemente estas librerías y su rol dentro del subsistema de interacción desarrollado en este trabajo.

4.1. Sphinx

CMU Sphinx [12] es una serie de sistemas de reconocimiento de voz desarrollados en la Universidad de Carnegie Mellon. En este trabajo se emplea una versión ligera de Sphinx adaptada a dispositivos móviles, Pocket Sphinx, con el modelo acústico y el diccionario para el idioma inglés.

La capacidad de reconocer voz sin la necesidad de una conexión a internet constante favoreció a esta librería frente a otras, como podría ser el sistema de reconocimiento vocal de Google.

Esta librería se ha usado para implementar el módulo de detección de habla, que se explica con detenimiento en la sección 5.5.1.

4.2. TarsosDSP

TarsosDSP [13] es una librería en Java, de código libre, para procesado de sonido. La librería proporciona una serie de métodos de procesado musical entre los que se encuentran:

- Detector de sonidos percusivos
- Detector de frecuencias, con múltiples implementaciones de algoritmos para esa finalidad
- Efectos de sonido
- Síntesis de audio

En este trabajo esta librería ha sido escogida para la implementación varios de los módulos de la librería de interacción por sonido dada la simplicidad de uso y la buena documentación de la misma.

4.3. OpenCV

OpenCV [14] es una librería de código libre de visión artificial ampliamente extendida y con una gran comunidad detrás. Esta librería originalmente está desarrollada en C++, sin embargo existen numerosos ports para los lenguajes de programación más comunes, en nuestro caso se usará el port a Java.

Esta librería contiene utilidades para el trabajo con imagen, tipos de datos específicos e implementaciones de los algoritmos más comunes de procesado de imágenes.

En este trabajo es empleada dentro del paquete de visión artificial, para el módulo de detección de colores, ya que la API de Android no está pensada para esta clase de procesado.

4.4. Gmail Background

Gmail Background [15] es una librería creada por el usuario de Github *yesid-lazaro* que permite el envío de correos electrónicos programaticamente sin interacción del usuario desde cuentas de Gmail de manera sencilla.

Esta librería fue empleada a la hora de implementar el módulo de mensajería y se escogió por su simplicidad de uso.

Capítulo 5

Desarrollo

En este capítulo se desarrollarán los detalles de la implementación de los diferentes módulos de interacción que se han diseñado para el ROBOBO así como la metodología de trabajo seguida.

5.1. Capacidades de interacción del ROBOBO

El ROBOBO gracias a la combinación de Smartphone (OBO) y base motorizada (ROB) presenta un gran potencial de formas de interacción.

El OBO, pone a nuestra disposición todas las características de un teléfono inteligente moderno que, diseñado específicamente para su uso por humanos, ofrece múltiples posibilidades y capacidades que facilitan el desarrollo de un sistema de interacción.

- **Gran capacidad de procesado de datos**, que nos permite procesar, por ejemplo, las imágenes capturadas por la cámara para desarrollar métodos de interacción complejos, como detección de caras o colores
- Variedad de sensores:
  - **Acelerómetro, giroscopio y magnetómetro**, permiten al robot tener conciencia de su posición espacial
  - **Sensores de luz ambiente**, que permiten reaccionar a los cambios lumínicos del entorno
  - **Micrófono**, que permite escuchar su entorno y actuar en consecuencia

- **Cámara frontal y trasera**, que permiten capturar imágenes y procesarlas posteriomente

- **Pantalla táctil de alta resolución**, que permite interacción en dos direcciones, ya sea mediante gestos táctiles como mediante gráficos en la pantalla
- **Conexión a internet**, que permite al robot comunicarse a través de este medio.
- **Altavoces**, permiten interacción mediante sonidos, música y habla.

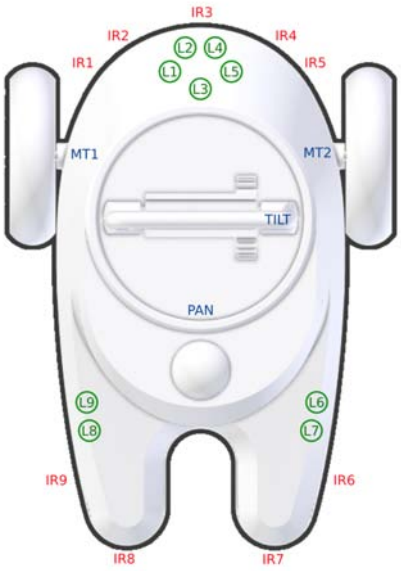

Figura 5.1: Esquema de sensores y actuadores del ROB

El ROB, cuyo esquema de sensores y actuadores podemos ver en la figura 5.1, proporciona ciertas posibilidades de interacción ya sea mediante sus motores, por ejemplo a través de bailes y gestos, como utilizando los 9 leds RGB que se encuentran a lo largo del chasis para expresar diferentes cosas.

Combinando ambas partes se obtiene un robot con un gran potencial para establecer interacción con personas.

5.2. Modelo conceptual para el subsistema de interacción

Partiendo de las características expuestas en el punto anterior y basándose en los ejemplos de interacción mostrados en el capítulo anterior se han definido, en una analogía a los sentidos, una serie de paquetes para dotar al ROBOBO de diversas funcionalidades de interacción.

El primero de estos paquetes sería el «sentido» de habla, que en la sección anterior se pudo ver su aplicación en robots como Nao, Dash y Dot. Siendo el habla, probablemente, la principal forma de interacción que se da entre los humanos, parece lógico que, teniendo la capacidad de proceso de un smartphone moderno, dotar al robot ya no solo de la capacidad de producir habla, sino de entender texto hablado es un concepto interesante. Este paquete daría al robot la posibilidad de comunicación bidireccional con los humanos, además de darle al robot cierta personalidad más «orgánica» , haciéndolo más atractivo, por ejemplo, a los niños.

El segundo, de forma análoga al sentido del tacto, teniendo una «piel artificial» cómo es la pantalla táctil del teléfono, dar la capacidad al robot de sentir toques y caricias parece una idea interesante a la hora de llevar una interacción cercana con el robot, este tipo de interacción mediante toques en la pantalla táctil es similar al del SmartPet, y de una manera más general, al sentido del tacto del robot Paro o el Aibo.

El tercero, siguiendo la analogía con los sentidos, sería el sentido del oído. La capacidad de producir y reconocer sonidos abre puertas a tipos de interacción interesantes. Por ejemplo, la capacidad de reaccionar ante sonidos fuertes como palmadas o la capacidad de reconocer notas musicales podrían permitir al robot comunicarse mediante el uso de la música, y estando el ROBOBO enfocado directamente a la educación, esta clase de interacción podría emplearse para la enseñanza musical en la educación infantil y primaria. También, la producción de sonidos, es una forma de comunicación que parece adecuada a la hora de interactuar con niños, ya sea mediante sonidos básicos (risas, alarmas, abucheos...) como a través de notas musicales, que podría usarse en conjunto con la capacidad de reconocimiento de las mismas a la hora de la enseñanza musical. Esta capacidad de producir música se puede ver en el robot Dash mediante el módulo del xilófono

y la de detección de música y palmadas en el Zowi.

El cuarto paquete, teniendo hoy en día todos los smartphones como mínimo una cámara, busca dotar al ROBOBO de sentido de la vista. Con este sentido, el robot podría detectar la presencia de gente y a que distancia se encuentran y de esta manera adaptar su comportamiento a su entorno, por ejemplo, alejándose si nota que alguien esta muy cerca o siguiendo con la «cara» a la gente a su alrededor. La capacidad de discernir diferentes colores también es una habilidad interesante de cara a la educación de niños muy pequeños. La capacidad de detección de caras y colores se pudo encontrar en los robots más complejos vistos en la sección anterior, como Nao o Aibo, que cuentan con cámaras en su cabeza.

Por último, teniendo la capacidad de conectarse a internet del smartphone, parece interesante dotar al robot de la capacidad de comunicarse a través de la red. El correo electrónico es un sistema de comunicación muy extendido y casi obligatorio hoy en día, así que, dada la extensión del sistema, y la cantidad de usuarios con los que cuenta, parece la opción correcta para este tipo de comunicación.

5.3. Arquitectura Global

El sistema de interacción humana del sistema ROBOBO fue desarrollado de una forma modular, de manera que las diferentes funcionalidades puedan ser utilizadas de forma separada.

La gestión de los módulos la realiza el ROBOBO! Framework(sección 2.1), que proporciona una interfaz común que debe ser implementada por todos ellos.

Cada «sentido» de los elaborados conceptualmente en la sección 5.2 se ha implementado en forma de módulo librería de Android, y cada funcionalidad especifica ha sido integrada en módulos que implementan IModule, interfaz común para los módulos del ROBOBO , para poder ser manejados por el ROBOBO! Framework [16].

- Paquete Speech: Gestiona las operaciones que involucran habla.
  - Módulo SpeechProduction: Permite utilizar la síntesis de voz.
  - Módulo SpeechRecognition: Permite el reconocimiento de voz, por palabras clave o por gramáticas.

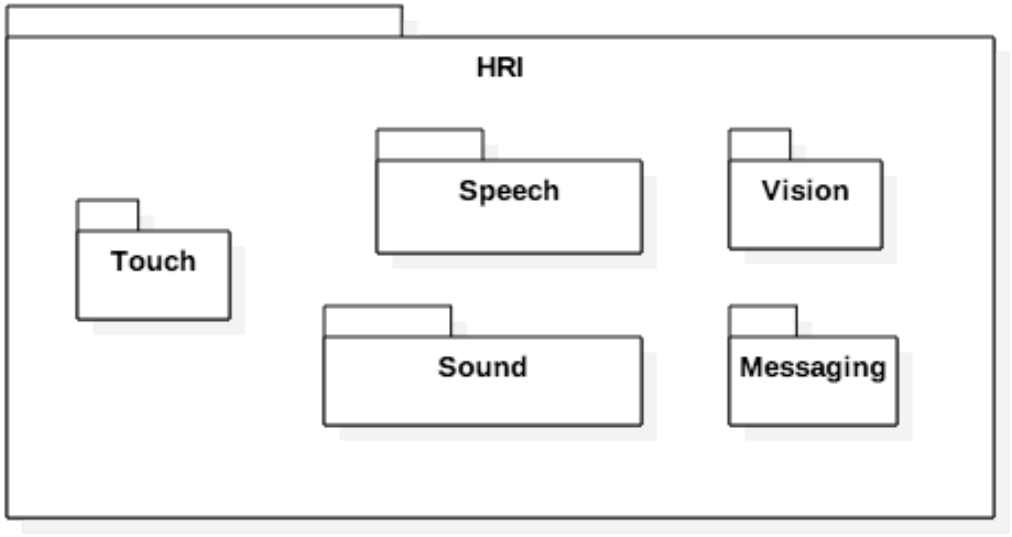

Figura 5.2: Estructura de paquetes dentro del paquete HRI del Robobo

- Paquete Touch: Gestiona el sentido del tacto del robot.
  - Módulo Touch: Permite la detec ción de gestos en la pantalla táctil del móvil.
- Paquete Sound: Gestiona el sentido del oído del robot.
  - Módulo SoundDispatcher: Módulo auxiliar para los módulos que emplean la librería TarsosDSP [13]
  - Módulo ClapDetectionModule: Módulo que permite la detección de sonidos percusivos.
  - Módulo PitchDetection: Módulo que permite la estimación de la frecuencia de un sonido.
  - Módulo NoteDetection: Módulo que detecta notas musicales a partir de frecuencias.
  - Módulo NoteGenerator: Módulo que genera tonos correspondientes a notas musicales.
  - Módulo Emotionsound: Módulo que permite la reproducción de una serie de sonidos prefijados con el objetivo de expresar sentimientos.

- Paquete Vision: Gestiona el sentido de la vista del robot.
  - Módulo BasicCamera: Provee de un stream constante de frames capturados desde la cámara frontal del smartphone.
  - Módulo FaceDetection: Permite detectar caras en los frames capturados por la cámara.
  - Módulo ColorDetection: Permite detectar colores sobre fondos blancos en los frames.
- Paquete Messaging: Gestiona los métodos de mensajería del robot
  - Módulo Messaging: Permite enviar correos electrónicos, pudiendo adjuntar imágenes.

Cada paquete se distribuye en forma de librería Android dentro del paquete HRI del ROBOBO! Framework.

5.3.1. Estructura de un módulo

Todos los módulos comparten una estructura de paquetes similar, un paquete con el nombre del módulo , que contiene la interfaz del módulo (de la forma *INombreModulo*) que extiende la clase *IModule*, una clase abstracta opcional (*ANombreModulo*) que implementa la interfaz anterior y gestiona labores comunes como la suscripción a los Listeners, y los paquetes de las diferentes implementaciones especificas. Opcionalmente, este paquete también podrá contener la interfaz de los listeners, y clases de utilidad como enumerados.

Las implementaciones de los módulos tienen el nombre *NombreImplementacionNombreModulo*, y deben extender la clase abstracta o la interfaz del módulo en caso de carecer de la primera.

5.4. Metodología

La metodología seguida fue un proceso iterativo en el cual en cada iteración se fueron añadiendo nuevos módulos de interacción y realizando correcciones en los módulos ya implementados.

5.4.1. Primera iteración

En esta primera iteración el trabajo de desarrollo se centró en los paquetes de producción y reconocimiento de habla (Paquete *Speech*) y el de tacto (Paquete *Touch*. Se implementaron las funcionalidades básicas de cada módulo:

- *SpeechProductionModule*: Se implementaron las funcionalidades de síntesis de voz, selección de voces y de localización.
- *SpeechRecognitionModule*: Se implementó el reconocimiento básico por palabras clave.
- *TouchModule*: Se implementaron el reconocimiento de toques largos(touch) y cortos(tap), así como el reconocimiento de caricias(caress) y deslizamientos rápidos(fling).

5.4.2. Segunda iteración

En la segunda iteración se desarrolló el paquete de interacción por sonidos y se realizaron correcciones tanto en el módulo de reconocimiento de voz cómo en el táctil.

Nuevas implementaciones en esta iteración:

- *SoundDispatcherModule*: Implementado para dar soporte al resto de módulos que usan la librería TarsosDSP [13].
- *ClapDetectionModule*: Implementado el módulo de detección de sonidos percusivos.
- *PitchDetectionModule*: Implementada el módulo de detección de frecuencias en hertzios.
- *NoteDetectionModule*: Implementada el módulo de detección de notas musicales. Implementada clase para representar las notas.

Actualizaciones de módulos de la anterior iteración:

- *SpeechRecognitionModule*: Implementada la búsqueda por gramáticas
- *TouchModule*: Mejorada la información producida por el evento *onFling*.

5.4.3. Tercera iteración

En esta iteración se desarrollaron los paquetes de visión y de mensajería. También se sacaron de dentro de los módulos los diferentes parámetros, para poder ser configurados desde el exterior mediante ficheros *properties*.

Nuevas implementaciones en esta iteración:

- *BasicCameraModule*: Implementado el notificador de frames.
- *FaceDetectionModule*: Implementado el modulo de detección de caras.
- *ColorDetectionModule*: Implementado el modulo de detección de colores.
- *MessagingModule*: Implementado el módulo de mensajería por gMail.

Actualizaciones de módulos:

- *SoundDispatcherModule*: Parametrización mediante properties.
- *ClapDetectionModule*: Parametrización mediante properties.
- *PitchDetectionModule*: Parametrización mediante properties.

5.4.4. Cuarta iteración

En esta iteración se añadieron dos módulos de producción de sonidos.
Nuevas implementaciones esta iteración:

- *EmotionSoundModule*: Implementada la reproducción de sonidos prefijados.
- *NoteGeneratorModule*: Implementada la generación de tonos.

Actualizaciones de módulos:

- *ColorDetectionModule*: Añadida una comprobación de la varianza del color para evitar falsos positivos.

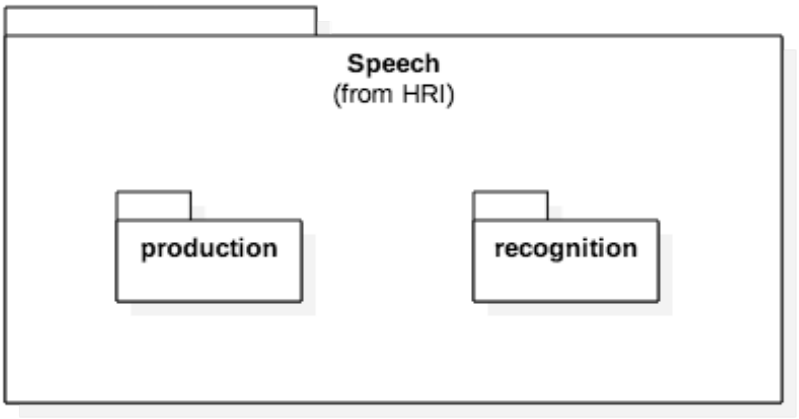

Figura 5.3: Paquete Speech

5.5. Librerías de interacción

A continuación se hablará sobre los detalles de implementación de las diferentes librerías de interacción desarrolladas.

5.5.1. Paquete Speech

Este subsistema gestiona todas las labores que tienen que ver con la generación o detención del habla. En su interior encapsula dos paquetes diferentes (figura 5.3) , Production y Recognition, que contienen respectivamente los módulos de producción y reconocimiento del habla.

5.5.1.1. Módulo recognition

El módulo recognition(figura 5.4) permite al usuario reconocer texto hablado de forma fácil. Provee las Interfaces *ISpeechRecognitionModule* y *ISpeechRecognitionListener*, así cómo la clase abstracta *ASpeechRecognitionModule*.

ISpeechRecognitionModule ofrece la declaración de métodos de configuración del reconocedor de voz y de suscripción de listeners.

Esta interfaz proporciona funcionalidades de:

- Añadir y eliminar palabras reconocibles.

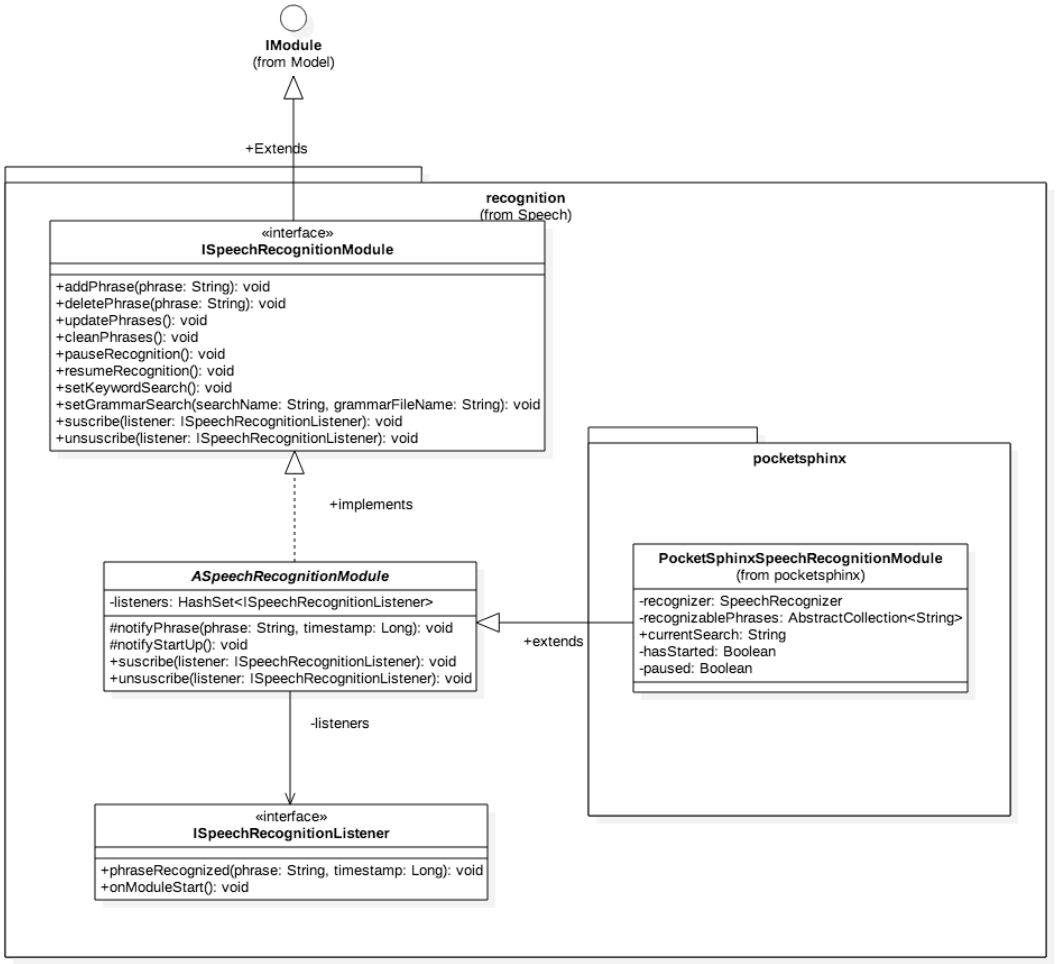

Figura 5.4: Módulo SpeechRecognition

- Actualizar el listado interno de frases
- Limpiar el listado interno
- Pausar y reanudar el reconocimiento
- Añadir búsquedas por gramática
- Utilizar búsqueda por palabras

ISpeechRecognitionListener es la interfaz que deben implementar las clases para ser notificados de los eventos que produce el módulo de reconocimiento.

Las clases suscritas este listener son notificadas cuando:

- El reconocedor ha sido inicializado
- Una frase ha sido detectada

ASpeechRecognitionModule es la clase abstracta que gestiona las tareas comunes a todas las implementaciones del modulo, a saber:

- Mantener un listado de los listeners
- Suscribir y desuscribir listeners
- Notificar a los listeners de los diferentes eventos

Pocketsphinx

Se ha decidido realizar la implementación del módulo utilizando la librería CMU Sphinx [12], desarrollada por la Universidad de Carnegie Mellon que se ha presentado en la sección 4.1. Específicamente se utiliza PocketSphinx, una versión de CMU Sphinx ligera, diseñada para su uso en sistemas embebidos.

PocketSphinx permite varios tipos de búsqueda diferentes, pero al contrario que las versiones más completas de la librería, sólo permite tener una búsqueda activa al mismo tiempo.

Entre los modos de búsqueda que ofrece se encuentran la búsqueda por Keywords y la búsqueda por Gramáticas, que son las que son ofrecidas por la interfaz del módulo. Adicionalmente PS también permite el uso de búsquedas por fonemas, pero no se ha contemplado su uso.

El módulo se llama *PocketSphinxSpeechRecognitionModule*, extiende *ASpeechRecognitionModule* e implementa la interfaz *RecognitionListener* que provee PocketSphinx.

La búsqueda por keywords permite al usuario definir, en tiempo de ejecución, una serie de palabras que podrán ser reconocidas, estas palabras deben estar contenidas en el diccionario del modelo fonético.

La búsqueda por gramática permite definir archivos de gramáticas en el formato JSFG 1.0 [17], lo cual permite detectar construcciones más complejas de palabras. Para funcionar correctamente deben colocarse en el directorio assets/sync/ de la aplicación tanto el modelo fonético del idioma, como los archivos de gramáticas que se vayan a utilizar, todo archivo dentro de este directorio debe contar con un fichero de mismo nombre y extensión .md5 conteniendo un hash del original en su interior. Los archivos deben ser declarados dentro del fichero assets.lst.

Dado que el inicio del reconocedor de voz se realiza de manera asíncrona, es necesario notificar a la aplicación principal de cuando el módulo esta preparado, de lo contrario, llamar a una operación del mismo podría ser inseguro. Le arranque del módulo se notifica a través del método onModuleStart() que debe ser implementado por cualquier clase que vaya a utilizar el módulo.

5.5.1.2. Módulo Production

El módulo production(figura 5.5) permite al usuario producir voz a partir de texto de manera simple. Provee la interfaces *ISpeechProductionModule* y *ITtsVoice*, así como la excepción *VoiceNotFoundException*. La primera contiene las declaraciones de los métodos del módulo en sí, la segunda proporciona una forma de representar las diferentes voces a utilizar por el módulo.

Las funcionalidades que facilita la interfaz son:

- Pronunciar un texto
- Cambiar la localización
- Seleccionar voces
- Recuperar las voces disponibles

Implementación específica:

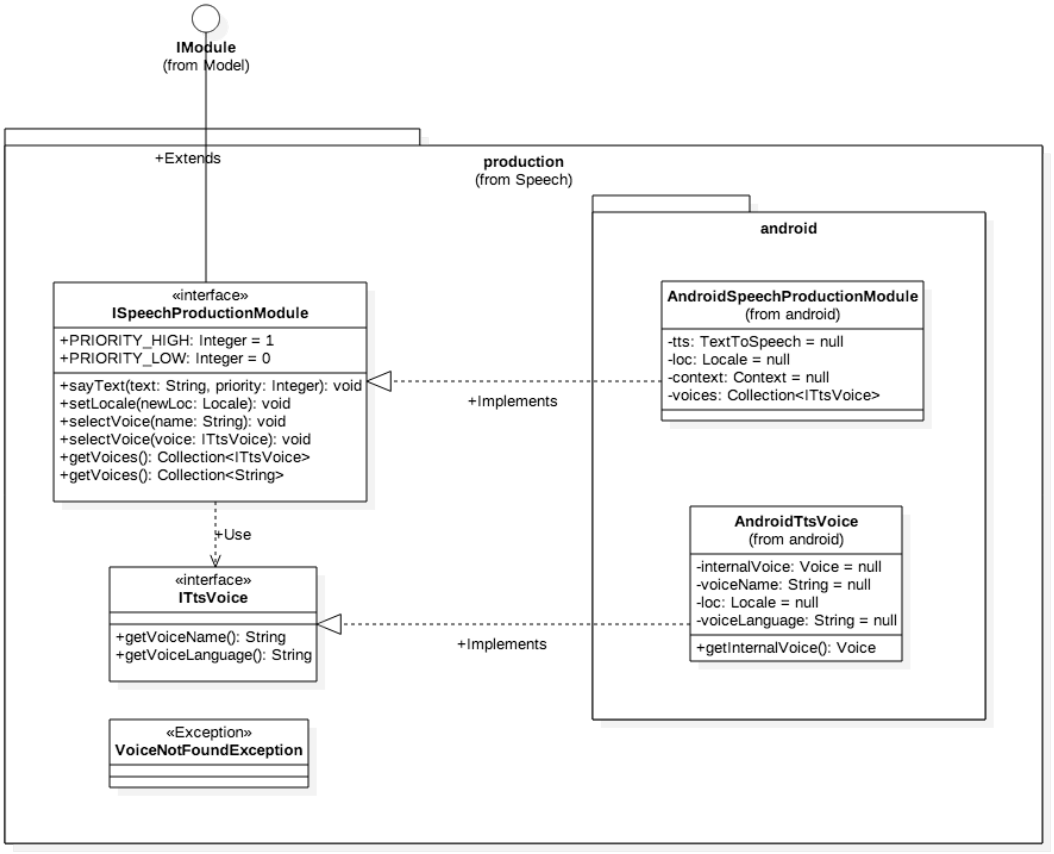

Figura 5.5: Módulo SpeechProduction

La implementación de la interfaz se ha realizado con las librerías del propio sistema Android (**android.speech.tts.TextToSpeech**)

Ejemplos de desarrollo

Para la prueba de los módulos de voz se realizaron dos ejemplos básicos, una aplicación simple en la que el usuario introduce un texto y el teléfono lo pronuncia, y un sistema de control mediante voz en el que el usuario le daba ordenes básicas al Robobo mediante la voz y este, al terminar la frase, ejecutaba los movimientos indicados. A continuación se puede observar una muestra del log de la aplicación de control por voz:

```
D/SpeechRecognitionModule: Recognized part rob
D/SpeechRecognitionModule: Recognized part rob front
D/SpeechRecognitionModule: Recognized part rob front now
I/SpeechRecognizer: Stop recognition
D/SpeechRecognitionModule: Recognized rob front now Prob: -4730
D/SpeechROB: rob front now
D/ROB-INTERFACE: Move forwards: 50 - 360
I/SpeechRecognizer: Start recognition "MovSearch"
D/SpeechRecognizer: Starting decoding
```

En este log se puede ver cómo el reconocedor detecta las tres partes de la frase, detiene el reconocimiento al detectar una frase válida en la gramática, se ejecuta un movimiento a través del módulo *Rob-Interface* y se reanuda la escucha. En el siguiente fragmento de código se puede ver la implementación del listener del reconocedor de frases.

```
@Override
public void phraseRecognized(final String phrase, Long timestamp) {
    try {
        Log.d("SpeechROB",phrase);
        if (phrase.equals("rob up now")) {
            movementModule.moveTilt(90);
        }
        if (phrase.equals("rob down now")) {
            movementModule.moveTilt(45);
```

```
            }
            if (phrase.equals("rob left now")) {
                movementModule.turnLeftAngle((short) 50, 360);
            }
            if (phrase.equals("rob right now")) {
                movementModule.turnRightAngle((short) 50, 360);
            }
            if (phrase.equals("rob front now")) {
                movementModule.moveForwardsAngle((short) 50, 360);
            }
            if (phrase.equals("rob back now")) {
                movementModule.moveBackwardsAngle((short) 50, 360);
            }
        } catch (InternalErrorException e){
        }
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                textView.setText(phrase);
            }
        });
    }
```

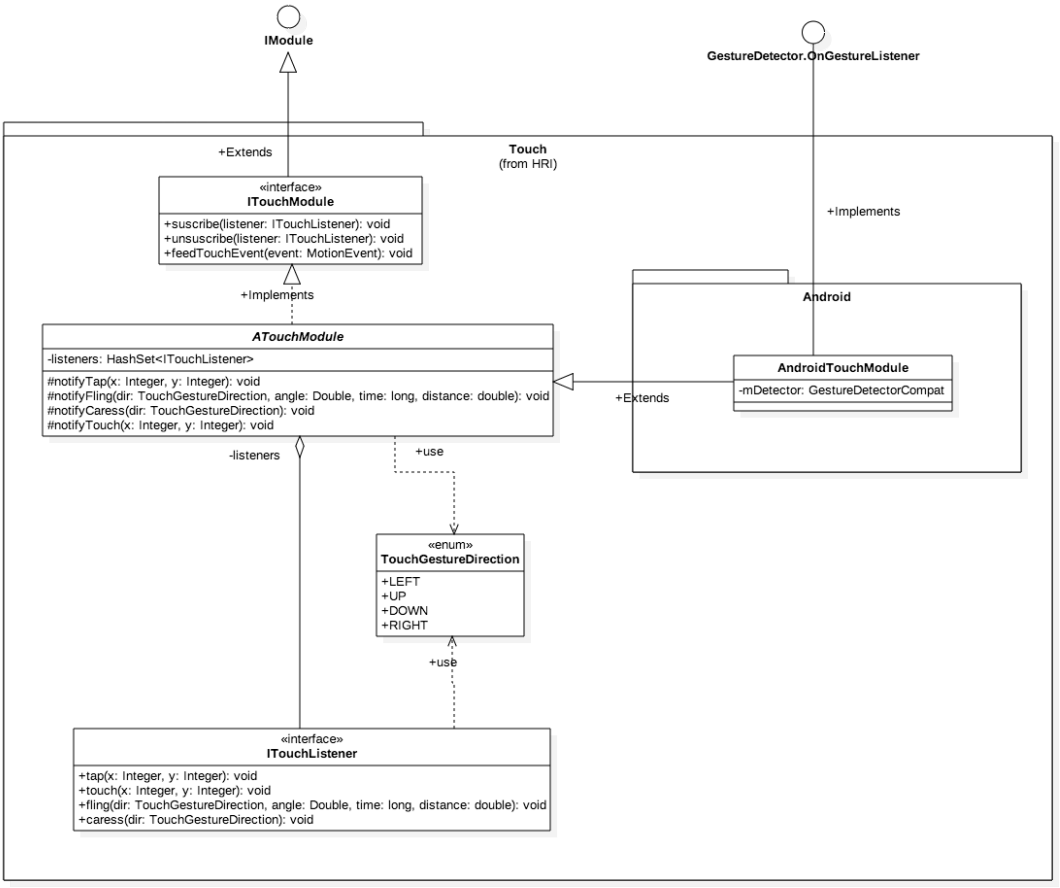

Figura 5.6: Módulo Touch

5.5.2. Paquete Touch

El subsistema Touch permite interacción con el robot en forma de gestos táctiles sobre la pantalla del teléfono. En su interior encontramos un único modulo, el módulo touch.

5.5.2.1. Módulo Touch

El módulo(figura 5.6) contiene las interfaces *ITouchModule* e *ITouchListener*, la clase abstracta *ATouchModule* y por la clase enumerado *TouchGestureDirection*.

ITouchListener es la interfaz que debe implementar cualquier clase que quiera ser informada de los gestos táctiles que ocurran en la pantalla del dispositivo. Las

acciones que se detectan son:

- Tap: Toque simple y rápido en la pantalla
- Touch: Toque mantenido en la pantalla
- Caress: Deslizamiento sobre la pantalla
- Fling: Al terminar un deslizamiento de manera rápida

ITouchModule es la interfaz que debe implementar cualquier implementación del módulo. El único método destacado de la misma es *FeedTouchEvent*, en el cual se le pasan los eventos táctiles al módulo desde la aplicación principal, que debe sobrecargar el método *OnTouchEvent* de la clase *Activity*, esto es debido a que el módulo no puede extender la clase *Activity*.

Detección táctil de Android

Para la implementación del módulo se escogió usar la clase *GestureDetector* propia de Android, que provee de varios listeners para diferentes eventos táctiles. Se han simplificado dichos eventos, para producir los definidos en *ITouchListener*

Ejemplos de desarrollo

Se implementó una simple aplicación que mostraba por pantalla los diferentes eventos producidos por el módulo *TouchModule*. A continuación se ve la implementación del listener *fling* en el ejemplo anterior:

```
@Override
 public void fling(TouchGestureDirection dir, double angle, long
     time, double distance) {
     speechProductionModule.sayText("Fling",ISpeechProductionModule.PRIORITY_HIGH)
     Log.d(TAG,"FLING");
     textView.setText("FLING: "+dir.toString()+" Time: "+time+"
         Distance: "+distance+" Angle: "+Math.toDegrees(angle));
 }
```

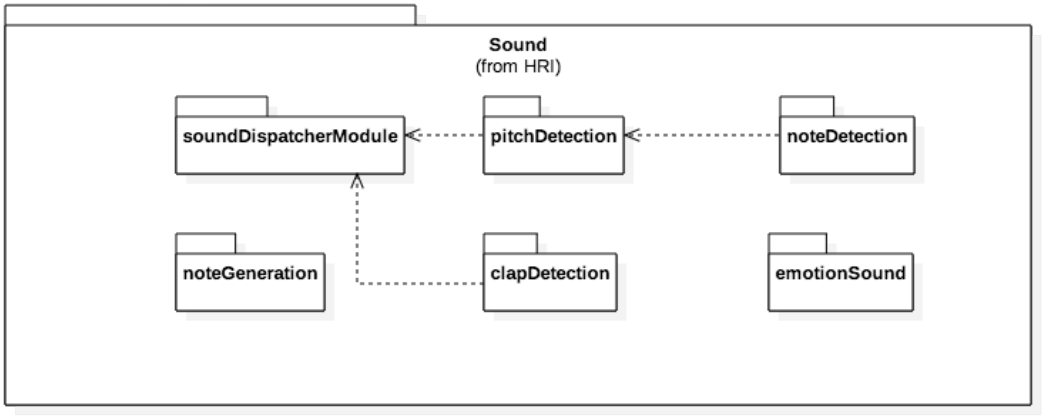

Figura 5.7: Paquete Sound

5.5.3. Paquete Sound

Este subsistema contiene los módulos(figura 5.7 que realizan diversas tareas de reconocimiento y producción de audio. Han sido implementadas tanto detección de tonos, notas musicales y palmadas, como síntesis de notas musicales y producción de sonidos prefijados.

5.5.3.1. Módulo Sound Dispatcher

SoundDispatcher(figura 5.8) es un módulo de utilidad, necesario para el uso de los módulos implementados a partir de la librería TarsosDSP [13]. Este módulo es el que se encarga de capturar el sonido del micrófono y segmentarlo en partes para su posterior procesado. Los distintos módulos se registran con sus procesadores en este módulo. El paquete contiene en su interior *ISoundDispatcherModule*, la interfaz del módulo, que permite

- Añadir procesadores de audio
- Eliminar procesadores de audio
- Iniciar el procesado de audio
- Parar el procesado de audio

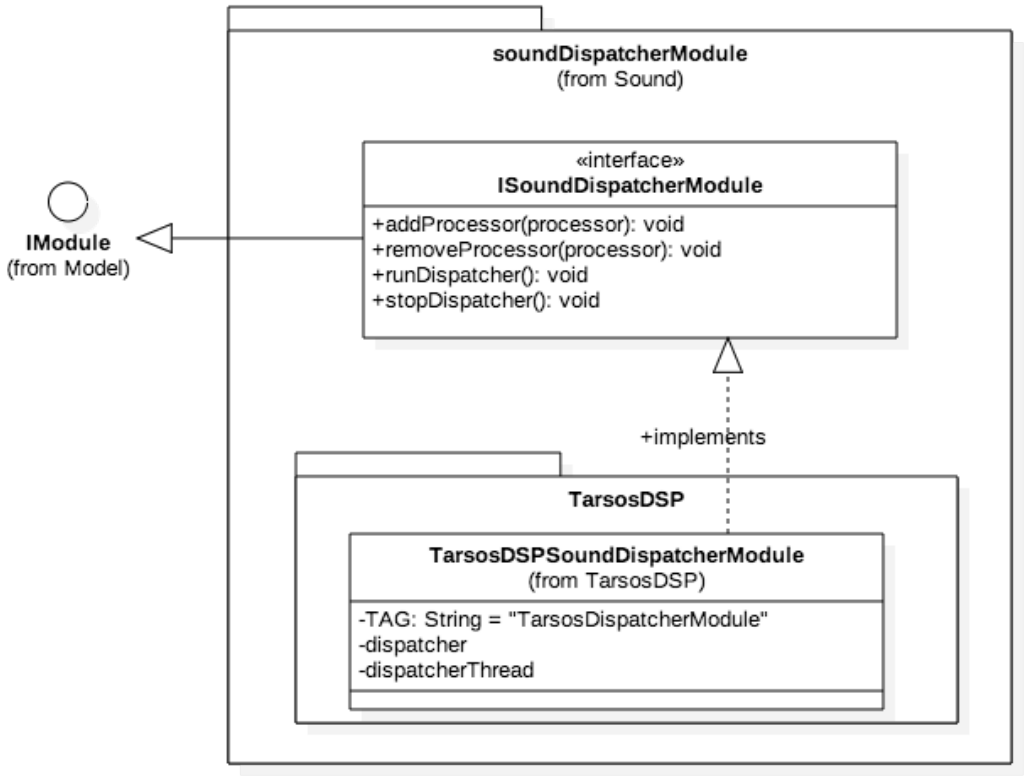

Figura 5.8: Módulo SoundDispatcher

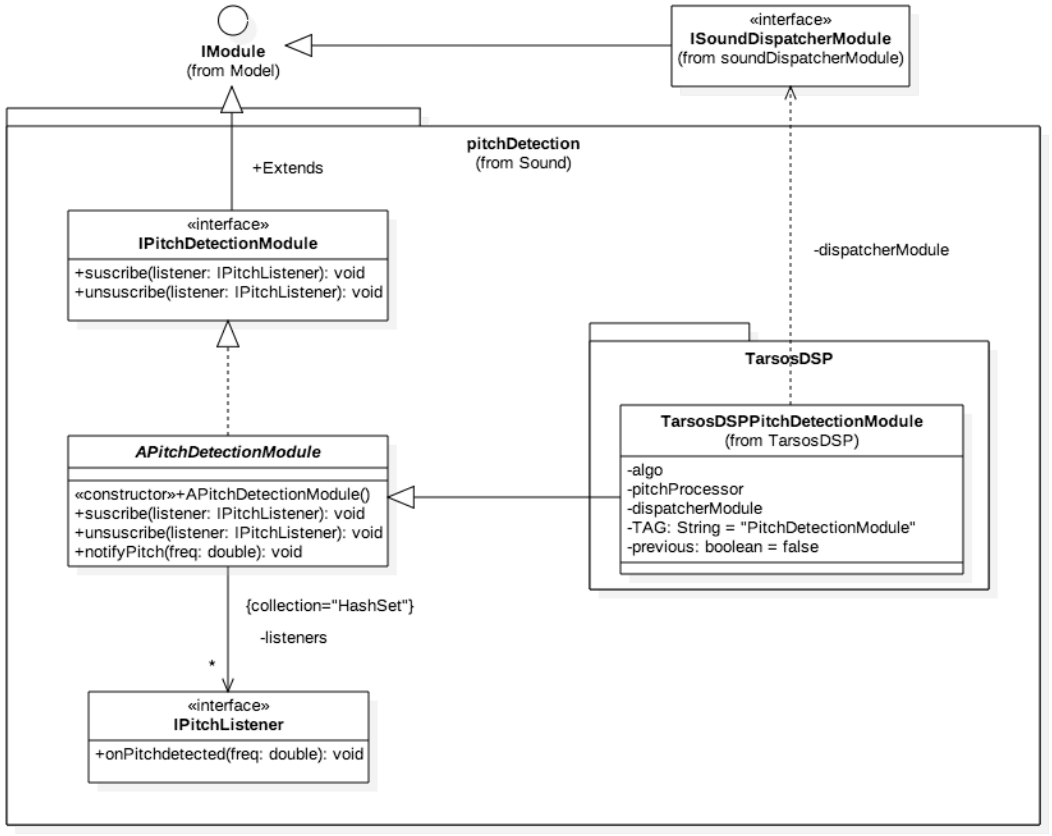

Figura 5.9: Módulo PitchDetection

TarsosDSP

La implementación se realiza en el módulo *TarsosDSPSoundDispatcherModule*, que implementa la interfaz *ISoundDispatcherModule*. En su interior se crea un objeto *AudioDispatcher*, que se encarga del procesado del audio, y al que pueden añadirse distintos procesadores de audio.

Los parámetros del objeto soundDispatcher pueden indicarse en el archivo sound.properties, pudiendo fijar el la tasa de muestreo(samplerate), el tamaño del buffer(buffer) y el solapado entre ventanas de audio(overlap).

5.5.3.2. Módulo Pitch Detection

PitchDetection(figura 5.9) permite detectar la frecuencia en Hertzios de sonidos, es utilizado, por ejemplo en el módulo de detección de notas musicales. En su interior encontramos la interfaz *IPitchDetectionModule* y la clase abstracta

APitchDetectionModule, que gestionan la suscripción de listeners a los eventos. También se encuentra *IPitchListener*, que debe ser implementada por cualquier clase que desee recibir notificaciones del módulo de detección de tonos.

TarsosDSP

La implementación del módulo se ha realizado en la clase *TarsosDSPPitchDetectionModule*. Esta implementación depende del módulo SoundDispatcher, ya que en su inicialización se registra a su procesador de audio en el soundDispatcher. Esta implementación utiliza el algoritmo YIN [18] para la detección de frecuencias. La implementación del algoritmo viene dada por la propia librería TarsosDSP. El módulo genera un evento con la frecuencia cuando un sonido es detectado, y al terminar la detección devuelve un -1.

5.5.3.3. Módulo Note Detection

El módulo NoteDetection(figura 5.10) busca proporcionar una manera simple de detectar notas musicales. Depende directamente del módulo PitchDetection, del cual obtiene la frecuencia que traduce en notas musicales. En su interior encontramos la interfaz del módulo *INoteDetectionModule*, la clase abstracta *ANoteDetectionModule*, que además de la gestión de listeners provee un método de conversión de frecuencias a notas, la interfaz *INoteListener* que recibe notificaciones al:

- Comenzar una nota
- Detectar una nota
- Terminar una nota

Toda clase que desee recibir los eventos producidos por el módulo debe implementar esta interfaz. Por último en el paquete encontramos una clase *Note* que se utiliza como representación de las notas musicales. Este enumerado contiene la información del índice de la conversión de notas, la nota musical en formato anglosajón, y la octava a la que pertenece la nota.

TarsosDSP

TarsosDSPNoteDetectionModule extiende *ANoteDetectionModule* e implementa *IPitchListener*. Simplemente convierte los tonos detectados a notas musicales,

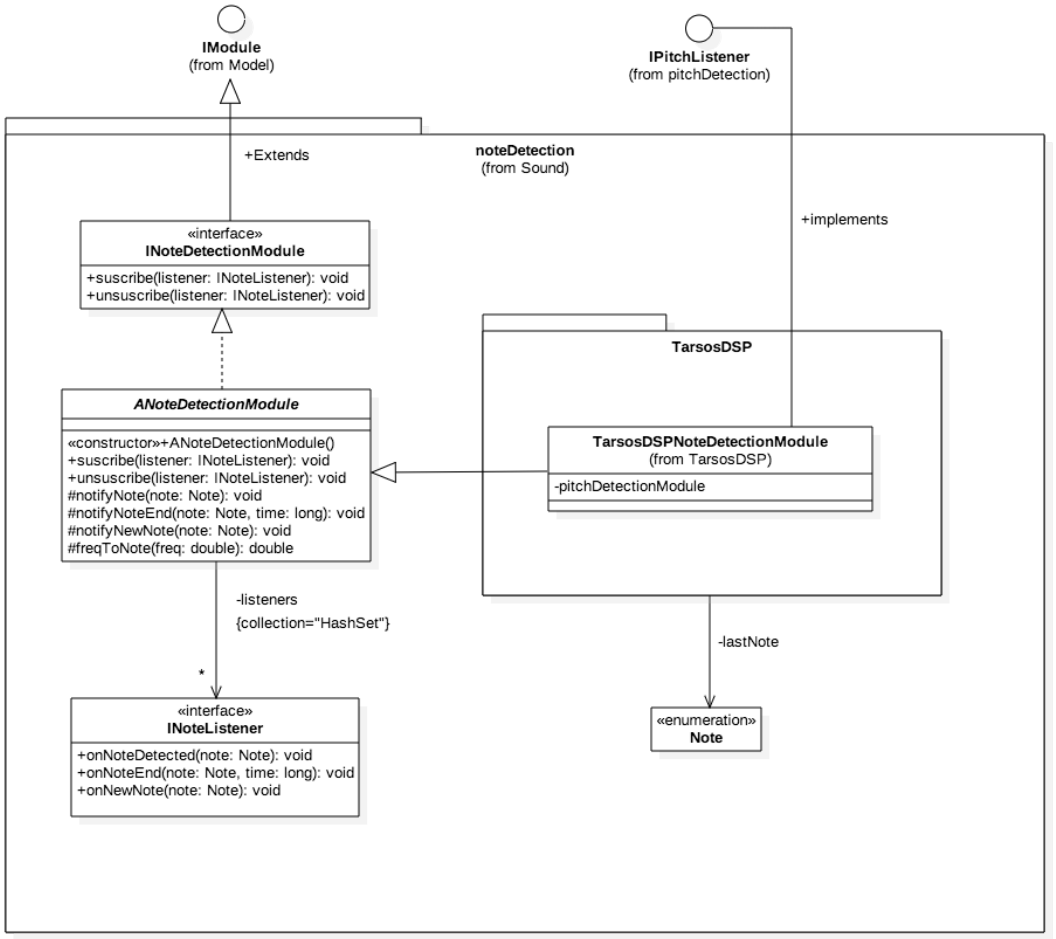

Figura 5.10: Módulo NoteDetection

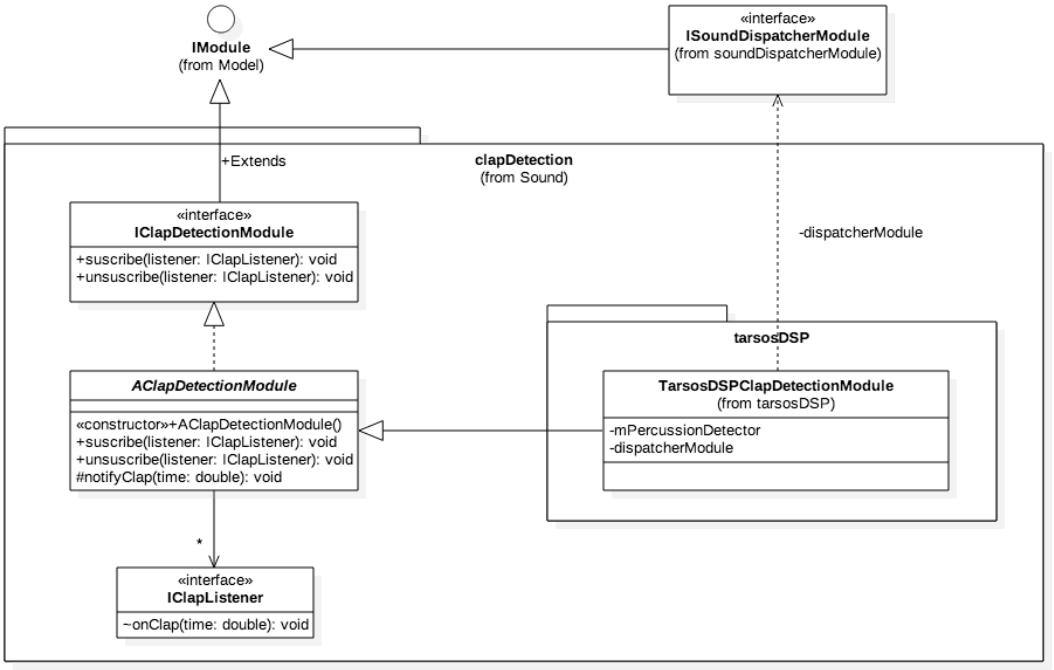

Figura 5.11: Módulo ClapDetection

filtra las notas desafinadas(descarta las que se desvían un 10 % respecto al indice) y controla los inicios y finalizaciones de las notas, generando eventos de los cuales serán notificados los listeners.

5.5.3.4. Módulo Clap Detection

El módulo ClapDetection(figura 5.11) permite detectar ruidos percusivos tales como palmadas. Igual que el PitchDetection, depende directamente del módulo SoundDispatcher para su funcionamiento. En su interior se encuentra la interfaz del módulo, *IClapDetectionModule*, la clase abstracta *AClapDetectionModule* y la interfaz del listener, *IClapListener*. Las clases que implementen este listener recibirán una notificación cada vez que se detecte una palmada.

TarsosDSP

La implementación del módulo, contenida en *TarsosDSPClapDetectionModule*, emplea el objeto provisto por la librería TarsosDSP *PercussionOnsetDetector*, que es registrado en el *DispatcherModule* activo en ese momento. Este objeto permite

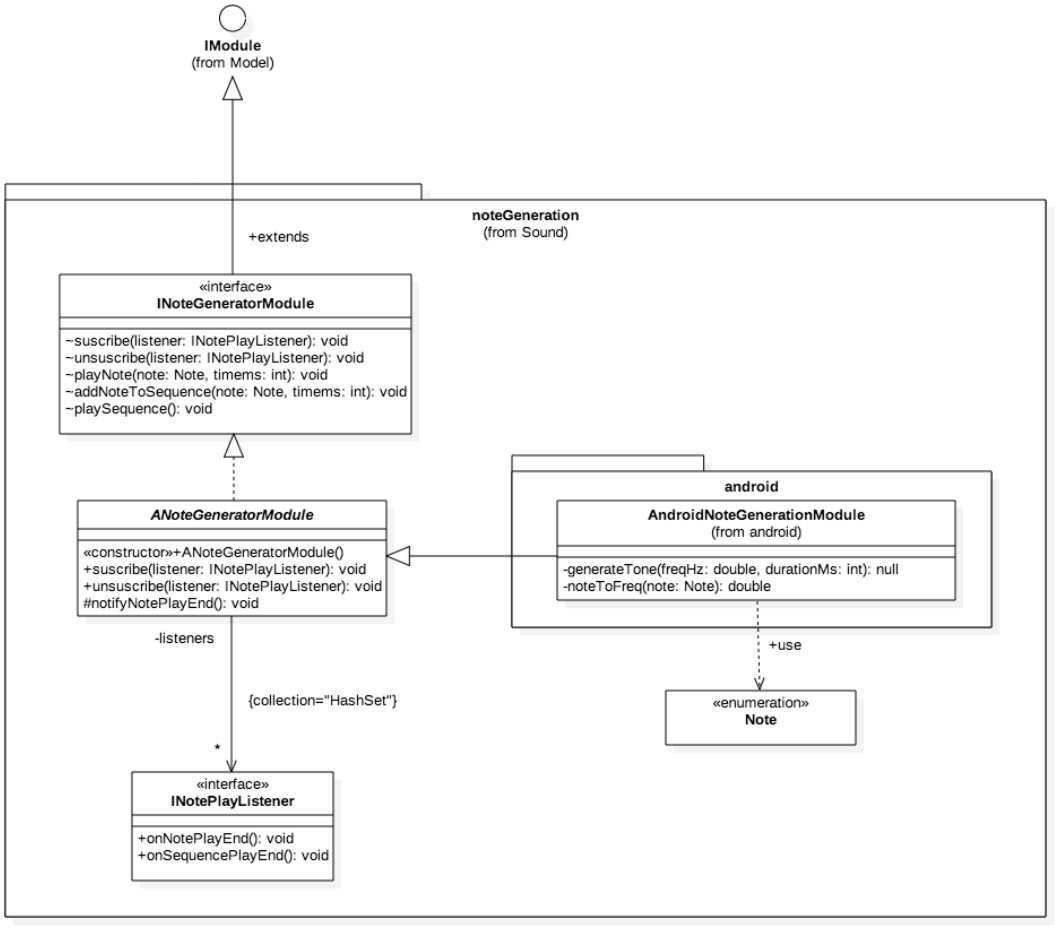

Figura 5.12: Módulo NoteGenerator

la detección de sonidos percusivos, como palmadas, de una forma sencilla.

5.5.3.5. Módulo NoteGenerator

El módulo NoteGenerator(figura 5.12) provee una forma de sintetizar tonos musicales y reproducir secuencias de notas. En su interior se encuentra la interfaz del módulo *INoteGeneratorModule* que define las funcionalidades del módulo y la clase abstracta *ANoteGeneratorModule* que gestiona la suscripción a los listeners. La interfaz del listener se encuentra en *INotePlayListener* y su finalidad es notificar cuando una nota o una secuencia de notas termina de reproducirse. La clase *Note* es la representación de las notas musicales en el sistema, contiene la nota, el índice y la octava.

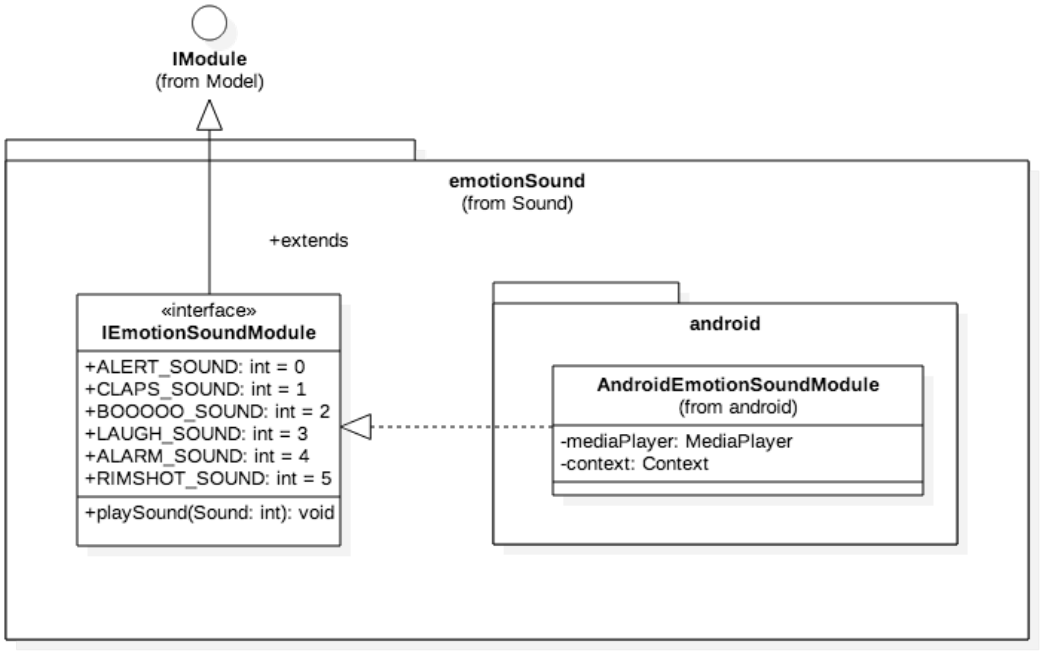

Figura 5.13: Módulo EmotionSound

Android

La implementación especifica del módulo se encuentra en *AndroidNoteGeneratorModule* y se ha realizado mediante la clase *AudioTrack* proporcionada por la API de Android. El objeto *AudioTrack* se crea sample a sample utilizando la ecuación de una onda senoidal de la frecuencia de la nota requerida. La reproducción de secuencias se implementa mediante el uso de las clases *Timer* y *TimerTask*

5.5.3.6. Módulo EmotionSound

El módulo EmotionSound(figura 5.13) busca proporcionar una manera de reproducir una serie de sonidos prefijados para expresar diferentes emociones. La interfaz del módulo esta definida en *IEmotionSoundModule*, y en ella se declaran, en forma de constantes, los sonidos utilizables. Los sonidos se encuentran en el directorio /raw de la carpeta /res del paquete Sound.

Android

La implementación de este módulo se ha realizado mediante la clase *MediaPla-*

yer provista por la API de Android, que permite reproducir diferentes recursos audiovisuales. El módulo está definido en la clase *AndroidEmotionSoundModule*.

Ejemplos de desarrollo

Para probar la detección de notas musicales se desarrolló un control mediante música en el que el usuario ejecutaba una secuencia de notas musicales en una escala, cada una correspondiendo a una acción de movimiento, y al recibir la nota de cierre de secuencia, el ROBOBO ejecutaba los movimientos indicados. A continuación se muestra un ejemplo de log de la ejecución:

```
D/MusicActivity: Note: A
D/MusicActivity: recording
D/MusicActivity: Note: As
D/MusicActivity: Note: D
D/MusicActivity: adding command
D/MusicActivity: Note: Ds
D/MusicActivity: Note: E
D/MusicActivity: adding command
D/MusicActivity: Note: G
D/MusicActivity: adding command
D/MusicActivity: Note: A
D/MusicActivity: executing
D/MusicActivity: Executing command $NoteCommand@14bb8f87
D/ROB-INTERFACE: Move backwards: 60 - 861
D/MusicActivity: Executing command $NoteCommand@3016eb4
D/ROB-INTERFACE: Turn left: 60 - 1816
D/MusicActivity: Executing command $NoteCommand@a8f2add
D/ROB-INTERFACE: Turn right: 60 - 1816
```

Se puede ver como la nota A(La) abre la secuencia de comandos, As(La sostenido) y Ds(Re sostenido) son ignorados al no estar en la escala definida (A, C, D, E, G), son añadidos tres comandos de movimiento (Correspondientes a las notas D(Re), E(Mi) y G(Sol)) y la nota A cierra la secuencia de comandos que serán ejecutados a través de la *Rob-Interface*.

A continuación se muestra la implementación del listener *onNoteEnd* usada en el módulo anterior:

```
@Override
public void onNoteEnd(Note note, long time) {
    Log.d(TAG,"Note: "+note.note);
    if (!executing) {
        if (note.note == (triggerNote)) {
            if ((recording)&&(time>100)) {
                recording = false;
                executing = true;
                Log.d(TAG,"executing");
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        rellayout.setBackgroundColor(Color.BLUE);
                    }
                });
                executeCommands(commandList);
            } else {
                Log.d(TAG,"recording");
                recording = true;
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                       rellayout.setBackgroundColor(Color.GREEN);
                    }
                });
            }
        } else {
            if (recording) {
                if (scale.contains(note.note)) {
                    Log.d(TAG,"adding command");
                    commandList.add(new NoteCommand(note, time));
                } else {
                    //Bad note
}}}}}
```

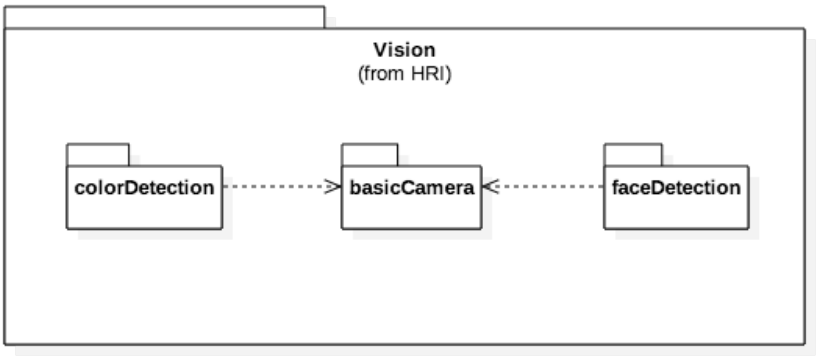

Figura 5.14: Paquete Vision

5.5.4. Paquete Vision

Este subsistema(figura 5.14) contiene los diferentes módulos de captura y procesado de imagen

5.5.4.1. Módulo Basic Camera

Este es el módulo básico de cámara(figura 5.15) , del cual hacen uso el resto de módulos de procesado de imagen. Este modulo produce un stream constante de imágenes capturadas de la cámara frontal del dispositivo y notifica a los listeners suscritos. La interfaz del módulo se encuentra en *ICameraModule*, la clase abstracta que gestiona los listeners en *ACameraModule* y la interfaz del listener en el que se notifica la captura de las imágenes en *ICameraListener*. Además, el modulo proporciona una clase Frame, que representa a las imágenes capturadas con sus características.

Android Camera2

Para realizar la implementación del módulo se ha empleado la clase *Camera2* que proporciona Android. Esta implementación permite obtener el stream mencionado anteriormente sin mostrar las imágenes en la pantalla, pero produce una velocidad de captura de imágenes baja, de alrededor de dos cuadros por segundo en el smartphone de pruebas (BQ Aquaris M5) variando de teléfono a teléfono.

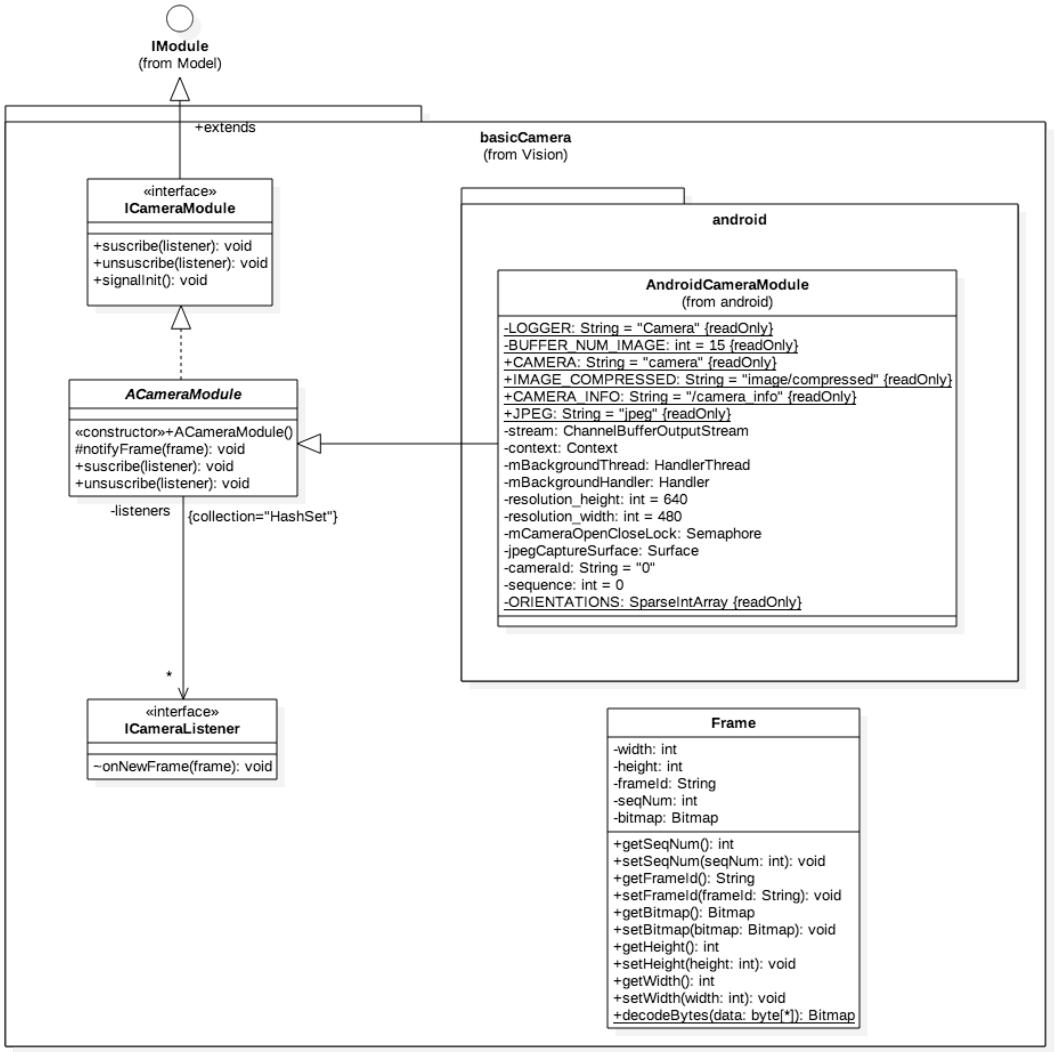

Figura 5.15: Módulo BasicCamera

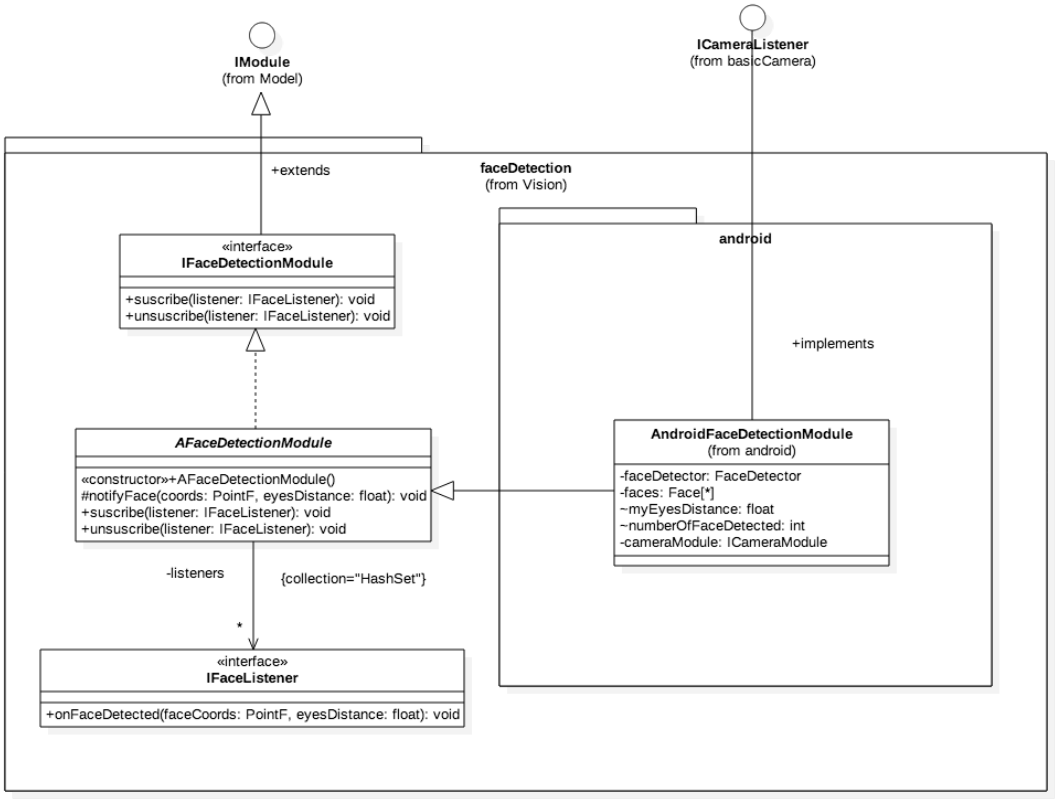

Figura 5.16: Módulo FaceDetection

5.5.4.2. Módulo Face Detection

El módulo FaceDetection(figura 5.16) permite detectar caras en los frames producidos por el módulo BasicCamera. La interfaz del módulo es *IFaceDetectionModule* y su clase abstracta *AFaceDetectionModule*. El listener que debe ser implementado por la clase que utilice el módulo es *IFaceListener*, que notifica de las coordenadas del centro de la cara y de la distancia entre ojos cuando una cara es detectada. Solo se contempla la detección de una cara al mismo tiempo.

Android FaceDetector

Para implementar este módulo se ha empleado la clase *FaceDetector* provista por el paquete Media de Android, el módulo se llama *AndroidFaceDetectionModule*.

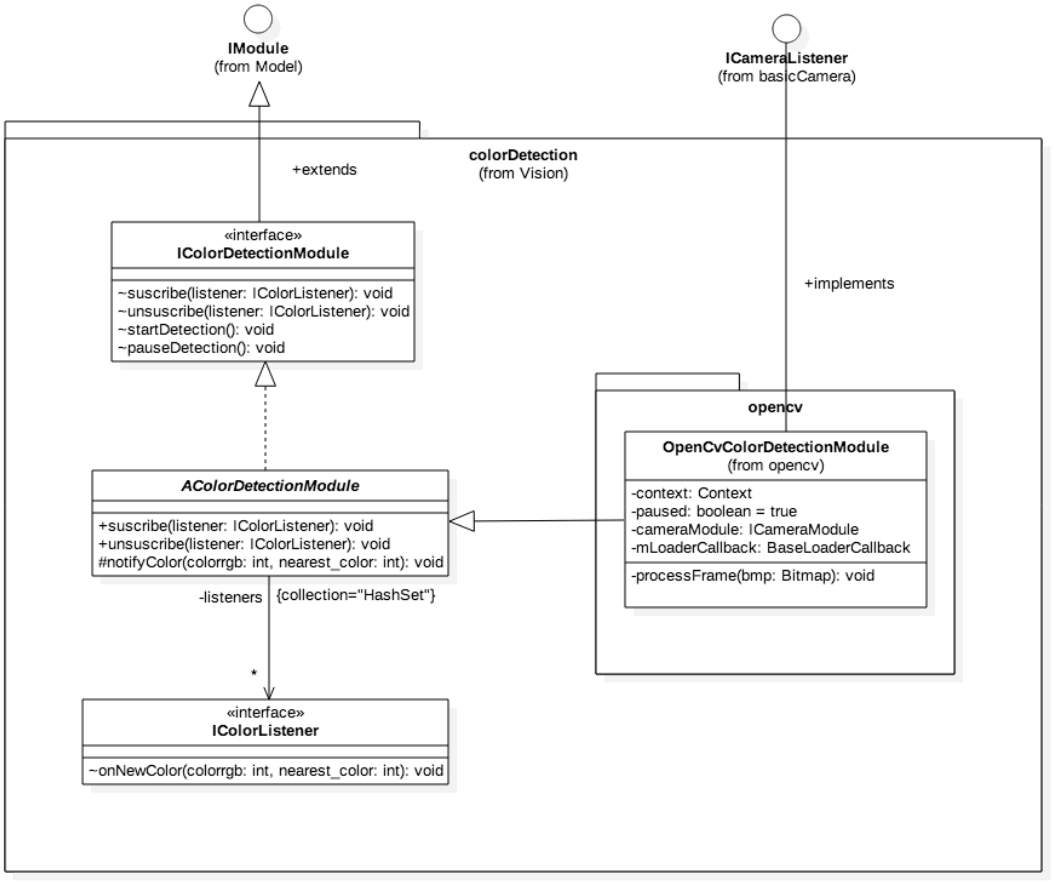

Figura 5.17: Módulo ColorDetection

5.5.4.3. Módulo ColorDetector

Este módulo(figura 5.17) provee la funcionalidad de detección de colores sobre fondos con alto contraste. La interfaz del módulo se puede encontrar en *IColorDetectionModule*, la clase abstracta *AColorDetectionModule* gestiona los listeners en los que será notificada la detección de los colores. La interfaz de dicho listener se encuentra en la clase *IColorListener* y debe ser implementada por toda clase que desee recibir las notificaciones de los colores.

OpenCV

Para realizar la implementación de este módulo se ha empleado la librería OpenCV [14] (sección 4.3) una de las más usadas en visión por computador. Este módulo, contenido en *OpenCVColorDetectionModule* realiza los siguientes pasos

para la detección de colores:

- Detección de bordes mediante el algoritmo de Canny, obteniendo el contorno con mayor area.
- Creación de una máscara binaria con el área detectada
- Conversión de la imagen original al espacio de color HSV
- Media de color en los pixeles no nulos en el canal H de la imagen resultante de un AND entre la máscara y la imagen convertida
- Filtrado por la varianza del color, si supera 1.1 es descartado
- Clasificación del color detectado

Ejemplos de desarrollo

Se desarrolló una aplicación que podría considerarse el primer prototipo del ejemplo *ROBOBO Vigilante*(sección 6.1.2) que cuando detectaba una cara en la imagen capturada por la cámara mandaba un mensaje por correo electrónico al usuario adjuntando la imagen, se puede ver un ejemplo de este mensaje en la figura 5.23. A continuación se muestra la implementación del listener de caras del ejemplo:

```
    @Override
public void onFaceDetected(PointF faceCoords, float eyesDistance) {
    if ((System.currentTimeMillis()-lastDetection)>10000) {
        lastDetection = System.currentTimeMillis();
        msgModule.sendMessage("FACE DETECTED",
            "xxxxxxxxxx@gmail.com", actualFrame.getBitmap());
    }}
```

Para probar el módulo de detección de color se programó una pequeña aplicación que cambiaba el color de fondo de la pantalla acorde al color detectado por el módulo(figuras 5.18,5.19,5.20,5.21).

A continuación se puede ver la implementación del listener de color del ejemplo anterior:

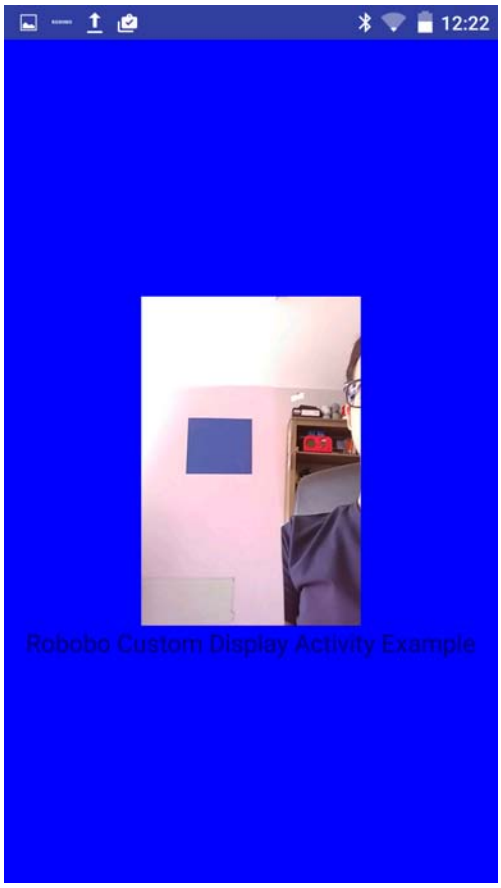

Figura 5.18: Color azul detectado

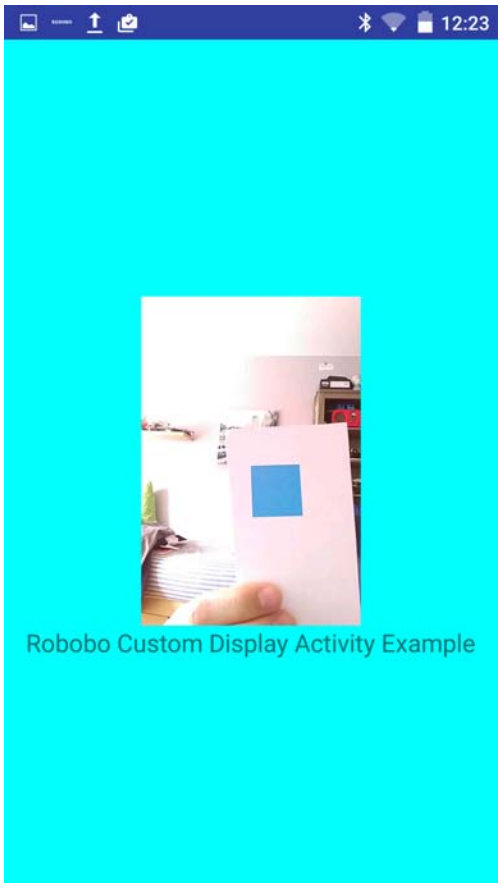

Figura 5.19: Color cían detectado

```
@Override
public void onNewColor(final int colorrgb,final int nearest_color) {
  runOnUiThread(new Runnable() {
        @Override
        public void run() {
            imageView.setImageBitmap(lastFrame.getBitmap());
            rellayout.setBackgroundColor(nearest_color);
        }
    });
}
```

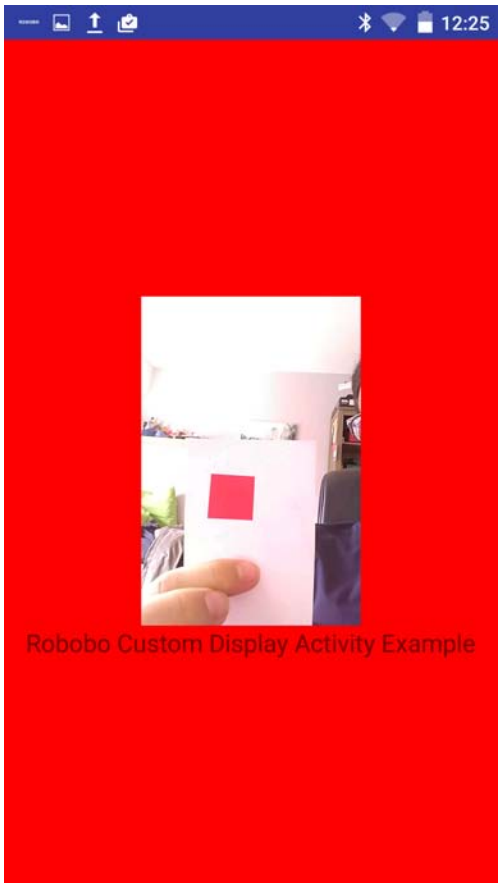

Figura 5.20: Color rojo detectado

Figura 5.21: Color verde detectado

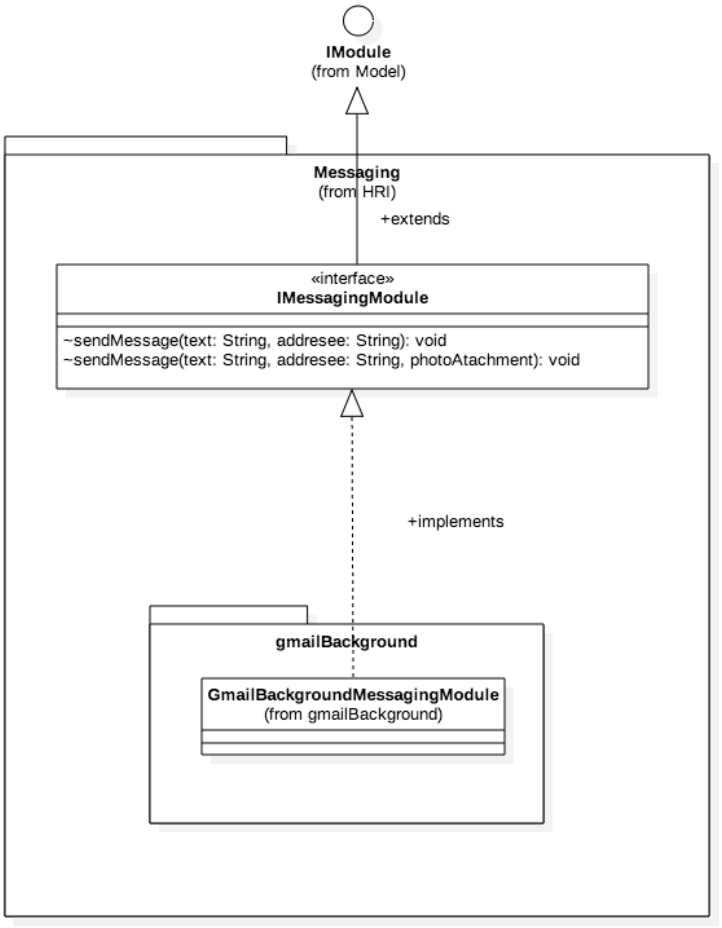

Figura 5.22: Módulo Email

5.5.5. Paquete Messaging

El subsistema Messaging aglomera las diferentes opciones de comunicación por mensajería de texto.

5.5.5.1. Módulo email

El módulo Email(figura 5.22) permite al usuario la comunicación mediante correo electrónico, pudiendo mandar tanto texto como imágenes, por ejemplo, las capturadas con el módulo *BasicCamera*.

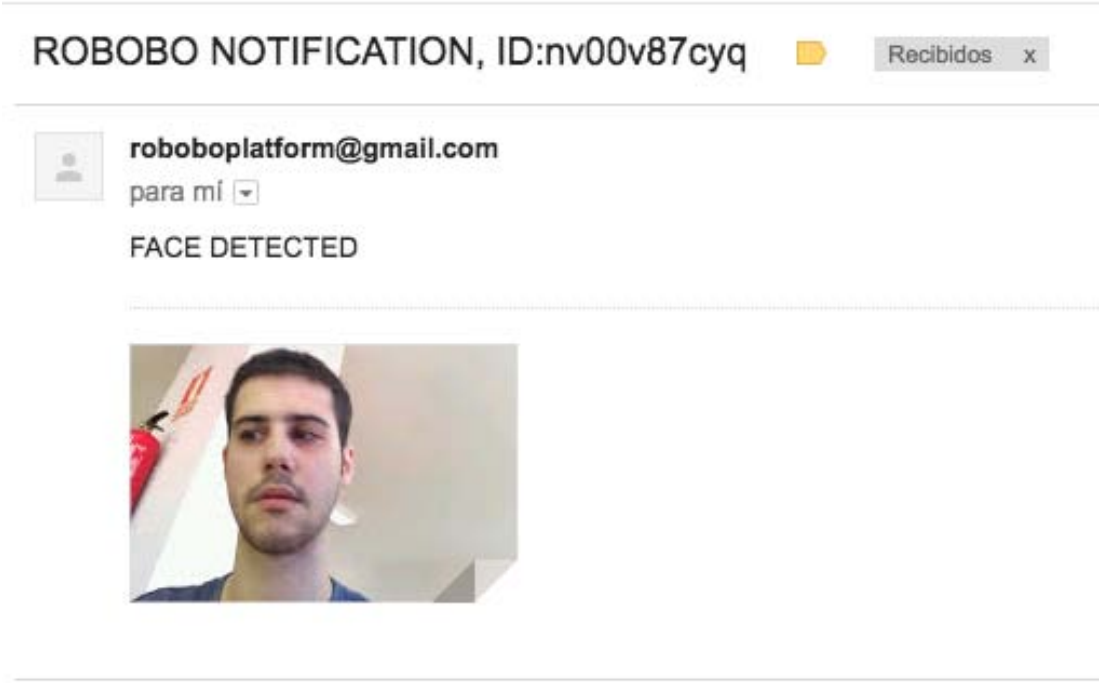

Figura 5.23: Ejemplo de correo electrónico enviado

GmailBackground

Para implementar el módulo, se ha empleado la librería GmailBackGround [15] la cual se presentó en la sección 4.4 de este trabajo , que permite el envío de mensajes de correo electrónico con una cuenta de Gmail. El módulo se encuentra en la clase *GmailBackgroundMessagingModule*.

Ejemplos de desarrollo

Se aprovechó la aplicación de prueba comentada en la sección anterior para comprobar el correcto funcionamiento del módulo de mensajería.

Capítulo 6

Resultados y pruebas

En este capítulo se expondrán los resultados y el funcionamiento de los módulos desarrollados en este trabajo mediante tres aplicaciones de ejemplo.

6.1. Ejemplos de uso

Para probar el correcto funcionamiento de los módulos de interacción, a parte de los ejemplos de desarrollo que se comentaron en el capítulo 5.5, se desarrollaron tres aplicaciones para ROBOBO utilizando las librerías de interacción diseñadas y desarrolladas en este proyecto.

6.1.1. Simón dice musical

El primero de los ejemplos desarrollados es un juego educativo musical, similar al juego Simón dice, en el que los participantes deben repetir una secuencia de colores que se va ampliando en cada iteración de la partida, pero utilizando notas musicales. El objetivo final de este juego es que el usuario consiga asociar los tonos escuchados con la nota musical correspondiente, además de ejercitar la memoria. En este ejemplo se han utilizado los siguientes módulos:

- **SoundDispatcherModule:** Necesario para el funcionamiento de los módulos de sonido
- **PitchDetectionModule:** Usado indirectamente por el NoteDetectionModule

- **NoteDetectionModule:** Utilizado a la hora de detectar las notas musicales producidas por el usuario.
- **ClapDetectionModule:** Se utiliza para iniciar el juego, una primera palmada prepara al robot y la segunda inicia el juego.
- **EmotionSoundModule:** Diferentes sonidos son reproducidos en función del resultado del juego.
- **NoteGeneratorModule:** Se emplea para generar los diferentes tonos que debe reconocer el jugador.
- **SpeechProductionModule:** Se usa para dar información al usuario y dar una mayor sensación de interacción con el robot

Además de los módulos creados en este trabajo, se emplean varios módulos que provee el ROBOBO! Framework:

- **EmotionModule:** Provee la cara del robot, con la posibilidad de cambiar sus expresiones, es la interfaz gráfica que se le muestra al usuario.
- **DefaultMovementModule:** Permite el uso de la plataforma robótica de forma simple, permitiendo el movimiento del robot.
- **IRobInterfaceModule:** Proporciona un método para obtener la clase IRob, que se emplea para un control más avanzado del Rob, por ejemplo, para utilizar los leds de la base.

El robot comienza la partida desactivado (figura 6.3), el jugador inicia la partida dando una palmada, momento en el que el robot pronunciará a través del modulo de voz «Ready»(6.4 , una segunda palmada comenzará la partida en si, el robot dirá una frase retando al usuario, por ejemplo «follow me if you can», y procederá a reproducir un tono musical. En el momento en el que termine el tono, cambiará la cara (figura 6.5 dando a entender que está escuchando al usuario, en el momento en el que el jugador termine de producir la nota se comprueba si corresponde al tono original, si coincide el robot realizará una pequeña celebración y volverá a producir tonos, añadiendo uno más a los reproducidos anteriormente. En caso de que el jugador falle, el robot pondrá cara de disgusto (figura 6.6),

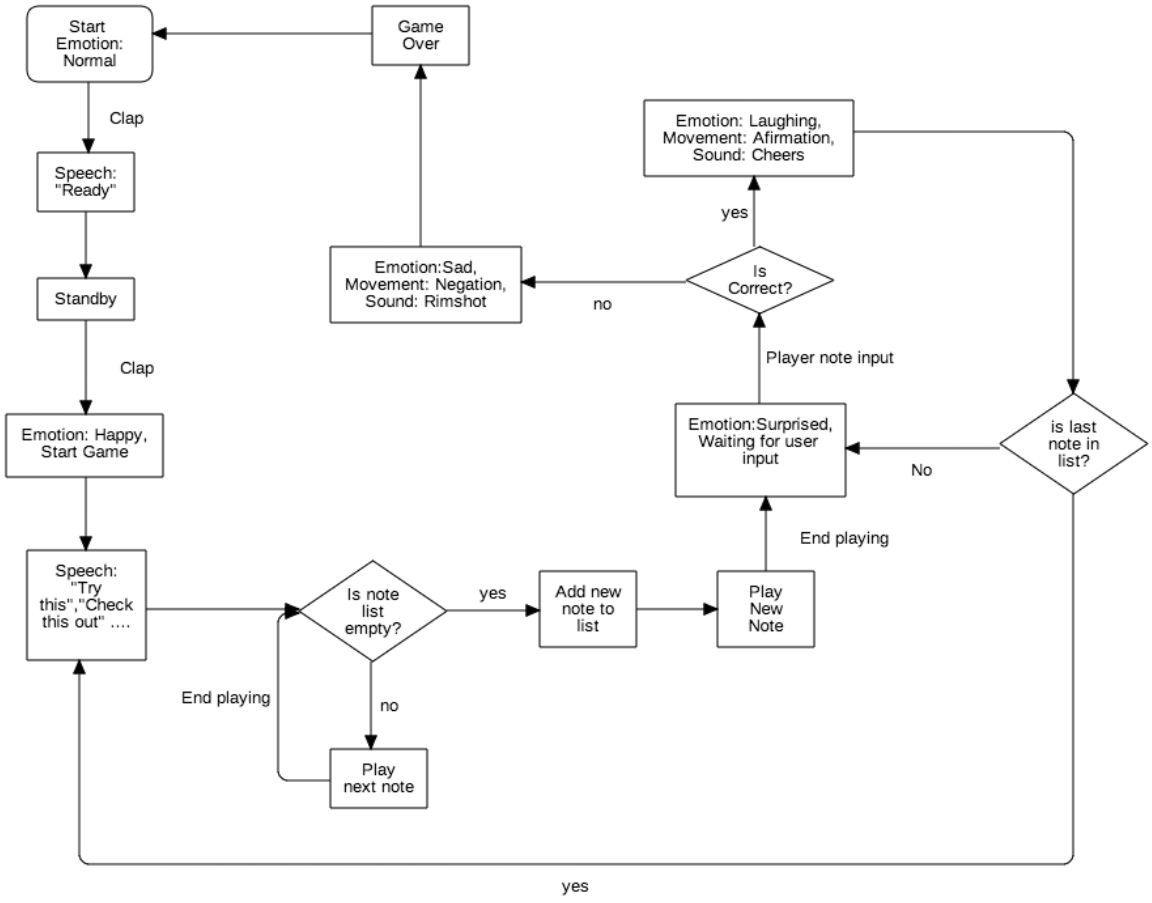

Figura 6.1: Diagrama de flujo del ejemplo musical

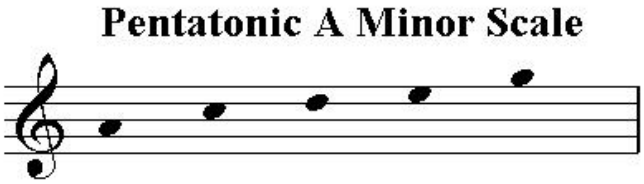

Figura 6.2: Escala pentatonica menor de La

producirá un sonido de abucheo y volverá al estado inicial, esperando a una palmada para iniciar el juego. Este proceso puede observarse en el diagrama de flujo que se muestra en la figura 6.1.

Figura 6.3: ROBOBO Desactivado

Figura 6.4: ROBOBO en Standby

En este ejemplo se pueden ver en funcionamiento múltiples módulos de interacción, por ejemplo a continuación puede verse el código empleado para iniciar la partida, que hace uso del listener *OnClap* que proporciona el *ClapDetectionModule*.

```
@Override
    public void onClap(double time) {
        if ((ready)&&(!playing)){
            Log.d(TAG,"PLAYING");
            playing = true;
```

Figura 6.5: ROBOBO escuchando al usuario

Figura 6.6: ROBOBO tras fallar la nota

```
            startGame();
        }
        if ((!ready)&&(!playing)){
            Log.d(TAG,"READY");
            ready = true;
            emotionModule.setCurrentEmotion(Emotion.SMYLING);
            speechModule.sayText("Ready",0);
        }
    }
```

El siguiente fragmento de código es la implementación del listener *onNoteEnd* del módulo *NoteDetectionModule*, que es usado a la hora de detectar las notas que el jugador hace sonar. Para evitar falsos positivos, solo se tienen en cuenta las notas cuya duración supera los 200 milisegundos.

```
@Override
    public void
        onNoteEnd(com.mytechia.robobo.framework.hri.sound.noteDetection.Note
        note, long time) {
```

```
        Log.d(TAG, "NOTEEND!!!!!!! Playing: "+playing+" playingnotes
            "+playingnotes+" listening "+listening);

        if ((time>200)&&(playing)&&(!playingnotes)&&(listening)) {
            try {
                Log.d(TAG,"CheckResult");
                boolean checkresult = checkNote(convertNote(note));
                if (!checkresult){
                    Log.d(TAG,"GameOver");
                    gameOver();
                }
            }catch (NoSuchElementException e){
                Log.d(TAG,"EndList");
                listening = false;
                movTask = new AsienteClass();
                timermovement.schedule(movTask,0);
            }
        }
    }
```

También se ha implementado el listener de final de reproducción de secuencia *onNoteSequenceEnd* en el que se crea la lista de notas a comprobar y se indica al sistema que debe comenzar a escuchar.

```
    @Override
    public void onSequencePlayEnd() {
        emotionModule.setCurrentEmotion(Emotion.SURPRISED);
        Log.d("TAG","PLAYING = FALSE");
        playingnotes = false;

        listening = true;
        checkNotes = new LinkedList<>(notes);
        Log.d("TAG", checkNotes.toString());

    }
```

En esta implementación del juego, para que tenga una sonoridad agradable, solamente se producen tonos dentro de la escala pentatónica menor de La (figura 6.2)

6.1.2. ROBOBO Vigilante

El siguiente ejemplo es un sistema de seguridad que utiliza las capacidades del ROBOBO de reconocer caras, detectar sonidos, producir sonidos y mandar correos electrónicos a modo de cámara de seguridad con aviso automático. En este ejemplo se han utilizado los siguientes módulos desarrollados en el trabajo:

- **EmotionSoundModule**: Diferentes sonidos son reproducidos en función del resultado del juego.
- **SpeechProductionModule**: Se usa para dar información al usuario y dar una mayor sensación de interacción con el robot
- **SpeechRecognitionModule**: Usado para activar y desactivar el modo de vigilancia
- **BasicCameraModule**: Permite la captura de imágenes desde la cámara en segundo plano
- **FaceDetectionModule**: Permite la detección de caras para poder notificar al usuario.
- **MessagingModule**: Usado para comunicar al usuario por correo electrónico cualquier posible intrusión del espacio vigilado.

Además de los módulos creados en este trabajo, se emplean varios módulos proporcionados ROBOBO! Framework:

- **EmotionModule**: Provee la cara del robot, con la posibilidad de cambiar sus expresiones, es la interfaz gráfica que se le muestra al usuario.
- **DefaultMovementModule**: Permite el uso de la plataforma robótica de forma simple, permitiendo el movimiento del robot.

- **IRobInterfaceModule**: Proporciona un método para obtener la clase IRob, que se emplea para un control más avanzado del ROB, por ejemplo, para utilizar los leds de la base.

En la figura 6.12 se puede ver el diagrama del flujo de ejecución de este ejemplo. El usuario activará el modo de vigilancia mediante comandos de voz, con la frase «Rob activate now», en este momento el *Pan* del ROB comenzará a girar describiendo un arco de 180 grados y se activarán los diferentes sistemas de seguridad (figura 6.8). Con el modo de seguridad activado, y la configuración por defecto, el robot notificará al usuario cada vez que detecte una cara enviándole un correo electrónico una notificación con una imagen adjunta de la cara que hizo saltar el detector (figura 6.11). En el momento en que es detectado un intruso, el ROB activará las luces de alarma, parpadeo en azul y rojo, activará una alarma acústica y pondrá una expresión de enfado en su cara (figura 6.9). Los modos de actuación del robot podrán ser configurados mediante voz, siempre y cuando el modo de vigilancia esté desactivado, mediante los siguientes comandos:

- *Rob switch silent mode now*: En este modo el ROBOBO no se mueve ni activa ninguna clase de alarma que pueda ser detectada por el intruso, pero sigue notificando al usuario (figura 6.10).
- *Rob switch face detection now*: Activa y desactiva los avisos por detección de caras.
- *Rob switch movement now*: Activa y desactiva el movimiento del robot.

También podrá consultar el estado de las opciones de vigilancia mediante el comando de voz *Rob status*. Para desactivar la vigilancia se utilizará el comando *Rob deactivate now* volviendo al modo de standby (figura 6.7).

La gramática definida para este ejemplo es la siguiente:

Figura 6.7: ROBOBO en standby

Figura 6.8: ROBOBO en modo patrulla

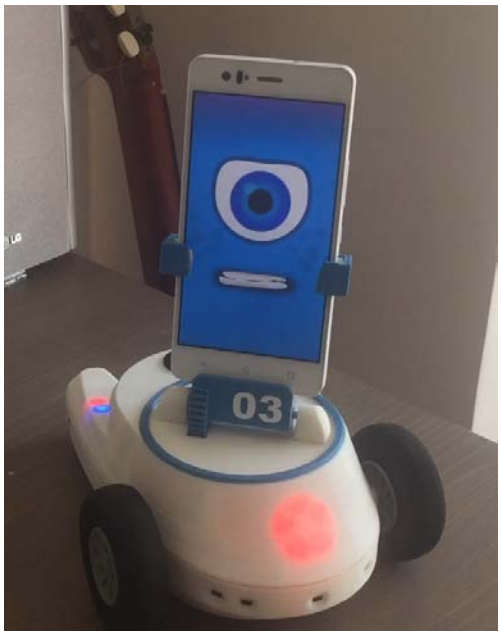

Figura 6.9: ROBOBO con la alarma activada

Figura 6.10: ROBOBO con la alarma activada en modo silencioso

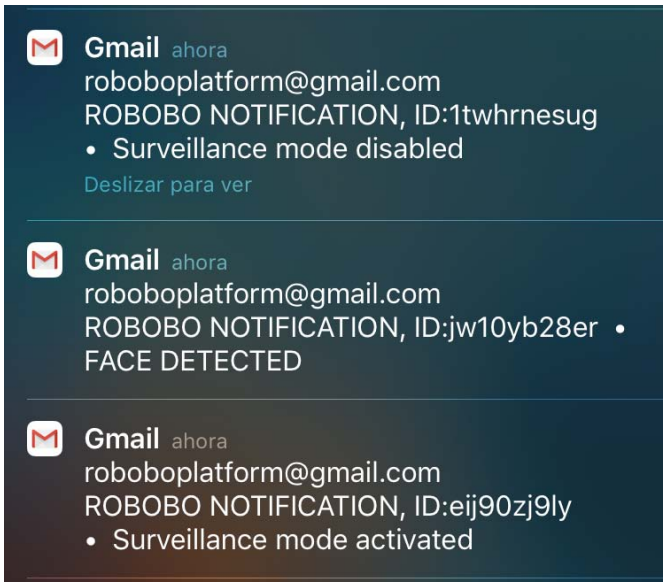

Figura 6.11: Mensajes enviados por el ROBOBO

```
#JSGF V1.0;
grammar voicecontrol;

<starter> = rob;
<activation> =activate|deactivate;
<mode>= face detection|sound detection|silent mode|movement;
<ending> = now;

<options> = <starter> switch <mode> <ending>;
<activations> = <starter> <activation> <ending>;

public <order> = <options>|<activations>;
```

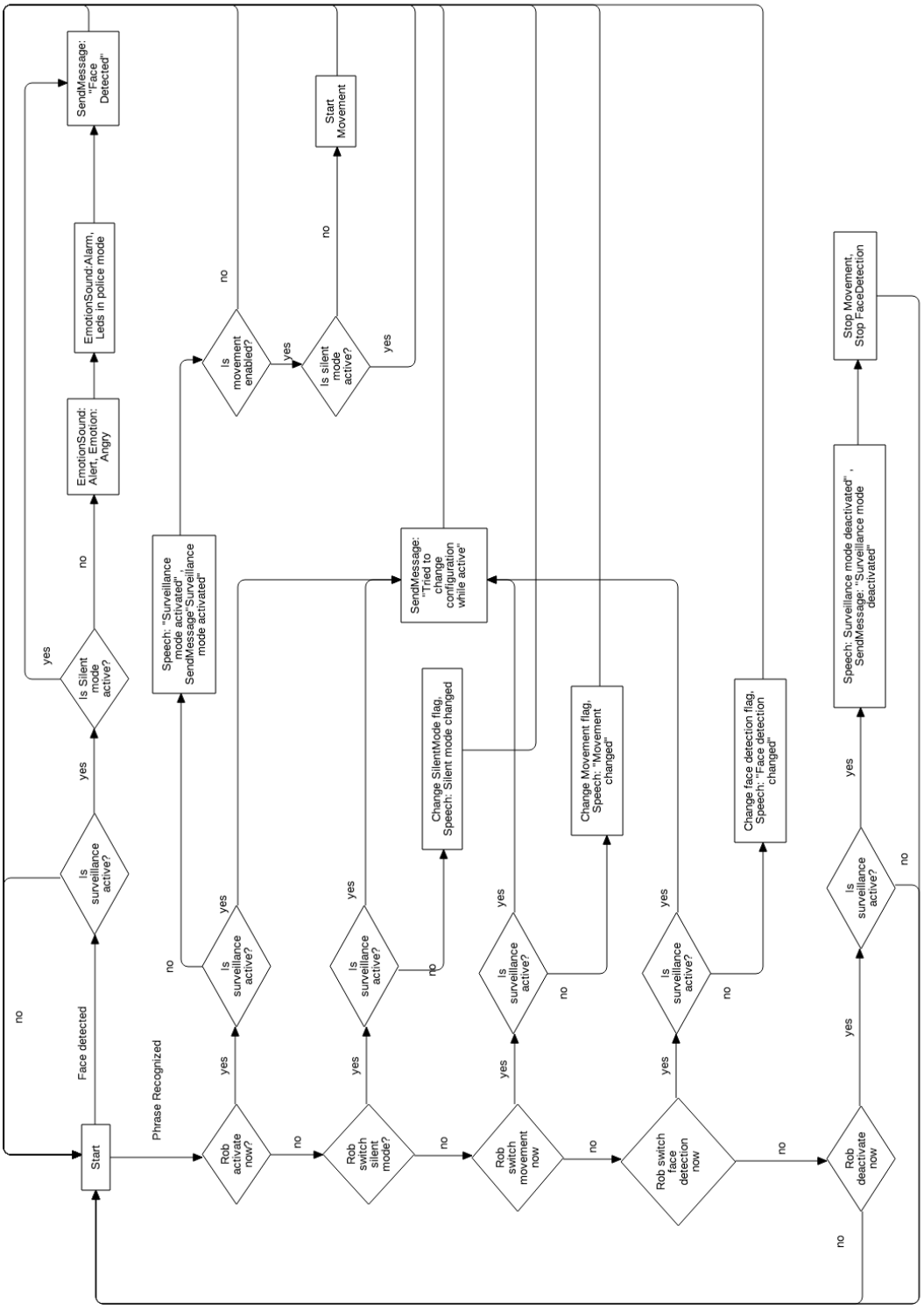

Figura 6.12: Diagrama del flujo ejecución del ejemplo de vigilancia

A continuación se puede ver un fragmento de código, la implementación del listener de detección de caras, en el cual se puede ver en funcionamiento el módulo de mensajería y el de sonidos prefijados:

```
@Override
public void onFaceDetected(PointF faceCoords, float eyesDistance) {
    if (facedetection&&active) {
        if ((System.currentTimeMillis() - lastDetection) > 10000) {
            lastDetection = System.currentTimeMillis();
            if (!silentmode) {
                policeTask = new PoliceTask();
                policeTimer.scheduleAtFixedRate(policeTask, 2000,
                    200);
                stopTimer.schedule(new StopTask(), 10000);
                emotionSoundModule.playSound(IEmotionSoundModule.ALERT_SOUND);
                alarmTimer.schedule(new AlarmTask(),1000);
            }
            emotionModule.setTemporalEmotion(Emotion.ANGRY, 10000,
                Emotion.NORMAL);
            msgModule.sendMessage("FACE DETECTED",
                "xxxxxxxxx@gmail.com", actualFrame.getBitmap());
        }
    }
}
```

En el siguiente fragmento se puede observar un trozo del listener del módulo *SpeechRecognition*, en el cual se muestra la detección de frases y el módulo de producción de habla en funcionamiento:

```
@Override
public void phraseRecognized(String phrase, Long timestamp) {
    String status = "";
    if (!active){
    if (phrase.equals("rob activate now")) {
        msgModule.sendMessage("Surveillance mode activated",
            "lfllamas93@gmail.com");

        active = true;
```

```
        speechProductionModule.sayText("Surveillance mode is now
            active",ISpeechProductionModule.PRIORITY_HIGH);
        startSurveillance();

    }

    if (phrase.equals("rob switch silent mode now")) {
        silentmode = !silentmode;
        if (silentmode){
            status = "active";
        }else{
            status = "disabled";
        }
        speechProductionModule.sayText("Silent mode is now
            "+status,ISpeechProductionModule.PRIORITY_HIGH);

    }
```

Este ejemplo no trata de ser un sistema de seguridad eficaz, sino demostrar una posible aplicación de los módulos desarrollados en este trabajo.

6.1.3. ROBOBO Mascota

Este este ejemplo, el ROBOBO interactuará a modo de mascota con el usuario, le pedirá que lo alimente, tratará de llamar la atención a su «cuidador» y reaccionará a sus acciones.

En este ejemplo se han utilizado de forma conjunta la mayoría de los módulos desarrollados para permitir una interacción fluida con el robot. Los módulos empleados se presentan a continuación:

- **EmotionSoundModule**: Diferentes sonidos son reproducidos en función de la interacción con el robot.
- **SpeechProductionModule**: Usado para transmitir las necesidades del robot al usuario.
- **SpeechRecognitionModule**: Usado para la comunicación por voz con el robot
- **BasicCameraModule**: Permite la captura de imágenes desde la cámara en segundo plano
- **FaceDetectionModule**: Permite la detección de caras y que el robot reaccione en función de su proximidad.
- **TouchModule**: Permite la interacción con el robot mediante gestos táctiles, por ejemplo acariciándolo o haciéndole cosquillas.
- **NoteProductionModule**: Permite al robot realizar secuencias de notas para llamar la atención al usuario.
- **ColorDetectionModule**: Permite la «alimentación» del robot mediante colores.

Además de los módulos creados en este trabajo, se emplean varios módulos proporcionados por el ROBOBO! Framework:

- **EmotionModule**: Provee la cara del robot, con la posibilidad de cambiar sus expresiones, es la interfaz gráfica que se le muestra al usuario.

- **DefaultMovementModule**: Permite el uso de la plataforma robótica de forma simple, permitiendo el movimiento del robot.
- **IRobInterfaceModule**: Proporciona un método para obtener la clase IRob, que se emplea para un control más avanzado del ROB, por ejemplo, para utilizar los leds de la base.

Este ejemplo no sigue un camino de ejecución definido como los ejemplos anteriores, sino que es guiado por la interacción con el usuario y una serie de eventos aleatorios, dichos eventos son los siguientes:

- Movimientos: De vez en cuando el robot realizará diferentes movimientos, por ejemplo, girando sobre sí mismo, para llamar la atención del usuario.
- Caricias: A veces pedirá al usuario que lo acaricie.
- Hambre: Un contador de hambre irá disminuyendo a lo largo del tiempo, en el momento que baje de cierto umbral pedirá al usuario que lo alimente con un color determinado
- Sed: De manera semejante al hambre, un contador de sed irá disminuyendo y en el caso de ser bajo, el usuario deberá darle de beber con el color azul.
- Silbidos: Para llamar la atención del usuario el robot hará sonar una melodía aleatoria a modo de silbido.

La interacción mediante caras depende de la distancia a la que se encuentre el usuario, si está demasiado cerca, el ROBOBO tendrá verguenza y se alejará(figura 6.13), sin embargo si considera que el usuario esta muy lejos le dirá que se acerque. Si además cuando detecte una cara es la primera detección, saludará al usuario y se presentará.

El siguiente fragmento de código es la implementación del listener del detector de caras, empleado para saber la distancia del usuario al robot:

```
//region FaceListener
 @Override
 public void onFaceDetected(PointF faceCoords, float eyesDistance) {
     Log.d(TAG,"Eyes distance: "+eyesDistance);
     if (eyesDistance>150){
```

```
            emotionModule.setTemporalEmotion(Emotion.EMBARRASED,2000,Emotion.NORMAL);
            speechModule.sayText("You are too
                close!",ISpeechProductionModule.PRIORITY_LOW);
            try {
                movementModule.moveBackwardsTime((short)20,2000);

            } catch (InternalErrorException e) {
                e.printStackTrace();
            }
        }else if (eyesDistance<50){
            emotionModule.setTemporalEmotion(Emotion.SURPRISED,2000,Emotion.NORMAL);
            speechModule.sayText("You are too far! Come
                here!",ISpeechProductionModule.PRIORITY_LOW);

        }else{
            if (firstface){
                firstface = false;
                emotionModule.setTemporalEmotion(Emotion.HAPPY,5000,Emotion.NORMAL);
                speechModule.sayText("Hey there! Im robobo!",
                    ISpeechProductionModule.PRIORITY_HIGH);

            }
        }
    }
    //endregion
```

La interacción táctil es diferente según el gesto que se realice:

- Tap: dependiendo de donde sea el toque reaccionará de forma distinta
  - Ojo: Se quejará por meterle el dedo en el ojo y retrocederá un poco(figura 6.15)
  - Boca: Reaccionará de manera similar a con el ojo
  - Resto del la pantalla: Pondrá cara de risa y dirá que tiene cosquillas (figura 6.16)
- Fling: Ejecutará diferentes movimientos en función de la dirección del fling

Figura 6.13: ROBOBO retrocediendo de una cara cercana

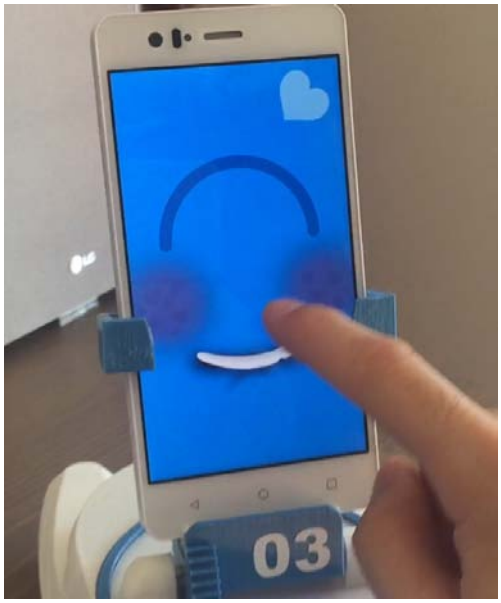

Figura 6.14: ROBOBO ante caricias

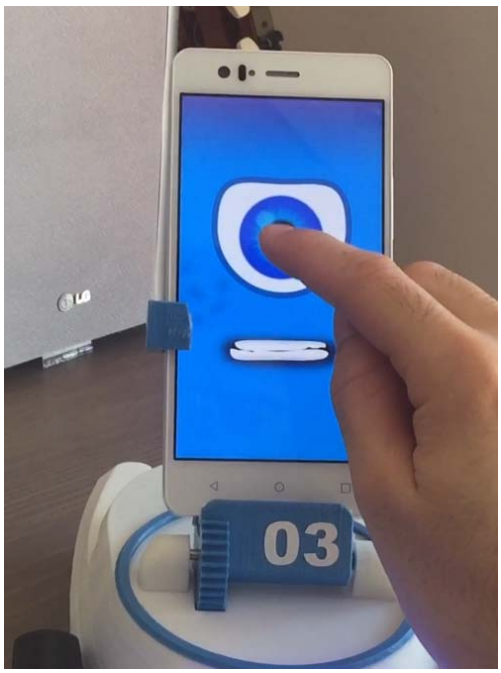

Figura 6.15: ROBOBO tras tocarle el ojo

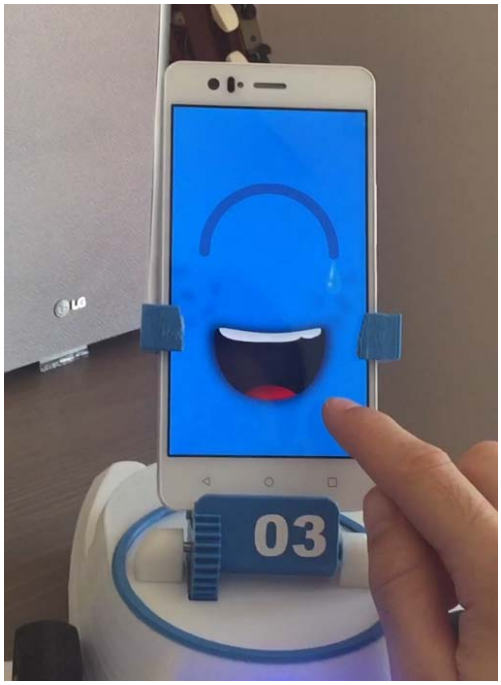

Figura 6.16: ROBOBO con cosquillas

- Touch: El tilt del ROB volverá a su posición original
- Caress: Solo reaccionará si ha pedido previamente una caricia, agradeciéndoselo al usuario(figura 6.14).

A continuación se ve la implementación del listener del tap:

```
@Override
 public void tap(Integer x, Integer y) {
     Log.d(TAG,"X: "+x+" Y: "+y);
     if ((x>230)&&(x<900)&&(y>425)&&(y<1025)){
         emotionModule.setTemporalEmotion(Emotion.ANGRY,2000,Emotion.NORMAL);
         speechModule.sayText("Ouch! Don't poke my
             eye!",ISpeechProductionModule.PRIORITY_HIGH);

     }else if ((x>250)&&(x<800)&&(y>1285)&&(y<1500)){
         emotionModule.setTemporalEmotion(Emotion.ANGRY,2000,Emotion.NORMAL);
         speechModule.sayText("Dont put your finger in my
             mouth!",ISpeechProductionModule.PRIORITY_HIGH);

     }else{
         emotionModule.setTemporalEmotion(Emotion.LAUGHING,2000,Emotion.NORMAL);
         speechModule.sayText("That
             tickles!",ISpeechProductionModule.PRIORITY_HIGH);
     }
 }
```

La interacción por voz se usa para comunicar al robot diferentes cosas:

- *Here comes the food*: Prepara al robot para recibir comida en forma de color
- *Here comes the drink*: Prepara al robot para recibir bebida en forma de color.
- *Hello*: Saluda al robot, que responderá al saludo.
- *How are you*: Pregunta al robot su estado, que responderá con sus niveles de hambre y sed.
- *Do a trick*: Pide al robot que realice un truco.

El módulo de detección de colores solo será activado después de que el usuario manifieste su intención, mediante voz, de alimentar o dar de beber al robot. Si el color mostrado al robot es el adecuado, este dará las gracias y el contador de comida/agua será restablecido al máximo. En caso de fallar el color, el robot se quejará y continuará la detección de colores.

A continuación se muestra la implementación del listener de colores para alimentar al ROBOBO:

```
//region ColorListener
 @Override
 public void onNewColor(int colorrgb, int nearest_color) {
     if (expectedColor == nearest_color){
         speechModule.sayText("Thank You!",
             ISpeechProductionModule.PRIORITY_LOW);
         if ((expectedColor ==
             android.graphics.Color.GREEN)||(expectedColor ==
             android.graphics.Color.RED)){
             foodlevel=5;
         }else{
             waterlevel= 5;
         }
         colorDetectionModule.pauseDetection();
     }else{
         speechModule.sayText("I didnt ask for this!",
             ISpeechProductionModule.PRIORITY_LOW);
     }
 }
 //endregion
```

6.2. Resultados de los ejemplos

Los tres ejemplos fueron probados y respondieron de la manera esperada, sin embargo, como en todo proyecto de HRI, el ajuste es una parte importante del desarrollo y la experiencia de interacción mejorará según el sistema sea usado por diversos usuarios y se reciba información por parte de los mismos que permita un ajuste fino de las librerías desarrolladas.

6.3. Problemas conocidos

- La tasa de refresco del módulo *BasicCamera* es baja y varía mucho entre terminales móviles.
- El *ColorDetectionModule* puede confundirse si el fondo no es homogéneo, se recomienda usar tarjetas con colores sobre fondo blanco.
- El *EmailModule* puede causar el bloqueo de la cuenta de Gmail si esta no es configurada previamente para usar mediante IMAP.
- El *TouchModule* requiere el paso explícito de los TouchEvents de la actividad en pantalla.
- El sistema Android no permite el uso del micrófono por más de un thread al mismo tiempo, lo que hace que los módulos que usan el micrófono de la librería Sound sean incompatibles con el módulo *SpeechRecognition*.

Capítulo 7
Conclusiones

En el primer capítulo se presentó el objetivo de este proyecto, que era el *desarrollo de un framework de interacción básica para el ROBOBO*, a continuación se resumen los resultados:

- **Librería de interacción por voz:** Se diseñaron e implementaron módulos de producción y reconocimiento de habla, que permiten una interacción a través de la voz. El módulo de producción de voz fue implementado mediante el propio motor de síntesis de voz de Android, mientras que en el de reconocimiento se empleó la librería *PocketSphinx*, para permitir el reconocimiento sin conexión a internet.

- **Librería de interacción por sonido:** Se diseñaron y desarrollaron múltiples módulos que permiten la interacción mediante sonidos en ambas direcciones, tanto reconocimiento como producción. Los módulos de reconocimiento de sonidos fueron implementados mediante la librería *Tarsos DSP*, mientras que los de producción utilizan las propias herramientas disponibles en la API de Android.

- **Librería de interacción táctil:** Se diseñó e implementó una librería que permite la detección de gestos táctiles. Se implementó usando el reconocedor de gestos táctiles provisto por Android.

- **Librería de interacción por imagen:** Se diseñó e implementó una librería que permite el fácil acceso a la cámara sin mostrar las imágenes y permite detectar caras y colores. Tanto el módulo básico de cámara como el de

reconocimiento facial no requieren de librerías adicionales, mientras que el de detección de colores utiliza la librería *OpenCV*.

- **Librería de interacción mediante mensajes:** Se diseñó e implementó una librería que permite al ROBOBO interactuar mediante correos electrónicos con el usuario. Se empleó la librería *GmailBackground* para su implementación.

Para comprobar la validez del framework de interacción, así como sus capacidades para dotar a los desarrolladores de una multitud de opciones de interacción Humano-Robot con las que crear aplicaciones interactivas para el ROBOBO, se han desarrollado tres aplicaciones de ejemplo completas. Estas aplicaciones, en forma de juegos educativos y utilidades, permiten demostrar la utilidad del framework, que con el conjunto de "sentidos" propuesto posibilita la construcción de aplicaciones capaces de interactuar de una forma natural y atractiva para los usuarios:

- **Simón dice musical**: Este juego fue desarrollado con la intención de demostrar el correcto funcionamiento de la librería de sonido, y se emplean todos los módulos de la misma.

- **ROBOBO vigilante**: En este ejemplo se demuestra el buen funcionamiento del módulo de detección de caras, la librería de mensajería, y la librería de interacción por voz.

- **ROBOBO mascota**: En este ejemplo, que busca una interacción natural con el robot a modo de mascota, se emplean la mayoría de las librerías desarrolladas de manera conjunta. Aquí se demuestra el funcionamiento de la librería de interacción táctil.

En conclusión, se han diseñado una serie de librerías que permiten una interacción básica con el ROBOBO, las cuales han sido probadas en los ejemplos del capítulo 6.

7.1. Trabajo Futuro

Dada la naturaleza modular del framework desarrollado, existe una gran cantidad de trabajo que se puede realizar a posteriori, desde mejoras en los módulos actuales hasta la ampliación de las librerías de interacción implementados o incluso la incorporación de nuevas librerías que permitan otro tipo de interacción.

A continuación se presenta un listado de posibles líneas de trabajo futuro:

- Mejora de los módulos de visión para aumentar la tasa de refresco
- Aumentar la funcionalidad del módulo de mensajería utilizando otros medios, como Twitter o Telegram.
- Aumentar la funcionalidad de la librería de visión con un módulo de detección de formas simples.
- Crear un subsistema de interacción mediante movimientos del ROB.
- Buscar una forma de que los módulos que emplean el micrófono puedan funcionar simultáneamente.

Apéndice A

Manual de uso

En este apéndice se detallan aspectos de los módulos que no se encuentran en la propia documentación en Javadoc del código, y que son necesarios para el buen funcionamiento de los mismos.

A.1. Carga de módulos

A la hora de cargar los módulos necesarios para la aplicación, se necesita declararlos en un fichero *modules.properties* que debe colocarse en la carpeta **assets** de la aplicación. Los módulos son declarados de la forma que se indica en la figura A.1, escribiendo el espacio de nombres hasta llegar a la implementación específica del módulo que se desee emplear. El orden de la declaración de módulos es importante, ya que es el que se seguirá a la hora de cargarlos al ejecutar la aplicación se deben tener en cuenta las dependencias entre módulos, por ejemplo, no declarando el *FaceDetectionModule* antes que el *BasicCameraModule*.

Para usar los módulos declarados en el fichero de propiedades es necesario utilizar la clase *RoboboManager* que proporciona el método *getModuleInstance*.

```
#######SIMON ROB##########
robobo.module.0=com.mytechia.robobo.framework.hri.sound.soundDispatcherModule.TarsosDSP.TarsosDSPSoundDispatcherModule
robobo.module.1=com.mytechia.robobo.framework.hri.sound.pitchDetection.TarsosDSP.TarsosDSPPitchDetectionModule
robobo.module.2=com.mytechia.robobo.framework.hri.sound.noteDetection.TarsosDSP.TarsosDSPNoteDetectionModule
robobo.module.3=com.mytechia.robobo.framework.hri.sound.emotionSound.android.AndroidEmotionSoundModule
robobo.module.4=com.mytechia.robobo.framework.hri.sound.noteGeneration.android.AndroidNoteGenerationModule
robobo.module.5=com.mytechia.robobo.framework.hri.emotion.DefaultEmotionModule
robobo.module.6=com.mytechia.robobo.framework.hri.speech.production.android.AndroidSpeechProductionModule
robobo.module.7=com.mytechia.robobo.framework.hri.sound.clapDetection.tarsosDSP.TarsosDSPClapDetectionModule
robobo.module.8=com.mytechia.robobo.rob.BluetoothRobInterfaceModule
robobo.module.9=com.mytechia.robobo.rob.movement.DefaultRobMovementModule
```

Figura A.1: Declaración de los módulos para el ejemplo del Simon dice musical

En la figura A.2 se puede ver el proceso de instanciación e inicialización para los módulos del ejemplo del Simón dice musical. En dicha figura *robobo* es la instancia del *RoboboManager*.

A.2. Librería Speech

A.2.1. Módulo Production

El módulo SpeechProduction no requiere de ninguna consideración especial.

A.2.2. Módulo Recognition

Para el uso del modulo Recognition implementado con PocketSphinx es necesario tener en cuenta ciertos aspectos previos:

- Debe crearse una carpeta **sync** en el directorio **assets** de la aplicación, en el interior de la cual se deben introducir tanto el modelo de lenguaje como el diccionario provistos con pocketsphinx.
- Dentro de la carpeta sync es necesario un archivo *assets.lst*, en el cual deben declararse todos los archivos que se encuentren dentro del directorio **sync**.
- Todo archivo que se añada o modifique en la carpeta **sync** debe ir acompañado de un archivo que contenga su firma md5, por ejemplo, en el caso de de querer introducir una nueva gramática *gramatica.gram*, deberá ser acompañado por el archivo *gramatica.gram.md5*. Este fichero md5 no debe ser declarado en el fichero *assets.lst* que se menciona en el punto anterior.
- El inicio del reconocedor de voz se realiza de manera asíncrona e interactuar con el antes de su completo arranque puede producir errores. Para evitar esta clase de fallos se notificará del arranque del mismo a través del evento *onModuleStart()* declarado en el listener *ISpeechRecognitionListener*.
- Para funcionar correctamente debe estar declarado el permiso

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

  en el AndroidManifest.

```
protected void startRoboboApplication() {

    roboboHelper.launchDisplayActivity(WebGLEmotionDisplayActivity.class);

    try {
        soundDispatcherModule=
                robobo.getModuleInstance(ISoundDispatcherModule.class);
        emotionModule =
                robobo.getModuleInstance(IEmotionModule.class);
        speechModule =
                robobo.getModuleInstance(ISpeechProductionModule.class);
        noteDetectionModule=
                robobo.getModuleInstance(INoteDetectionModule.class);
        noteGeneratorModule=
                robobo.getModuleInstance(INoteGeneratorModule.class);
        emotionSoundModule=
                robobo.getModuleInstance(IEmotionSoundModule.class);
        clapDetectionModule=
                robobo.getModuleInstance(IClapDetectionModule.class);
        movementModule=
                robobo.getModuleInstance(IRobMovementModule.class);
        interfaceModule=
                robobo.getModuleInstance(IRobInterfaceModule.class);
    }
    catch(ModuleNotFoundException e) {
        final Exception ex = e;
        runOnUiThread(() → { showErrorDialog(ex.getMessage()); });
    }

    clapDetectionModule.suscribe(this);
    noteGeneratorModule.suscribe(this);
    noteDetectionModule.suscribe(this);

    try {
        interfaceModule.getRobInterface().resetPanTiltOffset();
    } catch (InternalErrorException e) {
        e.printStackTrace();
    }
    soundDispatcherModule.runDispatcher();

}
```

Figura A.2: Instanciación de los módulos para el ejemplo del Simon dice musical

La creación de gramáticas se hace a través del formato JSFG, a continuación se verá un ejemplo:

```
#JSGF V1.0;

grammar movements;

<starter> = rob;
<direction> =front|back|up|down|right|left;
<ending> = now;
public <movements> = <starter> <direction> <ending>;
```

En este ejemplo se define una gramática para detectar comandos de movimiento estructurados. Solo se detectarán aquellas estructuras marcadas como public, como en este caso lo sería *movements*, que esta formada por un símbolo de inicio*starter*, una dirección *direction* y un símbolo de finalización *ending*, así, una frase valida en este sistema podría ser «rob front now», mientras que «front now» o «rob front» serían ignoradas por el sistema.

Se pueden consultar los detalles de este tipo de gramáticas en la página del proyecto [17].

A.3. Librería Touch

A.3.1. Módulo Touch

El único detalle a tener en cuenta a la hora de utilizar el *TouchModule* es la necesidad de pasarle los *TouchEvent* capturados desde la actividad activa en ese momento, en la figura A.3 puede verse el proceso.

A.4. Librería Vision

A.4.1. Módulo BasicCamera

El módulo BasicCamera necesita tener en cuenta varios aspectos para su correcto funcionamiento:

```
@Override
public boolean onTouchEvent(MotionEvent event) {

    touchModule.feedTouchEvent(event);
    return super.onTouchEvent(event);
}
```

Figura A.3: Paso de los TouchEvents al módulo Touch

- Debe colocarse en la carpeta **assets** el archivo *vision.properties*, donde se especificará la resolución de las imágenes capturadas de la siguiente manera:

  ```
  resolution_height= 640
  resolution_width = 480
  ```

- Se necesita el permiso

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

  para su correcto funcionamiento declarado en en *AndroidManifest.xml*

La tasa de captura de imágenes es de 2 frames por segundo en el smartphone empleado para el desarrollo y las pruebas, BQ Aquaris M5, pero se tiene constancia de que esta cifra varía al cambiar de terminal.

A.4.2. Módulo FaceDetection

El único aspecto a tener en cuenta en el módulo de reconocimiento facial es su dependencia hacia el *BasicCameraModule*, que debe ser declarado antes que el de detección de caras en el *modules.properties*

A.4.3. Módulo ColorDetection

Para el correcto funcionamiento del modulo detector de color es necesario tener en cuenta varios detalles:

- Depende directamente del módulo *BasicCameraModule*, que debe ser declarado antes que el de detección de colores en el fichero *modules.properties*

- Depende de la librería OpenCV, por ello, el smartphone que vaya a utilizar este módulo necesitará tener instalada previamente la aplicación **OpenCV-Manager**.
- La resolución de las imágenes a procesar afecta directamente al rendimiento del módulo.

A.5. Librería Sound

A.5.1. Módulo SoundDispatcher

El módulo *SoundDispatcher* debe ser configurado en el archivo *sound.properties* definiendo las siguientes características:

```
samplerate = 44100
buffersize = 2048
overlap = 1024
```

Dichas características se refieren a:

- **Samplerate** hace referencia a la frecuencia de muestreo del sonido.
- **Buffersize** define el tamaño del buffer de audio.
- **Overlap** define el solapamiento entre muestras adyacentes.

Este módulo requiere el permiso

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

declarado en el AndroidManifest de la aplicación.

Para comenzar el procesado de audio debe llamarse al método del módulo *runDispatcher()*, de lo contrario no funcionará ningún módulo que dependa de este.

A.5.2. Módulo PitchDetection

El modulo *PitchDetection* depende directamente del módulo *SoundDispatcher*, que debe ser declarado en el archivo *modules.properties* antes que el *PtichDetectionModule*.

A.5.3. Módulo ClapDetection

Deben tenerse en cuenta varios detalles antes de poder utilizar el módulo *ClapDetection*:

- Depende directamente del módulo *SoundDispatcher* que debe ser declarado previamente en el *modules.properties* de la aplicación.
- Este módulo debe ser configurado en el archivo *sound.properties*, definiendo las siguientes características, siendo ambas parametros de sensibilidad del detector de palmadas:

```
clap_threshold = 8
clap_sensitivity = 50
```

A.5.4. Módulo NoteDetection

Deben tenerse en cuenta varios detalles antes de poder utilizar el módulo *NoteDetection*:

- Depende directamente del módulo *PitchDetection*, y por tanto del *SoundDispatcher* también, que deben ser declarados previamente en el *modules.properties* de la aplicación.
- Este módulo debe ser configurado en el archivo *sound.properties*, definiendo las siguientes características

```
minThreshold = 0.1
maxThreshold = 0.9
```

  Las dos características hacen referencia al porcentaje de desafinado permitido respecto al índice de una nota para considerarla válida.

A.5.5. Módulo NoteGenerator

El módulo *NoteGenerator* no requiere de indicaciones especiales.

A.5.6. Módulo EmotionSound

El módulo *EmotionSound* debe ser empleado con cautela, ya que si dos sonidos son reproducidos en un corto intervalo de tiempo podrían solaparse.

A.6. Librería Messaging

A.6.1. Módulo Messaging

Varios aspectos deben tenerse en cuenta antes de usar el módulo de mensajería:

- Debe configurarse la cuenta de correo que se vaya a utilizar para el envío de mensajes en el archivo *messaging.properties*

```
emailaccount = xxxxxxxxxxxx@gmail.com
emailpasswd = xxxxxxxx
```

- Deben añadirse los siguientes permisos al AndroidManifest de la aplicación:

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Bibliografía

[1] "D.r.e.a.m. project," http://www.robotsthatdream.eu/, accessed: 2016-15-6.

[2] "Robot operating system (ros)," http://www.ros.org/, accessed: 2016-08-14.

[3] M. A. Goodrich and A. C. Schultz, "Human-robot interaction: a survey," *Foundations and trends in human-computer interaction*, vol. 1, no. 3, pp. 203–275, 2007.

[4] "Worker at volkswagen plant killed in robot accident," http://www.ft.com/cms/s/0/0c8034a6-200f-11e5-aa5a-398b2169cf79.html, accessed: 2016-08-19.

[5] J. A. Corrales Ramon, G. J. Garcia Gomez, F. Torres Medina, and V. Perdereau, "Cooperative Tasks between Humans and Robots in Industrial Environments," *International Journal of Advanced Robotic Systems*, vol. 9, no. 94, pp. 1–10, Jun. 2012. [Online]. Available: http://hal.upmc.fr/hal-00737775

[6] "Definition of service robots," http://www.ifr.org/service-robots/ , accessed: 2016-08-19.

[7] "Paro robots," http://www.parorobots.com/index.asp, accessed: 2016-08-19.

[8] "Nao robot, aldebaran robotics," https://www.ald.softbankrobotics.com/en/cool-robots/nao/find-out-more-about-nao, accessed: 2016-08-19.

[9] "Sony aibo — the history of the robotic dog," http://www.sony-aibo.com/, accessed: 2016-08-21.

[10] "Scratch language project main page," https://scratch.mit.edu/, accessed: 2016-08-20.

[11] "Lenguaje gráfico bitbloq," accessed: 2016-08-28.

[12] "Cmu sphinx open source speech recognition kit," http://cmusphinx.sourceforge.net/, accessed: 2016-08-6.

[13] J. Six, O. Cornelis, and M. Leman, "TarsosDSP, a Real-Time Audio Processing Framework in Java," in *Proceedings of the 53rd AES Conference (AES 53rd)*, 2014.

[14] "Open source computer vision library," https://github.com/opencv, 2015.

[15] "Gmail background library," https://github.com/yesidlazaro/GmailBackground, accessed: 2016-08-4.

[16] "Robobo framework," https://bitbucket.org/mytechia/robobo-framework, accessed: 2016-08-13.

[17] "Jspeech grammar format," https://www.w3.org/TR/jsgf/, accessed: 2016-08-6.

[18] A. de Cheveigné and H. Kawahara, "Yin, a fundamental frequency estimator for speech and music," *J Acoust Soc Am*, vol. 111, pp. 1917–1930, 2002. [Online]. Available: http://audition.ens.fr/adc/pdf/2002_JASA_YIN.pdf