ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Teleoperación y control de un robot Lego a través de un terminal Android

AUTOR: Joaquín Roca Soler
DIRECTOR: Juan Ángel Pastor Franco

Septiembre / 2013

Autor	Joaquín Roca Soler
E-mail del autor	joaquinrs91@gmail.com
Director	Juan Ángel Pastor Franco
E-mail del director	JuanAngel.Pastor@upct.es
Título del PFC	Teleoperación y control de un robot Lego a través de un terminal Android
Descriptores	Android, Lego mindstorm, sensores, leJOS, bluetooth

Resumen

Este proyecto tiene como objetivo la interoperabilidad entre un robot Lego y un terminal Android a través de una conexión inalámbrica, para de este modo conseguir un reparto de funcionalidad entre ellos.

Para conseguirlo se utilizará la tecnología de conexión bluetooth, para conectar el robot programado en java, con una aplicación Android.

En el proyecto también se hace uso de los diferentes sensores del robot; como sensor de choque, de proximidad EOPD o de color.

Titulación	Ingeniería Técnica de Telecomunicación, esp. Telemática
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de presentación	Septiembre-2013

Quisiera dar las gracias en primer lugar a mi familia,
por su apoyo y por darme la oportunidad de
aprender cometiendo mis propios errores.
A mis amigos, y compañeros de carrera por
confiar siempre en mí.
Y a Breaking Coco, por darme la oportunidad
de aprender las dificultades del mundo laboral.

Índice

Índice de fuguras

1- Introducción

1.1- Objetivos

Este proyecto tiene como objetivo el aprendizaje del sistema operativo Android [1] y de cómo se realizan aplicaciones para él, mediante el desarrollo de una aplicación que sirva de guía para adentrarse poco a poco en el mundo de Android.

El objetivo de la aplicación es controlar un robot Lego [2] a través de un terminal Android utilizando una conexión inalámbrica. Dado su enfoque docente, la aplicación tiene que ser sencilla y atractiva. Sencilla para no complicar excesivamente la labor docente, ni distraer al alumno con detalles innecesarios. Atractiva para provocar la suficiente motivación.

También se pretendía conseguir un reparto de la funcionalidad entre el terminal Android y la unidad de control del robot Lego, para así asegurar un intercambio de paquetes entre ambos dispositivos acompañado de una sincronización correcta.

Por último se pretendía que la aplicación realizada fuera lo suficiente potente como para poder subirla a la tienda de aplicaciones de Android, acompañada de un manual de uso, para el disfrute de todo aquel lo desee.

1.2- Planteamiento

Para desarrollar este proyecto se han tenido que llevar a cabo dos líneas de aprendizaje, una sobre el robot Lego y la otra sobre programación en Android.

Lo primero que se llevó a cabo fue aprender a realizar pequeños programas para el robot, en el que se fue avanzando poco a poco: desde aprender a controlar sus motores de diferentes formas, hasta la realización de pequeños programas que utilizaban sensores y servían de ejemplo para entender su funcionamiento.

Después se empezó la línea de aprendizaje Android, en el que se estudió el funcionamiento de una aplicación, cuáles eran sus posibles estados, la creación de interfaces gráficas y detección de eventos al pulsar diferentes botones, así como la creación de menús.

Después se procedió a conectar mediante bluetooth [3] el robot y el dispositivo Android, para lo que se estudió en la API del robot cómo funcionan los métodos que controlan su conexión bluetooth. El robot hace de servidor.

Una vez creado un pequeño programa con el robot actuando de servidor, se procedió al estudio del bluetooth en Android. En este caso fue más sencillo, pues la API de Android dispone de un ejemplo de un chat bluetooth en el que se puede ver cómo se realiza la búsqueda de dispositivos y la conexión a ellos, y así adaptarlo a nuestra aplicación.

Cuando se había conseguido que el dispositivo Android se conectara con el robot y que funcionara el intercambio de mensajes en ambas direcciones, se procedió a diseñar el esquema de los diferentes paquetes que se iban a intercambiar entre ambos.

En este punto, se realizó la primera aplicación en la que el Smartphone controlaba al robot indicándole 4 direcciones.

Una vez conseguido un control total del robot mediante el Smartphone, se añadieron al robot los diferentes sensores que usa, creando hilos de envío y recepción en ambos programas y poniendo en uso la opción de inteligencia, que indicaba donde se quería utilizar el módulo de inteligencia.

Una vez que se consiguió que la opción de inteligencia funcionara correctamente en sus tres modos, ya se había conseguido el objetivo de repartir la funcionalidad entre el robot y el terminal Android. Ahora ya solo quedaba añadirle algunas opciones extras. En este caso reproducir un archivo de sonido que se encuentra en la memoria del robot, y crear el programa seguidor de línea [4], para probar también el altavoz del robot y el sensor de color.

1.3- Organización del documento

Este proyecto ha sido dividido en varios capítulos con el fin de facilitar al lector el seguimiento del desarrollo, desde el comienzo, hasta la aplicación ya terminada.

- Capítulo 2: se explica detalladamente cuales han sido las tecnologías implicadas en la implementación, dando al lector una visión general de los puntos fuertes de estas.

- Capítulo 3: se expone una descripción de la aplicación, explicando cual es el funcionamiento de esta, los pasos a seguir para configurar el robot y el dispositivo Android, la descripción de los entornos de desarrollo y su distribución, cómo se compilan y ejecuta la aplicación en el robot y en el dispositivo Android, y por último se facilita un manual de uso.

- Capítulo 4: se explica paso a paso como se ha desarrollado la implementación del programa contenido en el robot y la aplicación Android. Se explica el funcionamiento de todas sus clases, cómo se controlan sus sensores, se realiza la conexión bluetooth y el intercambio de mensajes, incluyendo la gestión de la concurrencia.

- Capítulo 5: se presentan las conclusiones generales de este proyecto, y se exponen una serie de posibles mejoras para este.

Para representar los métodos, tipos de dato, constantes del lenguaje, y sus diferentes palabras reservadas, se usará el tipo de letra `Consolas`.

2- Tecnologías empleadas

2.1- Robot Lego Mindstorms NXT

Lego Mindstorms es un juego de robótica para niños fabricado por la empresa Lego que exhibe todos los elementos básicos de las teorías robóticas, como la unión de piezas y la programación de acciones, en forma interactiva. Este robot fue comercializado por primera vez en septiembre de 1998.

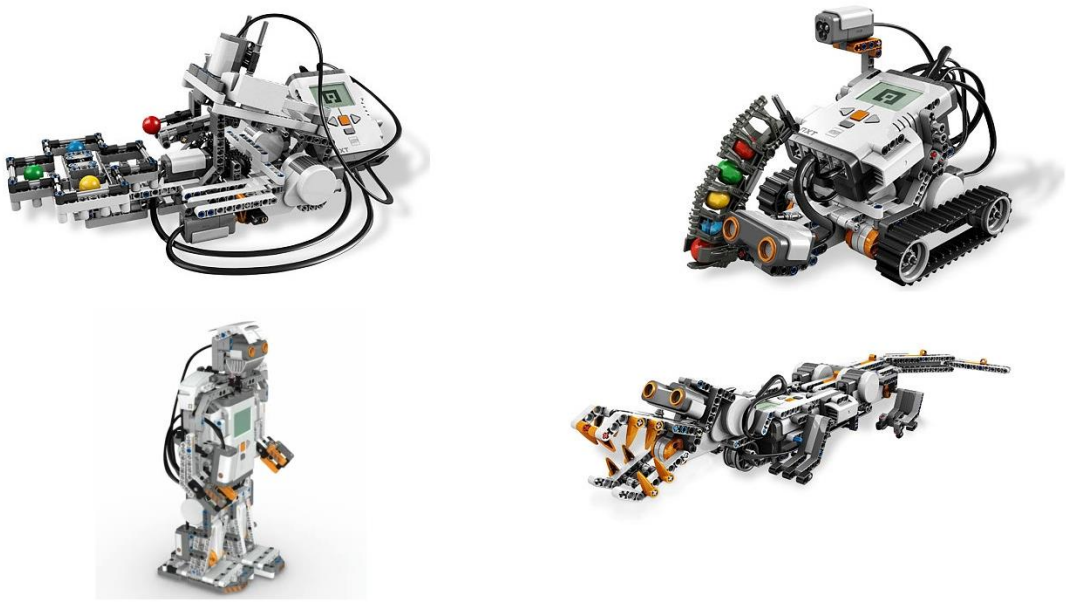

Figura 1. Diferentes montajes de un robot Lego

Puede ser usado para construir un modelo de sistema integrado con partes electromecánicas controladas por computador. Prácticamente todo puede ser representado con las piezas tal como en la vida real, como un elevador o robots industriales.

La construcción del robot se basa en la unión de bloques de plástico, característicos de Lego, junto con piezas plegables y algunas piezas que permiten la rotación de ruedas o piezas. El modelo se debe centrar en el bloque programable, ya que este bloque provee la energía necesaria para el movimiento del robot creado. Además, se pueden fijar los sensores que se adjuntan en el kit, para que sean útiles en el desarrollo del robot.

Este juego se compone de 4 partes principales:

- El microprocesador (que se encuentra en el bloque NXT, también llamado ladrillo).
- Los sensores.
- Los servomotores.
- Las piezas de Lego para la construcción.

En este proyecto se ha usado la segunda versión del robot, su antecesora poseía el bloque RCX que contaba con un hardware más reducido.

Las especificaciones hardware de la versión NXT son las siguientes:

- Microcontrolador ARM 7 de 32 bits.
- 256 KB de memoria FLASH.
- 64 KB de memoria RAM.
- Microcontrolador AVR de 8 bits con 4 KB FLASH y 512 Byte RAM
- Bluetooth 2.0
- 4 puertos de entrada para conectores RJ-12
- 3 puertos de salida para conectores RJ-12
- Puerto USB (12 Mbit/s)
- Pantalla LCD 100x64, con PAD de 4 botones.
- Altavoz de 8KHz.

Figura 2. Bloque Lego NXT

Dado que es un juguete para niños a partir de 11 años, Lego creo un software de programación caracterizado por su entorno visual, el cual emula la construcción por bloques, dando la posibilidad a cualquier usuario aprendiz de acostumbrarse rápidamente a la programación de bloque. Este lenguaje, llamado NXT-G, permite las instrucciones secuenciales, instrucciones de ciclos e instrucciones de decisiones, éstas últimas, basadas en los datos reportados por los sensores que se puede añadir al robot. Este sencillo, pero potente software, permite a los niños iniciarse en el mundo de la programación, adquiriendo unos conocimientos básicos.

Debido a la gran cantidad de sensores, a las diferentes posibilidades de montaje, y a la potencia que posee, a pesar de ser un juguete para niños, ha sido muy bien aceptado por toda la comunidad de desarrolladores, aportando nuevos sistemas operativos y firmware para el robot, que reemplazan el original y permiten programarlo en otros lenguajes. Los sistemas Lego más interesantes, desde el punto de vista de este proyecto, son NXC y ROBOTC. Ambos utilizan un lenguaje muy parecido a C (con algunas limitaciones), si bien el primero es de código abierto, y el segundo es de pago. El primero destaca por ser el tercero que más apoyo tiene de la comunidad, gracias a ser software libre; mientras que el segundo destaca por su apoyo comercial.

Por otra parte, en este proyecto se ha usado **leJOS** [5], proyecto de código abierto, que reemplaza el firmware del robot introduciéndole una máquina virtual de Java.

Figura 3. Logo leJOS

La ventaja de leJOS frente a los dos anteriores, es que mientras ellos son una versión limitada de C, este cuenta con todas las funcionalidades de Java. Esta ventaja proporciona a los desarrolladores la opción de contar con orientación a objetos y multihilo.

Su API también ofrece un control sobre los motores de alta precisión, soporte de navegación avanzada, y compatibilidad con receptores GPS y una gran cantidad de sensores. Otro de sus puntos fuertes es que es un proyecto de código abierto, y multiplataforma, y todo esto lo convierte en el más apoyado por la comunidad de desarrolladores, lo que proporciona a nuevos desarrolladores multitud de información y ayuda.

2.2- Android

Android es un sistema operativo basado en Linux, enfocado para ser utilizado en móviles y tablets, aunque ya se puede encontrar en televisiones, netbooks e incluso se está empezando a implementar hasta en relojes. Además se encuentra bajo licencia apache, lo que significa que es software libre y código abierto, por lo tanto cualquier persona puede descargar su código fuente, y modificarlo si desea.

Fue inicialmente diseñado por Android, Inc., empresa que Google respaldo económicamente y posteriormente compró en 2005. Fue presentado en 2007 por la Open Handset Alliance, una un consorcio compuesto por 84 compañías, liderado por Google y que se dedica al desarrollo de estándares abiertos para dispositivos móviles.

Android permite programar en un entorno de trabajo (framework) de Java, aplicaciones sobre una máquina virtual Dalvik (una variación de la máquina de Java con compilación en tiempo de ejecución). Esta máquina virtual está optimizada para requerir poca memoria, y diseñada para ejecutar varías instancias, delegando al núcleo subyacente el soporte de aislamiento entre procesos. Si fuese necesario también se podrían desarrollar aplicaciones sobre el propio núcleo en lenguaje C o C++.

A continuación se muestra una figura que representa la estructura del sistema operativo Android.

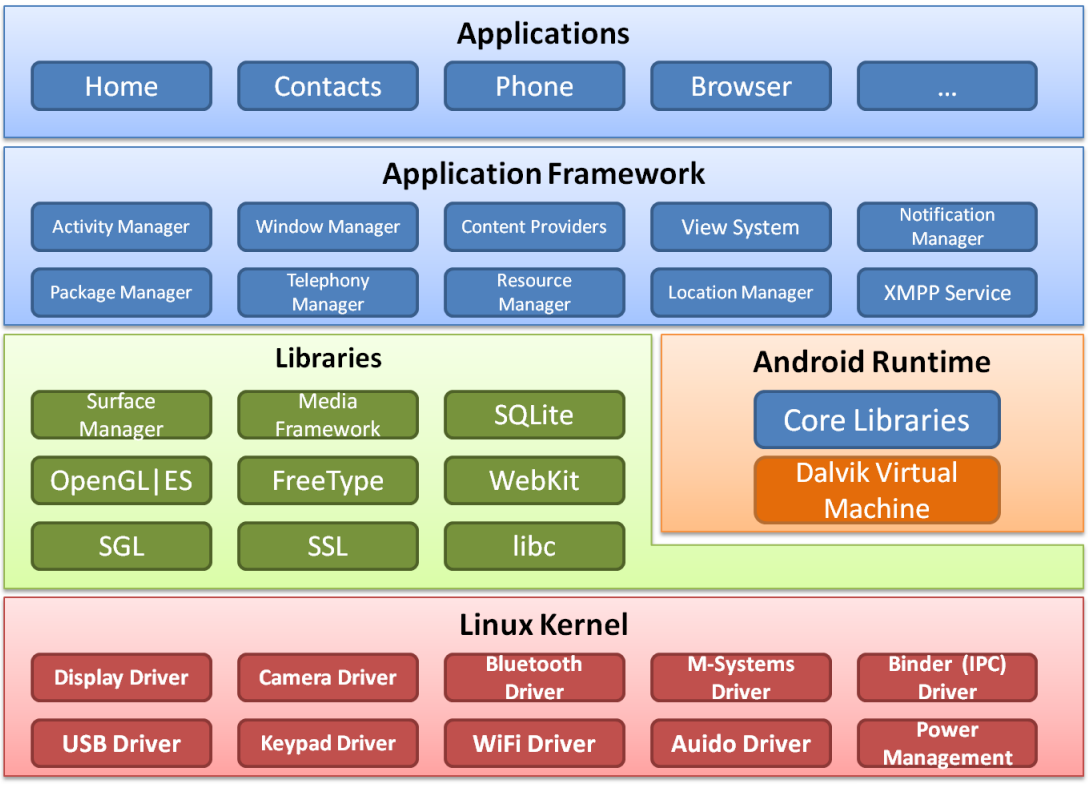

Figura 4. Estructura Android

En esta estructura se encuentran diferentes elementos:

- **Linux Kernel:** el núcleo de Android está formado por el sistema operativo Linux, versión 2.6. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de drivers para

dispositivos. Esta capa del modelo actúa como capa de abstracción entre el hardware y el resto de la pila, por lo tanto, es la única que es dependiente del hardware.

- **Libreras**: estas librerías nativas están escritas en C/C++ y son usadas en varios componentes de Android. Están compiladas en el código nativo del procesador. Muchas de estas librerías utilizan proyectos de código abierto.
- **Android Runtime**: esta capa está compuesta por la máquina virtual Dalvik, de la que ya se ha hablado previamente.
- **Application Framework**: esta capa ha sido diseñada para simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas. Este mismo mecanismo permite a los usuarios reemplazar componentes.
- **Applications**: es la última capa y está compuesta por el conjunto de aplicaciones instaladas.

La clave del éxito de Android es haber creado un sistema multiplataforma, libre y gratuito, en el que para programar en este sistema operativo o incluirlo en un dispositivo no hay que pagar nada, lo cual ha permitido la creación de una gran comunidad de desarrolladores en la que no solo se crean nuevas aplicaciones para él sino que también lo han mejorado mediante nuevas versiones del Kernel o del mismo sistema operativo.

Otra de las claves del éxito de Android fue incluir una tienda de aplicaciones, en la que los desarrolladores pudieran subir sus aplicaciones, para ofrecerlas de forma gratuita o de pago, y facilitar la inclusión de publicidad en ellas, para quien quisiera optar por esta forma de negocio.

Dado a la gran aceptación y rápido crecimiento de Android, conjunto al avance en la tecnología de los Smartphone, se ha creado un gran problema de fragmentación, que dificultad el trabajo de los desarrolladores para conseguir que sus aplicaciones sean compatibles con todas las versiones de Android.

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.2%
2.2	Froyo	8	2.5%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	33.0%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	22.5%
4.1.x	Jelly Bean	16	34.0%
4.2.x		17	6.5%

Data collected during a 14-day period ending on August 1, 2013.
Any versions with less than 0.1% distribution are not shown.

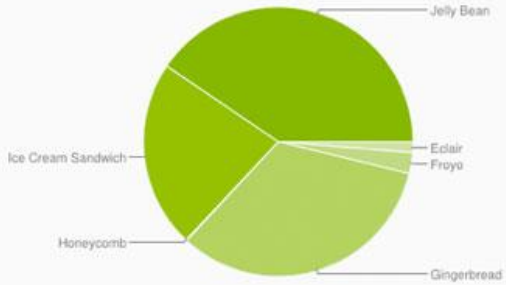

Figura 5. Fragmentación Android

Otro gran inconveniente para los desarrolladores es otra de las claves del éxito de Android, y recae en la gran variedad de dispositivos que hay. Existe una gran variedad de tamaños de pantallas, y una gran variedad de resoluciones para ellas; y cuando el desarrollador quiere escapar un poco de la interfaz típica de Android, basada en estructurar la aplicación en listas y bloques, puede encontrar verdaderos problemas para adaptarla a todos los dispositivos.

Por último para entender cómo funcionan las aplicaciones en Android, es necesario comprender el ciclo de vida en estas.

Una aplicación Android se puede encontrar en cualquiera de estos cuatro estados:

- Activa (Running): la actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.
- Visible (Paused): la actividad se encuentra semi-suspendida, es decir, aún se está
- ejecutando y es visible, pero no es la tarea principal. Se debe guardar la información en este estado para prevenir una posible pérdida de datos en caso de que el sistema decida prescindir de ella para liberar memoria.
- Parada (Stopped): cuando la actividad no es visible, se recomienda guardar el estado de la interfaz de usuario, preferencias, etc.
- Destruida (Destroyed): cuando la actividad termina al invocarse el método `finish()`, o es matada por el sistema Android, sale de la pila de actividades.

A continuación se muestra el diagrama de las posibles transiciones de estado, los cuadrados representan las llamadas a los métodos, y los óvalos el estado de la aplicación en ese momento.

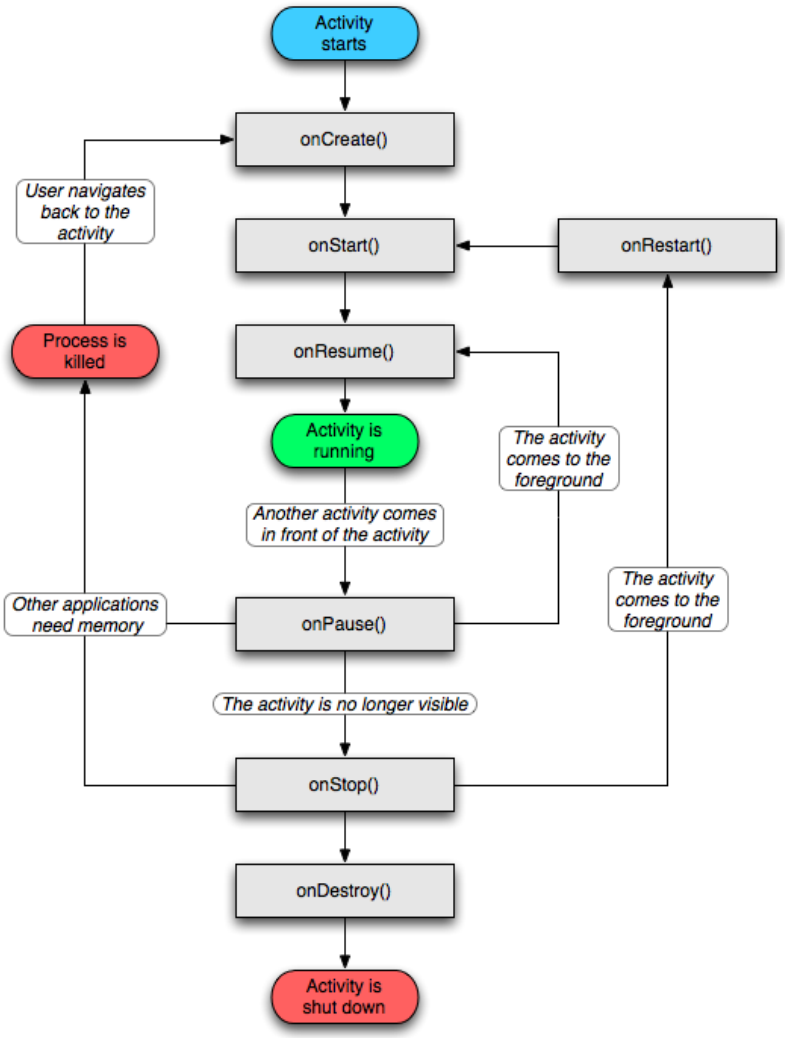

Figura 6. Ciclo de vida de una Activity

Como se puede observar existen siete métodos diferentes, que el programador puede implementar si desea, si no lo hace se llamará al método de la superclase.

A continuación se realizara una breve descripción del funcionamiento de estos siete métodos.

- `onCreate(Bundle)`: Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de instancia (en una instancia de la clase `Bundle`), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.

- `onStart()`: En este método la actividad está a punto de ser mostrada al usuario.

- `onResume()`: Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.

- `onPause()`: Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra aplicación es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.

- `onStop()`: La actividad ya no va a ser visible para el usuario. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

- `onRestart()`: Indica que la actividad va a volver a ser representada después de haber pasado por `onStop()`.

- `onDestroy()`: Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón <volver> o cuando se llama al método `finish()`. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

2.3- Bluetooth

Figura 7. Logo Bluetooth

Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz. Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar los cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

El nombre viene de Harald Bluetooth, un Vikingo y rey de Dinamarca a de los años 940 a 981, fue reconocido por su capacidad de ayudar a la gente a comunicarse. Durante su reinado unió Dinamarca y Noruega.

Bluetooth ofrece posibilidades de uso casi ilimitadas, entre ellas:

- Conexión sin cables vía OBEX [6].
- Transferencia de fichas de contactos, citas y recordatorios entre dispositivos vía OBEX.
- Reemplazo de la tradicional comunicación por cable entre equipos GPS y equipamiento médico.
- Controles remotos (tradicionalmente dominado por el infrarrojo).
- Enviar pequeñas publicidades desde anunciantes a dispositivos con Bluetooth. Un negocio podría enviar publicidad a teléfonos móviles cuyo Bluetooth (los que lo posean) estuviera activado al pasar cerca.
- Las consolas Sony PlayStation 3, Microsoft Xbox360 y Wii incorporan Bluetooth, lo que les permite utilizar mandos inalámbricos, aunque los mandos originales de la Wii funcionan mezclando la tecnología de infrarrojos y Bluetooth.
- Enlace inalámbrico entre sistemas de audio y los altavoces (o altoparlantes) correspondientes.

3. Descripción de la aplicación

Esta aplicación permite controlar un robot Lego Mindstorms desde un Smartphone con Android, aportando un módulo de inteligencia mediante la utilización de sensores; dicho modulo se podrá configurar desde la aplicación situándolo en el Smartphone o en el robot. Aparte, posee alguna funcionalidad extra como la reproducción de un archivo de sonido, o el programa seguidor de línea.

Para el desarrollo de la aplicación es necesario configurar dos entornos de programación; uno para Android y otro para el lenguaje usado en el robot Lego "leJOS".

3.1- Tipos de paquetes

Para entender el funcionamiento de la aplicación, es necesario conocer el intercambio de paquetes que se produce, entre el robot y el dispositivo Android. Todos los paquetes que se envían son una cadena de 3 bytes de longitud. Se ha elegido byte, ya que el flujo de salida es el único tipo de datos que permite enviarlo en forma de array; y dado que los valores con los que hay que trabajar son pequeños, se encaja perfectamente a las necesidades del proyecto.

Primer byte	Segundo byte	Tercer byte
[0] Cambio de dirección	[0] Stop	
	[8] Adelante	
	[4] Izquierda	
	[6] Derecha	
	[5] Atrás	
	[1] Viajar distancia	[X] Distancia
	[3] Rotar ángulo	[X] Ángulo
[1] Sensor	[0] Choque	
	[1] Proximidad EOPD	
	[2] Proximidad ultrasonido	
	[3] Sonido	
	[4] Brújula	
	[5] Color	
	[6] Giroscopio	
[2] Configuración	[0] Inteligencia	[0] Robot
		[1] Móvil-Automática
		[2] Móvil-Usuario
	[1] Desconexión	
[3] Extras	[0] Leyes de la Robótica	
	[1] Seguidor de línea	[0] Preparar
		[1] Sensor colocado
		[2] Sensor sobre fondo
		[3] Sensor sobre línea
		[4] Detener

En los bytes marcados con una X irá el valor que el programador haya considerado necesario para el correcto funcionamiento de la aplicación.

3.2- Robot Lego

3.2.1- Reemplazar firmware del robot

Para que el robot lego acepte el programa, se debe reemplazar su firmware, en este caso por uno llamado leJOS NXJ que permitirá programar el robot en java.
Para ello lo primero que hay que hacer será instalar el Java Development Kit (JDK), si no lo está.
Se puede desde esta dirección.
http://www.oracle.com/technetwork/java/javase/downloads/index.html

El siguiente paso será instalar los drivers del robot, para que el ordenador lo reconozca. Se debe instalar el Fantom Driver que es la versión más actualizada.
http://mindstorms.lego.com/en-us/support/files/Driver.aspx

Y por último queda instalar este software que permitirá realizar varias tareas con el robot, desde un navegador para introducir o eliminar archivos, hasta aplicaciones para probar sus sensores y el estado de los motores. La que interesa en este punto se llama NXJ Flash y es la que permitirá flashear el robot para cambiarle el firmware.
http://sourceforge.net/projects/lejos/files/lejos-NXJ/

Una vez abierto el programa NXJ Flash se muestra esta ventana.

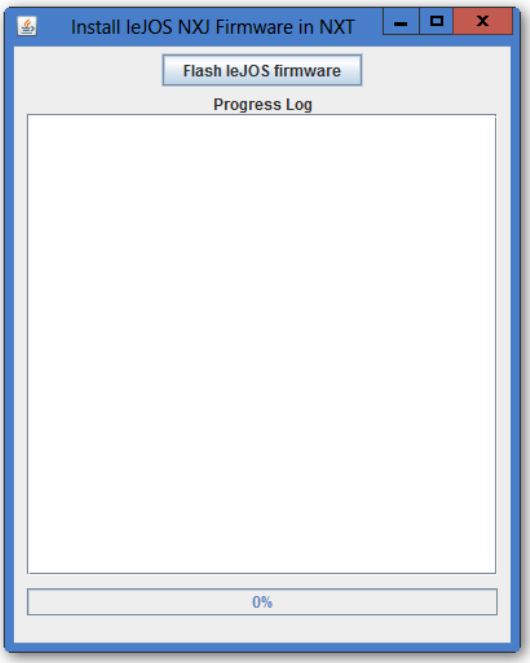

Figura 8. Programa NXJ Flash

Lo primero que se debe hacer antes de nada será encender y conectar el robot por USB al ordenador. Como los drivers han sido instalados, lo debe reconocer sin problemas.

Una vez conectado se pincha sobre el botón “Flash leJOS firmware”, y saldrá esta ventana.

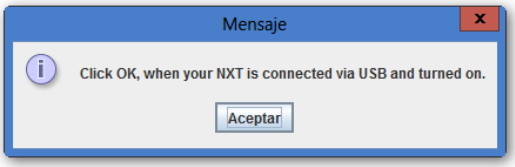

Figura 9. Mensaje de aviso en el programa NXJ Flash

Se pulsa el botón “Aceptar”, ya que se ha conectado y encendido el robot previamente.
Ahora saldrá el siguiente mensaje.

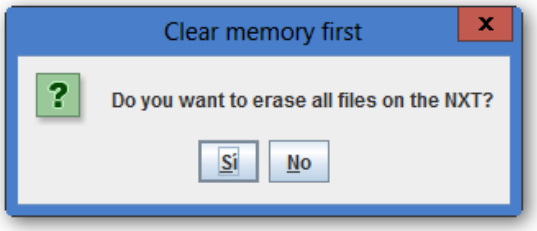

Figura 10. Mensaje borrar memoria en el programa NXJ Flash

Este mensaje pregunta si se quiere borrar todos ficheros del robot. Hay que pinchar en “Sí” y empezara el proceso.
Cuando salte este mensaje, indicará que todo ha ido bien y el proceso ha acabado.

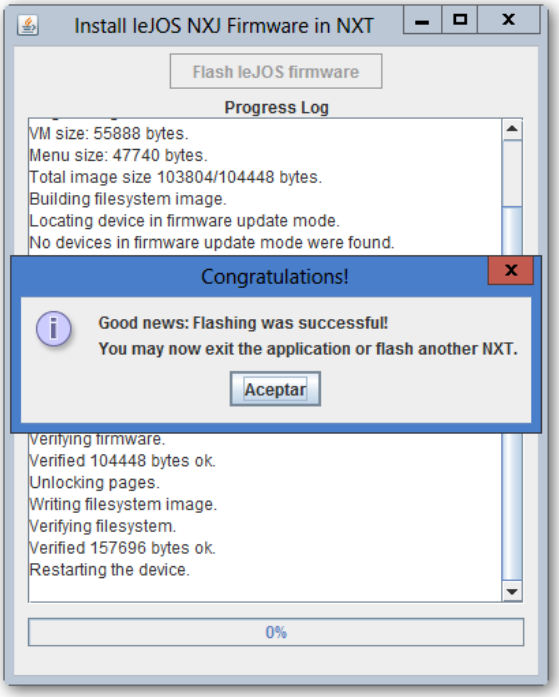

Figura 11. Mensaje flasheo realizado con éxito

3.2.2- Configuración del entorno de desarrollo

Una vez reemplazado el firmware, el siguiente paso es configurar el entorno de desarrollo desde el cual se desarrollaran programas y cargaran en el robot.
Para ello lo primero que se hará será descargar el IDE eclipse; entre todas las opciones hay que seleccionar la versión estándar.
http://www.eclipse.org/downloads/

Una vez descargado, solo es necesario descomprimir el fichero y ya se puede ejecutar.
Cuando eclipse este abierto se procederá a instalar el plugin.

Para ello se pulsa sobre el botón "Help" de la barra de navegación, y en el menú desplegable se selecciona "Install New Software…".

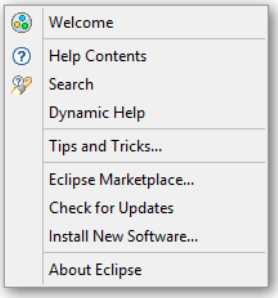

Figura 12. Menú Help en eclipse

En la ventana que se abrirá se introduce en el campo "Work with:" la siguiente dirección:
http://www.lejos.org/tools/eclipse/plugin/nxj/
Y se pulsa sobre el botón "Add…".

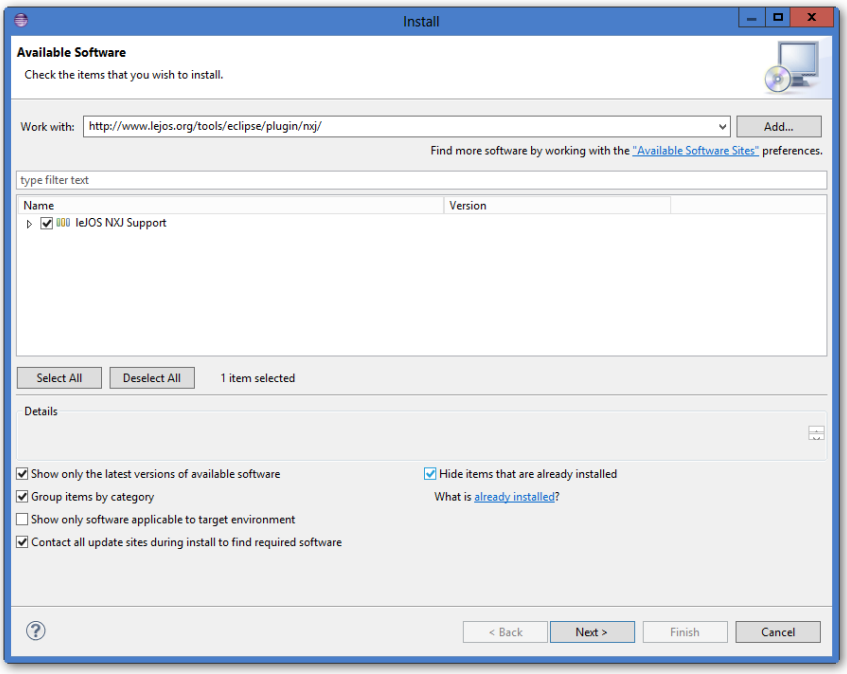

Figura 13. Instalación del plugin de leJOS

Se selecciona la opción "LeJOS NXJ Support" y se pincha sobre el botón "Next".
Se aceptan los términos y se le sigue dando a siguiente, hasta que empiece a instalarse. A mitad de la instalación saldrá este mensaje de advertencia.

Figura 14. Aviso en instalación del plugins de leJOS

Hay que pinchar sobre el botón “OK”.
Por último saldrá un mensaje preguntando si se quiere reiniciar eclipse. Se pulsa sobre “Yes”, y una vez reiniciado el entorno estará configurado para realizar lo programas que se deseen.

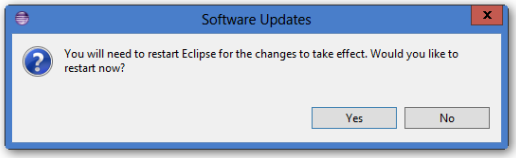

Figura 15. Petición reiniciar eclipse

3.2.3- Proyecto leJOS

Un proyecto leJOS es igual que un proyecto Java, con la única diferencia de que tiene la librería “LeJOS NXT Runtime”, que contiene las clases de la API de leJOS que permiten controlar los sensores, motores, y todos los aspectos del robot.

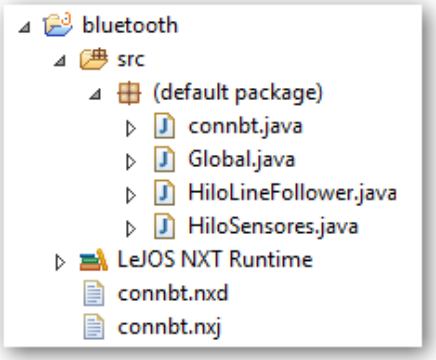

Figura 16. Estructura proyecto leJOS

3.2.4- Ejecución de una aplicación en el robot

Para ejecutar un programa en el robot hay que conectar el robot por USB al ordenador, o sincronizarlo por bluetooth, y tenerlo encendido. Si se sincroniza por bluetooth la contraseña por defecto es “1234”.
Una vez conectado el robot con el ordenador, solo se tiene que ir a la clase que contiene el main del programa y pincharla con el botón derecho, se abrirá un menú desplegable en el que hay que buscar la opción “Run As” y se desplegará, para después pinchar sobre “LeJOS NXT Program”.
Si está conectados por USB se ejecutará casi instantáneamente, si por el contrario está por bluetooth tardará unos 10 segundos.

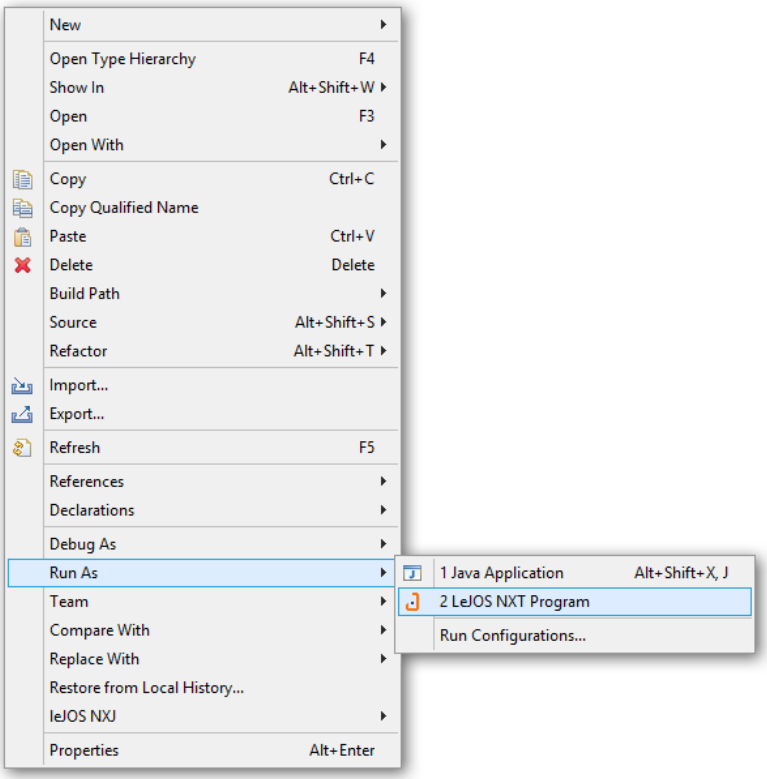

Figura 17. Menú desplegable de la opción Run As

3.2.5- Cargar archivos de audio en el robot

El robot lego permite reproducir archivos de audio, en formato wma, a una calidad de 8 KHz, gracias a que posee un altavoz interno y un microcontrolador de 8 bits dedicado a este fin. Se pueden reproducir desde el menú del propio ladrillo del robot o desde un programa que se desarrolle, pero en ambos casos se tiene que introducir el archivo de audio manualmente.
Para ello el usuario debe abrir el programa "NXJ Browse".

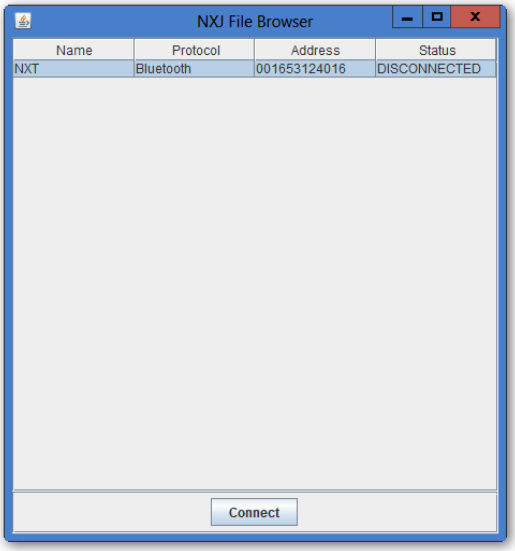

Figura 18. Programa NXJ File Browser

Se puede conectar por bluetooth o USB. Una vez conectado, aparecerán los archivos que hay dentro de la memoria del robot, y se pueden eliminar, descargar en el ordenador, seleccionar un archivo que se quiera que se ejecute por defecto al arrancar el robot, cambiar a un archivo de nombre, ejecutar los programas ya cargados o subir un archivo nuevo, que es la opción que es de interés en este caso.

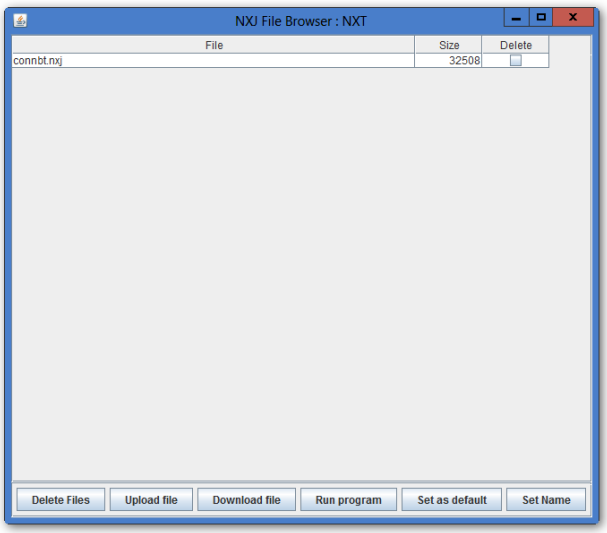

Figura 19. Programa NXJ File Browser

Una vez subido el archivo ya se puede cerrar el programa.

3.2.6- Diferentes sensores del robot

Se puede encontrar una gran variedad de sensores para el robot. A continuación se hablará sobre algunos de ellos.

3.2.6.1- Sensor de choque

Este sensor permite detectar si se ha sufrido una colisión.
Al chocar contra un objeto la pequeña cabeza naranja se contrae, cerrando un circuito interno, y así se genera una señal que permitirá al robot saber que se ha producido un choque.

Figura 20. Sensor de choque

3.2.6.2- Sensor de ultrasonidos

Este sensor permite detectar objetos antes de colisionar con ellos mediante ultrasonidos.
En las pruebas que se han realizado con él no ha funcionado correctamente, ya que detectaba los objetos demasiado tarde, y en la mayoría de veces al robot no le daba tiempo a frenar antes de colisionar con ellos. Por esta razón no se ha utilizado para la aplicación final.

Figura 21. Sensor de ultrasonidos

3.2.6.3- Sensor de color

Este sensor permite detectar cualquier color.
También se puede configurar para capturar la cantidad de luz reflejada por un color, esto puede ser de utilidad; más adelante se explicará cómo se ha usado para hacer un sigue líneas.

Figura 22. Sensor de color

3.2.6.4- Sensor de proximidad electro-óptico (EOPD)

Este sensor permite detectar objetos a una distancia de hasta 20cm mediante un pulso de luz. El sensor lee la diferencia antes de que el pulso se emita, y mientras el pulso se está emitiendo, y restando la diferencia puede saber si se encuentra ante un objeto.

Figura 23. Sensor de proximidad EOPD

3.2.6.5- Sensor brújula (Compass Sensor)

El sensor brújula permite calcular la dirección a la que se enfrenta el robot. Para conseguirlo en su interior tiene una brújula digital magnética, que mide el campo magnético de la tierra y calcula el rumbo.
En las pruebas que se han podido realizar con él, en pequeños programas, ha funcionado bastante bien.

Figura 24. Sensor brújula

3.2.6.6- Sensor de sonido

Este sensor puede detectar presiones de sonido, de hasta 90 decibelios.

Figura 25. Sensor de sonido

3.2.6.7- Sensor de giro

Este sensor mide en grados por segundo la rotación, aparte de la dirección en la que se ha producido esa rotación.

Figura 26. Sensor de giro

3.3- Android

3.3.1- Configuración del entorno de desarrollo

Para obtener el entorno de desarrollo de Android lo mejor es descargar el paquete ADT de su página oficial, que proporciona el eclipse con el puglin configurado y el SDK de Android.
http://developer.android.com/sdk/index.html

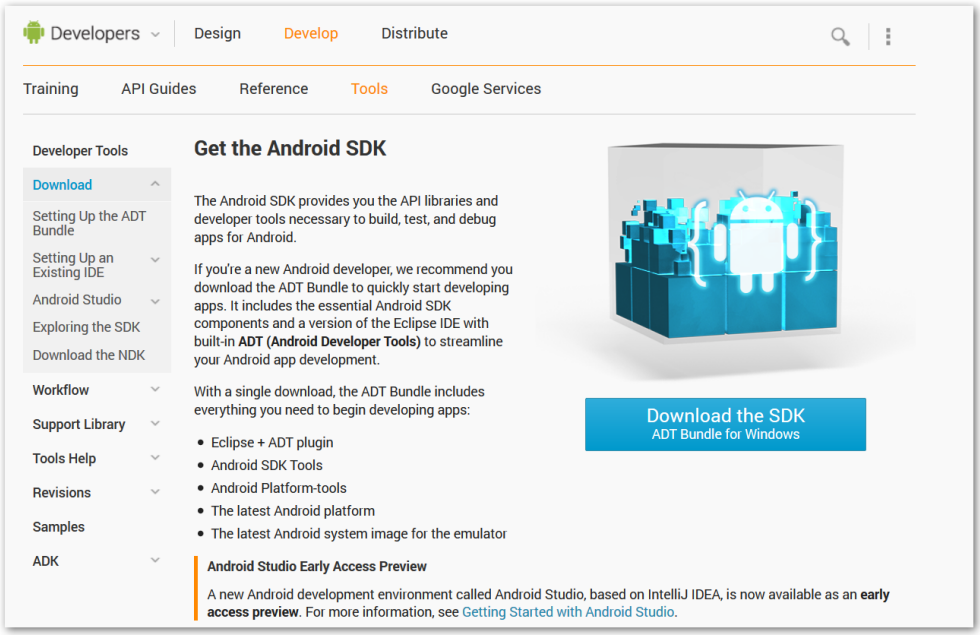

Figura 27. Pagina descarga paquete ADT de Android

Una vez descargado solo hay que descomprimir el archivo y ya estará listo para utilizarlo.
Es importante comprobar antes el SDK manager, pues normalmente solo viene instalada la última versión de la API y se puede considerar necesario instalar cualquier otra versión.

3.3.2- Estructura de un proyecto Android

Al generar un nuevo proyecto Android con el entorno de desarrollo eclipse, este generará automáticamente la distribución de carpetas que contendrá la aplicación, y que será común a todos los proyectos Android.

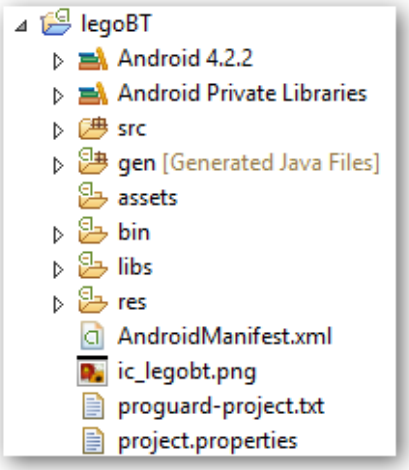

Figura 28. Estructura de un proyecto Android

A continuación se describirá el significado de cada carpeta por separado:

- Carpeta src

Recoge la totalidad del código fuente (Java) de la aplicación.

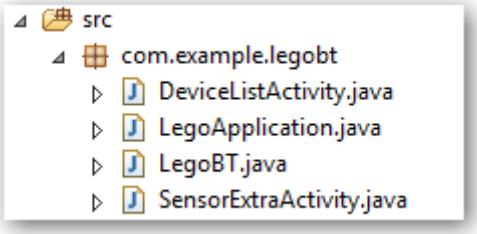

Figura 29. Contenido carpeta src

- Carpeta res

Contiene los recursos necesarios de una aplicación Android.

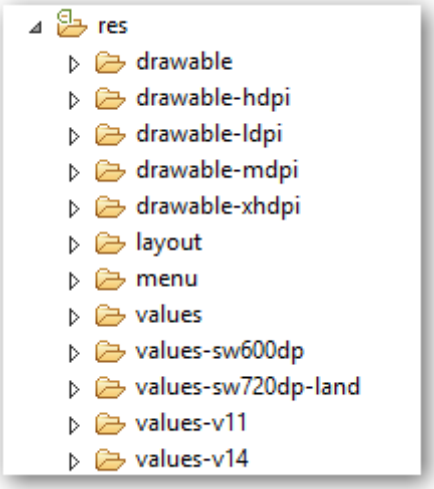

Figura 30. Contenido carpeta res

- La carpeta drawable contiene las imágenes necesarias para el proyecto, y se divide en hdpi, ldpi, mdpi y xhdpi según la resolución.
- La carpeta layout contiene los archivos que contienen la interfaz gráfica, siempre en XML.
- La carpeta values guarda los datos y tipos que utiliza la aplicación, tales como colores, cadenas de texto, estilos, dimensiones…
- La carpeta menú contiene la estructura de los menús de las diferentes actividades.

- Carpeta gen

Ésta carpeta guarda un conjunto de archivos (de código Java) creados automáticamente cuando se compila el proyecto, para poder dirigir los recursos de la aplicación. El archivo R ajusta automáticamente todas las referencias a archivos y valores de la aplicación (guardados en la carpeta res).

- Carpeta assets

Guarda el resto de archivos necesarios para el correcto funcionamiento de la aplicación, como los archivos de datos o de configuración. La principal diferencia entre los recursos que almacena ésta carpeta y los que guarda la carpeta “res”, es que los recursos de ésta última generan un identificador por recurso, identificador que se encargará de gestionar el fichero R y sólo se podrá acceder a ellos a través de determinados métodos de acceso, mientras que los

recursos almacenados en la carpeta “assets” no generan identificador alguno y se accederá a ellos a través de su ruta, como se hace con cualquier otro fichero.

- Archivo AndroidManifest.xml
Éste archivo es uno de los más importantes de cualquier aplicación Android. Se genera automáticamente al crear el proyecto, y en él se encuentra definida la configuración del proyecto en XML (Activities, Intents, los permisos de la aplicación, bibliotecas, etc.).

3.3.3- Compilación y ejecución

Para ejecutar la aplicación lo primero que hay que hacer será crear un nuevo perfil de ejecución. Para ello hay que desplegar el menú “Run” y pinchar sobre la opción “Run Configurations”, y aparecerá la siguiente ventana.
Sobre la categoría “Android Application” se pincha con el botón derecho y se elige la opción “New”.

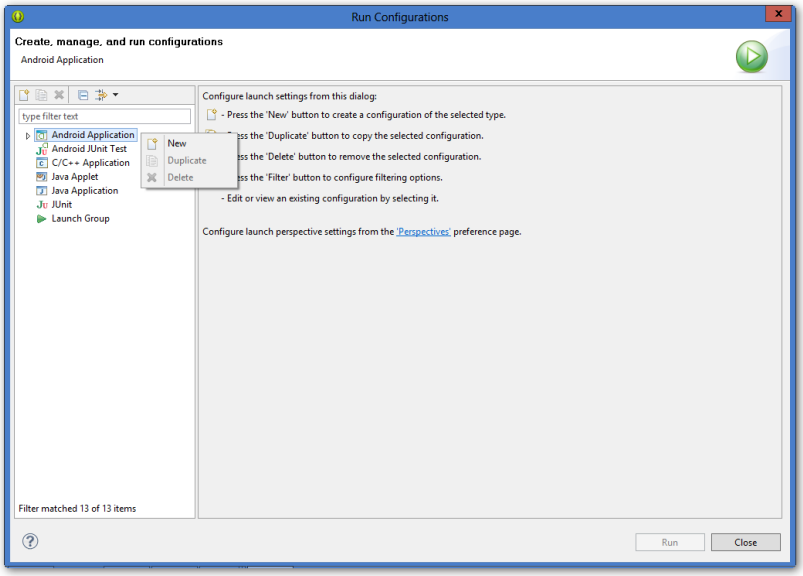

Figura 31. Ventana Run Configurations

En la siguiente pantalla en el campo “Name:” se introduce el nombre del perfil de ejecución y en el campo “Project” se pulsa sobre el botón “Browse…” para seleccionar el proyecto que se quiere ejecutar.
Por último se pulsa sobre el botón “Apply” y después “Run” para ejecutarlo.

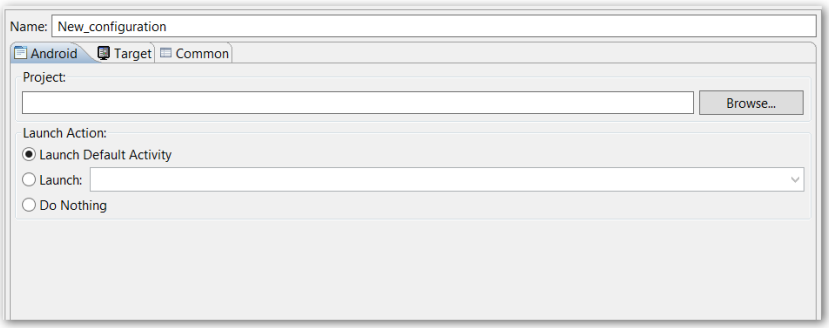

Figura 32. Ventana Run Configurations

No hay que olvidar, que para que funcione hay que tener el móvil conectado en modo debug.

3.4- Manual de uso

Lo primero que se debe hacer será ejecutar el programa creado para el robot.
Para el correcto funcionamiento se deben tener los sensores de choque conectados en el puerto 1 y 2, y el sensor de proximidad EOPD conectado en el puerto 4.

Figura 33. Robot Lego con sensores de choque y proximidad

Después se procederá a ejecutar la aplicación Android, si el bluetooth está desactivado mostrará una ventana emergente preguntando si se desea activar.
Si se pulsa la opción “Denegar” la aplicación se cerrará automáticamente, ya que es necesario tener el bluetooth encendido para poder usarla. Por el contrario si el usuario pulsa la opción “Permitir” se continuará con la ejecución normal del programa.

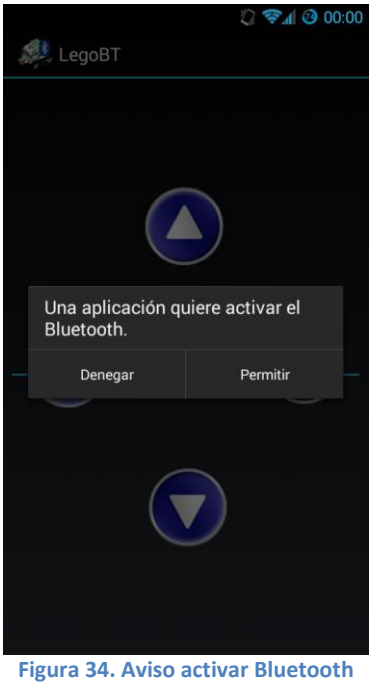

Figura 34. Aviso activar Bluetooth

Lo siguiente será conectar la aplicación con el robot mediante bluetooth, para ello se pulsa la tecla menú del dispositivo, y en el menú desplegable se seleccionará la opción “Conectar con un dispositivo”.

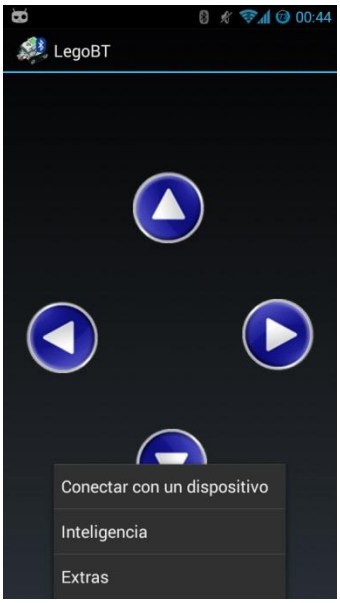

Figura 35. Menú de la aplicación

La nueva ventana que se abre muestra los dispositivos emparejados, y también permite buscar nuevos dispositivos.
Si es la primera vez que se utiliza, habrá que buscar al robot pulsando sobre el botón “Buscar dispositivos”, y emparejarlo pulsando sobre él; pedirá la contraseña (1234 por defecto) y se conectará.

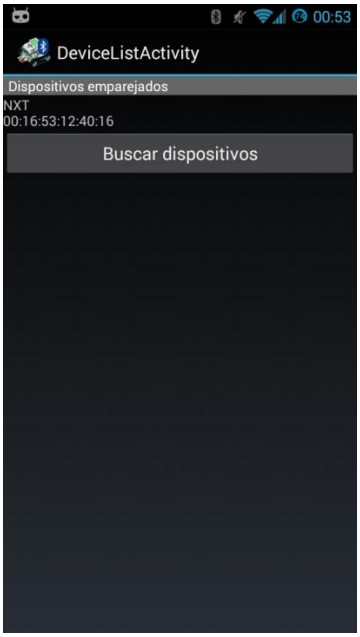

Figura 36. Ventana buscar dispositivos

Una vez conectado, la aplicación volverá a la pantalla anterior, en la que se podrá manejar el robot pulsando sobre las flechas de dirección.

Mediante los sensores de proximidad y choque, el robot evitará obstáculos.
El usuario puede elegir si quiere que el módulo de inteligencia lo posea el robot, el móvil de manera automática, o el usuario a través del móvil; esta última opción bloquea la tecla de dirección adelante, y no dejará usarla hasta que el usuario no corrija la dirección del robot.
Para elegir estas opciones de inteligencia se pulsara sobre la tecla menú del Smartphone, y en el menú desplegable sobre "Inteligencia", este abrirá una ventana emergente en la que se seleccionará la opción a convenir por el usuario.

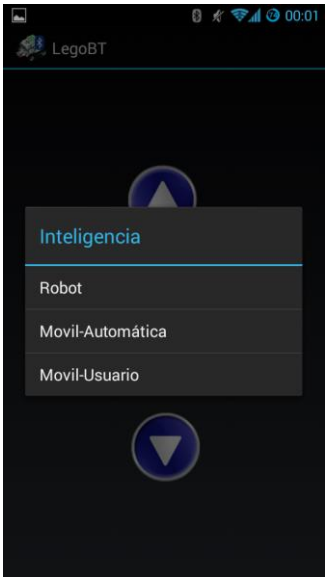

Figura 37. Menú inteligencia

Por último en ese mismo menú desplegable, se localiza el apartado de extras, en el que el usuario puede ejecutar funciones adicionales.
La primera, "Leyes de la Robótica" es una prueba del altavoz interno del robot, en la que se ejecutará un archivo de sonido introducido previamente en la memoria del robot, que en este caso dice las leyes de la robótica.
La segunda opción, "Seguidor de Línea" consiste en un programa que sigue una línea de color negro, utilizando el sensor de color del que se dispone. Para ello habrá que colocarlo mirando hacia el suelo a una distancia de unos 5mm de él.

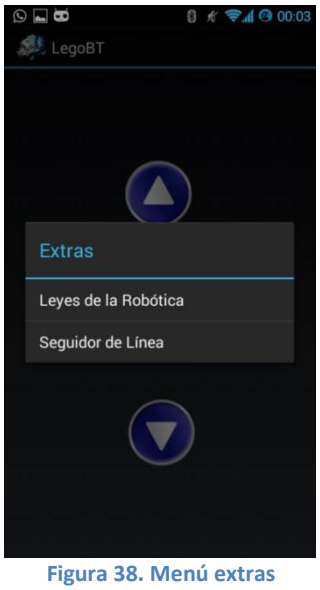

Figura 38. Menú extras

Si se selecciona la opción de seguidor de línea, mostrará la siguiente ventana, en la que se deben obedecer las indicaciones.

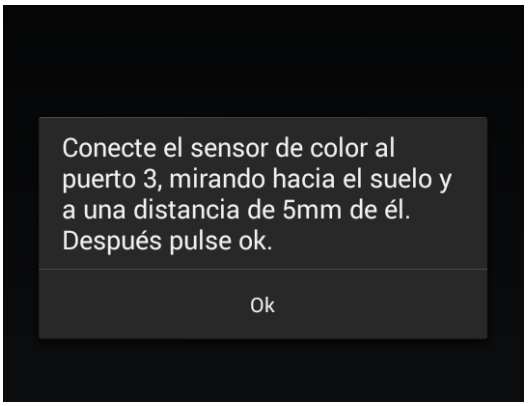

Figura 39. Aviso conectar sensor de color

El robot quedaría así.

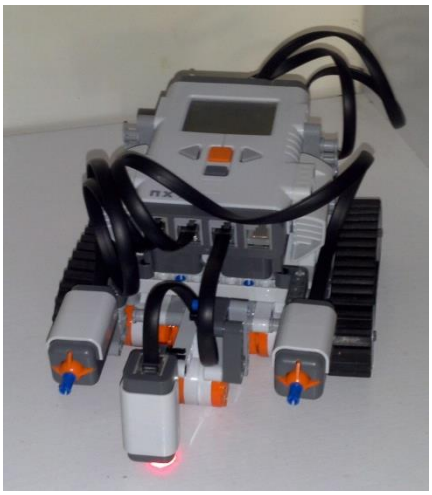

Figura 40. Robot con sensor de color

Una vez colocado el sensor, se pulsará “Ok” y mostrará el siguiente aviso.

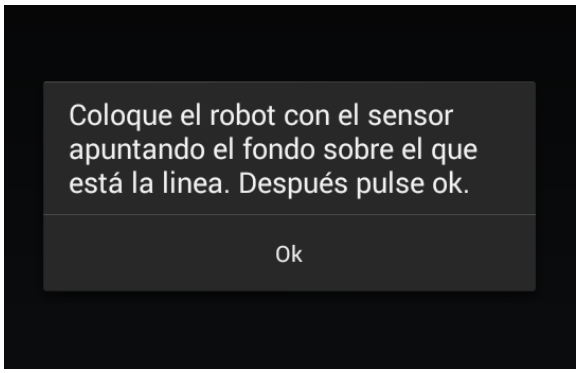

Figura 41. Aviso colocar robot sobre fondo de la línea

Haciéndole caso se debería colocar el robot como en la siguiente ilustración, antes de pulsar "Ok".

Figura 42. Robot sobre el fondo de la línea

Por último una vez pulsado "Ok", la aplicación mostrará el siguiente aviso.

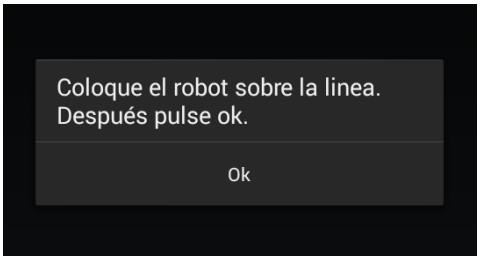

Figura 43. Aviso colocar robot sobre la línea

Y otra vez haciéndole caso, el robot se debería colocar así.

Figura 44. Robot sobre la línea

Una vez colocado sobre la línea, al pulsa el botón “Ok”, el robot comenzará a moverse siguiendo el borde entre la línea negra y el fondo blanco.
Mientras la aplicación mostrará esta pantalla, en la que si el usuario pulsa la tecla atrás del Smartphone el robot se detendrá.

Figura 45. Ventana Seguidor de Línea

En ese momento el usuario deberá volver a conectar los sensores de proximidad y choque si los ha desconectado, para poder continuar con la ejecución normal de la aplicación, controlando el robot con las flechas de dirección y evitando obstáculos.

4- Descripción de la implementación

4.1- Descripción del código del robot Lego

Como ya se ha dicho anteriormente, el programa ejecutado en el robot, es un programa en java. Este programa se encarga de ser el puente de comunicación entre los sensores y el dispositivo Android, y dependiendo de la configuración seleccionada, tomando algunas decisiones también por su cuenta.

Se dispone de una completa API para ver todas las clases con las que se puede programar y dar forma al proyecto.
http://www.lejos.org/nxt/nxj/api/index.html

4.1.1- Clase Connbt

Esta clase es el hilo principal del programa en el robot, y está orientada al flujo de recepción de mensajes.

En ella se encuentra declarado un objeto del tipo `DifferentialPilot`, que permite controlar el robot de una forma más completa, al contener un control de navegación más avanzado, en el que se puede indicar al robot que:

- Avance una distancia en concreto
- Gire cierto ángulo.
- Avance o gire indefinidamente hasta que se le diga que pare.
- Avance un cierto ángulo indicándole el radio.

Las posibilidades que ofrece esta clase son muchas. Para que el robot sea capaz de hacer todo esto en sus argumentos se le debe pasar el radio de las ruedas, la distancia entre ejes, el puerto al que está conectado el motor izquierdo, y el puerto al que está conectado el motor derecho.

```
pilot=new DifferentialPilot(1.7f, 9.5f, Motor.C, Motor.A);
```

Figura 46. Declaración de objeto tipo DifferencialPilot

En la API pone que hay que pasarle el diámetro de la rueda, pero durante el desarrollo del proyecto se ha podido comprobar que para el correcto funcionamiento necesita el radio, si no todos los cálculos los realizará de manera incorrecta.

En esta clase también se puede encontrar declarada la conexión bluetooth. En este proyecto el robot hace de servidor y es el dispositivo Android el que se conecta a él, por tanto se tiene que esperar la solicitud de una nueva conexión.

Esto es algo muy sencillo pero se debe estar atento a los parámetros introducidos.

```
btc = Bluetooth.waitForConnection(0, NXTConnection.RAW);
```

Figura 47. Declaración del Bluetooth

En el primer parámetro se le indica el tiempo que debe esperar a recibir una solicitud de conexión. El valor "0" indica que el tiempo es infinito.

El segundo parámetro es el realmente importante, pues se indica el modo de conexión que se requiere. Si se llama a la función sin ningún parámetro, esta los preestablecerá por defecto, y la conexión dará error, ya que el modo por defecto no permite la conexión con dispositivos Android.

Los tres modos que se pueden seleccionar son:

- `NXTConnection.RAW`: este modo es el que se utiliza para conexiones que no sean con otros dispositivos Lego NXT, como Smartphone. Y es el que se ha seleccionado en este proyecto.
- `NXTConnection.LCP`: este modo se utiliza para conexiones a través del protocolo LCP. Este protocolo creado por Lego, se utiliza para mandar comandos al robot, sin la necesidad de estar ejecutando un programa en él. Estos comandos pueden ser comandos directos, que permiten manejar el robot, o comandos del sistema. A pesar de ser un protocolo muy útil, no se ha usado en este proyecto, porque no permite comunicación bidireccional entre dispositivos.
- `NXTConnection.PACKET`: este es el modo predeterminado, y es el mejor si lo que se quiere es conectarse a otro dispositivo Lego NXT.

También en esta clase se encuentra una muestra del manejo del LCD del robot, escribiendo mensaje sobre él o borrándolo, mediante dos funciones muy sencillas como:

```
LCD.drawString("Esperando...", 2, 1);
```

Figura 48. Escribiendo en el LCD

Esta función muestra un mensaje por el LCD, pasándole por sus argumentos la cadena a mostrar, y las coordenadas (x, y) del punto donde se quiere mostrar el mensaje. El punto (0, 0) es la esquina superior izquierda de la pantalla.

```
LCD.clear();
```

Figura 49. Borrando el LCD

Esta función limpia todo el contenido que este mostrando el LCD en ese momento.

Justo después de que se establezca la conexión, se arranca un hilo (instancia de la clase `HiloSensores` que se explica más adelante) que lee los sensores de choque y proximidad del robot y toma decisiones de control en función de las lecturas. A este hilo se le pasa:

- El objeto del tipo `DifferentialPilot`, para que sea capaz de tomar decisiones y controlar el robot al encontrar un objeto, sí la opción de inteligencia esta activa en el robot.
- El flujo de salida de datos, para poder enviarle al dispositivo Android el aviso del obstáculo encontrado.

```
hsensor = new HiloSensores(out, pilot);
```

Figura 50. Declarando el hilo que controla los sensores

Como ya se dijo al principio, esta clase principal va orientada al flujo de recepción de paquetes y gestión de ellos; esto se consigue mediante un bucle `while` del que solo se puede salir si el dispositivo envía el mensaje de cerrar conexión y que contiene dentro una serie de condiciones para analizar los paquetes recibidos.

Este bucle está compuesto principalmente por un `switch` que analiza los paquetes en los que su primer byte es el "0", el "2", o el "3". A los de tipo "1" no se les presta atención, ya que son avisos de los sensores, y solo se mandan desde el robot hacia el móvil.

En los mensajes de tipo "0" se puede ver que se tienen en cuenta todas las direcciones indicadas en la definición de los paquetes, y que si se recibe una orden de rotar cierto ángulo o avanzar una distancia concreta, se tomará para ello el valor introducido en el tercer byte.

```
switch(imput[1]){
case 8:
    pilot.forward();
    break;
case 4:
    pilot.rotateLeft();
    break;
case 6:
    pilot.rotateRight();
    break;
case 5:
    pilot.backward();
    break;
case 0:
    pilot.stop();
    break;
case 1:
    pilot.travel(imput[2]);
    break;
case 3:
    pilot.rotate(imput[2]);
    break;
}
```

Figura 51. Switch que analiza los paquetes de movimiento

En los mensajes de tipo "2" solo hay dos opciones, recibir un cambio de inteligencia, en el que se introducirá el valor recibido en la variable global de inteligencia (accesible por todas las clases del programa), pues está asociado el tipo de inteligencia a ese valor.

La otra opción es recibir una petición de cierre de conexión, en este caso se cerraran los flujos de entrada y salida, y la conexión bluetooth, y se detendrá el hilo que está en ejecución así como el objeto que controla el navegador.

```
if(imput[1]==0){
    Global.inteligencia=imput[2];
}else if(imput[1]==1){
    hsensor.detener();
    pilot.stop();
    dis.close();
    btc.close();
    btc=null;
    LCD.clear();
    LCD.drawString("Cerrando conexion", 0, 1);
```

Figura 52. Condición que analiza los paquetes de configuración

Para los mensajes de tipo "3", solo hay dos tipos de mensajes extras, pero se ha dejado el código preparado con un `switch` para que las posibles ampliaciones del programa se puedan desarrollar de forma más cómoda.

El primer tipo de mensaje extra, indica que se quiere reproducir un archivo interno del robot en el que se encuentran grabadas las leyes de la robótica. Reproducir un archivo de audio es muy simple, solo hay que definir un objeto de tipo `File`, en el que se le pasa el nombre del archivo. Después solo hay que utilizar el método que proporciona la API para reproducir archivos de audio, al que se le pasa como parámetros el objeto `File` y el volumen al que se quiere reproducir el archivo. La llamada a este método es no bloqueante, por lo que se puede seguir utilizando el robot de manera normal mientras se reproduce.

```
case 0:
    Sound.playSample(soundFile, 99);
    break;
```

Figura 53. Muestra de cómo reproducir un sonido

El segundo tipo de mensaje extra lanza el programa "Seguidor de línea", que ayudado del sensor de color el programa es capaz de seguir una línea. Se pueden recibir 5 mensajes de este tipo, ya que son las fases que se necesitan para preparar el lanzamiento del hilo, y el posterior cierre.

- El primer mensaje que se recibe, es un aviso de que se va a iniciar ese programa, y pausa el hilo que controla los sensores que localizan obstáculos para que no interfieran.
- El segundo mensaje avisa de que el sensor ha sido colocado en el puerto correcto y en su posición, por lo tanto declara el sensor, y lo activa en modo `setFloodlight` que actúa capturando la luz de ambiente reflejada por los diferentes objetos.
- El tercer mensaje avisa de que el robot ha sido colocado con el sensor apuntando al suelo sobre el que esta la línea a seguir, y guarda el valor de la luz reflejada por este.
- El cuarto mensaje avisa que el robot ha sido colocado sobre la línea a seguir, captura la luz reflejada por esta y arranca el hilo pasándole ambas variables.
- El quinto mensaje pide la detención del programa, cierra el hilo del sigue líneas y vuelve a reanudar el hilo que controlaba los sensores de proximidad y choque.

```
case 1:
    if(imput[2]==0){
        hsensor.pause();
    }else if(imput[2]==1){
        lSensor = new ColorSensor(SensorPort.S3);
        lSensor.setFloodlight(true);
    }else if(imput[2]==2){
        suelo = lSensor.getRawLightValue();
    }else if(imput[2]==3){
        linea = lSensor.getRawLightValue();
        lSensor=null;
        hline = new HiloLineFollower(linea, suelo);
        hline.start();
    }else if(imput[2]==4){
        hline.detener();
        hsensor.reanudar();
    }
    break;
```

Figura 54. Pre-configuración del hilo del seguidor de línea

4.1.2- Clase Global

Esta clase, se utiliza para crear variables globales, en este caso solo es necesario una, la que contiene el valor de la inteligencia. Es razonable que sea una variable global porque en cualquier momento puede cambiar su valor, y el estado de la inteligencia es otro, y se necesita que el resto de hilos que la utilizan accedan a su valor actualizado.

Para definirla se crea en la clase `Global` la variable `inteligencia`, y no se le añade ningún modificador de acceso, se le deja por defecto, así será accesible por cualquier clase del mismo paquete en el que se encuentra, pero oculto a otros paquetes. Si en posible ampliaciones fuera necesario que tuviera más visibilidad se le cambiaria ese modificador.

También se usa el modificador `static` para indicar que esta variable no pertenece a las instancias de la clase, si no a la misma clase, de esta forma se puede acceder a ella sin haber creado una instancia de la clase, escribiendo `Global.inteligencia`.

```
public class Global {
    public static int inteligencia;

}
```

Figura 55. Clase Global

4.1.3- Clase HiloSensores

Esta clase es un hilo que controla los sensores de choque y el de proximidad del robot, y toma decisiones dependiendo del modo de inteligencia en el que se encuentre.

Independientemente de este modo de inteligencia, esta clase siempre manda al dispositivo Android un mensaje indicando que se ha chocado o que se ha detectado un objeto.

Para que esto sea posible, al constructor de la clase es necesario pasarle un objeto del tipo `DifferencialPilot`, que permite controlar la navegación del robot, y el flujo de salida de datos, para poder enviarle los paquetes al robot.

```
public HiloSensores(DataOutputStream out, DifferentialPilot pilot)
```

Figura 56. Constructor clase HiloSensores

En el constructor de la clase también se declaran los tres sensores que utiliza, para poder detectar los eventos de ellos.

```
bump = new TouchSensor(SensorPort.S1);
bump2 = new TouchSensor(SensorPort.S2);
electroSensor = new EOPD(SensorPort.S4, true);
```

Figura 57. Declaración de sensores

El cuerpo principal del método `run`, que es el que contiene el código que va a ejecutar el hilo, está compuesto por un `while` del que solo se puede salir, si el valor de una variable del tipo `boolean` cambia.

```
public void run(){
    while(!stop){
    /**
     * Codigo a ejejutar
     */
    }
}
```

Figura 58. Método run

Para detener la ejecución del hilo cambiando el valor de esa variable, se ha creado un método llamado `detener`, en el que se cierra el flujo de salida de datos del hilo, y se cambia el valor de esa variable.

```
public void detener(){
    try {
        out.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    stop=true;
}
```

Figura 59. Método detener

Dentro del `while` que contiene todo el código a ejecutar por el hilo se pueden encontrar dos partes, la primera, es otro bucle `while` controlado también por una variable booleana y en el que solo se entrará si se llama al método `pause` que cambia el valor de esta.

```
public void pause(){
    flag=false;
}
```

Figura 60. Método pause

Este bucle sirve para mantener pausado al hilo mientras se está realizando otra tarea en la que no interesa detectar obstáculos.
Para que no esté haciendo comprobaciones continuamente mientras está pausado, se ha introducido dentro del bucle un `sleep` que duerme al hilo durante 2 segundos antes de volver a comprobarlo.

```
while(!flag){
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Figura 61. Bucle de pause

Para reanudar este estado de pausa, se ha creado otro método llamado `reanudar`, que vuelve a cambiar el estado de la variable, provocando que salga del bucle automáticamente.

```
public void reanudar(){
    flag=true;
}
```

Figura 62. Método reaunudar

La segunda parte de la que se ha hablado contiene la lógica de actuación de los sensores, que comprueba si se ha producido un choque o detectado un obstáculo. Para diferenciar el tipo de evento, se ha considerado lógico utilizar una variable booleana, que se pondrá a true si se ha producido un choque.

Se ha comprobado que los valores que devuelve el sensor de proximidad EOPD oscilan entre 1023 cuando no está detectando ningún objeto, y 350 cuando el sensor está pegado al objeto; por lo tanto se ha considerado necesario tomar la medida de que cuando el valor sea inferior a 1020 salte el evento de objeto detectado, para que así tenga el tiempo necesario para frenar y tomar las decisiones adecuadas.

```
if(electroSensor.readRawValue()<1020 ||( choque=bump.isPressed())||( choque=bump2.isPressed())){
```

Figura 63. Condición eventos sensores

Lo primero que se hace nada más detectar un evento, sea el que sea, es detener el robot.
Después se comprueba si ha sido un choque o se ha detectado un obstáculo, la diferencia entre ambos solo será el mensaje que se le envié al dispositivo Android indicando el tipo de

evento, y que si se ha detectado un obstáculo y la configuración de inteligencia está en el robot se producirá un giro de 90 grados, mientras que si lo que se ha producido ha sido un choque se retrocede una distancia antes de girar.

También, justo después de enviar el mensaje al dispositivo Android, se utiliza la función `flush` que vacía los buffer de salida y obliga a enviar todos los mensajes pendientes; si no se utiliza esta función el programa esperará a que se cierre el hilo para enviar todos los paquetes y no funcionará correctamente.

Por último se ha comprobado la configuración de inteligencia, si se encuentra en el robot se tomaran las decisiones citadas anteriormente, si por el contrario se encuentra en el móvil, se dormirá el hilo durante un segundo para que haya el suficiente tiempo a tomar una decisión antes de que se vuelva a detectar el mismo evento.

```
if(choque){
    salida[1]=0;
    out.write(salida);
    out.flush();
    Sound.playTone(1200, 100);
    if(Global.inteligencia==0){
        pilot.travel(-10);
        pilot.rotate(90);
    }else{
        Thread.sleep(1000);
    }
}else{
    salida[1]=1;
    out.write(salida);
    out.flush();
    Sound.playTone(1000, 100);
    if(Global.inteligencia==0){
    pilot.rotate(90);
    }else{
        Thread.sleep(1000);
    }
}
```

Figura 64: Control evento sensores

4.1.4- Clase HiloLineFollower

Esta clase es otro hilo dedicado al algoritmo de un seguidor de línea, en el que el robot, utilizando el sensor de color es capaz de seguir una línea.

Para entender el código primero se debe comprender la teoría de la solución tomada.
En este caso se ha construido un seguidor de línea proporcional, con el que se pretende conseguir que el robot siga la línea sin paradas ni giros bruscos.

Para ello se ha decidido que en vez de seguir la línea, lo que el robot seguirá será el borde entre la línea y el suelo. Esto conlleva a seguir a un nivel de luz que esté entre el color de la línea y el del suelo. Este nivel se llamará offset y será la media entre la luz reflejada por el suelo y por la línea.

$$offset = \frac{linea + suelo}{2}$$

Para saber hacia dónde se quiere girar, a la velocidad máxima establecida en cada motor, se le irá sumando y restando un ajuste que irá corrigiendo la dirección del robot.

$$Vmizquierdo = Vmax + ajuste$$
$$Vmderecho = Vmax - ajuste$$

El ajuste puede ser un valor positivo o negativo, por lo tanto ira variando la velocidad de los motores para corregir la dirección a la derecha o izquierda según este valor.

Para calcular el ajuste se plantea esta gráfica, en la que la velocidad de giro (ajuste), avanza en función del error.

El error es la diferencia entre el valor que se está recogiendo actualmente por el sensor y el offset.

$$error = LuzActual - offset$$

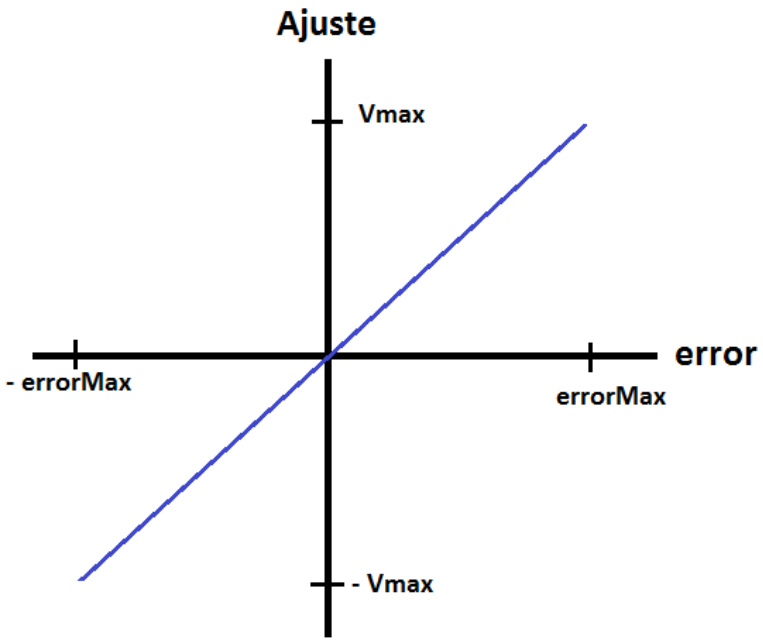

Figura 65. Gráfica ajuste seguidor de línea

En función de esta gráfica y observando que es una recta simple, se puede obtener que el ajuste es la pendiente por el error. Siendo la pendiente la constante de proporcionalidad de nuestro sistema (Kp).

$$ajuste = Kp * error$$

Solo queda calcular la pendiente, que está definida como la diferencia en el eje Y entre la diferencia en el eje X, por lo tanto para calcularla se necesitan coger dos puntos de la recta.

En este caso se ha decidido coger el punto de cuando se encuentra en error máximo, es decir cuando el sensor está sobre el suelo y no capta nada de la línea; en el punto de error máximo la velocidad de giro del robot será la velocidad máxima que se le haya asignado.

$$errorMax = suelo - offset$$

$$Kp = \frac{Vmax - (-Vmax)}{errorMax - (-errorMax)} = \frac{2Vmax}{2errorMax}$$

$$Kp = \frac{Vmax}{suelo - offset}$$

Una vez explicada la teoría se va a proceder a explicar cómo se ha implementado esta solución.

En el constructor de la clase, se reciben los parámetros de la luz reflejada por el suelo, y la luz reflejada por la línea, esenciales para los cálculos. También se define la velocidad máxima del sistema; se ha elegido un valor con el que el robot es capaz de seguir la línea sin perderse.

Después de definir estos tres valores, se calcula el offset y la constante de proporcionalidad.

En el constructor de la clase, también se declara el sensor de color, necesario para todo el proceso.

```
public HiloLineFollower(float linea, float suelo){
    cs= new ColorSensor(SensorPort.S3);
    flag = true;
    velMax=200;
    offset  = (linea + suelo)/2;
    kp = velMax/(suelo-offset);
    lineFollower = new Thread(this);
}
```

Figura 66. Constructor HiloLineFollower

Cuando el hilo se arranca, se ejecuta el código que posee el método `run`, que en este caso está compuesto por un bucle `while`, del que solo se puede salir si se llama al método `detener`, que cambia el valor de la variable.

```
public void detener(){
    flag=false;
}
```

Figura 67. Método detener

Dentro de este bucle, se pueden ver aplicadas las formulas definidas arriba.
Primero se calcula el error tomando del sensor el valor actual de la luz reflejada, y con él se calcula el ajuste.

Se restablece la nueva velocidad de cada motor y se llama a la función `forward` en cada uno de ellos para que avance con su nueva velocidad.

Cuando el programa sale del bucle se llama a la función `stop` para detener ambos motores.

```
public void run(){
    while(flag){
        error = cs.getRawLightValue() - offset;
        ajuste = kp * error;
        Motor.C.setSpeed(Math.round(velMax + ajuste));
        Motor.A.setSpeed(Math.round(velMax - ajuste));
        Motor.A.forward();
        Motor.C.forward();
    }
    Motor.A.stop();
    Motor.C.stop();
}
```

Figura 68. Método run

4.2- Descripción del código Android

En este apartado se va a proceder a describir la implementación desarrollada para la aplicación que se ejecutará en el dispositivo Android.

Esta aplicación es el puente de comunicación entre el usuario y el robot.

4.2.1- Clase LegoApplication

Esta clase hereda de la clase `Application`, que es a su vez la clase base de toda aplicación Android, y necesaria para aquellos que deseen mantener el estado global de la aplicación.

El contenido de esta clase es lo primero que se ejecuta en toda aplicación Android, por lo que se ha considerado la más correcta para la declaración de las variables globales.

Se han tomado las mismas decisiones en cuanto a modificadores que en la clase Global del robot, se ha elegido dejar el modificador de acceso por defecto para que solo pueda ser accesible por cualquier clase del paquete en el que se encuentra; y se ha usado el modificador `static` para indicar que las variables no pertenecen a las instancias de la clase, si no a la misma clase, y se pueden modificar sin tener que crear una instancia de la clase.

```
public class LegoApplication extends Application{

    static Button left, right, forward, backward;
    static int inteligencia;
    static DataOutputStream nxtDos;

}
```

Figura 69. Clase LegoApplication

4.2.2- Clase LegoBT

Esta clase es la actividad principal del programa, es la encargada de controlar los botones que dirigen al robot, así como el menú de opciones, y el hilo de recepción de los paquetes que envíe el robot.

4.2.2.1- Ciclo de vida de la clase

El primer método que se ejecuta de esta clase es el método `onCreate`, en este método está declarada la interfaz gráfica, el adaptador bluetooth, los botones y sus imágenes.

Para definir la interfaz gráfica, se le asigna la que previamente se ha programado en lenguaje xml, y se encuentra en la carpeta layout, en el documento "activity_lego_bt.xml".

```
setContentView(R.layout.activity_lego_bt);
```

Figura 70. Asignar contenedor de vista

En este método también se comprueba inmediatamente después de definir el adaptador bluetooth, si ha sido posible hacerlo y existe, en caso contrario se cierra la aplicación, ya que el bluetooth es imprescindible para su funcionamiento.

```
// Get local Bluetooth adapter
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
// If the adapter is null, then Bluetooth is not supported
if (mBluetoothAdapter == null) {
    Toast.makeText(this, "Bluetooth is not available", Toast.LENGTH_LONG).show();
    finish();
    return;
}
```

Figura 71. Declaración adaptador Bluetooth

Para definir los botones, se hace mediante el identificador que se les asigno al declararlos en el documento xml de la interfaz gráfica.

Cada botón tiene dos imágenes, una para cuando el botón está en reposo y otra para cuando se encuentra pulsado. Estas imágenes se encuentran en la carpeta "drawable" del proyecto.

```
LegoApplication.right=(Button)findViewById(R.id.Right);
Resources res = getResources();
bright=res.getDrawable(R.drawable.btright);
brightpul=res.getDrawable(R.drawable.btrightpul);
```

Figura 72. Declaración botón y sus imagenes

Por último hace una llamada el método `botones`.

Este método contiene el controlador de eventos de cada botón. Detecta tanto como si se pulsa como si se suelta.

Para detectar cuando se pulsa y se suelta un botón en vez de utilizar el método `setOnClickListener`, que es el que se suele utilizar para detectar cuando se ha clicado un botón, se ha utilizado el método `setOnTouchListener` que proporciona un evento de lo que ha ocurrido.

```
LegoApplication.left.setOnTouchListener(new OnTouchListener() {
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_UP){
         //codigo cuando se pulsa
        }
        if (event.getAction() == MotionEvent.ACTION_DOWN){
            //codigo cuando se suelta
        }

    return true;
    }
});
```

Figura 73. Control eventos botones

Cuando un botón se pulsa, se cambia la imagen del botón por el mismo pero pulsado, inmediatamente después se envía un mensaje al robot que contiene el movimiento que se desea hacer.

También en los botones de atrás, derecha e izquierda se hace una comprobación que al cumplirse, habilita al botón de adelante. Esto ocurre porque cuando el módulo de inteligencia lo posee el usuario, al detectarse un obstáculo o choque, se deshabilita el botón de adelante, y no se vuelve a habilitar hasta que se ha corregido la dirección.

```
if (event.getAction() == MotionEvent.ACTION_UP){
    if (currentapiVersion > android.os.Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1){
        LegoApplication.left.setBackground(bleft);
    }else{
        LegoApplication.left.setBackgroundDrawable(bleft);
    }
    if(nxtBTsocket!=null){
        mov[1]=0;
        LegoApplication.nxtDos.write(mov);
    }
    if(sensor==true){
        LegoApplication.forward.setEnabled(true);
        sensor=false;
    }
}
```

Figura 74. Evento soltar un botón

Cuando el botón se suelta, se vuelve a cambiar la imagen por la del botón en estado de reposo, e inmediatamente después se manda un mensaje al robot diciéndole que pare.

```
if (event.getAction() == MotionEvent.ACTION_DOWN){
    if (currentapiVersion > android.os.Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1){
        LegoApplication.left.setBackground(bleftpul);
    }else{
        LegoApplication.left.setBackgroundDrawable(bleftpul);
    }
    if(nxtBTsocket!=null){
        mov[1]=4;
        LegoApplication.nxtDos.write(mov);
    }
}
```

Figura 75. Evento pulsar un botón

Se puede observar que para cambiar la imagen del botón se realiza una comprobación, y dependiendo del resultado, se utiliza el método `setBackgroud` o `setBackgroudDrawable`. Esta comprobación compara la versión de Android del Smartphone que está ejecutando la aplicación con la versión "Ice Cream Sandwitch", esta versión es la API 15 de Android y a partir de la API 16, el método `setBrackgroundDrawable` ha sido declarado obsoleto y sustituido por el método `setBackgroud`.

Por lo tanto con esta comprobación adapta la aplicación a todas las versiones de Android, evitando de cara al usuario el problema de fragmentación de versiones que sufre.

El siguiente método que se ejecuta en la aplicación es el `onStart`, este método comprueba si está habilitado el bluetooth, y si no es así muestra una solicitud para activarlo.

```
public void onStart() {
    super.onStart();
    if (!mBluetoothAdapter.isEnabled()) {
        Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
    }
}
```

Figura 76. Método onStart

Para mandar esta solicitud se crea un `Intent` que solicita al sistema la activación del bluetooth.

Este `Intent` se envía mediante el método `startActivityForResult`, este método espera una respuesta que vendrá marcada por la etiqueta `REQUEST_ENABLE_BT`.

Cuando se reciba esta respuesta, saltará automáticamente el método `onActivityResult`, en el que en sus argumentos recibe la etiqueta que marca la respuesta de la actividad lanzada, la respuesta que indica si ha sido ejecutada correctamente, y datos que se puedan requerir.

En este proyecto, en el método `onActivityResult` se analizan dos etiquetas, una de la que ya se ha hablado, indica el resultado de si el bluetooth ha sido activado o no. Si no lo ha sido se cerrara la aplicación automáticamente, mostrando un mensaje que indicara que el bluetooth esta desactivado.

```
case REQUEST_ENABLE_BT:
    // When the request to enable Bluetooth returns
    if (resultCode != Activity.RESULT_OK) {
        Log.e(tag,"No se ha conectado el bluetooth");
        Toast.makeText(this, "Bluetooth was not enabled", Toast.LENGTH_SHORT).show();
        finish();
    }
    break;
}
```

Figura 77. Evento Bluetooth activo

La otra etiqueta es del mensaje devuelto al cerrar, la actividad que permite seleccionar el dispositivo con el que se quiere establecer conexión. Sí se ha seleccionado un dispositivo, devuelve como dato la MAC de este, y se ejecutara el método `robot` para proceder a conectarse a él.

```
case REQUEST_CONNECT_DEVICE:
    // When DeviceListActivity returns with a device to connect
    if (resultCode == Activity.RESULT_OK) {
        robot(data);
    }
    break;
```

Figura 78. Evento dispositivo seleccionado

Al método `robot` se le pasan los datos, de los que se extraerá la MAC del dispositivo con el que se quiere establecer la conexión.

```
String address = data.getExtras().getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
```

Figura 79. Extrae la MAC del dispositivo

Una vez extraída la MAC se procede a conectar el dispositivo Android con el robot, para ello primero se debe tener un objeto `BluetoothDevice` que representa al dispositivo remoto, y al que se le pasa la dirección MAC obtenida.

```
BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
```

Figura 80. Se crea el objeto BluetoothDevice

A continuación utilizando el objeto `BluetoothDevice` se puede declarar el objeto `BluetoothSocket` que define el socket que utilizará la conexión, asignándolo un identificador único de la conexión (UUID). Una vez creado el socket ya se puede proceder a establecer dicha conexión.

```
UUID uuidBt =UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
nxtBTsocket = device.createRfcommSocketToServiceRecord(uuidBt);
nxtBTsocket.connect();
```

Figura 81. Creación del socket y establecimiento de conexión

El socket que se ha creado es del tipo RFCOMM [7] que es soportado por la API de Android y orientado a conexión.

Una vez que se ha conectado correctamente, se definen el flujo de entrada y salida de datos.

```
LegoApplication.nxtDos = new DataOutputStream(nxtBTsocket.getOutputStream());
nxtIn = new DataInputStream(nxtBTsocket.getInputStream());
```

Figura 82. Creacion de los flujos de entrada y salida

Y cuando están definidos estos flujos, se ejecuta el hilo de recepción de paquetes.

El último método que queda por explicar, del ciclo de vida de la aplicación, es el método `onDestroy`, este método será el que se ejecute al cerrar la aplicación. En él se le manda un mensaje al robot indicándole que se va a cerrar la aplicación, se para el hilo de recepción y se cierra el socket y los flujos de entrada y salida de datos.

```
if(nxtBTsocket!=null){
    try {
        out[0]=2;
        out[1]=1;
        out[2]=0;
        LegoApplication.nxtDos.write(out);
        flag=false;
        imput.cancel(true);
        nxtBTsocket.close();
        nxtBTsocket = null;
        LegoApplication.nxtDos.close();
        LegoApplication.nxtDos = null;
        nxtIn.close();
        nxtIn=null;
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figura 833. Método onDestroy

4.2.2.2- Hilo de recepción de paquetes

El hilo que se ha utilizado en este proyecto hereda de la clase `AsyncTask` [8], esto ofrece algunas soluciones frente a trabajar con un `Thread`.

Cuando en Android se usa un hilo, es muy normal que se necesite acceder a la interfaz gráfica para modificar algún elemento, pero sin embargo existe la segunda regla del modelo de un único proceso, esta regla dicta que no se puede modificar la interfaz de usuario fuera del hilo principal de dicha interfaz.

Para solucionar este problema Android ofrece algunos métodos de ayuda, para acceder a la interfaz de usuario desde un `Thread`. Pero cuando el hilo empieza a ser largo, esa manera de trabajo se hace algo complicada, ya que el código se hace difícil de leer.

Por ello aporta otra solución, que lo hace todo más sencillo y limpio. Esta solución es la clase `AsyncTask`, de la que se heredan cuatro métodos:

- `doInBackground()`: el código en este método será el ejecutado en segundo plano.
- `onPreExecute()`: aquí se encuentra el código que se quiere ejecutar antes de empezar la tarea en segundo plano.
- `onProgressUpdate()`:este método es el único donde se puede introducir código que acceda a la interfaz de usuario. Para acceder a este método se debe llamar a `publishProgress()` desde el método `doInBackground()`.
- `onPostExecute()`: aquí está el código a ejecutar inmediatamente después de que la tarea en segundo plano finalice.

Para implementar este proyecto solo han sido necesarios utilizar dos de esos métodos.

En el método `doInBackground()`, se encuentra un bucle `while` que se encarga de la espera de nuevos paquetes. Una vez que se ha recibido uno se le envía al método `onProgressUpdate()` para ser analizados.

```
protected Boolean doInBackground(Void... arg0) {
    byte [] inDatos=new byte[3];
    while(flag){
        try {
            nxtIn.read(inDatos);
        } catch (IOException e) {
            e.printStackTrace();
        }
        publishProgress(inDatos[0],inDatos[1],inDatos[2]);
    }
    return flag;
}
```

Figura 84. Método doInBackground

Los paquetes son analizados en el método `onProgressUpdate()` ya que toda decisión que se tome, causará un cambio en la interfaz gráfica.

El dispositivo Android solo recibe paquetes avisando de que se ha producido un choque o se ha detectado un objeto, e independientemente del tipo de configuración de inteligencia que se tenga, se mostrará un mensaje por pantalla indicando lo que ha ocurrido, y quien debe tomar la decisión.

Independientemente del tipo de inteligencia configurado se pueden dar tres casos.

En el primero la inteligencia está configurada en el robot y el dispositivo Android solo mostrará un mensaje por pantalla de lo que ha ocurrido.

```
if(values[1]==0){
    Toast.makeText(getApplicationContext(), "Se ha producido un choque, " +
            "el robot tomará una decisión.", Toast.LENGTH_SHORT).show();
}else if(values[1]==1){
    Toast.makeText(getApplicationContext(), "Se ha detectado un obstaculo, " +
            "el robot tomará una decisión.", Toast.LENGTH_SHORT).show();
}
```

Figura 85. Decisión con inteligencia en el robot

El segundo es el que la inteligencia está configurada en el móvil de forma automática y aparte de mostrar el mensaje por la pantalla del dispositivo, también enviará al robot el paquete indicándole que debe hacer.

```
if(values[1]==0){
    Toast.makeText(getApplicationContext(), "Se ha producido un choque, " +
            "el movil tomará una decisión.", Toast.LENGTH_SHORT).show();
    try {
        out[0]=0;
        out[1]=1;
        out[2]=-10;
        LegoApplication.nxtDos.write(out);
        out[0]=0;
        out[1]=3;
        out[2]= 90;
        LegoApplication.nxtDos.write(out);
    } catch (IOException e) {
        e.printStackTrace();
    }
```

```
}else if(values[1]==1){
    Toast.makeText(getApplicationContext(), "Se ha detectado un obstaculo, " +
            "el movil tomará una decisión.", Toast.LENGTH_SHORT).show();
    out[0]=0;
    out[1]=3;
    out[2]=90;
    try {
        LegoApplication.nxtDos.write(out);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figura 86. Decisión con inteligencia en el móvil-automática

En la tercera opción la inteligencia la posee el usuario, y la medida que se tomará aparte de mostrar el mensaje por pantalla, será bloquear el botón de dirección "adelante" para que no se pueda usar mientras no se cambie dicha dirección. También se le cambiará la imagen para mostrarlo en reposo. Se ha utilizado una variable booleana para que el resto de botones pueda saber si está bloqueada la dirección "adelante."

```
if(values[1]==0){
    Toast.makeText(getApplicationContext(), "Se ha producido un choque. " +
            "Corrija su dirección.", Toast.LENGTH_SHORT).show();
}else if(values[1]==1){
    Toast.makeText(getApplicationContext(), "Se ha detectado un obstaculo. " +
            "Corrija su dirección.", Toast.LENGTH_SHORT).show();
}
sensor=true;
LegoApplication.forward.setEnabled(false);
if (currentapiVersion > android.os.Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1){
    LegoApplication.forward.setBackground(bforward);
}else{
    LegoApplication.forward.setBackgroundDrawable(bforward);
}
```

Figura 87. Decisión con inteligencia en el móvil-usuario

4.2.2.3- Menú de opciones

En esta actividad, se dispone de un menú de opciones. Para controlar los eventos al pulsar dichas opciones se utiliza el método `onOptionsItemSelected`, al que se llamará el sistema al pulsar una de las opciones.

La primera opción que dispone el menú es el botón "Conectar con un dispositivo", esta opción lanza la actividad definida en la clase `DeviceListActivity` (que se explicará posteriormente). Para lanzarla utiliza el método `startActivityForResult` que espera la recepción de una respuesta al terminar dicha actividad, marcada por la etiqueta `REQUEST_CONNECT_DEVICE`.

```
Intent serverIntent = new Intent(this, DeviceListActivity.class);
startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
```

Figura 88. Lanzar actividad buscar dispositivos

La segunda opción es el menú inteligencia, que al pulsarlo abre un submenú donde se puede elegir cualquiera de las tres opciones de inteligencia. El proceso al pulsar cualquiera de ellas será el mismo, se cambiará la variable de inteligencia, y se enviará un mensaje al robot indicándole la nueva opción seleccionada.

```
LegoApplication.inteligencia=0;
try {
    if(nxtBTsocket!=null){
        out[0]=2;
        out[1]=0;
        out[2]=0;
        LegoApplication.nxtDos.write(out);
    }else{
        Toast.makeText(getApplicationContext(), "Error al enviar el mensaje. " +
                "Ningún dispositivo Bluetooth encontrado.", Toast.LENGTH_SHORT).show();
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

Figura 89. Cambiar opción de inteligencia

La tercera opción es la de extras, que al pulsarla de nuevo abrirá un submenú, esta vez mostrando dos opciones, “Leyes de la Robótica” y “Seguidor de línea”. De nuevo se pulse la que se pulse se enviará un mensaje al robot con la opción elegida.

4.2.3- Gestión de dispositivos bluetooth: Clase DeviceListActivity.

Esta clase es una actividad que muestra los dispositivos bluetooth asociados con el Smartphone y también permite buscar nuevos dispositivos.

Antes de nada y para entender esta clase, hay que explicar la necesidad de crear un objeto del tipo `BroadcastReceiver` con el fin de registrar los eventos del sistema que se le hayan asignado. Así cuando el sistema encuentre un dispositivo bluetooth nuevo se podrá capturar el evento y recoger su información. En este caso esa información será su nombre y su dirección MAC.

```
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
                mNewDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
            }
        }
    }
};
```

Figura 90. Objeto del tipo BroadcastReceiver

Como se puede observar se comprueba si el evento recibido es del tipo `ACTION_FOUND`, que es el que indica en Android que se ha encontrado un nuevo dispositivo, y si es así, se comprueba si ya era un dispositivo emparejado, y si no lo es se agrega a la lista de nuevos dispositivos.

Una vez explicado esto, se va a proceder a hablar sobre el funcionamiento los diferentes métodos que tiene esta clase.

El primero que se va a analizar, es el método `onCreate` que como en toda actividad Android, es el primero que se ejecuta. En este método se inicializan los adaptadores, necesarios para añadir los nuevos dispositivos encontrados a la lista que se mostrará por pantalla.

En este caso se usaran dos adaptadores, uno para los dispositivos emparejados, y otro para los dispositivos nuevos, ya que se posee una lista para para cada caso.

Para añadir los elementos a la lista, al estar cada fila compuesta por una única cadena de texto, se puede solucionar utilizando el objeto `ArrayAdapter`, si fuera más compleja se tendría que definir un adaptador, indicando como se rellena cada parámetro de la fila.

Para inicializar el objeto `ArrayAdapter` lo primero que se debe hacer es indicarle en sus argumentos cual es el documento XML que contiene la estructura de cómo se mostrará la fila gráficamente.

```
mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.device_name);
mNewDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.device_name);
```

Figura 91. Declaración de los ArrayAdapter

Una vez inicializado, el siguiente paso es declarar el elemento `ListView` al que se le asignará el adaptador.

```
ListView pairedListView = (ListView) findViewById(R.id.paired_devices);
```

Figura 92. Declaración del ListView

Cuando la lista se ha declarado, se le asigna el adaptador, y se activan los eventos, para detectar que elemento de la lista se ha seleccionado.

```
pairedListView.setAdapter(mPairedDevicesArrayAdapter);
pairedListView.setOnItemClickListener(mDeviceClickListener);
```

Figura 93. Asignación del ArrayAdapter al ListView

Lo siguiente que se hará será asignar los tipos de eventos que se quiere que el objeto `BroadcastReceiver`, explicado anteriormente, detecte.

En este caso se detectarán dos tipos de eventos, cuando se detecte un nuevo dispositivo, y cuando se haya finalizado la búsqueda.

```
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
this.registerReceiver(mReceiver, filter);
filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
this.registerReceiver(mReceiver, filter);
```

Figura 94. Registro de eventos a detectar

Por último en este método se buscaran los dispositivos emparejados, y se añadirán a la lista, si se encuentra alguno.

```
if (pairedDevices.size() > 0) {
    findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
    for (BluetoothDevice device : pairedDevices) {
        mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
    }
} else {
    String noDevices = "NO DEVICES";
    mPairedDevicesArrayAdapter.add(noDevices);
}
```

Figura 95. Búsqueda de dispositivos emparejados

Otro método que se encuentra en esta clase es el `doDiscovery`, en el que se inicia la búsqueda de nuevos dispositivos, antes de ello se comprueba si la búsqueda ya estaba activa, para cancelarla e iniciarla de nuevo.

En este método también se pone visible la barra de título “Nuevos dispositivos”, que esta oculto hasta que no se inicia la búsqueda.

```
findViewById(R.id.title_new_devices).setVisibility(View.VISIBLE);
if (mBtAdapter.isDiscovering()) {
    mBtAdapter.cancelDiscovery();
}
mBtAdapter.startDiscovery();
```

Figura 96. Búsqueda de nuevos dispositivos

Por último queda por analizar el objeto del tipo `OnItemClickListener`, en el que se analizan los eventos al pulsar un elemento de la lista.

Aquí se empaqueta la dirección MAC del elemento seleccionado para crear la respuesta que obtendrá la actividad que llamó a esta.

```
private OnItemClickListener mDeviceClickListener = new OnItemClickListener() {
    public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {
        mBtAdapter.cancelDiscovery();
        String info = ((TextView) v).getText().toString();
        String address = info.substring(info.length() - 17);
        Intent intent = new Intent();
        intent.putExtra(EXTRA_DEVICE_ADDRESS, address);
        setResult(Activity.RESULT_OK, intent);
        finish();
    }
};
```

Figura 97. Controlador de eventos de la lista

4.2.4- Clase SeguidordeLineaActivity

Esta actividad es la encargada de indicarle al usuario los pasos a seguir, para preparar al robot para el programa "Seguidor de línea" explicado en el capítulo 4.1.4 de esta memoria.

Estas indicaciones al usuario se realizan a través de tres cuadros de dialogo, declarados e iniciados en el método `onCreate`.

Estos cuadros de dialogo son exactamente iguales, la diferencia entre ellos es únicamente, el mensaje a mostrar y la acción que se produce al pulsar el botón "Ok".

Cuando se crea un cuadro de dialogo, se tienen multitud de opciones, para este proyecto solo se le han aplicado los métodos:

- `setMessage`: en el que se le pasa el mensaje que se quiere mostrar.
- `setCancelable`: en el que se selecciona si se quiere que el cuadro de dialogo se pueda quitar pulsando el botón de atrás del Smartphone, o se obligue a tener que pulsar alguno de los botones disponibles en el cuadro de dialogo.
- `setNeutralButton`: en el que se crea el botón que tendrá y, el evento de lo que ocurre al pulsar dicho botón. Otras opciones aplicables por separado o en conjunto para botones son `setPositiveButton` y `setNegativeButton`.

Este es un ejemplo del primer cuadro de dialogo que se crea, como se puede ver al pulsar el botón "Ok" se envía el paquete al robot y se muestra el siguiente cuadro de dialogo.

```
AlertDialog.Builder builderSensor = new AlertDialog.Builder(this);
builderSensor.setMessage("Conecte el sensor de color al puerto 3, mirando " +
            "hacia el suelo y a una distancia de 5mm de él. Después pulse ok.")
       .setNeutralButton("Ok", new OnClickListener() {

        public void onClick(DialogInterface dialog, int which) {
            out[0]=3;
            out[1]=1;
            out[2]=1;
            try {
                LegoApplication.nxtDos.write(out);
            } catch (IOException e) {
                e.printStackTrace();
            }
            AlertDialog alertBlanco = builderBlanco.create();
            alertBlanco.show();
        }
    }).setCancelable(false);

AlertDialog alertSensor = builderSensor.create();
alertSensor.show();
```

Figura 98. Declaración y creación de un cuadro de dialogo

Si se desean ver los tres mensajes que muestran los cuadros de dialogo, se puede hacer en el Capítulo 3.4 donde se encuentra el manual de uso, y se puede ver una captura de ellos.

Si se quiere finalizar el programa seguidor de línea, solo se tiene que pulsar el botón atrás del Smartphone, que llamara al método onDestroy.

En este método se envía un mensaje al robot para que cancele la ejecución del programa.

```
out[0]=3;
out[1]=1;
out[2]=4;
try {
    LegoApplication.nxtDos.write(out);
} catch (IOException e) {
    e.printStackTrace();
}
```

Figura 99. Método onDestroy

4.2.5- Documento AndroidManifest

El documento AndroidManifest es un documento escrito en XML donde se declaran todas las especificaciones de la aplicación.

En él se puede encontrar desde la versión mínima necesaria de Android para ejecutar la aplicación, hasta los permisos que se necesitan para dicha ejecución. También se encuentra las actividades que la poseen y cuál es la principal, el icono y estilos de la aplicación, y hasta cual es el documento que contiene la clase `Application`, si esta se ha modificado.

En el documento se puede encontrar que se ha definido la versión mínima del skd como la 8, esto quiere decir que la aplicación funcionará en cualquier Smartphone que posea la versión "Android Froyo" o superior.

La etiqueta `tarjetSdkVersion` indica la versión de Android con la que se han realizado las pruebas.

```
<uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="18" />
```

Figura 100. Asignación de la versión mínima de la API

Lo siguiente que se ha definido en este documento son los permisos que se necesitan para el funcionamiento de esta aplicación. En este caso solo se requiere el permiso de acceso a la interfaz bluetooth.

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

Figura 101. Declaración de permisos

Después se pasa a definir donde está el documento que hereda de la clase `Application`, el icono de la aplicación, su nombre, y el tema que utiliza.

```
android:name="com.example.legobt.LegoApplication"
android:allowBackup="true"
android:icon="@drawable/ic_legobt"
android:label="@string/app_name"
android:theme="@style/AppTheme" >
```

Figura 102. Declaración de las opciones de la Aplicación

Para terminar se encuentran definidas las actividades que conforman la aplicación, la primera que es definida es la principal, que es la que se encuentra al abrir la aplicación.

Para definirlas es necesario indicarle el documento que contiene dicha actividad, el título de esta, y en este caso se han forzado a que la orientación de la aplicación sea siempre en vertical.

```
<activity
    android:name="com.example.legobt.LegoBT"
    android:label="@string/title_activity_lego_bt"
    android:screenOrientation="portrait" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name="com.example.legobt.DeviceListActivity"
    android:label="@string/title_activity_device_list"
    android:screenOrientation="portrait" >
</activity>
<activity
    android:name="com.example.legobt.SeguidordeLineaActivity"
    android:label="@string/title_activity_sensor_extra"
    android:screenOrientation="portrait" >
</activity>
<activity
    android:name="com.example.legobt.MarcarRumboActivity"
    android:label="@string/title_activity_marcar_rumbo"
    android:screenOrientation="portrait">
</activity>
```

Figura 103. Declaración de las diferentes Activities

5- Conclusiones y futuras mejoras

5.1- Conclusiones

Se llega al final de este proyecto habiendo conseguido una aplicación muy completa empleando para ella diferentes tecnologías. Tras todo el desarrollo llega el momento de sacar conclusiones de lo que se ha observado.

5.1.1- Android

Se ha conseguido desarrollar una aplicación para Android, que deja como conclusión que si se tienen unos conocimientos básicos sobre programación orientada a objetos, desarrollar aplicaciones sencillas para Android puede resultar fácil. Posee una API muy completa, y mucha documentación por distintas webs, gracias a la gran comunidad de desarrolladores que tiene.

Los puntos más importantes que se han aprendido sobre el desarrollo de aplicaciones Android son:

- El ciclo de vida de una aplicación.
- La utilidad de la clase `Asynctask`.
- EL manejo del bluetooth en aplicaciones Android.

Este proyecto también ha ayudado a comprobar que la fragmentación en Android es un problema real, y que un desarrollador puede tener serios problemas, si quiere adaptar su aplicación a todas las versiones.

5.1.2- Robot Lego

Durante el desarrollo de este proyecto, se ha podido comprobar el potencial que tiene el robot lego, con un hardware muy limitado. Posee una gran cantidad de sensores, y junto con la posibilidad de programarlo en Java usando todo su potencial, se convierte en una herramienta muy potente, que puede dar pie a muchos proyectos diferentes.

A la hora de introducir archivos de audio, se ha encontrado con la limitación de la memoria Flash, que puede llegar a ser insuficiente.

También, se ha comprobado que algunos sensores como el de ultrasonidos, no son fiables.

Salvo estos pequeños inconvenientes las impresiones que ha dejado el robot Lego son realmente buenas.

5.1.3- Bluetooth

Durante el desarrollo de este proyecto se ha sacado como conclusión, que sí se tiene el conocimiento adecuado de la API del dispositivo donde se va a trabajar con el bluetooth, las conexiones son sencillas y muy estables.

También se ha obtenido como conclusión que el uso del bluetooth ha sido suficiente para el desarrollo de este proyecto, cubriendo todas las necesidades que se tenían.

4.2- Futuras mejoras

Como posibles ampliaciones a este proyecto, se sugieren las siguientes ideas:

- Implementar un módulo de inteligencia más eficiente, el cual haga capaz al robot de esquivar obstáculos sin detenerse.
- Añadir nuevas funcionalidades extras, como marcarle un rumbo a seguir utilizando el sensor de la brújula.
- Desarrollar un reconocimiento de voz en la aplicación que permita controlar al robot con la voz.
- Permitir a la aplicación conectarse automáticamente con el robot al iniciarse, habiendo sido configurada esta opción previamente.
- Crear un menú de configuración, en el que se pueda seleccionar los sensores que se desean utilizar y el puerto al que están conectados.
- Utilizar el sensor GPS para que el robot le mande al Smartphone, las coordenadas de su posición.
- Permitir dirigir al robot girando el Smartphone, utilizando los sensores de este.
- Ampliarle la conectividad inalámbrica mediante Wi-Fi.
- Controlar el robot remotamente a través de internet, utilizando la interfaz Wi-Fi, y ayudándose de una cámara que emitiría video en directo. La cámara estaría controlada por un microordenador tipo Raspberry Pi [9]. Y todo esto estaría acoplado en el robot.

Todas estas mejoras se pueden desarrollar en aplicaciones independientes, o en una sola aplicación final que las incluya todas.

6- Referencias

[1] Web oficial de desarrollo en Android.

http://developer.android.com

[2] Web oficial del robot Lego Mindstorms

http://mindstorms.lego.com/

[3] Bluetooth

http://es.wikipedia.org/wiki/Bluetooth

[4] Tutorial para realizar un seguidor de línea

http://www.lejosconlego.com/2012/07/como-hacer-un-line-follower-seguir-la.html

[5] Web del proyecto leJOS

http://www.lejos.org/

[6] Protocolo de comunicación OBEX

http://es.wikipedia.org/wiki/OBEX

[7] Diferentes protocolos bluetooth

http://es.wikipedia.org/wiki/Protocolos_Bluetooth

[8] Tutorial de diferencias de uso entre Thread y AsyncTask

http://www.sgoliver.net/blog/?p=3099

[9] Web oficial de Raspberry Pi

http://www.raspberrypi.org/