Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

App de control remoto de un robot sobre plataforma Android

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Daniel Dapena Pérez

Tutor: Alberto Pérez Jiménez

2013-2014

Resumen

Los teléfonos inteligentes, en adelante *Smartphones*, son algo más que un dispositivo electrónico. Se están convirtiendo en herramientas personales cada vez más necesitadas y solicitadas. En vista de esas necesidades, los smartphones se están convirtiendo en herramientas cada vez más potentes, y por ende, en soluciones para problemas de distinta índole.

La robótica, por el otro lado, también está en auge. No sólo al nivel industrial, sino también al nivel comercial y personal. Son una opción de presente y futuro.

Teniendo esto en cuenta, *Alive Client* es el nombre de la aplicación que intenta unir ambos campos: telefonía inteligente y robótica. *Alive Client* es una aplicación móvil para sistemas *Android* capaz de comunicarse con un Robot a través de un servidor instalado en él, de tal manera que nos permite poder ver a través de robot, controlar sus sensores, cambiar su comportamiento e incluso controlarlo como si fuera un radiocontrol

Para la consecución de tal fin, se ha desarrollado una aplicación *Android* con soporte multilenguaje capaz de comunicarse por internet y traducir las órdenes del usuario en órdenes que el servidor del robot entienda lo que el usuario quiere que haga.

Palabras clave: aplicación, android, robot, aliveclient, java

Abstract

Smartphones are more tan just electronic devices. They are turning into personal tools that have a growing demand and necessity. In view of those needs, smartphones are becoming more powerul and are also progressively turning into solutions for different problems of any kind.

Robotics, on the other hand, is also booming. Not only on an industrial level, but also at a comercial and personal one. They are an option for the present and the future.

Bearing this in mind, Alive Client is the name of an applications that tries to unite both fields: smartphones and robotics. Alive Client is an applications for Android phones which is capable of communicating with a robot through a server that is installed in it. This lets us see from the robot's perspective, control its sensors. Change its behaviour and even control it the same way you wold an RC.

To achieve this goal, it has been developed an Android application with multilingual support. It is able to communicate via Internet and translate the orders from the user so the server of the robot can understand what the user wants it to do.

Keywords: application, android, robot, aliveclient, java

Tabla de contenidos

1. Introducción

Un poco de historia

El mundo de las telecomunicaciones y el de la robótica están en continuo cambio y evolución. Sin embargo, si echamos la vista atrás, nos damos cuenta que los de mi generación, años 90, somos de los primeros nacidos en plena era tecnológica.

Aunque los nativos digitales son considerados todos aquellos nacidos a partir de los años 80, no es hasta mi generación que nacieron prácticamente con el ordenador en casa. Y esto nos da qué pensar lo rápido que ha sido.

Generaciones anteriores han visto esos cambios. La evolución de la tecnología ha dejado en los hogares un sinfín de electrodomésticos y equipos electrónicos que han ido poco a poco facilitando la vida a las personas.

No obstante es en la informática donde se puede apreciar la mayor evolución. La informatización ha ido in crescendo hasta encontrarnos en cada hogar prácticamente un ordenador que nos puede hacer tareas impensables hace 20 años.

En el presente proyecto, se aúna la tecnología de la telecomunicación con la robótica. Dos ramas que están totalmente en pleno apogeo: hace años era difícil ver en las casas teléfonos móviles, ahora se ven, y a pares. Por otra parte, cada vez es más usual ver en las casas drones de juguete, que en resumidas cuentas son robots. Y no es muy compleja la idea de pensar que los robots poco a poco irán adentrándose en la vida de las personas.

Evolución de la telefonía móvil

La historia de los teléfonos móviles se remonta al año 1983, cuando *Motorola* creo su *DynaTAC*. Desde entonces, los saltos evolutivos de la telefonía móvil han sido cuantiosos, haciendo posible que un simple teléfono móvil que permitía llamar, haya ido adquiriendo funcionalidades extras.

Entre dichas funcionalidades, destacan el envío de mensajes *SMS*, la posibilidad de gestionar una agenda electrónica de contactos, un calendario, reproducción de archivos musicales. Ya con los *Smartphone*, la explosión funcional fue más notoria, incluyendo al teléfono la capacidad de poder conectarse a internet, hacer fotografías, vídeos, jugar, utilizar sus numerosos sensores de movimiento, e incluso dotándole de un sistema de posicionamiento global, *GPS*.

Sin embargo, aunque existan conceptos de *Smartphone* desde el año 1992, cuando *IBM* presentó su *Simon Personal Communicator*, que incluía ya un panel táctil, o del año 1997, el Ericsson R380, ya con su propio sistema operativo *Symbian OS*, no es hasta el apogeo de *Apple* con su famoso *iPhone* y su sistema operativo *iOS* cuando el *Smartphone* se hace popular, cuando presentaba en su *keynote* de 2007 como el "primer *Smartphone*".

Ya en ese mismo año, *Android* sería presentado bajo la fundación del *Open Handset Alliance* [1]: un grupo de compañías tanto de *hardware* como *software* y telecomunicaciones comprometidas en el avance de estándares de dispositivos móviles.

Un año más tarde llegaría el primer teléfono con *Android*, el *HTC Dream*. Éste fue gratamente aceptado, consiguiendo un millón de unidades vendidas en Estados Unidos en poco más de un año, y con una crítica que lo definía como robusto, sólido y futurista [2].

Desde entonces, en cada revisión del sistema operativo de Android, se han ido añadiendo nuevas funcionalidades al Smartphone, agregándole al dispositivo valor, y haciéndolos cada día más indispensable para la vida cotidiana.

Evolución de la robótica

En el campo de la robótica el salto ha sido (y está siendo) más lento. Se puede considerar que la leva, el resorte, las poleas de *Arquímedes* (287-212 a.C.), junto con *Herón de Alejandría* (siglo I a.C.) que, por su parte, creó una serie de mecanismos a chorro que emulaban aves que gorjeaban y bebían, son padres de los robots.

Mucho más tarde, en el Renacimiento, *Leonardo Da Vinci* crearía la bomba centrífuga, la transmisión por correa, cadena de rodillos, paracaídas, tornillos sin fin, torcedora de seda, e incluso un león animado (1499).

Sin embargo, no es hasta el siglo XIX con el *Telar de Jacquard*, en los inicios de la revolución industrial, que no nos encontramos ya un sistema parecido a lo que hoy en día conoceríamos como robot propiamente dicho. Dicho telar funcionaba ya con programación de tarjetas perforadas.

La robótica, al igual que la informática, tiene gran influencia de la ciencia ficción del siglo XX. Tal es así que *Isaac Asimov* formularía las leyes de la robótica en el año 1942, las cuales formulaban que:

1. Un robot no puede dañar a un ser humano o, por inacción, permitir que un ser humano resulte dañado
2. Un robot debe obedecer las órdenes dadas por los seres humanos excepto cuanto tales órdenes entren en conflicto con La Primera Ley.
3. Un robot debe proteger su propia existencia hasta donde esta protección no entre en conflicto con La Primera y/o Segunda Ley.

Los precursores más inmediatos de la robótica actual ya los encontramos en los manipuladores teleoperados de *Goertz* de la década de los 40 y 50. Justo después, empezarían a aparecer las primeras aplicaciones industriales. En el año 1967 concretamente sería la primera aplicación industrial: el *Unimate 2000* en la planta de *General Motors*, fabricando carrocerías.

Actualmente, la robótica la podemos encontrar en dos sentidos. En la industria, y en la industria de servicios. Robots para fabricar, y robots para servir. Para hacernos una idea, en el año 2012 se llegaron a vender 160.000 unidades de robots, moviendo 2.5 billones de dólares en cuanto a software, periféricos e ingeniería se refiere [3].

Así pues, mientras que la telefonía móvil tiene un recorrido desde los años 80, la robótica lo ha tenido a largo de siglos. No obstante, si nos centramos en la robótica industrial, nos podemos percatar que la historia es bastante reciente. ¿Quién no conoce a día de hoy a alguien nacido en la década de los 50? Eso nos debería dar una idea de la gran evolución.

Motivación

El presente proyecto presenta la conexión entre la telefonía móvil y la robótica. Aprovechando las virtudes del sistema operativo *Android*, y las facilidades que se otorgan para el desarrollo de aplicaciones, se ha desarrollado una aplicación capaz de comunicarse con un robot a través de un servidor.

De esta forma se intenta conectar estos dos grandes campos tecnológicos con tanto presente y futuro.

¿Para qué se iba a necesitar una aplicación como ésta?

La respuesta a esta pregunta es casi tan amplia como el campo de la robótica pueda serlo. La mayoría de robots se entienden como autónomos, pero aun estando automatizados, es posible que fallen.

El ámbito de uso de la aplicación alberga desde líneas industriales con robots trabajando, donde el operario pueda controlar el robot en caso de falla, o ver lo que el robot tenga a su alrededor, desde la comodidad de su teléfono o *Tablet*.

Por el otro lado, a nivel particular, podemos conectar la aplicación a robots domésticos, ya bien sean de asistencia o de ocio. En el caso de asistencia, *¿qué pasaría si el robot estuviera cuidando de una persona en un hospital?* Dicho robot asistencial podría estar supervisado por una persona a través de su teléfono, y podría incluso adaptarse para poder hablar al teléfono y que el robot hiciera de intermediario. En el caso del ocio, las posibilidades son también muy amplias. Cualquier tipo de robot radiocontrol podría ser manejado desde el dispositivo móvil.

2. Objetivos

El proyecto de desarrollo de una aplicación, como es evidente, tiene como objetivo principal la realización de dicha aplicación. No obstante, y dado las características del soporte de la misma, es decir, un móvil con sistema operativo *Android*, se derivan del objetivo principal una serie de objetivos secundarios.

Objetivo Principal:

- Desarrollar una aplicación capaz de comunicarse con un servidor para enviar órdenes a un robot

Objetivos Secundarios:

- Diseñar la aplicación según los cánones de diseño de *Android*.

- Implementar la aplicación versión Tableta para aprovechar las virtudes de pantallas más grandes.

- Solventar el problema de fragmentación de dispositivos, haciendo la aplicación consistente en todo dispositivo *Android*.

- Crear una aplicación con soporte multilingüe.

- Minimizar el uso de memoria *RAM* de la aplicación, así como del uso de *CPU*, para incrementar la vida de la batería, punto crítico en móviles.

- Minimizar el uso de la conexión a internet, también para incrementar la vida de la batería.

- Flexibilizar el uso de la aplicación: que sea capaz de mandar órdenes a cualquier servidor de robot, haciendo uso de una interfaz tipo consola.

La consecución de los objetivos secundarios es casi tan vital como el del principal, dado que la oferta a nivel de aplicaciones móviles es muy extensa. Dichos objetivos pueden ser la nota diferenciadora que podrá hacer que los usuarios en potencia se decanten por una u otra.

3. Estado del arte

Con la aplicación *Alive Client* se abre una puerta que nos puede llevar a un pasillo enorme. Tanto es así, que las posibilidades, como se ha mencionado con anterioridad, son infinitas.

Por vez primera se crea una aplicación capaz de controlar un robot remotamente, con el único requerimiento, para el usuario, de tener conexión a internet.

Al hablar de una aplicación de móvil para *Android*, es casi inevitable tener que mencionar al lenguaje utilizado, Java. Un lenguaje desarrollado por *James Gosling* de *Sun Microsystems*, y que acabaría siendo comprada por *Oracle*.

Si la historia de los robots tal y como los conocemos es corta, y la de los teléfonos móviles más aún, ni que decir tiene que el lenguaje Java les gana en juventud. *Java* fue publicado en el año 1995.

Java presentaba una peculiar y novedosa ventaja. La portabilidad que ofrece su *bytecode*, o código intermedio, permitiendo poder ejecutarlas en cualquier máquina es su punto fuerte. Gracias a su máquina virtual *Java* (*JVM*), se puede interpretar dichas compilaciones sin tener ningún problema con la índole de la arquitectura de la máquina donde se ejecute.

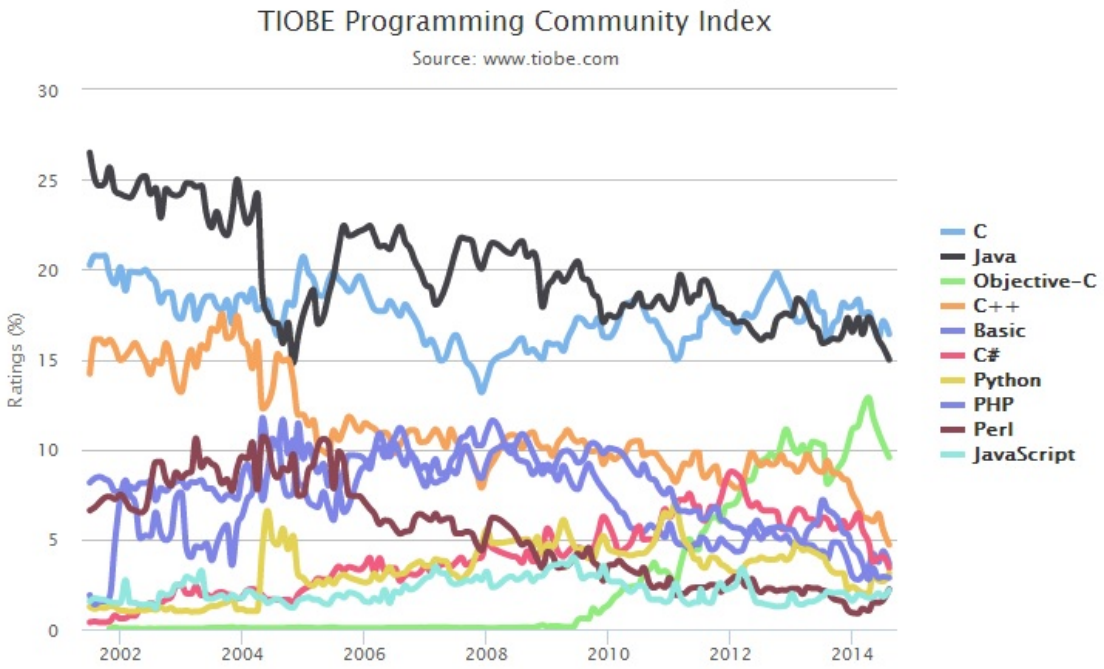


Ilustración 1. Uso de lenguajes de programación. Fuente: TIOBE [4]

Como se puede comprobar en la ilustración 1, *Java* ha ido perdiendo popularidad año tras año. No obstante, aun así sigue siendo el segundo lenguaje más utilizado, siendo sólo adelantado por el ilustre *lenguaje C* de *Dennis Ritchie*. Es curioso que la salida de *Android* no frenó casi la caída de Java, y sin embargo, la salida al mercado de *iOS* sí que ha repuntado en un incremento importante en la utilización del *Objective-C*.

El “Java” de Android

Aunque *Android* trabaje con *Java*, no es *Java* todo lo que reluce. El auténtico Java utiliza una máquina virtual de Java para interpretar los códigos compilados en *bytecode*. Sin embargo, esto no es lo que ocurre en *Android*.

Una aplicación *Android* no es compilada en *bytecode*, sino que su *JVM* ha sido sustituida por la *DVM* que obedece a *Dalvik Virtual Machine*, la cual no ejecuta *bytecode*, sino *Dalvik Executable (DEX)*.

Esta máquina virtual ha sido optimizada teniendo en cuenta los condicionantes que los dispositivos móviles imponen: recursos hardware escasos de *CPU* y memoria (aunque cada vez menos) y batería escasa principalmente.

Además del compilador de *Java* modificado, también hay que tener en cuenta que una aplicación no es completamente *Java*. El *XML* también tiene un peso importante en la aplicación, dado que es utilizado para todo aquello que el usuario ve: la interfaz gráfica. El *XML* no suele ser considerado, pero ni que decir tiene que una buena interfaz gráfica hace de una aplicación una aplicación mejor.

4. Diseño del proyecto

En el presente apartado se detallan las dos partes del proyecto que no son aplicación: Robot y Servidor, para una mejor comprensión del conjunto del proyecto.

Robot

Robot concebido inicialmente como un proyecto para demostración de algoritmos gráficos, tales como detección de caras, objetos y *OCR*, se ha reconvertido en todo un robot autónomo, creado a partir de una placa base de formato *mini-ITX* y un procesador *Dual Core* de *32bits*, 2 cámaras (de las cuáles una es *Kinect*), sensores de ultrasonido, giroscopios, acelerómetros y un compás.

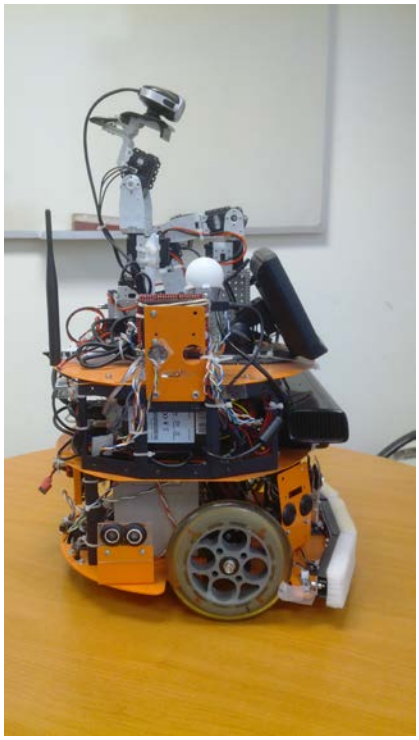


Ilustración 2. Perfil del robot. En la parte superior se observa la cámara principal del robot, que se puede mover con el commando move.

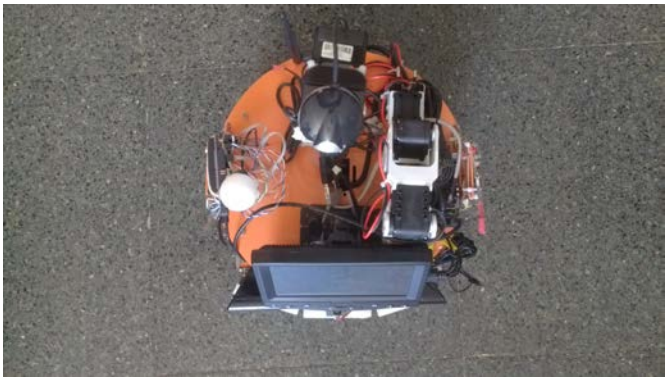


Ilustración 3. Planta del robot.

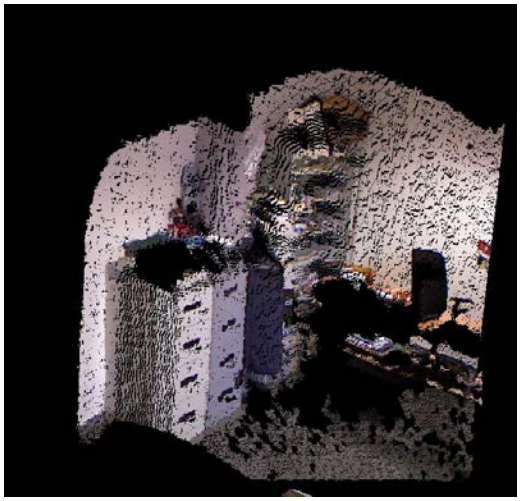


Ilustración 4. Reconstrucción en 3D con Kinect.

El futuro del robot pasa por desarrollar algoritmos de auto guiado basados en imágenes *2D* o gracias a la información *3D* que la cámara *Kinect* pueda devolver, reconstruyendo las imágenes en un mapa *3D*, como se aprecia en la ilustración 4.

Servidor

El servidor del robot está embebido al propio robot, y es la parte del proyecto con la que realmente la aplicación se comunica. Éste es el encargado de interpretar las órdenes y dirigirlas al *hardware* del robot.

La comunicación es por comandos en *ASCII* a través de dos *Sockets*, uno de control y otro de imagen, con protocolo *TCP/IP*. Es por ello que el servidor entiende una serie de comandos exactamente, detallados a continuación.

Comandos del servidor

Comando move

Sintaxis: *move <type> [<value1> <value2>]*

Este comando se utiliza principalmente para mover el robot. Con el tipo podemos especificar qué o cómo lo queremos mover:

1. Tipo 1: *FREE*. El robot se mueve libremente. Necesita del valor 1 (*value1*) para concretar la velocidad de las dos ruedas (se moverá linealmente), o de ambos valores para concretar la velocidad de sus dos ruedas motoras.
2. Tipo 2: *STOP*. El robot frenará.
3. Tipo 3: *CAM_SPEED*. Velocidad de movimientos de la cámara *2D* principal. Valores de entre 0 y 100.
4. Tipo 4: *PAN*. Movimiento *PAN* (movimiento horizontal) de la cámara principal. Valores entre 0 y 1023.
5. Tipo 5: TILT. Movimiento *TILT* (movimiento vertical) de la cámara principal. Valores entre 0 y 1023.
6. Tipo 6: *CAM*. Requerirá los valores uno y dos, siendo el primero el *PAN* y el segundo el *TILT*. Aúna los comandos tipo 4 y 5 en uno.
7. Tipo 7: *CAM_MOVE*. Movimiento relativo de la cámara.

Comando help

Sintaxis: *help*

El servidor responde con una lista de comandos disponibles para su uso. Si se desea mayor información sobre cómo se usa un cuando en concreto, basta con escribir ese comando a secas. Por ejemplo, si al comando anterior *move* no escribimos nada más que *move*, devolverá una respuesta con el uso adecuado de dicho comando.

Comando bat

Sintaxis: *bat*

Devuelve toda la información referente a la batería. Una cadena de 3 valores, a discernir voltaje del sistema, amperaje (corriente) y voltaje de la batería. La diferencia entre voltajes es debida a que, como es evidente, en carga el voltaje de la batería es 0.

Un ejemplo de respuesta sería: *bat 14.5 2.5 14.7*

Comando info

Sintaxis: *info*

Devuelve toda la información de los sensores, en una cadena que concatena con espacios cada uno de los valores.

Un ejemplo de respuesta sería: *info S0 S1 S2 S3 C0 C1 C2 E D 0 A 0 C B* donde:

- S1, S2, S3 y S4 son los cuatro sensores que tiene el robot controlando el suelo, en la base del mismo. Operador lógico (0 o 1).
- C0 C1 C2 son los sensores de choque del robot, situados en la parte frontal. Operador lógico (0 o 1).
- A: Sensor frontal de distancia. Devuelve centímetros.
- B: Sensor derecho de distancia. Devuelve centímetros.
- C: Sensor trasero de distancia. Devuelve centímetros.
- D: Sensor izquierdo de distancia. Devuelve centímetros.
- E: Sensor de distancia del suelo. Devuelve centímetros.

Comando sound

Sintaxis: *sound <type> <volumen>*

Reproduce un sonido. Dependiendo del tipo, 1 o 2, reproducirá ¡Peligro, peligro, peligro! o ¡Atención, atención, atención!, respectivamente. Ambos sonidos son creados gracias al sistema sintetizador de voz Festival [5].

Comando task

Sintaxis: *task <task_number> <OFF/ON=0/1> <WINDOW OFF/ON=0/1>*

Task permite ejecutar tareas en el robot. Estas tareas son:

1. Tarea 0: "*Follow*". El robot se pondrá en modo persecución.
2. Tarea 1: "*Faces*". El robot se pondrá en modo detección de rostros.
3. Tarea 2: "*Navigation*". El robot entrará en modo estado de ánimo (vea comando *mood*).

El último de los *tokens* del comando es para especificar si se quiere hacer en modo ventana o no. En el monitor del robot se abrirá una ventana de seleccionarse con un uno. Posible para casos de depuración.

Comando mood

Sintaxis: *mood <type>*

Según el tipo, el robot estará en un estado de ánimo y otro, y consecuentemente, cambiará su funcionamiento.

1. Tipo 0: "*Classic*". Se mueve autónomamente sin golpearse.
2. Tipo 1: "*Explorer*". Se mueve autónomamente sin golpearse, intentando llegar lo más lejos posible siempre. Hará comprobaciones de vez en cuando de hacía dónde moverse para llegar lo más lejos posible.
3. Tipo 2: "*Curious*". De vez en cuando, el robot, como está en modo curioso, parará y buscará algo a lo que seguir, como por ejemplo una cara.
4. Tipo 3: "*Debug*". Con fines de depuración.

Comando root

Sintaxis: *root <password>*

El usuario del robot por líneas de comando, de poner bien la contraseña, se pondrá en modo *superusuario*. La diferencia entre un usuario y un *superusuario* radica directamente en el uso del robot. Un usuario no puede cambiar el estado del robot (moverlo o cambiar su comportamiento), mientras que un *superusuario* sí.

5. Análisis

Requisitos de la aplicación

Se enumeran a continuación los **requisitos funcionales** que la aplicación debe cumplir en su versión móvil (menos de 7 pulgadas de pantalla):

1. Al iniciarse, debe aparecer una *pantalla de conexión* donde el usuario pueda elegir a qué servidor de robot quiere conectarse.
2. La *pantalla de conexión* deberá dejar seleccionar al usuario si sólo va a controlarlo o si quiere controlarlo y visualizar las imágenes. La aplicación entonces gestionará las pertinentes conexiones en función de lo que el usuario seleccione.
3. La *pantalla de conexión* deberá rechazar toda solicitud de conexión cuando no se disponga de conectividad a internet suficientemente estable y rápida (*vía Wifi o 3G* mínimo), o cuando el servidor sea inalcanzable.
4. Una vez conectado, se mostrará al usuario un menú cual mando a distancia, donde podrá elegir entre las opciones: *Controlar, Ver imágenes, Ver sensores, Crear y ejecutar órdenes, Consola de comandos* y una ventana de *Preferencias*.
5. En toda navegación en la aplicación, una vez conectados al servidor, será preciso que se muestre el *nivel de la batería* del robot.
6. En todo momento, también se podrá *desconectar* del servidor vía botón.
7. La aplicación deberá guardar registro de todos los eventos en su propio *fichero de registros*.
8. En la pantalla de *Preferencias*, el usuario podrá elegir el modo de uso de los controles del robot; a saber, *dos controladores tipo sliders* (uno de avance/retroceso y otro de izquierda/derecha), o un *joystick*.
9. En *Preferencias*, también podrá el usuario leer el *fichero de registros*, enviarlo directamente al desarrollador de la aplicación, o borrarlo.
10. Por último, en *Preferencias*, el usuario podrá activar la *auto conexión* (para evitar tener que poner el servidor y el puerto en la pantalla de conexión), desactivar la *introducción de bienvenida*, o activar las *notificaciones* de los sensores.
11. Las *notificaciones* de los sensores avisarán al usuario, en cualquier pantalla, si el robot está acercándose demasiado por alguno de los sensores, para evitar cualquier posible golpe. Dicha notificación, visual, vendrá acompañada de una notificación en modo de vibración.
12. La pantalla de *Control* del robot deberá posibilitar al usuario, de la manera más cómoda, manejar al robot. Además, deberá mostrar imágenes del robot.
13. La pantalla de *ver imágenes* deberá mostrar las imágenes del robot, así como permitir elegir la cámara del mismo.
14. La pantalla de *Crear y ejecutar órdenes* deberá permitir que el usuario pueda agregar sus acciones favoritas (comandos), y ejecutarlos desde ahí, además de borrarlos. Dichas órdenes deben ser locales al servidor. Si el usuario se conecta a otro robot, debe mostrar las órdenes del otro robot.
15. La pantalla de *consola* permitirá al usuario comunicarse con el servidor vía línea de comandos, recibiendo las pertinentes respuestas.

Por otra parte, la aplicación también tiene una serie de requisitos no funcionales, que son:

1. El dispositivo debe tener una versión de *Android* superior o igual a la 4.0 Ice *Cream Sandwich*.
2. Para una óptima visualización, se aconseja una pantalla de 4 pulgadas o más.
3. El dispositivo deberá tener un procesador de más de *750Mhz*.
4. El dispositivo deberá tener una memoria *RAM* de *512MB* o superior.
5. El dispositivo deberá tener conexión estable a internet vía *WIFI*. Si es conectividad móvil vía satélite, *3G* o superior
6. La aplicación deberá contar con dos versiones: una versión móvil y una versión Tablet. La versión *Tablet* (*Alive Client HD*), será adaptada para dispositivos de 7 pulgadas o más con una interfaz más simple.

Casos de uso

A continuación se detallan los diagramas de casos de uso de la aplicación, que detallan qué puede hacer el usuario en cada instante haciendo uso de la aplicación. Además, se muestra para cada caso de uso una tabla con la información relativa, para mejor comprensión.

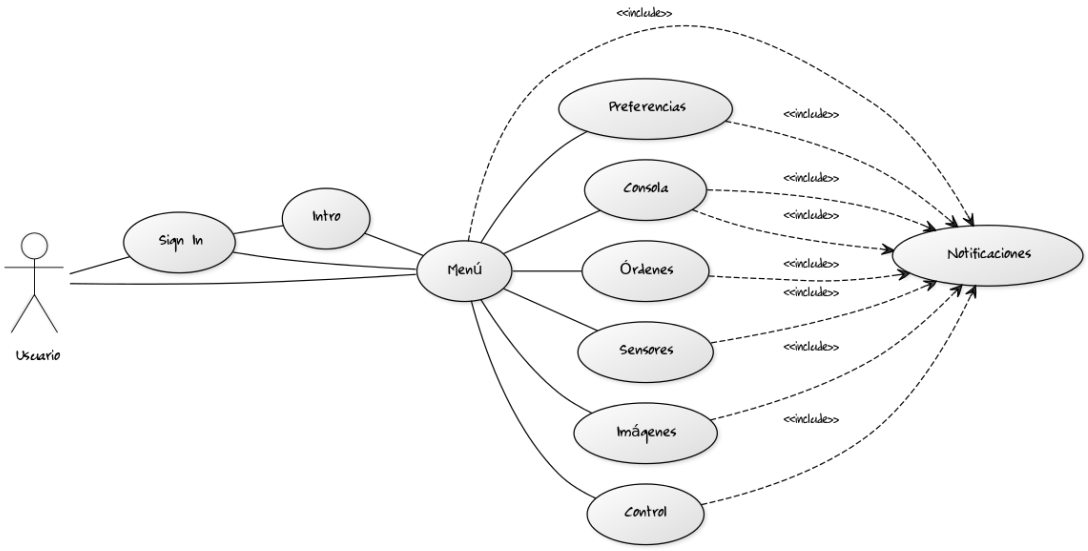


Ilustración 3. Casos de uso de la aplicación

Nombre	**Sign in**
Descripción	Pantalla de conexión a servidor donde se le pregunta al usuario sobre el nombre o dirección del servidor, puerto de escucha, y si desea manejarlo y/o ver imágenes
Actores	Usuario
Pre-condición	Abrir la aplicación y tener la opción de auto conectar desactivada
Post-condición	El usuario se conecta
Flujo	1. El usuario inicia la aplicación 2. El usuario introduce los datos solicitados del servidor 3. El usuario se conecta, o bien, se le advierte de que no se puede por: a. No tiene conexión a internet b. El servidor no responde

Tabla 1

Nombre	**Intro**
Descripción	Animación de introducción a la aplicación
Actores	Usuario
Pre-condición	Conectarse a un servidor y tener la opción de saltar intro deshabilitada
Post-condición	El cliente podrá acceder al menú
Flujo	1. El usuario cumplimentó los datos de conexión 2. El usuario visualiza la animación con sonido incluido 3. El usuario visualiza el nombre de la aplicación: *Alive Client* 4. El usuario pulsa la pantalla para proseguir

Tabla 2

Nombre	**Menú**
Descripción	Pantalla de menú tipo *dashboard* en la que el usuario puede acceder a donde desee
Actores	Usuario
Pre-condición	Haberse conectado a un servidor operativo
Post-condición	El cliente podrá operar a su gusto
Flujo	1. El cliente se conectó a un servidor operativo 2. Visualizó (o no) la introducción

Tabla 3

Nombre	**Notificaciones**
Descripción	El usuario recibe notificaciones sobre los sensores de distancia, compás y batería
Actores	-
Pre-condición	Haberse conectado a un servidor operativo
Post-condición	El usuario es alertado o bien del nivel de batería, o bien de la distancia de algunos de los sensores
Flujo	1. El usuario se conectó a un servidor operativo

Tabla 4

Nombre	**Control**
Descripción	Ventana de Control del Robot
Actores	Usuario
Pre-condición	Haber seleccionado en menú el botón de Control
Post-condición	Se abre la ventana de Control del robot
Flujo	1. El usuario se conecta a un servidor operativo 2. El usuario pulsa el botón de Control 3. El usuario puede manejar el robot según sliders o joystick 4. La aplicación comunica el movimiento al servidor

Tabla 5

Nombre	**Imágenes**
Descripción	Ventana de visualización de imágenes recibidas del robot
Actores	Usuario
Pre-condición	Haber seleccionado en menú el botón de Imágenes
Post-condición	Se abre la ventana de Imágenes del robot
Flujo	1. El usuario se conecta a un servidor operativo 2. El usuario pulsa el botón de Imágenes en el menú 3. El usuario puede visualizar las imágenes

Tabla 6

Nombre	**Sensores**
Descripción	Ventana de visualización de sensores del robot
Actores	-
Pre-condición	Haber seleccionado en menú el botón de Sensores
Post-condición	Se abre ventana de Sensores del robot
Flujo	1. El usuario se conecta a un servidor operativo 2. El usuario pulsa el botón de Sensores en el menú 3. El usuario puede visualizar los valores de los sensores

Tabla 7

Nombre	**Órdenes**
Descripción	Ventana de órdenes/macros del robot
Actores	Usuario
Pre-condición	Haber seleccionado en menú el botón de Órdenes
Post-condición	Se abre ventana de Órdenes del robot
Flujo	1. El usuario se conecta a un servidor operativo 2. El usuario pulsa el botón de Órdenes en el menú 3. El usuario puede añadir/borrar/ejecutar órdenes

Tabla 8

Nombre	**Consola**
Descripción	Ventana de consola de comandos del robot
Actores	Usuario
Pre-condición	Haber seleccionado en menú el botón Consola
Post-condición	Se abre ventana de Consola de comandos del robot
Flujo	1. El usuario se conecta a un servidor operativo 2. El usuario pulsa el botón de Consola en el menú 3. El usuario puede comunicarse por línea de comandos con el servidor y recibir respuesta

Tabla 9

Nombre	**Preferencias**
Descripción	Ventana de Preferencias (Ajustes) del robot/servidor
Actores	Usuario
Pre-condición	Haber seleccionado en menú el botón Preferencias
Post-condición	Se abre la ventana de Preferencias del robot
Flujo	1. El usuario se conecta a un servidor operativo 2. El usuario pulsa el botón de Preferencias en el menú 3. El sistema carga las preferencias ya guardadas 4. El usuario puede modificar las preferencias y leer archivo de log

Tabla 10

El usuario podrá navegar por la aplicación hasta el menú de 3 formas distintas:

1. Iniciando la aplicación, conectándose y viendo la introducción
2. Iniciando la aplicación, conectándose y sin ver la introducción *(saltar intro)*
3. Iniciando la aplicación *(auto conectar y saltar intro)*

Una vez en el menú, puede acceder a las distintas pantallas directamente.

Pantalla Control

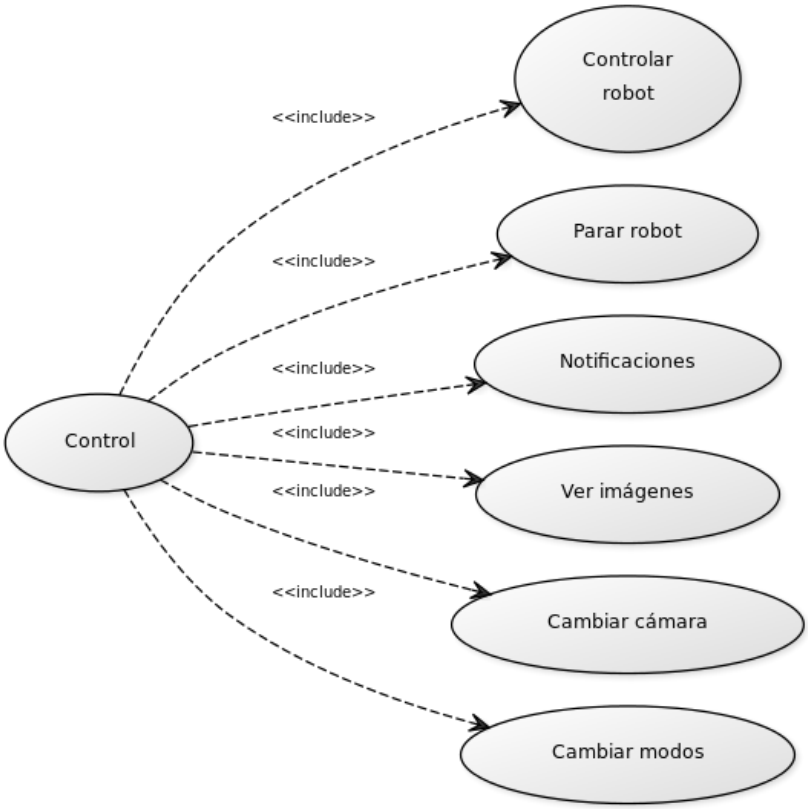


Ilustración 4. Detalle de casos de uso de Control.

Nombre	**Controlar robot**
Descripción	Joystick o sliders que sirven para controlar el robot
Actores	Usuario
Pre-condición	Haber abierto la ventana de Control
Post-condición	El robot se moverá según lo deseado
Flujo	1. El usuario accede a ventana de Control 2. El usuario interactúa con los controles del robot 3. El sistema envía la traducción del movimiento al servidor del robot 4. El robot se moverá

Tabla 11

Nombre	**Parar robot**
Descripción	Botón de parada total del robot
Actores	Usuario
Pre-condición	Haber abierto la ventana de Control
Post-condición	El robot frenará en seco
Flujo	1. El usuario accede a ventana de Control 2. El usuario pulsa el botón de Parada 3. El sistema informa al servidor del robot sobre el stop 4. El robot se parará

Tabla 12

Nombre	**Ver imágenes**
Descripción	Visualizador de imágenes del robot
Actores	Usuario
Pre-condición	Haber abierto la ventana de Control / Imágenes
Post-condición	Se visualiza las imágenes recibidas del robot
Flujo	1. El usuario accede a ventana de Control / Imágenes 2. El sistema refresca la pantalla con la imagen 3. El usuario puede visualizar la imagen

Tabla 13

Nombre	**Cambiar cámara**
Descripción	Botonera adjunta al visualizador de imágenes para poder cambiar la vista
Actores	Usuario
Pre-condición	Haber abierto la ventana de Control / Imágenes
Post-condición	Se visualiza una nueva imagen de una nueva cámara
Flujo	1. El usuario accede a ventana de Control / Imágenes 2. El usuario pulsa una nueva cámara 3. El usuario puede visualizar la nueva imagen

Tabla 14

Nombre	**Cambiar modos**
Descripción	Botonera que permite al usuario habilitar y deshabilitar los modos: 1. Modo dock: Uso o desuso del sensor de suelo. Con fines de depuración. 2. Modo log: Dejar rastro en el servidor 3. Modo HD: Imágenes de más resolución (320x240 o 640x480)
Actores	Usuario
Pre-condición	Haber abierto la ventana de Control
Post-condición	Se modifica el modo pulsado
Flujo	1. El usuario accede a ventana de Control 2. El usuario pulsa un modo, cambiando su estado 3. El sistema envía la orden al servidor, haciendo efectivo el cambio

Tabla 15

Pantalla Imágenes

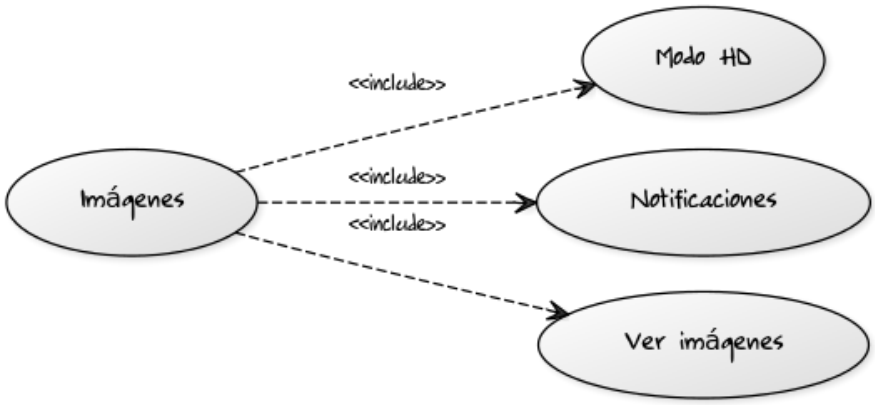


Ilustración 5. Detalle de casos de uso de Imágenes.

Véanse tablas anteriores 15, 4 y 13 respectivamente.

Pantalla Sensores

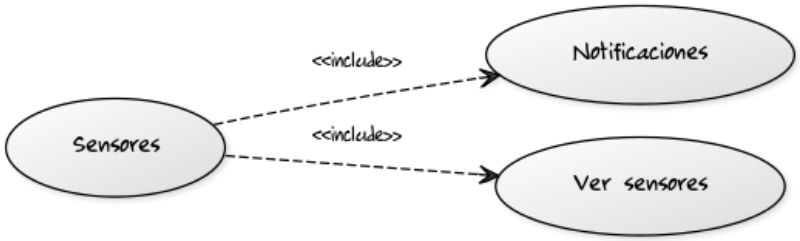


Ilustración 6. Detalle de casos de uso de Sensores.

Nombre	**Ver sensores**
Descripción	Visualizador de sensores del robot
Actores	Usuario
Pre-condición	Haber abierto la ventana de Control / Imágenes
Post-condición	Se visualiza los valores recibidos de los sensores del robot
Flujo	1. El usuario accede a ventana de Sensores 2. El sistema refresca la pantalla con el valor de los sensores

Tabla 16

Pantalla Órdenes

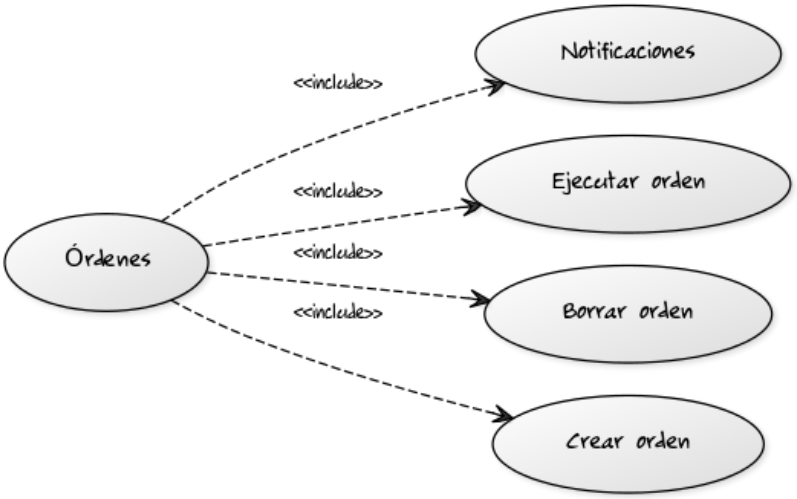


Ilustración 7. Detalle de casos de uso de Órdenes.

Nombre	**Ejecutar orden**
Descripción	Se manda al servidor la orden seleccionada para ejecutar
Actores	Usuario
Pre-condición	Haber abierto la ventana Órdenes y seleccionado una orden
Post-condición	El robot realiza la acción deseada
Flujo	1. El usuario accede a ventana de Órdenes 2. Selecciona una orden 3. Envía la orden a ejecutar mediante botón

Tabla 17

Nombre	**Borrar orden**
Descripción	Se borra de la BD del robot la orden seleccionada
Actores	Usuario
Pre-condición	Haber abierto la ventana Órdenes y seleccionado una orden
Post-condición	La BD se actualiza con una orden menos
Flujo	1. El usuario accede a ventana Órdenes 2. Selecciona una orden 3. Borra de la BD la orden mediante pulsación de botón

Tabla 18

Nombre	**Crear orden**
Descripción	Se añade una nueva entrada con una orden a la BD del robot
Actores	Usuario
Pre-condición	Haber abierto la ventana Órdenes y cumplimentado el formulario pertinente
Post-condición	La BD se actualiza con una orden nueva
Flujo	1. El usuario accede a ventana Órdenes 2. El usuario pulsa el botón de añadir orden 3. El sistema pregunta por los datos mediante pop-up 4. El usuario cumplimenta el comando de la orden 5. El usuario, opcionalmente, la describe 6. Pulsa aceptar 7. La BD se actualiza con la orden añadida

Tabla 19

Pantalla Consola

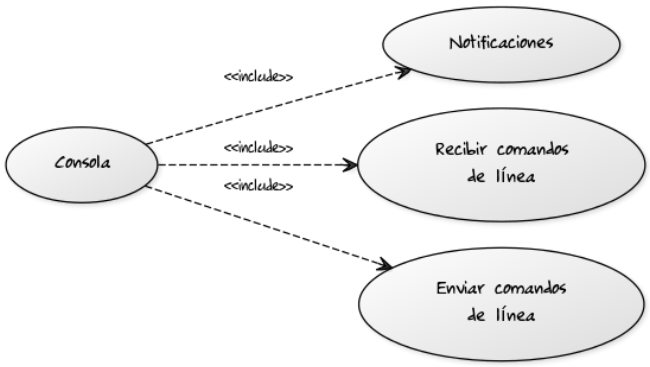


Ilustración 8. Detalle de casos de uso de Consola.

Nombre	**Enviar comandos de línea**
Descripción	Se envía un comando a la consola del servidor que controla el robot
Actores	Usuario
Pre-condición	Haber abierto la ventana de Consola y haber enviado una orden
Post-condición	El servidor recibe el comando
Flujo	1. El usuario accede a ventana de Consola 2. El usuario escribe un comando a enviar 3. El usuario pulsa el botón de enviar 4. El sistema muestra en consola la respuesta del servidor
Flujo alternativo	4. No se muestra respuesta, dado que la orden no tiene respuesta alguna

Tabla 20

6. Diseño

Diagrama de secuencia

El usuario interactúa con la más alta capa de la aplicación, es decir, la interfaz gráfica. No obstante, para la comunicación mediante internet, *Android* precisa de un hilo en segundo plano. Si esta conexión no se tratase en segundo plano, podría resultar bloqueante, y en tal caso, la aplicación no respondería al usuario. Además, transcurrido un tiempo, la aplicación se detendría con la excepción *NetworkOnMainThreadException*.

Por ello, se ha tenido que idear una forma de poder comunicarnos con el hilo secundario de tal manera que responda a las acciones del usuario enviando órdenes al servidor, y recibiéndolas en caso que se precise.

El siguiente diagrama de secuencia muestra una ejecución de la ventana *Control*, en la que el usuario además, mueve el *joystick* y acto seguido pulsa el botón de parada.

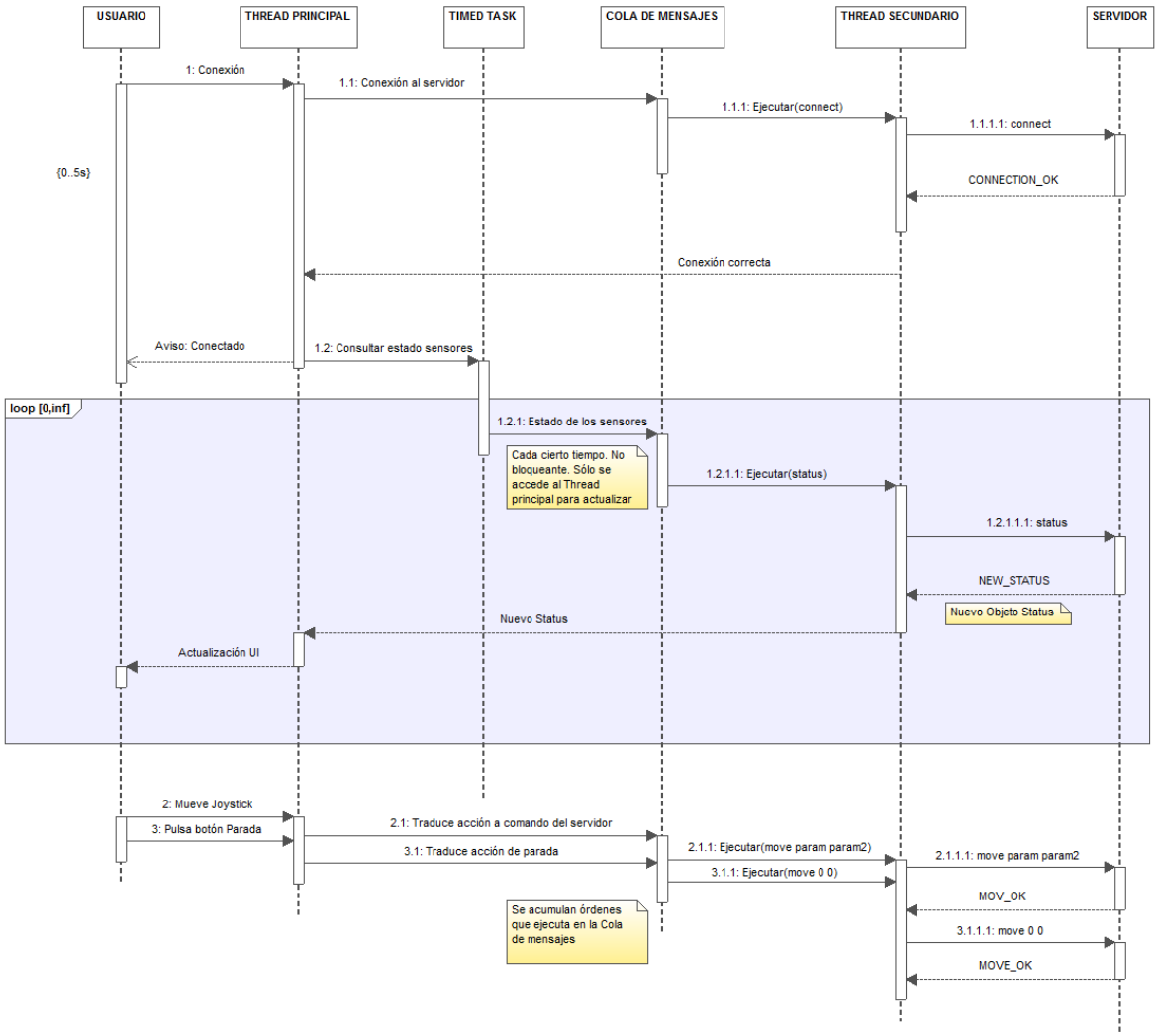


Ilustración 9. Diagrama de secuencia de la lógica de la aplicación.

En él (ilustración 11), se puede observar la ejecución típica de la aplicación. El usuario entra en la ventana. La aplicación se conecta al servidor. Para ello, crea una cola de mensajes y acto seguido un *thread* que se comunicará con el servidor.

A partir de entonces, el proceso principal, responsable de la interactuación usuario-aplicación, irá llenando la cola de mensajes con las órdenes del usuario, además de que, cada varios segundos, agregará a la cola por sí mismo (mediante la *timed task*) una orden de comprobar el estado de los sensores.

El proceso secundario, a su vez, irá buscando qué hacer en la cola de mensajes. Como su nombre indica, es una cola. Una cola *FIFO*, *First-In First-Out*. Cada vez que encuentre en la cola mensajes, los procesará, enviándolos al servidor.

En el apartado de implementación se detalla este proceso, además de justificarse, dado que existía más de una posibilidad a la hora de enfrentarnos a este problema de comunicación síncrona y asíncrona a la vez (mensajes de estado de sensores y acciones del usuario respectivamente).

Interfaz Gráfica

La interfaz gráfica ha sido, en un primer momento, pensada para su aplicación en móviles de 4 pulgadas. Gracias a los *mockups*, se fue ideando ventana a ventana cómo debía ser. Como tal, fue utilizada a modo de evaluación entre un grupo reducido de personas.

De dicha evaluación se sustrajo una idea muy importante: la aplicación controla un robot. Un robot puede resultar peligroso, y por ende, la interfaz tenía que ser sencilla y rápida. *¿Dónde tenemos un ejemplo de sencillez y rapidez?* En los mandos a distancia. Aunque normalmente no sirven para utilizar objetos que puedan dañar, son un ejemplo a seguir dado lo concisos que resultan.

De cara al final del proyecto, se apreciaba que la aplicación en versión tableta necesitaba remodelarse. El sistema de navegación entre ventanas resultaba tedioso en una pantalla tan grande. Así pues, la versión tableta (*Alive Client HD*), para tabletas de 7 pulgadas o más, dispone de otra interfaz gráfica.

Mockups de la Interfaz Gráfica

El diseño de la aplicación pasó por un proceso de prototipado a través de la herramienta Pencil.

Mockup de conexión

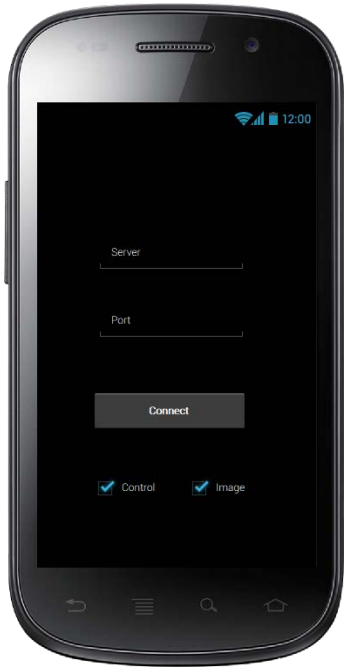


Ilustración 10

La interfaz gráfica de la ventana de conexión debía ser inmediata: no hay nada más directo para el usuario que el simple formulario de servidor y puerto donde conectarse, seguido de un botón de conectar.

Contiene:

2 EditText

1 Button

2 ToggleButton

Mockup de menú

Ilustración 11

En este caso, el menú debía simular el mando a distancia de una televisión. El motivo, ya comentado anteriormente, no es más que la inmediatez que ello otorga. Este tipo de menú es del tipo *Dashboard*, y posibilita el acceso a cualquier pantalla de la aplicación.

Contiene:

6 Button

Mockup de Control

Ilustración 12

En la pantalla de control, al usuario se le da el poder de pilotar el robot. En un primer momento, la idea principal fue la de seguir emulando un mando, con controles tipo cruceta, como los de las primeras generaciones de consolas.

Contiene:

1 ImageView

4 TextView

5 Botones

Mockup de Imágenes

Ilustración 13

La imagen debe ocupar la mayor parte de la pantalla, dado que es el objetivo de la misma. Además, debe poseer los controles con los que el usuario pueda cambiar de cámaras.

Contiene:

1 ImageView

1 TextView

6 Button

Mockup de Sensores

Ilustración 14

Con una representación de la planta del robot, esta pantalla debe mostrar cada uno de los sensores de distancia con sus respectivos valores.

Además, en la parte superior, más visible para los usuarios que la parte inferior, deberá mostrar los datos del compás y la batería.

Contiene:

1 ImageView

9 TextEdit

Mockup de Órdenes

Ilustración 15

Pantalla capaz de mostrar al usuario las órdenes almacenadas gracias a una lista que se actualiza con la base de datos. Además, contiene botonera para operar con ella: añadir, borrar, ayuda (descripción) y ejecutar.

La lista se mantiene en la parte central de la pantalla para otorgarle de mayor notoriedad. Como siempre, los controles disponibles en la parte baja.

Contiene:

1 LISTA

2 TextView

4 Button

Mockup de Consola

Sencilla pantalla que emula una consola de comandos, la cual posee una línea de entrada en la parte inferior y un botón de envío, que actualizan la salida de la consola, parte principal de la pantalla que ocupa el 90% del espacio de pantalla.

Contiene:

1 TextView (Consola)

1 EditText (Entrada de datos)

1 Button (Envío de comando)

Ilustración 16

Mockup de Alive Client HD

La aplicación para tabletas debe cumplir el requisito de ser capaz de aprovechar toda la pantalla para poder tener todos los controles a mano. Resultaba un reto.

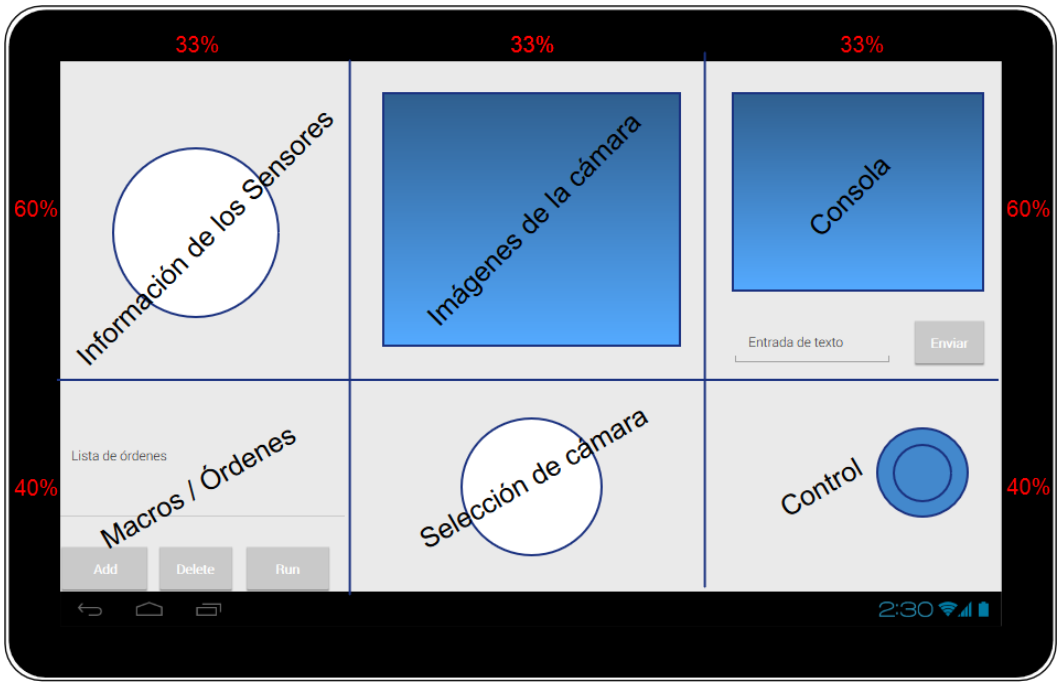


Ilustración 17

En el centro de la ventana se dispone, en la parte superior (es justo la parte de la Tablet que más se mira indirectamente), las imágenes del robot. Abajo a la izquierda tenemos los controles de las macros, accesibles con el pulgar izquierdo, y abajo a la derecha, podremos controlar el robot con nuestro pulgar derecho.

Como se puede observar en el *mockup*, se ha jugado con la posición y los tamaños para dotar de importancia a lo que realmente lo tiene. De tal modo, las 3 ventanas superiores tienen más peso que las 3 inferiores, y las laterales inferiores están pensadas para su interacción inmediata.

El diseño de la aplicación en su versión *HD* tiene la única limitación de tener que controlarse mediante *joystick*.

Consideraciones varias de diseño

El diseño de una aplicación no se resume en *mockups*. Una aplicación contiene una serie de elementos estéticos capaces de atraer y agradar al usuario. Tanto es así, que *Google* tiene en la página de desarrollo de *Android*, un gran apartado dedicado a tal propósito.

Colores

Haciendo uso de la guía de estilo de *Android* se ha considerado que la aplicación, para ser consistente con el sistema operativo, ha de interactuar con el usuario mostrando unos colores afines al sistema operativo, además de ser resaltados los contenidos con colores con gran contraste.

Ilustración 18. Color principal de la aplicación.

Ilustración 19. Colores secundarios de la aplicación junto con el color principal.

En las figuras anteriores, se muestra el color principal de la aplicación, el verde oscuro con código *RGB #669900*, junto con los colores contraste escogidos: el blanco, el negro, y el naranja oscuro de código *#ff8800*.

Fuente

Además de los colores, la tipografía también llama la atención. Con una fuente como *Roboto*, recomendada por *Google* para la creación de aplicaciones [6], se consigue una limpieza que facilita al usuario la lectura, y crea en él una sensación comodidad visual.

Ilustración 20. Tipografía utilizada en la aplicación.

Botones

Los botones deben destacar, para que el usuario sepa dónde pulsar. Debe ser auto-descriptivo de tal manera que el usuario, al verlo, sepa que puede o tiene que pulsarlo.

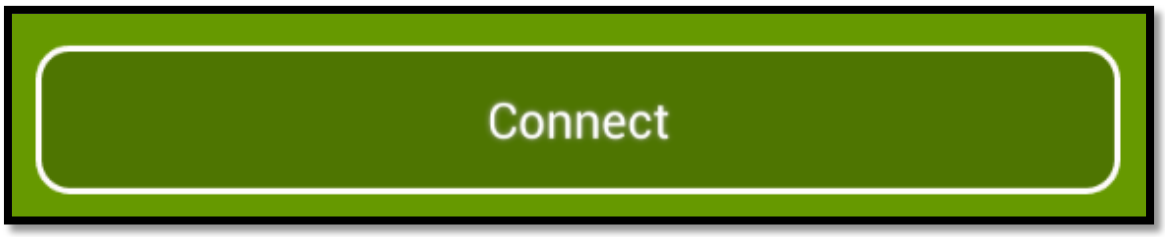


Ilustración 21. Botón ejemplo de la aplicación.

Se ha creado un estilo en *XML* con el que se bordea el botón con blanco para resaltarlo, y se redondea los bordes para hacerlo amigable a la vista. Además, se le añade un verde un poco más oscuro para que contraste más aún con el fondo si cabe.

Iconografía

Logotipo

Uno de los puntos fuertes de toda aplicación es la iconografía. En primer lugar, el logotipo de la aplicación debe conseguir destacar entre la multitud, véase en la Play Store o entre todas las aplicaciones que uno pueda tener instaladas.

Con el logotipo de *Alive Client* se ha jugado un poco con lo que es: una aplicación *Android* para el control de robots. La fórmula empleada es sencilla. Se tiene que describir las tres palabras clave de la breve descripción: *android*, control y robot.

Podríamos entonces crear un logotipo con 3 objetos que pudieran identificar cada una de las palabras. O bien, y tal y como se ha hecho, considerar que el robot de *Android* es suficientemente descriptivo. Si le añadimos también un mando de consola, conseguimos el efecto que justo queríamos, y en poco espacio, idóneo para ser impactante por simpleza y no un entresijo de dibujos.

Ilustración 22. Logotipo Alive Client.

Pero no sólo queda ahí la creación. Haciendo caso de los cánones de creación de iconos de aplicaciones *android* [7], se le ha añadido:

1. Color verde de android, descriptivo del robot
2. Iluminación clara, para resaltar
3. Bordeado de color negro, para fortalecer el icono
4. Fondo invisible. Actualmente los iconos ya han dejado de ser cuadrados para tener cada uno su contorno. Otro elemento descriptivo. *¿Quién no identifica el bocadillo de Whatsapp?*

Menú

Para la pantalla principal del menú, unos iconos muy representativos de las 6 opciones que se tienen a disposición. Un volante para la pantalla de pilotaje, una cámara para la de ver imágenes. Unas ondas alrededor de un ítem, que bien puede representar un sensor. Para las órdenes, una lista, y para la consola, el famoso símbolo del *cmd*. Por último, el ya conocido recurso del engranaje para las preferencias.

Ilustración 23. Iconos del menu principal.

Nivel de batería

La aplicación controla en todo momento el nivel de batería del robot que se está controlando. Es por ello que el usuario debe recibir en todo momento información de ello, dado que es un dato crítico: es posible que se quede sin batería en mitad de una acción, y puede tener consecuencias negativas.

Por tanto, la aplicación está equipada con una *Action Bar* que le facilita al usuario un icono de una pila donde pulsar. Dicho icono varía según el nivel de batería del robot, calculado a partir de su sensor de voltaje.

Ilustración 24. Diferentes estados de la batería.

Dependiendo del valor de voltaje que reciba la aplicación, y ponderándola con su valor máximo, obtiene un porcentaje, el cual utiliza para mostrar un icono u otro. De los mostrados en la ilustración 26, de izquierda a derecha: mayor del 75%, entre 75% y 50%, entre 50% y 25%, entre 25% y 12% y menor del 12% (rojo alerta).

Otros iconos

Para el resto de iconos, con diversos propósitos, se hace uso de los que Google proporciona para la *Action Bar*, con temática *Holo Dark*. Es decir, iconos comunes al resto de aplicaciones en su mayoría, con lo cual el usuario ya es afín a su significado y, además, blancos, concordantes con los colores de la aplicación, como se puede apreciar en la ilustración 27.

Ilustración 25. Iconos de Google utilizados.

De izquierda a derecha: desconectar, eliminar, enviar email, añadir, anterior, siguiente, enviar, stop y alerta.

Diálogos

En tanto en cuanto a los diálogos, la aplicación contiene de dos tipos: el primero, sólo de aviso, que se muestra cuando está realizando la conexión. El segundo, tipo formulario en el que pregunta al usuario sobre la seguridad de realizar algunas acciones como la de desconectar o borrar, o en otro caso, pregunta al usuario qué añadir a la nueva orden.

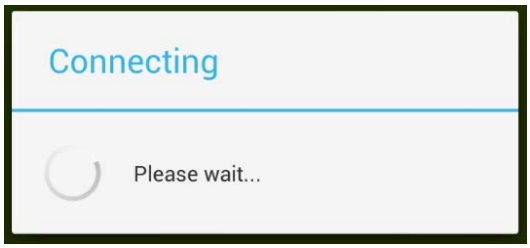


Ilustración 27. Mensaje de conexión.

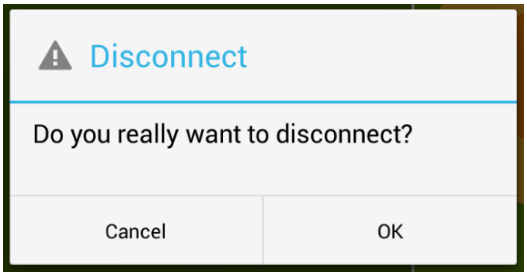


Ilustración 26. Diálogo de desconexión.

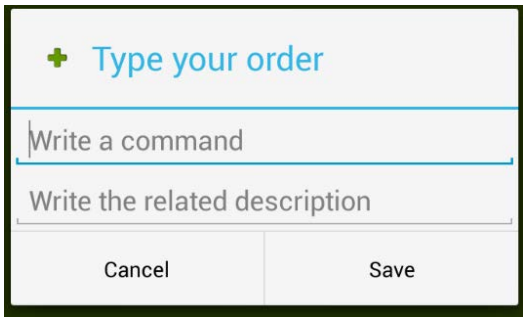


Ilustración 29. Diálogo añadiendo una macro.

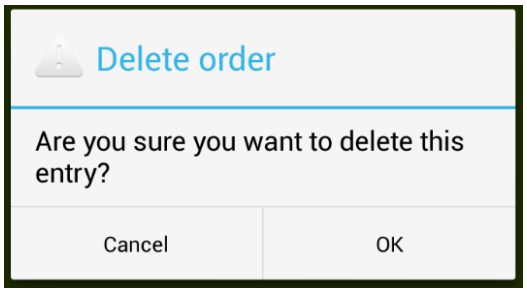


Ilustración 28. Diálogo borrando una orden.

Interfaz gráfica resultante

La suma de los *mockups* junto con las consideraciones de estilo comentadas anteriormente sobre colores, botones, iconografía ha resultado en una interfaz gráfica limpia y coherente, afín al sistema operativo y a la herramienta a la cual está conectada. Amigable y fácil de entender.

Ilustración 32. Login.

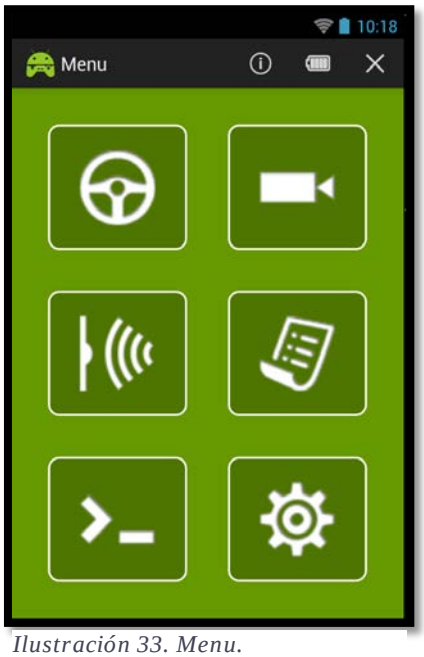


Ilustración 33. Menu.

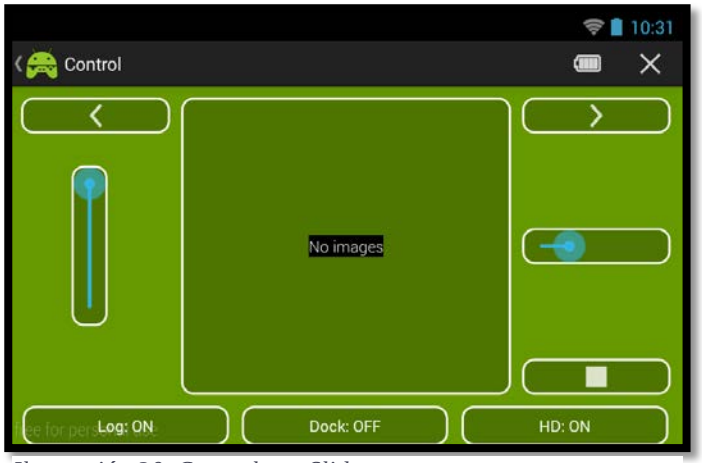


Ilustración 30. Control por Slider.

Ilustración 31. Control por Joystick.

Ilustración 32. Imágenes.

Ilustración 33. Sensores.

Ilustración 35. Órdenes.

Ilustración 34. Consola.

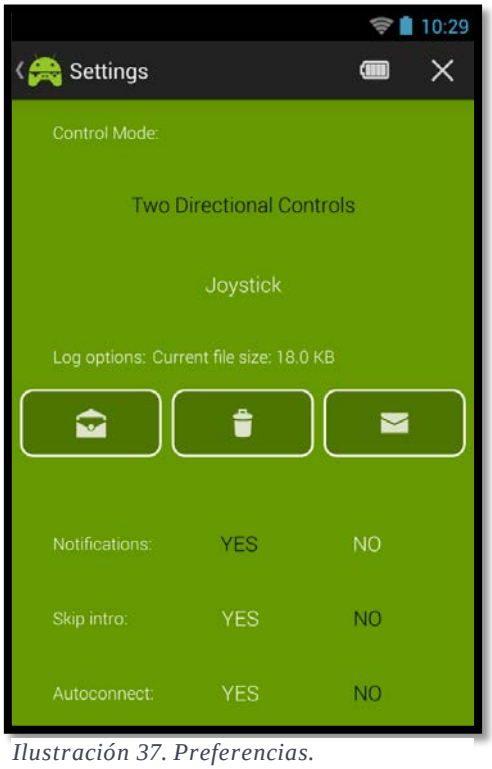


Ilustración 37. Preferencias.

Ilustración 36. Log.

Ilustración 38. Interfaz gráfica de Alive Client HD.

El mayor cambio respecto el prototipado inicial es la ventana de control. La primera idea, bajo la cual se diseñó, era un control mediante botones. Sin embargo, se vio que era un método arcaico y poco ortodoxo. Su renovación pasó en un primer momento por los dos sliders.

Sin embargo, seguía siendo complicada de manejar, y se optó por acostar la pantalla, hacerla horizontal. De este modo conseguimos que el usuario utilice justo los dos pulgares para el control, y tenga mucho más sentido. Ahora parecía un mando de consola.

Además, finalmente, se optó por añadirle otra opción de movimiento. Un *joystick* que facilita aún más la tarea de controlarlo. No obstante, se decidió dejar los dos tipos de controles dado que a cada persona le puede resultar más cómodo uno u otro.

El segundo cambio más notorio es el de la pantalla de órdenes. Se ha modificado casi por completo. Ahora no es un desplegable tal y como se pensó en el prototipado, sino que es una lista deslizable que ocupa la mayor parte de la pantalla. Además, la botonera se ha normalizado al espacio y ya no tiene discrepancias entre los tamaños; ahora son todos los botones iguales, dándole más coherencia visual.

Por último, la pantalla de Preferencias, la cual no fue diseñada en un primer momento dado que muchas opciones fueron añadidas en tiempo de programación. Sobre la pantalla de lectura de log, se ha considerado oportuno no visualizar la barra de estado del móvil para abstraer al usuario en la lectura.

7. Implementación

En este penúltimo apartado, se comentará cómo y dónde ha sido programado, al igual que algunas consideraciones y decisiones que se han tenido que tomar.

Entorno de desarrollo

Para el desarrollo de la aplicación se ha utilizado indistintamente dos ordenadores, uno portátil y otro de sobremesa.

	Sobremesa	*Portátil*
Procesador	i5-750 @ 4 x 2.67GHz	i7−2675QM @ 4 x 2.20GHz
Memoria RAM	8GB @ 1333MHz	8GB @ 1333MHz
S.O.	Windows 8 Professional	OS X Mavericks

Tabla 21

Además de dicho equipo de trabajo, se ha probado la aplicación con distintos dispositivos *Android*, tanto físicos como virtuales. La descripción de los físicos la podemos ver en la tabla 22.

	Samsung Galaxy S2	*Samsung Galaxy Tab 2*
Tipo	Móvil	Tableta
Procesador	Cortex-A9 @ 2 x 1.2GHz	Cortex-A9 @ 2 x 1GHz
Memoria RAM	1GB	1GB
Pantalla	4.3” @ 480x800	10.1” @ 800x1280
S.O.	Android 4.1.2	Android 4.2.2

Tabla 22

Por último, el tercer vértice del triángulo que compone el desarrollo de aplicaciones para móvil es el del software utilizado para tal propósito. En este caso, ha sido el descrito en la tabla 23.

	Versión	*Descripción*
Android Studio	0.5.9	Programación
Genymotion	2.2.2	Creación de unidades virtuales para emulación
Evolus Pencil	2.0.5	Creación de mockups
Photoshop	CS6	Creación de iconografía
yUML	Online	Creación de casos de uso
Altova UModel	2014 2-sp1	Creación de diagramas de secuencia

Tabla 23

Recursos de una aplicación

Los recursos que tenemos a mano en un proyecto de aplicación *android* son importantes de manejar y controlar.

En nuestro caso, destacan los siguientes, tanto para la versión móvil como la versión tableta:

- **anim**: ficheros *XML* que gestionan las animaciones del proyecto. Utilizados en su mayoría para la animación de introducción. También utilizado para las animaciones de transición entre pantallas.
- **drawable**: archivos formato *PNG* con la iconografía, además de contener los estilos de la botonera y selectores de estilo en *XML*, los cuales permiten cambiar la apariencia de los botones a la hora de ser presionados.
- **menu**: ficheros *XML* que definen las acciones disponibles en la *Action Bar*.
- **raw**: ficheros no *XML* de otro propósito. En nuestro caso, un archivo *mp3* para la introducción.
- **values**: contiene ficheros *XML* con valores constantes, tales como colores y cadenas de texto.
- **values-es**: utilizado para la traducción de la aplicación al castellano. Contiene las cadenas de texto en español.

Los recursos *drawable* han sido escalados para que tengan compatibilidad con todo dispositivo, aunque tengan distintas resoluciones y densidades de píxel.

Cabe destacar que la aplicación ha sido programada para el soporte multilenguaje, tal y como se especificaba en los objetivos como requerimiento. De tal modo, toda cadena de texto de la aplicación está referenciada a su correspondiente fichero de cadenas *res/values/strings.xml*, y la traducción a cualquier idioma no lleva más de 5 minutos, dado que no hay más que traducir un fichero *XML* y ponerlo en su correspondiente directorio *values-XX* (XX representa en *Android* [FUENTE]).

Además, la aplicación ha sido programada tanto para su versión móvil como para su versión tableta, y es por eso que nos encontramos unas diferencias entre ambas aplicaciones en los recursos referidos a la interfaz gráfica.

En la versión móvil encontramos los recursos **layout** y **layout-land**, los cuales contienen los XML que describen la interfaz gráfica para las pantallas tanto en formato *portrait* como en *landscape*. Sin embargo, en la versión Tableta, los recursos *layout* se resumen en cuatro concretamente: **layout-sw600dp**, **layout-sw600dp-land**, **layout-sw720dp** y **layout-sw720dp-land**. Las etiqueta *sw600dp* describe la interfaz para dispositivos de 7 pulgadas, mientras que la etiqueta *sw720dp* se utiliza para tabletas de 10 pulgadas o más.

AndroidManifest

AndroidManifest.xml es probablemente la *alma mater* de una aplicación Android. Contiene información tan útil como el nombre del paquete que contiene la aplicación, descripción de los componentes que componen la aplicación, procesos que habrá, permisos y describe el nivel mínimo de API bajo el cual la aplicación podrá ser instalada y ejecutada.

Componentes de la aplicación

La aplicación en su versión móvil consta de 11 actividades, a distinguir (tabla 24).

	Descripción	*Interfaz*
LoginActivity	Conexión al servidor	Ilus. 32
LoginAnimationActivity	Animación de introducción	n/d
MenuActivity	Menú principal tipo Dashboard	Ilus. 33
ControlActivity	Control con dos sliders del robot	Ilus. 34
SensorActivity	Gestión de sensores del robot	Ilus. 37
ConsoleActivity	Consola de comandos	Ilus. 38
OrderActivity	Macros/Órdenes del robot	Ilus. 39
ImagesActivity	Gestión de las cámaras del robot	Ilus. 36
SettingsActivity	Preferencias	Ilus. 41
ControlJoystickActivity	Control con Joystick del robot	Ilus. 35
LogActivity	Gestión del fichero de log	Ilus. 40

Tabla 24

En su versión tableta, por el contrario, resulta bastante más liviana en cuando a clases (tabla 25).

	Descripción	*Interfaz*
LoginActivity	Conexión al servidor	n/d
LoginAnimationActivity	Animación de introducción	n/d
MainActivity	Control y gestión íntegra del robot	Ilus. 42
SettingsActivity	Preferencias	n/d

Tabla 25

Permisos de la aplicación

Alive Client requiere autorización para operar con distintos elementos hardware del móvil. Son en total cuatro los permisos que tendrá la aplicación y que por tanto el usuario deberá aceptar a la hora de su instalación:

1. Permiso **INTERNET**: Necesario para que la aplicación se pueda comunicar por internet.
2. Permiso **ACCESS_NETWORK_STATE**: Utilizado para la comprobación de red móvil disponible.

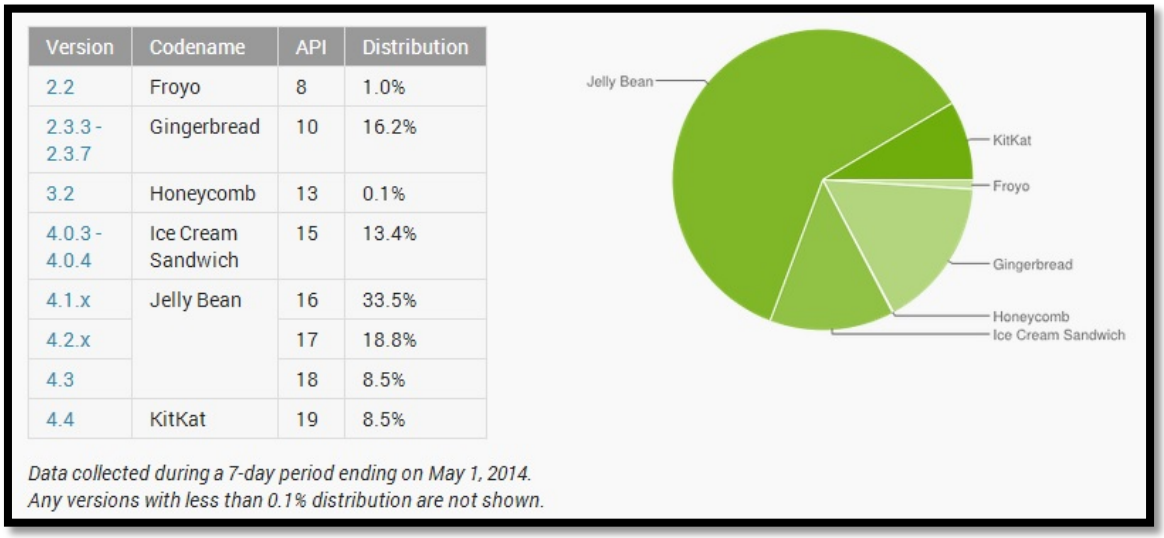
3. Permiso **ACCESS_WIFI_STATE**: Utilizado para la comprobación de conexión *WiFi* disponible.
4. Permiso **VIBRATE**: La aplicación podrá vibrar a modo de aviso si algún sensor está cercano a colisionar.

Nivel de API mínimo

La aplicación ha sido programada para soportar un nivel mínimo 14 de API. Esto quiere decir que se requiere de un móvil con *Android* 4.0 (*Ice Cream Sandwich*) instalado o versiones más recientes.

Aunque parezca que es una gran limitación, no lo es tanto. Parece que la versión 4.0 está muy cercana a la actual 4.4 *KitKat*. Sin embargo, y según datos de *Google*, ésta limitación afecta al 14.3% de dispositivos que se conectan a *Play Store*, y la evolución, evidentemente, es a disminuir ese porcentaje, dado que un móvil de gama media-baja actualmente ya tiene instalada, como mínimo una versión 4.2 de *Android* (véase *ZTE Blade C2*, de unos 70 euros).

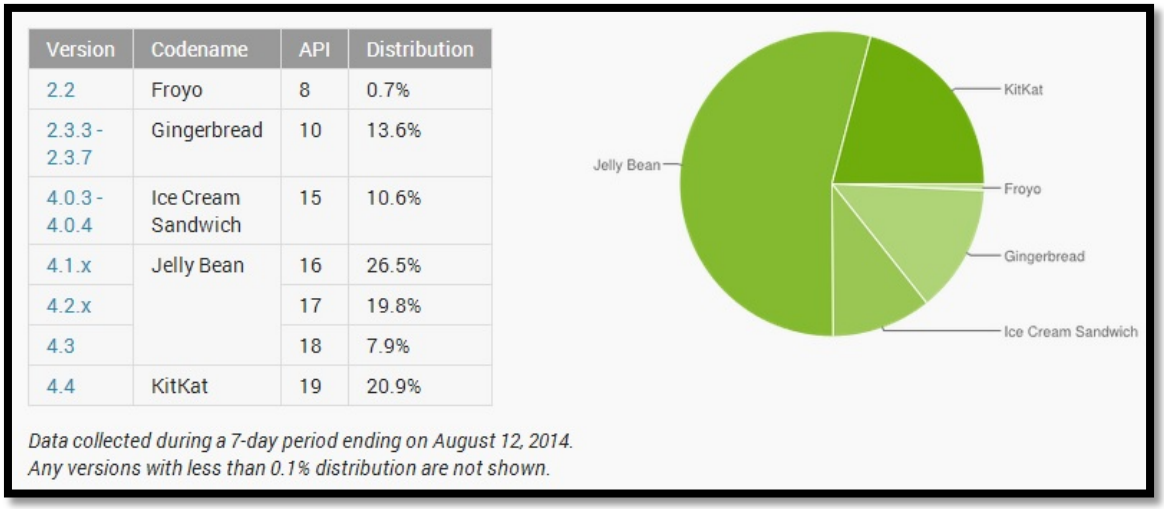
De hecho, desde mayo de 2014 a agosto de 2014, la compatibilidad con *Alive Client* se ha incrementado en un 3% de dispositivos (17.1% de incompatibles frente a 14.1%).



Version	Codename	API	Distribution
2.2	Froyo	8	1.0%
2.3.3 - 2.3.7	Gingerbread	10	16.2%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	13.4%
4.1.x	Jelly Bean	16	33.5%
4.2.x		17	18.8%
4.3		18	8.5%
4.4	KitKat	19	8.5%

Data collected during a 7-day period ending on May 1, 2014.
Any versions with less than 0.1% distribution are not shown.

Ilustración 39. Dispositivos que accedieron a Play Store a finales de Abril de 2014. Fuente: Android [8]



Version	Codename	API	Distribution
2.2	Froyo	8	0.7%
2.3.3 - 2.3.7	Gingerbread	10	13.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	10.6%
4.1.x	Jelly Bean	16	26.5%
4.2.x		17	19.8%
4.3		18	7.9%
4.4	KitKat	19	20.9%

Data collected during a 7-day period ending on August 12, 2014.
Any versions with less than 0.1% distribution are not shown.

Ilustración 40. Dispositivos que accedieron a Play Store a principios de Agosto 2014. Fuente: Android [8]

Actividades

El ciclo de vida ha sido controlado para cada actividad, de tal modo que no se pueda perder ningún dato ni pueda saltar excepción alguna. En tal caso, la excepción nunca repercutirá a la experiencia del usuario, sino que será registrada en el archivo de log para que el desarrollador pueda visualizarlo si el usuario desea remitírsela.

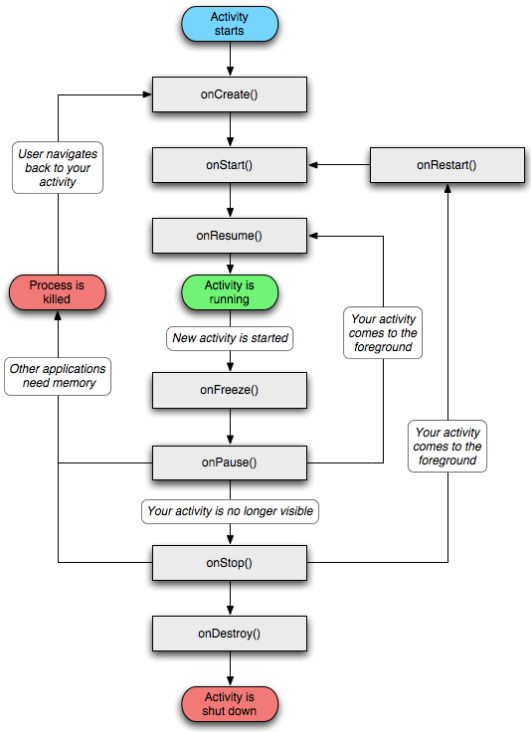


Ilustración 41. Ciclo de vida de las actividades Android. Fuente: androidutp [9]

En la gestión del ciclo de vida de las actividades, se han implementado **onCreate**, **onStart** y **onPause:**

- onCreate: inicializa la actividad con su respectiva interfaz (recurso *layout*).
- onStart:
    - Cierra el proceso en segundo plano si éste ya estaba inicializado.
    - Lee las preferencias de la aplicación.
    - Inicializa elementos y objetos de la interfaz.
    - Inicializa el proceso en segundo plano.
    - Inicializa el proceso de segundo plano *TimerTask* para actualización de sensores.
    - Inicializa Base de datos (si procede).
- onPause:
    - Cierra la base de datos si procede.
    - Guarda las preferencias si procede.
    - Cancela el proceso de actualización de status (*TimerTask*).
    - Termina y cancela el proceso en segundo plano.

Aunque esta gestión parezca un poco extrema, minimiza el uso de recursos dado que, a la mínima que el usuario no interactúe con la actividad, desconecta todos sus servicios. En el apartado de decisiones tomadas se comenta con más extensión.

Procesos en segundo plano

Alive Client se conecta, como bien se ha comentado con anterioridad, a un servidor vía Internet para poder comunicarse con un Robot. Debido a esta conexión, la aplicación necesita de un proceso en segundo plano que gestiona la conexión, para que la interfaz gráfica de la aplicación no resulte bloqueada por cualquier problema en la conexión.

Para tal fin se han ponderado numerosas fórmulas. *Android* ofrece multitud de posibilidades: *AsyncTask*, *Service*, *IntentService* y *Threads*.

En este apartado se repasa la programación y lógica del utilizado, el *Thread*. Posteriormente, en el apartado “Decisiones tomadas” se especifica el porqué de dicha decisión.

En tanto en cuanto al *Thread*, éste se ejecuta al inicio de la actividad, y se concluye cuando la actividad se pausa. Se podría decir, por ende, que su ciclo de vida está ligado directamente al de la actividad. Es de vital importancia ésta gestión dado que un proceso en segundo plano podría quedarse “deambulando” sin actividad, pero consumiendo recursos de procesador, memoria y sobretodo, batería.

El *Thread*, en adelante *ProcessThread*, es una clase declarada interna de cada actividad. Su constructor sólo posee el único parámetro que se le debe pasar, una *BlockingQueue*. Dicha *BlockingQueue*, en adelante *queue*, recibirá órdenes directamente desde la actividad y se irán encolando para ser procesadas por *ProcessThread*.

```
/* *
     * Constructor of Thread
     * @param queue queue of orders to send to server
     */
    public ProcessThread(BlockingQueue<String> queue) {
        this.queue = queue;
    }
```

Código 1. Constructor del ProcessThread.

Como todo *thread*, una vez ejecutado se inicia con su método *run*(). La lógica del método *run* es sencilla, y está bastante detallada gráficamente en el diagrama de secuencia (ilustración 8).

1. Se conecta al servidor remoto, inicializando los objetos de Entrada/Salida.
2. Busca en *queue* si hubiera alguna orden que ejecutar
    a. Si la tiene, la ejecuta.
        i. Cuando acaba de ejecutarla, vuelve a 2.
    b. Si no la tiene, vuelve a 2.

Mientras tanto, desde la actividad, *queue* se va actualizando. La actividad registra, en su *thread* principal, el que rige la interfaz gráfica, las órdenes del usuario y las va introduciendo en *queue*. Toda esa lógica, en la programación se resume en el fragmento de código 2 de la página siguiente.

La comunicación Actividad-*Thread* es a través de la *BlockingQueue*, pero *¿cómo se comunica del Thread a la Actividad?* La respuesta está en los *Handlers* y los *Messages*.

```
public void run() {

      int conn = connect(server, port, controlEnabled, imageEnabled);

      […]

      while(running) {

            […]

            while(!queue.isEmpty()) { //Comprueba la cola

                  String cmd = queue.take() + “”;

                  /* Procesar orden: cmd */

            }
```

Código 2. Código simple de ejecución de tarea en segundo plano.

Cuando el *ProcessThread* ha procesado una orden, recibe una respuesta (o no) del servidor, la cual ha de repercutir en la experiencia del usuario con la aplicación: se le ha de notificar.

El caso más representativo es el del mensaje de status (dada su reiteración), el cual obtiene el estado de los sensores del robot. Este mensaje es agregado a la cola de procesado cada 5 segundos por un proceso secundario temporizado (*TimerTask*).

Cuando recibe los datos, lo idóneo sería que el propio *ProcessThread* actualizase la interfaz. No obstante, esto no es posible, y está limitado por el sistema *Android*.

Para tal fin, se dispone de un *Handler* y un objeto *Message*, de tal manera que el *thread* principal pueda recibir mensajes desde el proceso en segundo plano. Cuando el proceso en segundo plano emite un mensaje, el *thread* principal lo gestiona, dado que se encontraba “escuchando” para encontrar mensajes. El código en cuestión, extendiendo al anterior del proceso *run*() se encuentra en el código 3.

```
public void run() {
	int conn = connect(server, port, controlEnabled, imageEnabled);
	[…]
	while(running) {
		[…]
		while(!queue.isEmpty()) { //Comprueba la cola
			String cmd = queue.take() + “”;
			If(cmd.equals(MSG_STATUS_CMD)) {
				/* Orden de actualizar sensores */
				/* status = variable estática de la clase */
				status = getSensorStatus();
				msg = handler.obtainMessage();
				bundle = new Bundle();
				/* Establece el tipo de mensaje */
				bundle.putInt("type", MSG_STATUS);
				msg.setData(bundle);
				/* Envía el mensaje */
				handler.sendMessage(msg);
			}
		}
	}
	disconnect();
	queue = null; //Garbage collector will erase
}
```

Código 3. Código del proceso en segundo plano extendido.

Por la otra parte, en la actividad (o proceso en primer plano), escucha de la forma que se detalla en el Código 4.

```
private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
                /* Recogiendo tipo del mensaje */
                Bundle bundle = msg.getData();
                int type = bundle.getInt("type");
                if(type == MSG_STATUS) {
                        /* Status ha sido actualizado */
                        /* Actualizamos el nivel de la batería */
                        batteryLevel = (status.voltage / MAX_VOLT)*100;
                        /* Se actualiza el icono de la batería de la action bar */
                        invalidateOptionsMenu();
                        // Se actualiza los elementos de la interfaz que sea necesario
                        updateUI();
                        //Si la opción de notificar los choques está activada...
                        if(notifications)
                                if(status.warningDistance())
                                        /* Mostrar advertencia de colisión */
                }
        }

}
```

Código 4. Código del Handler: escuchando desde la actividad al thread.

Clases de distinto propósito

La aplicación no se resume en Actividades y el *Thread*. Además, y dado que estamos ante un lenguaje orientado a objetos, se han utilizado otras clases, detalladas a continuación.

Base de datos

La base de datos ha sido utilizada para almacenar las órdenes que uno quiera asignar al robot. Esta base de datos es local y hace uso del *SQLite* incluido en *Android*. La base de datos es sencilla, y no tiene más que una tabla con los siguientes datos de una Orden:

ID	Entero clave primaria	Clave primaria
orderCommand	Texto	Comando a ejecutar
comment	Texto	Descripción del comando
server	Texto	Servidor al que está asociado

Tabla 26. Objeto almacenado en table de datos: Order

De tal forma, si uno mismo con su aplicación se conecta a dos servidores distintos, tendrá guardado para cada uno de los servidores sus correspondientes órdenes.

Las clases que gestionan la base de datos son *Order.java* y *SQLiteHelper.java*. Order.java representa el objeto de la tabla. Está constituido con su constructor y con sus métodos *get & set*. Por el otro lado, *SQLiteHelper* es el objeto utilizado para gestionar la base de datos. Esta clase posee los métodos de creación y destrucción, inserción, borrado y extracción. Da soporte a lo que se conoce como *CRUD* (*Create*, *Read*, *Update* y *Delete*) en bases de datos, es decir, las 4 funciones básicas que todo almacenamiento persistente debe tener.

Log

La clase LogHelper.java, por su parte, gestiona el fichero interno de log (valga la redundancia), en el que registran los eventos que el usuario crea con su uso. Esta clase tiene un constructor que se inicializa cada vez que se abre una actividad, y accede a un fichero interno de la aplicación.

De entre sus métodos, destacan *read*, *write*, *getSize* y *delete*, los cuales, como es evidente, sirven para leer el fichero, escribirlo, comprobar su tamaño y borrarlo.

La utilidad de la clase Log es la de abstraer el código principal y hacer uso de las posibilidades de Java mediante la creación de objetos.

Constantes

Clase de vital importancia, en la que se definen las constantes que rigen a la aplicación. Ofrece gran limpieza de código y hace al código más entendible.

Nombre Constante	*Valor*	*Utilidad en*
SETTINGS_NAME	SETTINGS_ALIVECLIENT	SharedPreferences
CONTROL_OPTION	CONTROL_MODE	SharedPreferences
SKIP_INTRO	SKIP_INTRO	SharedPreferences
NOTIFICATIONS	NOTIFICATIONS	SharedPreferences
AUTOCONNECT	AUTOCONNECT	SharedPreferences
LAST_SERVER	LAST_SERVER_TAG	Conexión
LAST_PORT	LAST_PORT_TAG	Conexión
CONTROL_PORT_ENABLED	CONTROL_PORT_ENABLED	Conexión
IMAGE_PORT_ENABLED	IMAGE_PORT_ENABLED	Conexión
PREDEFINED_SERVER	Goriz.disca.upv.es	Conexión
PREDEFINED_PORT	5600	Conexión
FILENAME	LOG AliveClient.txt	Log
DISTANCE_ALERT	100	Notificación golpe
DISTANCE_FLOOR_ALERT	50	Notificación suelo
DISTANCE_FLOOR	30	Notificación suelo
DISTANCE_FLOOR_WINDOW	20	Filtrado notif. suelo
REFRESH_RATIO	5	Segundos refresco status
CONNECTION_ERROR	-1	Conexión
CONNECTION_OK	0	Conexión
SLIDER_RESOLUTION	100	Control de sliders
TILT_CLEREANCE	0	Control de joystick
PAN_CLEARANCE	10	Control de joystick
JOYMAX	75	Control de joystick
MSG_CONNECT	3	Thread (tipo mensaje)
MSG_OK	0	Thread (tipo mensaje)
MSG_ERR	-1	Thread (tipo mensaje)
MSG_UNKNOWN	-2	Thread (tipo mensaje)
MSG_STATUS	1	Thread (tipo mensaje)
MSG_CONTROL	2	Thread (tipo mensaje)
MSG_XXX_CMD	XXX	Comando del robot: XXX
POLL_INTERVAL	200	Refresco de cola (ms)
IMG_HIGH_WIDTH	640	Imagen. Resolución
IMG_HIGH_HEIGHT	480	Imagen. Resolución
IMG_LOW_WIDTH	320	Imagen. Resolución
IMG_LOW_HEIGHT	240	Imagen. Resolución
MAX_VOLT	14	Batería
MAX_AMP	5	Batería
MAX_CAPACITY	1024	Tamaño de queue
MEDIUM_CAPACITY	512	Tamaño de queue
LOW_CAPACITY	128	Tamaño de queue

Tabla 27. Constantes de la aplicación.

Las constantes MSG_XXX_CMD representan cada uno de los tipos de comandos que el robot soporta. En nuestro caso, por ejemplo, MSG_HELP_CMD tiene como valor *help*, y es la orden *help* del robot. Seccionar las órdenes con estas constantes permite poder hacer un tratamiento especial para cada tipo de orden de manera sencilla.

Speed

Clase que gestiona la velocidad de las dos ruedas del robot, en base a los movimientos de los sliders o *joystick* que el usuario ha hecho.

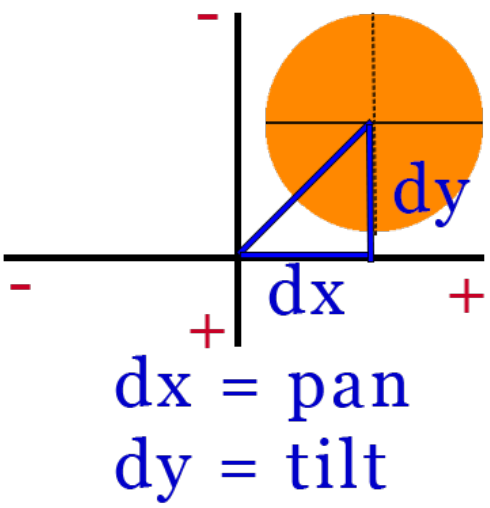


Ilustración 42. Gráfico del cálculo

Para el cálculo de la velocidad de las ruedas con el control del Joystick, el sistema entiende al Joystick como un eje de coordenadas en el que, al desplazar el joystick, obtenemos unos valores *pan* y *tilt*, los cuales son los vectores x e y desde el centro del eje de coordenadas a la posición del joystick. El eje de *tilt*, por la implementación del *widget[11]*, tiene el eje invertido.

El algoritmo que traduce la acción en el joystick en velocidades contiene dos partes:

La primera de ellas, de filtrado. Filtra la información obtenida de pan y tilt. Así el algoritmo entenderá que el usuario quiere hacer un movimiento *pan*/*tilt* siempre y cuando supere una ventana de valor 10. Es decir, si el usuario mueve muy poco la palanca hacia delante, se filtrará como que no se ha movido. Dado que es táctil y muy sensible, es imprescindible el filtrado de la señal obtenida.

Después, para la rueda izquierda (*left*) y la derecha (right) se calculan sus velocidades:

$$left = \max(-JOYMAX, \min(JOYMAX, -tilt + pan))$$

$$right = \max(-JOYMAX, \min(JOYMAX, -tilt - pan))$$

Pongamos por ejemplo que el usuario mueve la palanca hacia delante a tope, con un *pan* de 1 y un *tilt* de -85 *(¡signo de eje Y al contrario*!). Tras el filtrado, obtenemos unos valores de 0 y -85 (filtrado por sensibilidad, *¿quería el usuario moverlo ese 1? Nadie tiene tal precisión*).

Según las constantes (véase Constantes, apartado anterior), la fórmula calcula:

$$left = \max(-75, \min(75, 85)) = 75$$

$$right = \max(-75, \min(75, 85)) = 75$$

Obtenemos unas velocidades de 75 para cada rueda, que harán que se mueva como queríamos: hacia delante y a máxima velocidad.

Status

Clase que almacena los estados de los sensores para su uso. Además del constructor, posee dos métodos utilizados para la opción de advertir al usuario sobre el posible percance. El método *warningDistance* devuelve true en caso que alguno de los sensores esté dentro del rango definido en las constantes *DISTANCE*_ALERT, mientras que *getWarningCameraName* nos devuelve un *String* con el nombre del sensor que está próximo a colisionar.

InternetConnectionChecker

Clase de comprobación. Utilizada para comprobar la disponibilidad de la conexión a internet y que ésta sea mediante *WIFI* o *3G*.

Decisiones tomadas

Sobre el ciclo de vida

Este apartado ha sido reservado para aclarar varios aspectos sobre la implementación de la aplicación. En primer lugar, cabe comentar sobre la implementación del ciclo de vida de las actividades que se ha hecho de esta manera para evitar el uso excesivo de los recursos del móvil.

Si la destrucción del proceso en segundo plano o del *thread* temporizado fuera en el ciclo de *onStop*, al cambiar de pantalla, es probable que la aplicación se mantuviera en memoria, y por ende, esos procesos activos. De esta manera se asegura que lo activo de la aplicación va a ser siempre lo que el usuario ve.

Sobre el proceso en segundo plano

Como se comentó antes, a la hora de conectarse con internet, las posibilidades son varias, y sin embargo parece que se haya optado por la más arcaica.

Quizás sea así, pero con un motivo. El *AsyncTask* fue descartado en un primer momento por la naturaleza del mismo. Está diseñado para tareas asíncronas que tienen un fin. En nuestro caso, queremos que esté en completa ejecución.

Se podría haber hecho que cada vez que el usuario mueve el joystick o cada vez que se refrescara el estado de los sensores, se conectara al servidor, hiciera la petición y

desconectara. Resulta altamente ineficiente estar estableciendo esa conexión continuamente, creando y destruyendo objetos por doquier.

Otra de las alternativas eran los Servicios. Diseñados para ejecuciones en segundo plano, eran buenas opciones. Sin embargo, en este caso, aunque *IntentService* directamente estuviera ligado a la vida de la actividad, una de las características que buscamos, la comunicación de ida y vuelta era un tanto compleja y tediosa.

Con la solución de un *Thread* interno a la clase, la comunicación, descrita anteriormente, es mucho más sencilla.

Sobre el uso de Base de Datos

El uso de la base de datos puede resultar icónico, pero no lo es. La eficiencia a la hora de almacenar las órdenes es mucho mayor que en un simple fichero. Además, las bases de datos están optimizadas para ocupar poco. Si se usara sobre muchos robots (un ejército, quien sabe), el fichero empezaría a ocupar mucho, y hay que recordar que la memoria secundaria en un dispositivo móvil es un recurso valioso que hay que cuidar con lupa.

Sobre el uso de la batería

La vida de la batería es el gran problema de todo móvil. Consecuentemente, se ha programado el proceso en segundo plano para que no use muchos recursos de batería.

Inicialmente, en la primera prueba de ejecución de la aplicación completamente operativa, llegó a consumir el 47% de la batería en media hora, sólo recibiendo los datos de los sensores y con la pantalla encendida todo el tiempo.

El problema no era otro que el proceso en segundo plano. Dentro del bucle infinito, hay un bucle que comprueba si la cola está vacía. Pero, ¿qué pasa si la cola está siempre vacía, y sólo le llega una orden cada 5 segundos? En tal caso, la comprobación de la cola se realiza tantas veces como el procesador del móvil pueda.

Dicho problema llega a afectar al móvil de tal manera que no sólo consume su batería de manera desmesurada, sino que incluso llego a afectar a la temperatura del móvil y la pantalla, empezando a parpadear.

La solución ha sido trivial y eficaz. Se ha añadido una pausa a cada comprobación de la cola de 200ms, descrita en la constante POLL_INTERVAL. Esos 200ms son el tiempo medio de respuesta visual de un ser humano aproximadamente [10]. Si mientras está procesando una orden, el usuario añade a cola, pongamos por ejemplo, 10 órdenes, éstas serán atendidas sin tener que hacer espera de 200ms.

8. Conclusiones

Para concluir este proyecto se especifican las dificultades que he han tenido a lo largo de su desarrollo, así como unas breves notas sobre la experiencia obtenida y el futuro que *Alive Client* pueda tener.

Dificultades

La dificultad mayúscula en esta aplicación es sin duda la implementación de la lógica del proceso en segundo plano. Al tener tantas posibilidades (especificadas en "Sobre el proceso en segundo plano"), resultó difícil en un primer momento enfocarlo como se debía.

Probablemente la dificultad fue mayor de lo esperado porque se intentó por cada medio la implementación, y fue la óptima la última probada.

Otra de las dificultades es propia de *Android*. Su fragmentación hace que te debas preocupar, y con razón, más de la interfaz gráfica. No por el mero hecho de que en tu dispositivo se vea bien quiere decir que se vaya a ver bien en todos los dispositivos. No obstante, una vez entendido a la perfección el uso de los distintos *layouts*, es un problema menor.

Experiencia

La experiencia obtenida en la realización de este proyecto no puede ser calificada de otro modo que no sea óptima. La creación de un proyecto *Android* de pies a cabeza hace que te preocupes de cosas que quizás un programador directamente no vaya a tener en cuenta, pero que son necesarias.

Desde la creación de los distintos diagramas para focalizar el proyecto, pasando por el diseño gráfico de la misma, y finalmente codificando. Se han pasado por todas las fases de desarrollo de software.

Y lo que es más importante. Se ha adquirido experiencia en el desarrollo de aplicaciones cuya cuota de mercado es, probablemente, la mayor en cuanto a software se refiere. Y además, en Java, lenguaje también muy extendido.

El futuro del proyecto

El proyecto tiene un futuro muy amplio. La robótica está extendida y en extensión, y la telefonía móvil más aún. Sus aplicaciones en un futuro pues son innumerables, y por ende, sus posibles expansiones son cuantiosas:

1. Añadir al proyecto interacción con voz.
2. Añadir al proyecto algoritmos de reconocimiento de imágenes.
3. Programar comportamientos de robot como agresión, miedo o amor.
4. Control del robot mediante movimientos del móvil, utilizando los sensores de movimiento. Programado pero comentado. Falta calibración dado que la señal tiene mucho ruido.
5. Integrar la recepción de imágenes de la nueva versión del servidor. La aplicación fue adaptada a la anterior versión del servidor y actualmente no recibe las fotografías.

Desde el momento que puedes jugar con algoritmos de imagen, las posibilidades se multiplican:

1. Filtro paso-alto para realización de mapeados en 3d mientras el robot se mueve.
2. Búsqueda de caminos factibles.
3. Reconocimiento facial (robot vigía).
4. Órdenes mediante gestos físicos (a la cámara).

9. Referencias

- [1] Open Handset Alliance, 'Industry Leaders Announce Open Plataform for Mobile Devices', http://www.openhandsetalliance.com/press_110507.html, November 2007, (accessed 10 August 2014).

- [2] T-Mobile G1 review (2009), http://www.engadget.com/2008/10/16/t-mobile-g1-review-part-1-hardware/, (accessed August 2014)

- [3] IFR Statistical Departement via http://www.worldrobotics.org/, (accessed August 2014)

- [4] TIOBE, 'Index for August 2014', http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, (accessed August 2014)

- [5] The Festival Speech Synthesis System, http://www.cstr.ed.ac.uk/projects/festival/, (accessed August 2014)

- [6] Android Design, https://developer.android.com/design/index.html, (accessed July 2014)

- [7] Android Icon Guideline, http://developer.android.com/design/style/iconography.html, (accessed July 2014).

- [8] Android, Dashboards, https://developer.android.com/about/dashboards/index.html?utm_source=ausdroid.net, (accessed May and August 2014)

- [9] Androidutp via Google Images, https://androidutp.files.wordpress.com/2012/04/tutoa8.png, (accessed August 2014)

- [10] Taoka, George T., *Brake Reaction Times of Unalerted Drivers*, ITE Journal 59 (3): 19-21

- [11] Mobile-Anarchy-Widgets, https://code.google.com/p/mobile-anarchy-widgets/wiki/JoystickView, (accessed June 2014)

- Meier, R., *Professional Android 2 Applicacion Development*, Wiley Publishing, 2010.

- Ableson, F., Collins C. and Sen R., *Android: Guía para desarrolladores*, Anaya Multimedia, 2010.

- Martín Sierra, Antonio J., Programador Certificado JAVA 2 Curso Práctico, 3ª edición, Ra-Ma, 2010.

- Stackoverflow, http://stackoverflow.com/ (accessed July 2014)

- Vogella, http://www.vogella.com/ (accessed July 2014)

- Android, 'Code Style Guidelines for Contributors', https://source.android.com/source/code-style.html, (accessed July 2014)

- Android Asset Studio, http://romannurik.github.io/AndroidAssetStudio/index.html (accessed August 2014)

- Android Developers, http://developer.android.com/index.html (accessed July-August 2014)