industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

Universidad Politécnica de Cartagena

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Estudio, Diseño y Desarrollo del Sistema de Control de un UAV a Través de un Dispositivo Móvil Inteligente

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Antonio Pastor Conesa**
Director: Juan Antonio López Riquelme
Codirector: Nieves Pavón Pulido

Cartagena, a 10 de Julio de 2016

Agradecimientos

- A Juan Antonio López y Nieves Pavón por su ayuda durante la realización de este trabajo, sin la cual, no habría sido posible llevarlo a cabo.

- A mis padres por apoyarme, ayudarme e insistir todos los días de la carrera para que siguiera adelante.

- A todos los compañeros de la carrera, por estos increíbles años y por todos sus apoyos hacia mí.

Índice de Contenido

Índice de Figuras

Capítulo 1

Introducción

1.1 Contexto del trabajo

En la última década se han desarrollado una gran variedad de vehículos aéreos no tripulados o UAVs (Unmanned Aerial Vehicle). Este tipo de vehículos pueden ser controlados remotamente de forma manual o de forma totalmente automática. En sus inicios estaban destinados a usos militares, pero hoy en día son muy útiles para diversas tareas, como puede ser la lucha contra incendios, la seguridad, la vigilancia, la medida de indicadores en grandes cultivos y el entretenimiento.

Los vehículos aéreos no tripulados tienen que ser rápidos, eficientes y capaces de realizar todo tipo de tareas sin poner en peligro la vida de un piloto. En función de la tarea que han de realizar, este tipo de vehículos pueden tener diferentes estructuras, software y hardware, como por ejemplo, los UAVs de uso militar suelen tener una estructura de avión y los de uso doméstico una estructura de cuadricóptero, que consiste básicamente en un helicóptero con cuatro rotores.

Para hacer posible la navegación, tanto autónoma como manual, los vehículos son equipados con toda clase de sensores que captan la información del entorno y del propio vehículo, los cuales se denominan estereoceptivos y propioceptivos, respectivamente.

1.2 Motivación

Debido a que existen multitud de Drones, y cada uno de ellos posee un sistema de control distinto, con su propia aplicación y órdenes de control, resulta de interés desarrollar un único sistema que sea capaz de controlar todo tipo de Drone, independientemente de la marca y su método de control.

Tal y como se verá más adelante, una buena alternativa para conseguir este objetivo global es la utilización de un middleware para robótica, como puede ser ROS..

La utilización de ROS tiene la ventaja de utilizar cualquier hardware similar desde una perspectiva de más alto nivel. De este modo, todos los Drones enviarán y recibirán el mismo tipo de mensajes. Simplemente, se debe usar un paquete para que interprete las órdenes de control y las traduzca al lenguaje del Drone. También es importante mencionar que a día de hoy la comunidad ROS es muy grande y existen paquetes para el control de muchos robots.

Por todo ello, la idea principal de este Trabajo de Fin de Grado es conseguir crear una aplicación para dispositivos Android que sea capaz de comunicarse con cualquier UAV, a través de un dispositivo intermedio que corra ROS, y ese sistema será el encargado de interpretar las ordenes de la aplicación Android y enviar al UAV la orden de control en su lenguaje adecuado. El proceso completo será explicado en detalle más adelante.

1.3 Objetivos

El objetivo principal es conseguir la comunicación entre el usuario, mediante un Smartphone, y el Drone. Esto se puede dividir en varios sub-objetivos:

- Programar una aplicación para un Smartphone con sistema Android, donde se pueda tanto controlar el Drone como visualizar la información recibida del mismo.

- Programar el software necesario en el sistema ROS que sea capaz de recibir órdenes de control de la aplicación Android y traducirlas al lenguaje del UAV que se usará.

1.4 Estructura de la memoria

La estructura de esta memoria es la siguiente:

- Capítulo 1: Contiene información acerca de la temática de este TFG y de su estructura.

- Capítulo 2: Contiene una introducción acerca de los UAVs y una breve descripción de cada elemento utilizado en el TFG.

- Capítulo 3: Este capítulo explica, en más profundidad, las características del UAV que se usa, así como el resto de materiales software utilizados.

- Capítulo 4: En este capítulo se incluye el código de la aplicación Android, así como la explicación de las partes más importantes.

- Capítulo 5: En este capítulo se presentan las conclusiones obtenidas tras la realización de este TFG. Incluyendo un apartado donde se proponen trabajos futuros para continuar con la investigación.

1.5 Fases

Para la realización de este TGF se han seguido las siguientes fases:

- Fase 1: Estudio de la arquitectura ROS.

- Fase 2: Diseño de módulos software compatibles con ROS.

- Fase 3: Estudio de la arquitectura Android.

- Fase 4: Diseño de la aplicación Android para enviar los datos a la plataforma que corre ROS.

- Fase 5: Validación del sistema en un entorno simulado y solución de errores.

Capítulo 2

Estado del arte

2.1. Introducción

En este capítulo se analizan los principales materiales utilizados para la realización de este Trabajo de Fin de Grado, que consiste en el desarrollo de un conjunto de componentes software para poder realizar el control de un Drone mediante una aplicación móvil.

En primer lugar se realizará una introducción a los UAVs desde los comienzos hasta la actualidad y, posteriormente, se analizarán algunos modelos disponibles, así como los middlewares y sistemas operativos que podrían ser utilizados para implementar el sistema propuesto.

2.2 UAVs

Se puede definir un UAV como un vehículo capaz de volar sin la necesidad de un piloto sobre él, además puede ser de forma manual o automática. Un ejemplo sería el Elbit Systems Hermes 900 (ver Figura 2.1), que es una UAV israelí de tamaño medio y utilizado para desarrollar misiones tácticas de media altitud y larga duración. Concretamente, tiene una autonomía de más de 30 horas, puede volar a una altitud máxima 9144 m, y su misión principal es de reconocimiento, vigilancia y retransmisor de comunicaciones.

2.2.1 Antecedentes

Ha día de hoy existen multitud de UAVs diseñados para tareas muy diversas, por lo que es importante la utilización de sistemas de control autónomos que sean rápidos, fiables y eficaces.

Dentro de los UAV no sólo están los de estructura de avión o helicóptero, sino que también entran en esta definición los globos e incluso los misiles.

Figura 2.1. UAV israelí HERMES 900.

Los UAV han sufrido una constante evolución hasta llegar a lo que hoy conocemos. El origen de los UAV viene de aproximadamente un siglo atrás.

Fue en el siglo XX, durante la primera Guerra Mundial, cuando se desarrollaron los primeros misiles crucero, o torpedos aéreos, como fueron bautizados entonces. Consistían en un misil con unas pequeñas alas, un motor y usaban un sistema de guía inercial que estaba formado por un conjunto de giroscopios. De esta forma el misil podía ajustar su trayectoria de forma automática para impactar con el blanco prefijado (ver Figura 2.2) [2].

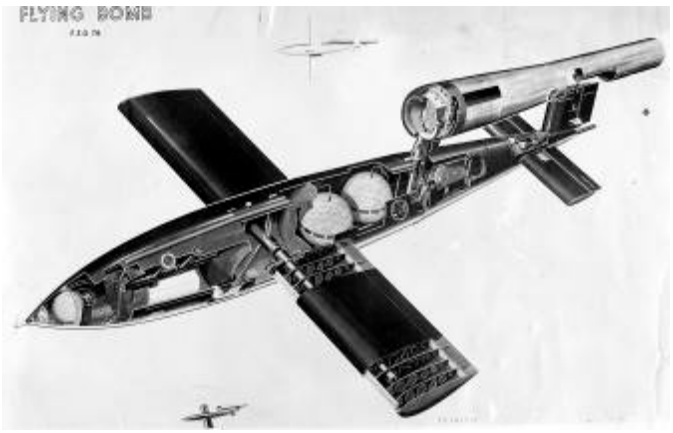

Figura 2.2. Primer misil crucero.

Actualmente, los misiles cruceros han evolucionado mucho, son capaces de volar hasta 3000 km e impactar en el blanco con un error de 1 m. Tampoco es necesario que se tengan que lanzar desde la superficie, es posible lanzarlos bajo el agua (ver Figura 2.3) [3].

Figura 2.3. Misil crucero actual.

Más tarde, durante la Segunda Guerra Mundial, los británicos comenzaron a crear aviones guiados por radiocontrol, que eran utilizados como dianas aéreas. Su control era bastante más impreciso que los de la actualidad (Figura 2.4).

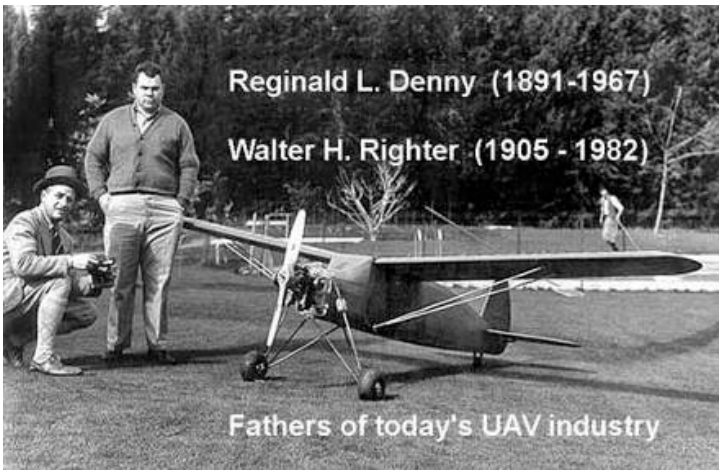

Figura 2.4. UAV como diana aérea.

Posteriormente, durante la guerra fría, se le fueron incorporando diversos sensores y cámaras a los UAVs, para poder ser utilizados en labores de exploración del terreno. Fue a finales del siglo XX cuando se desarrollaron UAVs eficaces para el ataque militar.

Hasta ese momento, la finalidad básica de los UAVs era totalmente militar, no se usaba para uso civil ni para otros usos. Sin embargo, al finalizar la guerra fría, USA decide emplear la información y materiales utilizados para los UAVs de uso militar para comenzar a crear UAVs de uso civil y recreativo.

Por ello, a partir de ese momento y hasta la actualidad se han ido desarrollando los UAVs para diversas tareas de uso civil, como la vigilancia o el rescate de personas, como también para el entretenimiento. Con el paso del tiempo los UAVs están mejor equipados a nivel de hardware y software, lo que los hace más rápidos y eficaces.

En la Figura 2.5 podemos observar un UAV utilizado en Nueva Zelanda para el transporte de pequeños paquetes.

Figura 2.5. UAV envío postal

En la actualidad, podemos clasificar los UAVs según su misión principal en estos seis tipos [4]:

- Blanco: Sirven para simular aviones o ataques enemigos en los sistemas de defensa de tierra o aire.
- Reconocimiento: Utilizados para el reconocimiento del terreno y recabar información militar.
- Combate: Para combatir y llevar a cabo misiones que suelen ser muy peligrosas.
- Logística: Diseñados para transportar carga.

- Investigación y desarrollo: En ellos se prueban e investigan los sistemas de desarrollo.
- Comerciales y civiles: Son diseñados para propósitos civiles, como puede ser rodar películas o el entretenimiento.

Además, tal y como se muestra en la Figura 2.6 existen UAVs de muchos tamaños y tipos de estructuras.

Figura 2.6. UAV Diversos diseños y tamaños.

Un sistema UAV tiene dos segmentos claramente definidos:

- Segmento de vuelo: Formado por el vehículo aéreo y los sistemas de recuperación (sistema de aterrizaje).
- Segmento de tierra: Formado por la estación de control. Está en tierra y es la encargada de enviar la información de control al dron y recibir la información de los sensores abordo.

Por lo tanto, para pilotar estos Drones de forma remota es necesario un sistema que permita comunicarnos con los mismos. Para esto existen multitud de herramientas, desde aplicaciones para el control del Drone, hasta la transmisión de vídeo de las cámaras de abordo. En concreto, para el modelo Ar. Drone 2.0 de Parrot existen varias aplicaciones, tanto oficiales como libres.

2.2.2 Introducción a los cuadricópteros

Para el correcto uso de un vehículo no tripulado es necesario definir un sistema de referencia, mediante el cual, podremos saber en todo momento la posición y orientación de

nuestro vehículo. La posición del vehículo viene dada por una serie de variables que cambian, en forma y número, según el sistema de referencia usado.

A la hora de definir un sistema de referencia para un vehículo terrestre podemos usar el sistema de referencia básico con tres variables para determinar su posición. Este sistema consta de tres variables (X, Y, θ), donde x es la coordenada que tiene la posición en el eje x, y es la coordenada que tiene la posición en el eje y, y θ es la orientación del vehículo, normalmente expresado en grados. Mediante este sistema de referencia se puede representar un objeto en un plano (ver Figura 2.7).

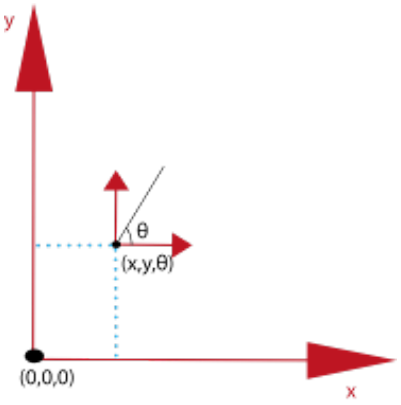

Figura 2.7. Sistema referencia para un vehículo terrestre.

Cuando hablamos de un vehículo aéreo el sistema de referencia anterior no es suficiente, ya que no es un plano donde nos estamos moviendo, sino un espacio tridimensional. Esto es debido a que hay que considerar la altitud del vehículo, por lo que se añade el eje z. Sin tener en cuenta la orientación del vehículo ya necesitamos tres coordenadas (x, y, z) para saber dónde está ubicado (ver Figura 2.8).

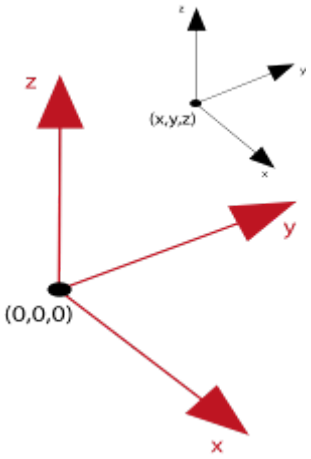

Figura 2.8. Sistema de referencia básico para un vehículo aéreo.

En el vehículo terrestre, la orientación venía dada por un solo ángulo, que variaba con respecto a la orientación del sistema de referencia. En un vehículo aéreo, se necesita un sistema de orientación compuesto por tres valores, asociados a cada eje anteriormente citados, cuyos nombres son [5]:

- Pitch: Corresponde con la inclinación del morro del vehículo, o con la rotación del eje ala-ala.
- Roll: Corresponde con la rotación del eje morro-cola.
- Yaw: Corresponde con la rotación alrededor del eje vertical perpendicular al avión, en nuestro caso sería el eje z.

En la Figura 2.9 podemos observar una representación conjunta, tanto de los ejes de orientación como de los de rotación, que constituyen un sistema de referencia completo de un vehículo aéreo.

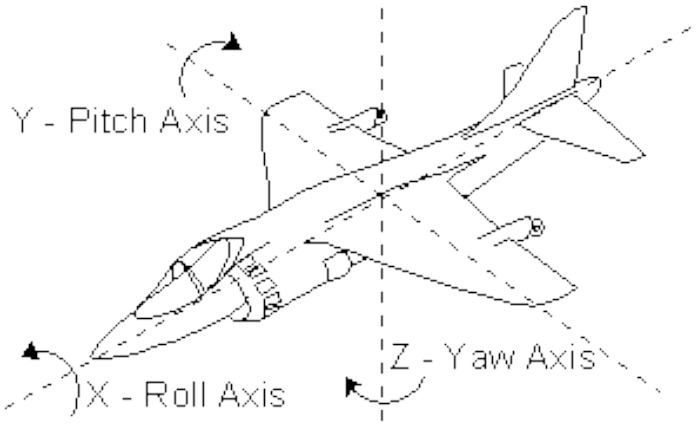

Figura 2.9. Sistema de referencia completo vehículo aéreo

En nuestro caso nos centraremos en un cuadricóptero. Un cuadricóptero tiene cuatro hélices que le permiten realizar los diversos movimientos. Cada hélice es controlada por un motor independiente del resto, por lo que, aumentando o disminuyendo la potencia que se le da a cada motor conseguiremos un movimiento u otro del cuadricóptero. La potencia de cada motor se expresa mediante el símbolo Ω y el aumento de esta mediante el símbolo Δ.

Hay varios tipos de estructuras posibles para un cuadricóptero, en nuestro caso está será una estructura en cruz, uniéndose los soportes en un punto común. Cada par de hélices debe girar en la misma dirección (ver Figura 2.10).

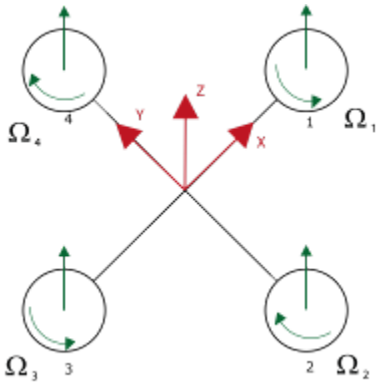

Figura 2.10. Estructura del cuadricóptero.

La hélice número 1 es la delantera, la número 3 la trasera, la número 2 la derecha y la número 4 la izquierda.

Como se puede observar en la Figura 2.10, un par de hélices debe girar en sentido horario (en este caso el 2 y 4) y el otro par en sentido horario (en este caso 1 y 3). Esto es así para conseguir la anulación de pares producidos por cada motor.

Los diversos movimientos posibles con un cuadricóptero se consiguen variando la potencia de cada hélice en varias combinaciones. Los principales movimientos son [6]:

- Ascensión: es el que consigue que el cuadricóptero aumente su altura durante un tiempo, es decir, un aumento de la coordenada del eje z. Para conseguir este movimiento es necesario aportar a las cuatro hélices el mismo aumento de potencia (ver Figura 2.11). En función la potencia aportada el cuadricóptero sufrirá una ascensión con mayor o menor velocidad.

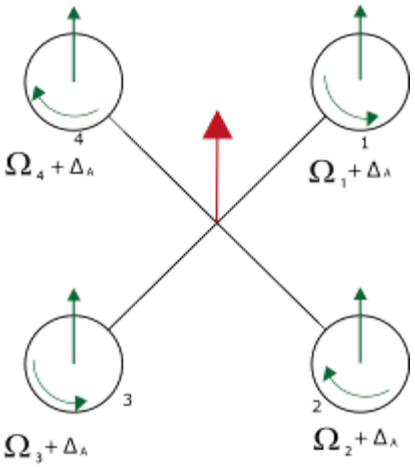

Figura 2.11. Movimiento de ascensión de un cuadricóptero.

- Alabeo: Es el que consigue que el cuadricóptero se desplace hacia la izquierda o hacia la derecha durante un período de tiempo, es decir, un cambio en las coordenadas del eje y. Para conseguir este movimiento hay que aportar la misma potencia a las hélices 1 y 3, y en función de si queremos ir a la derecha o izquierda variar la potencia de las hélices 2 y 4. En el caso de querer girar hacia la derecha debemos aportar menos potencia a la hélice 2 y más a la 4 (ver Figura 2.12). De esta forma, debido al desequilibrio de pares, se produce un giro hacia la derecha, que manteniéndolo un tiempo, el cuadricóptero avanzará en esa dirección.

  La velocidad con la que el cuadricóptero se moverá está ligada a la potencia aportada a las hélices 1 y 3, mientras que el grado de inclinación vendrá dado por la potencia de las hélices 2 y 4.

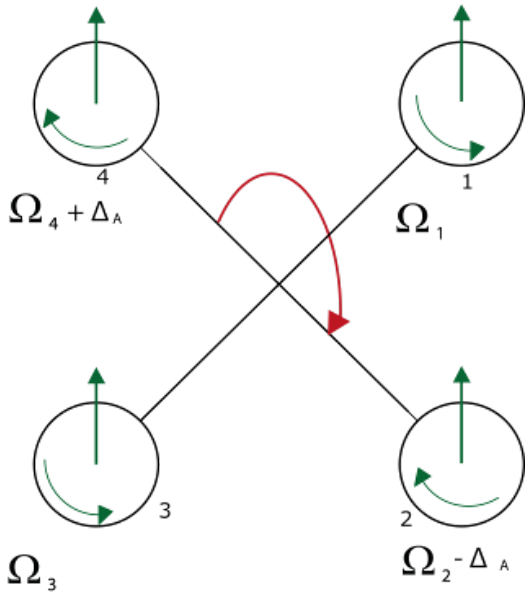

Figura 2.12. Movimiento de alabeo de un cuadricóptero.

- Cabeceo: Es el que se consigue cuando el cuadricóptero se mueve hacia delante o hacia atrás, es decir, un cambio en las coordenadas del eje x. Para conseguir que el cuadricóptero se mueva hacia delante debemos aportar la misma potencia a las hélices 2 y 4, y más potencia a la hélice 3 que a la 1. De esta forma conseguimos que el cuadricóptero gire respecto del eje x durante un tiempo y, manteniendo esas potencias, avanzará en esa dirección (ver Figura 2.13). La velocidad de avance va determinada por la potencia aportada

a las hélices 2 y 4, mientras que el grado de inclinación por la potencia aportada a las hélices 1 y 3.

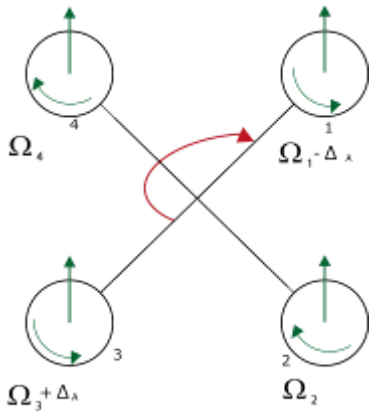

Figura 2.13. Movimiento de cabeceo de un cuadricóptero.

- Guiñada: Consiste en hacer girar el cuadricóptero sobre sí mismo para así cambiar su orientación, es decir, realizar un movimiento rotatorio sobre el eje z. Para conseguir este movimiento es necesario aportar la misma potencia a cada par de motores. Si queremos girar a la derecha (ver Figura 2.14), los motores 1 y 3 tendrán que girar más rápido que el par 2 y 4. Si queremos girar a la izquierda, los motores 2 y 4 deben girar más rápido que los motores 1 y 3. Este movimiento es debido a la diferencia de potencias suministradas a cada par de hélices. El giro se produce en el sentido contrario al sentido de las hélices a las que se le aporte más potencia. En este caso se le aporta más potencia a los motores 1 y 3, como su sentido es antihorario, el cuadricóptero girará en sentido horario.

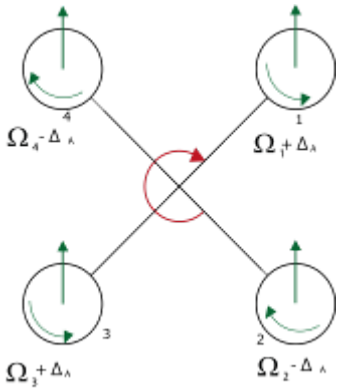

Figura 2.14. Movimiento de guiñada de un cuadricóptero.

2.2.3 Drones comerciales

2.2.3.1 Ar Drone 2.0

Contiene una cámara HD de 720p para grabar video y capturar fotos con un dispositivo remoto o con una unidad flash USB. Se pilota, entre otras, con la aplicación AR. Freeflight, desde la que se pueden manejar las cámaras, el Drone y obtener los datos de navegación.

Está fabricado en materiales ligeros de alta calidad, como fibra de carbono, espuma aislante, nanorevestimiento impermeable, nylon, y todas las partes son reparables.

Cuenta con una computadora a bordo con CPU ARM Cortex A8 de 1 GHz y un DSP TMS320DMC64x de 800 MHz para el control del vídeo, con 1 GB de RAM DDR2. Funciona gracias al kernel Linux 2.6.32 y soporta USB 2.0 y WiFi.

El sistema de control cuenta con giroscopo de 3-ejes, acelerómetro de 3-ejes, magnetómetro de 3-ejes, sensor de presión y sensores de ultrasonido para medir la altitud.

Los 4 motores inrunner sin escobillas le permiten volar alto y rápido, que gracias a los engranajes de Nylatron el ruido es bajo y se controlan mediante un microcontrolador con arquitectura AVR de 8 bits, como los que podemos encontrar en las placas de Arduino. Su batería es de 1000 mAh, por lo que su autonomía sube considerablemente (15 min) con respecto a los anteriores modelos.

Como se ha comentado anteriormente, la aplicación oficial para controlar este Drone es AR.Freeflight 2.0 (ver Figura 2.15), disponible tanto para sistemas operativos Android como iOS. Desde el menú principal se pueden acceder a varias opciones [7]:

- **FreeFlight**: acceso a la aplicación de pilotaje. El jugador puede grabar vuelos, hacer vídeos HD o tomar fotografías y guardarlos en su dispositivo de pilotaje. Pueden guardarse todos los datos de vuelo (altitud, velocidad, duración y lugar), ser comprobados por el piloto y compartidos con la comunidad.
- **Guest Space**: acceso a una presentación del AR.Drone 2.0, los mejores vídeos de vuelo e información práctica.
- **Drone Update**: acceso a actualizaciones gratuitas de software de AR.Drone 2.0.

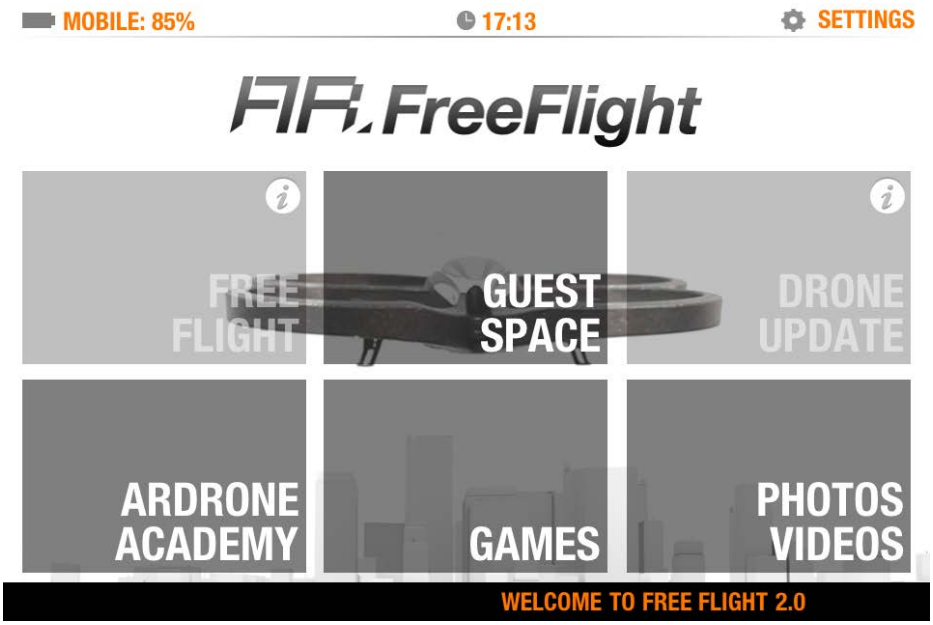

Figura 2.15. Menú principal de la aplicación AR. Freeflight 2.0.

- **AR.Drone Academy**: acceso a datos de geolocalización sobre las mejores zonas de vuelo, ver vídeos de otros pilotos y acceso a sus datos de vuelo.
- **AR.Games**: acceso a aplicaciones y juegos disponibles para el AR.Drone.
- **Photos/Vídeos**: acceso directo a tus vídeos y fotografías, que pueden verse o cargarse en Youtube para que la comunidad pueda disfrutar de ellos.

La pantalla de control es muy fácil de usar e intuitiva (ver Figura 2.16).

Figura 2.16. Pantalla de Control de la aplicación AR.Freeflight.

2.2.3.2 DJI Phantom 2 Vision

Este Drone [8] incluye una videocámara de 14 MP con capacidad de grabación HD, RAW y JPG en una microSD de 4 GB. El control de la cámara se realiza mediante la app DJI VISION y un sofisticado sistema anti-vibración estabiliza la cámara durante el vuelo. El control de la aplicación funciona mediante Wi-Fi y por ello se puede tener un alcance de hasta 300 metros. Su autonomía de vuelo es de 25 min por su batería de polímero de lítio de 5200 mAh de capacidad, y también es importante mencionar que incorpora un sensor GPS.

Figura 2.17. Captura de pantalla de la App DJI VISION.

Con esta aplicación, además de controlar la cámara, puedes crear la ruta que quieres que siga el Drone, pero para realizar un control en tiempo real del mismo, es necesario un mando.

Es un Drone muy fácil de pilotar gracias a su sistema de control de vuelo DJI Naza-M, formado por una unidad inercial, altímetro barométrico, una brújula, unidad GPS, indicadores LED de vuelo y un controlador, que permite que todos estos dispositivos trabajen en conjunto.

Figura 2.18. DJI Phantom 2 Vision.

2.2.3.3 Erle-Copter

Es un cuadricóptero desarrollado por la compañía española Erle Robotics. Es el primer Drone comercial que emplea Ubuntu como sistema operativo y el primero que se conecta al sistema mavros ROS package por WiFi.

Hay varias apps oficiales, pero al estar basado en Ubuntu, con el sistema ROS, cualquiera puede desarrollar su propia aplicación. Cuenta con una comunidad donde los usuarios pueden subir las aplicaciones que han creado para compartirla con los demás.

También dispone de modo automático, donde le marcas los puntos que quieres que siga, y el Drone efectúa el recorrido.

En cuanto a características técnicas, Erle-Copter cuenta con GPS, brújula, sensores de ultrasonidos, WiFi, radiocontrol 2,4 GHz, batería de 5.000 mAh, que le permite volar durante 20 minutos, un procesador de 1 GHz, 512 MB de RAM y 4 GB de almacenamiento, ranura microSD. No lleva cámara incorporada, pero es posible acoplarle una GoPro. Es capaz de levantar una carga o payload de 2Kg.

Figura 2.19.Erle Copter.

2.2.3.4 Walkera Scout X4

El Drone Scout X4 [9] de la compañía Walkera es un cuadricóptero semi-profesional, utilizado para la captura de vídeo gracias a sus prestaciones.

- Posee una batería LiPo de 5400 mAh 6S, que ofrece un tiempo de vuelo de 25 minutos.

- Chasis de fibra de carbono, lo que lo hace más ligero y maniobrable.
- Telemetría en tiempo real.
- Funciones especiales como: vuelta al punto de inicio automáticamente, modo de vuelo 'follow me', vuelo automático por Waypoints.
- Posibilidad de convertir sus 4 motores en 8, para cargar con más peso.
- Alcance de vuelo de hasta 1500m.

Figura 2.20. Cuadricóptero Walkera Scout X4.

El control del Drone se efectúa mediante un mando con funciones de FPV, pero además es compatible con la aplicación Wakera Flight Assistant, tanto en Android como en iOS. Con esta aplicación puedes tener un control total del Drone. La conexión del mismo con la aplicación se efectúa mediante una pequeña estación en tierra Bluetooth.

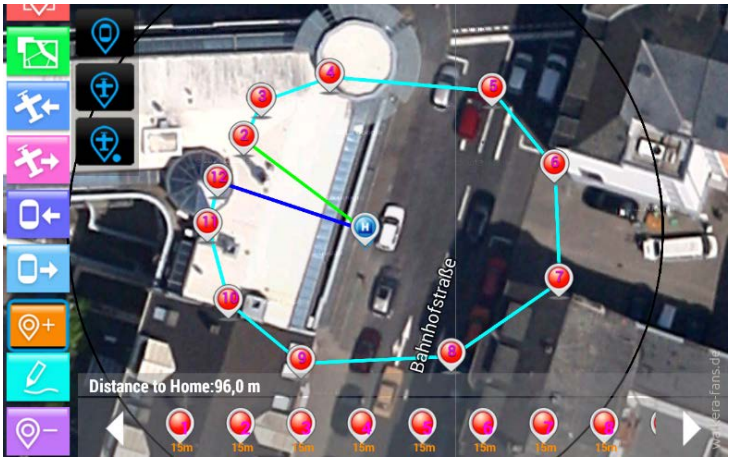

Figura 2.21. App Walkera.

2.3 Middleware para robótica

Un middleware es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware o sistemas operativos. Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones y sincronizaciones que son necesarias en los sistemas distribuidos. De esta forma, se provee una solución que mejora la calidad de servicio, así como la seguridad y el envío de mensajes.

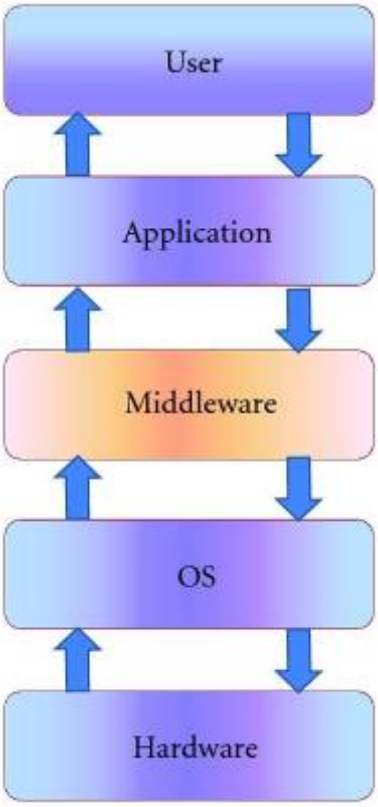

Figura 2.22. Esquema presentando las capas de un sistema middleware.

Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El middleware abstrae de la complejidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para conseguir una fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funciones necesarias, serán útiles diferentes tipos de servicios de middleware.

2.3.1 ROS

ROS es un meta-sistema operativo para robots que provee una serie de servicios orientados a facilitar el desarrollo de software para robots, así como su operación. Entre los servicios ofrecidos presenta abstracción de hardware, control de bajo nivel, comunicación entre procesos, y manejo de paquetes de software. Además incluye herramientas y librerías para facilitar el desarrollo, compilación y ejecución de código de forma distribuida en diversas plataformas de procesamiento.

ROS está diseñado teniendo en mente la reutilización de código, por lo que es muy transparente. Esto significa que es posible integrar ROS con librerías ya existentes, así como entornos de desarrollo para robótica. Por otra parte, ROS soporta actualmente diversos lenguajes, como pueden ser, C++, Python y Java, entre otros. Esto permite una mayor flexibilidad a la hora de programar y de implementar código ya existente.

ROS tiene una gran comunidad de usuarios donde es posible encontrar y compartir multitud de aplicaciones y librerías concretas de forma sencilla.

Comunicación en ROS

Para poder explicar las formas que tiene ROS de comunicarse hay que explicar una serie de conceptos que representan cada uno de los elementos involucrados en la comunicación.

- Nodo: Los nodos son los procesos ejecutables que realizan el cómputo. En un sistema de control robótico suelen existir varios nodos, donde cada uno se encarga de una tarea específica. Por ejemplo, un nodo puede encargarse de leer el dispositivo laser, otro de realizar el planeamiento de trayectorias y otro de otorgar una interfaz gráfica de control.
- Master: El programa master es el proceso central que mantiene un registro de todo el grafo computacional. Es el que permite la comunicación entre los nodos. Además, cuenta con un servidor de parámetros que permite centralizar cierta información.
- Mensaje: Los nodos se comunican entre sí enviándose mensajes. Un mensaje es una estructura de datos simple basada en campos de datos con un tipo definido, que pueden ser tipos primitivos (entero, punto flotante, booleano) o un conjunto de estos.

Los tipos de mensajes son definidos en archivos de texto con determinación *.msg*, y sus nombres siguen la nomenclatura de ROS, es decir, el mensaje definido en *std_msgs/msg/String.msg* es de tipo *std_msgs/String*.

- Topic: Los mensajes se envían generalmente en base a un sistema de publicación/suscripción. Un nodo envía un mensaje publicándolo bajo un Topic, cuyo nombre identifica el contenido del mensaje (como puede ser la odometría). Todos los nodos que quieran recibir la odometría, se suscriben entonces a ese Topic y recibirán todos los mensajes que el nodo original publique, sin que este nodo se deba preocupar de quienes están suscritos o no a él. De esta manera se facilita y generaliza el proceso y se simplifica el desarrollo de nodos.
- Servicio: Un modo alternativo de comunicación entre nodos, en vez de un paradigma de publicación/suscripción presenta un sistema basado en peticiones y respuestas. Un nodo puede ofrecer un servicio, que define dos tipos de mensaje: una petición y una respuesta. Un nodo puede enviar un mensaje del tipo petición al nodo que ofrece el servicio, y este le responderá un mensaje del tipo respuesta.

En el siguiente gráfico se pueden observar los dos métodos de comunicación entre nodos explicados anteriormente.

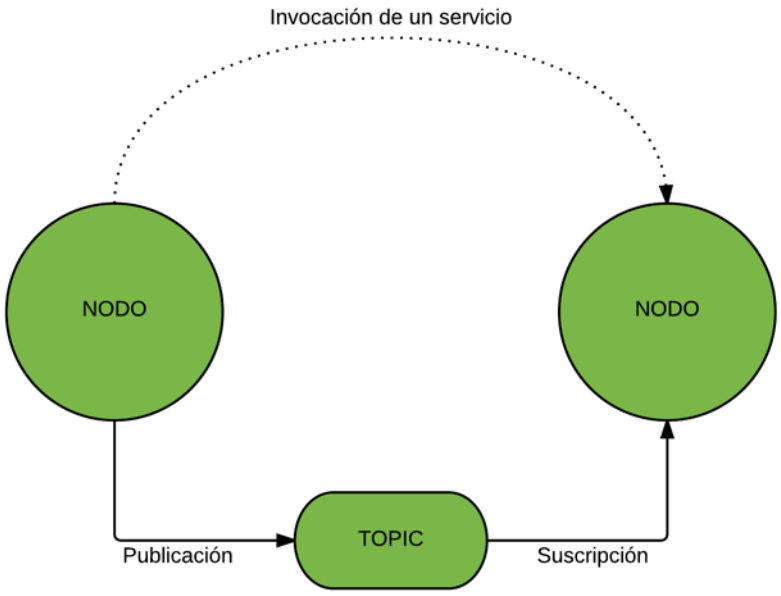

Figura 2.23. Esquema del intercambio de información usando ROS.

2.3.2 QGroundControl

QgroundControl [10] es un potente software de tipo open-source para utilizar con UAVs y Drones. Nos permite visualizar y controlar nuestro UAV durante el desarrollo y operación de una misión.

Las características principales son:

- Trabaja con el protocolo Open Source MAVlink (Micro Air Vehicle Communication).
- Incorpora mapas aéreos en 2D y 3D para controlar nuestro UAV mediante Waypoints.
- Manipulación de parámetros de nuestro UAV en tiempo real en pleno vuelo.
- Visualización en tiempo real de datos de sensores y telemetría.
- El protocolo MAVlink soporta hasta 255 vehículos en paralelo.
- QgroundControl funciona en plataformas Windows, Linux y MacOS.

Este software es compatible con el Ar Drone 2.0 y, además con otros muchos "autopilots" open source, como pueden ser Ardupilot o pxIMU.

Es un sistema flexible con librerías de código abierto que permiten trabajar con datos, variables y estructuras típicas del lenguaje C.

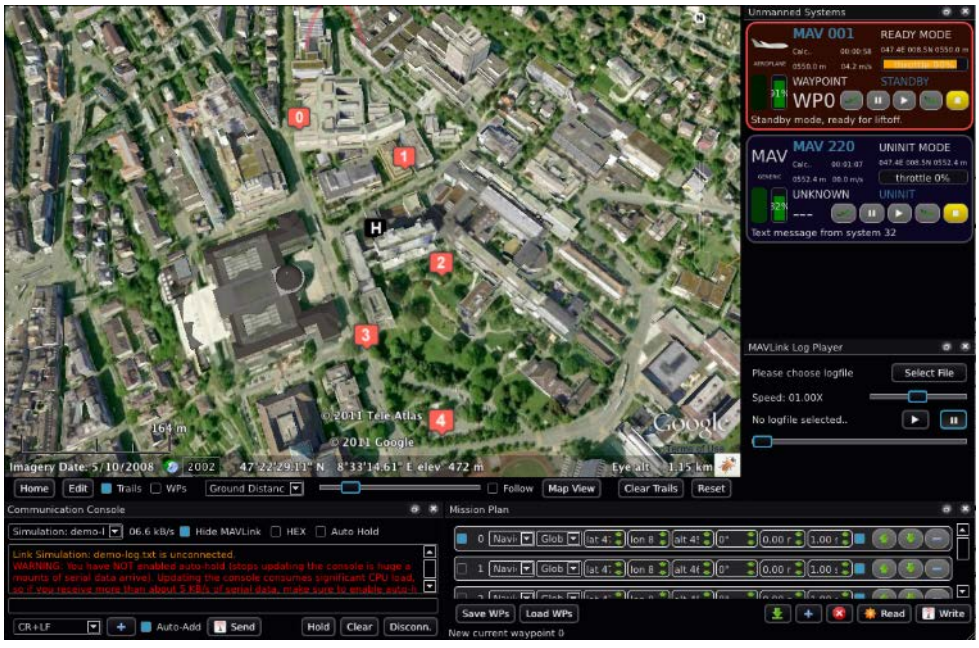

Figura 2.24. Captura de pantalla del software QgroundControl.

2.4 Sistemas operativos para dispositivos móviles inteligentes

2.4.1 Android

Android [11] es un sistema operativo móvil basado en Linux, que junto con aplicaciones middleware (software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones) está enfocado para ser utilizado en dispositivos móviles como Smartphones, Tablets, Google TV y otros dispositivos.

Fue desarrollado inicialmente por Android Inc., una firma comprada por Google en 2005. Pero en realidad es el principal producto de la Open Handset Alliance, un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicios.

El anuncio del sistema Android se realizó el 5 de Noviembre de 2007, junto con la creación de la Open Handset Alliance. Google libero la mayoría del código de Android bajo la licencia Apache, una licencia libre y de código abierto.

El sistema operativo se halla en una zona de memoria de sólo lectura por dos motivos: para evitar que el usuario lo dañe sin querer y para que se sea fiel a las pequeñas modificaciones y aplicaciones integradas que los fabricantes suelen incluir en sus modelos.

Al ser un sistema operativo de código fuente abierto, Android permite toda clase de modificaciones. Además de las ROM oficiales, es muy habitual encontrar ROM hechas por grupos de voluntarios que toman el código base y le añaden o quitan características o interfaces de usuario. Incluso uno mismo puede crear la suya.

Para actualizar un móvil Android se tienen varias opciones, siempre dependiendo de la operadora y sobre todo del fabricante de nuestro dispositivo. Algunos permiten actualizar por medio de la conexión USB entre el móvil y el PC, y otros directamente en el dispositivo descargando un archivo a la microSD y encendiendo el móvil. Sin embargo, lo más normal es que se actualice un Android por OTA (Over The Air) o inalámbricamente, sin tener que conectar el móvil por cable ni para descargar un archivo ni para actualizar mediante un programa.

Version	Codename	API	Distribution
2.2	Froyo	8	0.3%
2.3.3 - 2.3.7	Gingerbread	10	5.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.1%
4.1.x	Jelly Bean	16	14.7%
4.2.x		17	17.5%
4.3		18	5.2%
4.4	KitKat	19	39.2%
5.0	Lollipop	21	11.6%
5.1		22	0.8%

Figura 2.25. Versiones del sistema operativo Android.

La arquitectura de Android está distribuida en diferentes capas, las cuales se describen a continuación (Ver Figura 2.26):

- Applications (Aplicaciones): Las aplicaciones bases incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.
- Application Framework (Marco de trabajo de aplicaciones): Los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Este mecanismo permite que los componentes sean reemplazados por el usuario.
- Libraries (Bibliotecas): Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del framework de aplicaciones de Android. Las bibliotecas escritas en lenguaje C/C++ incluyen un administrador de pantalla táctil (surface manager), un framework Open Core (para el aprovechamiento de las capacidades multimedia), una base de datos relacional SQLite, una API gráfica OpenGL ES 2.0 3D, un motor de renderizado WebKit, un motor gráfico SGL, el protocolo de comunicación segura SSL y una biblioteca estándar de C, llamada “Bionic” y desarrollada

por Google específicamente para Android a partir de bibliotecas estándar "libc" de BSD.

- Android Runtime (Funcionalidad en tiempo de ejecución): Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima.
- Linux Kernel (Núcleo Linux): Android dispone de un núcleo basado en Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores, y también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

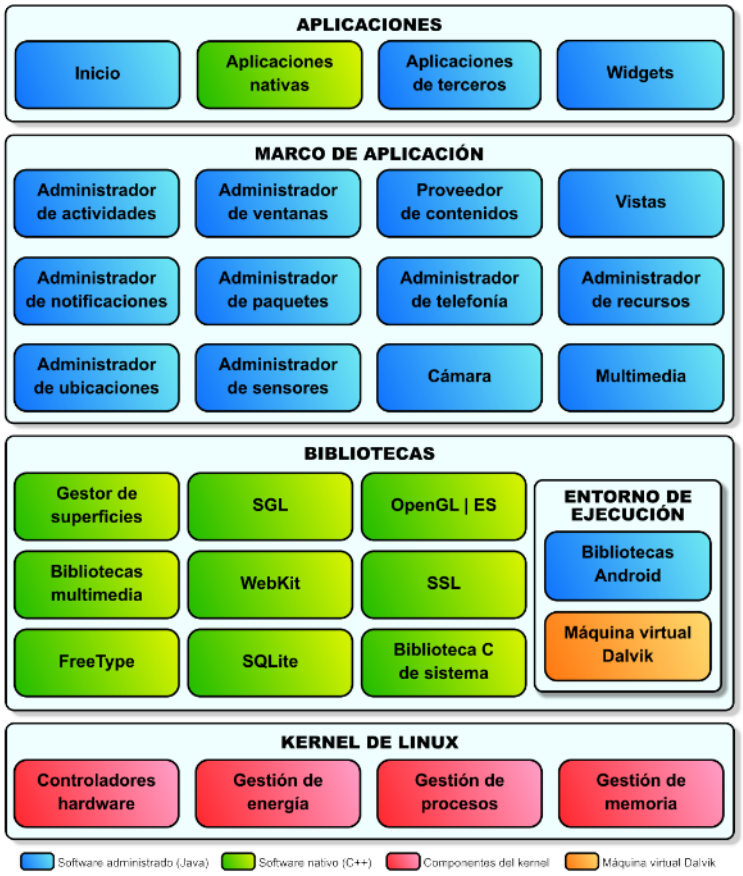

Figura 2.26. Arquitectura del sistema operativo Android.

2.4.2 iOS

iOS [13] es un sistema operativo móvil de Apple. Fue originalmente desarrollado para el Smartphone iPhone, siendo después usado en dispositivos como el iPod Touch, iPad y el Apple TV.

La interfaz de usuario de iOS está basada en el concepto de manipulación directa, usando gestos multitáctiles (multitouch). Los elementos de control son deslizadores, interruptores y botones. La respuesta a las órdenes del usuario es inmediata y provee de una interfaz fluida. La interacción con el sistema operativo incluye gestos como deslices, toques, pellizcos, los cuales tienen definiciones diferentes dependiendo del contexto de la interfaz.

iOS se deriva de Mac OS X, que a su vez está basado en Darwin BSD (plataforma de código abierto), que es un sistema operativo tipo Unix. iOS cuenta con cuatro capas de abstracción: la capa del núcleo del sistema operativo, la capa de "Servicios Personales", la capa de "Medios" y la capa de "Cocoa Tocuh". La versión actual del sistema operativo (iOS 9) ocupa más o menos 1.5 GB, en función del modelo.

2.4.3 Windows Phone

Windows Phone (abreviado WP) es un sistema operativo móvil desarrollado por Microsoft, como sucesor de Windows Mobile. A diferencia de su predecesor está enfocado en el mercado de consumo en lugar de en el mercado empresarial. Con Windows Phone Microsoft ofrece una nueva interfaz de usuario que integra varios de sus servicios propios como OneDrive, Skype y Xbox Live en el sistema operativo. Compite directamente contra Android de Google e iOS de Apple. Su última versión disponible y definitiva es Windows 8.1, lanzado el 14 de abril de 2014.

Debido a la evidente fragmentación de sus sistemas operativos, Microsoft anunció en enero de 2015 que dará de baja a Windows Phone, para enfocarse en un único sistema más versátil denominado Windows 10 mobile, disponible para todo tipo de plataformas (teléfonos inteligentes, tabletas y computadoras) qué a día de hoy aún no ha anunciado la fecha de lanzamiento.

2.4.4 Blackberry 10 OS

El BlackBerry 10 OS es un sistema operativo multitarea construido en el QNX Neutrino RTOS. El sistema operativo implementa la cantidad mínima de software en el espacio del núcleo y ejecuta otros procesos en el espacio de usuario. Mediante la ejecución de la mayoría de los procesos en el espacio de usuario, el BlackBerry OS 10 puede gestionar los procesos que no responden de forma aislada. Esta arquitectura crea un ambiente receptivo y seguro mediante la prevención de daños en el sistema operativo y otras aplicaciones en ejecución.

El BlackBerry OS 10 es resistente a la manipulación, seguro y eficiente:

- Resistente a las manipulaciones: El sistema operativo realiza una prueba de integridad antes de que comience. Si la prueba de integridad revela el daño, el sistema operativo no se inicia.
- Seguro: El sistema operativo valida las solicitudes de recursos del sistema. Un gestor da la autorización y evalúa las solicitudes de aplicaciones para acceder a las capacidades del sistema operativo. Por ejemplo, cuando una aplicación solicita acceso a la cámara, el gerente de autorización muestra un cuadro de diálogo que especifica la capacidad requerida, y ofrece al usuario la oportunidad de conceder o rechazar el acceso a esa capacidad. Además, el sistema operativo está diseñado para verificar la autenticidad de aplicaciones. Todas las aplicaciones deben estar firmadas por el BlackBerry.
- Eficiente: El SO minimiza el consumo de energía cuando está inactivo

2.6 Conclusiones

Tras estudiar todos los datos nos damos cuenta de que existen multitud de UAVs y cada uno posee su aplicación de control propia. Por ello, vemos necesario crear una aplicación que sea capaz de controlar cualquier tipo de UAV, independientemente de su lenguaje de control.

En nuestro caso usaremos el Drone de la compañía Parrot Ar-Drone 2.0, debido a su bajo coste y facilidad de manejo, en comparación con el resto de modelos estudiados. También usaremos el sistema Android, ya que es el que más usuarios poseen.

Capítulo 3

Descripción del sistema: materiales y métodos

3.1 Introducción

Tras analizar en el capítulo anterior las distintas opciones con las que podemos trabajar para llevar a cabo este proyecto, hemos seleccionado el Ar-Drone 2.0 de la compañía Parrot, el sistema Android y un ordenador portátil sobre el que correrá el middleware ROS.

En este capítulo se describen en profundidad los materiales, tanto físicos como a nivel de software, utilizados para la realización de este Trabajo de Fin de Grado.

Este capítulo está dividido en tres puntos principales, y en cada uno de estos puntos se explica un material utilizado en este Trabajo Final de Grado.

Las herramientas utilizadas vienen a ser las siguientes:

- Ar Drone 2.0 (Ver Figura 3.1): Es un UAV comercial de la casa Parrot, de estructura tipo cuadricóptero, que será el que pilotará de forma remota desde la aplicación.

  Este cuadricóptero tiene una serie de sensores y cámaras que nos aportarán la información del mismo y del entorno.

Figura 3.1. Ar Drone 2.0

- ROS (Robot Operating System): Es un meta-sistema operativo de código abierto para usar en robótica. Provee diferentes servicios que se esperarían de un sistema operativo, como abstracción de hardware, control de dispositivos a bajo nivel, funciones que se usan comúnmente, comunicación de procesos mediante mensajes y administración por paquetes. También incluye herramientas y librerías que permiten obtener, construir, escribir y ejecutar programas entre varios computadores [1].

  Este sistema funciona sobre un ordenador portátil. Y es el encargado de interpretar las órdenes recibidas de la aplicación Android y mandarlas al Drone, así como recabar la información del Drone y mandarla a la aplicación Android.

- Entorno Eclipse: Es una plataforma de desarrollo de software compuesto por un conjunto de herramientas de programación de código abierto para el desarrollo de diversas aplicaciones. En nuestro caso debemos usar el plugin SDK de Android para poder desarrollar aplicaciones para este SO.

3.2 Ar Drone 2.0

En este TFG controlaremos de forma remota el Ar Drone 2.0 de la compañía Parrot. Este cuadricóptero cuenta con distintos sensores que aportan todo tipo de información del vuelo.

En este apartado se comentarán las especificaciones de hardware propias del Ar Drone 2.0.

3.2.1 Características técnicas

A continuación se detallan las especificaciones técnicas y el hardware que permiten al Drone cumplir con todos los requisitos de un cuadricóptero. Estas especificaciones están clasificadas según su función.

3.2.1.1 Centro de control

Para que los parámetros de vuelo puedan ser modificados de forma autónoma, es necesario un dispositivo que controle y gestione todo de forma continuada. Para ello, el Drone dispone de un procesador ARM cortex A8 de un 1 GHz de frecuencia, que ejecuta el sistema operativo Linux 2.6.32. También posee 1 GB de memoria RAM DDR2 a 200 MHz.

El procesador realiza diversas funciones según la configuración del Drone, como puede ser, no sobrepasar de una altura fijada o realizar alguna acción concreta al recibir un comando.

3.2.1.2 Captura de datos

Este Drone posee varios sensores que captan los datos de diversas magnitudes. En concreto, viene equipado con los siguientes sensores:

- Giróscopo de 3 ejes: mediante este sensor el Drone puede conocer en todo momento su orientación. Tiene una precisión de 2000 °/s.

- Acelerómetro de 3 ejes: mediante este sensor el Drone puede medir la aceleración con la que se mueve, y así, calcular su velocidad. Tiene una precisión de +/- 50 mg.

- Magnetómetro de 3 ejes: mediante este sensor el Drone conoce su orientación exacta respecto al norte en todo momento.

- Sensor de presión: mediante este sensor el Drone es capaz de medir la altura a la que se encuentra. Tiene una precisión de +/- 10 Pa.

- Sensores de ultrasonidos (Ver Figura 3.2): mediante estos dos sensores ubicados en la parte baja del Drone, éste mide la distancia respecto al suelo. Sólo tienen un alcance de 6 metros. La función de aterrizaje automático es posible gracias a estos sensores.

Figura 3.2. Vista inferior del Ar Drone 2.0

- Cámara vertical QVGA: esta cámara permite medir la velocidad del Drone analizando el movimiento del Drone con respecto al suelo. Permite hasta 60 FPS.

3.2.1.3 Movimiento

Como en el resto de cuadricópteros, el Ar Drone posee 4 hélices controladas por un motor sin escobillas "brushless". Cada motor tiene un microcontrolador AVR de 8 MIPS, que tiene la labor de controlar el motor y de comunicarse con el procesador principal. Gracias a esto, el procesador puede enviar órdenes independientes a cada motor.

En la Figura 3.3 se puede observar un motor del Ar Drone 2.0 con su propio microcontrolador.

Estos motores son de 14,5 W y pueden girar hasta 28500 RPM.

Figura 3.3. Motor y controlador

La energía necesaria para el funcionamiento de los motores es suministrada por una batería recargable Li-Po de 1000 mAh (Ver Figura 3.4). Este batería, debido a su reducido tamaño y peso, es perfecta para el Drone, ya que no afecta demasiado a su estabilidad. En el mercado hay baterías para Ar Drone 2.0 con más capacidad, para un vuelo más largo. Con la batería de serie, el Drone puede efectuar un vuelo de unos 10 minutos.

Figura 3.4. Batería ArDrone

3.2.1.4 Captura de video

El Ar Drone 2.0 dispone de dos cámaras, una frontal y otra inferior. La cámara delantera es HD de 720p de resolución y permite capturar hasta 30 fps, la inferior es VGA de 480p a 60 fps. El vídeo capturado puede ser enviado vía Wi-Fi a la aplicación móvil y, además ser almacenado mediante USB.

3.2.1.5 Comunicación

Para poder efectuar las comunicaciones, el Drone crea su propia red inalámbrica Wi-Fi en modo Ad-hoc, por lo que el cliente sólo tiene que conectarse a dicha red y ya está listo para transmitir y recibir información.

Puesto que el Drone utiliza una red Wifi b/g/n la distancia máxima recomendable para operarlo es de 50 m, a partir de ahí puede ser muy inestable la conexión, y sería recomendable utilizar algún tipo de dispositivo range-extender.

3.2.1.6 Estructura

El Ar Drone 2.0 tiene una estructura en forma de cruz hecha de tubos de fibra de carbono (Ver Figura 3.5). Tiene un peso de 380 g con la carcasa exterior, y 420 g con la carcasa interior. En la zona de los sensores posee espuma para aislar las vibraciones producidas por los motores. También posee un revestimiento para repeler los líquidos de los sensores de ultrasonidos.

Figura 3.5. Estructura ArDrone

3.3 Robot Operating System

3.3.1 Introducción

Como ya se explicó en el Capítulo 2, ROS es un middleware para robots que provee una serie de servicios orientados a facilitar el desarrollo de software para robots, así como su operación. Entre los servicios ofrecidos presenta abstracción de hardware, control de bajo nivel, comunicación entre procesos, y manejo de paquetes de software. Además incluye herramientas y librerías para facilitar el desarrollo, compilación y ejecución de código de forma distribuida en diversas plataformas de procesamiento.

Figura 3.6. Logo ROS

ROS provee los servicios estándar de un sistema operativo, tales como abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Está basado en una arquitectura de grafos donde el procesamiento toma lugar en los nodos que pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones y actuadores, entre otros. La librería está orientada para un sistema UNIX (Linux), aunque también se está

adaptando a otros sistemas operativos como Fedora, Mac OS X, Arch, Debian o Microsoft Windows, considerados como plataformas 'experimentales'.

ROS tiene dos partes básicas: la parte del sistema operativo, ros, como se ha descrito anteriormente y ros-pkg, una suite de paquetes aportados por la contribución de usuarios (organizados en conjuntos llamados pilas o en inglés stacks), que implementan las funcionalidades, tales como localización y mapeo simultáneo, planificación, percepción, simulación, etc.

3.3.2 Conceptos previos

Para poder comprender como funciona ROS necesitamos explicar unos conceptos previos:

3.3.2.1 Nodos

Un nodo en realidad no es más que un archivo ejecutable dentro de un paquete de ROS. Los nodos de ROS utilizan una biblioteca cliente para comunicarse con otros nodos. Los nodos pueden publicar o suscribirse a un topic. Los nodos pueden utilizar o proporcionar algún servicio.

La biblioteca cliente de ROS permite escribir los nodos de ROS en diferentes lenguajes de programación:

- rospy = biblioteca cliente en python
- roscpp = biblioteca cliente en C++

Podemos usar la herramienta 'rosnode' en la línea de comandos para obtener diversa información acerca de los nodos. Los subcomandos más útiles son.

- Rosnode info <nombre del nodo>. Este comando muestra información acerca del nodo escrito, por ejemplo, donde está publicando y donde está subscrito.
- Rosnode list. Este comando muestra una lista de todos los nodos activos.

3.3.2.2 Topics

Los topics son canales de información entre los nodos. Un nodo puede emitir o suscribirse a un topic. El nodo que emite no controla quién está suscrito. La información es, por tanto, unidireccional (asíncrona). Si lo que queremos es una comunicación síncrona (petición/respuesta) debemos usar servicios (Ver Figura 3.7).

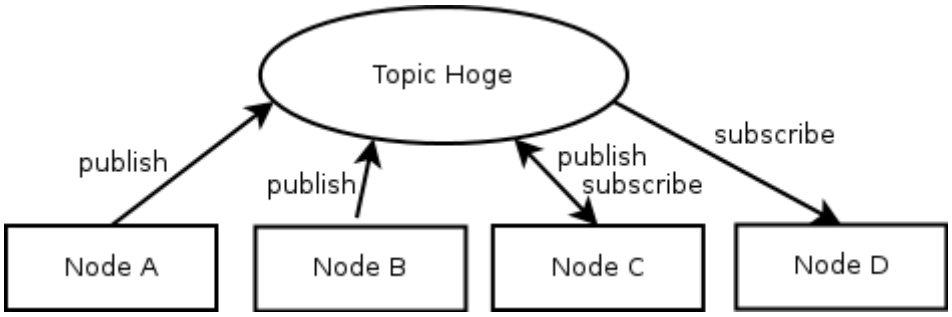

Figura 3.7. Esquema Topics

Por tanto, un topic no es más que un mensaje que se envía.

Al igual que con los nodos, también podemos usar la herramienta 'rostopic' en la línea de comandos para obtener diversa información acerca de un topic. Los subcomandos más útiles son:

- Rostopic echo <nombre del topic>. Muestra por pantalla los mensajes que está recibiendo el topic en cuestión.
- Rostopic info <nombre del topic>. Muestra por pantalla información acerca del topic en cuestión.
- Rostopic list <nombre del topic>. Muestra por pantalla una lista de todos los topics activos.
- Rostopic pub <nombre del topic> /<tipo de topic> [datos]. Para publicar datos en un topic. Por ejemplo, el siguiente comando:
  $ rostopic pub my_topic std_msgs/String "Hola mundo"
  Está publicando el mensaje 'Hola mundo', que es de tipo 'std_msgs/String' en el topic 'my_topic'.
- Rostopic type <nombre del topic>. Muestra por pantalla el tipo de topic que es muy útil para poder publicar mensajes en él.

3.3.2.3 Paquetes

El software en ROS está organizado en paquetes. Un paquete puede contener un nodo, una librería, conjunto de datos, o cualquier cosa que pueda constituir un módulo. Los paquetes pueden organizarse en pilas (stacks), siendo una pila un conjunto de nodos que proporcionan alguna funcionalidad.

Para crear un nuevo paquete con el que trabajar debemos seguir unos pasos, los cuales están explicados de manera detallada en la wiki de ROS, la cual se encuentra en el siguiente enlace: http://wiki.ros.org/ROS/Tutorials/CreatingPackage

3.3.2.4 Servicios

Los servicios en ROS son usados cuando es necesario enviar información y recibir una respuesta de los nodos, ya que esto no se puede realizar con los topics.

Un servicio está definido por un par de mensajes, uno para la petición y otro para la respuesta.

Un determinado nodo en ROS ofrece un servicio bajo un nombre, y el cliente llama al servicio enviando el mensaje de petición y esperando la respuesta.

Los servicios están definidos usando los archivos 'srv', los cuales son compilados en el código fuente de una biblioteca cliente de ROS.

Al igual que con los nodos y topics, también podemos usar la herramienta 'roservice' en la línea de comandos para obtener diversa información acerca de un servicio. Los subcomandos más útiles son:

- Rosservice call /<nombre del servicio> <argumentos del servicio>. Llama al servicio especificado con los argumentos apropiados.
- Rosservice find <Tipo del servicio>. Muestra por pantalla los servicios del tipo especificado.
- Rosservice list. Muestra por pantalla todos los servicios que estén actualmente disponibles.
- Rosservice info /<Nombre del servicio>. Muestra por pantalla la información relacionada con el servicio especificado.

La lista completa de subcomandos, con una explicación más detallada, e incluyendo ejemplos, se encuentra en la wiki oficial de ROS: http://wiki.ros.org/rosservice

3.3.3 Gazebo

3.3.3.1 Introducción

Gazebo es un potente y realista simulador 3D de robots y ambientes. Se pueden crear escenarios 3D e incluir robots propios, o de terceros, añadiendo también obstáculos y muchos otros objetos. Gazebo usa un motor físico para la iluminación, gravedad, inercia de los objetos, creando así una simulación que se adapta perfectamente a la realidad. Gazebo está pensado para evaluar y testear los robots creados en situaciones peligrosas o difíciles de reproducir en la vida real, sin ninguna consecuencia para el robot real.

Originalmente Gazebo fue diseñado para evaluar algoritmos para robots. Para muchas aplicaciones era necesario hacerle test al robot para ver cual iba a ser su comportamiento real, como manejo de errores, vida de las baterías, localización y navegación. Gazebo ha ido mejorando hasta llegar al simulador de hoy en día, capaz de simular varios robots a la vez interactuando entre ellos.

La instalación de los paquetes necesarios para que Gazebo funcione en la distribución de ROS está explicada paso a paso en la página oficial de Gazebo. (http://gazebosim.org/tutorials?tut=ros_installing&cat=connect_ros)

Una vez instalados los paquetes, ya se puede pasar a correr Gazebo, para ello hay que escribir los siguientes comandos:

```
roscore & rosrun Gazebo_ros Gazebo
```

Con esto comenzará a correr el roscore y posteriormente aparecerá la interfaz de Gazebo totalmente vacía (Ver Figura 3.8).

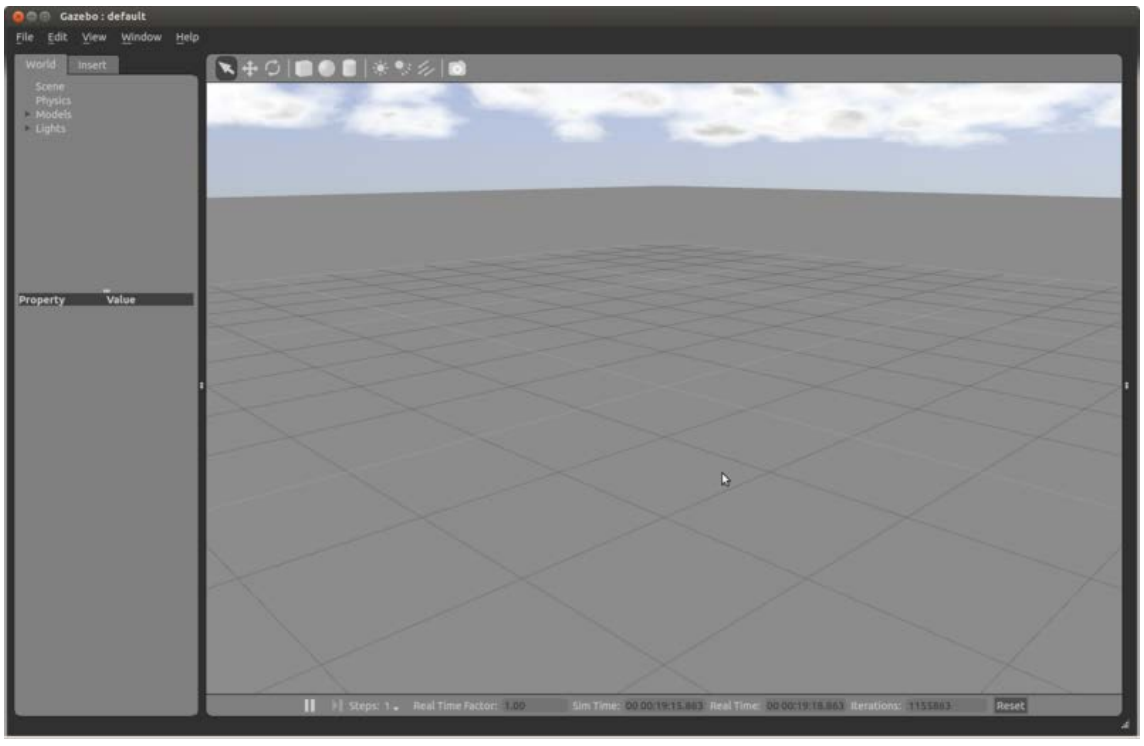

Figura 3.8. Gazebo

Para comprobar que Gazebo está conectado con ROS podemos escribir el siguiente comando:

```
rostopic list
```

Y deberían de aparecer varios topics relacionados con Gazebo, tal y como se muestra a continuación (Ver Figura 3.9).

```
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
```

Figura 3.9. Ejemplo topics

3.3.3.2 Ar-Drone en Gazebo

En este caso estamos trabajando con el Ar-Drone 2.0 de la compañía Parrot, el cual ya tiene un modelo 3D para Gazebo.

La instalación de los paquetes necesarios para usar este modelo es muy sencilla, y están explicados paso a paso en la siguiente dirección. http://wiki.ros.org/arDrone_autonomy

Este conjunto de paquetes nos va a permitir, además de poder usar el modelo del Ar Drone 2.0 con sus características reales, poder hacer un control sobre el modelo y sobre el Drone real al conectarlo a la misma red WiFi que el ordenador donde está corriendo ROS (Ver Figura 3.10).

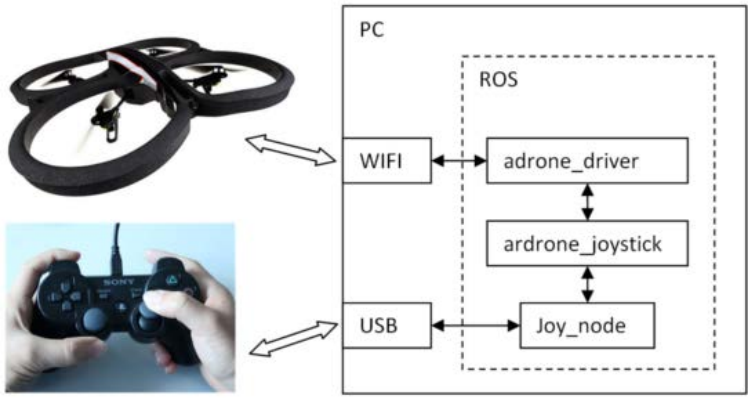

Figura 3.10. Ar Drone en Gazebo

3.3.4 RosBridge

Rosbridge es una librería de ROS con múltiples funcionalidades, entre ellas, incluye un servidor WebSocket para interactuar con ROS mediante un navegador Web.

Para hacer uso de esta librería simplemente hay que introducir en la consola el siguiente comando:

```
sudo apt-get install ros-<rosdistro>-rosbridge-suite
```

Donde debemos reemplazar '<rosdistro>' por la distribución de ROS que estemos usando, en nuestro caso indigo.

Después de instalar estos paquetes debemos asegurarnos que el sistema es consciente de su instalación, para eso, configuramos el entorno mediante el siguiente comando en la línea de comandos:

```
source /opt/ros/<rosdistro>/setup.bash
```

Ahora, todo lo que queda es ejecutar rosbridge. Para iniciar rosbridge y sus paquetes sólo hay que lanzar un archivo con el siguiente comando:

```
roslaunch rosbridge_server rosbridge_websocket.launch
```

Esto lanzará rosbridge y creará un WebSocket en el puerto 9090 por defecto. Este puerto puede ser configurado modificando el archivo '`rosbridge_websocket.launch`'.

3.3.5 Mjpeg Server

Podemos comprobar el estado de nuestro robot en tiempo real mediante vídeo o mediante capturas de foto. Esto es gracias a un servidor en streaming llamado MJPEG_server, que lo que hace es subscribirse al topic de la imagen solicitada en ROS y publicar este topic como MJPEG vía HTTP. El formato MJPEG es una sucesión de archivos JPEGs. En concreto, es un formato en el que cada frame del vídeo a transmitir es comprimido por separado como una imagen en JPEG y enviado en formato binario. Esto hace que sea una forma eficiente y optimizada para el transporte de un archivo de vídeo en tiempo real.

La instalación del paquete necesaria para usar este nodo es muy sencilla, sólo es necesario ejecutar lo siguiente en la línea de comandos:

```
sudo apt-get install ros-<rosdistro>-mjpeg-server
```

El paso siguiente a este es ejecutar el nodo. Para ello abrimos un nuevo terminal y debemos correr el mjpeg_server, ejecutando el siguiente comando en la línea de comandos:

```
rosrun mjpeg_server mjpeg_server
```

Por defecto el servidor empezará a funcionar en el puerto 8080, y si este puerto ya está en uso por otro proceso se mostrará un mensaje de error. Si es necesario, puedes especificar el puerto que quieres usar al lanzar el mjpeg_server de la siguiente manera:

```
rosrun mjpeg_server mjpeg_server _port:=8181
```

Una vez esté el nodo corriendo, ya podemos visualizar el streaming mediante cualquier navegador web. Si el servidor está corriendo en una red local, podemos visualizar el streaming accediendo a la siguiente URL:

```
http://localhost:8080/snapshot?topic=/IMAGE_TOPIC
```

Donde 'IMAGE_TOPIC' es el topic a visualizar.

Este paquete posee más características y parámetros personalizables, los cuales están todos explicados en la wiki oficial: http://wiki.ros.org/mjpeg_server

3.3.6 Conjunto de paquetes

En nuestro caso, el conjunto de paquetes utilizados e interconectados entre si queda resumido en el siguiente esquema (Ver Figura 3.11).

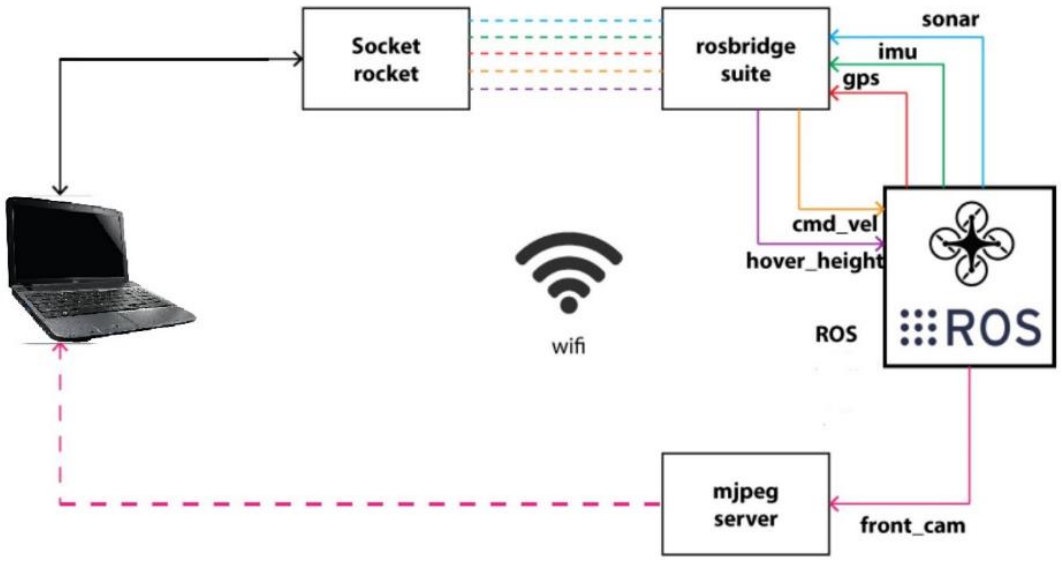

Figura 3.11. Esquema del conjunto.

Por un lado, tenemos el sistema ROS corriendo sobre el ordenador portátil. Mediante el paquete de arDrone_autonomy se recogen los datos del Drone real, como la velocidad actual, altura, posición GPS y se envían mediante el paquete de Rosbridge, donde los datos son transformados para poder ser recibidos desde un navegador cualquiera en un ordenador portátil o en la aplicación desarrollada para Android. A la vez que esto, también se está capturando la cámara delantera del Drone y esa información es recogida por el paquete mjpeg_server, que transforma las imágenes para poder ser enviadas y visualizadas como un vídeo en streaming desde un navegador.

3.4 Eclipse

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ), que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse fue liberado originalmente bajo la Common Public License, pero después fue re-licenciado bajo la Eclipse Public License. La Free Software Foundation ha dicho que ambas licencias son licencias de software libre, pero son incompatibles con Licencia pública general de GNU (GNU GPL).

La interfaz que presenta el software Eclipse mientras se está trabajando con un proyecto es la siguiente (Ver Figura 3.12).

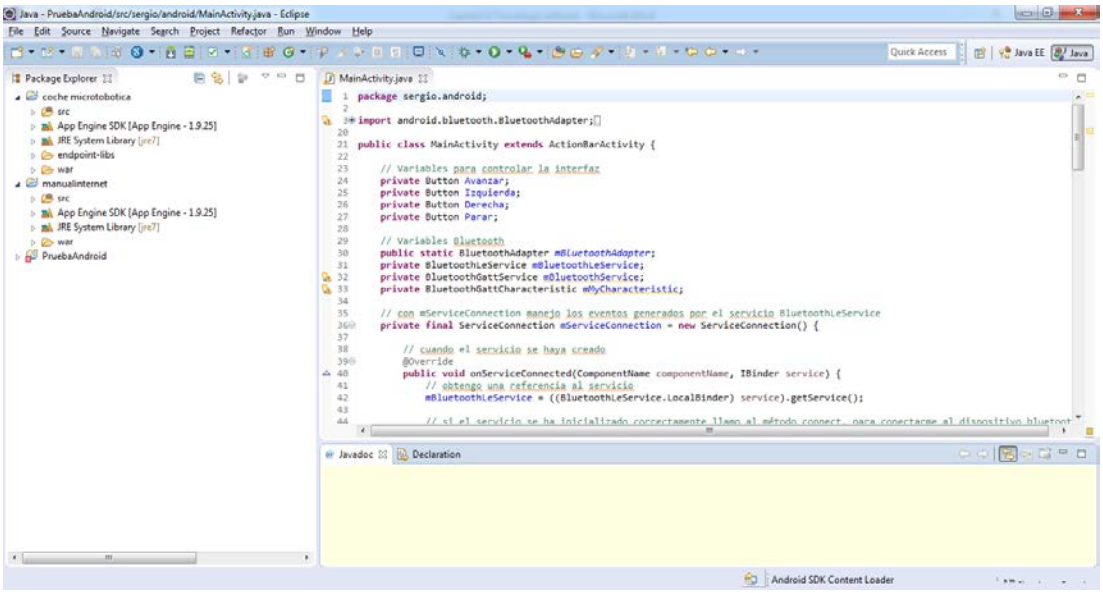

Figura 3.12. Interfaz Eclipse

No obstante, la interfaz que ofrece el software al ejecutar el programa una vez arrancado en nuestro ordenador es la siguiente (Ver Figura 3.13).

Figura 3.13. Pantalla principal Eclipse

A la hora de crear un nuevo proyecto, basta con pulsar en el menú superior sobre "File" -> new -> y seleccionar el tipo de proyecto que queremos generar. Al hacer estos pasos se despliega una variedad de tipos de proyectos más comunes, pero si el proyecto buscado no se encuentra entre esa lista, al final de esta se encuentra la opción de "Others" y al pulsar sobre ella se abre un "wizard" para la selección de todo tipo de proyectos que soporta el software Eclipse.

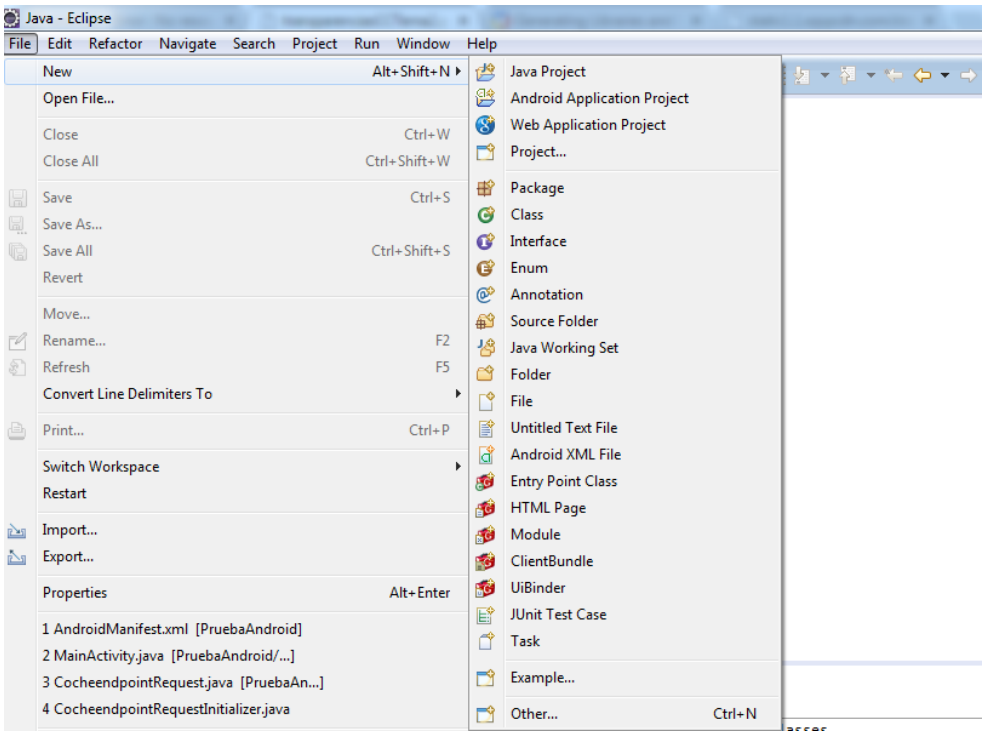

Figura 3.14. Creación de nuevo proyecto

En nuestro caso usamos el tipo "Android Application Project", pero para que todo funcione correctamente es necesario instalar el plugin "SDK de Android". Llevar esto a cabo es tan sencillo como descargarlo desde la página oficial, https://developer.android.com/studio/tools/sdk/eclipse-adt.html, e instalarlo como cualquier otro programa.

Una vez realizado estos pasos, ya estarán disponibles todas las herramientas para programar para el sistema Android, así como un simulador para probarlas.

3.4.1 Estructura de un proyecto Android

Para poder hacerse una mejor idea de la estructura que tiene el proyecto es necesario realizar una modificación en Eclipse. Dicho cambio implica pulsar en la lista desplegable situada en la parte superior izquierda, y cambiar de "Android" a "Project". En la nueva jerarquía lo primero que aparece es una carpeta con el nombre del proyecto, y otras carpetas que representan los distintos módulos de la aplicación.

La organización de los archivos en un proyecto Android es la siguiente (Ver Figura 3.15).

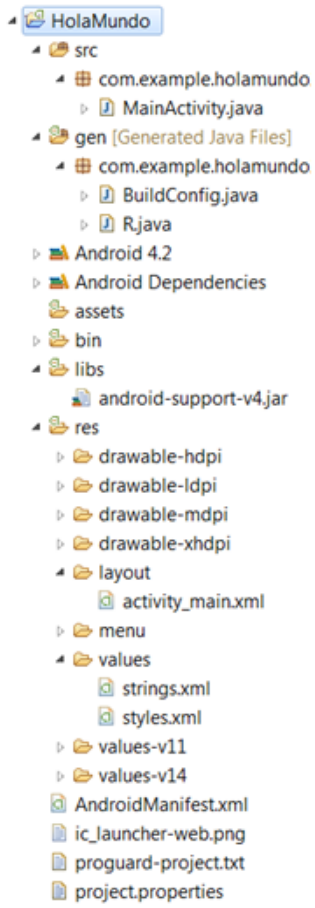

Figura 3.15. Organización proyecto Android

En la carpeta *src* es donde está guardado el código fuente de la aplicación, es decir, los archivos .java.

La carpeta *gen* es la que contiene el código generado de forma automática por el SDK de Android. Nunca debemos modificar estos archivos manualmente. Dentro de esta carpeta observamos dos archivos:

- BuildConfig.java: Define la constante *DEBUG* para que desde Java puedas saber si la aplicación está en fase de desarrollo.
- R.java: Define una clase que asocia los recursos de la aplicación con identificadores. De esta forma los recursos podrán ser accedidos desde Java.

La carpeta *Android X.X* contiene el código JAR, el API de Android según la versión seleccionada.

La carpeta *Android Dependencies* contiene las librerías asociadas al proyecto.

En la carpeta *bin* es donde se compila el código y se genera posteriormente el .apk, que es el archivo final para instalar la aplicación en un dispositivo Android.

La carpeta *res* contiene los recursos usados por la aplicación, esta carpeta está dividida en subcarpetas:

- *Dr*awable: En esta carpeta se almacenan los ficheros de imágenes (JPG o PNG) y descriptores de imágenes en XML.
- *Layout*: Contiene ficheros XML con vistas de la aplicación. Las vistas nos permitirán configurar las diferentes pantallas que compondrán la interfaz de usuario de la aplicación. Se utiliza un formato similar al HTML usado para diseñar páginas Web.
- *Menu*: Contiene los ficheros XML con los menús de cada actividad.
- *Values*: También utilizaremos ficheros XML para indicar valores del tipo string, color o estilo. De esta manera podremos cambiar los valores sin necesidad de ir al código fuente. Por ejemplo, nos permitiría traducir una aplicación a otro idioma.
- *AndroidManifest.xml*: Este fichero describe la aplicación Android. En él se indican las actividades, intenciones, servicios y proveedores de contenido de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica la versión mínima de Android para poder ejecutarla, el paquete Java, la versión de la aplicación, etc.

Cuando se inicie una aplicación de Android, el primer fichero que se ejecutará será el MainActivity.java, por lo que se debe configurar adecuadamente. Dado que la clase *MainActivity* extiende de una clase *Activity*, su ciclo de vida es el mismo que el de cualquier Activity en Android (Ver Figura 3.16).

El primer método que se ejecuta cuando se crea una *Activity* es el método "onCreate()", por lo que en este método se debe establecer el archivo .xml del que se obtendrá la apariencia de nuestra Activity, así como obtener la referencia a los distintos elementos que se hayan situado en la pantalla.

A su vez, los métodos *"onStart()", "on*Resume*()", "onPause()", "onStop()*" y *"onDestroy"* permiten realizar distintas acciones en función de los procesos por los que vaya pasando una Activity, ya que puede ser que se cambie de una Activity a otra, que se cierre la aplicación, o

que posteriormente se vuelva a la misma Activity. Por lo tanto, con los métodos listados anteriormente, se pueden procesar adecuadamente todo este tipo de situaciones.

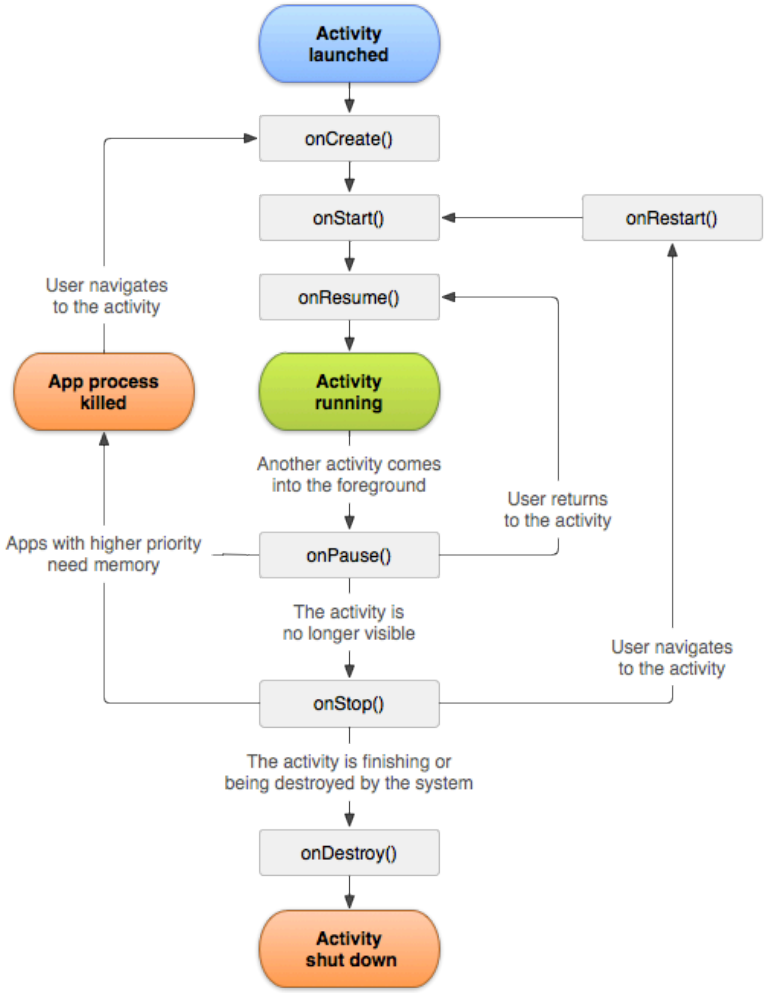

Figura 3.16. Ciclo de vida de una Activity en Android

3.4.2 Instalar la aplicación en un dispositivo Android

Una vez creada la aplicación, se puede realizar la compilación e instalación en un dispositivo con Android para probarla y detectar posibles fallos.

Para ello, lo primero es haber configurado correctamente el dispositivo móvil con Android para poder depurar aplicaciones mediante USB, y conectarlo mediante el cable USB al ordenador. A continuación, se pulsa sobre el botón de la barra superior "Run 'app'", tras lo cual Eclipse comenzará a compilar el proyecto, hasta que muestre una ventana como la siguiente (Ver Figura 3.17). En esta ventana se puede elegir entre ejecutar la aplicación en un dispositivo virtual, o instalarla en un dispositivo real. Por lo que, en este trabajo se elegirá el dispositivo que aparece en la lista.

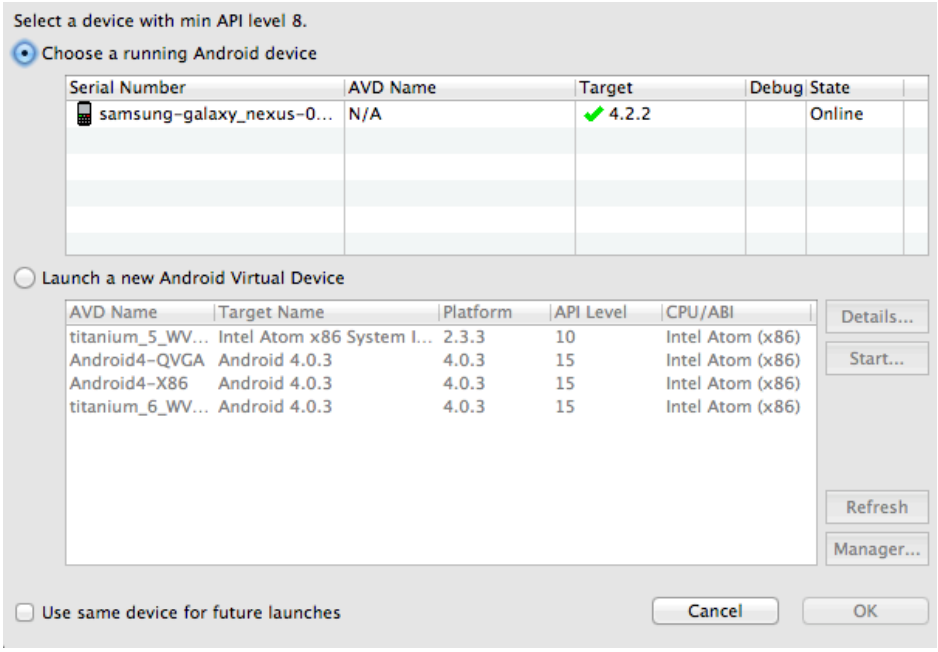

Figura 3.17. Dispositivos para instalar la aplicación

Cuando la aplicación se haya instalado y esté en ejecución, en la parte inferior de Eclipse se podrá hacer uso de la pestaña llamada "Android Monitor", dentro de la cual se seleccionará la pestaña "logcat" y visualizar en tiempo real la información relativa al dispositivo Android, por lo que si la aplicación experimenta algún problema, se verá reflejado en este lugar y será posible detectar el fallo.

3.5 Conclusiones

Tras haber descrito las distintas herramientas utilizadas para desarrollar este trabajo, en esta sección se van a recapitular los distintos elementos que son necesarios para llevar a cabo el desarrollo del sistema usado en este Trabajo Fin de Grado.

En primer lugar debemos tener corriendo el sistema ROS en un ordenador. Sobre este sistema estará lanzado el paquete "Rosbridge", encargado de interpretar las órdenes recibidas de la aplicación Android y mandarlas al Drone en el formato correcto. También es necesario el paquete "arDrone_autonomy", que son los drivers necesarios para el control del Ar Drone 2.0 a través de ROS.

Por otro lado, debemos tener la aplicación de control, corriendo sobre un sistema Android. Esta aplicación tendrá las órdenes básicas de control de un Drone, y será independiente del Drone a controlar, ya que el encargado de traducir estas órdenes para que el Drone las interprete correctamente es el sistema ROS.

Capítulo 4

Descripción del Hardware y Software Desarrollado

4.1 Introducción

En el capítulo anterior se describió tanto el hardware como el software necesario para la realización de este Trabajo Final de Grado.

Es este capítulo se detallará la implementación del hardware y el software desarrollado para culminar el estudio, diseño y desarrollo del sistema de control de un UAV a través de un dispositivo móvil inteligente.

4.2 Software

4.3.1 Introducción

En este capítulo se describirá la aplicación Android realizada para permitir que el UAV pueda ser contralado desde un dispositivo móvil inteligente, así como la aplicación web diseñada para el control del dron a través de un navegador Web.

4.3.2 Aplicación Android

La aplicación Android es muy sencilla, consta solamente de una vista con los controles del dron (ver Figura 4.1), tanto para manejarlo en todas direcciones como para dar las órdenes de despegar y aterrizar. Para llevar a cabo todas estas tareas, es necesario que se realicen diversas acciones en el entorno ROS, tales como suscribirse y darse de baja de topics. Estas funcionalidades han sido adecuadamente implementadas, probadas y depuradas.

Figura 4.1. Aplicación Android

4.3.2.1 Código Fuente Android

En esta sección se describen los componentes software que se han diseñado para la aplicación Android. Sólo ha sido necesario implementar una actividad para proporcionar todos los elementos de interacción al usuario.

Las funcionalidades de la aplicación Android son las siguientes:

- Conectar el dispositivo Android a la red ROS.
- Llevar a cabo tareas de suscripción a un topic concreto, así como posibilitar la acción contraria (dar de baja la suscripción realizada).
- Enviar las órdenes de control al dron.
- Visualizar los datos del topic al que se está suscrito.

A continuación, se describe la estructura de la aplicación Android, así como alguno de los métodos utilizados en la dicha aplicación. Finalmente, se describen algunas de las funcionalidades, en detalle.

En el Listado 4-1 se muestra la estructura de la aplicación Android.

```
public class MainActivity extends Activity {

	...

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
			...
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		getMenuInflater().inflate(R.menu.main, menu);
		return true;
	}

	public void subscribe(View v) {
		...
	}

	public void unsubscribe(View v) {
		...
	}
	public void despegar(View v){
		...
	}
	public void despegando() {
		...
	}
	public void aterrizar(View v){
		...
	}
	public void aterrizando() {
		...
	}
	public void go(){
		...
	}
	public void adelante(View v){
		...
	}
	public void atras(View v){
		...
	}
	public void girarD(View v){
	...
	}
	public void girarI(View v){
		...
	}
```

```
	public void stop(View v){
		...
	}
	public void derecha(View v){
		...
	}
	public void izquierda(View v){
		...
	}
	public void subir(View v){
	...
	}
	public void bajar(View v){
	...
	}
```

Listado 4-1: Estructura Aplicación Android

La clase creada por defecto para una aplicación Android extiende de un "Activity", en nuestro caso extiende de la clase "Activity" que nos proporciona Android para crear actividades con sus métodos asignados.

A continuación, el método "onCreate" es ejecutado por el sistema cuando se produce el evento de creación de la actividad. Si la actividad ya había sido creada y ciertos datos de inicialización están disponibles, se pueden conservar cuando la actividad se recrea a través del objeto "savedInstanceState". Nótese que, en la aplicación propuesta, esto no es necesario. Por otra parte, cuando se crea la actividad, es necesario asociarla con la interfaz de usuario descrita en tiempo de diseño (a través del archivo XML correspondiente). Para ello, se llama al método "setContentView".

En el Listado 4-2 se muestra el método "onCreate" de la aplicación completo.

```
protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
		try {
			uri = new URI("ws://192.168.1.40:9090");
			ws = new MyWS(uri);
			ws.setA(this);
			Tarea t = new Tarea();
			t.setWs(ws);
			t.setA(this);
			t.execute();
		} catch (URISyntaxException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
```

Listado 4-2: Método onCreate

Cuando se produce el evento de creación, es necesario también definir una URI (Identificador de Recursos Uniforme) con la dirección IP del dispositivo en el que se estará ejecutando el nodo roscore de ROS. Por defecto, el puerto utilizado, en este caso, es "9090", que es el predeterminado cuando se lanza el paquete rosbridge para la comunicación mediante WebSockets con aplicaciones que no utilizan el framework de ROS directamente (*non-ROS applications*). El objeto URI es pasado al constructor de la clase "MyWS", para crear un nuevo objeto que se encargará de manejar el websocket abierto entre la aplicación y rosbridge. La apertura del websocket se hace dentro de una tarea asíncrona, para asegurar que el hilo principal no va a quedar bloqueado si se produce algún problema con la red.

En el Listado 4-3 se muestra parte de la clase "MyWS" de la aplicación.

```
public class MyWS extends WebSocketClient{

	private Activity a;

	public Activity getA() {
		return a;
	}

	public void setA(Activity a) {
		this.a = a;
	}

	public MyWS(URI serverURI) {
		super(serverURI);
	}

		...

	@Override
	public void onMessage(String arg0) {
		try {
			JSONObject _m = new JSONObject(arg0);
			JSONObject msg = _m.getJSONObject("msg");
			final String response = msg.getString("data");
			a.runOnUiThread(new Runnable(){

				@Override
				public void run() {

	((TextView)a.findViewById(R.id.textView1)).setText(response);

			...
```

Listado 4-3: Clase MyWS

Como ya se ha comentado anteriormente, la clase "MyWS" define los manejadores de eventos que se pueden producir cuando se intercambia la información a través del websocket. En particular, una de las acciones más importante desde el punto de vista de la aplicación propuesta es la de recibir mensajes de ROS a través de Rosbridge y transformar la información, adecuadamente, para hacerla visible mediante un 'TextView' en la aplicación Android. Obsérvese que, se ha de asegurar que la modificación del textview se realiza en el hilo principal (también llamado hilo de la interfaz de usuario). Por esta razón se utiliza el método runOnUIThread, que permite asignar una sección de código al hilo principal. La sección de código asignada es la que aparece dentro del método run.

En el Listado 4-4 se muestra parte de la clase "Tarea" de la aplicación.

```
public class Tarea extends AsyncTask<Void, Void, Boolean>{

	private MyWS ws;
	private Activity a;

	@Override
	protected Boolean doInBackground(Void... params) {
		try {
			ws.connectBlocking();
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
			return false;
		}
		return true;
	}

	@Override
	public void onPostExecute(Boolean result){
		if (result.booleanValue()){
			Toast.makeText(a, "Conectado", Toast.LENGTH_SHORT).show();
		}
		else{
			Toast.makeText(a, "Fallo de conexiÃ³n",
Toast.LENGTH_SHORT).show();
		}

	}


	...
```

Listado 4-4: Clase Tarea

La clase "Tarea" implementa una tarea asíncrona que permite abrir el websocket. De este modo se conecta la aplicación Android con la red ROS a través de rosbridge. El mensaje "Conectado" se muestra cuando se ha llevado a cabo la conexión de forma correcta.

En el Listado 4-5 se muestra parte del método "subscribe" de la aplicación. Este método implementa el manejador del evento "onclick" de un botón.

```
public void subscribe(View v) {
             JSONObject msusc = new JSONObject();
             topic = "/sonar_height/range";
             typetopic = "/sensor_msgs/Range";
             try {
                    msusc.put("op", "subscribe");
                    msusc.put("topic", topic);
                    msusc.put("type", typetopic);
                    ws.send(msusc.toString());
                    Toast.makeText(this, "Suscrito",
Toast.LENGTH_SHORT).show();
```

Listado 4-5: Método subscribe

El método "subscribe" es el encargado de suscribirse al topic indicado, en este caso se suscribiría al topic encargado de recoger el dato del ultrasonidos inferior del dron.

Primero se crea un objeto Json y se van colocando los campos necesarios, según el protocolo dado por rosbridge, para que la información pueda ser interpretada por ROS. Una vez que el objeto está completo con toda la información, se transforma a un String y se manda a la dirección indicada mediante WebSocket. Finalmente, se muestra en la pantalla el mensaje "Suscrito", para dar a conocer esta información al usuario.

El método "unsubscribe" tiene la misma estructura que éste, solamente cambia el campo "op" de "subscribe" a "unsubscribe".

En el Listado 4-6 se muestra parte del método "despegar" de la aplicación.

```
public void despegar(View v){

             try {
                    JSONObject mdesp = new JSONObject();
                    if (mdesp != null){
                           mdesp.put("op", "advertise");
                           mdesp.put("topic", "/arDrone/takeoff");
                           mdesp.put("type", "/std_msgs/Empty");
                           ws.send(mdesp.toString());
                           despegando();
                           Toast.makeText(this, "Despegando",
Toast.LENGTH_SHORT).show();
                    }

             ...
```

Listado 4-6: Método despegar

La función del método "despegar" se corresponde con una operación de tipo "advertise" en ROS, ya que para que el drone despegue, es necesario enviar un mensaje vacío al topic /arDrone/takeoff. La operación de suscripción es necesaria antes de publicar un mensaje en cualquier topic.

Para llevar la operación a cabo primero se crea un objeto JSON según el protocolo de Rosbridge, en el que se incluye el tipo de operación a llevar a cabo, el topic sobre el que se va a publicar posteriormente y el tipo de topic. Por último, se convierte el objeto JSON a String y se envía mediante WebSocket. A continuación, se ejecuta el método "despegando", que es el encargado de publicar el mensaje correspondiente en el topic sobre el que se ha hecho el "advertise" para que se produzca realmente el despegue.

En el caso de que se desee aterrizar, se usa el método "aterrizar" para hacer el "advertise" y éste ejecuta el método "aterrizando" para publicar el mensaje correspondiente en el topic adecuado. La estructura de estos dos métodos es la misma que los utilizados para despegar.

En el Listado 4-7 se muestra parte del método "despegando" de la aplicación.

```
public void despegando() {

            JSONObject msg = new JSONObject();
            try {
                  if (msg != null){

                              msg.put("op", "publish");
                              msg.put("topic", "/arDrone/takeoff");
                              ws.send(msg.toString());
                  }

            ...
```

Listado 4-7: Método despegando

Para el control del dron, cada botón puede producir un evento onclick que tiene asignado un método que maneja dicho evento. Cada método ejecuta una acción distinta mediante la configuración de los parámetros de velocidad lineal y velocidad angular. Una vez asignadas las velocidades se invoca el método "go", que es el encargado de empaquetar la consigna utilizando la clase "JSONMessageCMD_VEL.java". El empaquetamiento de la consigna incluye la creación de un mensaje JSON compatible con el protocolo rosbridge. El objeto JSON es transformado a un String que se envía a través del websocket.

En el Listado 4-8 se muestra el método "adelante" de la aplicación, que es el encargado de asignar los valores a las variables correspondientes para que el dron avance.

```
public void adelante(View v){
            omega = 0;
            velocidad = 1;
            velocidadY = 0;
            velocidadZ = 0;
            go();
      }
```

Listado 4-8: Método adelante

En esta parte del código se le asigna una velocidad de "1" a la velocidad lineal en la dirección X, y "0" en el resto de velocidades. Luego, se ejecuta el método "go".

El resto de métodos para controlar las distintas direcciones del dron tienen la misma estructura que éste, solo cambia el valor de las variables.

En el Listado 4-9 se muestra el método "go" de la aplicación.

```
public void go(){
            JSONMessageCMD_VEL msgcmdvel = new JSONMessageCMD_VEL();
            msgcmdvel.setVx(velocidad);
            msgcmdvel.setVy(velocidadY);
            msgcmdvel.setVz(velocidadZ);
            msgcmdvel.setWx(omega);
            JSONObject msg = msgcmdvel.encode_JSON();
            String msgstring = msg.toString();
            ws.send(msgstring);
            Toast.makeText(this, "Vamos", Toast.LENGTH_SHORT).show();
      }
```

Listado 4-9: Método go

4.3.3 Código HTML

Además de la aplicación Android para el control de dron mediante un dispositivo móvil, también se ha desarrollado una aplicación Web en HTML para poder controlarlo desde un navegador en cualquier ordenador. De este modo, es posible tele-operar el dron desde, prácticamente, cualquier dispositivo y no solo desde uno que incluya Android como sistema operativo.

Esta aplicación es muy sencilla, está compuesta por dos archivos, uno que con código en HTML y otro archivo ".js" con código Javascript, encargado de facilitar la comunicación con Rosbridge.

En el Listado 4.1 se muestra el código HTML de la aplicación.

```
<html>
<head>
<title>Control Drone mediante Rosbridge </title>
</head>
<body>

<script type="text/javascript" src="ros.js"></script>
<script type="text/javascript">
  // ---------------publicar en topics ----------------------
    var con = new Bridge("ws://192.168.1.34:9090");




    function takeoff() {
       con.publish('/arDrone/takeoff', { });
          //con.subscribe(,'/cmd_vel');
    }
    function land() {
       con.publish('/arDrone/land', { });
    }
    function start_moving(keycode) {
              console.log("Starting to move, keycode: " + keycode);
              if (keycode == 38) {
                    // Adelante
                    con.publish('/cmd_vel',
{"linear":{"x":1.0,"y":0,"z":0},"angular":{"x":0,"y":0,"z":0}});
              }
else if (keycode == 87) {
                    // SUBIR
                    con.publish('/cmd_vel',
{"linear":{"x":0.0,"y":0,"z":3.0},"angular":{"x":0,"y":0,"z":0.0}});
              }

else if (keycode == 83) {
                    // BAJAR
                    con.publish('/cmd_vel', {"linear":{"x":0.0,"y":0,"z":-
3.0},"angular":{"x":0,"y":0,"z":0.0}});
              }

else if (keycode == 40) {
                    // AtrÃ¡s
                    con.publish('/cmd_vel', {"linear":{"x":-
1.0,"y":0,"z":0},"angular":{"x":0,"y":0,"z":0}});
              } else if (keycode == 39) {
                    // Derecha
                    con.publish('/cmd_vel',
{"linear":{"x":0,"y":0,"z":0},"angular":{"x":0,"y":0,"z":-1.0}});
              } else if (keycode == 37) {
```

```
                    // Izquierda
                    con.publish('/cmd_vel',
{"linear":{"x":0,"y":0,"z":0},"angular":{"x":0,"y":0,"z":1.0}});
                } else if (keycode == 13) {
                takeoff();
            } else if (keycode == 32) {
                land();
            }
    }



    function stop_moving(keycode) {
                console.log("ELECTRIC Done moving, keycode: " + keycode);
                if (keycode >= 37 && keycode <= 40) {
                    con.publish('/cmd_vel',
{"linear":{"x":0,"y":0,"z":0},"angular":{"x":0,"y":0,"z":0}});
                }
    }
    function main() {
            document.addEventListener('keydown', function(e) {start_moving(e.keyCode); },
true);
            document.addEventListener('keyup', function(e) {stop_moving(e.keyCode); },
true);
    }
    main();
</script>

<h1>Control ArDrone
<p>Antonio Pastor Conesa
</h1>

DESPEGAR CON ENTER Y ATERRIZAR CON ESPACIO
<p>
W PARA SUBIR
<p>
S PARA BAJAR

<p>

<img src="http://192.168.0.103:8080/stream?topic=/arDrone/image_raw">



<p>
</body>
</html>
```

Listado 4.10: Código HTML

Este código es muy sencillo, ya que usa funciones ya creadas incluidas en el archivo".js". Lo primero que hacemos es enlazar este código con dicho archivo. Acto

seguimos establecemos la dirección IP que nos dará Roscore para poder establecer la comunicación. Posteriormente, se crean las funciones para despegar, aterrizar y controlar el dron a traves del teclado del ordenador. Por último, se introduce la direción URL, donde el dron está emitiendo las imágenes de la cámara frontal, dentro de un visualizador de imágenes.

La visualización de este código es la siguiente (Ver Figura 4.2).

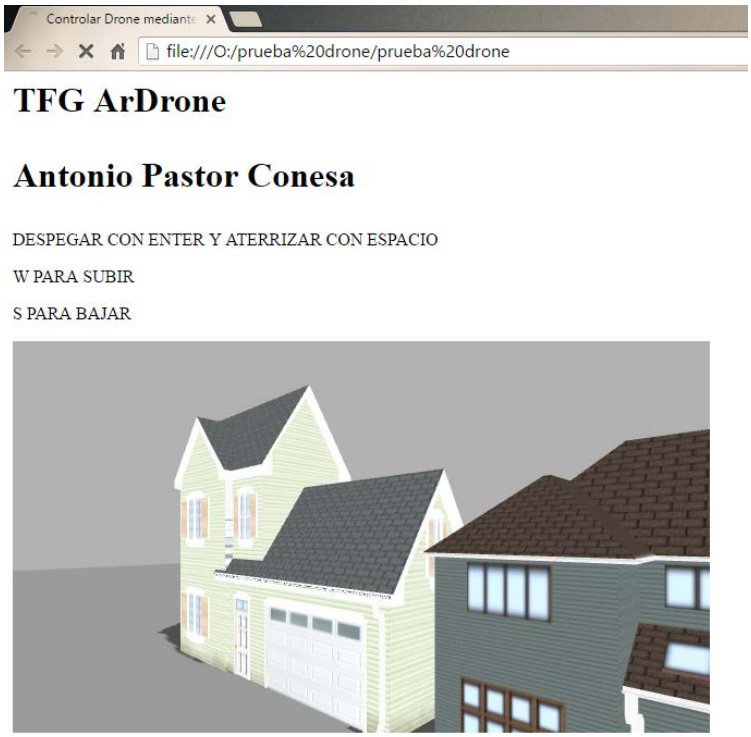

Figura 4.2. Aplicación Web

4.3 Conclusiones

En este capítulo, se han presentado y explicado, en detalle, desde los pasos necesarios para la instalación de las herramientas necesarias, hasta la programación que conlleva el diseño y la implementación de una aplicación de estas características. Además, se ha mostrado parte del código fuente de las aplicaciones Android y Web desarrolladas para llevar a cabo el control del drone desde un dispositivo móvil inteligente con Android como sistema operativo, y desde cualquier otro dispositivo que incorpore un navegador capaz de interpretar HMTL y Javascript.

Capítulo 5

Conclusiones y trabajos futuros

5.1 Conclusiones

En este proyecto se ha llevado a cabo el estudio de un sistema que debe ser capaz de controlar cualquier UAV compatible con el driver ROS ardrone_autonomy,.

Esto es posible gracias al sistema ROS, que puede ejecutarse en un ordenador con sistema operativo Linux. El nodo ROS roscore es el encargado de centralizar las comunicaciones dentro de la red ROS. Además, se ha utilizado Rosbridge para permitir la interacción con aplicaciones que no pueden conectarse a la red ROS de forma directa.

En el trabajo presentado se ha desarrollado una aplicación Android y una aplicación Web capaces de teleoperar y controlar un drone mediante el uso de ROS. Dichas aplicaciones se han probado usando el simulador Gazebo, junto con su driver ROS que simula el dron en un entorno virtual y, posteriormente con el UAV real Ar-Drone 2.0 de la compañía Parrot. El sistema propuesto podría funcionar con cualquier otro drone que posea los drivers adecuados y sea compatible con el driver ardrone_autonomy. También es posible crear, o incluso adaptar, un nuevo driver para otro tipo de UAVs. En este caso, se sustituiría el driver ardrone_autonomy por el nuevo driver adaptado al drone específico.

Finalmente, y a modo de resumen, con este estudio se ha pretendido comprobar la gran utilidad del sistema ROS para el control de todo tipo de robots, gracias a que posee un lenguaje de muy alto nivel y no importa cuál sea el robot a controlar. Todo esto se ha llevado a cabo simplemente con un ordenador portátil y un Smartphone, por lo que es muy barato y fácil de recrear.

5.2 Trabajos futuros

Como posible trabajo futuro, se contempla la inclusión de un SBC, como la Raspberry Pi, sobre el dron a controlar. Esta SBC sería la encargada de ejecutar el sistema ROS, y así evitar la necesidad de usar un ordenador portátil para este fin.

Si a esta idea se le añade la de incorporar un servicio de Cloud Computing, como el que ofrece Google o Amazon, sería posible llevar a cabo el control de cualquier UAV sin la necesidad de estar en la misma red WiFi, así como guardar en una base de datos toda la información proporcionada tanto por el dron, como por el usuario para su posterior visualización y estudio.

BIBLIOGRAFÍA

[1] "ROS" http://www.taringa.net/posts/ciencia-educacion/11780362/ROS---Robotic-Operative-System.html

[2] "Historia de los drones" http://munDrone.blogspot.com.es/p/historia-de-los-Drones.html

[3] "Misil crucero" http://es.gizmodo.com/el-misil-de-crucero-mas-rapido-vuela-a-mach-3-y-maniobr-1610490358

[4] "Vehículos aéreos no tripulados: definición, clasificación, aplicaciones…" https://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado

[5] "Movimientos Yaw, pich y roll" http://planning.cs.uiuc.edu/node102.html

[6] "Movimientos de los drones" http://cuadricopteros.org/cuadricopteros/

[7] "Especificaciones Ar-Drone 2.0" http://www.tecnofanatico.com/nuevo-ar-Drone-2-0-de-parrot-especificaciones-detalladas-video/

[8] "Especificaciones DJI-Phantom 2" http://www.cnet.com/es/analisis/dji-phantom-2-vision-plus/

[9] "Especificaciones Scout-x4" http://Droneshop.mx/producto/scout-x4/

[10] "Aplicación QGroundControl" http://qgroundcontrol.org/

[11] "Sistemas operativos para Smartphones" http://myr-smartphone-info2.blogspot.com.es/2013/12/sistemas-operativos-y-software-en-los_11.html