UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicaciones sobre ARDrone

Proyecto Final de Carrera

Ingeniería Técnica de Informática de Gestión

Autor: Carla Martínez Poveda
Director: Juan Carlos Ruiz García
Curso 2012 - 2013

El presente Proyecto de Final de Carrera se realizó durante el curso 2012 - 2013 bajo la titulación de Ingeniería Técnica de Informática de Gestión, presentándose el año siguiente durante el curso 2013 - 2014.

Quiero agradecer a mi director de proyecto, a mi novio Albert y a todas esas personas que estuvieron ahí para darme la "chispa" necesaria, su ayuda y dedicación a la hora de realizar mi último paso de la carrera.

Índice general

Capítulo 1

Motivación y objetivos

Hoy en día con el avance de las nuevas tecnologías, los drones se han convertido en una herramienta indispensable para ciertos propósitos en el campo de la informática. Ya sea para inspeccionar un terreno peligroso o para facilitar la tarea de observación de áreas remotas, su uso es ampliamente utilizado en diferentes sectores y mejorado a partir de nuevos modelos.

Figura 1.1: OQ-2 radioplane, drone utilizado en la Segunda Guerra Mundial

Tal y como se figura en [1], los drones surgieron por primera vez en la primera Guerra Mundial, pero no fue hasta la segunda cuando se emplearon como método de entrenamiento de los cañones antiaéreos. Sin embargo, el auge de los llamados VANT (Vehículos Aéreos no Tripulados) llegó bastante más tarde, a finales del siglo XX. En la guerra de Vietnam se utilizaron drones de reconocimiento y se sabe que anteriormente, durante la guerra del Golfo y la guerra de Bosnia, también se pusieron en marcha ciertos disposi-

tivos con fines bélicos.[2]

Figura 1.2: Draganflyer X6. Fotografía de Joe McNally.

Si tuviésemos que clasificar a los drones en tres tipos esenciales, éstos serían los siguientes:

- Drones de reconocimiento
- Drones de combate
- Drones de investigación y desarrollo

En la actualidad, los del segundo tipo tienen un reconocimiento lamentablemente popular. Se sabe que a lo largo de la historia, se han hecho servir en muchas de las guerras que han acontecido en el último siglo y en la actualidad permanecen a la orden del día en los medios de comunicación. Sin embargo, nuestro proyecto se cataloga dentro del tercer tipo, ya que está destinado a unos objetivos orientados al desarrollo de aplicaciones, los cuales concretaremos más adelante.

Además, entre los drones de tercer tipo hay una línea muy delgada en lo referente a desarrollo y el ocio, ya que algunos de éstos modelos destinados a tal efecto suelen ser ”modificables” hasta cierto punto, ya sea a nivel Hardware (modding, tunning, etc...) como a nivel de desarrollo Software. En el mercado se encuentran muchos de estos modelos en páginas de radiocontrol o en las mismas webs de las compañías que los fabrican. Por nombrar algunos modelos comerciales:

- Parrot AR.Drone
- Walkera QR
- RC Logger RC Eye
- VFV Copter
- X230 Mini Quadcopter

No es casualidad que en primer lugar se encuentre el AR.Drone, ya que es uno de los más populares y el que, además, se utilizará en este proyecto. Su descripción completa y características principales serán detalladas más adelante, pero podemos afirmar que es un cuadricóptero de uso recreativo civil que se controla mediante un dispositivo móvil o a través de un ordenador con un joystick [3]. Actualmente se encuentra en su versión 2.0 (desde julio de 2012) y pueden desarrollarse aplicaciones con un SDK que puede ser descargado libremente en una plataforma destinada a tal efecto [4].

Figura 1.3: Parrot AR.Drone

Las razones por las que nos hemos decantado por ese modelo de drone y no por otro son varias.

- En primer lugar, su creciente fama ha hecho que mucha gente se interese por él, por lo tanto se puede afirmar que hay multitud de información y documentación en la red.
- En segundo lugar, el hecho de que esté soportado por dispositivos móviles y por algunos sistemas operativos, hace el que el SDK sea muy funcional y adaptable.

- De la misma forma, el hecho de que exista una plataforma orientada al desarrollo de aplicaciones para el AR.Drone es un punto a su favor.
- También podemos decir que, entre las características del AR.Drone, caben destacar un par de cámaras a través de las cuales se puede ver el vuelo. Ésto hace que sea ideal para ciertas aplicaciones de Realidad Aumentada.
- En la plataforma del SDK existe mucha interactividad entre los desarrolladores, ya que dispone de un foro para resolver dudas y un API más o menos completo.

En definitiva, el SDK es un punto fuerte en el cual basarnos como argumento de nuestra elección. No existen demasiados drones comerciales que dispongan de un método de desarrollo de aplicaciones, ya que para algo más sofisticado tenemos que migrar a plataformas microcontrolador tales como Arduino o a proyectos concretos de ciertas universidades [5].

Además, anteriormente hemos comentado que el AR.Drone se encuentra en su versión 2.0. De la primera versión a la segunda existen pocas diferencias, pero algunas son relevantes para poder llevar a cabo ciertas tareas en lo que a la programación de aplicaciones se refiere. Hay programas que funcionan en la versión 2.0 y no lo hacen para la 1.0. Es por eso que se considera importante la posibilidad de adaptar aplicaciones entre una versión a otra.

Concluyendo, el objetivo de nuestro proyecto será poder ofrecer aportes de lo que funciona en el AR.Drone 1.0 y en el AR.Drone 2.0. De la misma forma, analizar la estructura de sus aplicaciones y portar una aplicación entre una y otra versión para poder hacernos con el control del dispositivo.

Capítulo 2

Parrot AR.Drone

2.1. Descripción

En el presente capítulo describiremos con más o menos detalle los elementos que contiene el AR.Drone y sus principales diferencias entre una versión y otra. Cabe remarcar que nos centraremos solo en las partes importantes que necesitaremos para poder realizar nuestro proyecto, por lo que es probable que la información aquí resulte incompleta para poder comprender la totalidad del AR.Drone. Hay que tener en cuenta que durante la realización de éste proyecto, las fuentes consultadas y sus referencias durante su redacción son abundantes, por lo cual no las incluiremos absolutamente todas, solo las importantes.

Desde su salida en 2010, el AR.Drone ha dado lugar a muchos proyectos bajo diferentes ramas de desarrollo. Si en cualquier buscador se escribe "Drone", su nombre aparece entre los primeros resultados. Es por eso que su creciente popularidad no es baladí, y eso ha sido determinante para que en multitud de comunidades de desarrollo, en universidades y en portales de internet se le tenga como sujeto de sus experimentos. Apuntábamos anteriormente a que esto era un punto a favor: es un dispositivo que continuamente se está moviendo, hay gente desarrollando para él. Debido a ésto, permanece siempre activo.

2.2. AR.Drone 1.0 vs AR.Drone 2.0

2.2.1. Vista general

Características generales

El AR.Drone es un cuadricóptero controlado mediante radiocontrol de uso civil y recreativo. La estructura mecánica consiste en cuatro hélices movidas por un motor cada una y que se cruzan en el centro del dispositivo. En dicho centro se encuentra la estructura central. Además, cada AR.Drone lleva dos carcasas para los diferentes tipos de vuelo: interior o exterior.

(a) AR.Drone en interior (b) AR.Drone en exterior

Figura 2.1: Tipos de vuelo del AR.Drone

Esqueleto y especificaciones técnicas

Su estructura central está formada por el cuerpo que contiene los elementos más significativos del dispositivo. Tiene una batería de litio recargable que permite una autonomía de vuelo de aproximadamente 15 minutos (según condiciones).

El sistema informático integrado consiste en un microprocesador ARM9 RISC de 32 bits @ 468 MHz, una memoria DDR SDRAM de 128 MB, un sistema operativo con núcleo Linux, un modem Wi-Fi b/g y además dispone de un conector USB de alta velocidad para poder almacenar los vídeos y las fotos que se realicen.

Podemos decir que tiene una estructura ligera, ya que su peso máximo es de 420 gramos con la carcasa incluida. Eso y el alcance de la conexión Wi-Fi le permiten llegar a altitudes de hasta 50 - 120 metros (dependiendo de las condiciones climáticas) y alcanzar velocidades de hasta 18 Km/h [3].

Sensores

Con respecto a los sensores, el AR.Drone dispone de:

- Sensor de ultrasonidos para medir los cambios de altitud.
- Sensor de altitud para correcciones.
- Acelerómetro digital de tres ejes para monitorizar los movimientos de posición.
- Giróscopo de dos ejes y giróscopo preciso piezoeléctrico para medir el giro y en cabeceo.
- Magnetómetro de tres ejes para la orientación.

Todos estos sensores están impresos en un circuito montado sobre un esqueleto de fibra de carbono. En su totalidad resultan cruciales para determinar la posición y orientación del drone en el vuelo.

Volar con el AR.Drone

Con respecto al sistema de vuelo, el procedimiento es bien sencillo. Supongamos que queremos hacer volar el AR.Drone de la forma tradicional: mediante el dispositivo móvil.

Lo primero que hay que hacer es descargar la aplicación oficial de Parrot y actualizarla con el último firmware mediante una opción que encontraremos en el programa. Posteriormente nos conectaremos a la wifi del AR.Drone y solo entonces podremos iniciar el vuelo con las opciones del menú. Cuando el AR.Drone despega, sube a una altura de aproximadamente un metro. A partir de ahí los movimientos que puede realizar son: adelante (cabeceo), atrás, izquierda, derecha, arriba, abajo, rodar hacia la derecha y hacia la izquierda. Además, la App oficial permite modificar parámetros como la altura, la velocidad, el ángulo de cabeceo, etc.

Mediante la forma no tradicional (*Joystick*), el sistema cambia bastante, pero no por ello resulta difícil. Los detalles de vuelo con *Joystick* los explicaremos más adelante.

Formas de comunicarse con el AR.Drone

Con respecto a la comunicación del AR.Drone con los diferentes dispositivos, la transmisión se realiza cuando éstos se conectan a la wifi del drone. Según los tipos de datos transmitidos, la emisión se realiza mediante un vía u otra.

Existen cuatro vías principales:

- Para controlar y configurar el AR.Drone se utilizan los **AT Commands**, que se transfieren por el puerto UDP 5556 con una frecuencia de 30 veces por segundo.
- La información sobre el estado del AR.Drone está contenido en los **Navdata**, bajo puerto UDP 5554 y en dos frecuencias: 15 veces por segundo en modo *Demo* y 200 veces por segundo en modo *Debug*. Éste último se utiliza en la creación de juegos de Realidad Aumentada.
- El **Video Stream** se mueve bajo TCP en el puerto 5555. Las imágenes que recibe el dispositivo pueden decodificarse usando el Códec incluído en el SDK.
- Los **Datos críticos** van al puerto de Control TCP 5559.

A la hora de comprender nuestra aplicación, estos datos resultarán relevantes. Asimismo, en el glosario describimos con detalle algunos términos cuya comprensión resulta de gran utilidad.

2.2.2. Arquitectura

En la figura 2.2 se puede apreciar que el AR.Drone dispone de una arquitectura multicapa de cuatro niveles: nivel de aplicación, nivel de hilos, host Sw y host Hw. En el AR.Drone 1.0 se podía acceder a un nivel que el AR.Drone 2.0 no puede actualmente, ya que ésto es debido a que cada vez se realiza un sistema más cerrado. Actualmente solo los niveles de aplicación y el nivel de hilos (de forma parcial) pueden ser modificados en mayor o menos medida.

Hay que destacar que cualquier proyecto personal que pueda realizarse con el SDK requiere de la gestión de hilos para poder lanzar cualquier aplicación. Dicha gestión está gestionada por la capa de la gestión de niveles y se comunica con el control de motor de AR.Drone, las librerias necesarias para poder

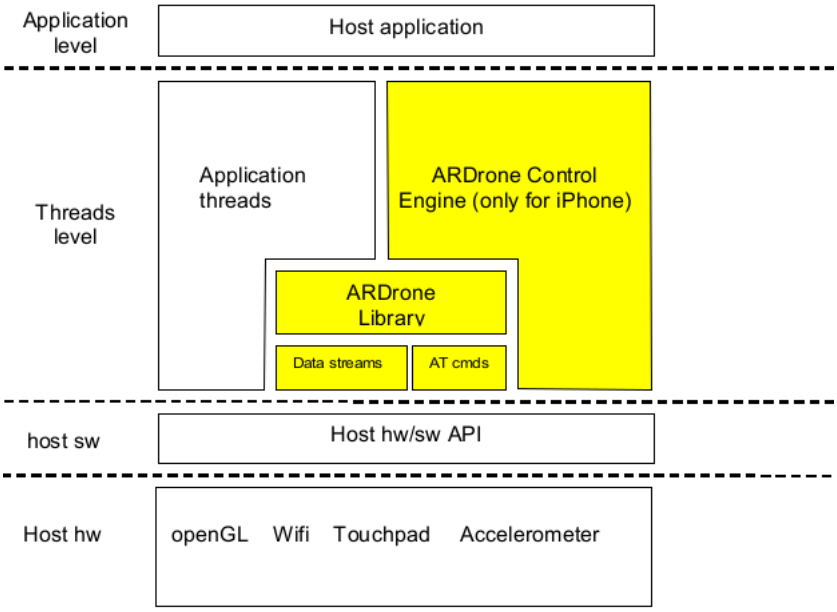

Figura 2.2: Arquitectura AR.Drone

compilar el proyecto y a su vez comunica con la capa superior del nivel de aplicación.

2.2.3. SDK

Descripción

El Software Development Kit del AR.Drone permite a los desarrolladores realizar aplicaciones y proporciona los APIs necesarios para compilar los programas. Según la versión, tiene soporte para crear aplicaciones en iOS, Linux, Windows y Android.

Es gratuito y se puede obtener en la página destinada a proyectos del AR.Drone [4], de la misma forma que el software Open Source se puede encontrar en la página de Parrot [11]. Actualmente en el momento de la redacción de éste documento, el SDK se encuentra en la versión 2.0.1.

Estructura principal

El SDK comprende las siguientes carpetas (centrándonos sólo en las importantes):

- **ARDroneLib:** Contiene el API para configurar el AR.Drone.

- **Control Engine:** Contiene ficheros específicos para iOS.
- **Examples:** Contiene ejemplos compilados realizados con ARDroneLib.

La parte más importante es la carpeta ARDroneLib, que está estructurada de la siguiente forma:

- **ARDroneLib.xcodeproj:** Proyecto de xcode para poder desarrollar aplicaciones en el entorno de desarrollo integrado de Apple para iOS.
- **FFMPEG:** Librería de código abierto que permite grabar, convertir y procesar audio y vídeo. Multiplataforma. En este caso es la encargada de procesar el vídeo del ARDrone.
- **ITTIAM:** Librería preconstruida (*ARM v7 + Neon*) y altamente optimizada de decodificación de vídeo para aplicaciones iOS y Android.
- **Soft:** Contiene el código específico del drone.
- **VLIB:** Librería de procesamiento de vídeo del AR.Drone 1.0. Contiene las funciones para recibir y decodificar el *video stream*.
- **VP_SDK:** Librerías de propósito general.

Las carpetas importantes en éste caso son *Soft* y *VP_SDK*, ya que la primera es el esqueleto necesario para poder desarrollar cualquier aplicación para el AR.Drone y la segunda contiene librerías útiles para poder gestionar ciertos aspectos de la programación de la aplicación.

Profundizando un poco más en *Soft/Lib/ardrone_tool* (figura 2.3), nos encontramos con la parte central de la estructura del AR.Drone. Cada una de las carpetas engloban una parte de la gestión del drone y a la hora de crear una nueva aplicación la estructura será semejante para nuestro programa.

Los ficheros y carpetas principales son:

- **ardrone_tool.c:** Inicializa comunicaciones con el drone. Incluye el *main()* principal.
- **UI:** Contiene el código para manejar el drone a través de un Gamepad.
- **AT:** Contiene funciones para controlar el drone a través de los AT Commands.

- **Navdata:** Recibe y decodifica los Navdata, los cuales informan del estado del drone.

- **Control:** Configura y gestiona el drone.

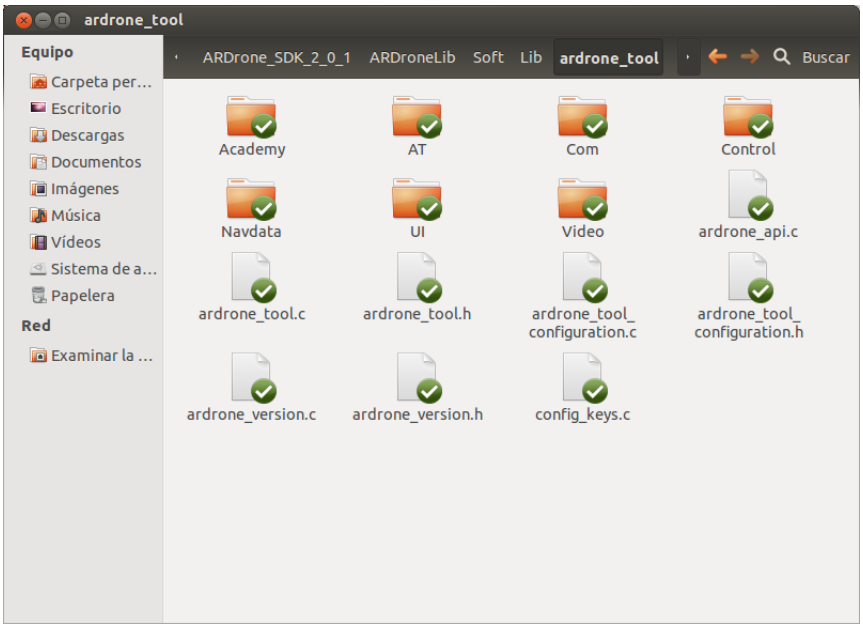

Figura 2.3: Carpeta *ardrone_tool*

Ciclo de vida y programa principal

La figura 2.3 muestra el ciclo de vida del SDK del AR.Drone. Los primeros trés métodos pueden encontrarse en el fichero *ardrone_tool.c* y es el punto de partida de cualquier aplicación para el AR.Drone. El *main()* llama al método *ardrone_tool_setup_com()* que a su vez llama a *ardrone_tool_init()*. Finalmente se entra en el bucle principal del que solo se sale cuando la aplicación finaliza, haciendo uso de la función *ardrone_tool_shutdown*.

A la hora de explicar nuestra aplicación, entraremos con más detalle en cada uno de los métodos importantes del ciclo de vida de una "*Custom application*" estándar.

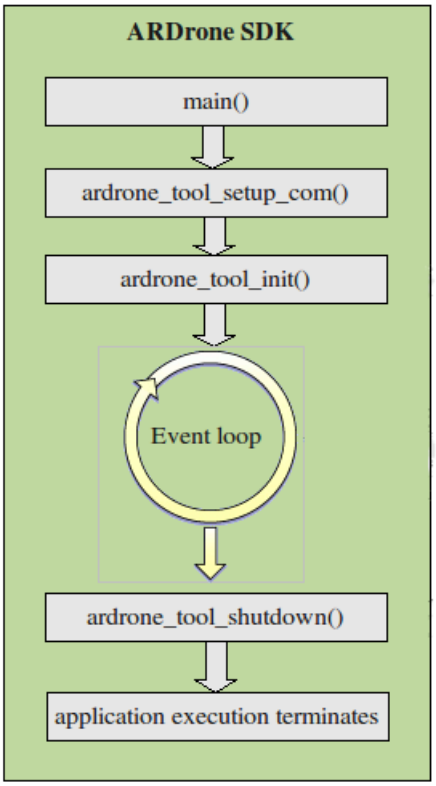

Figura 2.4: AR.Drone bucle principal

Compilación de ejemplos

En el SDK existe la carpeta *Ejemplos*, la cual nos permite ver programas ya creados con el *ardrone_tool*. Se ofrece la posibilidad de que el usuario los compile y los pruebe bajo las plataformas Windows, Linux, iOS y Android. En nuestro caso, explicaremos los pasos para compilar los ejemplos bajo Linux y con el SDK 2.0.1:

1. Hay una serie de librerías que hay que instalar para poder compilar los programas. Éstas son:
   - *daemontools*
   - *libxml2-dev*
   - *libudev-dev*
   - *libsdl-dev*
   - *libgtk2.0-dev*

- *libiw-dev*

2. Descargamos el SDK y lo descomprimimos.
3. Nos conectamos a la red WiFi del AR.Drone, que suele tener el siguiente nombre (en nuestro caso): *ardrone2_wifi*
4. En la carpeta *ARDrone_SDK/ARDroneLib/Soft/Build* hay que modificar del documento *custom.makefile* la siguiente línea:
   USE_LINUX=yes
5. Hacer **make** en esa carpeta.
6. En *ARDrone_SDK/Examples/Linux* hacer **make**.
7. Ahora, según el ejemplo que queramos compilar primero, iremos a cualquiera de las dos carpetas *Navigation* o *sdk_demo* y haremos **make** dentro de *Build*. La primera contiene un programa de gestión de vuelo y la segunda solo indica parámetros haciendo uso de los *Navdata*.
8. Una vez compilado, nos movemos hasta la carpeta *Linux/Build/Release* ejecutamos la aplicación deseada (*ardrone_navigation* o *linux_sdk_demo*).

Para compilar los ejemplos, es conveniente disponer de un *Joystick*, ya que no es posible realizar eventos mediante el teclado. Más tarde enumeraremos los modelos que hemos usado para nuestro proyecto.

2.2.4. Mejoras y carencias

Como paso lógico de una versión a otra, existen ciertas diferencias a tener en cuenta entre el AR.Drone 1.0 y el 2.0.

- El AR.Drone 1.0 tiene menos sensores que el AR.Drone 2.0. En la segunda versión se añadieron el mangetómetro y el sensor de presión.
- Los "Tablero de navegación" no es el mismo en las dos versiones.
- Las cámaras: el AR.Drone 2.0 tiene una cámara frontal HD (720px - 30fps) mientras que el 1.0 usa QVGA (320*240). De la misma forma, la cámara ventral de la versión 2.0 es de mayor calidad.
- El AR.Drone 2.0 tiene un puerto USB master, con un conector estándar USB-A.

- El AR.Drone 1.0 tiene más AT Commands que el AR.Drone 2.0 (AT*LED y AT*ANIM).
- En el SDK de la versión 2.0 existe la carpeta *Google API*, la cual no se encuentra en en la versión 1.0.
- En lo referente a la detección de patrones para RA, hay ciertas variables que se crearon nuevas para la versión 2.0 (*orientation_angle[i], rotation[i], translation[i], camera_source[i]*).
- El alcance de la WiFi del AR.Drone 2.0 es mayor que la del 1.0.
- En el AR.Drone 1.0, actualizar la aplicación oficial implica migrar a la versión 2.0.
- En general, crear una aplicación con el SDK cambia ligeramente entre una versión y otra, ya que hay ficheros, clases y métodos que se ven modificados para optimizarlos.
- El AR.Drone 2.0 tiene, además del modo de control relativo, el modo de control absoluto (usando el magnetómetro).

2.2.5. Mercado

En el mercado existen numerosas aplicaciones de móvil para el AR.Drone, desde la app oficial que permite controlar el vuelo del dispositivo, hasta aplicaciones para realizar luchas entre drones o de Realidad Aumentada. Por nombrar algunas:

- **AR.FreeFlight:** App oficial de Parrot, disponible para dispositivos Android y iOS [6].
- **Flight Record:** Permite grabar el vuelo del drone y tomar fotografías. Existe otra aplicación parecida llamada Drone Ace.
- **AR.PowerFlight:** Aplicación de control de vuelo. Solo para dispositivos iOS [7].
- **Drone Master:** Permite volar el drone mediante magnetómetro. Solo para dispositivos iOS [8].
- **Flying Ace:** Aplicación de Realidad Aumentada que permite una lucha entre drones. Para dispositivos iOS [9].
- **Drone Escape:** Similar al Flying Ace, pero hay que combatir contra aliens. Para dispositivos iOS [10].

Capítulo 3

Nuestra aplicación

3.0.6. Primera aproximación y características

El objetivo de nuestra aplicación es poder hacer que el drone realice una serie de movimientos de vuelo de forma completamente controlada mediante el *Joystick*. En otras palabras, vamos a hacernos con el control del dispositivo, pero con la diferencia de que adaptaremos una aplicación de la versión 1.0 a la 2.0. De esta forma, el *"port"* nos dará luz a las posibles diferencias que hayan a nivel de código entre una versión y otra.

Ésto a primera vista parece no aportar ninguna novedad, de no ser porque gracias a la adaptación de un programa para la versión 1.0 del drone, veremos hasta qué punto cambia el código, los métodos y funciones que encontraremos en ambas versiones.

De la misma forma, hay varias formas de realizar una aplicación para el AR.Drone y las dos tienen algo en común: la asociación de hilos (*Threads*) y funciones al método principal.

Antes hablábamos del ciclo de vida de la aplicación del AR.Drone y por donde pasaban sus funciones. Ahora explicaremos lo mismo pero orientado a una aplicación customizada que utilice la librería principal proporcionada por el SDK, ésto es, *ARDroneLIB*.

En la figura 3.1 podemos ver el mismo ciclo de vida que mostrábamos en el apartado 2.2.3, pero esta vez lo vemos aplicado a una aplicación realizada por nosotros mismos usando las herramientas del SDK. En ella se hace referencia a nuevos métodos que explicaremos a continuación siguiendo los pasos tal y

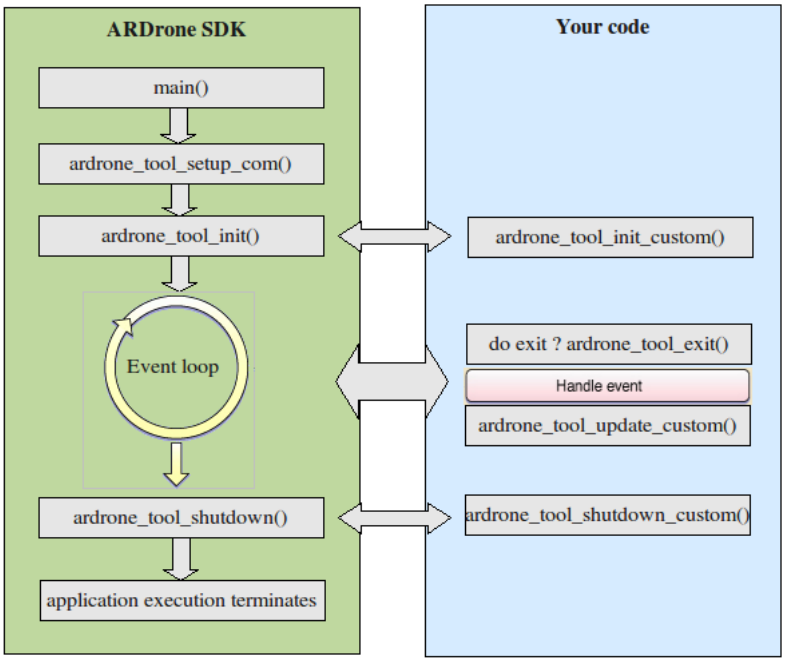

Figura 3.1: Ciclo de vida y *Custom application*

como se indican el manual [13]:

En primera instancia, para crear una aplicación propia para el AR.Drone 2.0, la forma más sencilla es **basarse en el ya existente *sdk_demo***, disponible en la carpeta *Examples*. Los pasos a seguir son los siguientes:

1. Crear el nuevo hilo para tu aplicación y añadirlo a la *THREAD_TABLE* (se encuentra al final del fichero *ardrone_testing_tool.c*) para poder enviar comandos independientes. (Figura 3.2)
2. Llamar a la función *ardrone_tool_main* desde tu aplicación
3. Crear cualquier controlador *Navdata* necesario y añadirlo a la tabla *ardrone_navdata_handler_table* (se encuentra en *ardrone_navdata_client.c*). (Figura 3.3)

Otra opción que figura el manual es **customizar el cliente inicial**, ya que éste contiene el código base para comenzar una aplicación propia. Una de las indicaciones que resume el procedimiento a seguir es la siguiente:

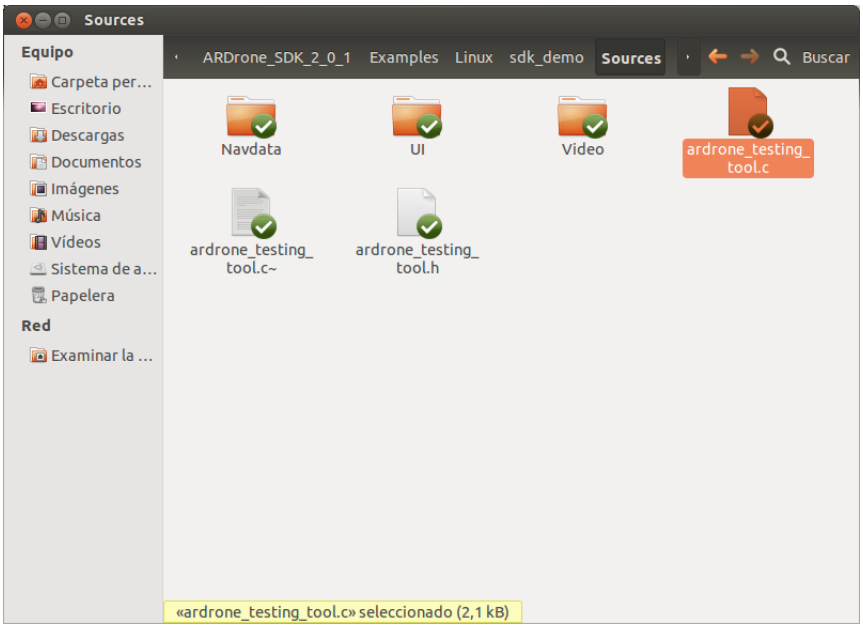

Figura 3.2: Ruta *ardrone_testing_tool.c*

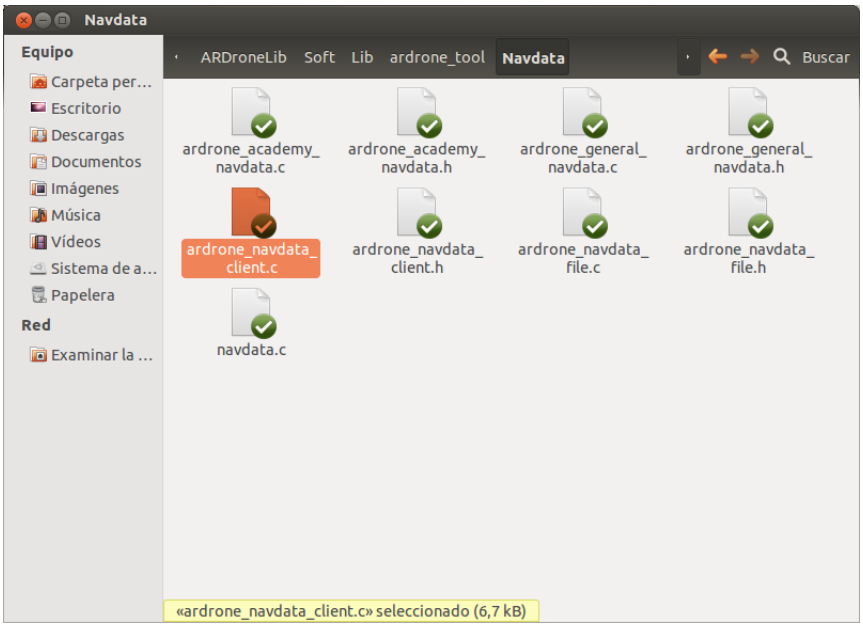

Figura 3.3: Ruta *ardrone_navdata_client.c*

"Al escribir tu aplicación, debes compilarla junto con el **ARDroneLIB** y

llamar a su función (*ardrone_tool_main*) para obtener una aplicación cliente completamente funcional."

Asímismo, el procedimiento del ciclo de vida adaptado a una aplicación custom es el siguiente:

1. **ardrone_tool_main()** se encuentra en *ardrone_tool.c* y no requiere modificación. Cada aplicación que se cree, tendrá una función **ardrone_tool_main()** que serán idénticas entre sí. Las funciones que realiza **ardrone_tool_main()** son:
   - Configura la red WiFi.
   - Inicializa los puertos de comunicación (*AT Commands, Navdata*, vídeo y control).
   - Llama a la función *ardrone_tool_init_custom*. Su prototipo está definido en *ardrone_tool.h* y debe ser definido y customizado por el desarrollador.
2. En *ardrone_tool.h* debemos tener:
   - La inicialización local de nuestra propia aplicación.
   - La inicialización de los dispositivos de entrada (*Input Devices*) con la función *ardrone_tool_input_init()*.
3. Se lanza el hilo **navdata_update** que está situado en el fichero *ardrone_navdata_client.c*. Para hacer correr apropiadamente esta rutina, el usuario debe declarar la tabla *ardrone_navdata_handler_table* en *ardrone_navdata_client.h*.
4. Se lanza el hilo **ardrone_control** que está situado en el fichero *ardrone_control.c*.
5. En este punto, reconoce el drone para indicar que está listo para recibir *Navdatas*.
6. Finalmente, se llama a la función **ardrone_tool_update()** en bucle, la cual se mantiene hasta que se sale de la aplicación. Ésta recupera la información del dispositivo para enviársela al AR.Drone. En este caso, el usuario debe declarar la función **ardrone_tool_update_custom()**, que será llamada por ésta función.

7. En el momento en el que la aplicación se detiene, a su vez lo hace el bucle. Se llaman a los métodos **ardrone_tool_exit()** y **ardrone_tool_shutdown()** para salir del bucle principal y liberar los recursos. La función que se encarga del *shutdown* también se encarga de llamar a su función custom: *ardrone_tool_shutdown_custom()*.

Éste sería el procedimiento a seguir para cualquier aplicación. En nuestro caso, haremos uso de la primera opción que nos sugiere la guía: modificar la aplicación ejemplo *sdk_demo*, de manera que la gestión de los hilos será mínima y eso nos permite tomar el ejemplo antiguo del *sdk_demo* del SDK 1.6 (solo es operativo en el AR.Drone 1.0) y modificarlo para adaptarlo al nuevo.

Especificación de requisitos

Para realizar nuestra aplicación, utilizaremos el sistema operativo Ubuntu en su versión 12.04, además de otras características que muestra las siguientes figuras (3.4 y 3.5).

Figura 3.4: Características del equipo

El *Joystick* que utilizaremos será **Logitech, modelo G-UF13A** (figura 3.6).

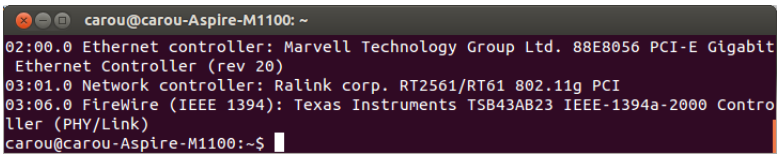

Figura 3.5: Tarjeta de red

Figura 3.6: Logitech Gamepad

3.0.7. Refinamiento

Estructura

A la hora de realizar nuestra aplicación, los primeros pasos a seguir fueron los siguientes:

1. Descargar y descomprimir el SDK 2.0.1.

2. Hacer lo mismo con el SDK 1.6.

3. Compilar los ejemplos del SDK 2.0.1 según los pasos descritos en el apartado 2.2.3.

Es conveniente recordar que nuestra aplicación será una adaptación de cambio de versión, es decir, a partir de una aplicación existente del AR.Drone 1.0 realizaremos un "*port*" para hacer que funcione en el AR.Drone 2.0. La aplicación existente que usaremos será el ejemplo ***sdk_demo*** del SDK 1.6.

Nuestro proyecto de AR.Drone para el SDK 2.0.1 tendrá la siguiente estructura: dentro de la carpeta ejemplo (*sdk_demo*)(figura 3.7) estarán las carpetas ***Build*** (figura 3.8) y ***Sources*** (figura 3.9). La primera contendrá el *Makefile* para compilar la aplicación y en la segunda estarán los diferentes módulos de nuestra aplicación.

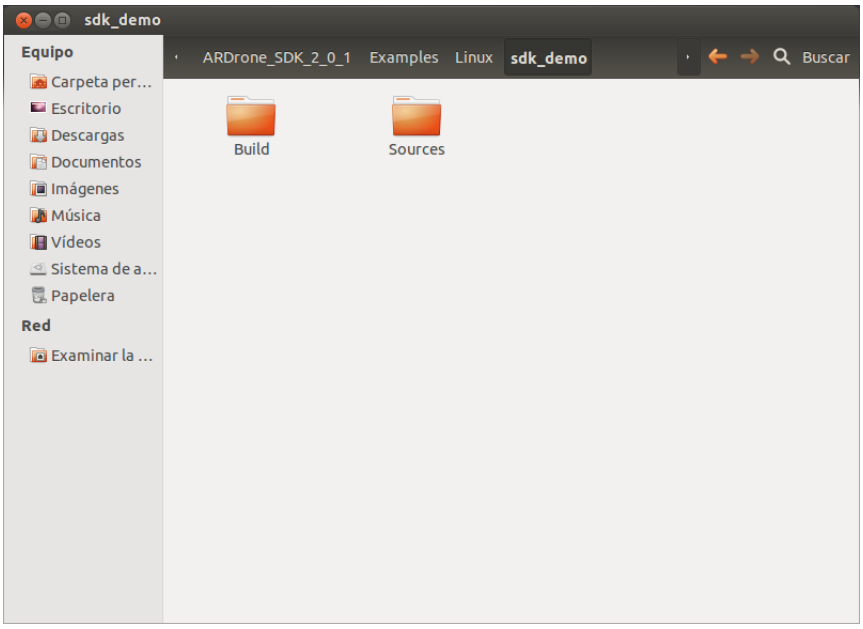

Figura 3.7: Carpeta *sdk_demo*

La función de cada una de las carpetas es la siguiente:

- ***Navdata*** se encarga de gestionar todo lo relacionado con los *Navdatas*, que aportarán información sobre los diferentes estados del AR.Drone.
- ***UI*** contiene toda la parte que tiene que ver con el vuelo del AR.Drone mediante *Joystick*.
- ***Video*** permite gestionar todo lo relacionado con el vídeo. En principio nosotros no utilizaremos esa carpeta, ya que pertece a otro ejemplo a compilar en el mismo SDK llamado *sdk_video_demo*. Sin embargo, vamos a conservarla en su ubicación actual sin tocarla.

Con respecto al fichero ***ardrone_testing_tool***, contiene el programa principal customizado del que hablábamos cuando explicábamos el ciclo de vida de una aplicación de AR.Drone y básicamente establece las llamadas necesarias para ejecutar la aplicación.

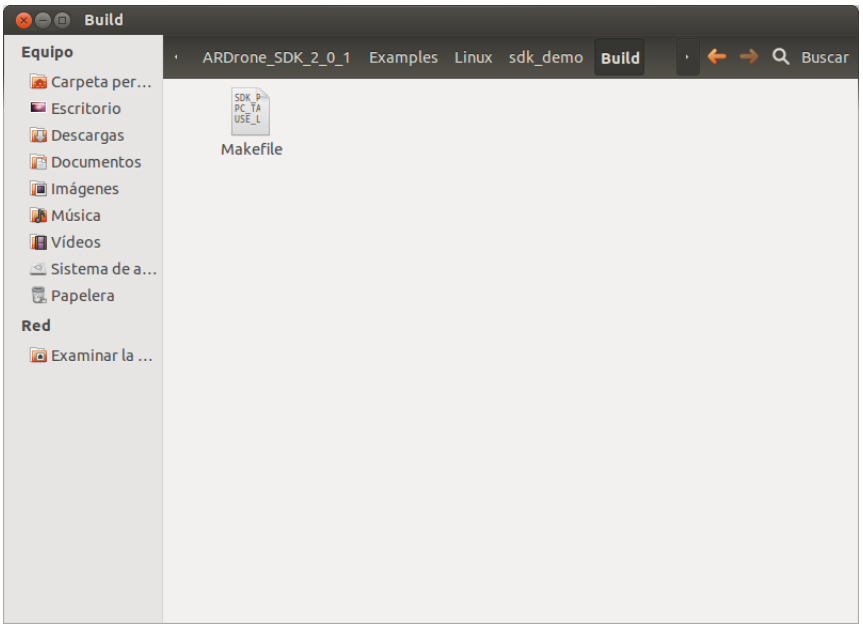

Figura 3.8: Carpeta **Build**

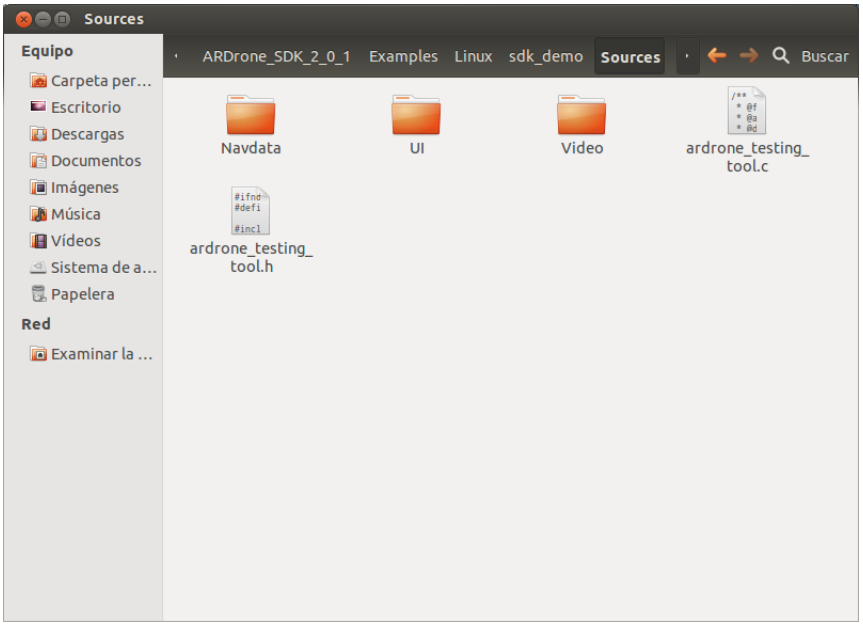

Figura 3.9: Carpeta **Sources**

En el SDK 1.6 la estructura es igual, pero el objetivo de la aplicación es diferente: aquí proporciona la posibilidad de controlar el AR.Drone mediante un *Joystick* o *Gamepad*. En el SDK 2.0.1 éste programa fue modificado para que solo mostrara datos sobre parámetros del drone que se consultaban directamente de los *Navdatas*. El sistema de vuelo por *Joystick* es lo que intentaremos recuperar para esta nueva versión del SDK.

Para añadir al nuevo programa el control de vuelo, optamos por copiar el directorio *UI* del SDK 1.6 a nuestro directorio *Sources* (se puede ver como ya sale la figura 3.2). Posteriormente habría que modificar el fichero ***Makefile***, que es el que se encargará de compilar nuestra aplicación.

Dentro de la nueva carpeta *UI* tenemos los siguientes programas (figura 3.10):

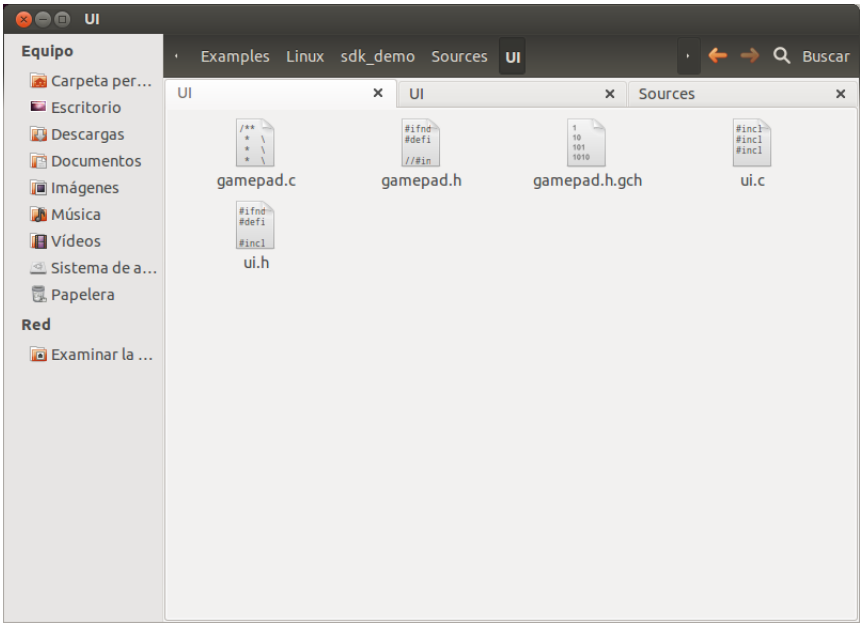

Figura 3.10: Carpeta **UI**

- ***gamepad*** será el programa encargado totalmente del control de vuelo del drone. En él modificaremos ciertos parámetros para que se ajusten al nuevo código del SDK 2.0. El fichero con extensión gch es una cabecera precompilada que se genera de forma automática.
- ***ui*** es un programa que heredamos por el paso de la carpeta principal *UI*, pero que no modificaremos para nada.

Cambios en el *Makefile*

- Definimos nuestra nueva carpeta *UI* de la siguiente forma

```
UI_SRC_PATH = UI
```

  Este cambio es para que el compilador reconozca la carpeta y la incluya en el proceso de compilación. Posteriormente lo asociamos a los ficheros *gamepad* y *ui*, de tal forma que sus ficheros fuente genérico binario queda así:

```
$(UI_SRC_PATH)/gamepad.c \
$(UI_SRC_PATH)/ui.c \
```

- Para hacer uso de una función que definimos en el fichero *gamepad.c*, necesitamos definir en *GENERIC_INCLUDES* la siguiente librería:

```
/usr/include/libxml2
```

- En *GENERIC_LIBS* finalmente se hacen uso de las siguientes librerías:

```
GENERIC_LIBS=-liw -lpc_ardrone -lgthread-2.0
-lgtk-x11-2.0 -lrt -lxml2 -ludev -lswscale
-lSDL
```

El resto del fichero *Makefile* se conserva sin cambios.

Cambios en *Gamepad*

- Hemos definido los siguientes *include*:

```
#include <ardrone_tool/UI/ardrone_input.h>
#include <ardrone_tool/ardrone_tool_
configuration.h>
#include <ardrone_tool/Navdata/ardrone_
academy_navdata.h>
```

- Todo lo relacionado con **PS3Gamepad** lo eliminamos. Nos quedamos con **Gamepad**.

- En el método ***update_gamepad***:
  - Sustituimos la siguiente línea:

```
static int32_t x = 0, y = 0;
```

    Por la siguiente:

```
static int32_t stick1LR = 0,
stick1UD = 0 , stick2LR = 0 ,
stick2UD = 0;
```

    Con ésto nos definimos nuevos ejes para manejar el drone desde dos puntos más.
  - Comentamos la siguiente línea, debido a que ésta clase ya no se utiliza en la versión 2.0. :

```
input_state_t* input_state;
```

  - En vez de usar las variables *theta_trim*, *phi_trim* y *yaw_trim*, nos definimos las cuatro siguientes:

```
static int center_x1=0;
static int center_y1=0;
static int center_x2=0;
static int center_y2=0;
```

  - Comentamos las siguientes líneas:

```
refresh_values = FALSE;
input_state = ardrone_tool_
get_input_state();
```

    El primero es porque no nos interesa que refresque valores, y el segundo porque ya no existe ese método en la versión 2.0.
  - Dentro del *switch* hemos renombrado los *cases*. Se pueden conservar como están, pero los hemos cambiado para hacerlos más intuitivos de ver a primera vista. No es un cambio especialmente relevante pero hace falta remarcarlo para evitar confusiones. En **gamepad.h** se encuentran:

```
typedef enum {
        ARRIBA = 3,
        ABAJO = 1,
        IZQUIERDA = 0,
        DERECHA = 2,
```

```
                SUBE = 5,
                BAJA = 4,
                RUEDA_IZQ = 6,
                RUEDA_DER = 7,

                PAD_SELECT = 8,
                PAD_START = 9,
                PAD_NUM_BUTTONS
        }PAD_BUTTONS;
```

Cada uno de los números define un evento del *Joystick* (más referencias en glosario).

* El contenido de los *cases* del *switch* cambia a los siguientes valores:

```
    switch( js_e_buffer[idx].number)
{
    case RUEDA_IZQ :
            stick2LR = -32767 * js_e_buffer
            [idx].value;
            break;
    case BAJA :
            stick2UD = 32767 * js_e_buffer
            [idx].value;
            break;
    case SUBE :
            stick2UD = -32767 * js_e_buffer
            [idx].value;
            break;
    case ARRIBA : //adelante
            stick1UD = -32767 * js_e_buffer
            [idx].value;
            break;
    case ABAJO : //atras
            stick1UD = 32767 * js_e_buffer
            [idx].value;
            break;
    case RUEDA_DER :
            stick2LR = 32767 * js_e_buffer
            [idx].value;
            break;
```

```
case IZQUIERDA :
        stick1LR = -32767 * js_e_buffer
        [idx].value;
        break;
case DERECHA :
        stick1LR = 32767 * js_e_buffer
        [idx].value;
        break;
case PAD_SELECT :
        ardrone_tool_set_ui_pad_select(js_e_buffer
        [idx].value);
        break;
case PAD_START :
        if( js_e_buffer[idx].value ){
                start ^= 1;
                ardrone_tool_set_ui_pad_
                start( start );
        }
        break;
default:
        break;
}
```

* A partir de la siguiente línea:

```
else if(js_e_buffer[idx].type & JS_EVENT_AXIS )
{
```

Comentamos o eliminamos el siguiente código:

```
refresh_values = TRUE;
(...)
case PAD_X:
        x = ( js_e_buffer[idx].value + 1 ) >> 15;
break;
case PAD_Y:
        y = ( js_e_buffer[idx].value + 1 ) >> 15;
        break;
default:
        break;
```

Ésto se debe a que, como ya no usamos los parámetros x e y porque pasamos a los cuatro ejes que soporta nuestro *Joystick*, tenemos

que reescribir el código para que reconoxca los valores del axis, que es la referencia que toma el programa para la orientación y posterior dirección del AR.Drone.

* El código que antes habíamos comentado, lo sustituimos por éste:

```
int axis_value = (js_e_buffer[idx].number
+ 1) * (js_e_buffer[idx].value > 0 ? 1 :
-1);

if ((axis_value == default_control
->commands[PITCH_FRONT].value) && (default_
control->commands[PITCH_FRONT].type ==
AXIS)) {
        stick1UD = ( js_e_buffer[idx].value );
} else if ((axis_value == default_control
->commands[PITCH_BACK].value) && (default_
control->commands[PITCH_BACK].type ==
AXIS)) {
        stick1UD = ( js_e_buffer[idx].value );
} else if ((axis_value == default_control
->commands[ROLL_LEFT].value) && (default_
control->commands[ROLL_LEFT].type ==
AXIS)) {
        stick1LR = ( js_e_buffer[idx].value );
} else if ((axis_value == default_control
->commands[ROLL_RIGHT].value) && (default_
control->commands[ROLL_RIGHT].type ==
AXIS)) {
        stick1LR = ( js_e_buffer[idx].value );
} else if ((axis_value == default_control
->commands[SPEED_UP].value) && (default_
control->commands[SPEED_UP].type ==
AXIS)) {
        stick2UD = ( js_e_buffer[idx].value );
} else if ((axis_value == default_control
->commands[SPEED_DOWN].value) && (default_
control->commands[SPEED_DOWN].type ==
AXIS)) {
        stick2UD = ( js_e_buffer[idx].value );
} else if ((axis_value == default_control
```

```
->commands[YAW_LEFT].value) && (default_
control->commands[YAW_LEFT].type ==
AXIS)) {
        stick2LR = ( js_e_buffer[idx].value );
} else if ((axis_value == default_control
->commands[YAW_RIGHT].value) && (default_
control->commands[YAW_RIGHT].type ==
AXIS)) {
        stick2LR = ( js_e_buffer[idx].value );
}
```

* Todo lo que contiene la siguiente línea de código:

```
if(refresh_values)// Axis values to refresh
{ (...) }
```

Es sustituido por lo siguiente:

```
static int minActive = 32768 * (JOYSTICK_DEAD_
ZONE_PERCENT) / 100;
static int negMinActive = -32768 * (JOYSTICK_DEAD_
ZONE_PERCENT) / 100;

#define SATURATE_JOYSTICK(STICK, POS, NEG)
        do
        {
                if (POS > STICK && NEG <
                STICK){
                        STICK = 0;
                }
        } while (0)

int enable = 1;

SATURATE_JOYSTICK (stick1LR, minActive,
negMinActive);
SATURATE_JOYSTICK (stick1UD, minActive,
negMinActive);
SATURATE_JOYSTICK (stick2LR, minActive,
negMinActive);
SATURATE_JOYSTICK (stick2UD, minActive,
negMinActive);
```

```
if (0 == stick1LR && 0 == stick1UD)
{
        enable = 0;
}

ardrone_tool_set_progressive_cmd( enable,
        /*roll*/(float)(stick1LR-center_x1)
        /32767.0f,
        /*pitch*/(float)(stick1UD-center_y1)
        /32767.0f,
        /*gaz*/-(float)(stick2UD-center_x2)
        /32767.0f,
        /*yaw*/(float)(stick2LR-center_y2)
        /32767.0f,
        /*psi*/0.0,
        /*psi_accuracy*/0.0);
```

Éste código es muy parecido al que se ha definido para **PS3Gamepad**, del cual hemos tomado referencia para algunas partes del código.
En él se establecen ciertos parámetros para el manejo *Joystick*.

Capítulo 4

Pruebas obtenidas

Como resultado final, nuestra aplicación originada de una versión anterior del AR.Drone 1.0 permite ser utilizada en el AR.Drone 2.0 como un sistema de vuelo controlado mediante *Joystick* por el usuario. No se requiere que éste último tenga demasiada práctica con el manejo del dispositivo, ya que conociendo los controles básicos el funcinamiento es bastante intuitivo y hasta un niño puede utilizarlo.

Considerando el mando utilizado en nuestro proyecto (figura 4.1), los eventos relacionados con los botones son los siguientes:

- Botón 10: Despegue/aterrizaje
- Botón 9: Emergencia. Detiene los motores.
- Botón 4: Adelante. Cabeceo.
- Botón 2: Atrás.
- Botón 3: Derecha.
- Botón 1: Izquierda.
- Botón 7: Rotar hacia la izquierda.
- Botón 8: Rotar hacia la derecha.

Asímismo, el sistema de vuelo una vez ejecutada la aplicación permanece estable y las acciones se realizan sin ningún tipo de retardo. Gran parte de la documentación que aportamos para respaldar el resultado final de nuestra aplicación se encuentran en el CD anexo al proyecto.

Figura 4.1: Números de los botones en Logitech

Capítulo 5

Conclusiones

5.0.8. Con respecto a nuestra aplicación y el SDK

En lo referente a las diferencias de código entre una versión y otra para **ARDroneLib**, enumeramos las características más significativas:

- En el SDK 2.0 hay unas librerías que soporte (*FFMPEG, ITTIAM*) que no se encuentran en el SDK 1.6. Ésto es debido a que el AR.Drone 2.0 cambia el sistema de visionado de cámara , recordemos que el 2.0 usaba HD y el 1.0 QVGA. Eso hace que el procesado de los diferentes tipos de imagen se realicen de forma diferente y con herramientas distintas.

  De ahí que la nueva versión del SDK añada éstas dos nuevas librerías: la primera se encarga de la grabación, conversión y procesamiento de audio y vídeo. La segunda es una librería preconstruida y optimizada de decodificación de vídeo para aplicaciones iOS y Android.

- Se optimizan algunas funciones relacionadas con las comunicaciones del AR.Drone 2.0, ya que en el código de *ardrone_tool.c* aparecen de forma dispersa nuevas líneas de código orientado a la gestión de la conexión WiFi y sus respectivos *Navdatas*. En las figuras 5.1 y 5.2 se puede ver subrayadas.

- El SDK 2.0 permite definir *customs profiles* que vienen dados por una definición por defecto de éstos y que pueden ser modificados posteriormente. En las figuras 5.3 y 5.4 se puede ver el nuevo código y la referencia en el manual.

- *Academy* es una nueva plataforma que permite poder subir vídeos y fotos realizadas con el AR.Drone 2.0 a una comunidad alojada en la

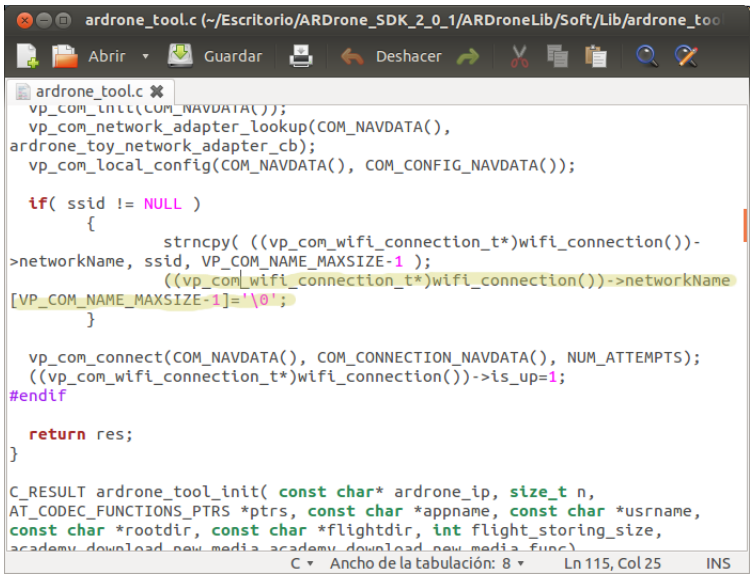

Figura 5.1

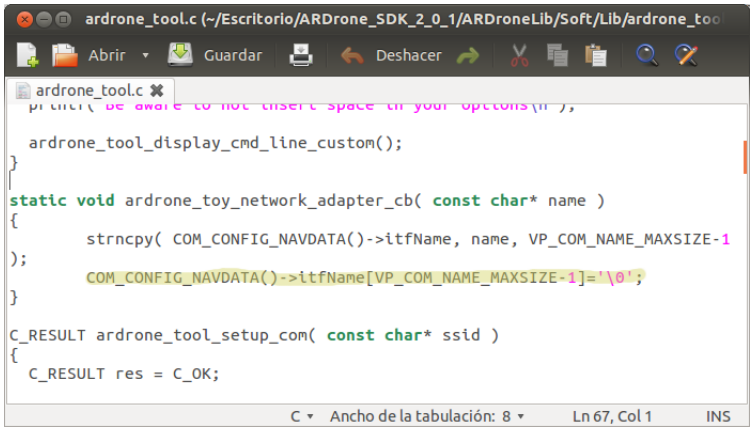

Figura 5.2

nube (figura 5.5). El SDK de esta nueva versión, concretamente el *ardrone_tool*, también contiene formas de configurar *Academy* para nuestras aplicaciones (figura 5.6).

- La estructura del código de uno de los métodos principales, *ardrone_tool_init()*, cambia completamente con respecto a su predecesora.

  En la versión 1.6, a la hora de llamar a esta función, habían dos modos: sin usar el bucle principal (con la notación *ifdef NO_ARDRONE_MAINLOOP*) o usando el bucle principal. Esto provocaba ceñirse a un modo en con-

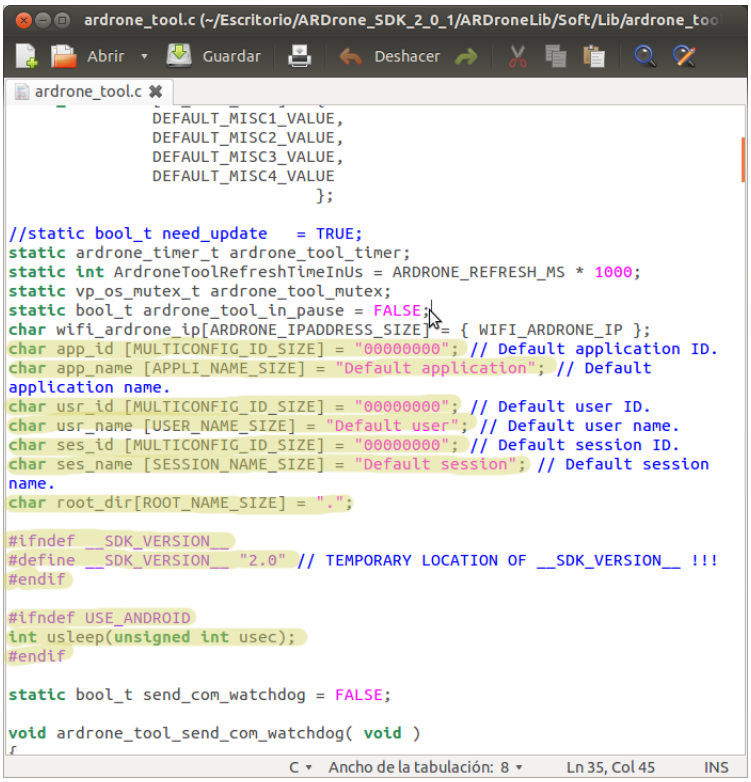

Figura 5.3

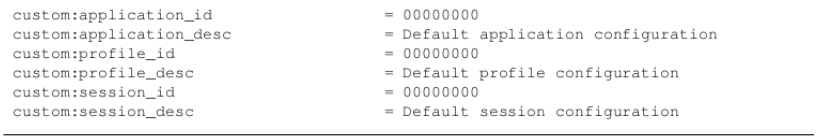

Figura 5.4

creto y no poder usar las opciones del otro modo de forma pública.

En esta nueva versión eso se elimina por completo, esos dos modos se unifican en un sistema ”híbrido”. Se optimizan ciertos aspectos del programa para obtener ventajas sobre éste cambio y otros que se producen para optimizar el código.

1. Se inicializan el mutex y la condición (esto no cambia).
2. Se inicializan las estructuras *ardrone_control_config* .
3. Inicialización de *ardrone_control_config_default*.
4. Inicialización de los valores por defecto definidos por la aplicación.

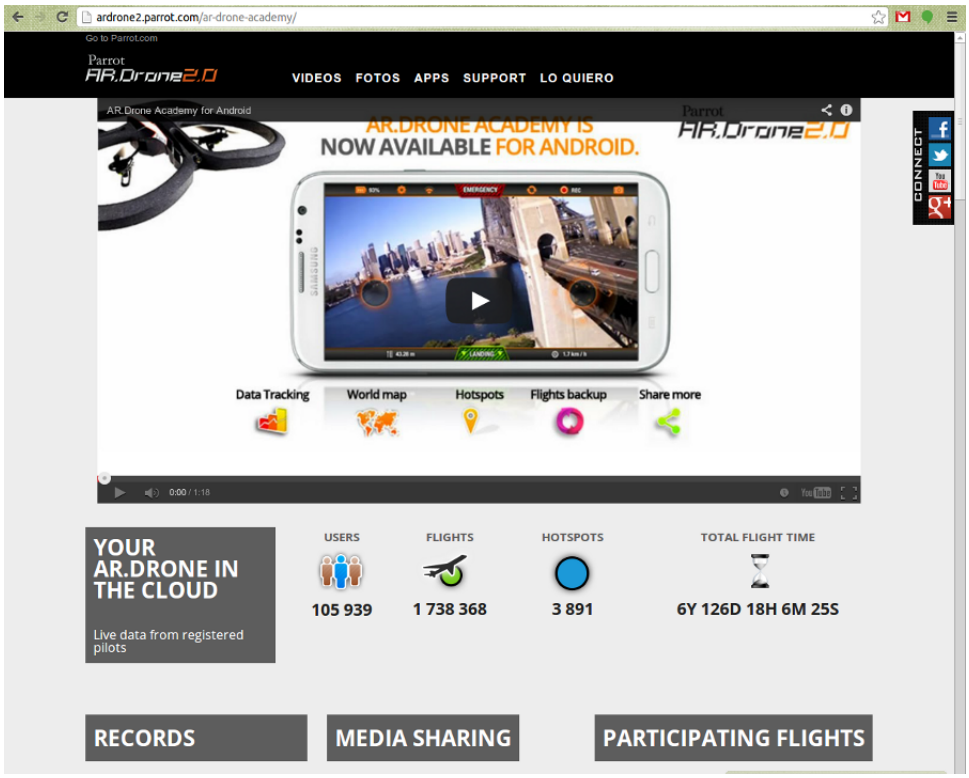

Figura 5.5: Web AR.Drone Academy

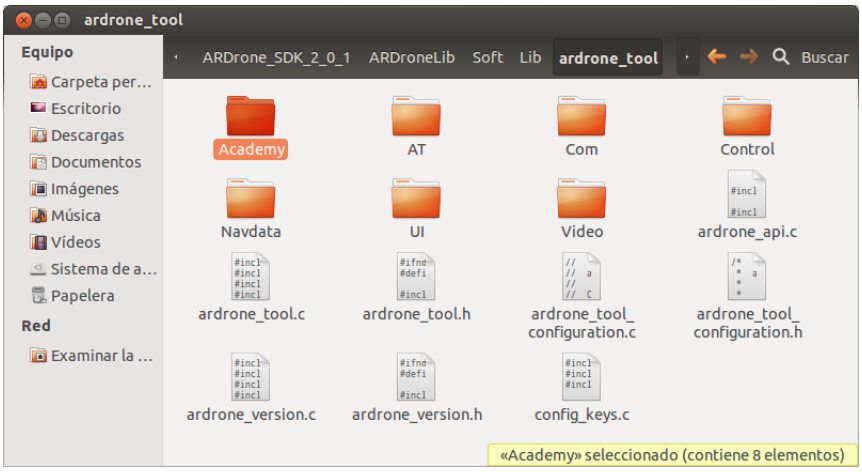

Figura 5.6: *Academy* en *ardrone_tool*

5. Guardar *appname/appid* para reconexiones.
6. Guardar *usrname/usrid* para reconexiones.
7. Crear una *session id* pseudoaleatoria.

8. Copiar el directorio *root*.
9. Rellenar la estructura *AT Codec* y construir la librería *AT Commands*.
10. Iniciar subsistemas.
11. Iniciar *ardrone_tool_init_custom()*.
12. Abrir conexiones al puerto AT.
13. Enviar una configuración de inicio.

- Aunque no es un gran cambio, algunos métodos son renombrados pero siguen desempeñando las mismas funciones. Un ejemplo entre varios es el caso del método *ardrone_tool_pause()*, encargado de detener el flujo de datos hacia el cliente *Navdata*, que ahora pasa a llamarse *ardrone_tool_suspend()*.

- En el SDK 2.0, ahora se le da más prioridad a si existe una aplicación custom para que se ejecute antes, mientras que en la versión anterior primero se ejecutaba la principal y posteriormente la custom (figuras 5.7 y 5.8).

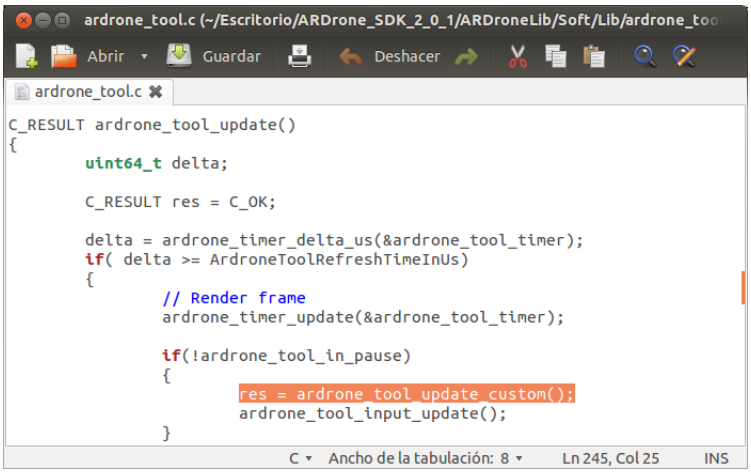

Figura 5.7: *ardrone_tool_update* en SDK 2.0

- Debido a que en la nueva versión se permite utilizar una dirección IP custom, el código se ve adaptado para contener esta función. Básicamente el método *ardrone_tool_main* se ve completamente reconstruido en comparación con el mismo método del SDK 1.6. Esto implica más flexibilidad para el usuario a la hora de realizar aplicaciones custom.

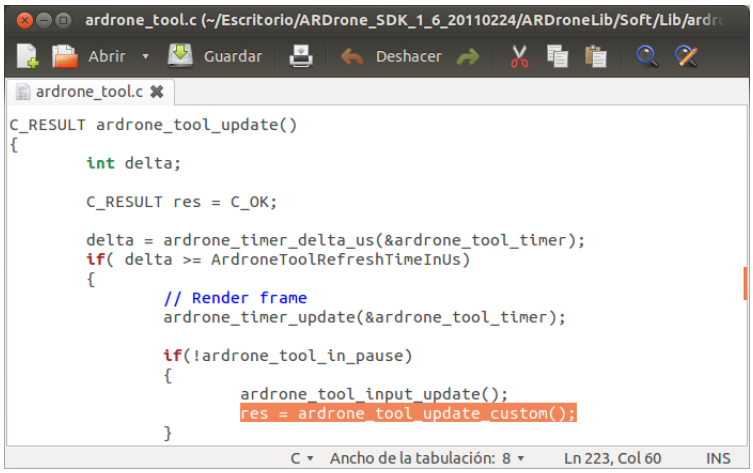

Figura 5.8: *ardrone_tool_update* en SDK 1.6

- El código del SDK 2.0 realiza la comprobación de si se está realizando una aplicación para una versión anterior del AR.Drone o de la actual. Esto viene dado por los ficheros de *ardrone_version* y se comprueba dentro del método *main* de *ardrone_tool*.

En general, los puntos más positivos que nos encontramos con el nuevo SDK 2.0 es la optimización de muchos puntos referidos a la customización de las aplicaciones programadas por el usuario y el sistema de comunicaciones entre drone y máquina. De la misma forma, la unificación de cierto código permite que éste resulte más flexible para cualquier tipo de modo de aplicación (recordemos el modo del 1.6 con *mainloop* y sin él), no solo ceñirse a un modo de ejecución para uno u otro.

Asimismo, la plataforma *Academy* también resulta ventajoso porque resulta una buena forma de acercar entre sí a los usuarios por mediación de los vídeos y fotos que se realizan con el AR.Drone. Podríamos decir que es la parte social de todo el universo AR.Drone.

Sin embargo, entre sus desventajas destaca la opacidad del código: aún con la guía resulta complicado comenzar a comprender ciertos conceptos a los que no estamos familiarizados (como es el caso de los *Navdata* o los *AT Commands*, como funciona el ciclo de vida, etc.). Sin la ayuda de Internet o los foros especializados (a destacar, la página soporte del AR.Drone[4] y GitHub[18]) la tarea de comprender el código es titánica. Debido a ésto puede resultar tedioso ponerse a programar una aplicación para el AR.Drone, ya que si el código y su estructura no resultan claros, la motivación disminuye.

Además, el manual explica los principales puntos a tener en cuenta del AR.Drone, pero hace un análisis muy por encima y no quedan claras algunas cosas que pueden resultar esenciales para comprender el dispositivo, continuamente se tiene la sensación de programar "a ciegas". Las incoherencias en dicho soporte también es algo muy negativo y puede dar lugar a confusión.

La página dedicada al desarrollo de aplicaciones para el AR.Drone (citada arriba), aún resultando ser uno de los apoyos clave para la comprensión del código, también tiene sus puntos negros: no es actualizada con la API de las nuevas versiones. Las partes más activas de toda la página son el foro y la página principal donde están las descargas de las versiones de los SDK.

5.0.9. Con respecto al AR.Drone

En lo que respecta al dispositivo, Parrot ha conseguido sacar un drone de uso recreativo y civil al alcance de dos grupos principales: los **aficionados al aeromodelismo** y los **desarrolladores de aplicaciones**. Primero lo hicieron con la primera versión del AR.Drone 1.0 y el éxito desencadenó la salida de la versión 2.0. Como es habitual en estos casos donde hay gustos compartidos, lo normal es que se busque satisfacer tanto a los aficionados en aeromodelismo como a los desarrolladores. Sin embargo, aquí es donde radica la diferencia en los intereses del AR.Drone.

Si buscamos en internet información sobre el dispositivo de Parrot, podemos ver como la mayor parte de las páginas que nos muestran son sobre *Reviews*, características, vídeos de vuelos, fotografías y poco más. Si profundizamos un poco más en la búsqueda, empezamos a encontrar páginas especializadas en *modding* para el AR.Drone: cómo cambiar las piezas rotas, el motor, modificar la carcasa y customizar físicamente el dispositivo. De aquí hacia adelante, solo nos quedan las poquísimas páginas dedicadas al desarrollo de aplicaciones para nuestro drone, páginas cuya información se nos presenta casi como a cuentagotas.

Ante ésto, un gravísimo error por parte de Parrot es dirigir un dispositivo como es el AR.Drone, con su código notablemente mejorado en su nueva versión 2.0, únicamente al público aficionado al aeromodelismo. De esta forma y debido también a los puntos expuestos en el apartado anterior, todo el potencial de la máquina se ve reducido para el vuelo con la aplicación oficial

de Parrot.

Por esa razón, el AR.Drone resulta práctico como un juego, ya que la información y los recursos que se ponen a disposición del usuario son agradables, completos y actualizados. Nada que ver si el usuario decide programarse una aplicación: tiene que lidiar con la dificultad de un código mal especificado en el manual y poco esclarecedor porque no existe información clara, intuitiva o actualizada para llevar a cabo su programa. Éste resulta ser el punto ciego más notable del AR.Drone, como herramienta de desarrollo se queda algo corta.

No obstante, hay otras desventajas con respecto a la física del AR.Drone. El dispositivo se anuncia con unas características robustas y resistentes, cuando en realidad la carcasa de gomaespuma se rompe ante cualquier caida leve en un entorno *indoors*. Éstos deslices muchas veces provocan que las hélices se vean afectadas de alguna manera y pueden llegar a deformarse ligeramente, perdiendo así estabilidad en el vuelo. En algunas pruebas de vuelo realizadas en un entorno cerrado para nuestro proyecto, las pocas caidas que se tuvieron provocaron que el drone se ladeara ligeramente hacia atrás alterando el momento estático del vuelo.

En definitiva, nos encontramos ante un drone que podría tener un gran potencial de desarrollo. Hablamos de un dispositivo asequible económicamente, con complementos de customización y que prácticamente hasta un niño puede pilotarlo. Podría estar a la altura de los drones experimentales que encontramos en algunas universidades,con sus limitaciones lógicas (licencias). Sin embargo, se queda solo en un dispositivo favorable únicamente para el vuelo.

Capítulo 6

Glosario

- **AT Commands:** La comunicación con el AR.Drone se realiza a través de los AT Commmands, que son cadenas de texto enviados como paquetes TCP y UDP. De esta forma, pueden enviarse órdenes para controlar el dispositivo. Los strings están codificados como caracteres ASCII de 8 bits. La sintaxis es la siguiente:

  AR.Drone 1.0: $AT*[nombreComando] = [n\acute{u}meroDecimal][, argumento1, argumento$
  $LF >$
  AR.Drone 2.0: $AT*[nombreComando] = [n\acute{u}meroDecimal][, argumento1, argumento$
  $CR >$

  Ejemplo: $AT * PCMD = 21625, 1, 0, 0, 0, 0 < LF >$

  Un único paquete UDP puede contener por lo menos un único AT Command o más de uno. La longitud máxima de éstos es de 1024 caracteres y hay un retardo de 30 ms entre uno y otro.

  Lista de AT Commands:

  - $AT * REF = secuencia, entero-$ Despegue / Aterrizaje / Emergencia. El número entero puede descomponerse en binario y los bits 9 y 8 determinan los valores de despegue / aterrizaje y emergencia. 1 cuando está activo y 0 cuando no lo está.
  - $AT * PCMD = flag, roll, pitch, gaz, yaw-$ Mueve el drone.
  - $AT*PCMD\_MAG = flag, roll, pitch, gaz, yaw, psi, psiAccuracy-$ Mueve el drone con control absoluto.
  - $AT * FTRIM-$ Configura la referencia del plano horizontal (el drone debe estar en el suelo o en una superficie plana).

- $AT*CONFIG = key, value-$ Configuración del drone.
- $AT*CONFIG\_IDS = session, use, applicationIds-$ Identificadores para AT*CONFIG
- $AT*COMWDG-$ Resetea la configuración del Watchdog.
- $AT*CALIB = deviceNumber-$ Le pide al drone que calibre el magnetómetro (éste debe estar volando).

AT Commands no documentados [12]:

- $AT*PMODE = secuencia, entero-$ Es el primer comando que aparece al inicializar el drone. Puede estar relacionado con el Progressive Video Mode. Los valores enviados son 1, 2.
- $AT*MISC = secuencia, entero, entero, entero, entero-$ Segundo comando que se configura al inicio. Los valores enviados son 2, 200, 2000, 3000.
- $AT*CTRL = secuencia, entero-$ Este comando está parcialmente documentado. Los valores enviados suelen ser 5, 0.

Otros AT Commands usados solamente en el AR.Drone 1.0:

- $AT*LED = secuencia, entero, float, entero-$ Configura los coeficientes de control de bucle para realizar una *animation* (movimiento predefinido).
- $AT*ANIM = secuencia, entero, entero-$ Hace que el drone ejecute una *animation*).

- **Navdata:** Son una serie de datos que proporcionan información sobre el estado del AR.Drone, tales como ángulos, altitud, cámara, LEDs, velocidad, etc. Se reciben periódicamente (<5 ms), son enviados al puerto UDP 5554 y su estructura es la que muestra la figura 6.1.

Podemos ver como cada estructura tiene un bloque de datos (llamados

Header	**Drone**	**Sequence**	**Vision**	**Option 1**			...	**Checksum block**		
0x55667788	**state**	**number**	**flag**	id	size	data	...	cks id	size	cks data
32-bit	32-bit	32-bit	32-bit	16-bit	16-bit	...	...	16-bit	16-bit	32-bit
int.	int.	int.	int.	int.	int.	...	...	int.	int.	int.

Figura 6.1: Estructura Navdata

"*Options*") denominado *Drone state*, el cual sigue este esquema:

En la figura 6.2 se puede ver como cada uno de los 32 bits hace re-

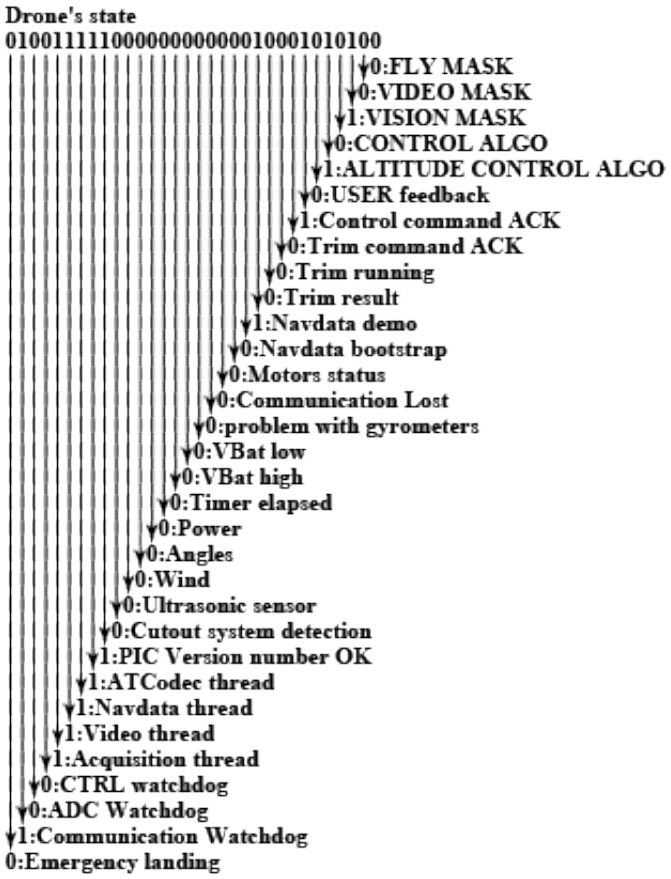

Figura 6.2: Drone state

ferencia a un *flag* en concreto de una serie de estados determinados del drone. Esto es muy utilizado para evitar errores de navegación del dispositivo. En la figura 6.3 se especifica en qué consiste cada *flag*.

- **Realidad Aumentada:** La realidad aumentada (RA) es el término que se usa para definir una visión directa o indirecta de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta en tiempo real.

  Consiste en un conjunto de dispositivos que añaden información virtual a la información física ya existente, es decir, añadir una parte sintética

```
//  Emergency landing : (0) no emergency, (1) emergency

typedef enum {
  ARDRONE_FLY_MASK            = 1 << 0,  /*!< FLY MASK : (0) ardrone is landed, (1) ardrone is flying */
  ARDRONE_VIDEO_MASK          = 1 << 1,  /*!< VIDEO MASK : (0) video disable, (1) video enable */
  ARDRONE_VISION_MASK         = 1 << 2,  /*!< VISION MASK : (0) vision disable, (1) vision enable */
  ARDRONE_CONTROL_MASK        = 1 << 3,  /*!< CONTROL ALGO : (0) euler angles control, (1) angular speed control */
  ARDRONE_ALTITUDE_MASK       = 1 << 4,  /*!< ALTITUDE CONTROL ALGO : (0) altitude control inactive (1) altitude control active */
  ARDRONE_USER_FEEDBACK_START = 1 << 5,  /*!< USER feedback : Start button state */
  ARDRONE_COMMAND_MASK        = 1 << 6,  /*!< Control command ACK : (0) None, (1) one received */
  ARDRONE_FW_FILE_MASK        = 1 << 7,  /* Firmware file is good (1) */
  ARDRONE_FW_VER_MASK         = 1 << 8,  /* Firmware update is newer (1) */
//  ARDRONE_FW_UPD_MASK         = 1 << 9,  /* Firmware update is ongoing (1) */
  ARDRONE_NAVDATA_DEMO_MASK   = 1 << 10, /*!< Navdata demo : (0) All navdata, (1) only navdata demo */
  ARDRONE_NAVDATA_BOOTSTRAP   = 1 << 11, /*!< Navdata bootstrap : (0) options sent in all or demo mode, (1) no navdata options sent */
  ARDRONE_MOTORS_MASK           = 1 << 12, /*!< Motors status : (0) Ok, (1) Motors problem */
  ARDRONE_COM_LOST_MASK       = 1 << 13, /*!< Communication Lost : (1) com problem, (0) Com is ok */
  ARDRONE_VBAT_LOW            = 1 << 15, /*!< VBat low : (1) too low, (0) Ok */
  ARDRONE_USER_EL             = 1 << 16, /*!< User Emergency Landing : (1) User EL is ON, (0) User EL is OFF*/
  ARDRONE_TIMER_ELAPSED       = 1 << 17, /*!< Timer elapsed : (1) elapsed, (0) not elapsed */
  ARDRONE_ANGLES_OUT_OF_RANGE = 1 << 19, /*!< Angles : (0) Ok, (1) out of range */
  ARDRONE_ULTRASOUND_MASK     = 1 << 21, /*!< Ultrasonic sensor : (0) Ok, (1) deaf */
  ARDRONE_CUTOUT_MASK         = 1 << 22, /*!< Cutout system detection : (0) Not detected, (1) detected */
  ARDRONE_PIC_VERSION_MASK    = 1 << 23, /*!< PIC Version number OK : (0) a bad version number, (1) version number is OK */
  ARDRONE_ATCODEC_THREAD_ON   = 1 << 24, /*!< ATCodec thread ON : (0) thread OFF (1) thread ON */
  ARDRONE_NAVDATA_THREAD_ON   = 1 << 25, /*!< Navdata thread ON : (0) thread OFF (1) thread ON */
  ARDRONE_VIDEO_THREAD_ON     = 1 << 26, /*!< Video thread ON : (0) thread OFF (1) thread ON */
  ARDRONE_ACQ_THREAD_ON       = 1 << 27, /*!< Acquisition thread ON : (0) thread OFF (1) thread ON */
  ARDRONE_CTRL_WATCHDOG_MASK  = 1 << 28, /*!< CTRL watchdog : (1) delay in control execution (> 5ms), (0) control is well scheduled */
  ARDRONE_ADC_WATCHDOG_MASK   = 1 << 29, /*!< ADC Watchdog : (1) delay in uart2 dsr (> 5ms), (0) uart2 is good */
  ARDRONE_COM_WATCHDOG_MASK   = 1 << 30, /*!< Communication Watchdog : (1) com problem, (0) Com is ok */
  ARDRONE_EMERGENCY_MASK      = 1 << 31  /*!< Emergency landing : (0) no emergency, (1) emergency */
} def_ardrone_state_mask_t;
```

Figura 6.3: Flags

virtual a lo real. Esta es la principal diferencia con la realidad virtual, puesto que no sustituye la realidad física, sino que sobreimprime los datos informáticos al mundo real. [15]

- **SDK:** *Software Development Kit*. Conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, plataformas de hardware, computadoras, videoconsolas, sistemas operativos, etc. [16]

- **Eventos de *Joystick*** : Entendemos por 'eventos de *Joystick*' a aquellas acciones que se producen en cierto medio cuando se presiona una tecla del *Gamepad*. De normal, cada tecla está relacionada con un número entero, que sirve como referencia a la hora de configurar los eventos.

  En Ubuntu, la forma más fácil de averiguar qué entero corresponde a un evento de *Joystick* es, mediante consola, instalar el paquete 'joystick':

  ```
  sudo apt-get install joystick
  ```

Después, escribiendo el comando:

```
jstest /dev/input/js0
```

Nos saldrá una pantalla donde cada tecla que apretemos del *Joystick* vendrá determinada por un número de entre todos los posibles que permite nuestro dispositivo.

Si nos fijamos en la figura 6.4, nuestro *Joystick* tiene varios valores

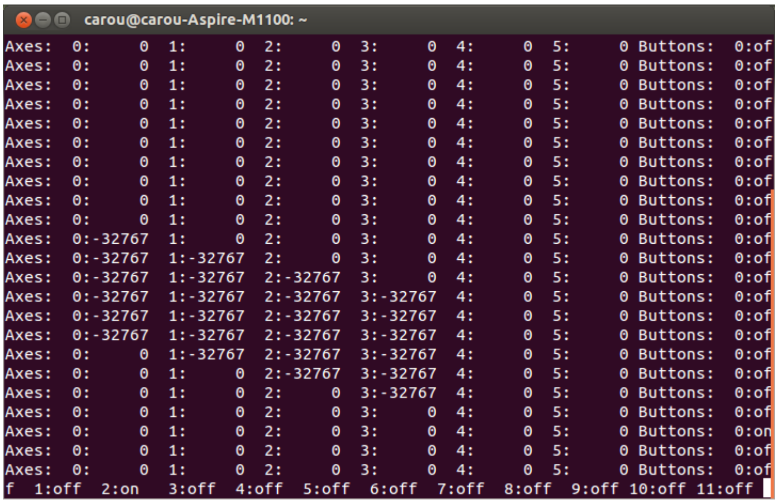

Figura 6.4: *jstest*

agrupados por ejes (*Axes*) y por botones (*Buttons*). En nuestro caso, tenemos 4 ejes y un quinto eje para el control absoluto. El rango de valores que toman los ejes va desde -32767 hasta 32767. [17]

Con respecto a los botones, nuestro dispositivo tiene hasta 12 teclas que se activan o desactivan (*On/Off*) según si les apretamos o no. Todos estos eventos pueden configurarse para el AR.Drone en el fichero *UI/gamepad*.

- **Axis:** Eje principal por el cual se soporta el *Joystick* y a partir del cual realiza los movimientos de dirección. El número que tiene el axis determina el número de direcciones que se mueve.

  Así pues, si se dice que un *Joystick* es de 2 ejes, esto son $2^2 = 4$ movimientos. En el AR.Drone 2.0 se usan 4 ejes, que son hasta 16 movimientos. Sin embargo, los *Joysticks* actuales pueden llegar a tener hasta 6 ejes.

- **Modding:** Arte o técnica de modificar estética o funcionalmente partes de las computadoras, ya sea el gabinete, mouse, teclado o monitor, y los componentes de los videojuegos, como pueden ser las consolas. Puede referirse tanto a las modificaciones al hardware como al software de las mismas, aunque este último también puede llamarse "chipping".

  A todo el que practica o hace el modding se le llama "modder". Sin embargo, la palabra modding se suele usar frecuentemente para las modificaciones realizadas a un ordenador o a algo relacionado con él, como son los periféricos, accesorios e incluso muebles que lo rodean.

  El modding es personalizar los dispositivos y componentes añadiéndoles, modificando o en muy raras ocasiones, sacándole partes, modificando la estructura de la caja ó creando la tuya propia, añadiendo componentes, modificando la forma de estos para obtener mayor espectacularidad y diseño, en definitiva es el arte de darle forma y color al PC poniendo en ello toda la imaginación que se pueda tener.[19]

- **Indoors:** En interior, dentro de un edificio, habitación o cubículo.

- **Outdoors:** En exterior, al aire libre.

Bibliografía

[1] Wikipedia - "*Vehículo Aéreo no Tripulado*"

[2] Historia de los Drones - *http://actualidad.rt.com/actualidad/view/80396-vehiculos-aereos-tripulados-hitos-historicos*

[3] Wikipedia - *Parrot AR.Drone*

[4] Projects AR.Drone - *https://projects.ardrone.org/*

[5] Aerospace Control Laboratory (MIT) - *http://acl.mit.edu/*

[6] Parrot AR.FreeFlight - *http://ardrone2.parrot.com/apps/*

[7] AR.PowerFlight - *https://itunes.apple.com/es/app/ar.powerflight/id392544438?mt=*

[8] Drone Master - *http://drone-apps.com/ios-apps/drone-master/*

[9] Flying Ace - *https://itunes.apple.com/us/app/free-flight/id373065271?mt=8*

[10] Drone Escape - *https://itunes.apple.com/es/app/drone-escape-for-ar.drone/id407360880?mt=8*

[11] Devzone Parrot - *https://devzone.parrot.com/*

[12] Undocumented AR.Drone Commands by bklang - *https://github.com/bklang/ARbDrone/wiki/UndocumentedCommands*

[13] Guía oficial AR.Drone 2.0 - *Disponible en https://projects.ardrone.org/*

[14] Guía oficial AR.Drone 1.0 - *Disponible en https://projects.ardrone.org/*

[15] Wikipedia - "*Realidad Aumentada*"

[16] Wikipedia - "*Software Development Kit*"

[17] Linux Joystick Device - *http://vrjuggler.org/docs/vrjuggler/2.0/configuration.guide/configur*

[18] GitHub - *https://github.com/*

[19] Wikipedia - "*Modding*"