



UNIVERSITAT
POLITÀCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño de un multicoptero para
investigación controlado vía terminal
Android

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Pablo Reyes Pausà

Tutor: Carlos Tavares Calafate

Curso 2016-2017

Resumen

El presente proyecto nace de la necesidad de crear un sistema de control remoto genérico aplicable todos los controladores de vuelo abiertos basados en el protocolo MAVLink, así como de la necesidad de crear un dron del tipo hexacóptero para labores de investigación. Respecto al sistema de control remoto propuesto, éste se basa en terminales móviles de manera que reemplace la emisora de radiocontrol tradicional, lo cual permite ahorrar costes, así como mejorar la seguridad del enlace entre controlador y multicoptero. Para poder realizar dichas funciones, el terminal móvil se enlazará a la controladora de vuelo mediante un computador de placa simple (SBC). Concretamente, la controladora de vuelo PIXHAWK será la encargada de actuar sobre el hardware del dron; en cuanto a la SBC, ésta será una Raspberry Pi 3 que servirá tanto de intérprete entre el dispositivo móvil y la controladora de vuelo, como de dispositivo para proporcionar una conexión WiFi segura entre la Raspberry Pi y el dispositivo móvil Android basándose en el estándar IEEE 802.11a. Con ello se pretende que el dispositivo móvil se convierta en una emisora radiocontrol con plenas funciones. En lo que respecta al hexacóptero desarrollado, tendrá un diseño ad hoc a partir de componentes comerciales con el propósito de lograr un dron con gran autonomía, y lo suficientemente flexible como para permitir ampliarlo según las necesidades de investigación que puedan surgir. Para el diseño del hexacóptero se han utilizado técnicas de impresión 3D para la creación de piezas a medida, y un hardware enfocado a la eficiencia energética. Finalmente, se realizará un ensayo en el cual se comprobará el correcto funcionamiento del hardware y software que componen el dron.

Palabras clave: Dron, Raspberry Pi, PIXHAWK, Android, impresión 3D.

Abstract

The current project arises from the need to create a generic remote control system applicable to all open flight controllers based on the MAVLink protocol, as well as the need to create a drone of the hexacopter type for research purposes. Regarding the proposed remote control system, it is based on mobile terminals in order to replace the traditional radio transmitter, which saves costs and allows improving the security of the link between controller and multicopter. In order to perform such functions, the mobile terminal will be linked to the flight controller through single board computer (SBC). Specifically, the PIXHAWK flight controller will be responsible for controlling the drone hardware. As for the SBC, this will be a Raspberry Pi 3 that will serve as both interpreter between the mobile device and the flight controller, and a device to provide a secure WiFi connection between the Raspberry Pi and the Android mobile device based on the IEEE 802.11^a standard. With this, it is intended that the mobile device becomes a radio transmitter with full functions. This way, it will have an ad hoc design from commercial components for the purpose of achieving a dron with great autonomy, and flexible enough to allow it to expand according to the research needs that may arise. For the design of the hexacopter, 3D printing techniques have been used to create custom pieces, and the hardware was focused on energy efficiency. Finally, a test was performed where the hardware and software that compose the drone were tested.

Keywords: Drone, Raspberry Pi, PIXHAWK, Android, 3D printing.



Índice

Índice	4
Índice de tablas	6
Índice de ilustraciones	7
1. Introducción	9
1.1 Objetivos	10
1.2 Estructura del documento	11
2. Situación actual	12
3. Solución propuesta.....	13
4. Hardware	16
4.1 Propulsores eléctricos	16
4.2 Variadores	17
4.3 Hélices	18
4.4 Batería.....	18
4.5 Computadores	20
4.5.1 PIXHAWK.....	20
4.5.2 Raspberry Pi	21
4.6 Conexión WiFi.....	22
4.7 Emisora radiocontrol	23
4.7.1 Receptor radiocontrol	25
4.8 Chasis.....	25
4.9 Ensamblaje	26
4.10 Impresión 3D	29
4.11 Configuración PIXHAWK vía Mission Planner.....	30
5. Software Raspberry Pi 3.....	35
5.1 Sistema operativo	35
5.1.1 Desactivar la salida de consola por puerto serie	35
5.1.2 Configuración JAVA.....	36
5.1.3 Configuración Puerto Serie	37
5.2 Configuración Puerto Serie PIXHAWK	38
5.3 Comunicación Raspberry Pi vía MAVLink	38
5.4 Punto de acceso mediante Raspberry Pi.....	39
5.4.1 Configuración del servidor DHCP	40
5.4.2 Configuración del Punto de Acceso	42



5.4.3	MAC: interfaces wlan0 y eth0.....	44
5.4.4	Daemon de inicio en Raspberry Pi.....	45
6.	Aplicación Android control remoto	46
6.1	Android manifests.....	46
6.2	Android res	49
6.3	Android JAVA	57
7.	Aplicación de enlace entre RPi y PIXHAWK	70
8.	Validación.....	76
9.	Conclusiones	77
	Trabajos futuros	78
	Bibliografía	79
	Anexo	81



Índice de tablas

Tabla 1 - Paquete MAVLink	15
Tabla 2 - Especificaciones técnicas propulsores DJI	17
Tabla 3 - Especificaciones técnicas variadores	17
Tabla 4 - Comparativa hélices	18
Tabla 5 - Voltaje máximo y mínimo de una celda	19
Tabla 6 - Clasificación baterías LIPO	19
Tabla 7 - Características batería.....	19
Tabla 8 - Características PIXHAWK	20
Tabla 9 - Características Raspberry Pi	22
Tabla 10 - Características emisora radiocontrol	23
Tabla 11 - Mapa de canales RC con movimientos.....	66



Índice de ilustraciones

Ilustración 1 - Esquema conexión.....	14
Ilustración 2 - Esquema Dron [7].....	16
Ilustración 3 - Propulsor eléctrico DJI.....	17
Ilustración 4 - Variador.....	17
Ilustración 5 - Hélices 9045	18
Ilustración 6 - Batería	19
Ilustración 7 - Puertos PIXHAWK	21
Ilustración 8 - Raspberry Pi 3.....	22
Ilustración 9 - Antena WiFi.....	23
Ilustración 10 - Emisora radiocontrol, controles	24
Ilustración 11 - Emisora radiocontrol, posición de la antena	24
Ilustración 12 - Receptor radiocontrol.....	25
Ilustración 13 - Chasis F550.....	25
Ilustración 14 - Conexión entre PCB, variador y motor	26
Ilustración 15 - Conexión receptor, antena GPS y conexión de un motor	27
Ilustración 16 - Conexión a batería y PIXHAWK	27
Ilustración 17 - Conexión de datos entre Raspberry Pi y PIXHAWK	28
Ilustración 18 - Esquema de conexión para la alimentación de Raspberry Pi y antena WiFi....	28
Ilustración 19 - Piezas para impresión 3D	29
Ilustración 20 - Mission Planner, “Initial Setup”	30
Ilustración 21 - Mission Planner, opción “Wizard”	31
Ilustración 22 - Mission Planner, tipo de aeronave.....	31
Ilustración 23 - Mission Planner, tipo de chasis	32
Ilustración 24 - Mision Planner, calibrar acelerómetro	32
Ilustración 25 - Mision Planner, calibrado GPS	33
Ilustración 26 - Mision Planner, calibrado emisora radiocontrol	33
Ilustración 27 - Mission Planner, modos de vuelo.....	34
Ilustración 28 - Mision Planner, Configuración “Geo fence”	34
Ilustración 29 - Menú raspi-config.....	35
Ilustración 30 - Raspberry Pi JAVA inicial	36
Ilustración 31 - JAVA instalado en Raspberry Pi.....	36
Ilustración 32 - GPIO Raspberry pi 3.....	37
Ilustración 33 - Serial Port Raspberry Pi swaped	38
Ilustración 34 - Estructura proyecto Android.....	46
Ilustración 35 - Icono aplicación	49
Ilustración 36 - inicio.xml.....	51
Ilustración 37 - Layout Control Remoto.....	55
Ilustración 38 - Layout Información	56
Ilustración 39 - Esquema de envío mediante paquetes.....	61
Ilustración 40 - Parámetros influyentes en los joystick	66
Ilustración 41 - Relación entre posición del joystick izquierdo y canales RC	66
Ilustración 42 - Relación entre posición del joystick derecho y canales RC.....	68
Ilustración 43 - Tipo de paquetes.....	70
Ilustración 44 - Proyecto Java RPi	70



Ilustración 45 - Prueba de vuelo 1	76
Ilustración 46 - Prueba de vuelo 2	76

1. Introducción

El primer concepto que se debe conocer es: ¿Qué es un Dron?, Aunque este término puede realmente hacer referencia a cualquier sistema autónomo no tripulado, en el ámbito de este proyecto asumimos la definición más común, que hace referencia a un vehículo aéreo no tripulado (UAV) el cual puede ser utilizado en muchos ámbitos distintos, desde el entretenimiento, la investigación, la agricultura... hasta para usos bélicos. En la actualidad es uno de los campos de investigación que más avances está experimentando no solo por el interés que despierta, sino también por la reducción de costes que ha tenido en los últimos años.

En general se pueden encontrar drones especializados de todo tipo, y que se encargan de tareas muy variadas, tales como:

- Vigilancia
- Misiones de rescate
- Inspecciones técnicas
- Modelado 3D
- Cine
- Fotografía

Las razones que llevan a que estos dispositivos sean cada vez más adoptados tienen que ver con sus numerosas ventajas:

- Bajo coste
- Elevados niveles de movilidad
- Capaz de permanecer inmóvil en una posición fija
- Potente
- Capaz de llevar diferentes tipos de dispositivos
- Fácil de controlar / administrar

Los drones de uso público se caracterizan por utilizar varios motores, desde 4 hasta 16, con pesos y tamaños contenidos, lo que ofrece una versatilidad muy alta en su uso. Podemos encontrar drones de distintas cualidades y configuraciones. Para este proyecto se opta por un hexacóptero que está formado por 6 motores para su propulsión, lo que ofrece una alta estabilidad.

Como toda tecnología emergente, el uso de drones presenta posibilidades de mejora, y una de esas posibles mejoras es la comunicación entre el piloto y la aeronave, ya que estudios recientes destacan que es posible que un atacante con el hardware adecuado se adueñe de cualquier multicóptero comercial. Este proyecto se centra en la posibilidad de mejorar dicha comunicación, gracias al uso de un hardware y un software para dicho propósito.

1.1 Objetivos

El principal objetivo del proyecto es la construcción de un dron y el desarrollo de un enlace seguro entre la aeronave y el controlador remoto mediante una conexión WiFi a 5GHz. Se parte de la necesidad de crear un dron para usos en investigación por ello la aeronave deberá cumplir unas especificaciones descritas a continuación.

En complemento a este primer objetivo, se marca una serie de objetivos secundarios, que no obstante son importantes para el desarrollo del proyecto.

El primero de ellos es la elección del hardware pertinente para construir una aeronave con una alta capacidad de tiempo de vuelo. Con este objetivo se pretende disponer de una aeronave para la investigación que sea útil en obtención de datos y en su uso gracias a esta duración de vuelo.

El segundo objetivo es la modularidad. Con ella se pretende que el dron sea fácil de reparar y de modificar. Este objetivo resulta de gran utilidad porque durante las pruebas existe la posibilidad de encontrar errores y al tratarse de un vehículo aéreo cualquier fallo puede causar la destrucción de alguno de sus componentes.

El último objetivo es la apertura de una ventana de comunicación entre el protocolo utilizado en las controladoras de vuelo (*Mavlink*) y los sistemas capaces de ejecutar JAVA. En este caso el dispositivo es un terminal Android, pero podría ser portado a otros dispositivos que utilicen esta tecnología.

1.2 Estructura del documento

El presente documento se divide en cuatro grandes bloques. En el primero de ellos se describe el hardware utilizado en la aeronave. En él se justifica la utilización de los componentes que componen la aeronave y cuál es la función de cada uno. Además se explicará de forma breve como se configura la controladora de vuelo PIXHAWK.

El segundo bloque, Software Raspberry Pi 3, describe los pasos necesarios para disponer de un enlace software entre la RPi y la controladora de vuelo. Para poder realizar este enlace se configurará el sistema operativo, que quedará descrito en el punto 5.1. A continuación se activará el puerto serie y por último en el punto 5.4 se realizará la conexión segura entre el terminal y el dron.

El tercer bloque (punto 6) contiene la información correspondiente al proyecto Android. En él se describe la funcionalidad implementada y las interfaces utilizadas para el desarrollo de la aplicación. También se incluye el protocolo utilizado en el envío de comandos *Mavlink* a través de ésta.

El punto 7 representa el último bloque, y abarca el software encargado de realizar la función de intermediario. En este punto, se muestra como el software desarrollado encamina los datos recibidos hacia la controladora de vuelo mediante *Mavproxy*.

Una vez los principales bloques relativos al desarrollo del sistema se han presentado, en el punto 8 presentamos la validación del sistema desarrollado. Por último, en el punto 9, presentamos las conclusiones del trabajo y posibles trabajos futuros.

2. Situación actual

En la actualidad no cabe duda que los drones están en auge y sus ventas han crecido exponencialmente, ya sea como herramienta de ocio o como objeto de investigación. En sus inicios los drones o más concretamente los vehículos aéreos no tripulados (UAV) eran un campo de uso exclusivo militar, pero poco a poco se ha ido reduciendo su coste de construcción favoreciendo la popularización de estos.

En España se está creando el primer dron de uso militar del país, llamado CEUS [1]. Estos drones tienen unas cualidades y una precisión extraordinarias pero su construcción tiene costes de millones de euros. El dron utilizado en el presente proyecto, así como los drones de uso público son muy distintos a los militares. A pesar de también ser controlados remotamente los drones de uso público tienen un tamaño menor y un coste ínfimo en comparación a los militares. Además, los primeros adoptan por lo general la forma de multicópteros, utilizando varios motores en posición horizontal para su propulsión.

Al tratarse de una tecnología novedosa y muy popular, está sufriendo una evolución constante. Podemos encontrar drones capaces de soportar golpes a altas velocidades como el desarrollado por *NCCR Robotics* [2]. Otros llevan nuestras compras online a la puerta de casa, como los desarrollados por *AMAZON*, los cuales ya están siendo utilizados en EEUU y permiten hacer envíos de productos en pocas horas [3]. Incluso existen drones encargados de llevar medicinas a áreas de difícil acceso o en estado de guerra, que con un coste de 10\$ podrían ser lanzadas en masa. Este proyecto está siendo desarrollado por *DARPA* y pretende ser puesto en marcha este mismo año [4].

La lista podría ser casi infinita, como se puede observar, y cualquier producto existente o nuevo se está intentando acoplar a estas aeronaves por la versatilidad que ofrecen y por su vista aérea.

Otros usos más enfocados al ocio son los drones comunes como los creados por la empresa *DJI*, que suelen ser utilizados para grabaciones aéreas. Estos están contruidos y configurados para que el comprador solo tenga que abrir la caja y utilizarlo. También podemos encontrar una nueva moda asociada a los drones: los drones de carreras. Estos son de un tamaño muy reducido pero alcanzan altas velocidades, rozando los 160km/h. En la Universidad Politécnica de Valencia se ha desarrollado el primero de ellos impreso íntegramente en 3D [5].

El principal problema que un usuario puede encontrarse a la hora de construir o adquirir un dron es la legislación vigente en nuestro país. Ésta limita extremadamente el uso de estas aeronaves y dificulta su utilización ya sea comercial o de ocio. En resumen, la ley actual prohíbe el vuelo nocturno, volar en zonas de uso aéreo, vuelos superiores a 120m y una distancia máxima de vuelo limitada.

Por último cabe destacar que todos ellos comparten una vulnerabilidad: el enlace entre la aeronave y el control remoto que fácilmente puede ser explotada [6]. Es por este motivo que, en el presente proyecto, se aportará una solución a dicha problemática.



3. Solución propuesta

En este proyecto se busca desarrollar una solución que incluya un multicoptero y un terminal móvil, y que sea capaz de cumplir con una serie de requisitos:

- Tiempo de vuelo elevado.
- Capacidad de recuperación tras el fallo de un motor.
- Soportar una carga superior a 600g (excluyendo su propio peso).
- Alta estabilidad.
- Modularidad.
- Soporte de desarrollo.
- Controladora abierta.
- Comunicación segura entre dron y control remoto.

Por sencillez, el proyecto se divide en dos bloques, donde en el primero se abordan los componentes hardware utilizados para la composición del dron, y en el segundo bloque se detalla el software desarrollado.

En lo que respecta a hardware, un dron se compone básicamente de un sistema de propulsión, uno de control y uno de alimentación. Para el presente proyecto optamos por un hexacóptero como plataforma. Como su nombre indica está dotado de seis motores para su propulsión. Se utiliza esta configuración para asegurarnos de lograr una alta estabilidad, esto es así porque cuantos más motores tenga un dron, más estable será pero menos rápido y ágil. Otro punto a favor de los hexacópteros es su capacidad de recuperación, ya que si uno de los motores falla, con el resto de motores sería posible controlar la aeronave y aterrizar sin problemas.

Para cumplir el objetivo de tener una elevada duración de vuelo, está dotado de una batería de gran capacidad. Esta batería junto a unas hélices de tamaño considerable, aportarán el tiempo de vuelo necesario para las pruebas de investigación.

La potencia de propulsión debe ser lo suficientemente alta como para poder soportar su propio peso más los añadidos que se tienen previstos. Estos serán una Raspberry Pi, una antena WiFi y todas las protecciones impresas en 3D para ellas. Además, deberá tener la suficiente potencia como para poder añadir más peso si en un futuro se desea. Por todo lo expuesto anteriormente se construirá un dron capaz de levantar 3Kg sin contar su propio peso. En el [*Anexo*](#) se pueden consultar los datos técnicos y los cálculos realizados de el dron.

El segundo bloque consta del software desarrollado para satisfacer el objetivo del proyecto, y para ello se crea un canal de comunicación entre el dron, la Raspberry Pi y el terminal Android (ver Ilustración 1).

Para poder realizar una comunicación entre la Raspberry Pi y la PIXHAWK, se utilizará el protocolo *Mavlink*. Este se podrá usar gracias al programa *Mavproxy*, el cual establecerá la comunicación entre las controladoras. Por otra parte, se crea un enlace entre el terminal Android y la Raspberry Pi usando un canal WiFi seguro.

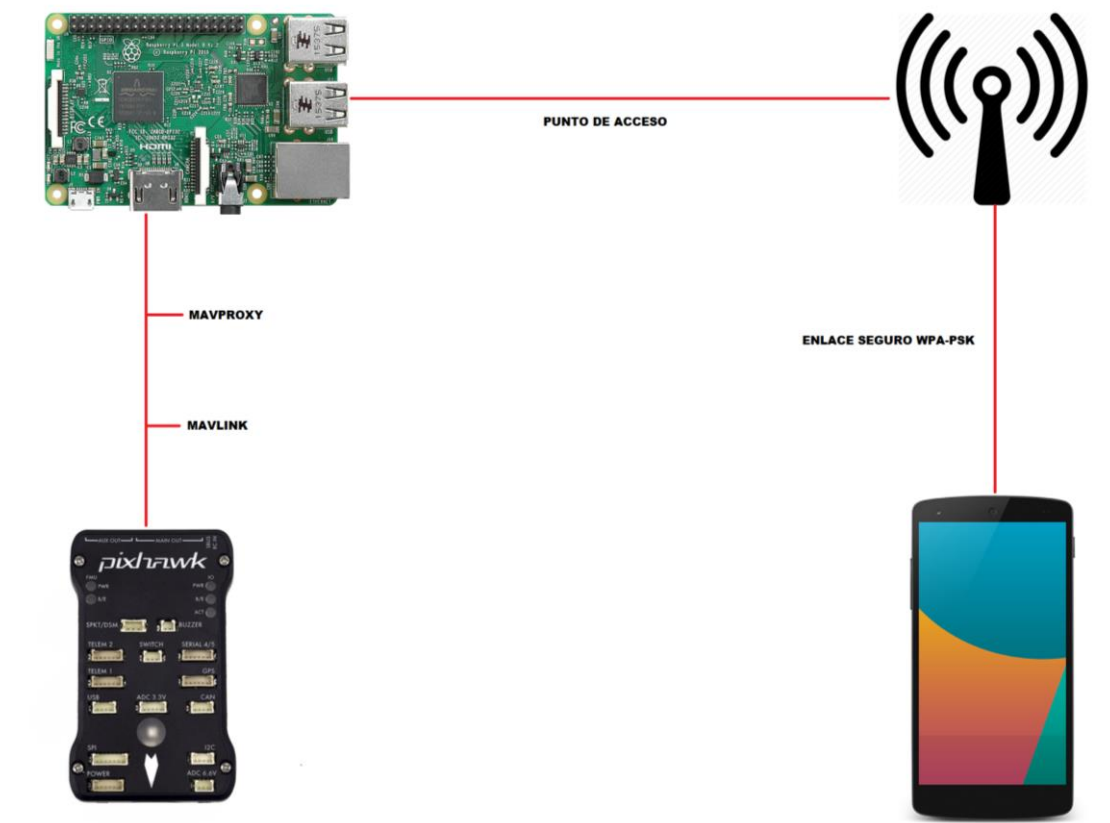


Ilustración 1 - Esquema conexión

Entrando en más detalle, se creará un punto de acceso WiFi mediante la Raspberry Pi, el cual será adjudicado a una antena externa, y no a la interna, para así poder utilizar la frecuencia de 5GHz. Este enlace se utilizará como vía de comunicación entre la Raspberry Pi y el controlador remoto Android. Este controlador será el encargado de emitir comandos del tipo MAVLink, que posteriormente serán tratados por el software de la Raspberry Pi y se encaminarán hacia el programa Mavproxy.

El enlace entre terminal móvil y Raspberry Pi usa como protocolo de seguridad WPA, el cual utiliza un método de jerarquía de claves denominado *Protocolo de integridad de clave temporal* (TKIP) para proteger la clave maestra de pares compartida (PMK). La clave maestra nunca se transmite a través de la red, sino que se almacena en cada dispositivo, y el cifrado se utiliza para generar otras claves temporales denominadas *Temporal Keys*. Estas claves temporales incluyen claves de sesión, claves de grupo y claves de cifrado de paquetes. Con la utilización de este protocolo nos aseguramos una seguridad no proporcionada en la conexión estándar entre dron y control remoto.

MAVProxy es el encargado de enviar los comandos a la PIXHAWK. Éste software es un GCS (Ground Control Station) completamente funcional para UAVS. La intención es disponer de una estación de control terrestre minimalista, portátil, y extensible para cualquier UAV que soporte el protocolo MAVLink (como la PIXHAWK). Es por este motivo, su compatibilidad, por el que se elige.

Respecto al MAVLink, o *Micro Air Vehicle Link* es un protocolo para comunicarse con pequeños vehículos no tripulados, siendo el protocolo más utilizado en las controladoras de vuelo actual.

La estructura que siguen los paquetes MAVLink es la siguiente:

NOMBRE DEL CAMPO	INDEX (BYTES)	USO
START-OF-FRAME	0	Marca el inicio del frame (v1.0: 0xFE)
PAYLOAD-LENGTH	1	Longitud del payload (n)
PACKET SEQUENCE	2	Cada componente cuenta su secuencia de envío. Permite la detección de pérdida de paquetes.
SYSTEM ID	3	Identificación del sistema envío. Permite diferenciar diferentes sistemas en la misma red.
COMPONENT ID	4	Identificación del componente de envío. Permite diferenciar diferentes componentes del mismo sistema.
MESSAGE ID	5	Identificación del mensaje - el identificador define lo que el payload "significa" y cómo se debe decodificar correctamente.
PAYLOAD	6 to (n+6)	La información en el paquete.
CRC	(n+7) to (n+8)	Check-sum del paquete

Tabla 1 - Paquete MAVLink

En el envío de datos desde el terminal Android, no se tratarán los datos como paquetes sino que se crearán *Strings* con órdenes predeterminadas existentes en el protocolo MAVLink para el envío de comandos.

Una vez finalizados los pasos anteriores se procederá a crear la aplicación Android que se encargará de enviar las órdenes al software desarrollado en la Raspberry Pi.

En los siguientes apartados se detallará el proceso de desarrollo y configuración de todo el hardware y software involucrado en este proyecto.



<i>Descripción</i>	<i>Datos</i>
KV	920KV
Rpm	13616rpm
Amperios	15A
Peso	54g
Alimentación	14.8V (4S)
Empuje	1100g
Consumo	217W

Tabla 2 - Especificaciones técnicas propulsores DJI



Ilustración 3 - Propulsor eléctrico DJI

4.2 Variadores

Los variadores son los encargados de regular la velocidad de giro del motor, para ello transforman el voltaje que obtienen de la batería (DC) a corriente alterna (AC).

Se puede observar en la *Ilustración 4*, como del propio variador salen tres cables dirigidos hacia el motor, dos que deben ser conectados a la batería de la aeronave, y un cable de un calibre más fino, que está compuesto a su vez por 3 cables, y que debe conectarse al controlador de la aeronave. Este cable es el encargado de enviarle la información sobre la aceleración que debe dar el variador al motor.

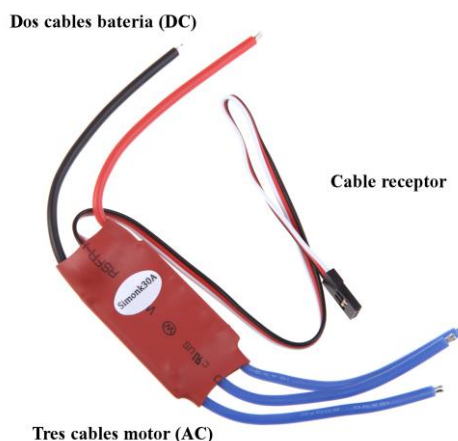


Ilustración 4 - Variador

Los variadores utilizados cuentan con un microcontrolador de tamaño reducido, el cual debe ser programado. Más adelante veremos cómo se debe de hacer dicha programación.

Las especificaciones técnicas del variador utilizado son las siguientes:

<i>Descripción</i>	<i>Datos</i>
Corriente máxima continua	30A
Corriente máxima (10s)	40A
Peso	28g
Baterías aceptadas	2-4S

Tabla 3 - Especificaciones técnicas variadores

4.3 Hélices

Las hélices son uno de los elementos más importantes para el sistema de propulsión. Dos parámetros son clave para la elección de las hélices:

- Paso de hélice
- Longitud de hélice

Cuanto mayor sea el paso de hélice, mayor caudal de aire mueve y mayor empuje tendrá el motor, pero el incremento del paso causa un consumo eléctrico mayor, por ello hay que tener en cuenta el segundo parámetro. La longitud de la hélice causa un efecto similar ya que a mayor longitud mayor empuje, pero mayor consumo, por ello debemos encontrar un equilibrio entre ambos parámetros.

Para esta configuración se van a utilizar unas hélices DJI 9450 (9,4 pulgadas longitud y 4,5 el paso de la hélice) ya que éstas son las más equilibradas en cuanto a consumo y empuje para el hexacóptero.

La conclusión anterior es alcanzada tras la comparativa de resultados obtenida con distintas configuraciones. En la siguiente tabla podemos observar los valores obtenidos en dichas pruebas:

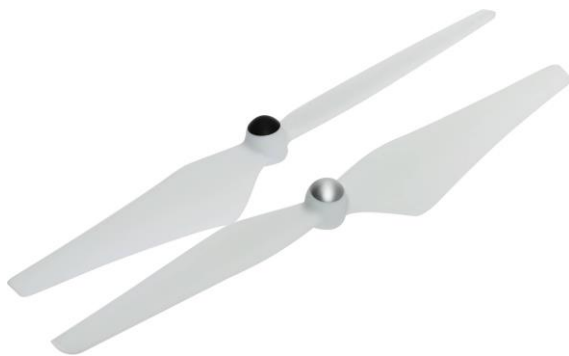


Ilustración 5 - Hélices 9045

<i>Hélice</i>	<i>Consumo</i>	<i>Empuje</i>
8045	195W	900g
9450	210W	1034g
1045	225W	1100g

Tabla 4 - Comparativa hélices

4.4 Batería

La batería seleccionada es de polímeros de litio (LiPo) [8]. Éste tipo de baterías están formadas por células idénticas en paralelo para aumentar su capacidad y su voltaje. Esta tecnología ofrece mayor tasa de descarga, menor peso, y tamaños más reducidos frente a las baterías convencionales utilizadas para estas aeronaves (NiMH) [9].

Las celdas tienen un voltaje cambiante, como se observa en la *Tabla 5*, siendo el voltaje mayor cuanto mayor es la carga.

Voltaje	Carga
3.65V	10%
3.7V	20%
3.75V	40%
3.82V	50%
3.85V	60%
3.9V	80%
4.2V	100%

Tabla 5 - Voltaje máximo y mínimo de una celda

Según la cantidad de celdas que combinemos obtendremos baterías de mayor o menor voltaje. En la *Tabla 6* observamos la clasificación de todas ellas.

Descripción	Voltaje
1S	3.7V
2S	7.4V
3S	11.1V
4S	14.8V
6S	22.2V

Tabla 6 - Clasificación baterías LIPO

Se debe tener precaución de nunca sobrepasar el voltaje máximo y de nunca bajar del voltaje mínimo, de lo contrario podríamos dañar la batería.

El modelo concreto utilizado consta de las siguientes características:

Descripción	Datos
Capacidad	9000mAh
Tasa descarga	25C
Voltaje	14.8V (4S)

Tabla 7 - Características batería



Ilustración 6 - Batería

Como se observa en la *Tabla 7*, aparece un nuevo parámetro llamado “Tasa descarga”. Este parámetro especifica la velocidad de descarga máxima que soporta la batería. Con el valor de la “Capacidad” y “Tasa descarga” se puede obtener el amperaje máximo que puede soportar la batería:

$$A_{Max} = Tasa\ descarga * Capacidad(A)$$

En este caso el amperaje máximo sería de 225A.

4.5 Computadores

Un multicoptero suele estar gobernado por un solo computador llamado controlador de vuelo. Estos computadores son de una potencia muy limitada y tienen poca capacidad de ampliación, por ello, se van a utilizar dos tipos distintos de computadores. Uno de ellos se encargará de ejercer como controlador de vuelo propiamente dicho (PIXHAWK), y el otro computador será el que nos ofrecerá una potencia de computación mayor y las posibilidades de ampliación, así como comunicación vía WiFi (Raspberry Pi).

4.5.1 PIXHAWK

La controladora PIXHAWK es un pequeño computador dotado de una serie de sensores capaces de estabilizar la aeronave, posicionar por GPS y de transferir datos de telemetría. Es capaz de controlar todo tipo de aeronaves de radiocontrol.

Esta controladora ofrece posibilidades de conexión con la Raspberry Pi a través de su puerto de telemetría secundario. Actualmente es la controladora más potente del mercado en la cual es posible realizar dicha conexión. Sus características principales son:

<i>Descripción</i>	<i>Datos</i>
Procesador	<ul style="list-style-type: none">• 32-bit ARM Cortex M4 core con FPU• 168 Mhz/256 KB RAM/2 MB Flash• 32-bit failsafe coprocesador
Sensores	<ul style="list-style-type: none">• MPU6000 como acelerómetro y giróscopo principal• ST Micro 16-bit como giróscopo secundario• ST Micro 14-bit acelerómetro\brújula• MEAS barómetro
Alimentación	<ul style="list-style-type: none">• Opción 1 → Utilizar la energía proveniente de los ESC• Opción 2 → Utilizar cable de alimentación directo a placa• Opción 3 → Combinación de ambas (Elegida en la investigación)
Puertos	<ul style="list-style-type: none">• 5x UART puertos serie• Spektrum DSM/DSM2/DSM-X Conexión GPS• Futaba S.BUS salida• PPM sum señal• RSSI (PWM) entrada• I2C, SPI, 2x CAN, USB• 3.3V y 6.6V ADC entradas
Dimensiones	<ul style="list-style-type: none">• Anchura 50 mm (2.0")• Altura 15.5 mm (.6")• Longitud 81.5 mm (3.2")

Tabla 8 - Características PIXHAWK

Como se observa en la *Tabla 8*, dispone de un alto número de puertos. En la siguiente ilustración se pueden observar cada uno de ellos.

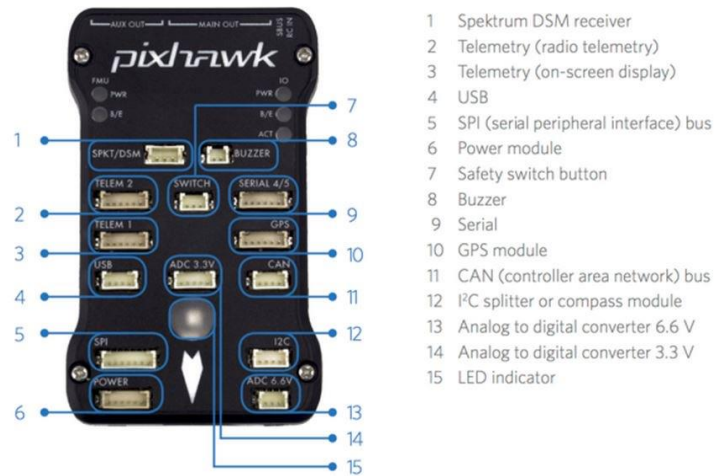


Ilustración 7 - Puertos PIXHAWK

4.5.2 Raspberry Pi

La Raspberry Pi es un computador de tamaño reducido que ofrece una gran potencia en comparación a la controladora de vuelo PIXHAWK. Sus características principales son la compatibilidad con sistemas operativos basados en UNIX y su modularidad.

La conectividad WiFi y su puerto serie son las características que se expresarán para poder desarrollar el software que será capaz de comunicar un terminal Android a cualquier controladora de vuelo basado en el protocolo MAVLink. En la *Tabla 9* se observan las características de la Raspberry Pi 3, que es la que se utilizará en la investigación.

<i>Descripción</i>	<i>Datos</i>
Socket	Broadcom BCM2837
CPU	4× ARM Cortex-A53, 1.2GHz
GPU	Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Conexiones de red	10/100 Ethernet, 2.4GHz 802.11n WiFi, Bluetooth 4.1
Almacenamiento	microSD
GPIO	40-pin
Puertos	HDMI, 3.5mm audio-video Jack analógico, 4× USB 2.0, (CSI), (DSI)

Tabla 9 - Características Raspberry Pi



Ilustración 8 - Raspberry Pi 3

4.6 Conexión WiFi

Para enlazar el terminal Android a la Raspberry Pi por vía inalámbrica existen dos posibilidades:

1. Utilizar el chip WiFi integrado en la propia Raspberry Pi.
2. Utilizar un dispositivo externo.

Se opta por la utilización de un dispositivo externo ya que el chip integrado en la placa solo puede utilizar la frecuencia de 2.4 GHz. En general el uso de la banda de 2.4 GHz sería preferible ya que permite lograr un mayor rango de alcance respecto a la banda de 5 GHz [10].

No obstante, hay un factor que se debe de tener en cuenta: las emisoras radiocontrol utilizadas en los multicopteros funcionan a 2.4 GHz y funcionan de una manera peculiar, ya que hacen un barrido constante por todos los canales de 2.4 GHz a máxima potencia saturando dichos canales. Esto se traduce en fallos de conexión en todo dispositivo cercano a ellos.

Antes de optar por la opción de los 5GHz WiFi se pudo realizar un ensayo en el cual se apreciaba como la emisora realizaba el barrido descrito anteriormente, por ello se decidió descartar el chipset interno y optar por el dispositivo externo que funcionase en 5GHz.

El dispositivo externo es una antena WiFi con conexión a través de USB capaz de soportar los protocolos IEEE 802.11a/b/g y IEEE 802.11n, además de ser compatible con sistemas operativos basados en UNIX. Otras características son:

- Chipset Ralink RT3572.
- Velocidades de transmisión hasta 150 Mbps.



Ilustración 9 - Antena WiFi

4.7 Emisora radiocontrol

Como equipo de radiocontrol se utiliza una emisora de la compañía *Taranis*. Se trata de una emisora genérica que se utilizará en investigaciones posteriores para el control remoto, y que en la investigación actual se utilizará de emisora de seguridad por si el sistema desarrollado en la investigación fallase de manera imprevista.

Sus especificaciones se pueden ver resumidas en la *Tabla 10*.

<i>Descripción</i>	<i>Datos</i>
<i>Transmisor</i>	FrSky X9D con telemetría
<i>Frecuencia de transmisión</i>	2,4GHz
<i>Nº canales</i>	16
<i>Alimentación</i>	6 a 15V NiMh
<i>Corriente máxima</i>	260mA
<i>Temperatura de operación</i>	-10 ~ 60 °C
<i>Almacenamiento</i>	60MB ampliable vía SD
<i>Modo de operación</i>	Modo 2 (Acelerador mano izquierda)
<i>Alcance</i>	Entre 2 y 3 Km

Tabla 10 - Características emisora radiocontrol

En la *Ilustración 10* podemos observar el aspecto de la emisora y todos sus dispositivos de control:

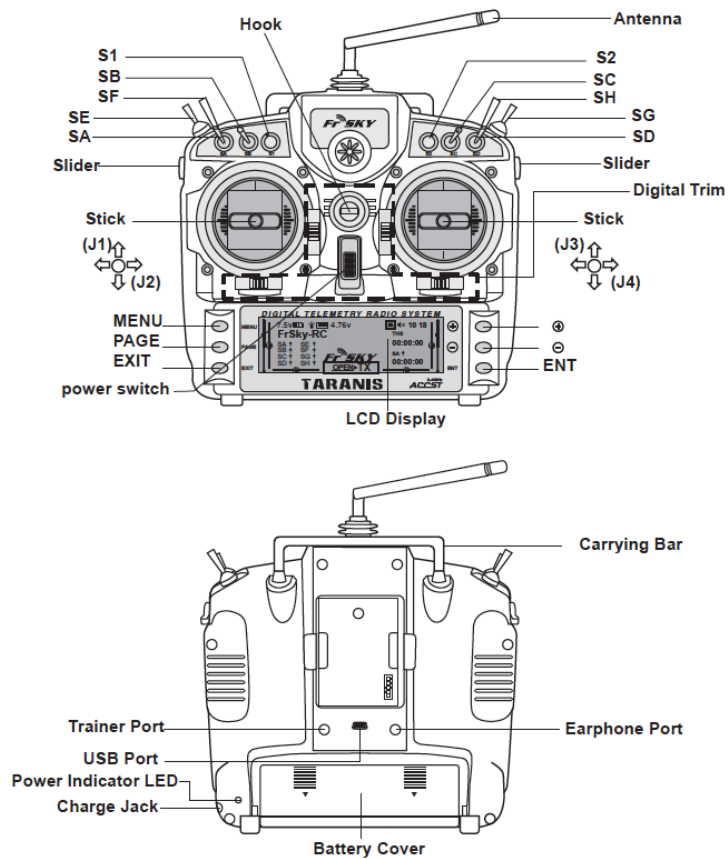


Ilustración 10 - Emisora radiocontrol, controles

En lo que respecta a la antena, esta se debe orientar de forma que se optimicen las comunicaciones con la aeronave, como se indica a continuación.

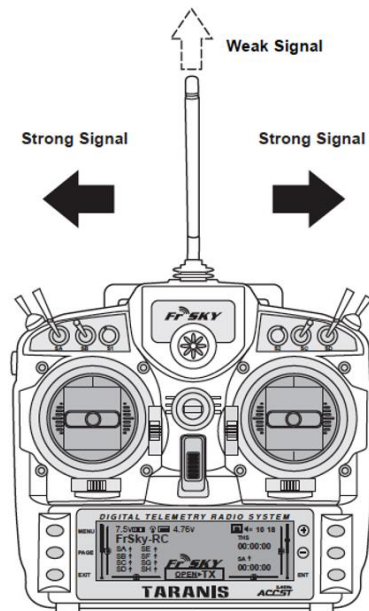


Ilustración 11 - Emisora radiocontrol, posición de la antena

4.7.1 Receptor radiocontrol

Un receptor es un dispositivo encargado de recibir las señales provenientes de la emisora radiocontrol y transmitir dichas señales a los variadores y demás dispositivos que se conecten a ella, como pueden ser servos, iluminación led o cualquier dispositivo que precise de una señal para actuar. Su característica principal son los canales, los cuales indican la cantidad máxima de dispositivos que son capaces de controlar al mismo tiempo. Por ejemplo, un receptor dotado de 16 canales es capaz de controlar 16 dispositivos.

Aunque para esta investigación no se vaya a usar directamente este dispositivo como dispositivo de recepción, esta sería la manera estándar de realizar la conexión entre emisora y aeronave. Por esa razón, se dotará igualmente a la aeronave de dicho componente para poder ser utilizado en futuras investigaciones.

En esta investigación se usará como elemento receptor principal la red WiFi creada mediante la Raspberry Pi, porque dichos receptores no son de código abierto y no permiten modificación alguna.

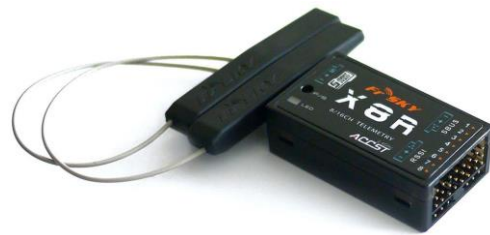


Ilustración 12 - Receptor radiocontrol

4.8 Chasis

Se utilizará como chasis el modelo *DJI F550*, que se trata de un chasis diseñado para hexacópteros, el cual utiliza como materiales para la estructura central fibra de vidrio y plástico para sus brazos.

Otro detalle integrado en el chasis es la inclusión de una placa de circuito impreso (PCB) que nos facilitará las conexiones electrónicas de los componentes.



Ilustración 13 - Chasis F550

4.9 Ensamblaje

Una vez descritos todos los componentes que forman la aeronave, se procede al ensamblaje de la aeronave paso a paso:

1. El primer paso debe ser el montaje del chasis [11], ya que así se dispondrá de la base donde ensamblar todos los demás componentes. Para ello basta con atornillar los tornillos que vienen con el chasis en los agujeros disponibles.
2. Montaje de los motores: se debe atornillar 4 tornillos en los extremos de los brazos que se encargará de fijar los motores. Hay que prestar especial atención a la tuerca superior del motor porque se encuentra en dos colores (negro y gris). Estos colores describen la rotación que deben seguir los motores, ya que estos son capaces de girar tanto en el sentido horario (CW) como en sentido anti-horario (CCW). El color negro es CW, y el gris CCW.
3. Soldar la conexión DC de los variadores a la placa PCB integrada en el chasis. Una vez realizada la conexión anterior se deberán soldar 3 bananas hembra a la conexión AC para así poder conectar los motores a los variadores. En la *Ilustración 14* podemos observar el diagrama de conexión de uno de los brazos. Habrá que realizar este procedimiento para cada uno de los brazos.

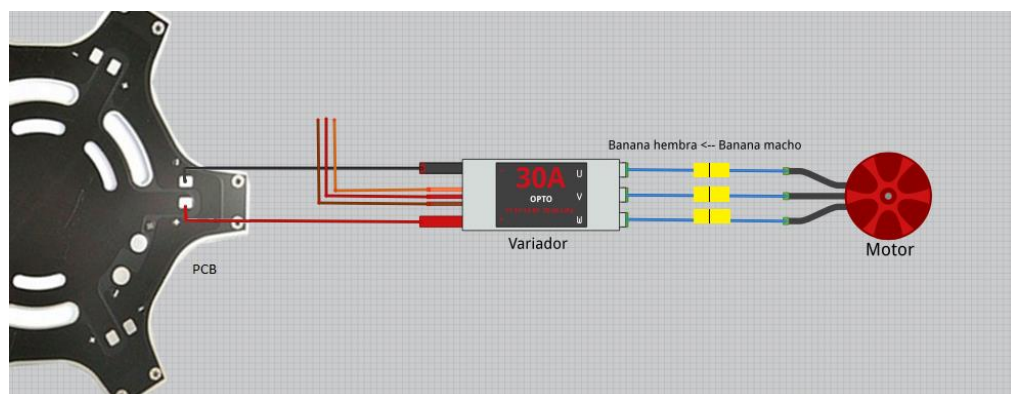


Ilustración 14 - Conexión entre PCB, variador y motor

4. Situar la controladora de vuelo en el centro exacto de la placa PCB; es necesaria esta posición para que los acelerómetros y giroscopios que la integran funcionen correctamente.
5. Conectar cada uno de los cables (compuestos por otros 3 cables) de los variadores al puerto “Main Out” de la PIXHAWK.
6. Conectar a la PIXHAWK el receptor radiocontrol (conector SBUS) al puerto “RC IN” de la PIXHAWK.

7. Conectar la antena GPS a los puertos “12C” y “GPS” de la PIXHAWK. El puerto “12C” es el encargado de alimentar la antena y el puerto “GPS” trasfiere los datos de posicionamiento a la controladora de vuelo.

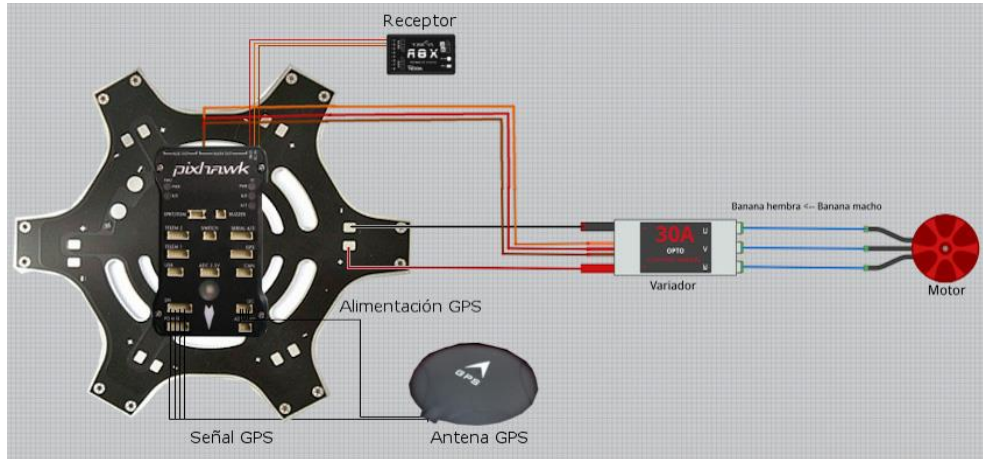


Ilustración 15 - Conexión receptor, antena GPS y conexión de un motor

8. Conectar el cable de alimentación específico de la PIXHAWK al puerto “Power” y soldar el conector a la PCB, la cual será la encargada de ofrecer energía a la PIXHAWK. Al final de este procedimiento quedara una conexión libre en dicho cable. Esta conexión será utilizada para conectar la batería.

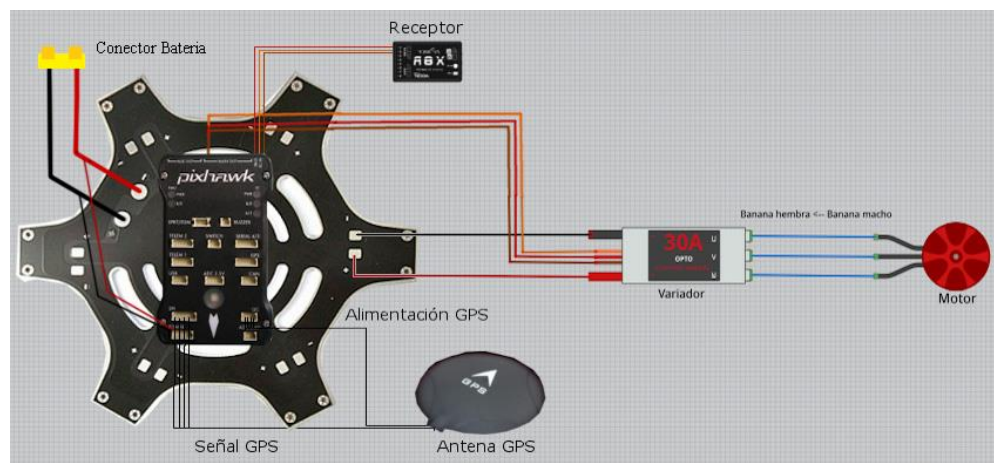


Ilustración 16 - Conexión a batería y PIXHAWK

9. Hasta este paso sería el ensamblaje de un dron convencional, pero para poder realizar las funciones deseadas debemos interconectar la PIXHAWK con la Raspberry Pi. En la *Ilustración 17* se puede observar dicha conexión, se obviaron el resto de conexiones que existen para simplificar el diagrama.

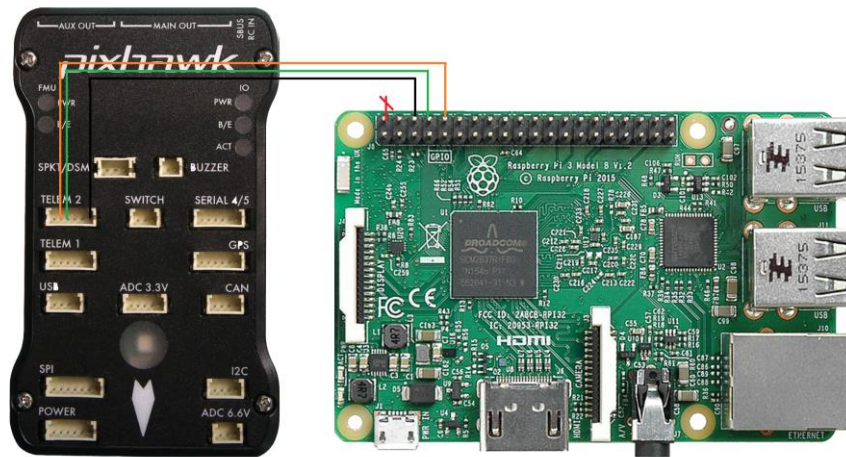


Ilustración 17 - Conexión de datos entre Raspberry Pi y PIXHAWK

El cable rojo queda sin conexión. Esta conexión se utiliza para alimentar la Raspberry Pi a través de la PIXHAWK pero en esta investigación no utilizaremos dicha conexión ya que el modelo Pi 3 consume más energía de la que puede suministrar la PIXHAWK a través del puerto “TELEM 2”, por ello se alimentará de forma externa.

10. En el paso anterior se describía la necesidad de crear un cable que alimente de forma externa la Raspberry Pi. Este cable se muestra en la *Ilustración 18* y como se observa se bifurca en dos cables, uno que se dirige a la Raspberry Pi, otro que se dirige a la antena WiFi, y por último un BEC.

El cable dirigido a la antena WiFi es necesario porque los puertos USB de la Raspberry Pi no ofrecen la cantidad de energía necesaria para que la antena funcione con normalidad. El BEC se utiliza como limitador de Voltaje, porque como se vio con anterioridad, la batería ofrece un voltaje de 14,8V y tanto la Raspberry Pi como la antena WiFi necesitan 5V.

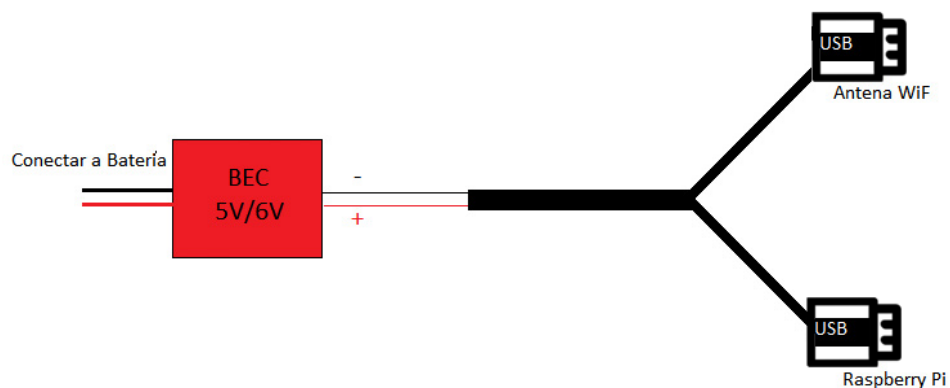


Ilustración 18 - Esquema de conexión para la alimentación de Raspberry Pi y antena WiFi

4.10 Impresión 3D

Por otra parte será necesaria la creación de piezas inexistentes en el mercado. Por ello, se utilizarán diseños creados por impresión 3D en el laboratorio ya que nos da la posibilidad de adaptar una Raspberry Pi al chasis de un dron. La impresora encargada de llevar a cabo dicha impresión es una *Prusa i3 MK2*.

Son necesarias tres piezas:

- Carcasa para proteger la Raspberry Pi.
- Tapadera de la carcasa.
- Patas de soporte de la carcasa.

Para poder realizar la impresión de dichas piezas, será necesario el uso de diversas herramientas para diseño y edición 3D. Partimos de la disposición de archivos en formato *.sldprt*. Este tipo de fichero es la extensión de un archivo de imagen. Se usa en el software *SolidWorks CAD* para la creación de objetos 3D. Este tipo de fichero no puede ser procesado por las impresoras 3D, razón por la cual es necesario utilizar otro software llamado *eDrawing* que se encarga de realizar la conversión de *.sldprt* a *.slt*.

Una vez obtenido el fichero *.slt* ya es posible el uso de un amplio abanico de programas para la edición de las piezas. Para poder realizar una impresión correcta, se posicionan las piezas sobre la parte que más contacto hará con la base de la impresora. Para realizar dicho cambio se usa *3D Builder* el cual viene por defecto en Windows 10 o superior.

Por último se utiliza el programa *CURA*, que se encarga de transformar los archivos *.slt* al formato específico que pueden interpretar las impresoras, el *.gcode*.

A continuación se observan las tres piezas utilizadas:

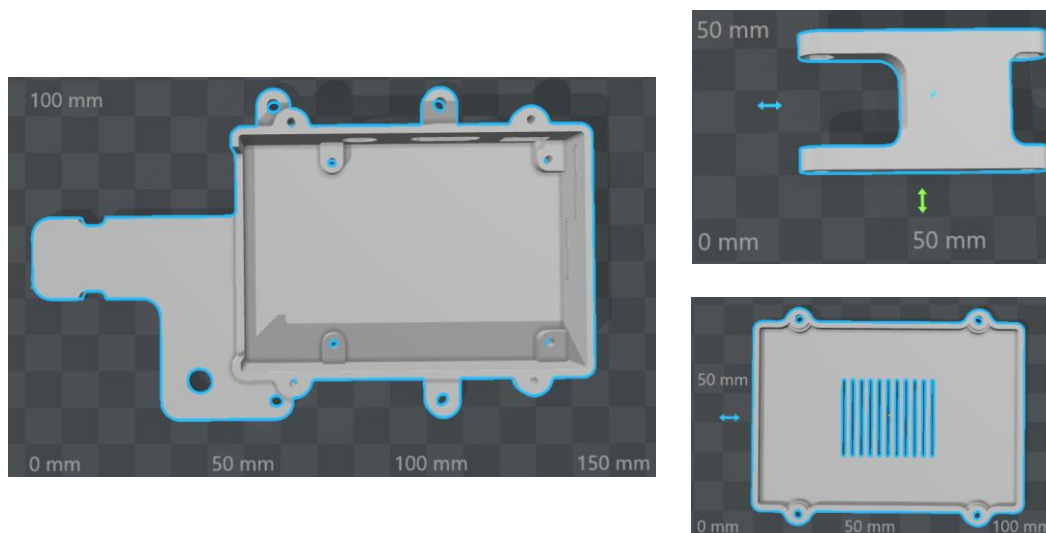


Ilustración 19 - Piezas para impresión 3D

4.11 Configuración PIXHAWK vía Mission Planner

Una vez se dispone del dron ensamblado se procede a configurar la controladora de vuelo, y para ello se utilizará el programa “Mission Planner”. Este paso es necesario ya que esta misma placa se puede usar en muchas aeronaves distintas, por ello esta configuración se encargará de personalizar el funcionamiento de la aeronave.

Mission planner es una herramienta que proporciona una manera simple y gráfica de configurar aeronaves basadas en el protocolo MAVLINK, ya que sin ella la configuración debería ser mediante línea de comandos.

Otra función que proporciona es la visualización de datos, porque obtiene toda la información que proviene de los sensores albergados en la controladora de vuelo.

El primer paso en Mission Planner es seleccionar “Initial Setup”. Comprobar previamente que se ha pulsado el botón “Connect”, situado en la esquina superior derecha, que es el encargado de establecer la conexión entre la controladora de vuelo y el ordenador.



Ilustración 20 - Mission Planner, “Initial Setup”

A continuación, se selecciona la opción “Wizard”. Esta opción se encarga de realizar una ruta entre las configuraciones mínimas necesarias que se deben realizar para poder disponer de una aeronave funcional.

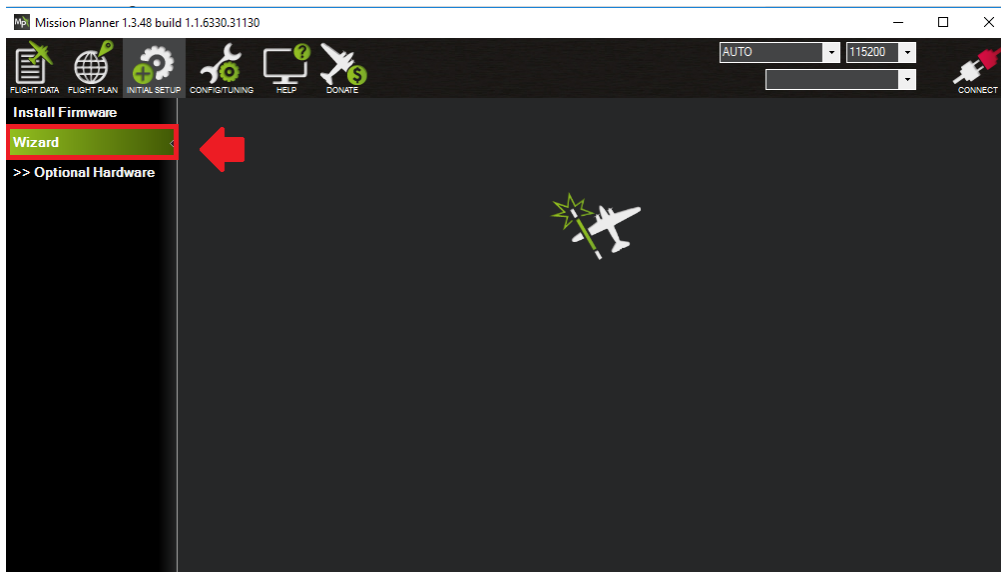


Ilustración 21 - Mission Planner, opción "Wizard"

Una vez ya dentro de "Wizard" el primer dato solicitado es el tipo de vehículo que se va a configurar, y se selecciona "Multirrotor".

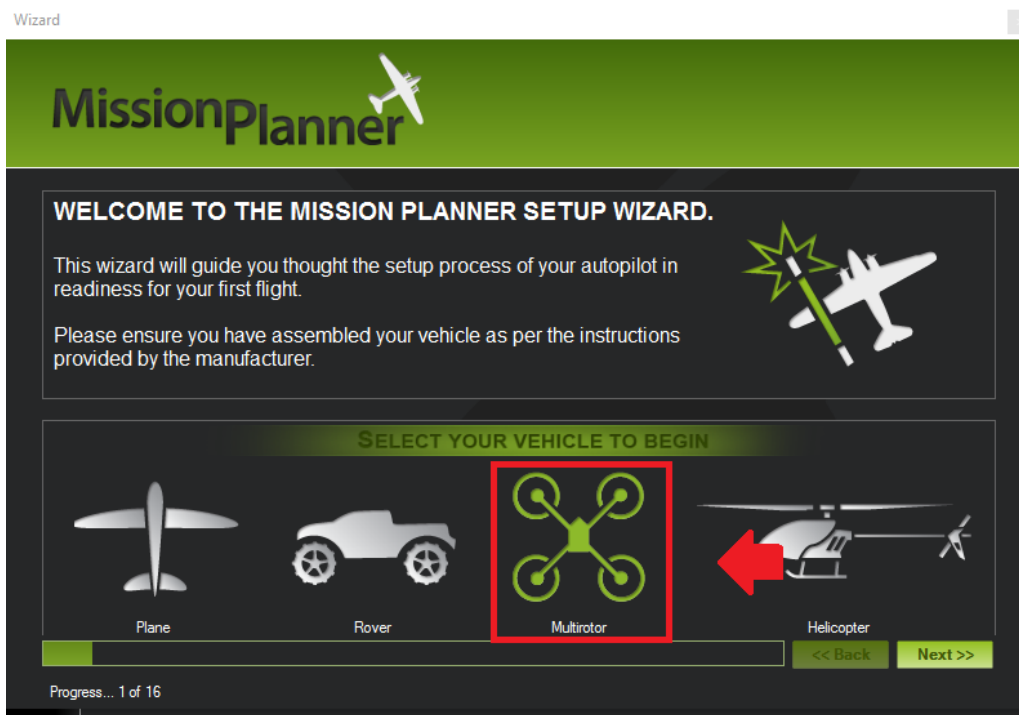


Ilustración 22 - Mission Planner, tipo de aeronave

Entre los drones existen distintos modelos, por ello en esta página de la configuración se selecciona la imagen que corresponde a un hexacóptero.

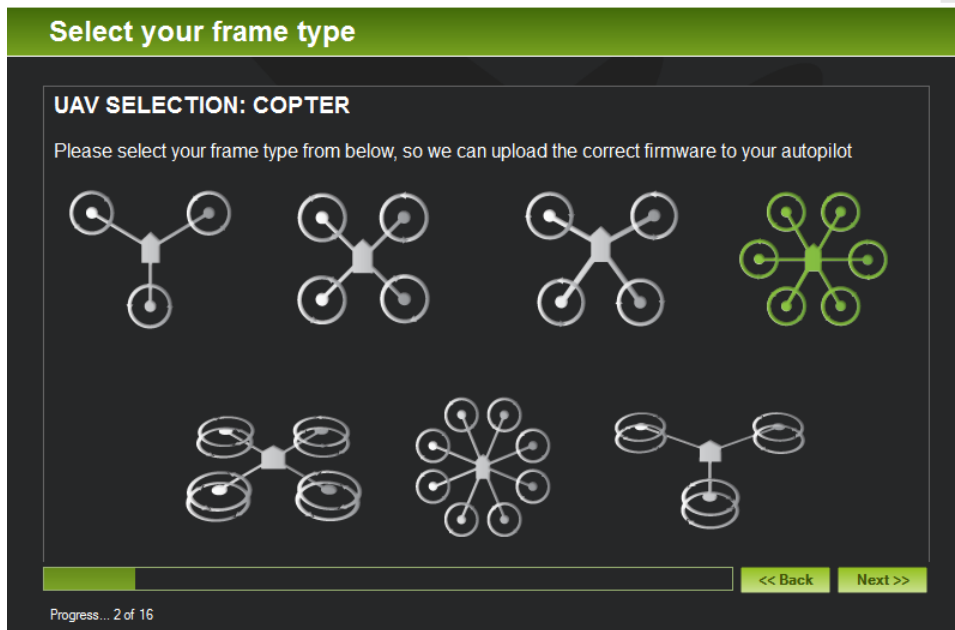


Ilustración 23 - Mission Planner, tipo de chasis

En la siguiente fase se configura el acelerómetro de la controladora, y para ello se posiciona el dron según nos indica el configurador. La precisión de este procedimiento es vital para un vuelo estable.

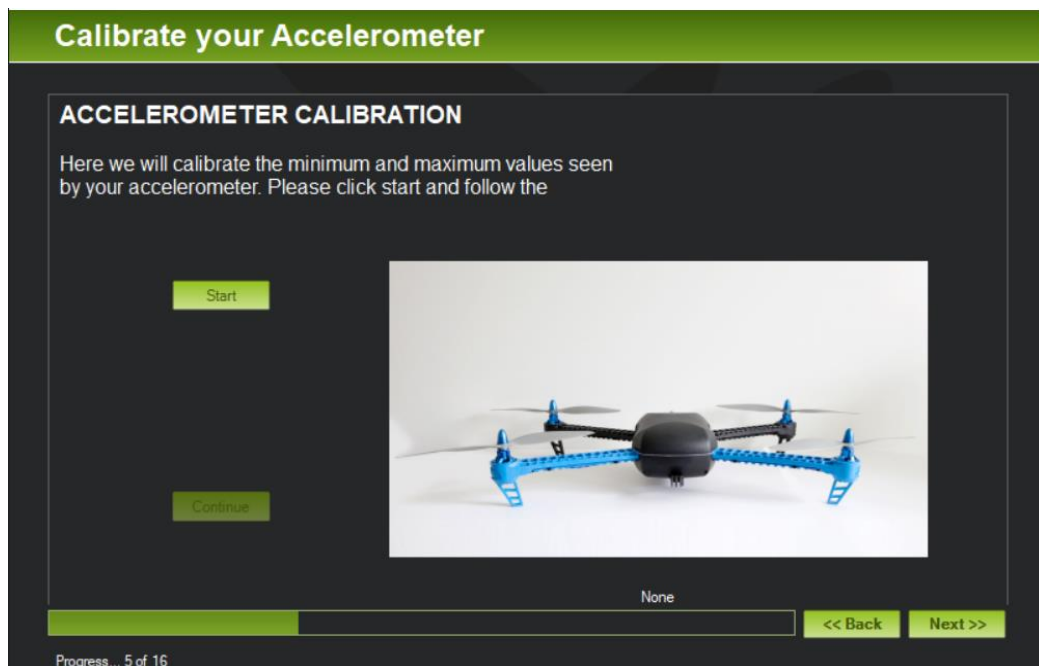


Ilustración 24 - Mision Planner, calibrar acelerómetro

El siguiente paso es la calibración del GPS, con ella conseguimos habilitar modos de vuelo no disponibles sin la antena GPS. Son de utilidad para el dron en vuelos precisos y cuando no se desea un comportamiento agresivo de la aeronave.



Ilustración 25 - Mision Planner, calibrado GPS

Para configurar la antena GPS se realizan giros completos en todos los sentidos, ya que de esta manera se obtienen los “offsets” del GPS y se almacenan en la controladora. Una vez se han obtenido los datos suficientes, el programa da por válida la configuración.

La configuración de la emisora es necesaria para que la aeronave conozca los límites de movimiento que tiene la misma, y para ello en el siguiente paso del configurador se mueven todos los controles hasta sus posiciones extremas. Así se consigue que la controladora de vuelo reconozca que valor numérico alcanzan nuestros controles, porque dependiendo del fabricante se obtienen unos valores u otros.



Ilustración 26 - Mision Planner, calibrado emisora radiocontrol

Los modos de vuelo se configuran según las necesidades de la investigación, por este motivo se configurarán los métodos más estables de vuelo. Estos son:

- **STABILIZE:** Controla la inclinación de la aeronave para evitar el vuelco.
- **LOITER:** Mantiene tanto altura como posición si el piloto no utiliza la emisora.
- **AUTO:** Programa vuelos desde la estación de tierra (Mision Planner).
- **RTL:** Vuelve a la posición de despegue sin la necesidad del control humano.

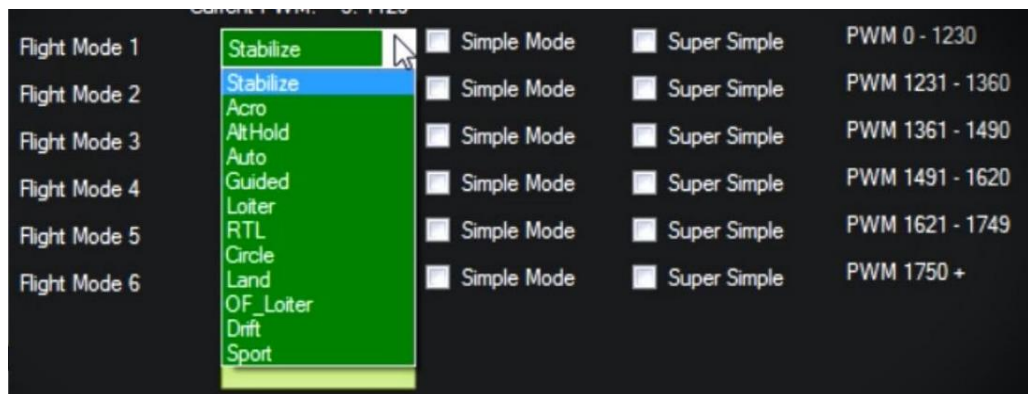


Ilustración 27 - Mission Planner, modos de vuelo

Por seguridad la última configuración realizada es “Geo Fence”, la cual consiste en un sistema encargado de velar por la seguridad de vuelo porque así se evita que se pueda volar a unas alturas o distancias no adecuadas. Si se incumplen los parámetros establecidos la aeronave automáticamente aterrizará en el lugar de despegue.

Se establece un límite de altura y de distancia suficiente para poder realizar la investigación de forma segura.

- **Altura:** 50m
- **Distancia:** 300m
- **Altitud de retorno:** 17m

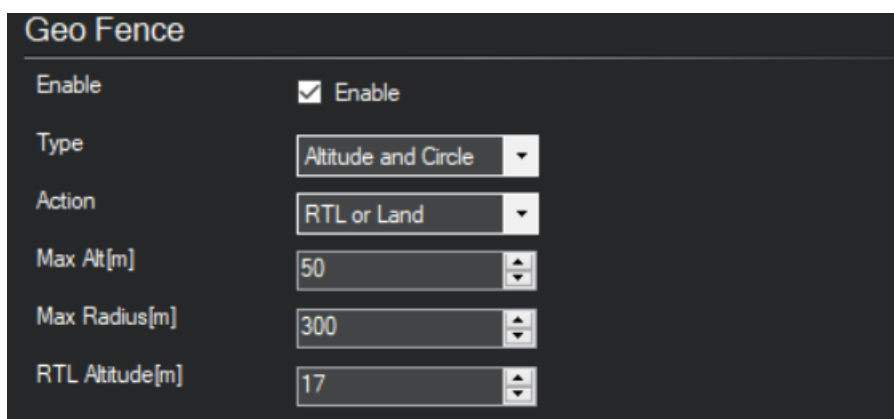


Ilustración 28 - Mission Planner, Configuración “Geo fence”

5. Software Raspberry Pi 3

Para poder usar el dron desde el terminal Android se programa un punto de acceso en la Raspberry Pi que permite la conexión segura entre la aplicación móvil y la aeronave. Para realizar dicha comunicación se necesita un intérprete a nivel de aplicación que sea capaz de obtener los datos provenientes del dispositivo móvil y los encamine hacia el controlador de vuelo. A continuación, se describen las etapas necesarias para poner en funcionamiento esta funcionalidad.

5.1 Sistema operativo

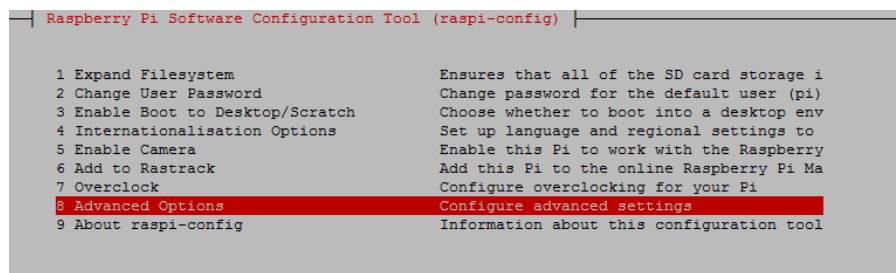
Para poder utilizar la Raspberry Pi el primer paso es la instalación del sistema operativo. Raspberry pi por defecto no contiene ningún sistema operativo instalado. Existe una gran variedad de sistemas para estas computadoras, pero se opta por *Raspbian*, que es la distribución más estable.

Raspbian es un sistema operativo libre basado en Debian optimizado para el hardware de la Raspberry Pi. Sin embargo, *Raspbian* ofrece más que un sistema operativo puro: viene con más de 35.000 paquetes software precompilados. La compilación inicial de más de 35.000 paquetes *Raspbian*, optimizada para mejorar el rendimiento en la Raspberry Pi, se lanzó en junio de 2012. Sin embargo, *Raspbian* aún está en desarrollo mejorando la estabilidad y el rendimiento del mayor número de paquetes *Debian* [12].

5.1.1 Desactivar la salida de consola por puerto serie

Por defecto *Raspbian* tiene habilitada la salida de consola del S.O por el puerto serie. Se desactiva esta salida porque, de lo contrario, se estaría enviando datos innecesarios continuamente por dicho puerto, y se pretende utilizar como enlace entre la Raspberry Pi y la PIXHAWK. Para ello basta con entrar a la configuración de la Raspberry Pi:

Abrir un terminal y teclear: `sudo raspi-config`. Tras ejecutar el comando se abre un menú con interfaz gráfica. En el menú, seleccionar: **8 Advanced Options** > **A7 Serial** y pulsar **No**.



```
Raspberry Pi Software Configuration Tool (raspi-config)
1 Expand Filesystem          Ensures that all of the SD card storage i
2 Change User Password      Change password for the default user (pi)
3 Enable Boot to Desktop/Scratch  Choose whether to boot into a desktop env
4 Internationalisation Options  Set up language and regional settings to
5 Enable Camera              Enable this Pi to work with the Raspberri
6 Add to Rastrack            Add this Pi to the online Raspberry Pi Ma
7 Overclock                  Configure overclocking for your Pi
8 Advanced Options           Configure advanced settings
9 About raspi-config         Information about this configuration tool

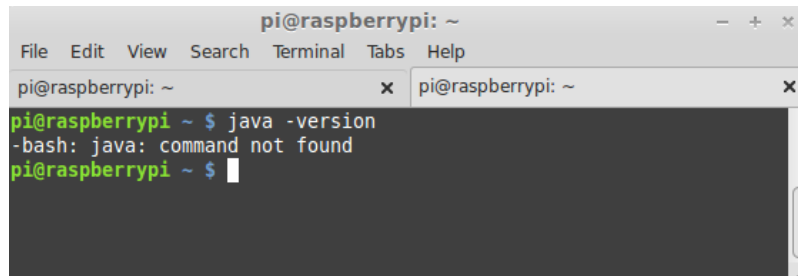
(Select) (Finish)
```

Ilustración 29 - Menú raspi-config

5.1.2 Configuración JAVA

Para comprobar la versión de JAVA actual, se abre una terminal y se teclea:

```
~$ java -version
```



```
pi@raspberrypi: ~
File Edit View Search Terminal Tabs Help
pi@raspberrypi: ~ x pi@raspberrypi: ~ x
pi@raspberrypi ~ $ java -version
-bash: java: command not found
pi@raspberrypi ~ $
```

Ilustración 30 - Raspberry Pi JAVA inicial

Se puede observar que en un principio no se dispone de ninguna versión JAVA preinstalada. Para obtener JAVA hay que descargar la versión correspondiente a Linux ARM, para ello hay que dirigirse a la página oficial de ORACLE y descargar la versión mencionada.

Una vez descargado el paquete, se abre un terminal en la carpeta */Descargas* y se teclea:

```
~$ mv jdk-8-ea-b36e-linux-arm-hflt.tar.gz /usr/local
```

Con este comando se mueve el archivo descargado a su directorio correspondiente. A continuación, hay que dirigirse al directorio donde está el archivo comprimido:

```
~$ cd /usr/local
```

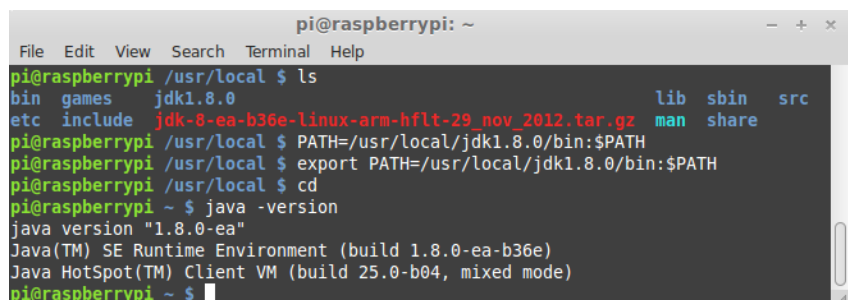
Tras esto, se descomprime y se ve el listado de directorios que se crearán bajo la carpeta **jdk1.8.0**:

```
~$ tar -xzf jdk-8-ea-b36e-linux-arm.tar.gz
```

Finalmente, se añade JAVA al PATH, para que sea ejecutable desde cualquier directorio:

```
~$ export PATH=/usr/local/jdk1.8.0/bin:$PATH
```

Tras realizar los pasos anteriores ya se dispone de JAVA, bastará con teclear de nuevo el comando **java -version** para asegurarse que dispone de una versión JAVA instalada [13].



```
pi@raspberrypi: ~
File Edit View Search Terminal Help
pi@raspberrypi /usr/local $ ls
bin  games  jdk1.8.0  lib  sbin  src
etc  include jdk-8-ea-b36e-linux-arm-hflt-29_nov_2012.tar.gz  man  share
pi@raspberrypi /usr/local $ PATH=/usr/local/jdk1.8.0/bin:$PATH
pi@raspberrypi /usr/local $ export PATH=/usr/local/jdk1.8.0/bin:$PATH
pi@raspberrypi /usr/local $ cd
pi@raspberrypi ~ $ java -version
java version "1.8.0-ea"
Java(TM) SE Runtime Environment (build 1.8.0-ea-b36e)
Java HotSpot(TM) Client VM (build 25.0-b04, mixed mode)
pi@raspberrypi ~ $
```

Ilustración 31 - JAVA instalado en Raspberry Pi



5.1.3 Configuración Puerto Serie

El puerto serie se activa introduciendo `enable_uart=1` al final del archivo `/boot/config.txt`. Los pasos a seguir son:

- Abrir terminal y teclear: `~$ sudo nano /boot/config.txt`
- Añadir al final del archivo: `enable_uart=1`

Esto sería suficiente en toda la gama de Raspberry Pi salvo en este caso ya que se utiliza una RPi 3 que tiene una conexión Bluetooth disponible que ninguna otra de la familia tiene. Por ello habrá que hacer un swap entre los *serial port*. El *serial port* por defecto, el cual puede controlarse desde los pines GPIO, está siendo utilizado por el Bluetooth, por ello se cambia esta conexión entre el *serial port* `/dev/ttyAMA0` y `/dev/ttyS0`. El puerto `/dev/ttyAMA0` será el que, tras el cambio, podremos utilizar para la conexión serie.

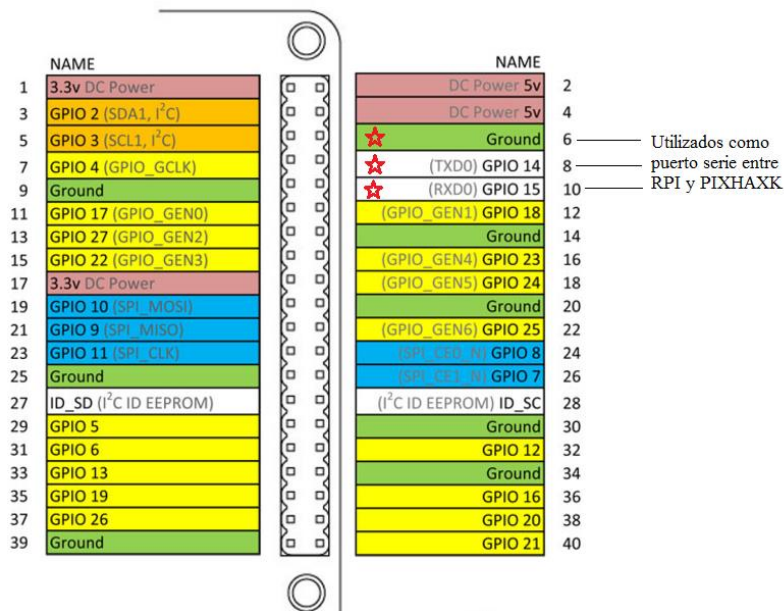


Ilustración 32 - GPIO Raspberry pi 3

Este cambio afecta a la velocidad y la calidad de señal Bluetooth, pero en este proyecto no se va a utilizar, por ello no afectará en nada el cambio. Para realizar el cambio se siguen los siguientes pasos:

- Abrir terminal y teclear: `~$ sudo nano /boot/config.txt`
- Añadir al final del archivo: `dtoverlay=pi3-miniuart-bt`

Para comprobar que los pasos anteriores se han realizado correctamente se introduce el siguiente comando por consola:

```
~$ ls -l /dev
```

Si todo ha ido bien debe aparecer como en la Ilustración siguiente, donde *ttyAMA0* está siendo utilizado por *serial0* y *ttyS0* por el *serial 1*:

```
crw-rw-r-- 1 root root    10,  58 May 27 22:04 rfkill
lrwxrwxrwx 1 root root     7 May 27 22:04 serial0 -> ttyAMA0
lrwxrwxrwx 1 root root     5 May 27 22:04 serial1 -> ttyS0
drwxrwxrwt 2 root root    40 May 27 22:04 shm
```

Ilustración 33 - Serial Port Raspberry Pi swapped

5.2 Configuración Puerto Serie PIXHAWK

Una vez configurada la Raspberry Pi, para poder transmitir datos a través de su puerto serie, se realiza el mismo paso en la PIXHAWK, pero en este caso dicha configuración es más rápida y sencilla.

En la PIXHAWK se habilita el segundo puerto de telemetría, creando un archivo vacío llamado *uartD.en* en el directorio *APM* de la tarjeta SD. Este archivo transforma el conector DF13 (FMU USART2) de 5 pines a un segundo puerto de telemetría capaz de utilizarse para la conexión Raspberry Pi → PIXHAWK.

5.3 Comunicación Raspberry Pi vía MAVLink

El primer paso para realizar la conexión vía puerto serie entre la Raspberry Pi y la PIXHAWK es la instalación de las librerías y los programas necesarios. Las librerías necesarias son:

- *screen python-wxgtk2.8 python-matplotlib python-opencv python-pip python-numpy python-dev libxml2-dev libxslt-dev*

Y los programas necesarios son:

- *pymavlink*
- *mavproxy*

Para instalar se introducen los siguientes comandos por consola:

```
~$ sudo apt-get update    #Actualiza la lista de paquetes
~$ sudo apt-get install screen python-wxgtk2.8 python-
matplotlib
python-opencv python-pip python-numpy python-dev libxml2-dev
libxslt-dev
~$ sudo pip install pymavlink
~$ sudo pip install mavproxy
```



Para probar que las dos computadoras son capaces de comunicarse entre sí, primero hay que asegurarse que la Raspberry Pi y la PIXHAWK se alimentan. A continuación en una consola dentro de la Raspberry Pi introducir el siguiente comando:

```
~$ sudo -s
mavproxy.py --master=/dev/ttyAMA0 --baudrate 57600
--aircraft hexa
```

El comando anterior realiza una conexión utilizando el programa *mavproxy.py* a través del puerto */ttyAMA0* configurado anteriormente para que actúe como puerto serie con una tasa de baudios de 57600. Esta cifra debe de estar configurada de igual manera en la PIXHAWK, y por defecto será la misma.

Por último, el parámetro `--aircraft` establece el nombre que recibe dicha conexión. En nuestro caso “hexa”.

El siguiente paso es cambiar el valor a dos parámetros, siendo el primer parámetro **ARMING_CHECK**. Este parámetro se debe modificar para que la controladora no detecte fallos en la comunicación entre emisora y el dron. Al utilizar un dispositivo móvil como control remoto, éste no es reconocido nativamente por la controladora, la cual, detecta un fallo de comunicación. Para realizar dicho cambio y evitar este error hay que realizar la conexión con *mavproxy.py* e introducir los siguientes comandos:

```
~$ param show ARMING_CHECK
~$ param set ARMING_CHECK 0
```

Estableciendo el valor a “0” se le indica que no deseamos que realice el **ARMING_CHECK**.

El siguiente parámetro que debemos cambiar es **FS_CRASH_TEST**, ya que este parámetro activa un algoritmo que se encarga de apagar los motores del dron si se detectan ciertas condiciones. Una de esas condiciones es que el dron se mueva sin tener una conexión vía radio control estándar. Por lo tanto, si no se desactiva este control, el dron se apagará cuando intentemos volar con el control vía terminal móvil. Para apagar dicho control se introducen los siguientes comandos:

```
~$ param show FS_CRASH_TEST
~$ param set FS_CRASH_TEST 0
```

5.4 Punto de acceso mediante Raspberry Pi

Antes de nada, se actualizan los programas instalados con estos comandos:

```
~$ sudo apt-get update
```

```
~$ sudo apt-get upgrade
```

A continuación, se instalan los programas necesarios para la creación del punto de acceso, **hostapd** y **isc-dhcp-server**. *Hostapd* se encarga de crear la red WiFi, y *isc-dhcp-server* es un servidor DHCP que se utilizará para asignar IPs a los dispositivos que se conectan a la red. Para instalar éstos se introduce el siguiente comando [14]:

```
~$ sudo apt-get install hostapd isc-dhcp-server
```

5.4.1 Configuración del servidor DHCP

Para configurar el servidor DHCP se abre el fichero *dhcpd.conf* con el siguiente comando:

```
~$ sudo nano /etc/dhcp/dhcpd.conf
```

Se buscan las siguientes líneas:

```
option domain-name "example.org";  
option domain-name-servers ns1.example.org, ns2.example.org;
```

Y se comentan para que no tengan efecto en el servidor. Quedando de la siguiente forma:

```
#option domain-name "example.org";  
#option domain-name-servers ns1.example.org, ns2.example.org;
```

A continuación, se buscan las siguientes líneas:

```
# If this DHCP server is the official DHCP server for the local  
# network, the authoritative directive should be uncommented.  
#authoritative;
```

Y se descomenta la siguiente línea:

```
# If this DHCP server is the official DHCP server for the local  
# network, the authoritative directive should be uncommented.  
authoritative;
```

Al tratarse de una Intranet propia, la Raspberry Pi tiene que actuar como servidor DHCP único, por esta razón hay que activar “authoritative”.

Para finalizar con el archivo *dhcp.conf*, se crea una *subnet* con las configuraciones deseadas. Para ello hay que añadir al final del archivo las siguientes líneas:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.10 192.168.1.50;
    option broadcast-address 192.168.1.255;
    option routers 192.168.1.1;
    default-lease-time 600;
    max-lease-time 7200;
    option domain-name "local";
    option domain-name-servers 8.8.8.8, 8.8.4.4;
}
```

El siguiente archivo a editar es **isc-dhcp-server**, el cual contiene información sobre qué interfaces son utilizados por el servidor *dhcpd* para escucha. Por defecto lo hace en la **eth0**, aunque en este caso no es la que se va a utilizar, y por ello se deben realizar las siguientes configuraciones:

Primero se edita el fichero *isc-dhcp-server*, y con el siguiente comando se accede a la información del fichero:

```
~$ sudo nano /etc/default/isc-dhcp-server
```

Una vez abierto, se busca la línea **INTERFACES=""** y entre las comillas se añade "wlan0": **INTERFACES="wlan0"**.

A continuación, se configura una IP estática en wlan0. De esta manera, siempre será la misma. Para ello, desactivamos la interfaz wlan0 con el siguiente comando:

```
~$ sudo ifdown wlan0
```

Una vez desactivada la interfaz, se modifica su configuración:

```
~$ sudo nano /etc/network/interfaces
```

Después de la línea **allow-hotplug wlan0**, se añade el siguiente texto:

```
iface wlan0 inet static
address 192.168.1.1
netmask 255.255.255.0
```

Para finalizar se comentan las tres últimas líneas del documento, por lo que el fichero debe de quedar de la siguiente forma:

```
auto lo
```



```
iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0

iface wlan0 inet static
address 192.168.1.1
netmask 255.255.255.0
#iface wlan0 inet manual
#wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
#iface default inet dhcp
```

Tras finalizar la edición se activa la interfaz y se le asigna la IP correspondiente:

```
~$ sudo ifconfig wlan0 192.168.1.1
```

5.4.2 Configuración del Punto de Acceso

En este apartado se detalla como configurar el punto de acceso con todos sus parámetros (SSID, contraseña, tipo de seguridad, etc.). Para ello se crea un nuevo fichero con el siguiente comando:

```
~$ sudo nano /etc/hostapd/hostapd.conf
```

Al no existir dicho fichero se crea uno en blanco y se introduce la siguiente información:

```
interface=wlan0
country_code=ES
ssid=GRCRemoteController
hw_mode=a
channel=36
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=dronegrc
```

```
wpa_key_mgmt=WPA-PSK
#wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

La configuración es delicada, ya que si no se utiliza el código de país adecuado, encontramos muchas frecuencias limitadas por el control de frecuencias que existe en las redes WiFi. Se debe utilizar el canal 36 para trabajar a 5GHz, y el código de país “ES”, de lo contrario el punto de acceso no funcionará. Se utiliza el estándar IEEE 802.11a (*hw_mode=a*) y seguridad WPA-PSK. Con ello se obtiene una mejora en cuanto a seguridad respecto al protocolo estándar de las emisoras radio control, que carecen de seguridad alguna. Otros datos de interés son:

- **SSID** → GRRemoteController
- **Contraseña** → dronegrc

Se podría utilizar otro canal de la frecuencia 5GHz, pero se tiene en cuenta que los sistemas de transmisión de video en directo (FPV) funcionan a 5,8GHZ y el canal 36 a 5,2GHZ, por lo que así se evita otra posible interferencia.

A continuación, se debe indicar a la Raspberry Pi donde se encuentra el fichero *hostapd.conf*. Para ello, se modifica la línea **#DAEMON_CONF** del siguiente archivo:

```
~$ sudo nano /etc/default/hostapd
```

La línea tras el cambio queda de la siguiente manera:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

En el siguiente paso se modifica el archivo *sysctl.conf*. Éste archivo de configuración se utiliza para pasarle al kernel parámetros de configuración en tiempo de ejecución. Con la modificación se activa el NAT (Network Address Translation) o Forwarding de paquetes para el protocolo IP.

Después de aplicar este cambio se pueden poner reglas de iptables que redirijan los paquetes entrantes de una interfaz o cualquier otra. Se introduce la siguiente línea en dicho fichero para la activación del NAT:

```
net.ipv4.ip_forward = 1
```

A continuación, ya es posible la configuración de las iptables y se realizará un cambio para el intercambio de datos entre las interfaces eth0 y wlan0:

```
~$ sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
~$ sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state
RELATED,ESTABLISHED -j ACCEPT
~$ sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
```



Este proceso se debería realizar cada vez que se reinicia la computadora, y por ello se automatiza el proceso guardando la configuración actual

```
~$ sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

A continuación, se abre el fichero encargado de gestionar las interfaces:

```
~$ sudo nano /etc/network/interfaces
```

Por último, se añade esta línea al final del archivo:

```
up iptables-restore < /etc/iptables.ipv4.nat
```

5.4.3 MAC: interfaces wlan0 y eth0

La Raspberry Pi 3 tiene una antena interna WiFi, pero en este proyecto se va a utilizar una externa capaz de utilizar la frecuencia 5GHz. Si no se realiza ninguna modificación la computadora asigna la interfaz **wlan0 aleatoriamente** entre la antena interna y la externa.

Esta situación causaría desconexiones y problemas en la puesta a punto del punto de acceso, por ello, hay que especificar que MAC se quiere que sea la interfaz **wlan0** y cual la **wlan1**. La solución pasa por la edición del siguiente fichero:

```
~$ sudo nano /etc/udev/rules.d/70-persistent-net.rules
```

Una vez abierto, hay que introducir las siguientes líneas y eliminar cualquier otra que haga referencia a wlan0:

```
# wlan0 wifi externa
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:c0:ca:90:31:fa", KERNEL=="wlan*", NAME="wlan0"

# wlan1 wifi interna
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="b8:27:eb:74:0c:d1", KERNEL=="wlan*", NAME="wlan1"
```



5.4.4 Daemon de inicio en Raspberry Pi

Una vez comprobado que el punto de acceso funciona, se crea un **daemon** que se ejecutará cada vez que la Raspberry Pi se enciende. Un daemon es un programa que no se invoca explícitamente, sino que se encuentra latente esperando que se produzcan algunas condiciones para activarse [15]. Para crear el daemon se siguen los siguientes pasos:

```
~$ sudo service hostapd start
~$ sudo service isc-dhcp-server start
```

Es posible conocer su estado con los siguientes comandos:

```
~$ sudo service hostapd status
~$ sudo service isc-dhcp-server status
```

Finalmente, para habilitarlos, hay que hacer:

```
~$ sudo update-rc.d hostapd enable
~$ sudo update-rc.d isc-dhcp-server enable
```

6. Aplicación Android control remoto

Para crear el control remoto en un terminal móvil se elige la plataforma Android porque ofrece un entorno de desarrollo gratuito y una curva de aprendizaje menor ya que la funcionalidad de sus aplicaciones se desarrolla en JAVA, lenguaje altamente estudiado en el grado.

Antes de describir la funcionalidad implementada, se muestra la estructura de proyecto, en la *Ilustración 34*.

6.1 Android manifests

Un proyecto Android se divide en 3 grandes directorios. El primero de ellos es “*manifests*”. El directorio *manifests* contiene en su interior un archivo llamado *AndroidManifest.xml*, el cual proporciona información esencial sobre la aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la app [16].

Entre otras cosas, el archivo *manifests* hace lo siguiente:

- Nombra el paquete de Java para la aplicación. El nombre del paquete sirve como un identificador único para la aplicación.
- Describe los componentes de la aplicación, como las actividades, los servicios, los receptores de mensajes y los proveedores de contenido que la integran. También nombra las clases que implementan cada uno de los componentes y publica sus capacidades.
- Determina los procesos que alojan a los componentes de la aplicación.
- Declara los permisos que debe tener la aplicación para acceder a las partes protegidas de una API, e interactúa con otras aplicaciones. También declara los permisos que otros deben tener para interactuar con los componentes de la aplicación.

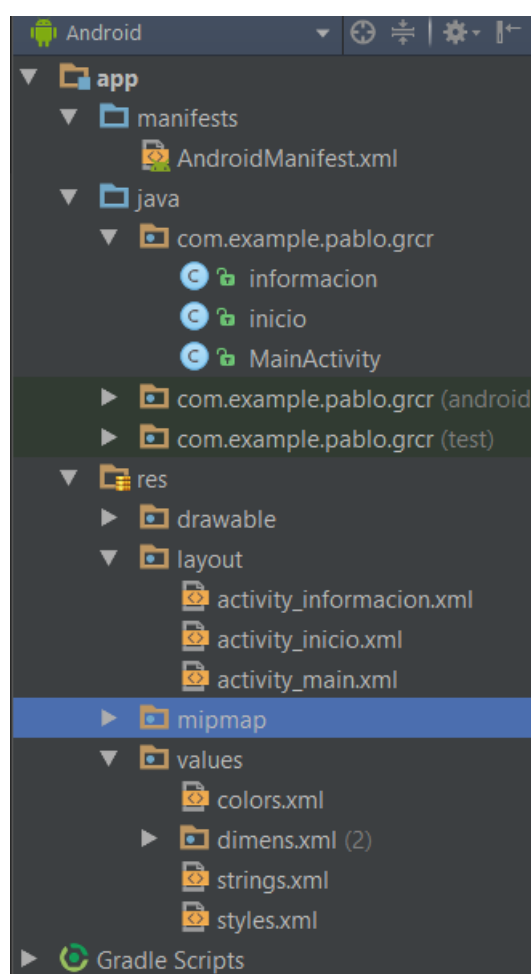


Ilustración 34 - Estructura proyecto Android

- Enumera las clases *Instrumentation* que proporcionan un perfil y otra información mientras la aplicación se ejecuta. Estas declaraciones están en el *manifests* solo mientras la aplicación se desarrolla, y se quitan antes de la publicación de esta.
- Declara el nivel mínimo de Android API que requiere la aplicación.
- Enumera las bibliotecas con las que debe estar vinculada la aplicación.

A continuación se puede observar el código *manifests* de la aplicación desarrollada:

```

    <?xml version="1.0" encoding="utf-8" ?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.pablo.grcr">
    <uses-permission
android:name="android.permission.INTERNET"/>
    <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity
            android:name=".MainActivity"
            android:screenOrientation="landscape">

        </activity>
        <activity
            android:name=".inicio"
            android:screenOrientation="landscape">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".informacion"
            android:screenOrientation="landscape">
        </activity>
    </application>

</manifest>

```

La primera información relevante que se encuentra en el fichero es los *uses-permission*, en esta aplicación se necesitan 4 de ellos:

```
<uses-permission
android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Los 4 permisos serán necesarios para poder obtener información sobre la conexión WiFi entre el terminal Android y la Raspberry Pi, para permitir la creación de sockets TCP y para el envío de paquetes a través de estos.

Dentro de los parámetros del tag *<application>* se describe el nombre de la aplicación, su icono y su estilo. Como se observa no aparecen explícitamente ya que se hace una referencia hacia el lugar donde se encuentran los recursos.

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
```

Por ejemplo los *Strings* están contenidos en un fichero llamado *string* (*android:label="@string/app_name"*) y los iconos en el directorio *mipmap* (*android:icon="@mipmap/ic_launcher"*). Más adelante se describirán con detalle estos directorios y/o ficheros.

Entre la apertura del tag *<application>* y su cierre *</application>* se encuentran las *Activity* que forman la aplicación. Una *Activity* es un componente de la aplicación que contiene una pantalla con la que los usuarios pueden interactuar para realizar una acción. Este proyecto contiene 3 *activities*: inicio, MainActivity e información.

En el archivo *AndroidManifests* se establece a cada una de ellas la propiedad:

```
android:screenOrientation="landscape"
```

Con esta propiedad obligamos al terminal Android a abrir cada una de las *activities* en modo apaisado, evitando así la desestructuración de la interfaz y la destrucción de la actividad.



Se evita dicha destrucción porque Android, cuando cambia el modo de visualización reabre la actividad. Esta acción no sería un problema en otras aplicaciones pero en este caso se establece una conexión entre una aeronave y un terminal. Si dicha conexión se cierra aunque solo sea por un breve periodo de tiempo podría causar un accidente.

6.2 Android res

Este directorio contiene los recursos de la aplicación. Se subdivide en otros 4 directorios los cuales se describen de manera detallada a continuación.

El primer directorio es **drawable**, el cual contiene las imágenes y otros elementos gráficos usados por la aplicación. Para poder definir diferentes recursos dependiendo de la resolución y densidad de la pantalla del dispositivo se suele dividir en varias subcarpetas dependiendo de la resolución y el tamaño de pantalla.

En este caso la aplicación está desarrollada para ser soportada en el siguiente tipo de dispositivos:

- mdpi (densidad media)
- hdpi (densidad alta)
- xhdpi (densidad muy alta)
- xxhdpi (densidad extra alta)

El siguiente directorio es **mipmap**, y contiene los iconos de escritorio (el icono que aparece en el *launcher* del terminal) para las distintas densidades de pantalla mencionadas anteriormente.



Ilustración 35 - Icono aplicación

El tercer directorio es **layout**, y 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.

En este caso se dispone de tres archivos *layout*, uno por cada fichero *.java*.

- Layout de *activity_inicio*:

```
<?xml version="1.0" encoding="utf-8" ?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
```

```

xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_inicio"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.example.pablo.grcr.inicio"
android:background="@drawable/fondo">

<Button
    android:id="@+id/buttonInfo"
    android:layout_width="140dp"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignStart="@+id/buttonPilotar"
    android:layout_marginBottom="26dp"
    android:backgroundTint="@color/colorButtons"

android:drawableStart="@drawable/ic_information_outline_white_18
dp"
    android:text="@string/botonInfo"
    android:textColor="#FFF" />

<Button
    android:id="@+id/buttonPilotar"
    android:layout_width="140dp"
    android:layout_height="wrap_content"
    android:layout_above="@+id/buttonInfo"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="16dp"
    android:backgroundTint="@color/colorButtons"

android:drawableStart="@drawable/ic_quadcopter_white_18dp"
    android:text="@string/botonPilotar"
    android:textColor="#FFF" />

<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/inocodrone"
    android:id="@+id/imageView2"
    android:layout_above="@+id/buttonPilotar"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="17dp" />

</RelativeLayout>

```

Esta interfaz está compuesta por dos botones y una imagen; estos botones son los encargados de llevar al usuario a las siguientes interfaces, y han sido editados para tener un estilo propio. A continuación, en la *Ilustración 36* se observa de manera gráfica este fichero.



Ilustración 36 - inicio.xml

- Layout de *activity_main*:

```

<?xml version="1.0" encoding="utf-8" ?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:custom="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.pablo.grcr.MainActivity">

    <android.support.v7.widget.Toolbar
xmlns:app="http://schemas.android.com/apk/res-auto"
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
        app:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:title="@string/textoinicio"
        app:titleTextColor="@android:color/white">

    </android.support.v7.widget.Toolbar>

    <io.github.controlwear.virtual.joystick.android.JoystickView
xmlns:custom="http://schemas.android.com/apk/res-auto"
        android:id="@+id/joystickViewIzquierdo"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_alignParentBottom="true"
  
```

```
    android:layout_alignParentStart="true"
    custom:JV_borderColor="@color/colorPrimary"
    custom:JV_borderWidth="4dp"
    custom:JV_buttonColor="@color/colorButtons" />
```

<View

```
    android:id="@+id/view"
    android:layout_width="match_parent"
    android:layout_height="3dp"
    android:layout_above="@+id/joystickView2Derecho"
    android:layout_alignParentStart="true"
    android:background="@color/colorPrimaryDark" />
```

<io.github.controlwear.virtual.joystick.android.JoystickView

```
    android:id="@+id/joystickView2Derecho"
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:layout_alignParentBottom="true"
    android:layout_alignParentEnd="true"
    custom:JV_borderColor="@color/colorPrimary"
    custom:JV_borderWidth="4dp"
    custom:JV_buttonColor="@color/colorButtons" />
```

<Button

```
    android:id="@+id/buttonARM"
    android:layout_width="114dp"
    android:layout_height="wrap_content"
    android:layout_below="@+id/view"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="69dp"
    android:backgroundTint="@color/colorButtons"
```

```
    android:drawableStart="@drawable/ic_quadcopter_white_18dp"
    android:elevation="0dp"
    android:text="@string/botonArmar"
    android:textAllCaps="false"
    android:textColor="#FFF"
    android:textSize="12sp" />
```

<RelativeLayout

```
    android:id="@+id/RelativeButtons"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_above="@+id/view"
    android:layout_alignParentStart="true"
    android:layout_below="@+id/toolbar"
    android:background="@color/colorBackground">
```

<Button

```
    android:id="@+id/buttonStabilize"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/buttonDisarm"
    android:layout_alignBottom="@+id/buttonDisarm"
    android:layout_toStartOf="@+id/buttonDisarm"
    android:backgroundTint="@color/colorPress"
    android:layout_marginRight="18dp"
    android:text="@string/botonStabilize"
    android:textColor="#FFF"
    android:textSize="12sp" />
```

<Button

```
    android:id="@+id/buttonRTL"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_marginStart="13dp"
    android:layout_marginTop="7dp"
    android:backgroundTint="@color/colorButtons"
    android:text="@string/botonRTL"
    android:textColor="#FFF"
    android:textSize="12sp" />
```

<Button

```
    android:id="@+id/buttonLoiter"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/buttonRTL"
    android:layout_alignBottom="@+id/buttonRTL"
    android:layout_marginStart="18dp"
    android:layout_toEndOf="@+id/buttonRTL"
    android:backgroundTint="@color/colorButtons"
    android:text="@string/botonLoiter"
    android:textColor="#FFF"
    android:textSize="12sp" />
```

<Button

```
    android:id="@+id/buttonDisarm"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentEnd="true"
    android:layout_alignParentTop="true"
    android:layout_marginEnd="13dp"
    android:layout_marginTop="7dp"
    android:text="@string/botonDisarm"
    android:backgroundTint="@color/colorButtons"
    android:textColor="#FFF"
    android:textSize="12sp" />
```

<TextView

```
    android:id="@+id/textoWifi"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/buttonLoiter"
    android:layout_centerHorizontal="true"
    android:text="@string/wifi"
```

```

        android:textAlignment="center"
        android:textAllCaps="true"

android:textAppearance="@android:style/TextAppearance.DeviceDefault"

        android:textColor="@color/colorWhite"
        android:textSize="10sp"
        android:textStyle="normal|bold" />

<TextView
    android:id="@+id/textoSenyal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/buttonLoiter"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="8dp"
    android:text="@string/senyal"
    android:textAlignment="center"
    android:textAllCaps="true"

android:textAppearance="@android:style/TextAppearance.DeviceDefault"

        android:textColor="@color/colorWhite"
        android:textSize="10sp"
        android:textStyle="normal|bold" />

</RelativeLayout>

</RelativeLayout>

```

Este *layout* se divide en dos partes, y se observa dicha división porque aparecen dos *RelativeLayout*. Estos se encargan de distribuir los elementos visuales de una forma determinada. En ese caso se dividirá la pantalla en una sección superior y en otra inferior.

La sección superior contiene los siguientes botones:

1. Botón RTL, encargado de enviar la orden de vuelo *Return To Home*.
2. Botón LOITER, encargado de establecer el modo de vuelo en *Loiter*.
3. Botón STABILIZE, encargado de establecer el modo de vuelo en *Stabilize*.
4. Botón DISARM, este se encarga de desactivar los motores.

Además se incluyen en la parte central dos *TextView* que se encargan de mostrar la SSID de la WiFi conectada y la calidad de señal de esta.

La sección inferior contiene los controles principales; se trata de un botón llamado *Arm Motors* y de dos *joystick*. El botón se encarga de encender los motores, por lo que, si no se presiona este botón la aeronave no responde a ninguna orden. Los *joystick* son los encargados de dirigir la aeronave. En secciones futuras se explicará con detenimiento el funcionamiento de estos *joysticks*.

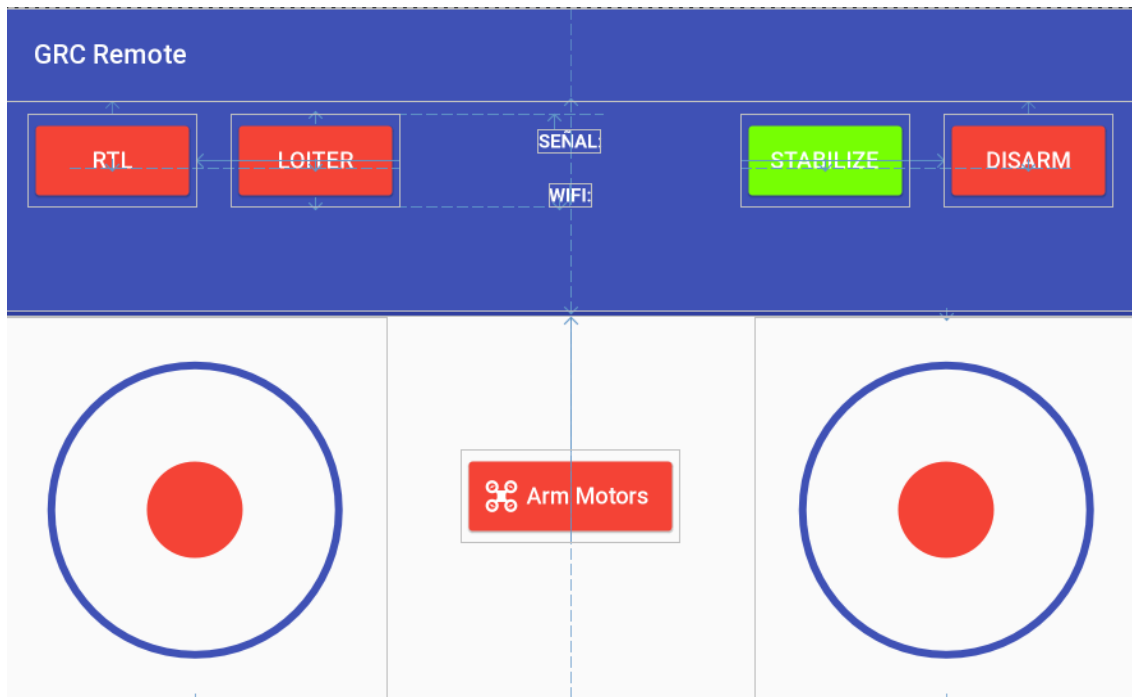


Ilustración 37 - Layout Control Remoto

- Layout *activity_informacion*:

```
<?xml version="1.0" encoding="utf-8" ?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_informacion"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="com.example.pablo.grcr.informacion">
</RelativeLayout>
```

Este *layout* es de una complejidad inferior ya que solo contiene información y opciones para contactar con el desarrollador. Como se observa en el código *.xml* carece de elementos porque estos serán creados desde la clase JAVA.

Aplicación creada por Pablo Reyes Pausà con la colaboración del grupo GRC

Versión 1.0

Conecta con nosotros

✉ Contáctenos

🐦 Síguenos en Twitter

📷 Síguenos en Instagram

Ilustración 38 - Layout Información

El último directorio que contiene la carpeta *res* es **values**, y en este directorio se almacenan tres ficheros:

1. **Colors.xml**. Este fichero contiene la definición de colores usada para nuestra aplicación. Es por este motivo que, cuando establecemos un color en un elemento, no se especifica explícitamente con el nombre del color, sino que se hace una referencia a este fichero. Por ejemplo, el color de fondo de uno de los botones de vuelo queda así:

```
android:backgroundTint="@color/colorButtons"
```

2. **Strings.xml**. Como sucede con los colores, los *strings* también son almacenados en este fichero, ya que así se facilita el cambio de texto y una posible traducción de la aplicación.
3. **Styles.xml**. Es un recurso de estilo el cual define el formato y el aspecto de una interfaz de usuario. Un estilo se puede aplicar a una vista individual (desde un archivo de diseño, una actividad, o la aplicación completa (desde el archivo de manifests).

El estilo de esta aplicación se define a continuación:

```
<resources>

  <!-- Base application theme. -->
  <style name="AppTheme"
    parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <!-- Habilitar la Toolbar -->
    <item name="android:windowFullscreen">true</item>
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
  </style>
</resources>
```



```

        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark
        </item>
        <item name="colorAccent">@color/colorAccent</item>
        <!--<item name="android:windowFullscreen">true</item>-->
    </style>
</resources>

```

6.3 Android JAVA

El directorio llamado **java** es el encargado de almacenar el código fuente de la aplicación. En este caso contendrá tres ficheros. A continuación se describe el contenido de cada uno de ellos y su finalidad.

- El primer fichero es **inicio.java**, y contiene la funcionalidad del *layout* **activity_inicio.xml**.

```

public class inicio extends AppCompatActivity {

    /**Botones**/
    Button pilotar;
    Button informacion;
    /**Evento al presionar los botones**/
    public void init(){
        pilotar = (Button) findViewById(R.id.buttonPilotar);

        pilotar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                /**Cambio de ventana actual a MainActivity**/
                Intent hacer = new Intent(inicio.this,
                MainActivity.class);
                startActivity(hacer);
            }
        });
    }
    public void inittwo(){
        informacion = (Button) findViewById(R.id.buttonInfo);

        informacion.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                /**Cambio de ventana actual a información*/
                Intent hacerdos = new Intent(inicio.this,
                informacion.class);
                startActivity(hacerdos);
            }
        });
    }

    @Override

```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_inicio);
    /**Llamda al método que se ha creado para el botón pilotar*/
    init();
    /**Llamda al método que se ha creado para el botón información*/
    inittwo();

}
}

```

Como se puede observar en el código, existen dos métodos llamados: *init* e *inittwo*, los cuales se encargan de capturar el evento *onClick()* de cada uno de los botones. Dichos métodos se inicializan en el *onCreate()*. Esto es así porque este método es el primero que se llama cuando se crea una actividad. Así se asegura que es posible lanzar el evento siempre que se ejecute la *activity*.

- El segundo fichero es **MainActivity.java**, y contiene el grueso de la aplicación. Se encarga de ofrecer funcionalidad a todos los modos de vuelo, joysticks y a la información sobre la conexión WiFi. A continuación se puede observar el contenido del fichero.

El primer paso será la declaración de las variables necesarias para el correcto funcionamiento de la lógica de aplicación. En el primer bloque de variables se declara una *buffer* para cada uno de los joysticks y otro para los botones de modo de vuelo. A continuación se declara el *DatagramPacket* que se utilizará para cada uno de los elementos mencionados.

```

/**Declaración de los sockets, arrays y paquetes de datos*/
DatagramSocket socket;
byte[] sentBufferDer;
byte[] sentBufferIzq;
byte[] sentFlightMode;
DatagramPacket sentPacketIzq;
DatagramPacket sentPacketDer;
DatagramPacket sentPacketMode;

```

El siguiente bloque de variables es necesario para poder utilizar la librería **Kryo**. Ésta es una librería para serialización en JAVA. Los objetivos del proyecto son la velocidad, la eficiencia y una API fácil de usar. Gracias a esta librería podemos obtener una velocidad de transferencia alta mediante su compresión de datos.

```

/**Variable utilizada en la serializacion con la libreria kryo*/
Output outputDer;
Output outputIzq;
Output outputMode;

```



```

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

    /**Inicialización de variables**/
    sentBufferIzq = new byte[1472]; /** 1472 es el tamaño máximo de
    datos quitando las cabeceras*/
    outputIzq = new Output(sentBufferIzq);

    sentBufferDer = new byte[1472]; /** 1472 es el tamaño máximo de
    datos quitando las cabeceras*/
    outputDer = new Output(sentBufferDer);

    sentFlightMode = new byte[1472]; /** 1472 es el tamaño máximo de
    datos quitando las cabeceras*/
    outputMode = new Output(sentFlightMode);

```

Como se comentó anteriormente, una vez conocidos los valores RC, solo queda inicializar las variables a dichos valores.

```

/**Inicialización de la variables RC de la aeronave con los valores
MIN y MAX de nuestra emisora*/
rc1min = 982;
rc1max = 2006;

rc2min = 982;
rc2max = 1997;

rc3min = 982;
rc3max = 2006;

rc4min = 982;
rc4max = 2006;

rc5min = 982;
rc5max = 2006;

```

Una vez inicializado el valor de cada una de las variables se procede a enlazar la aplicación móvil y el programa que se alojará en la Raspberry Pi. Para ello el primer paso es la inicialización de una variable del tipo **InetAddress**, la cual contendrá la dirección IP de la Raspberry Pi en la red WiFi creada por el punto de acceso.

```

InetAddress addr = null;
try {
    /**Dirección IP del servidor**/
    addr = InetAddress.getByName("192.168.1.1");
} catch (UnknownHostException e) {
    Toast.makeText(getApplicationContext(), "Servidor no
encontrado en la red WiFi", Toast.LENGTH_SHORT).show();
    e.printStackTrace();
}

/** "socket" es el punto de envío o recepción de paquetes*/
try {
    socket = new DatagramSocket();

```



```

    } catch (SocketException e) {
        Toast.makeText(getApplicationContext(), "Error
DatagramSocket", Toast.LENGTH_SHORT).show();
        e.printStackTrace();
    }

```

Tras inicializar la variable con la dirección IP de la Raspberry Pi, se inicializan los paquetes necesarios. Estos paquetes contienen la información del puerto al que van dirigidos (2510), la dirección de destino (192.168.1.1) y los datos a enviar.

```

/**Un paquete por cada tipo de control. Dirigido al puerto 2510 del
receptor*/
    sentPacketIzq = new DatagramPacket(sentBufferIzq,
sentBufferIzq.length, addr, 2510);
    sentPacketDer = new DatagramPacket(sentBufferDer,
sentBufferDer.length, addr, 2510);
    sentPacketMode = new DatagramPacket(sentFlightMode,
sentFlightMode.length, addr, 2510);

```

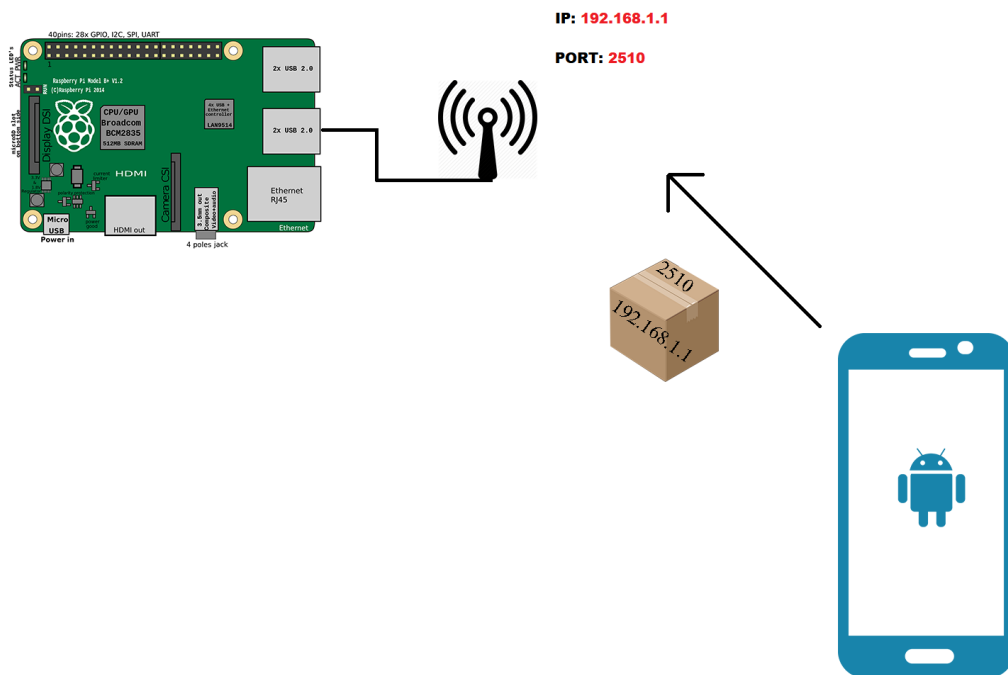


Ilustración 39 - Esquema de envío mediante paquetes

A continuación se programa el evento asociado a pulsar alguno de los botones de modo de vuelo. Para ello necesitaremos crear tres *listeners*, uno por cada modo de vuelo.

El primer paso que se debe realizar es la búsqueda del elemento gráfico que se enlazará con el código desarrollado. En el código que encontramos a continuación se puede ver cómo se enlaza cada **identificador** de elemento gráfico con las **variables botón** creadas en la clase JAVA.

```

RTL = (Button) findViewById(R.id.buttonRTL);
Stabilize = (Button) findViewById(R.id.buttonStabilize);
Loiter = (Button) findViewById(R.id.buttonLoiter);
final Context context = this;

```

Como ejemplo podemos observar el código del botón RTL, porque aunque cada uno active un modo de vuelo distinto el código es similar. Este primer fragmento del método se encarga del aspecto visual de la interfaz. Para facilitar la interacción con la aplicación, se coloreará de color verde el modo de vuelo seleccionado, y se dejará en rojo los demás.

```
/**Modo RTL (Return To Home)*/
RTL.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

RTL.setBackgroundTintList(context.getResources().getColorStateLi
st(R.color.colorPress));

Stabilize.setBackgroundTintList(context.getResources().getColorS
tateList(R.color.colorButtons));

Loiter.setBackgroundTintList(context.getResources().getColorStat
eList(R.color.colorButtons));
```

En esta parte se introduce la información que se desea enviar al servidor. La primera orden es limpiar la variable `outputMode`. A continuación se escribe un `Short`; esto es así porque, cuando los paquetes lleguen al programa alojado en la Raspberry Pi, hay que diferenciar de qué tipo se trata, ya que dependiendo del tipo recibido se procesará de una forma u otra. Por último se escribe un `String` que es el que contiene la información que es capaz de procesar MAVLINK y se obliga a la escritura de dichos valores mediante `flush()`.

```
outputMode.clear();
outputMode.writeShort(3);
outputMode.writeString("mode RTL");
outputMode.flush();
```

Para finalizar, se introduce la información en el paquete y se envía mediante un bucle `for()` tres veces consecutivas. Con ello se asegura la recepción de la información. A continuación se muestra el código de los botones restantes.

```
/**Definición de la información que contiene el
paquete*/
sentPacketMode.setData(sentFlightMode, 0,
outputMode.position());
/**Bucle para el envío de varios paquetes de modo de
vuelo. Para asegurar que alguno llegue al destino*/
for(int i = 1; i<=3; i++){
    try {
        socket.send(sentPacketMode);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
});
```

```

/**Modo STABILIZE*/
Stabilize.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

Stabilize.setBackgroundTintList(context.getResources().getColorStateList(R.color.colorPress));

RTL.setBackgroundTintList(context.getResources().getColorStateList(R.color.colorButtons));

Loiter.setBackgroundTintList(context.getResources().getColorStateList(R.color.colorButtons));

        outputMode.clear();
        outputMode.writeShort(3);
        outputMode.writeString("mode stabilize");
        outputMode.flush();
        /**Definición de la información que contiene el paquete*/
        sentPacketMode.setData(sentFlightMode, 0,
outputMode.position());
        /**Bucle para el envío de varios paquetes de modo de vuelo. Para asegurar que alguno llegue al destino*/
        for(int i = 1; i<=3; i++){
            try {
                socket.send(sentPacketMode);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
});

/**Modo LOITER*/
Loiter.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

Loiter.setBackgroundTintList(context.getResources().getColorStateList(R.color.colorPress));

RTL.setBackgroundTintList(context.getResources().getColorStateList(R.color.colorButtons));

Stabilize.setBackgroundTintList(context.getResources().getColorStateList(R.color.colorButtons));

        outputMode.clear();
        outputMode.writeShort(3);
        outputMode.writeString("mode loiter");
        outputMode.flush();
        /**Definición de la información que contiene el paquete*/
        sentPacketMode.setData(sentFlightMode, 0,

```



```

outputMode.position());
    /**Bucle para el envío de varios paquetes de modo de
    vuelo. Para asegurar que alguno llegue al destino*/
    for(int i = 1; i<=3; i++){
        try {
            socket.send(sentPacketMode);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
});

/**Botón ARM MOTORS, para que los motores se desbloqueen*/
ARM = (Button)findViewById(R.id.buttonARM);
ARM.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        outputMode.clear();
        outputMode.writeShort(3);
        outputMode.writeString("arm throttle");
        outputMode.flush();
        /**Definición de la información que contiene el paquete*/
        sentPacketMode.setData(sentFlightMode, 0,
outputMode.position());
        /**Repetición del envío*/
        for(int i = 1; i<=3; i++){
            try {
                socket.send(sentPacketMode);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
});

/**Botón DISARM MOTORS, para que los motores se bloqueen*/
DISARM = (Button)findViewById(R.id.buttonDisarm);
DISARM.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        outputMode.clear();
        outputMode.writeShort(3);
        outputMode.writeString("rc 3 900");
        outputMode.flush();
        /**Definición de la información que contiene el paquete*/
        sentPacketMode.setData(sentFlightMode, 0,
outputMode.position());
        /**Repetición del envío*/
        for(int i = 1; i<=3; i++){
            try {
                socket.send(sentPacketMode);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
});

```

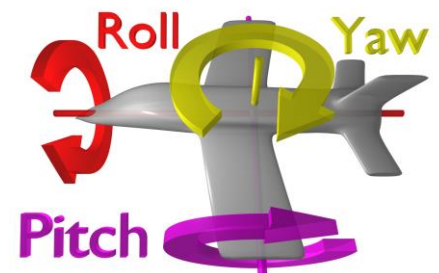

Para poder obtener información del estado de la conexión WiFi se desarrolla un método para dicho propósito. Se programa para que actualice la información cada 2 segundos, ya que así conseguimos una velocidad de refresco buena sin saturar el terminal.

Para poder realizar esta tarea se utiliza un *timer*, que es un hilo de fondo único utilizado para ejecutar todas las tareas del temporizador, de forma secuencial. Las tareas del temporizador deben completarse rápidamente. Si una tarea de temporizador toma un tiempo excesivo para completarse, "carga" el subproceso de ejecución de tareas del temporizador y puede detener la aplicación.

```
/**Metodo para obtener la información de señal WIFI cada 2 segundos**/  
  
timer = new Timer();  
timer.schedule(new TimerTask() {  
    @Override  
    public void run() {  
        MainActivity.this.runOnUiThread(new Runnable() {  
            public void run() {  
                WifiManager wifiManager = (WifiManager)  
getApplicationContext().getSystemService(Context.WIFI_SERVICE);  
                WifiInfo wifiInfo;  
                wifiInfo = wifiManager.getConnectionInfo();  
                if (wifiInfo.getSupplicantState() ==  
SupplicantState.COMPLETED) {  
                    ssid = wifiInfo.getSSID();  
                    senyal = wifiInfo.getRssi();  
                }  
                TextView textWifi = (TextView) findViewById(R.id.textoWifi);  
                TextView textSenyal = (TextView)  
findViewById(R.id.textoSenyal);  
                if(ssid != null){  
                    textWifi.setText("WiFi: "+ssid);  
                    nivelwifi = wifiManager.calculateSignalLevel(senyal, 100);  
                    textSenyal.setText("Señal: "+nivelwifi+"%");  
                }  
                else{  
                    textWifi.setText("WiFi connection: No WiFi");  
                    textSenyal.setText("Signal strength: No Signal");  
                }  
            }  
        }  
    }, 100, 2000);
```

A continuación se estudia el comportamiento de los joystick. Para entender éste se debe entender primero el comportamiento de una aeronave controlada por radio control.

Existen tres tipos de movimientos en las aeronaves, ellos son: Ángulo de deriva (**yaw**), ángulo de inclinación (**pitch**) y ángulo de alabeo (**roll**). Con el *pitch* se controla el ángulo de inclinación de la aeronave, y en los drones este movimiento permite avanzar o retroceder. El movimiento de *yaw* permite girar al dron sobre sí mismo, permitiendo elegir la dirección de movimiento (sobre el eje X). Por último el *roll* permite el movimiento hacia la izquierda o la derecha.



Una vez descritos los tres tipos de movimientos posibles, aparece otro factor que influye en el control de la aeronave: el acelerador (*throttle*). Con él se puede hacer ascender o descender el dron y si se combina con los otros movimientos se pueden realizar dichos movimientos a más o menos velocidad.

Las controladoras de vuelo y las emisoras tienen asignados estos movimientos a sus canales RC, y en este caso la asignación de canales queda como se observa a continuación.

CANAL RC	MOVIMIENTO
RC1	Roll
RC2	Pitch
RC3	Acelerador
RC4	Yaw
RC5	Auxiliar, se usa para controlar los modos de vuelo

Tabla 11 - Mapa de canales RC con movimientos

Para poder simular estos controles en el terminal Android se usarán los dos joystick que se vieron en la *Ilustración 37*. Estos joysticks han sido creados gracias a la librería *virtual-joystick-android* [17].

El funcionamiento de los joystick se basa en su posición y la distancia respecto del centro del círculo. A mayor distancia del centro mayor fuerza (*strength*) y según el ángulo en que se encuentre, se enlazarán a un canal RC u otro.

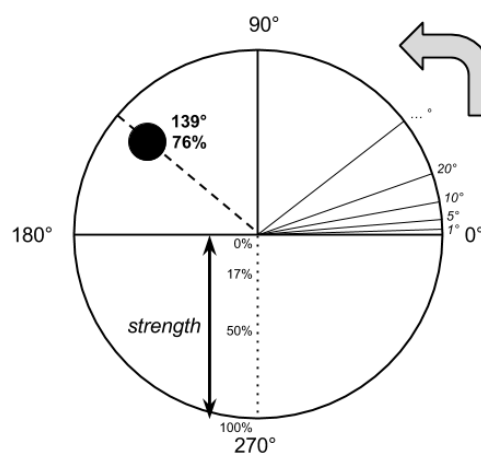


Ilustración 40 - Parámetros influyentes en los joystick

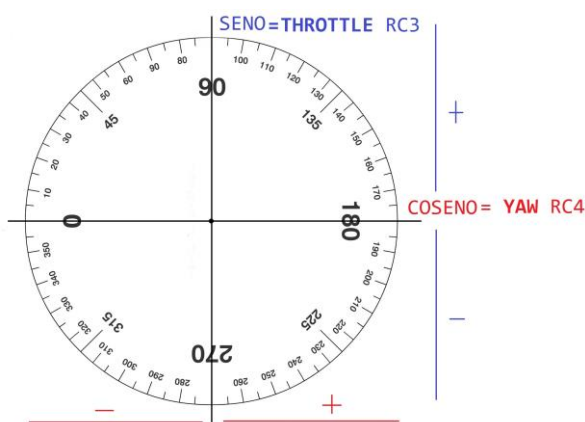


Ilustración 41 - Relación entre posición del joystick izquierdo y canales RC

El joystick izquierdo se encarga de controlar el Acelerador y el Yaw. Para poder realizar esta función se necesita obtener tanto el valor Yaw como la aceleración, respecto a la posición del joystick. Para poder obtener los valores mencionados se desarrolla la siguiente ecuación matemática.

- Canal RC3:

$$\text{Aceleración (throttle)} = \frac{rc3min + rc3max}{2} + \frac{fuerza}{100} * \left(\frac{rc3max - rc3min}{2}\right) * SIN(\text{ángulo})$$

La fuerza y el ángulo vienen dados por los joystick y la fuerza se divide entre 100 para obtener valores desde el +100 hasta el -100 con 1 punto de diferencia.

- Canal RC4:

$$\text{Yaw(deriva)} = \frac{rc4min + rc4max}{2} + \frac{fuerza}{100} * \left(\frac{rc4max - rc4min}{2}\right) * COS(\text{ángulo})$$

Para obtener el Yaw se utiliza la misma ecuación, salvo que se multiplica por el coseno, para así obtener los valores sobre el eje X. A continuación se puede observar este funcionamiento aplicado a JAVA.

```

/**Funciones del joystick izquierdo, Throttle canal 3 Yaw canal 4.
Velocidad de envio -> 1 paquete cada 100ms*/
JoystickView joystickIzq = (JoystickView)
findViewById(R.id.joystickViewIzquierdo);
joystickIzq.setOnMoveListener(new JoystickView.OnMoveListener() {
    @Override
    public void onMove(int angle, int strength) {
        // do whatever you want
        outputIzq.clear();
        anglerad = Math.toRadians(angle);
        valorThrottle = (0.5 * (rc3min + rc3max)) + ((strength * 0.01)
* (0.5 * (rc3max - rc3min)) * Math.sin(anglerad));
        valorYaw = (0.5 * (rc4min + rc4max)) + ((strength * 0.01) *
(0.5 * (rc4max - rc4min)) * Math.cos(anglerad));
        outputIzq.writeShort(1);
        outputIzq.writeInt(3);
        outputIzq.writeDouble(valorThrottle);
        outputIzq.writeInt(4);
        outputIzq.writeDouble(valorYaw);
        outputIzq.flush();
        /**Definición de la información que contiene el paquete*/
        sentPacketIzq.setData(sentBufferIzq, 0, outputIzq.position());
        try {
            socket.send(sentPacketIzq);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}, 100);
}

```

En la primera parte del código se obtiene la referencia al *joystick* presente en la interfaz, y una vez referenciado se crea el método *onMove()* que se encarga de capturar el ángulo y la



fuerza del joystick si éste se ha movido. Como ocurría en los botones de modo de vuelo se crea un *stream* con los datos necesarios y se introduce dicha información en el paquete a enviar.

Para diferenciar los paquetes provenientes de este joystick de los provenientes de los demás controles, se introduce el siguiente código:

```
outputIzq.writeShort(1);
```

Así podremos saber que los paquetes precedidos de un “1” son los que provienen del *joystick* izquierdo.

El joystick derecho tiene un funcionamiento similar, pero en este caso controlará el Pitch y el Roll. Como pasaba en el joystick izquierdo, se envía el valor del Roll y del Pitch en el mismo paquete para así poder realizar movimientos combinados de ambos. Si no se obtuvieran los movimientos de ambos canales RC la aeronave no realizaría movimientos naturales.

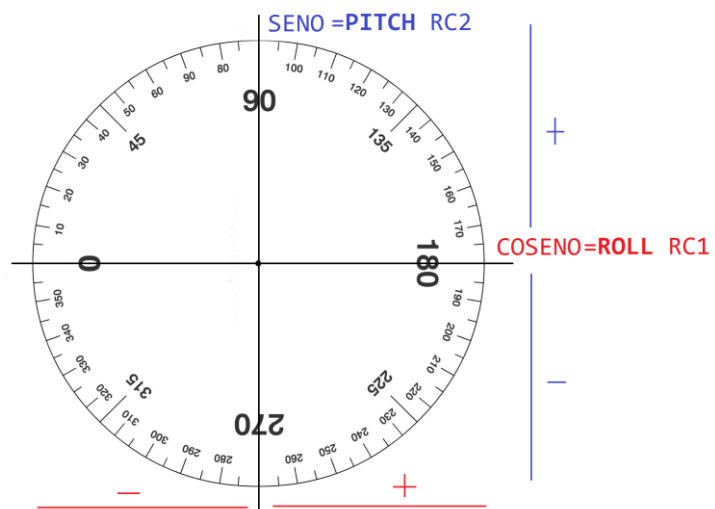


Ilustración 42 - Relación entre posición del joystick derecho y canales RC

Las ecuaciones necesarias desarrolladas para obtener dichos valores pueden ser consultadas a continuación.

- Canal **RC1**:

$$\text{Roll (alabeo)} = \frac{rc1min + rc1max}{2} + \frac{fuerza}{100} * \left(\frac{rc1max - rc1min}{2} \right) * \text{SIN}(\text{ángulo})$$

- Canal **RC2**:

$$\text{Pitch (incilinación)} = \frac{rc2min + rc2max}{2} + \frac{fuerza}{100} * \left(\frac{rc2max - rc2min}{2} \right) * \text{COS}(\text{ángulo})$$

Este joystick identificará sus paquetes con el *short* “2”.

Un comportamiento establecido en ambos *joystick* es la velocidad de envío; esta tasa se establece en 100ms para asegurarnos una buena cantidad de paquetes sin saturar el canal o el terminal Android. En el código que se presenta a continuación se ve el comportamiento completo del *joystick* derecho.



```

/**Funciones del joystick derecho movimiento de Pitch canal 2 Roll
canal 1. Velocidad de envio -> 1 paquete cada 100ms*/
JoystickView joystickDer = (JoystickView)
findViewById(R.id.joystickView2Derecho);
joystickDer.setOnMoveListener(new JoystickView.OnMoveListener() {
    @Override
    public void onMove(int angle, int strength) {
        // do whatever you want
        outputDer.clear();
        angleradtwo = Math.toRadians(angle);
        valorRoll = (0.5 * (rc1min + rc1max)) + (strength * 0.01) *
(0.5 * (rc1max - rc1min)) * Math.cos(angleradtwo);
        valorPitch = (0.5 * (rc2min + rc2max)) - (strength * 0.01) *
(0.5 * (rc2max - rc2min)) * Math.sin(angleradtwo);
        outputDer.writeShort(2);
        outputDer.writeInt(1);
        outputDer.writeDouble(valorRoll);
        outputDer.writeInt(2);
        outputDer.writeDouble(valorPitch);
        outputDer.flush();

        /**Definición de la información que contiene el paquete*/
        sentPacketDer.setData(sentBufferDer, 0, outputDer.position());
        try {
            socket.send(sentPacketDer);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
},100);

```

- El último fichero presente en el directorio JAVA es **información.java**. En él se especifica la información de contacto y datos sobre el desarrollador. Para crear esta ventana se utiliza una librería creada por el MIT que facilita el procedimiento.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_informacion);simulateDayNight(/**
DAY */ 0);

    View aboutPage = new AboutPage(this)
        .isRTL(false)
        .setDescription("Aplicación creada por Pablo Reyes Pausà
con la colaboración del grupo GRC")
        .addItem(new Element().setTitle("Versión 1.0"))
        .addGroup("Conecta con nosotros")
        .addEmail("pablo.reyes.pausa@gmail.com")
        .addTwitter("pabrep")
        .addInstagram("pabrep")
        .addItem(getCopyRightsElement())
        .create();

    setContentView(aboutPage);
}

```

7. Aplicación de enlace entre RPi y PIXHAWK

En el punto anterior se vio como la aplicación Android basaba el control remoto en el envío de órdenes. Estas órdenes venían precedidas por un entero que determinaba de qué joystick venía el paquete o si provenían de un botón.

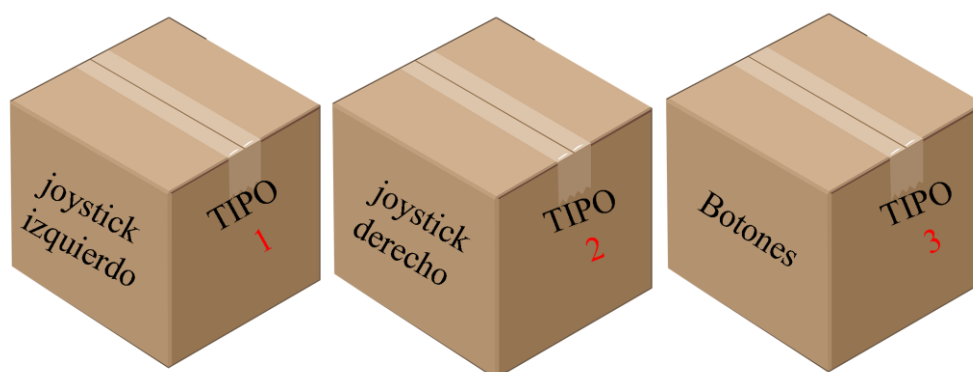


Ilustración 43 - Tipo de paquetes

En la aplicación alojada en la Raspberry Pi se interpretan los comandos recibidos y se pueden diferenciar los tres tipos existentes, para así tratarlos de una forma u otra. El proyecto encargado de realizar esta función sigue la estructura presente en la Ilustración 44.

El proyecto consta de tres archivos *.java*. El fichero *drone.java* es el principal que se encarga de sincronizar el *Listener.java* y *EscribirConsola.java*.

- **Listener.java.** Lee la información de *Mavlink* a través de *Mavproxy*.
- **EscribirConsola.java.** Escribe los comandos *Mavlink* provenientes de la aplicación Android en *mavproxy*.

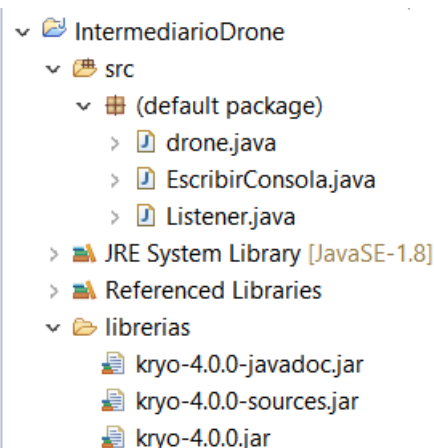


Ilustración 44 - Proyecto Java RPi

El *listener* se encarga de leer la información que proviene del programa *mavproxy*. Con esta lectura podremos consultar cualquier información proveniente de la PIXHAWK. Para obtener esta funcionalidad se utiliza el siguiente código.

```
public class Listener extends Thread {  
  
    private BufferedReader r;  
  
    public Listener(BufferedReader r) {
```

```

        this.r = r;
    }

    @Override
    public void run() {
        while(true) {
            try {
                String line = r.readLine();
                if (line != null){
                    System.out.println(line);
                }
            } catch (IOException e) {
            }
            try {
                //Para la velocidad de lectura de la consola a 0,01s
                Thread.sleep(100);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}

```

En el código se observa como mediante un *BufferedReader*, se lee la información que se muestra en el programa *mavproxy* con el método “*readLine()*”. Se añade también un “*Thread.sleep(100);*” para que no se sature el hilo haciendo lecturas, y se limita la consulta cada 100ms.

EscribirConsola tiene una complejidad mayor, y por ello se describe el código por partes para facilitar la explicación.

Para poder escribir en la consola de *mavproxy*, se crea una variable del tipo *BufferWriter*, una del tipo *DatagramSocket*, que posibilita la comunicación con la aplicación Android y un *DatagramPacket* donde almacenar la información de los paquetes provenientes de Android. La variable del tipo *Input* es la que posibilita la lectura de datos escritos por Kryo.

El resto de variables se utiliza para tratar los datos que envía el terminal.

```

private BufferedWriter w;
DatagramSocket socketReceiver;
DatagramPacket receivedPacket;
InetSocketAddress host;
Input input;
byte[] buffer;
String mode;

int rcChannel;
int rcValue;
int rcChannelSecond;
int rcValueSecond;
int rcChannelR;
int rcValueR;
int rcChannelSecondR;
int rcValueSecondR;

```



El método *EscribirConsola* abre un socket que escucha en el puerto 2510, esperando los datos enviados por el terminal Android. Una vez se tiene el socket preparado se inicializa el paquete de recepción con el tamaño del buffer, que en este caso es de 1472 bytes.

```
public EscribirConsola(BufferedWriter w) {
    this.w = w;

    try {
        socketReceiver = new DatagramSocket(2510);
    } catch (SocketException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }

    buffer = new byte[1472];
    receivedPacket = new DatagramPacket(buffer, buffer.length);
    input = new Input(buffer);
}
```

En el método *run()*, el primer paso que se realiza es leer el dato que identifica el tipo de paquete.

```
short tipo = input.readShort();
```

Una vez se obtiene el valor, se trata la información recibida de una forma u otra mediante un *switch*.

```
@Override
public void run() {
    try {
        w.write("mode stabilize");
        w.newLine();
        w.flush();
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    while(true) {
        try {
            receivedPacket = new DatagramPacket(buffer, buffer.length);
            socketReceiver.receive(receivedPacket);
            input.setPosition(0);
            short tipo = input.readShort();
            switch (tipo){
```

En el caso que el valor leído sea 1, la información recibida será tratada como el joystick izquierdo.

```
//joystick izquierdo que su tipo de paquete es el 1
case 1:
    rcChannel = input.readInt();
    rcValue = (int) input.readDouble();
    rcChannelSecond = input.readInt();
```




```

        rcValueSecond = (int) input.readDouble();

        w.write("rc "+rcChannel+" " + Integer.toString(rcValue));
        System.out.println("rc "+rcChannel+" "+rcValue);
        w.newLine();
        w.write("rc "+rcChannelSecond+" " +
Integer.toString(rcValueSecond));

        System.out.println("rc "+rcChannelSecond+"
"+rcValueSecond);

        w.newLine();
        w.flush();
break;

```

Para que el funcionamiento sea posible, se leen los datos recibidos por el socket y se escriben mediante el *BufferedWriter* (w) en el formato entendido por el protocolo *mavlink*. Por ejemplo, una orden transformada ya al formato *mavlink* quedaría de la siguiente forma:

```
Rc 3 1500
```

El joystick derecho sigue el mismo patrón para interpretar los datos, y será activado cuando el *short* recibido sea 2.

```

//joystick derecho que su tipo de paquete es el 2
case 2:
    rcChannelR = input.readInt();
    rcValueR = (int) input.readDouble();
    rcChannelSecondR = input.readInt();
    rcValueSecondR = (int) input.readDouble();

    w.write("rc "+rcChannelR+" " + Integer.toString(rcValueR));
    System.out.println("rc "+rcChannelR+" "+rcValueR);
    w.newLine();
    w.write("rc "+rcChannelSecondR+" "
+Integer.toString(rcValueSecondR));

    System.out.println("rc "+rcChannelSecondR+"
"+rcValueSecondR);

    w.newLine();
    w.flush();

break;

```

El último tipo de paquete que se puede recibir es del tipo 3. Este tipo de paquete contiene los datos de modo de vuelo y vienen definidos por un *String*. Por ejemplo: LOITER

El código encargado de procesar los paquetes se encuentra a continuación.

```

//modo de vuelo tipo de paquete 3
case 3:
    mode = input.readString();
    w.write(mode);
    System.out.println(mode);
    w.newLine();
    w.flush();
break;
}

```

```

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

Por último el fichero *drone.java* se encarga de sincronizar y unir la ejecución de las dos clases anteriores. En este primer fragmento de código, se ejecuta por consola el comando de *mavproxy*, el cual inicializa la conexión. Es muy importante que el *baudrate* sea el mismo que en la *PIXHAWK*, de lo contrario no será posible la conexión.

```

public static void main (String[] args) throws InterruptedException{

    try{

        //Para que raspberry pi tenga tiempo de cargar el sistema
        Thread.sleep(10000);

        Process process = Runtime.getRuntime().exec("sudo -s mavproxy.py
--master=/dev/ttyAMA0 --baudrate 57600 --aircraft MyCopter");
    }
}

```

A continuación se crean los dos buffers necesarios. Será necesario un buffer para la lectura de la consola *mavproxy* y otro para escribir en ella.

```

InputStream inputStream = process.getInputStream();

BufferedInputStream bufferedInputStream = new
    BufferedInputStream(inputStream);

BufferedReader r = new BufferedReader(new
    InputStreamReader(bufferedInputStream,
    StandardCharsets.UTF_8));

OutputStream outputStream = process.getOutputStream();

BufferedOutputStream bufferedOutputStream = new
    BufferedOutputStream(outputStream);

BufferedWriter w = new BufferedWriter(new
    OutputStreamWriter(bufferedOutputStream,
    StandardCharsets.UTF_8));

```

Para poder obtener información de la consola es necesario declarar una variable del tipo **InputStream**; esta obtiene el flujo de entrada del subprocesso:

```
InputStream inputStream = process.getInputStream();
```

En la escritura se utilizará otra variable del tipo **OutputStream**, la cual obtiene el flujo de salida del subprocesso. La salida se canaliza en el flujo de entrada estándar del proceso representado por este objeto de proceso.

```
OutputStream outputStream = process.getOutputStream();
```

Para ejecutar el software sin bloqueos ni pérdida de datos se crean hilos; el primer hilo que se crea es “escucha”. Una vez iniciado se detiene durante 15 segundos para que el comando de inicialización de *mavproxy* se inicie por completo.

```
//Inicio el hilo que va a leer la salida de consola

Thread escucha = new Listener(r);
escucha.start();

//Se duerme el hilo para esperar a mavproxy y su inicio

Thread.sleep(15000);
```

Por último se inicia el hilo “habla” y se lanzan los métodos *join()* para que se esperen entre sí los hilos.

```
//Inicio el hilo que va a escribir en consola
Thread habla = new EscribirConsola(w);
habla.start();
escucha.join();
habla.join();

} catch (IOException ioe) {
    System.out.println (ioe);
}
}
```

Por último se captura una **IOException** para comprobar si se ha producido un error en la entrada/salida.



8. Validación

Para validar el proyecto anterior se realizaron varias pruebas. La primera de ellas fue mediante el control remoto original. En ella se realizaron dos tipos de vuelos.

- Vuelo suave
- Vuelo acrobático

En estas pruebas se pretendía comprobar la duración de la batería y su estabilidad. La estabilidad fue buena porque junto a una configuración óptima y la utilización de seis motores para la propulsión se consiguió dicho objetivo.

El tiempo de vuelo para cada tipo de vuelo fue el siguiente:

- Suave → 24 minutos
- Acrobático → 18 minutos

Teniendo en cuenta la duración de vuelo de los drones comerciales se puede decir que este dron tiene una alta duración de vuelo. El dron más popular entre los comerciales, (DJI Phantom 4) ofrece una autonomía de unos 20 minutos, con un coste que duplica el de la aeronave creada.

La segunda prueba se realizó con el control remoto desarrollado en Android. La prueba fue satisfactoria y se podía controlar la aeronave mediante un canal seguro de la misma manera que se hace con el control estándar. Para documentar dicha prueba se realizó un video demostrativo el cual se puede consultar en el enlace que muestra a continuación.



Ilustración 45 - Prueba de vuelo 1



Ilustración 46 - Prueba de vuelo 2

https://youtu.be/4jmnE2z2_k8

9. Conclusiones

Una vez finalizado el proyecto, es posible obtener unas conclusiones sobre todo el trabajo realizado. Todo proyecto conlleva la consecución de unos objetivos marcados al inicio del mismo. En el caso del presente proyecto, se puede afirmar que se han cumplido todos los objetivos fijados.

El primer objetivo fue obtener una aeronave cuya estabilidad y duración de vuelo fuesen altas. Teniendo en cuenta la limitación existente en cuanto a las capacidades de las baterías, el tiempo de vuelo (24 minutos) se puede considerar como satisfactorio.

El segundo objetivo, la estabilidad, ha sido correcta gracias al uso de un hardware y una configuración adecuada para ello.

En cuanto al uso de tecnologías como los drones y las controladoras de vuelo basadas en Mavlink, destacar que, aunque no han sido tratadas durante el grado de Ingeniería Informática, se ha podido desarrollar software para estas tecnologías. Gracias a la utilización de éstas, se han podido ampliar conocimientos relacionados con la aviación y la programación en sistemas críticos.

El uso de una conexión segura proporciona al mundo de los drones una alternativa para el control de los mismos, asegurando una seguridad máxima entre la aeronave y el terminal de control. Este factor está muy poco cuidado en los drones comercializados y en un futuro pueden surgir problemas graves si no se abordan y se utilizan soluciones como la presentada en el presente proyecto.

Otro objetivo secundario alcanzado que cabe destacar, es la apertura de una ventana de comunicación entre terminales Android, dispositivos capaces de ejecutar Java, y aeronaves basadas en Mavlink. Gracias a esta ventana pueden realizarse infinitos proyectos que necesiten de un dron y capacidad de computación, la cual sería proporcionada por la Raspberry Pi.

Para finalizar, resaltar la dificultad añadida de estudiar un campo poco explorado, del cual se carece de información o la poca que se puede encontrar es de baja calidad y muy poco elaborada.

Trabajos futuros

Una vez alcanzados los objetivos propuestos en este proyecto se presentan las siguientes opciones para futuros trabajos.

- Ampliar la funcionalidad de la aplicación móvil para mostrar al piloto de la aeronave información de vuelos como la altura, el estado de la batería, la distancia respecto del punto de despegue, inclinación de la aeronave respecto a tierra, etc.
- Ampliar la distancia de control al máximo utilizando la tecnología 4G de las redes móviles como enlace entre el terminal Android y el dron.
- Habilitar la transmisión de video (FPV) aprovechando la velocidad de la conexión WiFi a 5GHZ en una calidad FULL HD o 4K. Este video se tendría que ver junto a los controles.
- Aprovechar los sensores del terminal Android como el acelerómetro y el giroscopio para controlar la aeronave sin necesidad de utilizar la pantalla.
- Desarrollar un algoritmo que obtenga la posición del terminal y el dron sea capaz de seguirlo a una altura y distancia determinada.
- Desarrollar una app móvil que mediante el trazado de una ruta sobre *Google Maps* sea capaz de seguir ese camino a una altura y velocidad determinada.
- Dotar a la aeronave de una cámara que, mediante el software alojado en la Raspberry Pi sea capaz de detectar personas y envíe la información de geoposicionamiento de dicha persona. El terminal Android procesará la posición y la mostrará en algún software de mapas, como puede ser *Google Maps*.



Bibliografía

- [1] «www.huelvaya.es,» 19 Mayo 2016. [En línea]. Available: <http://huelvaya.es/2016/05/19/defensa-asegura-que-buscara-recursos-para-el-proyecto-ceus/>.
- [2] N. Robotics, «www.nccr-robotics.ch,» [En línea]. Available: <http://www.livescience.com/58252-insect-inspired-drone-crashes-like-a-champ.html>.
- [3] «www.amazon.es,» [En línea]. Available: <https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011>.
- [4] D. R. (Olsson, «[ww.darpa.mil](http://www.darpa.mil),» [En línea]. Available: <http://www.darpa.mil/program/inbound-controlled-air-reasonable-unrecoverable-systems>.
- [5] «www.upv.es,» 20 Mayo 2015. [En línea]. Available: <http://www.upv.es/noticias-upv/noticia-7463-aeronaves-no-t-es.html>.
- [6] W. Wei, «www.thehackernews.com,» [En línea]. Available: <http://thehackernews.com/2016/10/how-to-hack-drone.html>.
- [7] [En línea]. Available: <http://quadsforfun.wixsite.com/quadsforfun/typical-electronic-layout-pixhawk?lightbox=c1sdv>.
- [8] MIT, «technology.mit.edu,» 14 Mayo 2012. [En línea]. Available: http://technology.mit.edu/technologies/17109_solid-state-lithium-polymer-as-electrolyte-for-lithium-ion-batteries.
- [9] B. Schneider, «[rogershobbycenter](http://rogershobbycenter.com),» 26 Mayo 2012. [En línea]. Available: <https://rogershobbycenter.com/lipoguide/>.
- [10] J. F. K. y. K. W. Ross, «WiFi: 802.11 Wireless LANs,» de *COMPUTER NETWORKING (6th edition)*, PEARSON.
- [11] E. Darack, «www.airspacemag.com,» Julio 2014. [En línea]. Available: <http://www.airspacemag.com/flight-today/build-your-own-drone-180951417/?page=1>.
- [12] «Raspbian,» 2012. [En línea]. Available: <http://www.raspbian.org/>.
- [13] B. C. y. G. Collins, «<http://www.oracle.com>,» [En línea]. Available: <http://www.oracle.com/technetwork/es/articles/java/java-se-embedded-raspberry-pi-1940405-esa.html>.



- [14] M. P. Estes, «<https://geekytheory.com>,» 2017. [En línea]. Available: <https://geekytheory.com/tutorial-rasperry-pi-como-crear-un-punto-de-acceso-wifi/>.
- [15] E. S. Raymond, «daemon,» de *The Jargon File*, 2008.
- [16] Google, «<https://developer.android.com>,» [En línea]. Available: <https://developer.android.com/guide/topics/manifest/manifest-intro.html>.
- [17] D. Brun, «GitHub,» [En línea]. Available: <https://github.com/controlwear/virtual-joystick-android>.


Anexo

Datos estimados de carga, tiempo de vuelo estimado, potencia eléctrica, temperatura exterior, empuje-peso y empuje específico con el hardware seleccionado para la construcción del dron:

General	Refrigeración del Motor: excelente	Nº de rotores: 6 simple	Peso del modelo: 1400 g 49.4 oz	Tamaño del armazon: sin batería	Limited de inclinación de la FCU: 25°	Altura del campo 100 m ASL 328 ft ASL	Temp. aire 25 °C 77 °F	Presion (QNH): 1013 hPa 29.91 inHg
Celdas batería	Tipo (Cont. / max. C) - nivel de carga: LiPo 8000mAh - 30/45C - llena	Configuración: 4 S 1 P	Capacidad por celda: 8000 mAh 8000 mAh total	descarga max. 95%	Resistencia: 0.0021 Ohm	Voltaje: 3.7 V	capacidad C de descarga: 30 C continua 45 C de pico	Peso: 201 g 7.1 oz
Variador	Tipo: max 30A	Corriente: 30 A cont. 30 A max.	Resistencia: 0.008 Ohm	Peso: 40 g 1.4 oz	Accesorios	Consumo de corriente: 0 A	Peso: 0 g 0 oz	
Motor	Fabricante - Tipo (Kv): DJI 2212-920 (920) buscando... Asistente KV hélice	KV (w/o torque): 920 rpm/V	Corriente sin hélice: 0.75 A @ 10 V	Límite (hasta 15s): 250 W	Resistencia: 0.098 Ohm	Longitud caja: 28 mm 1.1 inch	nº Polos mag.: 14	Peso: 50 g 1.8 oz
Hélice	Tipo de hélice: DJI - 0°	Diámetro: 9.4 inch 238.8 mm	Paso: 4.5 inch 114.3 mm	número de palas: 2	Const. de Potencia/Empuje: 1.10 / 1.0	Gear Ratio: 1 : 1	Calcular	

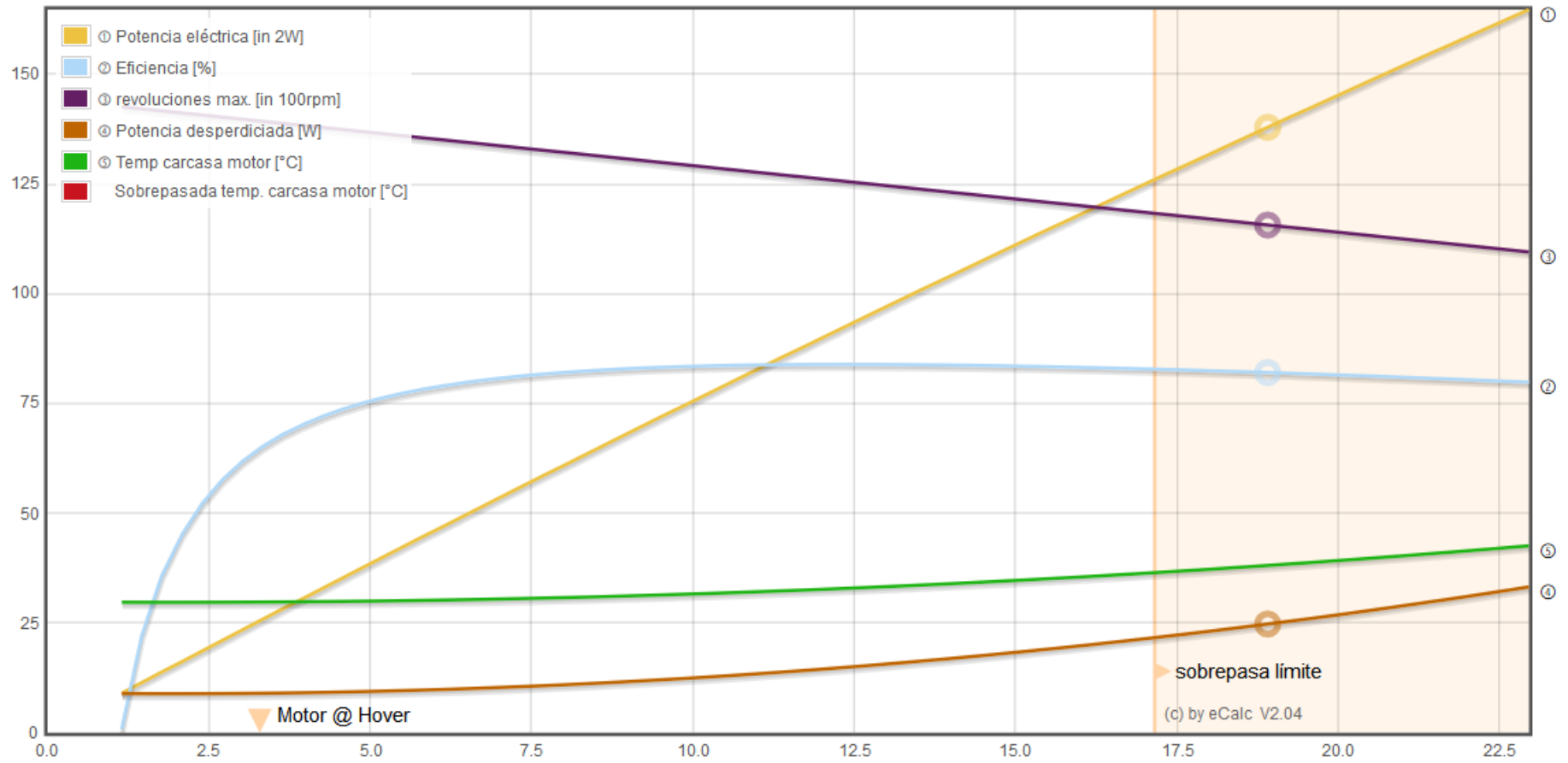
Carga:	Tiempo de vuelo estacionario:	Potencia eléctrica:	Temperatura ext.:	Empuje-Peso:	Empuje específico:	Configuración

A continuación se presentan los datos específicos del hexacóptero:

Batería	Motor a eficiencia óptima	Motor al Máximo	Motor @ Hover	Motorización Total	Multicóptero
Carga: 14.35 C	Corriente: 11.98 A	Corriente: 19.14 A	Corriente: 3.30 A	Peso de la Motorización: 1478 g	Peso total: 2204 g
Voltaje: 14.72 V	Voltaje: 14.99 V	Voltaje: 14.57 V	Voltaje: 15.49 V	52.1 oz	77.7 oz
Tensión nominal: 14.80 V	Revoluciones*: 12617 rpm	Revoluciones*: 11531 rpm	Revoluciones*: 5844 rpm	Empuje-Peso: 3.1 : 1	máximo peso adicional: 3635 g
Energía: 118.4 Wh	Potencia eléctrica: 179.5 W	Potencia eléctrica: 278.8 W	Acelerador (log): 30 %	Corriente en estacionario: 19.82 A	128.2 oz
Capacidad total: 8000 mAh	Potencia mecánica: 150.6 W	Potencia mecánica: 228.5 W	Acelerador (lineal): 46 %	Pot(entrada) en estacionario: 311.0 W	inclinación máxima: 25 °
Capacidad usada: 7600 mAh	Eficiencia: 83.9 %	Potencia-Peso: 759.0 W/kg	Potencia eléctrica: 51.2 W	Pot(salida) en estacionario: 251.1 W	velocidad máxima: 27 km/h
Tiempo min de vuelo: 4.0 min		344.3 W/lb	Potencia mecánica: 41.9 W	Eficiencia en estacionario: 80.7 %	16.8 mph
tiempo medio de vuelo: 15.4 min		Eficiencia: 81.9 %	Potencia-Peso: 141.1 W/kg	Corriente al máximo: 114.81 A	Trepada estimada : 12.0 m/s
Tiempo de vuelo estacionario: 23.0 min		Temperatura ext.: 38 °C	64 W/lb	Potencia(entrada) al máximo: 1801.2 W	2362 ft/min
Peso: 804 g		100 °F	Eficiencia: 81.8 %	Potencia(salida) al máximo: 1370.7 W	Area total del disco: 26.86 dm²
28.4 oz		Medidas de potencia	est. Temperatura: 27 °C	Eficiencia al máximo: 76.1 %	416.33 in²
		Intensidad: 114.84 A	81 °F		Fallo del motor: 
		Voltage: 14.72 V	Empuje específico: 7.18 g/W		
		Potencia: 1690.4 W	0.25 oz/W		



Características del Motor



Estimador de alcance

