Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Desarrollo de un sistema de control para *UAV* con capacidad *ATOL* entre lanchas de instrucción de la Escuela Naval Militar

Grado en Ingeniería Mecánica

ALUMNO: José María Liaño Cuquerella

DIRECTORES: José María Núñez Ortuño

Carlos Casqueiro Placer

CURSO ACADÉMICO: 2015-2016

Universidade Vigo

Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Desarrollo de un sistema de control para *UAV* con capacidad *ATOL* entre lanchas de instrucción de la Escuela Naval Militar

Grado en Ingeniería Mecánica

Intensificación en Tecnología Naval

Cuerpo General

Universidad de Vigo

RESUMEN

El objetivo de este Trabajo de Fin de Grado es construir una plataforma "*UAV*" (*Unmanned Aerial Vehicle*) que incorpore un Sistema de Control remoto con capacidad "*ATOL*" (*Automatic Take-Off and Landing*) en las lanchas de instrucción de la Escuela Naval Militar.
Para ello se ha construido un multicóptero utilizando en su mayoría hardware comercial de fácil adquisición. A este multicóptero se le ha incorporado una "*Raspberry Pi*" como servidor intermedio de mensajes de control, de forma que permita la inclusión del *UAV* en una red *WIFI*. Como red de control se ha utilizado la red "*MANET" (Movile ad hoc Network)* instalada en las lanchas de instrucción de la Escuela Naval Militar. El control se realiza desde una estación instalada en un ordenador incorporado a esta misma red. De la misma forma, la red ha permitido al Sistema de Control el acceso a la información de los sensores de las lanchas de instrucción, que se han utilizado como medio fundamental para la aproximación y el aterrizaje.

PALABRAS CLAVE

UAV, *Raspberry Pi*, Sistema de Control, *MANET*, *ATOL*, Dron

AGRADECIMIENTOS

Al profesor Don José María Núñez Ortuño, por su guía y dedicación, y al Dr. Don Carlos Casqueiro Placer por su entusiasmo y apoyo, sin los cuales este proyecto no hubiera sido posible.

A mi padre, por su consejo y predisposición en todo momento.

Al equipo del TFG de “Despliegue de una Red Móvil ad hoc” por su ayuda a lo largo del proceso de incorporación del *UAV* en la red.

A la Armada Española, la Escuela Naval Militar, y el Centro Universitario de la Defensa, por proporcionar los medios materiales y humanos necesarios, y concederme la oportunidad de trabajar en un proyecto tan apasionante como actual.

Contenido

Índice de Ilustraciones

Índice de Tablas

1 INTRODUCCIÓN

"Cuanto más se eleva un hombre, más pequeño les parece a los que no saben volar."

Friedrich Nietzsche

1.1 Introducción

Los sistemas autónomos son uno de los principales campos de desarrollo de todos los ejércitos del mundo. Estos sistemas permiten la ausencia de un controlador o piloto en la zona de operaciones. A menudo se refiere a sus labores como las 3 "*D*", del inglés "*dull, dirty and dangerous*" [1] (tedioso, sucio y peligroso), refiriéndose a aquellas labores en las que se prefiere evitar la presencia de un ser humano, ya sea por su eficacia, idoneidad o el peligro que entrañan. Además de evitar poner en riesgo a un ser humano, estos sistemas representan una serie de ventajas frente a los sistemas tripulados como:

- Ser más baratos por su menor carga, especificaciones, necesidades de mantenimiento e intervención humana.
- No están limitados por factores biológicos o psicológicos como los seres humanos.
- Mayor eficacia en el desempeño de funciones simples.
- Facilidad de diseño, sin el inconveniente de mantener un piloto con vida.

Estas son algunas de las razones que han propiciado el desarrollo de los sistemas autónomos. A pesar de ello, existen limitaciones en cuanto a las tareas que estos sistemas pueden llevar a cabo, y siempre habrá labores que necesiten intervención humana en alguna parte de su desarrollo.

Este proyecto de desarrollo busca crear un "*UAV*" *(Unmanned Aerial Vehicle)*, un vehículo aéreo no tripulado con un sistema de control que permita tanto su pilotaje de forma remota, como la automatización de tareas. A lo largo del proyecto se construirá un *multicóptero*, y se le equipará de los medios necesarios para su control a través de una red *WIFI*. Una vez listo esto, se procederá a desarrollar los programas necesarios para su control y autonomía.

1.2 Motivación

Este proyecto no busca la resolución de un problema, sino la apertura del CUD (*Centro universitario de la Defensa*) a un nuevo frente de desarrollo. La investigación sobre vehículos autónomos y sus posibilidades están en boga en diferentes instituciones tanto nacionales como internacionales, tanto públicas como privadas. Se trata de un tema de actualidad con gran cantidad de

aplicaciones. Teniendo en cuenta además el carácter militar de la institución, se trata de un proyecto de gran interés tanto para la ENM como para el CUD.

Este proyecto proporcionará los medios que harán posible el desempeño de próximos Trabajos de Fin de Grado. Estos trabajos se realizan cada año como parte fundamental de los estudios de los futuros oficiales de la Armada Española. Además supondrá un equipo de interés para la Escuela Naval Militar, por su posible incorporación en la enseñanza de procedimientos tácticos con aeronaves, o incluso como un medio de grabación avanzado para tareas cotidianas de la ENM.

Las motivaciones personales que llevaron a la elección de este proyecto frente a otras posibilidades existentes fueron lo actual del campo al que pertenece y la oportunidad de obtener un resultado tangible del proyecto como ha sido la plataforma obtenida. El esfuerzo económico realizado por parte del CUD y la ENM, el apoyo prestado, y la confianza depositada en el alumno han sido en todo momento incentivos para una máxima dedicación al trabajo, a pesar del breve periodo de tiempo asignado.

1.3 Objetivos

El objetivo fundamental de este proyecto es el desarrollo de un sistema de control para *UAV* con capacidad *ATOL* (*Automatic Take off and Landing*) entre las lanchas de instrucción de la Escuela Naval Militar. Este objetivo se ha dividido en cuatro menores que representan las diferentes etapas de desarrollo del proyecto. Estas metas intermedias son:

- Diseño y construcción de la plataforma
- Capacidad de vuelo autónomo basado en GPS
- Control del *UAV* a través de una red *WIFI*
- Desarrollo del programa de control con capacidad *ATOL*

Por otra parte, este trabajo busca ser la base sobre la que sustentes futuros proyectos de desarrollo del Centro Universitario de la Defensa. Por ello se añade como objetivo secundario la redacción de una memoria detallada y didáctica.

1.3.1 Diseño y construcción de la plataforma

Este objetivo engloba la investigación previa de las diferentes posibilidades que existen en el campo de los vehículos aéreos autónomos a fin de elegir el tipo de vehículo que mejor se ajuste a las necesidades del proyecto.

Abarca también un diseño general del aparato con el fin de evaluar los elementos necesarios para llevar a cabo el proyecto y su posterior estudio de mercado y adquisición.

Finalmente, este objetivo incluye la construcción física del *UAV* y su puesta en marcha.

1.3.2 Vehículo autónomo

Este objetivo hace referencia a la capacidad del *UAV* de llevar a cabo misiones. Deberá ser capaz de realizar vuelos pre-programados de forma autónoma basándose en situaciones GPS.

1.3.3 Control del UAV a través de una red WIFI

Este objetivo establece que el *UAV* desarrollado deberá ser capaz de controlarse a través de una red *WIFI*. Desde un principio se decidió que el *UAV* debería poder comunicarse con la red instalada en las lanchas de instrucción durante el desarrollo del Trabajo de Fin de Grado “Despliegue de una Red Móvil ad hoc” del A.F. López Porto-Andión. Está comunicación permitirá al *UAV* obtener datos de las lanchas y viceversa, así como un control positivo del vehículo desde la misma red. Sin embargo, en caso de existir problemas de compatibilidad, se utilizaría otra red.

1.3.4 Desarrollo del programa de control con capacidad ATOL

Este objetivo es el fin último del proyecto, sin embargo, para hacerlo posible es necesario haber cumplido los objetivos anteriores. El ser capaz de diseñar un programa de control que comunicándose a través de la red con el *UAV*, permita monitorizar y controlar su estado y movimiento.

La capacidad *ATOL* en las lanchas de instrucción en la Escuela Naval Militar es una meta que busca aunar y demostrar varios de las capacidades obtenidas mediante la resolución de los objetivos anteriores, como ejemplo de todas las posibilidades que una plataforma de este estilo proporciona. Además introduce nuevas capacidades, como la de la recepción y el análisis de datos de los sensores de las lanchas, de manera que puedan ser utilizados por el *UAV* en su comportamiento autónomo.

1.4 Futuros proyectos

La plataforma que se pretende desarrollar a lo largo de este proyecto no es un fin en sí misma, sino un primer paso hacia futuros TFG y proyectos de desarrollo, dada la increíble variedad de posibilidades que ofrece. Por esta razón se ha buscado en todo momento garantizar esta capacidad de evolución, tanto en la elección del "*hardware*", como en el diseño del sistema de navegación. Asimismo esta memoria busca describir de forma gráfica y meticulosa los pasos seguidos y las decisiones tomadas, de manera que futuros proyectos se puedan apoyar en la información que aquí se proporciona.

1.5 Organización de la memoria

Partiendo de la idea de que este proyecto es la apertura de un nueva línea de investigación para el CUD, y que se prevé realizar futuros trabajos basados en esta plataforma, la presente memoria trata de explicar de una forma detallada y secuencial todo el proceso seguido.

En el apartado "Estado del arte", se lleva a cabo una introducción a la tecnología *UAV* y su historia, centrándose en el ámbito en el que ha de desarrollarse este proyecto. También introduce una serie de conceptos básicos sobre protocolos de comunicación con *UAV*, sistemas de control y modos de operación. Finalmente describe someramente sobre qué clase de vehículo se desarrollará el sistema, explicando su funcionamiento básico y elementos principales.

En el apartado "Desarrollo" se explica inicialmente de qué forma se va a afrontar la resolución de los objetivos, y una serie de consideraciones iniciales. A continuación se describe cada uno de los elementos que compone la plataforma, así como las razones que motivaron su elección. Finalmente se relata la serie de pasos que se llevaron a cabo con el fin de alcanzar las metas del proyecto. Con la intención de facilitar la comprensión, se ha decidido dividir estos pasos en cuatro etapas, separadas por hitos importantes que constituyeron la resolución de alguno de los objetivos principales. La primera y segunda fase fueron etapas preparatorias cuyo objetivo final era disponer de una plataforma sobre la que desarrollar el sistema de control. La tercera y cuarta fases por otro lado, son el objetivo fundamental de este proyecto, la elaboración de un sistema de control con capacidad *ATOL*. Aunque varios de estos procesos coincidieron temporalmente, esta división permite una comprensión más profunda de las diferentes "capas" que componen el resultado final.

En el apartado "Resultados", se describen la plataforma final en sus diferentes configuraciones, y el Sistema de Control en los diferentes segmentos que lo componen.

Finalmente se listan las conclusiones obtenidas a lo largo del proceso. Estas conclusiones tratarán sobre la viabilidad del proyecto y las lecciones aprendidas durante su desarrollo. Además se comentarán algunas de las múltiples líneas futuras que este proyecto puede tener y las posibles combinaciones entre éste y otros Trabajos de Fin de Grado del Centro Universitario de la Defensa.

La memoria incluye también tres anexos que proporcionan información adicional: un glosario de términos, el código del programa de control, y un balance final de los costes del proyecto.

2 ESTADO DEL ARTE

"Acérquense al borde, les dijo. No podemos, tenemos miedo, contestaron. Acérquense al borde, repitió. Y se acercaron. Él los empujó...Y levantaron vuelo."

Guillaume Apollinaire

2.1 Introducción a los *UAV*

La palabra *UAV* es un acrónimo de "*Unmanned Aerial Vehicle*", o vehículo aéreo no tripulado. Un *UAV* puede ser tanto un vehículo a radio control, como un autómata capaz de llevar a cabo misiones sin intervención de un operador. Esta tecnología surgió en el ámbito militar con el fin de efectuar misiones de reconocimiento y ataque sin poner en riesgo vidas humanas. En algunos lugares el acrónimo *UAV* se ha actualizado a UAVS (*Unmanned Aerial Vehicule System*) o UAS (*Unmanned Aerial System*), con la intención de reflejar que forman parte de un sistema más complejo que incluye estaciones de control y otros elementos además de los propios vehículos. Sin embargo el término más común sigue siendo *UAV*, que ya ha pasado a formar parte del léxico popular [2].

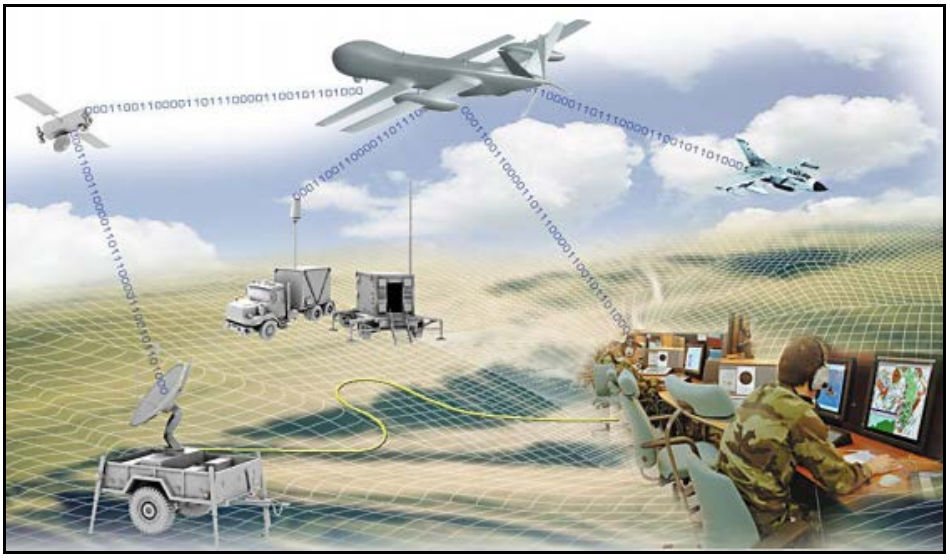

Ilustración 2-1 Representación de un UAVS (Tomado de [3]).

2.1.1 Historia de los UAV

Los *UAV*, o drones, como se les conoce comúnmente, tienen un antecedente histórico cuanto menos curioso. El primer ejemplo del que se tiene constancia del uso de una plataforma que cumple las condiciones necesarias para llamarse *UAV*, fue por parte del Imperio Austriaco en 1849. En una batalla contra la ciudad de Venecia, el ejército austriaco lanzó globos aerostáticos no tripulados cargados con explosivos desde uno de sus barcos (Ilustración 2-2). Estos globos contaban con un rudimentario mecanismo de activación basado en una pila galvánica que debía hacer estallar el globo cuando este pasara por encima de la ciudad. Desgraciadamente estos globos dependían enormemente del viento, y resultaron ser un arma muy poco precisa. Esta batalla pasaría a los anales también como el primer bombardeo aéreo de la humanidad. Aunque se aleja del concepto moderno de *UAV*, se trató del primer vehículo aéreo no tripulado capaz de llevar una carga útil [2].

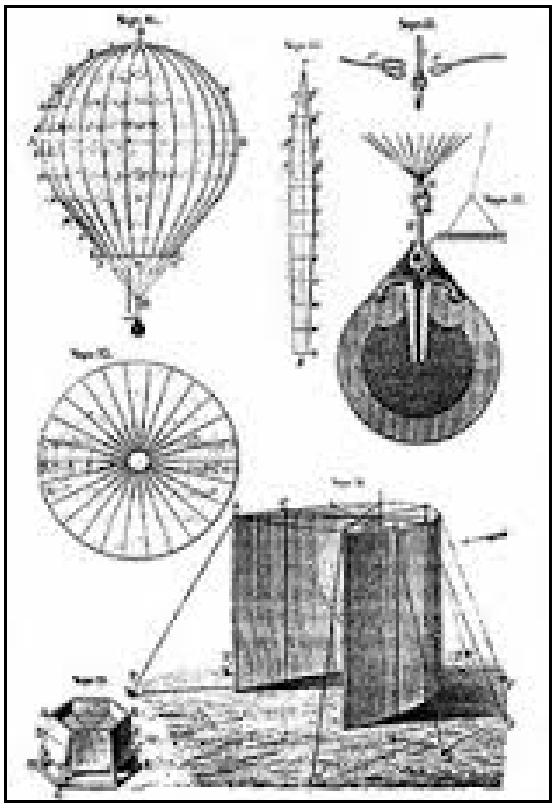

Ilustración 2-2 Globos Austriacos no tripulados 1849(Tomado de [4]).

Como en tantos otros casos, la guerra siempre supuso un incentivo en el desarrollo de la tecnología de los *UAV*. Durante la 1ª Guerra Mundial se desarrollaron los llamados "*Aerial Target*", unas pequeñas aeronaves tripuladas por radiofrecuencia que se utilizaron para mejorar el adiestramiento de la artillería antiaérea estadounidense. En 1917 la empresa "*General Motors*" realiza la primera demostración del "*Automatic Plane*", prototipo que llevaría a la construcción del "*Kettering Bug*" (Ilustración 2-3), una aeronave no tripulada diseñado bajo el concepto de "torpedo aéreo", convirtiéndose en el primer antepasado de los actuales misiles de crucero, que siguen una filosofía muy similar a los *UAV*.

Ilustración 2-3 "*Kettering Bug*" (Tomado de [4]).

Tras la *Gran Guerra*, los aviones no tripulados siguieron desarrollándose. En el periodo entre guerras surgió el "*Larynx*" diseñado por la "*Royal Navy*" entre 1927 y 1929. Fue uno de los primeros misiles jamás creados. Se montaba sobre la estructura de un pequeño monoplano que podía ser lanzado desde tierra o desde un buque de guerra. El éxito de los aviones pilotados mediante radiocontrol llevo al desarrollo de aviones que hacían de blanco en Gran Bretaña y EEUU en 1930-1931.

El siguiente paso importante se produjo con la llegada de la 2ª Guerra Mundial. En 1941 la "*Naval Aircraft Factory*" de los EEUU fabrica el primer dron de asalto. Conocido como "*Proyecto Fox*", este vehículo se controlaba desde una aeronave vecina, y contaba ya entonces con una cámara de televisión, y una pantalla en la nave controladora que le permitía guiar al *UAV*. En abril de 1942 este dron de asalto lanzo exitosamente un torpedo sobre un destructor a 20 millas del avión de control. Dicha capacidad fue escondida por parte del ejército norteamericano durante muchos años.

El siguiente hito en la evolución de los *UAV* llegó con la creación del Sistema de Posicionamiento Global, o GPS, que permitió dotar a estos vehículos de sistemas automáticos de navegación mucho más efectivos. El primer uso que se tiene registrado de un *UAV* equipado con GPS fue en 1980, en la Guerra de Yugoslavia.

No obstante, los *UAV* fueron siempre objeto de controversia. En muchas ocasiones se les consideró un caro juguete de escasa utilidad.

Esta mentalidad cambió drásticamente a nivel internacional tras la victoria de Israel frente las fuerzas sirias en 1982. El ejército israelí combinó el uso de aeronaves tripuladas y no tripuladas destruyendo hasta 86 aviones sirios. Los drones israelíes cumplían funciones de señuelo electrónico y perturbador, a la vez que les proporcionaban instrumentos de vigilancia en tiempo real, lo cual supuso una ventaja táctica y, finalmente, la derrota de Siria.

A partir de la 2ªGuerra del Golfo, los *UAV* han estado presentes en todos los conflictos armados de importancia. Sus posibilidades han ido creciendo a una increíble velocidad conforme aumentaban la capacidad de sus sensores y los medios de radiocomunicación.

Los que se han encontrado en la punta de lanza de su desarrollo, han sido las FAS de Estados Unidos de América, cuyo presupuesto en investigación de *UAV* ha aumentado exponencialmente en las últimas décadas.

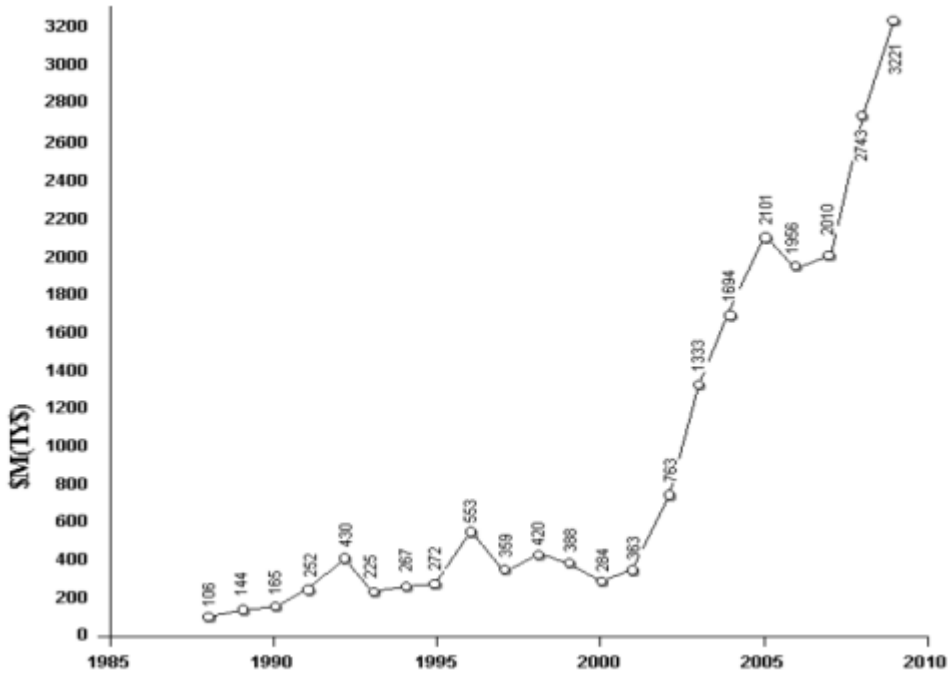

Ilustración 2-4 Presupuesto EEUU en desarrollo de *UAV* (Tomado de [3]).

Resultado de esta investigación, cabe destacar dentro de la industria armamentística el dron "*Predator*" (Ilustración 2-5) del ejército norteamericano, uno de los más avanzados que existen. Fue el primer avión no tripulado capaz de lanzar un misil de forma efectiva en el año 2002, y se encuentra a la vanguardia de la tecnología militar de vehículos no tripulados por su constante evolución.

Ilustración 2-5 Predator lanzando el misil Hellfire (Tomado de [5]).

2.1.2 Los UAV en la actualidad

Como se ha podido comprobar, hasta hace poco, los *UAV* han tenido fines exclusivamente militares. Sin embargo la evolución de la electrónica y las baterías, entre otros, han sido el caldo de cultivo que ha permitido su desarrollo en el ámbito civil.

La llegada de las baterías "*Lipo" (Litio Polímero)*, propició el desarrollo de los *UAV* eléctricos, lo que simplificó seriamente el mecanismo y permitió abaratar los costes. Gracias a ello se crearon los multicópteros actuales, cuyas aplicaciones en la vida cotidiana son incontables. La siguiente imagen recoge el crecimiento del mercado de los *UAV* en Europa durante los últimos años.

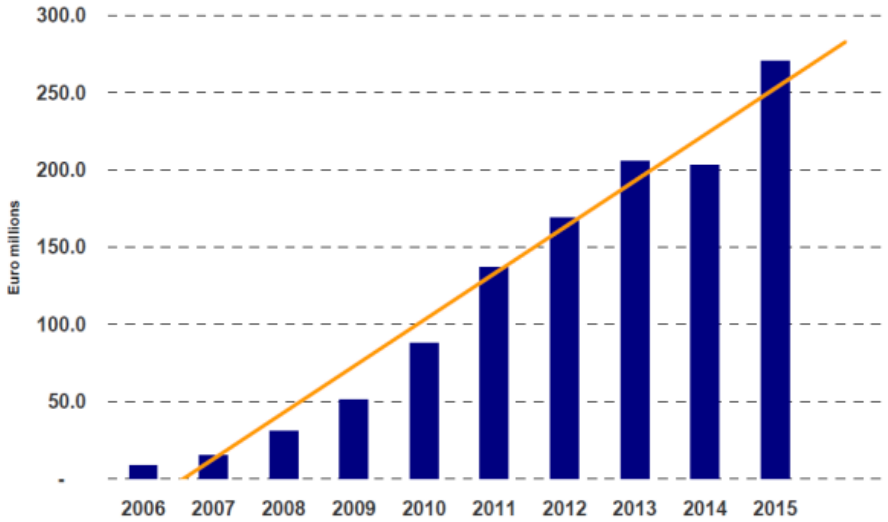

Ilustración 2-6 Valor *UAV* Europa 2006-2015 (Tomado de [3]).

Hoy en día existe una amplia gama de multicópteros y aviones con diferentes tamaños y grados de complejidad, desde los MAV (Micro Air Vehicules) que caben en la palma de una mano, hasta drones profesionales de grabación. Sus diferentes tamaños y su portabilidad los han convertido en un instrumento útil para policías, cuerpos de vigilancia y muchos otros. Varias universidades y empresas potencian proyectos de desarrollo basados en *UAV*, como la empresa de distribución Amazon con su dron de envío (Ilustración 2-8), la Universidad tecnológica de Delf (Ilustración 2-7) y su dron de primeros auxilios, equipado con desfibrilador [6].

Ilustración 2-7 Dron Ambulancia Universidad de Delf (Tomado de [6]).

Ilustración 2-8 Dron de reparto Amazon (Tomado de [7]).

Sin embargo, la proliferación de los *UAV* conlleva a su vez grandes riesgos, ya que su rápida evolución no ha permitido a las administraciones públicas legislar adecuadamente este nuevo tipo de plataformas. Por ejemplo, el dron "*A.R.*" de la empresa "*Parrot*" cuenta ya con más de 300.000 unidades vendidas a nivel mundial no registradas, y no existen leyes concretas que regulen su uso, más que una serie de notas improvisadas que regulan aspectos muy generales, a falta de una legislación más extensa [8].

En España, por ejemplo, solo existe una ley básica que restringe el uso profesional de drones a usuarios con licencia, lo cual no deja de ser poco realista por su falta de implementación. Dicha ley se aprobó el 4 de julio de 2014 y representó una solución rápida y tosca a un problema emergente. Para adquirir dicha licencia, hay que realizar una serie de pruebas teóricas y prácticas, y la plataforma debe estar homologada y contar con una serie de permisos. Solo se podrá volar en espacios aéreos previamente autorizados a tal efecto y los centros urbanos quedan totalmente prohibidos. Más tarde, en 2015 se redactó un borrador que se encuentra aún pendiente de aprobación, y modera alguna de las medidas promulgadas por la anterior ley, por ser demasiado restrictivas, acercándose a una ley que se ajuste más a la realidad vigente [9].

Por otro lado, el peligro de los drones no se limita solo al hecho de que un vecino pueda espiar a otro, o que una persona pueda ser golpeada por un aparato. Para los ejércitos y fuerzas de seguridad del estado los *UAV* suponen también una amenaza. Un dron puede ser utilizado como instrumento de espionaje internacional, contrabando, e incluso como arma. Esto no se refiere únicamente a drones militares con misiles guiados con láser, sino al dron teledirigido que se puede comprar en la juguetería de la esquina y es capaz de llevar una carga útil de 1 Kg, que podría utilizarse para acceder a recintos de seguridad y colocar explosivos, por ejemplo. Algunos cuerpos de seguridad han identificado ya esta amenaza y trabajan en medidas para contrarrestarla, como es el caso del "*Anti* Dron*e Squad*" de la policía japonesa, que irónicamente cuenta con drones como medio de captura (Ilustración 2-9) [10].

Ilustración 2-9 Dron caza-drones japonés (Tomado de [10]).

2.1.3 UAV en la Armada Española

Las Fuerzas Armadas Españolas se han unido al movimiento de desarrollo de *UAV*. El futuro de las FAS a nivel mundial, pasa por la potenciación del uso de sistemas no tripulados. Una publicación del Ministerio de Defensa [11] define las siguientes posibles aplicaciones de los sistemas no tripulados:

1. Información sobre el campo de batalla:
    a. – Análisis del daño causado.
    b. – Reconocimiento.
    c. – Inteligencia de señales.
    d. – Reconocimiento nuclear, biológico y químico.
    e. – Contra decepción.
    f. – Levantamiento digital del terreno.
2. Aplicación de la fuerza:
    a. – Señalización precisa de objetivos.
    b. – Ataque armado.
3. Mando y control:
    a. – Gestión del campo de batalla.
    b. – Relé de comunicaciones.
    c. – "*NEC*" (*Network Enabled Capability*).
4. Protección de la fuerza:
    a. – Defensa integrada de la base.
    b. – Vigilancia de convoy.
    c. – Reaprovisionamiento logístico.
    d. – Recuperación del combatiente.

Al igual que otras marinas del mundo, la Armada Española está intentando equiparse de la tecnología más actual, tanto por importación como desarrollo propio. La Armada estrenó en 2015 su

primer *UAV* en una misión real, el "*ScanEagle*" (Ilustración 2-10) de la empresa americana "*Boeing*". Durante el último despliegue del "*BAC*" (*Buque de Apoyo al Combate*) Galicia en la *Operación Atalanta* se utilizó por primera vez el dron con labores de vigilancia. El *ScanEagle* permitió obtener imágenes de los campamentos piratas situados en las playas somalíes [12].

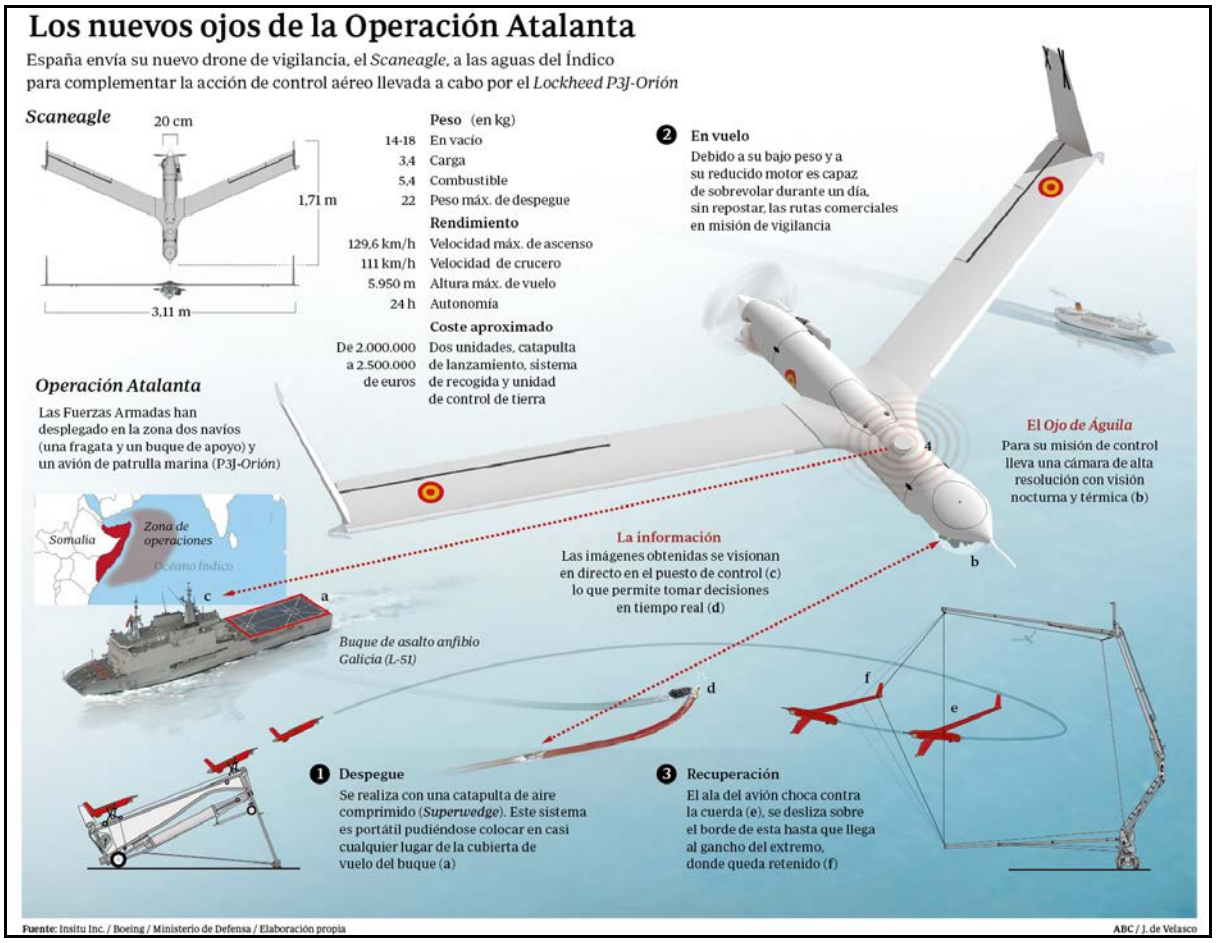

Ilustración 2-10 ScanEagle Armada Española (Tomado de [12]).

La empresa contratista de Defensa "Indra" cuenta con un proyecto de desarrollo propio de un helicóptero no tripulado, el proyecto "Pelicano". El sistema Pelícano está formado por tres o cuatro helicópteros no tripulados y una estación de control, que proporcionará capacidad para operar las 24 horas del día durante periodos prolongados. Su diseño ha sido pensado para responder a los requisitos y necesidades de las fuerzas armadas y de seguridad. La capacidad de despegue y aterrizaje vertical y pequeño tamaño del Pelicano, le convierte en la solución perfecta para respaldar cualquier tipo de operación naval. El UAVS Pelícano de Indra está pensado para prestar apoyo en tareas de vigilancia, control del tráfico marítimo, control de fronteras, en la lucha contra actividades de inmigración ilegal, narcotráfico, tráfico de armas, piratería y apoyo en operaciones de rescate, pudiendo desplegarse desde una plataforma naval o desde una base en tierra. Asimismo, estará preparado para dar apoyo, tanto en misiones de inteligencia como en gestión de emergencias, tales como desastres naturales o medioambientales, que impliquen el seguimiento, vigilancia y reconocimiento de amplias superficies, eliminando así pérdidas humanas **[13]**.

Ilustración 2-11 Pelicano INDRA (Tomado de [13]).

Finalmente, coincidiendo temporalmente con el desarrollo de este proyecto, infantes de marina de la Armada Española del Tercio Armada en Cádiz, se han estado entrenando durante el mes de enero en el manejo de multicópteros pilotados a control remoto. El dron en cuestión fue el "*Huggin X-1*", de la empresa danesa "*Sky-Watch*", recientemente adquirido por la Armada. El curso, impartido por técnicos y pilotos civiles, ha consistido en una parte teórica sobre operaciones de pilotaje y mantenimiento y una parte práctica de vuelo tanto en ambiente diurno como en nocturno.

Los futuros operadores han llevado a cabo ejercicios de preparación de misiones de vuelo, pilotaje automático y manual, localización de objetivos, y carga y descarga de productos e imágenes obtenidas durante el vuelo [14].

Ilustración 2-12 Dron Huggin X-1 Infantería de Marina Española (Tomado de [14]).

Este *UAV* concretamente pertenece por tamaño a una categoría muy similar al construido a lo largo de este proyecto y cuenta con unas prestaciones similares.

El uso de los *UAV* ha significado una gran ventaja para aquellos ejércitos que los han logrado incorporar. Mientras que el Ejército de Tierra y el Ejército del Aire llevan años explotando esta tecnología, la Armada ha empezado a hacerlo recientemente. Los *UAV* presentados en este apartado representan los primeros esfuerzos de la Armada para lograr este fin. Como figura en la Revista General de Marina, "*los UAV no son una capacidad militar del futuro, sino del presente. Por tanto es tiempo de dejar de hablar de concepto de empleo en la Armada Española y empezar a hacerlo de capacidad*" [15].

2.2 Drones "*amateur*" y de desarrollo

Se ha querido hacer hincapié en esta categoría por el hecho de ser aquella en que se desenvuelve el proyecto y donde se encuentran los elementos y documentación a los que se hará referencia a lo largo del trabajo. Este proyecto ha encontrado en esta comunidad una gran cantidad de información que ha facilitado la labor de investigación.

En los últimos 2 años los drones de uso "*amateur*" han evolucionado y se han abaratado, lo que ha permitido a instituciones públicas y privadas, así como a particulares desarrollar aplicaciones propias para esta tecnología. Existen empresas como *Parrot* o "*DJI*", que se han hecho famosas por sus drones de la categoría "*ready-to-fly*", drones cerrados listos para ser utilizados. Sin embargo también han proliferado las empresas como "*3DR*", "*Ardupilot*" o de nuevo *DJI*, que desarrollan componentes aislados, para que los usuarios construyan sus propias plataformas a su imagen y semejanza.

Dentro de este ámbito surge "*Dronecode*", un proyecto de desarrollo de vehículos aéreos no tripulados bajo una plataforma de código abierto. *Dronecode* reúne y centraliza los proyectos "*Open-Source*" para aviones no tripulados bajo una plataforma común con base en el "*Open UAV de 3D Robotics*" y tecnologías libres existentes para estos *UAV*, y también robótica en general, donde los sistemas embebidos con Linux tienen mucho peso.

Según las cifras proporcionadas por la "*Linux Foundation*", más de 1200 desarrolladores se encuentran trabajando en este proyecto de crear una biblioteca abierta de *software* y *hardware* para el desarrollo de *UAV*."*El Proyecto Dronecode tendrá una estructura de gobierno neutral y coordinará la financiación de los recursos y herramientas necesarias para la comunidad, de acuerdo con el anuncio. Todo será abierto, desde el hardware, el código de vuelo, las comunicaciones y las aplicaciones de estaciones terrestres, éstas ejecutables en entornos Linux, Windows, OS X, iOS y Android*" [16]

Es en esta comunidad en la que este proyecto encontrará su fuente de suministro, tanto material como de información en muchos aspectos, gracias a la amplia documentación disponible.

Dentro de esta categoría existen todo tipo de aeronaves, con formas tamaños y finalidades muy diferentes. Pueden utilizarse como "*hobby*", competiciones de acrobacias, carreras o simplemente exhibición. No obstante, la mayoría de ellos comparten una serie de características similares y protocolos de comunicación.

2.2.1 Protocolo Mavlink

Dentro de este ámbito no se puede hablar de *UAV* sin hablar del protocolo de mensajes "*Mavlink*" (*Micro air vehicules link*) (Ilustración 2-13). *Mavlink* fue creado en 2009 por Lorenz Meier, un estudiante de máster de la Universidad de Zúrich, bajo licencia "*GNU*" (*Licencia Publica General Reducida*), lo significa que es gratuito y abierto.

El protocolo *Mavlink*, es un protocolo de comunicación originado para vehículos aéreos no tripulados, aunque hoy en día se aplica también en determinados vehículos terrestres. Los mensajes que componen este protocolo están programados en el lenguaje de programación "*Python*", un lenguaje sencillo y de alto nivel. Su principal uso es la comunicación entre una "*GCS*" (*Ground*

Control Station) y un *UAV* así como en la intercomunicación de los subsistemas del vehículo. Puede utilizarse para transmitir información de la orientación espacial del vehículo, su velocidad, su situación y otros parámetros, así como órdenes por parte de la *GCS* al vehículo. Puede tratarse desde órdenes simples como un armado y desarmado, hasta comandos más complejos como la grabación de una ruta de *waypoints* o el control de una cámara incorporada en el vehículo [17].

Ilustración 2-13 Logo Protocolo *Mavlink* (Tomado de [18]).

Cuando un usuario da una orden a un *UAV* a través de una estación de control, ésta traduce los comandos del usuario a mensajes *Mavlink*, que son aquellos que el vehículo es capaz de interpretar. Cada mensaje *Mavlink* tiene una longitud de 17 bytes y tiene la siguiente estructura:

Mensaje=17 bytes=6 bytes Header+9 bytes payload+2 bytes checksum

- El "*header"* contiene la longitud del mensaje, el número de secuencia, la identificación de Sistema que lo manda y la del subsistema que lo ha generado. Introduce además cuál de los diferentes tipos de mensaje es.
- El "*payload*" contiene la información que se quiere transmitir. Dentro de este apartado, los datos que se pretende transmitir se estructuran de una determinada forma atendiendo al tipo de mensaje del que se trate. Algunos de los tipos de mensajes más comunes son:
  - *MAVLINK*_MSG_ID_HEARTBEAT: es el mensaje más importante. La *GCS* y el *UAV* se envían un latido de forma intermitente. Éste sirve para comprobar si la conexión es efectiva, y permite sincronizar algunos parámetros como el tiempo. Si un número determinado de latidos no son recibidos por el *UAV*, este procederá a activar el protocolo de emergencia que tenga programado ante una pérdida de señal de control.
  - *MAVLINK*_MSG_ID_REQUEST_DATA_STREAM: permite pedir información de diferentes índoles, como datos GPS, parámetros de vuelo o información de sensores adicionales.
  - *MAVLINK*_MSG_ID_COMMAND_LONG: contiene mensajes con órdenes de larga duración como, como RTL, LAND, Arm/Disarm, etc.
- El "*Checksum*" es una suma de verificación que comprueba que el mensaje ha llegado de forma íntegra.

A pesar de que a lo largo del proyecto la elaboración de los mensajes *Mavlink* se elaborará a un alto nivel, es necesario conocer y comprender en qué idioma está sucediendo la comunicación con el *UAV* para su correcto desarrollo.

2.2.2 Modos de vuelo

Los vehículos desarrollados con el software proporcionado por *Dronecode* y el protocolo *Mavlink* tienen asociados una serie de conductas predefinidas de vehículos llamadas *modos de vuelo*. Estos modos de vuelo definen comportamientos específicos del vehículo, y son un paso fundamental hacia su comprensión y desarrollo.

Existen muchos modos de vuelo diferentes soportados por *Mavlink*, y conforme se desarrollan nuevas aplicaciones surgen modos nuevos. Cada uno de estos modos define unas reglas de

comportamiento básicas por las cuales se regirá el *UAV* mientras se encuentre en él. No obstante, no todas las plataformas son capaces de soportar los mismos modos de vuelo, ya que muchos describen comportamientos exclusivos de algunos tipos de vehículo. Por ejemplo, un avión no puede mantenerse estático en una posición, ya que depende de su velocidad como método de sustentación, mientras que un multicóptero si puede. Estos modos de vuelo se le ordenan a la controladora de vuelo mediante un enlace radio, o bien mediante un enlace a través de una *GCS* [19].

Es importante conocer los modos de vuelo que un vehículo es capaz de implementar, y qué reglas rigen cada uno de ellos. Esto permitirá sacarles el máximo partido y evitará poner en riesgo la seguridad del *UAV*. De la misma forma, en el desarrollo de las futuras aplicaciones, se combinarán en muchas ocasiones las oportunidades que ofrecen los diferentes modos.

Este proyecto utiliza como plataforma un multicóptero, por lo que a continuación se describen algunos de los modos de vuelo más importantes que es capaz de soportar:

- "*STABILIZE*": es el modo de vuelo más básico. Este modo busca únicamente estabilizar el multicóptero de forma automática. Esto no es necesario en un avión, o helicóptero, pero el movimiento de un multicóptero es el resultado de la composición de un complejo sistema de propulsores, por lo que este modo se vuelve imprescindible. En este modo el multicóptero reacciona a las órdenes del piloto, y en ausencia de estas se estabiliza solo.
- "*AUTO*": es el modo autónomo por excelencia. Basa su funcionamiento en información GPS. En este modo la plataforma ejecuta de forma automática una ruta pre planeada y grabada en la memoria interna de la controladora de vuelo. Dicha ruta puede contener diferentes tipos de órdenes, como aterrizajes, despegues, cambios de altura o tiempos de permanencia en un mismo punto ("*loiter*"). Los parámetros que rigen el movimiento del *UAV* en este modo, como la velocidad, la altura, o la precisión, pueden ser definidos con antelación. Este modo es también el más restrictivo en cuanto al control, ya que el *UAV* no obedecerá a ninguna orden mientras se encuentre en este modo.
- "*LOITER*": es un modo sencillo y útil que mantiene al *UAV* en la situación en la que se encuentre, tanto en latitud, longitud como en altura. Al igual que el modo *AUTO*, basa su funcionamiento en la información que recibe del GPS. Permite dar órdenes a la controladora y mover el vehículo, pero ante la ausencia de estas, mantendrá la situación en la que se encuentre.
- "*LAND*": este modo, como su nombre indica, aterriza el vehículo. En este modo el vehículo descenderá a una velocidad predeterminada, que puede ser alterada accediendo a la configuración. Este modo no depende de información GPS ni de altura. El vehículo seguirá descendiendo hasta que la controladora de vuelo detecte que a pesar de reducir el empuje de los rotores hasta un 20% no se reduzca la cota.
- "*RTL*": o "*return to launch*", es un modo de vuelo autónomo en el cual el *UAV* vuelve de forma automática a la posición "*Home*" y aterriza. La posición *Home* se corresponde con la última situación en la que el vehículo fue armado. Estando este modo activado, el vehículo no responderá a ninguna orden, más que a un cambio de modo. La velocidad y altura con las cuales se lleva a cabo este modo pueden variarse accediendo a la configuración de la controladora.
- "*GUIDED*": es un modo especializado para *GCS* que no se puede ejecutar desde una emisora. Es un modo autónomo GPS más versátil que el modo *AUTO*. Mientras que en el modo *AUTO* el vehículo efectúa una ruta prefijada grabada en la controladora de vuelo, en el modo *GUIDED* los objetivos se pueden definir desde la *GCS* de una forma dinámica. En este modo un comando puede ser interrumpido por uno posterior, y los parámetros de vuelo son configurables. Estas características lo convierten en un modo muy útil para el desarrollo de aplicaciones.

2.2.3 Ground Control Stations

Una "*GCS*" (*Ground Control Station)*, o estación de control en tierra, es una aplicación de *software* que se comunica con un vehículo autónomo de diferentes formas posibles, típicamente de forma inalámbrica. Despliega información en tiempo real de la plataforma y sus instrumentos, y permite tener un control de su posición y sus parámetros. Las *GCS* más complejas ofrecen funciones adicionales como registro y análisis de misiones.

En el caso de los *UAV* las *GCS* adquieren gran importancia, por la distancia a la que se puede encontrar el vehículo y por la cantidad y variedad de datos que permiten monitorizar. También dan la posibilidad en muchas ocasiones de grabar misiones en la plataforma, o llevar un control positivo de ella desde la *GCS* [20].

En el ámbito de los *UAV amateur* y de desarrollo entre los cuales se encuentra este proyecto, existen varios modelos de *software* comerciales que realizan la función de *GCS*, como "*Mission Planner*", "*QGroundcontrol*", "*Mavproxy*", o "*APMPilot*". Estos programas se comunican con el vehículo a través de mensajes *Mavlink*. Pueden encontrarse instalados en todo tipo de dispositivos electrónicos con diferentes posibilidades, como ordenadores portátiles, teléfonos móviles y tabletas, o como es en el caso de éste proyecto, en una "*Raspberry Pi*". Las instaladas en medios portátiles muchas veces ofrecen opciones como "*Follow me*", mientras que las instaladas en ordenadores permiten acceder a parámetros de configuración muy específicos, y realizar un intenso estudio de los datos registrados de la plataforma.

En el desarrollo de los objetivos del proyecto, será necesario el diseño de una pseudo-estación de control propia, que permitirá desarrollar comandos que se ajusten a las necesidades específicas de este proyecto. Sin embargo, a lo largo del trabajo, se hará uso en múltiples ocasiones de las herramientas que ofrece la *GCS Mission Planner*, tanto de configuración, como de control y monitorización de datos. Además la configuración final permitirá la conexión de diferentes equipos con una *GCS* instalada, de forma que el vehículo se podrá controlar desde diferentes estaciones.

2.3 Plataforma

La denominación *UAV* hace referencia a cualquier vehículo aéreo no tripulado, ya sea un biplano de ala fija, un helicóptero o un multicóptero, por lo que es un término ,en ocasiones, poco específico. De entre las diferentes alternativas, para la realización de este proyecto se ha escogido como plataforma un *multirotor*, o *multicóptero*. Por ello se ha creído conveniente una explicación más detallada de este tipo de vehículos, su funcionamiento y elementos principales.

2.3.1 Multicóptero

Un multicóptero es un aparato volador mecánicamente simple que cuenta con 3 o más hélices propulsadas por motores eléctricos, que le proporcionan sustentación. Un multicóptero controla su desplazamiento y estabilidad variando la velocidad de giro de los diferentes rotores, de forma que la dirección del vector resultante de las diferentes fuerzas sea la del movimiento deseado. De esta forma un multicóptero puede desplazarse en todas las direcciones del espacio y girar en torno a cualquiera de sus ejes, lo que le convierte en un medio extremadamente versátil.

Sin embargo, esta clase de aeronaves son aerodinámicamente inestables, ya que no basan su sustentación en su forma, sino en un empuje constante y difícil de regular producido por los rotores, y por ello necesitan una controladora de vuelo midiendo y corrigiendo constantemente para funcionar correctamente. Al contrario que en las aeronaves de ala fija, si se produce un fallo en la controladora o en un determinado número de propulsores, el aparato caerá irremediablemente. Esta controladora cuenta con una serie de sensores que previamente calibrados, le permiten conocer su posición espacial, como giróscopos y acelerómetros. Conociendo su orientación espacial, la controladora de vuelo ordena

a los "*ESC* "(*Electronic Speed Controller*) cambiar la velocidad de los rotores que tienen asignados, para con volver a la posición de equilibrio, u obtener el movimiento deseado.

2.3.2 Funcionamiento

El tipo de multicóptero más sencillo es el cuadrocóptero (Ilustración 2-14), que cuenta con cuatro hélices, normalmente dispuestas en 45º,135º,225º y 315º. Todos las hélices son propulsadas por motores eléctricos, y todas ellas cumplen tanto la función de sustentación, como de control, al contrario que en los helicópteros convencionales, cuyo método de sustentación es únicamente el rotor principal. Los rotores del dron generan dos tipos de reacciones: empuje vertical y par motor (Ilustración 2-15).

Ilustración 2-14 Cuadrocóptero (Tomado de [21]).

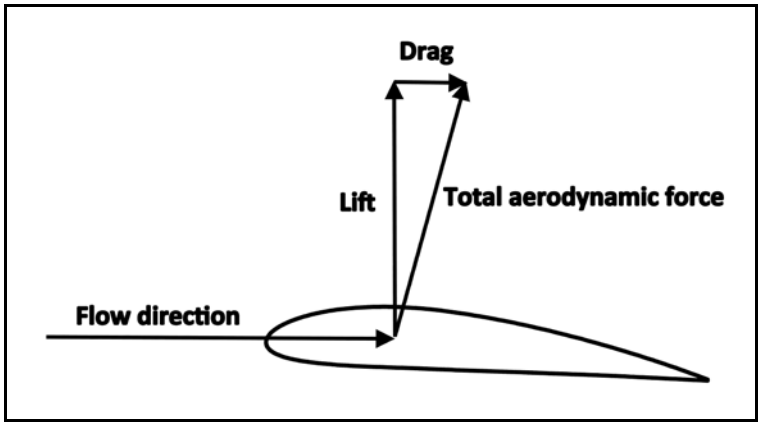

Ilustración 2-15 Reacciones hélice (Tomado de [20]).

El empuje vertical ("*lift*") es la reacción más intuitiva de la hélice. Su funcionamiento es igual al de un helicóptero o un ventilador. La diferencia de presiones provocada por la velocidad relativa de la hélice con respecto a la masa de aire en la que se mueve provoca una fuerza de la zona de mayor presión (bajo la pala) hacia la zona de menor presión (sobre la pala). De esta forma, si se aplica la misma potencia a cada uno de los rotores y la plataforma está correctamente equilibrada el multicóptero ascenderá verticalmente. Sin embargo, si una de las hélices gira a menor velocidad que

las demás, se producirá una inclinación en el vehículo, de forma que el empuje ya no será exclusivamente vertical, sino que tendrá una componente horizontal, lo que permitirá que el vehículo se desplace en esa dirección (Ilustración 2-17). A diferencia que los helicópteros, que deben cambiar la inclinación de su rotor principal con complejos sistemas hidráulicos, un multicóptero solo necesita aplicar la velocidad necesaria a cada hélice para obtener el vector de empuje deseado. Esto permite tener hélices fijas, lo que simplifica mecánicamente el aparato. Otra de las ventajas de este tipo de aeronaves, es que carecen de una proa en el sentido convencional de la palabra, por lo que, con la adecuada combinación de velocidades, se puede conseguir un desplazamiento en cualquiera de los ejes.

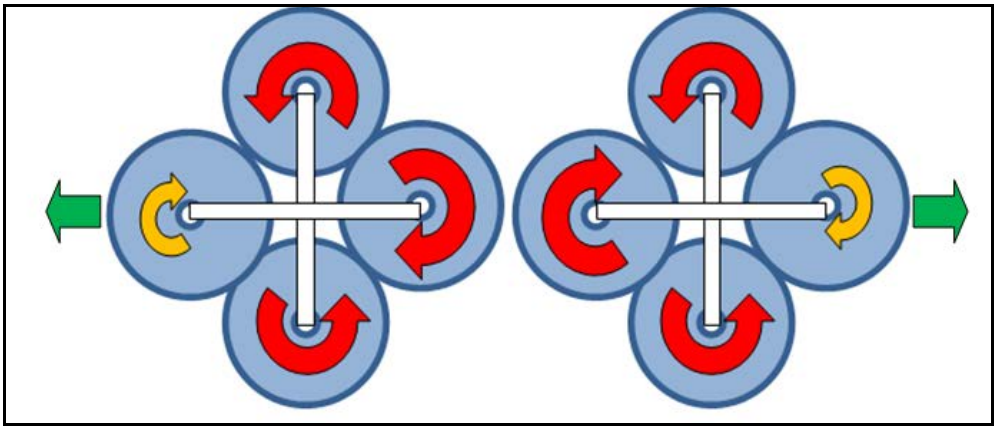

Ilustración 2-16 Ejemplo desplazamiento Multicóptero (Autoría Propia).

El par rotor consiste en la reacción horizontal que produce en el aparato el giro de la hélice. Por conservación de la cantidad de movimiento, cuando la hélice gira en un sentido, el cuadrocóptero "tratará" de girar en el sentido opuesto. Los helicópteros compensan esta reacción con el rotor de cola, que les proporciona un momento contrario a la reacción del par rotor, anulando a este último. La forma que tienen los multicópteros de solucionarlo, es mediante hélices contra rotatorias. Cada hélice gira en sentido opuesto a las dos adyacentes, de forma que el momento total se compensa, y el aparato puede permanecer estático. Sin embargo, de la misma forma que se ajusta la velocidad de las hélices para regular la inclinación del plano de la aeronave, podemos "descompensar" este momento de forma controlada, para que el cuadrocóptero gire sobre sí mismo. Esto queda descrito en la Ilustración 2-17.

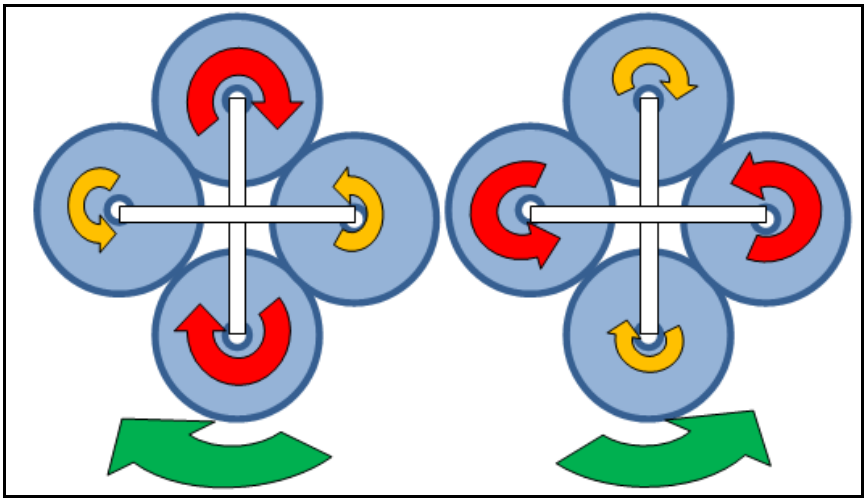

Ilustración 2-17 Ejemplo Giro Multicóptero (Autoría propia).

La combinación de todos estos movimientos convierte a los multicópteros en plataformas extremadamente versátiles, capaces de desplazarse en cualquier dirección espacial y girar sobre cualquiera de sus ejes.

El funcionamiento aquí descrito, a pesar de haber tomado como ejemplo el cuadrocóptero, es aplicable a los demás multicópteros, ya sean de 6, 8, 10, 12 hélices o más. A mayor número de rotores mayor estabilidad y mayor capacidad carga útil obtendremos.

2.3.3 Elementos principales

A pesar de la gran variedad de aparatos que existen la mayoría de los multicópteros comparten una serie de elementos, que pueden presentarse de distintas formas [22]:

- Controladora de vuelo ("*Flight controller"*): es una de las principales diferencias entre los multirotores y otros vehículos, como los aviones o helicópteros. En estos vehículos una acción del piloto conlleva una reacción concreta en el aparato. Tirar del colectivo de un helicóptero provoca una variación fija del paso de la hélice y pisar el acelerador de un coche aumenta la cantidad de combustible que se introduce en el motor siempre en la misma medida. Sin embargo, en un multicóptero, dada la complejidad que supone controlar manualmente las revoluciones de cada uno de los rotores, se utiliza una controladora de vuelo. Ésta consiste en un pequeño procesador que interpreta las órdenes del piloto, ya sea automático o no, y calcula las reacciones necesarias para obtener ese resultado. Esa información es transmitida como señales electrónicas a los ESC, que son los que efectivamente regulan la velocidad de los rotores. Generalmente cuentan con sensores que aportan información a esta toma de decisiones. Estos sensores van desde giróscopos (para poder medir la velocidad angular de los cambios de posición) y acelerómetros (que les permiten conocer su inercia) hasta barómetros, GPS y muchos otros. Existe una gran variedad de controladoras de vuelo de muy diferentes complejidades y precios, empezando por la de un juguete teledirigido, hasta la de un aparato de grabación profesional.
  Por lo tanto, si el piloto es el cerebro del vehículo, la controladora de vuelo se convierte en su médula espinal. Recibe las órdenes del piloto, y con el conocimiento del medio y de sí misma que le proporcionan sus sensores, lleva a cabo las acciones individuales que resultan en el cumplimiento de la orden

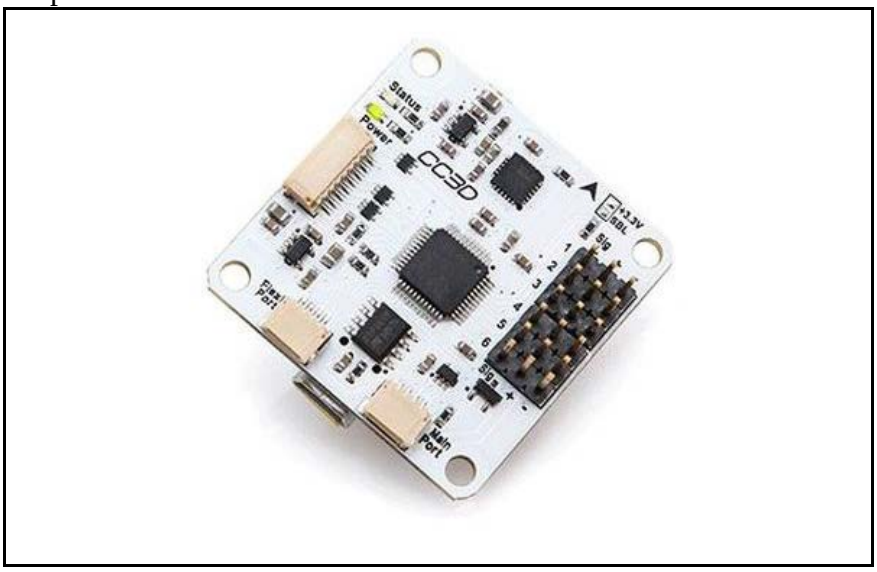

Ilustración 2-18 Controladora de Vuelo Lumenier CC3D (Tomado de [22]).

- ESC (*Electronic Speed Controller*): son los responsables de que los motores giren a la velocidad ordenada por la controladora de vuelo. Se colocan como intermediarios entre la fuente de alimentación y los motores, regulando así, de qué forma se les alimenta. Los ESC interpretan la señal recibida de la controladora, y ajustan la velocidad del rotor acordemente. Su funcionamiento está ligado al tipo de motor cuya velocidad nos interese regular, por lo que distinguiremos ESC de escobilla (*brushed*) y sin escobilla (*brushless*).
  Pueden ser individuales y estar cada uno asignado a un motor, como los que figuran en la Ilustración 2-19, o tratarse de un único elemento que controle la velocidad de los diferentes motores [23]. Los ESC *brushless* proporcionan una corriente trifásica de bajo voltaje que alimenta al motor. Permiten una variación mucho más suave, más precisa de la velocidad del motor y de una manera mucho más eficiente reguladores mecánicos de la velocidad.

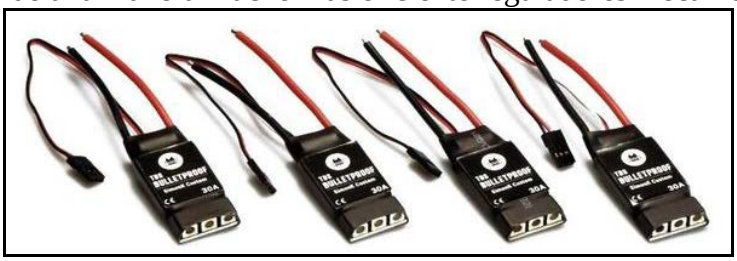

Ilustración 2-19 ESC Bulletproof (Tomado de [22])

- Servomotores: motores eléctricos de posición fija y pequeño tamaño a los cuales van acopladas las hélices. Su velocidad debe poder ser regulada.
  Pueden ser de escobillas (*brushed*) o sin escobillas (*brushless*), como el representado en la Ilustración 2-20 [22]. Los motores sin escobillas son conocidos como motores ECM (*Electronic Conmuted Motors*) y son los más utilizados. Los motores sin escobillas ofrecen mejores prestaciones que los motores con escobillas, incluyendo un mayor par por peso, mayor eficiencia eléctrica, mayor fiabilidad, reducción de ruido, mayor vida útil y la reducción de las interferencias electromagnéticas. Además, estos motores no requieren un flujo de aire a través de ellos como método de refrigeración, lo cual significa que los componentes internos del motor pueden ser totalmente cerrados y protegidos de la suciedad u otras materias extrañas [24].

Ilustración 2-20 Ejemplo de motor *brushless* (Tomado de [22]).

- Chasis: es la estructura que sustenta los demás elementos. Existen una gran variedad de formas, tamaños y materiales. Estructuras más rígidas proporcionan una mayor calidad de vuelo, ya que reducen las deformaciones y vibraciones. No obstante, si la estructura es demasiado rígida corre un mayor riesgo de dañarse en caso de ocurrir un accidente. Necesitan ser resistentes y lo suficientemente ligeras como para que no supongan un lastre. Estos requisitos hacen de la fibra de carbono y los plásticos unos de los materiales más utilizados.

  Algunos tipos de chasis llevan el cableado incorporado, de forma que se pueda obtener un resultado más organizado y fiable (Ilustración 2-21 Paneles superior e inferior con cableado incorporado (Tomado de [25]).

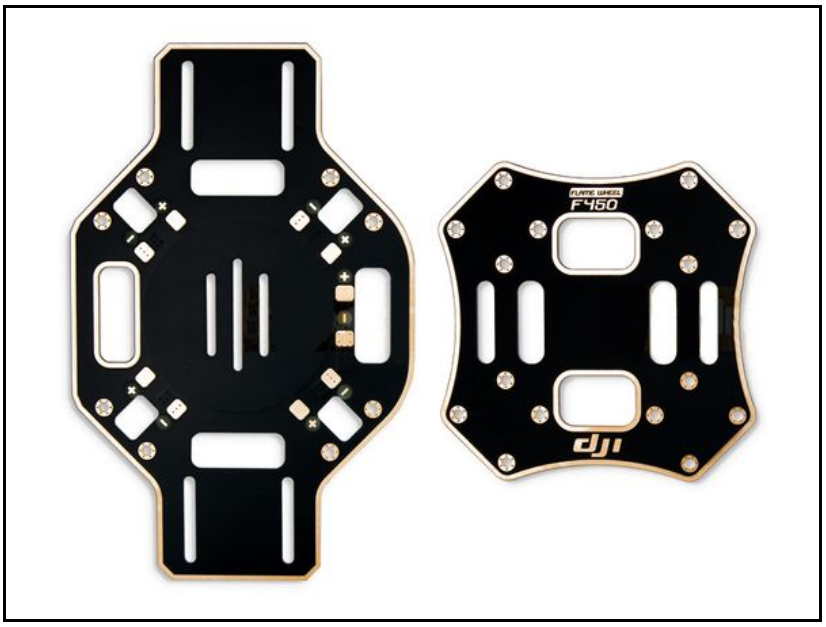

Ilustración 2-21 Paneles superior e inferior con cableado incorporado (Tomado de [25]).

- Brazos: son los elementos que sostienen los motores. Deben ser lo suficientemente resistentes para sostener la estructura. A pesar de ello, muchas veces se escogen materiales ligeramente frágiles, porque es preferible que sean el primer elemento en fallar en caso de haber un accidente. Este método busca proteger la electrónica frente a los brazos, que son más baratos y fácilmente sustituibles.

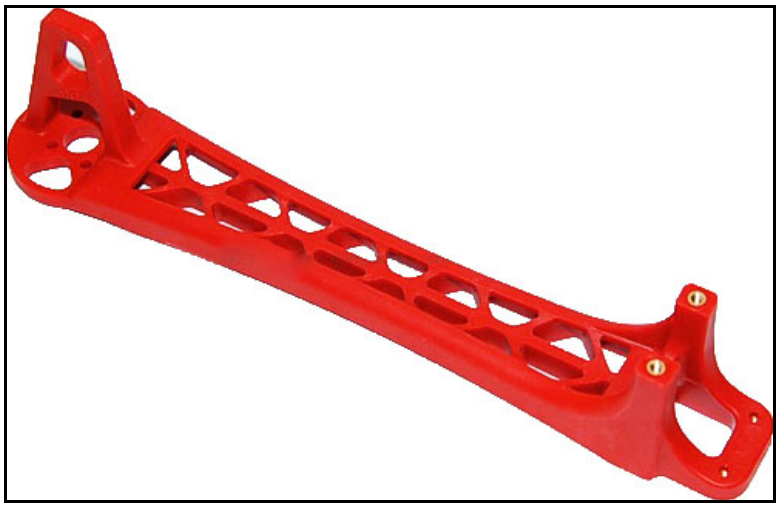

Ilustración 2-22 Brazo de multicóptero (Tomado de [25]).

- Emisor y receptor: estos dos elementos entran en escena cuando hablamos de sistemas de control remoto. El receptor va montado en la plataforma y va conectado a la controladora de vuelo aunque en ocasiones forma parte de ella. Consiste en una antena receptora de ondas de radiofrecuencia que funciona en una frecuencia determinada, siendo la más utilizada la banda de 2,4 GHz [26]. Por otra parte el emisor, puede adquirir muchas formas, pero clásicamente es un mando de control remoto cuya frecuencia es igual a la del receptor, o tiene capacidad de sintonizarse con él. Emisores más complejos permiten el uso de varios canales, así como de programar qué señal corresponde a cada interruptor.

Ilustración 2-23 Emisora Futaba 10-J (Tomado de [27]).

2.3.4 Baterías Lipo

Las baterías "*Lipo*" (Litio-Polímero) son un tipo de batería recargable que suelen utilizar los sistemas eléctricos de radiocontrol, especialmente los aviones, helicópteros y multicópteros. Estas

baterías son una de las razones por las cuales el vuelo eléctrico se ha convertido en una opción viable. [24]

Ilustración 2-24 Batería *Lipo* 3S 5000mAh 25C (Tomado de [28]).

Las razones que hacen de las baterías *Lipo* la elección perfecta para el vuelo a radiocontrol son:

- Son ligeras y pueden fabricarse de casi cualquier forma y tamaño.
- Tienen gran capacidad en relación a su tamaño.
- Permiten una tasa de descarga alta para alimentar los sistemas eléctricos más exigentes.

Por otra parte, este tipo de baterías también tiene una serie de inconvenientes:

- A causa del electrolito volátil que se utiliza exclusivamente en las *Lipo*, corren el riesgo de incendiarse o incluso explotar.
- Requieren un cuidado meticuloso para obtener una buena esperanza de vida. La carga, la descarga y almacenamiento deben hacerse en las condiciones adecuadas.

Las baterías *Lipo* siguen una estructura de celdas. A diferencia de las baterías *NiCd* o *NiHm* que contienen celdas de 1.2 voltios, en una batería *Lipo* cada celda tiene un voltaje de 3.7 V (4.2V cuando están completamente cargadas), y colocando una serie de celdas en serie obtenemos voltajes mayores. El número de celdas en serie que tiene una batería *Lipo* viene definido por un número seguido de la letra "*S*". De ésta forma:

- *Li-PO* 1S: una celda, 3.7 V.
- *Li-PO* 2S: dos celdas en serie, 7.4 V.
- *Li-PO* 3S: tres celdas en serie, 11.1 V.
- *Li-PO* 4S: cuatro celdas en serie, 14.8 V.

Los vehículos más pequeños suelen utilizar baterías de 1 o 2 celdas en serie, pero vehículos mayores pueden llegar a utilizar baterías de hasta 6 celdas o más si cuentan con elementos que utilicen alta tensión.

Los baterías también se pueden conectar en paralelo para aumentar su capacidad. Esto se indica mediante un número seguido de una "*P*". Por ejemplo *3S2P* indica 2 baterías conectadas en paralelo de tres celdas en serie cada una.

La capacidad indica cuánta energía puede almacenar la batería y se indica en miliamperios (mA/h o mAh). Así se indica la cantidad que es capaz de descargar la batería, medida en miliamperios, durante 1 hora para que la batería quede completamente vacía.

Por ejemplo, una batería *Lipo* de 1000 mAh tardaría una hora en descargarse conectada a un elemento que exigiese 1000 mA. Si el elemento pidiese 500 mA, la batería tardaría dos horas en descargarse.

Como es lógico deducir, para este tipo de vehículos, la capacidad de la batería es una característica vital. Al contrario que el voltaje, una mayor capacidad siempre va a resultar beneficioso. Sin embargo no se puede aumentar de forma indefinida, ya que un aumento de la capacidad trae consigo un aumento de peso y tamaño de la batería.

Otra característica importante de las baterías *Lipo* es la tasa de descarga. Esta viene definida por dos números seguidos por una “C”. La primera cifra indica el número de “C” o de capacidades que la batería es capaz de descargar de forma continua, es decir, una batería de 1000 mA/h 10C será capaz de descargar hasta 10000 mA. La segunda cifra representa la carga de pico, la mayor carga que es capaces de soportar durante un corto periodo de tiempo (en torno a 3 segundos) sin que esto dañe a la batería.

Estas baterías son, por tanto, una solución muy útil para el diseño de vehículos aéreos eléctricos por su relación voltaje/peso y capacidad/peso. Sin embargo son elementos delicados que requieren de un uso meticuloso y unas condiciones de carga, descarga y almacenamiento muy concretas.

3 DESARROLLO DEL TFG

"Con el genio se inician las grandes obras, pero sólo con el trabajo se las acaba."

Joseph Joubert

3.1 Consideraciones iniciales

Para el desarrollo del proyecto, se han utilizado en su gran mayoría elementos comerciales. Como quedó descrito en el apartado 2.2.3, en los últimos años los vehículos autónomos han sufrido una revolución, y no solo en aplicaciones militares y profesionales, sino también a nivel aficionado y particular. Con ello, muchas empresas lanzan al mercado sus productos, creando una enorme nube de entre la que hay que elegir los elementos que más se ajustan a las necesidades de cada uno.

En este primer apartado quedan recogidas las consideraciones iniciales que se tuvieron a la hora de escoger el tipo de plataforma y los modelos de *hardware*, con el fin de construir una plataforma versátil, capaz de cumplir los objetivos y con una amplia capacidad de desarrollo.

3.1.1 ¿Por qué un multicóptero?

Mientras que la propuesta del TFG hablaba de un *UAV*, éste desde un principio se ha encaminado a una plataforma muy concreta, los multicópteros. Tras definir brevemente que es un multicóptero, cómo funciona y los elementos que lo componen, es más sencillo entender las razones por las que se ha elegido este tipo de aeronave. Las principales ventajas que un multicóptero aporta son:

- Sencillez: son aparatos mecánicamente sencillos. Este es un concepto en el que se ha insistido repetidamente. Los únicos elementos propiamente mecánicos que los multicópteros presentan son los servomotores. Éstos tienen un funcionamiento relativamente sencillo y son independientes unos de otros, por lo que un fallo en uno de ellos no afecta a los demás. Los motores son además baratos y fácilmente sustituibles. El resto del funcionamiento se basa en dispositivos aislados cuyo cableado no es excesivamente complejo.
- Seguridad: las diferentes fuentes de sustentación proporcionan un margen de seguridad. Si el multicóptero ha sido adecuadamente dimensionado, el fallo de uno de los rotores puede suplirse con la ayuda de los demás en multicópteros de 6 o más hélices.
- Flexibilidad: el no estar especialmente condicionado por su forma, les proporciona infinidad de opciones a la hora del montaje de la carga de pago. Esto permite que los

cálculos necesarios para el vuelo del aparato no necesiten ser excesivamente precisos, sino ajustarse a unos márgenes de seguridad.

- Maniobrabilidad: como se ha dicho anteriormente, es una plataforma capaz de moverse en todas las direcciones espaciales y realizar giros sobre todos sus ejes. Puede alcanzar velocidades de un rango menor a las de un avión, pero no depende de su velocidad para conseguir sustentación, por lo que puede mantenerse estático en una posición en el aire ("*hover*").
- Estabilidad: la controladora de vuelo garantiza la estabilidad del vehículo, lo que protege los elementos que lo componen y lo hace apto para labores de grabación de imágenes.
- Capacidad de desarrollo: este tipo de vehículos se encuentran en pleno auge, y cada día surgen más complementos, programas y actualizaciones. Pueden llevar muchos tipos de sensores diferentes y configurarse para una amplia gama de labores.
- Precisión: la capacidad de llevar a cabo un vuelo estático y efectuar pequeñas correcciones de posición en el aire, permiten a este tipo de vehículos mantenerse y posarse en situaciones GPS muy concretas.
- Futuros proyectos de desarrollo: mientras que para este trabajo se plantean como objetivos construirlo y un par de aplicaciones concretas, es un buen punto de partida para muchos futuros proyectos.

Estos son pues, las causas que motivaron la elección de un multicóptero como la plataforma adecuada para realizar este proyecto.

3.1.2 Aproximación a los objetivos

Este apartado pretende explicar cuál fue el razonamiento que llevó a la configuración final del aparato de entre todas las posibles.

Partiendo de la base de que un multicóptero era la opción más adecuada, fue necesario de qué forma se resolverían los objetivos de la autonomía y la integración en la red de las lanchas de instrucción de la ENM, y una vez definidos los elementos necesarios, sería posible hacer una estimación de la plataforma que habría que construir.

Para cumplir los objetivos se consideró necesario que el *UAV* contara con las siguientes capacidades:

- Un entorno programable que permitiera el desarrollo de los programas necesarios para el gobierno autónomo del vehículo, así como la conexión a la red y extracción de datos de ella.
- Una antena compatible con la red "*MANET*" (*Mobile ad hoc network*) de las lanchas de instrucción y que le permita a su vez un gran alcance.
- Autonomía suficiente como para realizar misiones sencillas.
- Conocimiento de la situación GPS propia con un grado elevado de precisión
- Una estación de control en tierra que permita monitorizar el comportamiento de la plataforma e introducirle las misiones.

En base a estas necesidades básicas se concluyó que el *UAV* debería incorporar una computadora a bordo que se comunicara con la controladora de vuelo y le diera instrucciones basadas en los programas desarrollados. Esta computadora ha de ser capaz de comunicarse con la estación de control a través de la red, por lo que necesitará una antena que le permita el acceso a ella. Además el *UAV* necesitará también un módulo GPS que le permita conocer su posición, punto de partida para cualquier comportamiento autónomo. Se buscará también contar con métodos de control y medios materiales redundantes que aseguren la integridad del vehículo o su posible reparación o sustitución.

3.2 Elementos

Basándose en las necesidades definidas del proyecto, se adquirieron los elementos necesarios. En este apartado se describen los numerosos dispositivos que componen la plataforma y elementos asociados y se detallan las razones que motivaron su elección.

Los elementos adquiridos fueron los siguientes:

Tabla 3-1 Productos adquiridos para el proyecto

Elemento	Cantidad
Vehículo:	
Kit DJI F550	2
Raspberry Pi 2 B	2
Pixhawk	2
Módulo Compás+GPS para Pixhawk	2
EzUHF RX 8ch	1
Ubiquity Picostation M2 HP	1
Batería Desire Power V8	4
Cámara IP Edimax IC-3115W	1
Switch Edimax Es-3305P	1
Maxbotix 12CXL- MaxSonar EZ4	1
Asociados:	
Fr Sky Taranis	1
Cargador de baterías Graupner Ultramat	1
EzUHF TX LRS 433MHz	1

Muchos de estos dispositivos, especialmente aquellos que van incorporados en el vehículo se adquirieron duplicados, con la intención de disponer de un vehículo de sustitución que garantizara el desarrollo del trabajo en caso de la pérdida de uno. Esto aumentó la carga de trabajo que supuso el montaje, al tener que montar dos plataformas.

A pesar de ello, se trata de una plataforma flexible, que da la oportunidad de disponer de diferentes configuraciones en las cuales se puede prescindir de determinados dispositivos, gracias a la redundancia y la seguridad que proporcionan los diferentes medios de control.

3.2.1 Controladora de vuelo Pixhawk y módulo GPS Ublox M8N

Una de las decisiones más determinantes tomadas a lo largo del TFG fue la elección de la controladora de vuelo. Como queda reflejado en el punto 2.3.3, la controladora de vuelo es un elemento vital en todo multicóptero, ya que es la que le proporciona la información sobre su situación

espacial y la que calcula a partir de los comandos del piloto las órdenes, que las diferentes ESC traducirán a sus respectivos motores como señales que determinarán su velocidad.

Existen varias empresas que fabrican controladoras de vuelo. Sus propiedades y características se esparcen en una gran escala de valores. Existen controladoras muy básicas con una capacidad mínima de estabilización y control, y controladoras programables con potentes procesadores y memoria que les permiten ser configuradas para fines muy específicos.

Tras un rápido vistazo al mercado, es abrumadora la cantidad de opciones que existen. Se definen en un principio las características imprescindibles que la controladora ha de tener:

- Giróscopos y acelerómetros que proporcionen una información precisa de la situación del *UAV*.
- Procesador potente capaz de realizar cálculos y correcciones de posición en tiempo real.
- Capaz de soportar un módulo GPS.
- Compatible con diferentes sensores.
- Posibilidad de modos autónomo y guiado.
- Capacidad de memorizar *waypoints.*
- Precio asequible.

Contrastando la oferta con nuestra lista de condiciones, destacan tres controladoras: la controladora "*Naza M V-2*" de *DJI* (Ilustración 3-1), la "*APM 2.5*" de "*APM Copter*" (Ilustración 3-2) y la "*Pixhawk*" de "*3DR*" (Ilustración 3-3).

Ilustración 3-1 Naza M V-2(Tomado de [25]).

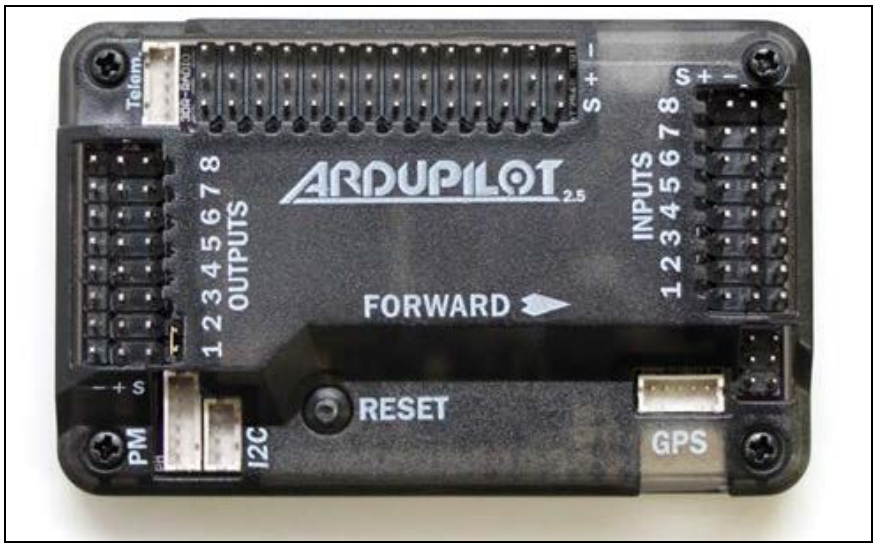

Ilustración 3-2 APM 2.5 (Tomado de [21]).

Ilustración 3-3 *Pixhawk* (Tomado de [29]).

Las controladoras *Naza* y *APM* son dispositivos ampliamente probados que han dado buenos resultados en proyectos similares. Soportan gran cantidad de modos de vuelo diferentes y existe mucha documentación de proyectos de desarrollo relacionados con ellas. Destacan también por una configuración simple y rápida. Sin embargo, ambas tienen problemas de compatibilidad con complementos que no pertenezcan a la casa. La controladora *Naza* solo acepta complementos de la empresa *DJI* y la *APM* solo acepta complementos de *Ardupilot*. Aunque por otro lado, se trata de dos grandes empresas muy consolidadas en el sector con una amplia gama de accesorios.

En la otra esquina se encuentra la *Pixhawk*, una controladora más moderna, con mayor memoria y un procesador más potente. Es una controladora menos probada que los anteriores, pero con unos resultados muy prometedores. Además sigue una filosofía similar a *Linux* o *Android*. Es de código abierto, lo que permite que desarrolladores publiquen todo tipo de funciones, y permite crear funciones propias que se ajusten a las necesidades del programa. También permite la compatibilidad con sensores de diferentes empresas.

Finalmente se decidió adquirir la controladora *Pixhawk*, por sus mejores prestaciones, su programabilidad y capacidad de desarrollo. La *Pixhawk* es una controladora de vuelo desarrollada por *3DR*, empresa que participa en el proyecto *Dronecode* desarrollado por la *Fundación Linux* [30]. Este proyecto busca desarrollar una plataforma de código abierto que aúne y divulgue proyectos de desarrollo de *UAV*. Esta plataforma proporciona numerosas herramientas e información para el

desarrollo de esta tecnología que resultarán de gran utilidad en el desarrollo del *UAV*. Además, la empresa 3DR contribuye a este programa con la creación de la "*API*" (*Aplication Programing interface*) "Dron*kit*", que proporciona un entorno de programación de alto nivel para drones con muchísimas posibilidades y del que se hablará más en detalle en apartados posteriores [29].

La controladora de vuelo *Pixhawk* cuenta con:

- Un procesador 168 MHz *Cortex M4F CPU*.
- 256 KB *RAM*, 2 MB *Flash*.
- Capacidad para incorporar una tarjeta "*SD*" para recolección de datos de telemetría
- Salida de control para hasta 16 servomotores.
- Una gran programabilidad, lo que facilitará el desarrollo del sistema de navegación.
- Giróscopo, magnetómetro y barómetro, que le permiten conocer su inclinación orientación y altura.

Además, la empresa *3DR* cuenta con el módulo compás+GPS "*Ublox M8N*", compatible con la controladora de vuelo *Pixhawk*. Este módulo combina una gran sensibilidad con un bajo consumo eléctrico, así como un reducido tamaño y peso. Pertenece a la categoría profesional de *Ublox* y proporciona una actualización de la situación con una frecuencia de 5HZ [31]. Teóricamente ofrece una precisión "*CEP*" del 50% (*Circular error probability*) de 2,5m de forma autónoma y 2m con el sistema con la ayuda del sistema "*SBAS*" (*Satellite Based Augmentation System)*. Esto quiere decir que el 50% de las situaciones tomadas durante 24 horas se encontraban en dentro de un círculo de 2,5 o 2 metros respectivamente centrados en la posición real GPS.

Ilustración 3-4 Módulo Compás+GPS *Ublox* N8M (Tomado de [32]).

Este módulo GPS es recomendado en numerosos foros de desarrollo de drones como el apropiado para las controladoras de vuelo *Pixhawk* y *APM*. Cuenta con una configuración preestablecida, que ha sido muy probada por miles de usuarios [33].

3.2.2 Raspberry Pi 2 Modelo B

El objetivo de instalar una *Raspberry* en el dron es el de que actúe como intermediario entre la estación de control y el vehículo. Con ello, se aumenta enormemente la capacidad del *UAV* de realizar procesos de forma autónoma, y su conectividad, además de abrir unas vías de desarrollo casi ilimitadas.

La *Raspberry Pi* es un ordenador de placa única de bajo coste que se desarrolló en Reino Unido por la "*Raspberry Pi Foundation"* (Ilustración 3-5 Raspberry Pi Foundation (Tomado de [34]) con objetivos didácticos. A pesar de que tiene apenas el tamaño de una tarjeta de crédito, la *Raspberry* se ha convertido en un recurso muy empleado tanto en la enseñanza como en el desarrollo de proyectos

que requieran procesadores de pequeño tamaño, como es este caso. Soporta diferentes sistemas operativos y puede realizar muchas de las funciones de un *PC*. No cuenta con pantalla ni teclado propios, pero sí se le pueden conectar a través de sus puertos *USB* y *HDMI*. Su gran versatilidad y su reducido coste han sido los factores determinantes por los que la *Raspberry* se haya convertido en el fenómeno en constante evolución que es hoy en día.

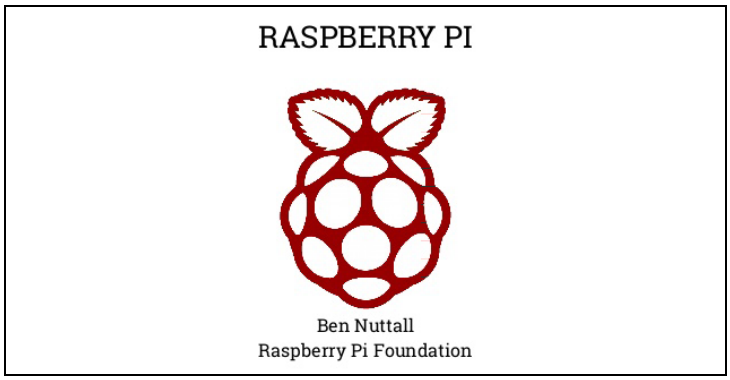

Ilustración 3-5 *Raspberry Pi* Foundation (Tomado de [34]).

El modelo de *Raspberry* que se ha adquirido para el proyecto es la *Raspberry Pi 2 Modelo B*, que incorpora una serie de mejoras con respecto a sus predecesoras. Las *Raspberry Pi 2* cuenta con:

- 1GB de memoria *RAM*.
- 40 pines "*GPIO*" (*General Purpose In/Out*)
- 4 puertos *USB2*
- Salida de audio estéreo
- *HDMI*
- Tarjeta Micro *SD*
- Alimentación por mini *USB*
- Puerto Ethernet

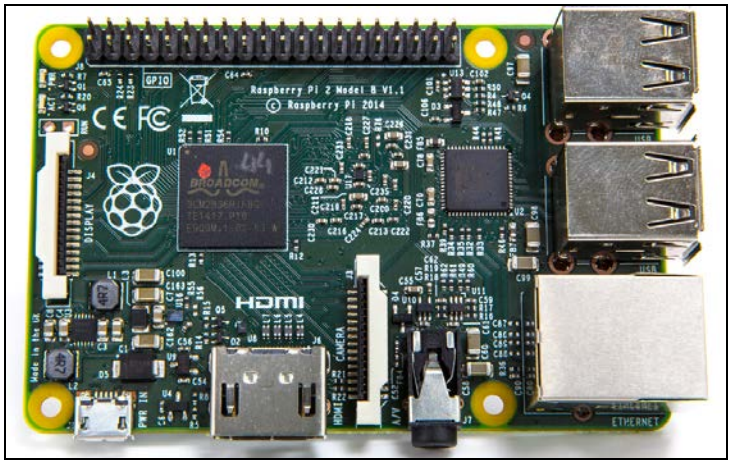

Ilustración 3-6 *Raspberry Pi* 2 Modelo B (Tomado de [34]).

Su pequeño tamaño, la capacidad de decisión y resolución de procesos, y la amplia conectividad que proporciona a través de sus números puertos y sus pines, han sido unas de las razones que ha decidido su incorporación al *UAV*.

3.2.3 Ubiquity Picostation MH2

La elección de la antena *"Ubiquity Picostation MH2"* fue una decisión coordinada con el equipo del TFG que va a llevar a cabo la instalación de una red *MANET* en las lanchas de instrucción de la

ENM. Teniendo en cuenta que uno de los objetivos de este proyecto es permitir la conectividad del *UAV* con dicha red, es lógico asumir que las antenas han de ser compatibles. Estas antenas han de ser capaces de soportar el protocolo *OLSR*, por el cual se comunicarán las redes internas de cada una de las lanchas, entre sí.

Con estas condiciones, se decidió que la antena *Picostation M2 HP* (Ilustración 3-7), de la empresa "*Ubiquity Networks*" [35] era la más apropiada para el proyecto. Se trata de una antena ligera y de pequeño tamaño, que a pesar de ello combina un gran alcance efectivo con un bajo consumo. Se trata de una antena "inteligente" con procesador propio. Además, pertenece a la misma casa que las antenas instaladas en las lanchas de instrucción, lo que facilita enormemente la configuración y la compatibilidad. Las características técnicas más importantes de cara al *UAV* son:

- Potencia de transmisión de hasta *28 dBm*
- Sensibilidad en recepción de hasta *-95 dBm*
- Alcance en torno a los 5 Km
- Peso 100 gr
- Frecuencia 2,4 GHz (*WIFI*)
- Consumo 4 W

Ilustración 3-7 Ubiquity Picostation M2 HP (Tomado de [35]).

Se trata por tanto de una antena potente, de gran alcance y sensibilidad, sin dejar por ello de ser ligera y de bajo consumo, aunando así las condiciones necesarias para su incorporación al *UAV*.

3.2.4 FrSky Taranis Plus X9D

A pesar de que este trabajo busca entre otros objetivos eliminar la necesidad de una emisora convencional, se consideró importante contar con una. Esta emisora se utilizaría en las etapas iniciales (hasta que el sistema de control estuviera operativo) y como método de control alternativo del *UAV* en caso de fallo del Sistema de Control

Partiendo de estas premisas, fue necesario definir qué clase de emisora adquirir. Se necesitaría una emisora multicanal, compatible con la controladora de vuelo *Pixhawk*, desde la que fuera posible llevar un control preciso del multicóptero. Además sería preferible que fuera una emisora que permitiera la configuración de varios modos de vuelo en los diferentes interruptores.

No obstante, el factor determinante fue la frecuencia de emisión. Hoy en día, la mayoría de los vehículos a radio control trabajan en la banda de 2,4 GHz. Dentro de esta frecuencia se crean una serie de canales, de forma que le receptor y el emisor "buscarán" uno que este libre, y lo utilizarán. En caso

de que se perdiera la conexión por alguna razón, la emisora volvería a hacer un barrido hasta encontrar el canal en el que esté trabajando el receptor. Esto permite que varios vehículos compartan este rango de frecuencias de forma automática.

Sin embargo, la red *MANET* trabaja de una forma similar y en el mismo rango de frecuencias. Por esto, la presencia de la antena *WIFI* a bordo del dron a tan poca distancia del receptor de radio control, trabajando ambos en una frecuencia parecida, provocaría interferencias mutuas que podrían comprometer la capacidad de control de ambos sistemas. Con la intención de conservar en todo momento la capacidad de control de la emisora como medida adicional de seguridad, se puso como condición que la emisora fuera capaz de trabajar en una frecuencia diferente a los 2,4 GHz.

Teniendo todo lo anterior en cuenta, se eligió la emisora "*FrSky Taranis"* (Ilustración 3-8), una emisora multicanal altamente programable con capacidad de instalación de módulos para diferentes frecuencias. La *Taranis* emite por defecto en 2,4 GHz, y viene de serie con un receptor "*X8R Combo*" para ese margen de frecuencias. Por lo tanto, se adquirieron a mayores unos módulos transmisor y receptor *EzUHF* de *433 MHz*, una frecuencia de uso común en aeromodelismo en Europa

Ilustración 3-8 Taranis FrSky (Tomado de [32]).

La emisora permitió probar los diferentes modos de vuelo y las capacidades de la plataforma antes de que el sistema de control estuviera desarrollado, y una vez desarrollado este supuso un elemento de seguridad de gran relevancia, especialmente durante las etapas iniciales del desarrollo de las aplicaciones de control.

3.2.5 Sonar Maxbotix, y Switch y Cámara Edimax

Se trata de tres elementos complementarios, que en caso de incorporarse podrían mejorar las capacidades del *UAV* en diferentes aspectos.

El sonar "*Maxbotix 12CXL*" [36] es un sonar altímetro de gran precisión compatible con la controladora de vuelo *Pixhawk*. Abarca un rango de hasta 6 metros con una resolución de 1 centímetro y tiene un haz muy direccional y está especializado en ambientes con mucho ruido. Podría mejorar el comportamiento a baja altura del *UAV* y facilitar las secuencias de aterrizaje.

Ilustración 3-9 Sonar Maxbotix 12CXL-EZ4 (Tomado de [36])

La cámara *IP* "*Edimax IC-3115W*" [37] es una cámara *IP* de diseño compacto con capacidad de grabación en *HD*. Dado que el *UAV* se comunicará a través de una red *WIFI*, el disponer de una cámara *IP* proporciona un sensor excelente, que permitire ver en tiempo real aquello que vea el *UAV*. Es una cámara compacta y de bajo consumo con gran calidad de video.

El *switch* "*Edimax ES-3305P*" [37] por otra parte, es una consecuencia de la instalación de la cámara. Dado que la antena *Ubiquity Picostation* consta de un único puerto de *Ethernet*, es necesaria la instalación de un *switch* que permita conectar a más dispositivos. El hecho de que el *switch* tenga 5 puertos proporciona capacidad de expansión de cara a la instalación de nuevos dispositivos que sean capaces de adquirir una *IP* propia dentro de la red y comunicarse a través de ella. Conociendo esta necesidad, se buscó un *switch* que se ajustara a las necesidades del proyecto en cuanto a tamaño y peso.

Ilustración 3-10 Cámara IP Edimax.

Ilustración 3-11 Switch de 5 puertos Edimax .

***Imágenes tomadas de* [37].**

A pesar de que estos dispositivos pueden aumentar las capacidades del *UAV*, ninguno de ellos es imprescindible para el desarrollo de las operaciones que se pretenden llevar a cabo. Dicho esto, no deja de ser interesante contar con ellos por las posibilidades de ampliación que ofrecen.

3.2.6 Kit DJI 550

Para decidir qué clase de multicóptero construir, fue necesario un estudio inicial de la "carga de pago". La carga de pago de un vehículo está compuesta por todos aquellos elementos que no sean indispensables para el funcionamiento de este. En este estudio se listó los distintos elementos y pesos que el multicóptero habría de incorporar según la Tabla 3-2.

Tabla 3-2 Carga de Pago

Elemento	**Peso**
Controladora de vuelo Pixhawk v.2.4.5	38 gr
Módulo GPS Ublox M8N para Pixhawk 2.4.5.	26 gr
Ubiquiti Picostation MH2	100 gr
Raspberry Pi 2 Modelo B	45 gr
Cámara IP Edimax IC-3115W	86 gr
Switch Edimax ES-33505P V3	82 gr
Maxbotix 12CXL-MaxSonar-EZ4 Sonar	6 gr
APM/Pixahawk Wireless Radio Module	17 gr
Carga de pago	**400 gr**

En esta tabla se realiza una aproximación inicial de la carga de pago que el multicóptero debería ser capaz de soportar. Cabe destacar que esta carga no incluye el peso propio del aparato en sí.

Esta aproximación inicial permite hacerse una idea de qué clase de máquina se debe seleccionar. Con un plazo mayor de trabajo hubiera sido posible diseñar un chasis desde cero porque no requieren formas aerodinámicas especialmente complicadas, y se hubieran adquirido por otra parte los motores y los ESC. Sin embargo la escasez de tiempo inclinó la balanza hacia módulos pre planeados de desarrollo ya dimensionados.

Tras investigar posibles opciones, se decidió que la más favorable era el "*Kit DJI 550"* [25]. El *DJI F550* pertenece a la serie "*Flame Wheel*" de *DJI*, una de las marcas más reconocidas en el campo de los drones. Esta empresa abarca toda clase de aparatos, desde los del estilo "*ready-to-fly*", plataformas cerradas, pre configuradas, y listas para utilizar desde el primer instante, hasta plataformas abiertas para *amateurs* y desarrolladores.

La serie *Flame Wheel* consta de tres modelos de multicópteros de diferentes tamaños, los cuadrocópteros *DJI350* y *DJI450*, y el hexacóptero DJI550 (el número hace referencia a los milímetros de diámetro). Estas plataformas son una solución muy útil para desarrolladores, ya que proporcionan una estructura básica sobre la cual empezar un proyecto individual. Tienen una amplia superficie sobre la que trabajar y colocar diferentes elementos, dividida en dos plataformas, con cableado integrado, lo que permite una distribución más organizada y menos susceptible a fallos. Además, los brazos construidos con materiales ligeros y de alta calidad, respaldada por una de las empresas más punteras en el terreno.

Dentro de esta serie, se eligió como idóneo el *DJI F550* (Ilustración 3-13), por su mayor capacidad de carga y espacio para montaje, de forma que pudiera equipársele con una amplia cantidad de

dispositivos de una forma cómoda y organizada. Por otra parte se consideró oportuno que se tratara de un hexacóptero, ya que aporta mayor estabilidad y un grado de seguridad añadido teniendo en cuenta que ante el posible fallo de uno de los motores, se podría mantener un mínimo de control sobre el vehículo como para aterrizarlo sin dañarlo.

Ilustración 3-12 Kit DJI F550 (Tomado de [25]).

Este módulo incorpora las siguientes piezas:

- Panel superior e inferior con cableado integrado (Ilustración 3-14). Conforman el cuerpo propiamente dicho del dron. Soportan la electrónica, la batería y los sensores. Cuentan con diferentes formas y orificios que facilitan la libre distribución y el ajuste de los elementos.

Ilustración 3-13 Paneles Superior (Izda) e inferior (Dcha) (Tomado de [25]).

- 6 brazos de *PA66+30GF*, un polímero ligero y resistente, (Ilustración 3-15), que componen la estructura. Este material aporta una gran resistencia con un bajo peso y es lo suficientemente flexible como para absorber parte de un hipotético impacto, evitando daños graves en la electrónica. Mientras 4 brazos son blancos, dos son rojos, proporcionando una referencia de proa. En el extremo de cada uno de ellos ira fijado un motor, con su ESC individual fijada en el propio brazo.

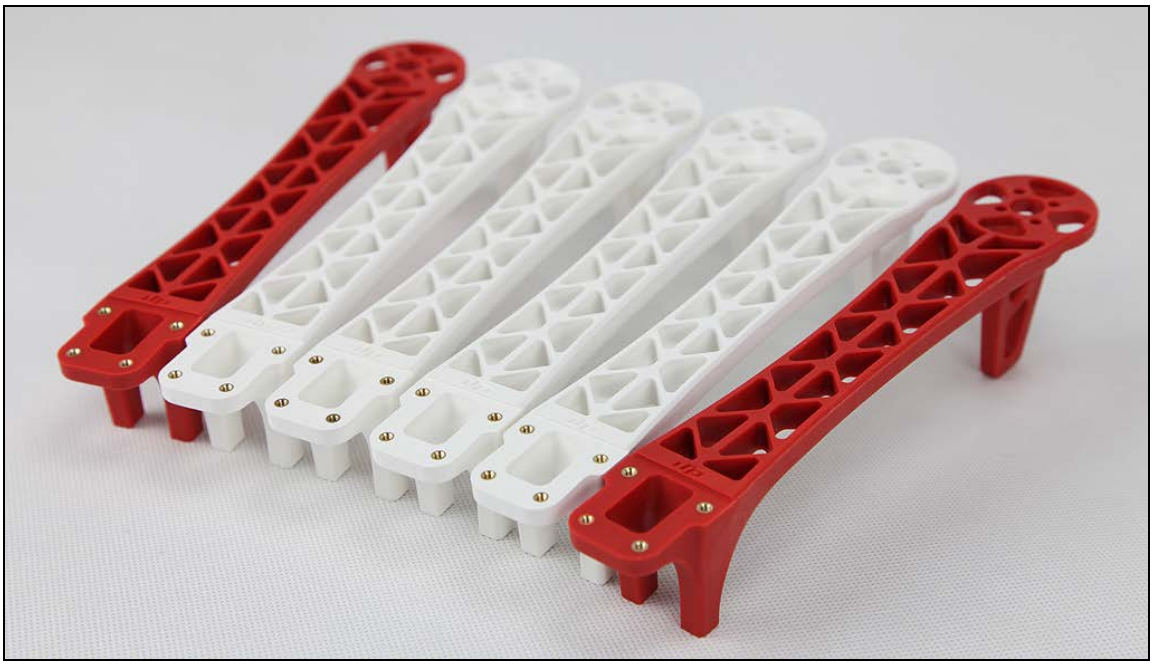

Ilustración 3-14 Brazos DJI FR550 (Tomado de [25]).

- 6 motores *2312E* de 960 KW, 3 de ellos de sentido horario y tres de sentido anti horario. A pesar de ser motores trifásicos sin escobillas cuyo sentido puede invertirse alterando las fases, el sentido de giro hace referencia al sentido de la rosca que une el eje del motor a la hélice, de forma la reacción de la hélice en el eje no provoque que esta se desenrosque. Según la empresa, cada uno de estos motores es capaz de generar hasta 710gr de empuje con la hélice adecuada.

Ilustración 3-15 Motores 2312E 960Kw DJI5 FR550 (Tomado de [25]).

- 6 ESC con una capacidad máxima de 30 A.

Ilustración 3-16 ESC DJI 420 Lite (Tomado de [25]).

- 8 hélices bipala 9.4x5, cuatro de ellas de rosca horaria, y cuatro de ellas de rosca anti horaria, lo que quiere decir que hay un repuesto para cada hélice.

Ilustración 3-17 Hélices 9.4x5 horaria y anti horaria DJI 550 (Tomado de [25]).

El "*kit*" incorpora además tornillos *Allen* propios y una herramienta para colocar las hélices. El ensamblado es relativamente fácil. Las piezas están especialmente diseñadas para encajar, y el único paso que requiere quizás mayor atención es la soldadura entre las ESC y la entrada de alimentación, con el panel inferior.

Cabe destacar que la elección de la plataforma condiciona en gran medida la elección de la batería. Cada motor tiene asociadas unas baterías recomendadas. En este caso, el fabricante recomienda las baterías del tipo *Lipo* de 3S del entorno de los 6000mA/h.

3.2.7 Batería Desire Power V8 Series 3S 6000mA/h 30C

Los motores eléctricos que emplea el aparato que se va a construir requerirán una fuente de alimentación de gran capacidad. El fabricante recomienda baterías de *3S* de 6000mAh, es decir, tres celdas en serie, lo que proporciona 11.1V.

La batería tiene previsto ser la única fuente de energía del aparato, tanto como para los rotores, como para toda la electrónica que éste ha de soportar. Mientras que los motores se conectan de forma directa a los 11.1V que les proporciona la batería, la mayoría de los equipos eléctricos no, por lo que serán necesarias etapas reductoras y elevadoras de tensión para alimentar cada aparato de la forma adecuada.

La duración de la batería se verá gravemente afectada por las diferentes configuraciones del aparato, que variarán su peso y su consumo. También variará según el perfil de vuelo que se realice, por lo que no es sencillo hacer una previsión del tiempo de autonomía del *UAV* basándonos únicamente en la batería que incorpora.

Dentro de la amplia gama de fabricantes de baterías *Lipo*, teniendo en cuenta los posibles peligros que puede acarrear su uso, se creyó conveniente adquirir baterías previamente recomendadas, con unas mínimas garantías de seguridad. Basándose en foros de desarrolladores de *UAV* se decidió adquirir baterías "*Desire Power V8 Series*" [32], por su diseño compacto y calidad.

Ilustración 3-183.2.7 Batería Desire Power V8 Series 3S 6000mA/h 30C (Tomado de [32]).

Junto con las 4 baterías, se adquirió también un cargador Graupner Ultramat que permite la carga segura de la batería y el equilibrado del voltaje entre las diferentes celdas.

3.3 Cálculo de pesos y autonomía

Una vez definidos los elementos que irán a bordo del *UAV* y la plataforma que lo compondrá, es lícito hacer un cálculo más detallado de los pesos. En aeronaves de pala rotatoria como son los multicópteros, la carga de pago es uno de los factores que más limitan la autonomía. Partiendo de una autonomía ya reducida, un dimensionado incorrecto de los elementos que debe incorporar el multicóptero puede reducir drásticamente su tiempo de vuelo.

En la Tabla 3-3, podemos observar un desglose del peso total del módulo.

Tabla 3-3 Peso Módulo DJI 550 (Datos tomados de [25]).

Elemento	Peso
Estructura (Brazos x6+Paneles x2)	478 gr
ESC (x6)	162 gr
Motores (x6)	342 gr
Hélices (x6)	78 gr
Total DJI 550	1060gr

Una de las principales ventajas de este proyecto es su modularidad. Es sencillo retirar o añadir elementos para conseguir una configuración más adecuada a unas circunstancias concretas.

Esta previsión se ha realizado para la configuración estándar, en la que el *UAV* incorpora todos sus sensores y una única batería. Sumando el peso total del módulo (Tabla 3-3), la carga de pago (Tabla 3-2) y el peso de la batería se obtendrá una buena previsión del peso total del *UAV* para esta

configuración. A esta previsión se le añade un margen de 100gr para cableado y elementos adicionales de fijación.

$$400gr\ (Carga\ de\ Pago) + 1060gr\ (DJI550) + 420gr\ (Batería) + \ 100gr\ (Cableado) = 1880\ gr$$

La pregunta ahora es ¿es capaz la plataforma de maniobrar con regularidad con esta carga? Los fabricantes y desarrolladores recomiendan que para que un multicóptero sea capaz de maniobrar ágilmente, su peso no ha de superar el 50% del empuje que le proporcionan sus hélices, es decir, la relación peso-empuje deberá ser de 2:1. Esto se traduce en que el dron deberá ser capaz de mantenerse en vuelo estático (“*hover*”) con la mitad de la potencia máxima. Aunque realmente sí debería ser capaz de soportar un peso mayor, esta regla garantiza que la maniobrabilidad y la autonomía del dron no se vean comprometidas. Sin embargo, para determinadas aplicaciones, es posible sacrificar parte de la maniobrabilidad y velocidad de respuesta del dron para aumentar la carga máxima.

Según el fabricante, teniendo en cuenta el motor, las hélices y la batería que se ha escogido, cada uno de los motores es capaz de proporcionar un empuje máximo de 710gr.

$$\frac{710gr}{motor} * 6\ motores = 4260gr$$

De forma que el mayor empuje que los 6 motores juntos son capaces de generar es de 4260gr. Si el límite para un funcionamiento adecuado es el 50% del empuje máximo:

$$4260gr * \frac{50}{100} = 2130\ gr$$
$$1980gr\ (1880 + 100) < 2130gr$$

Esto implica que el peso total del *UAV* no ha de superar los 2130 gr. Siendo el peso total de nuestro multicóptero de 1880gr, y añadiendo un margen de seguridad de 100 gr para cableado y elementos de fijación, se confirma que la plataforma escogida se ajusta correctamente a las necesidades del proyecto.

En la siguiente gráfica se representa la relación entre la carga de pago con la relación empuje-peso y el tiempo de vuelo del *UAV* diseñado:

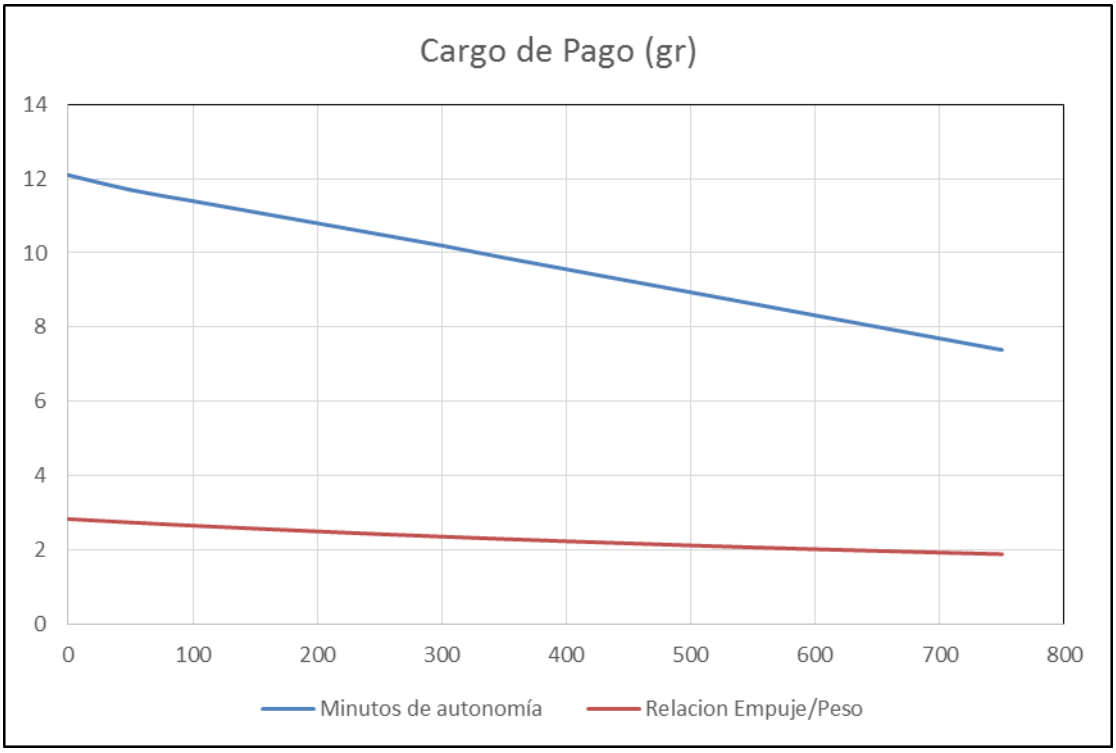

Ilustración 3-19 Relación Carga de Pago con Autonomía y Relación Empuje/Peso (Autoría Propia).

En cuanto al tiempo de vuelo, esta tabla no deja de ser una simplificación que representa el llamado "*tiempo de vuelo mixto*", tiempo que combina periodos estáticos y periodos en movimiento, pero permite formarse una idea del efecto que tienen unos pocos gramos en la autonomía del vehículo.

3.3.1 Software de cálculo ECALC

El software de cálculo "*ECALC*" es una herramienta "*online*" de dimensionado de *UAV*. Permite realizar una simulación de las características finales de una plataforma basándose en los elementos que la componen y las condiciones de vuelo previstas.

El apartado de introducción de datos cuenta con una amplia base de datos de piezas, con la cual es sencillo configurar una simulación del vehículo construido en este proyecto.

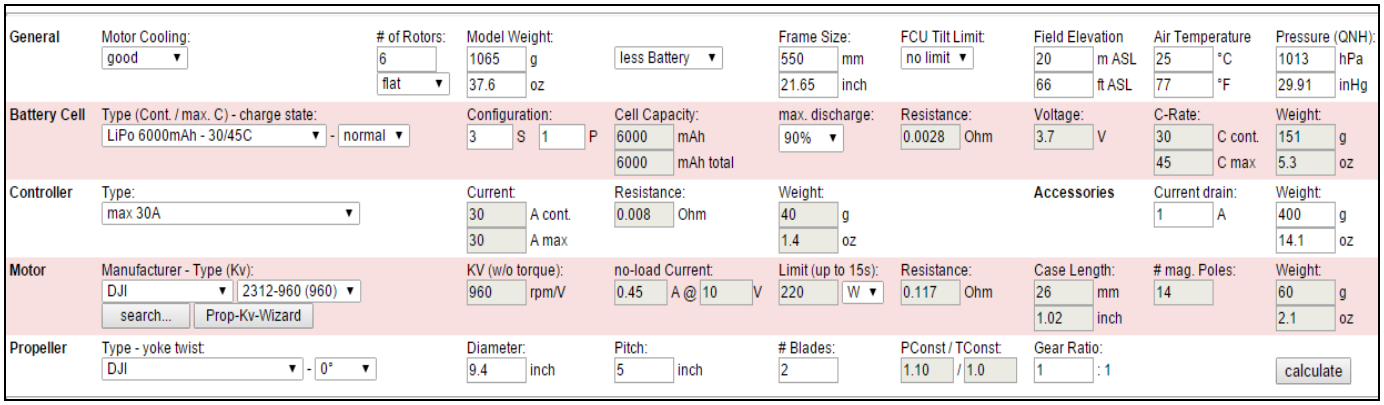

Ilustración 3-20 ECALC Introducción de datos (Tomado de [38]).

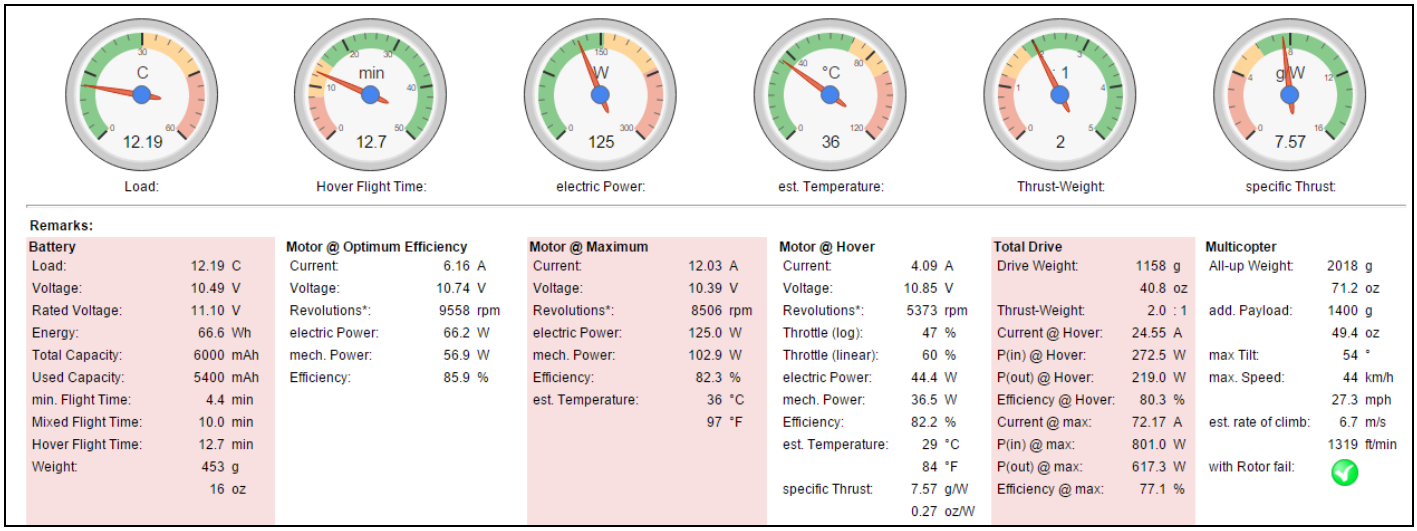

Ilustración 3-21 Resultado simulación ECALC (Tomado de [38]).

De los resultados obtenidos se extraen los datos más relevantes:

Tabla 3-4 Resultados Simulación ECALC.

Multicóptero		**Autonomía**	
Peso total	2018 gr	**Tiempo min. de vuelo**	5 minutos
Empuje/peso	2:1	**Tiempo de vuelo mixto**	10 minutos
Vel. máxima	44 Km/h	**Tiempo de vuelo estático**	13 minutos
Max. ángulo de guiñada	54º	**Batería**	
Vel. Vertical máxima	6.7 m/s	**Corriente descarga máxima**	72 A
Control con fallo de rotor	Bueno	**Potencia máxima**	800 W

De la información obtenida mediante la simulación se extraen las siguientes conclusiones sobre la plataforma diseñada:

- El software confirma que la relación de empuje/peso se encuentra dentro de los márgenes recomendables garantizando la maniobrabilidad del vehículo.
- En caso de fallo de un rotor, se debería poder conservar el control del aparato.
- El voltaje y la capacidad de descarga de la batería está correctamente dimensionados y permiten un funcionamiento correcto.
- Como se había previsto, el principal problema es una autonomía limitada de 10 minutos.

3.4 Fase 1: Montaje y configuración inicial de la plataforma

Esta fase describe el proceso de montaje y configuración del multicóptero sobre el cual habría de desarrollarse el Trabajo de Fin de Grado. El objetivo de esta fase inicial era el de disponer de una plataforma voladora estable sobre la que el desarrollo del sistema de control fuera viable. La primera fase incluye el montaje del multicóptero, la configuración de la controladora de vuelo y el cableado básico que permitirían volar a la plataforma. Cabe destacar también, que todo el proceso aquí recogido se realizó por duplicado, ya que se montaron dos multicópteros ante posibles eventualidades.

3.4.1 Montaje del multicóptero

Una vez adquiridos los dispositivos necesarios se procede a la construcción de la plataforma.

El *kit DJI550* está diseñado para montarse de una forma sencilla, mediante el siguiente proceso (Ilustración 3-24):

- Se atornillan los brazos al panel central superior, con la ayuda de una llave *Allen*.
- En el panel central inferior se sueldan las tomas de corriente de las ESC, a través de las cuales se alimentará a los diferentes motores. Como se comentó anteriormente, este panel cuenta con un cableado incorporado, lo que permite soldar la ESC directamente, así como la toma a la que se conectará la batería (Ilustración 3-23).

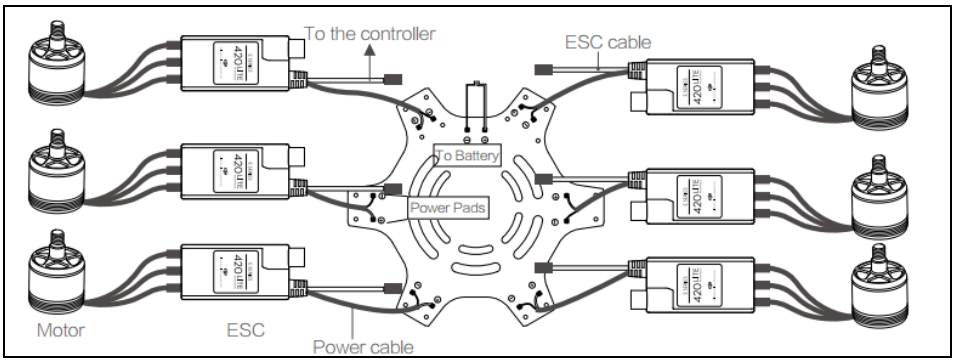

Ilustración 3-22 Montaje ESC (Tomado de [25]).

- Una vez soldadas las tomas de corriente, se atornillan los brazos al panel inferior.
- Después se atornillan los motores en los sitios diseñados para ello, teniendo en cuenta que tienen que alternarse motores de giro horario y motores de giro anti horario.
- Los motores tienen 3 salidas de las tres señales de corriente trifásica que se conectan directamente a las ESC.
- Cada una de las hélices (teniendo en cuenta su sentido de giro) se fija al eje de uno de los motores.
- Finalmente se instalan la patas, adquiridas por separado que proporcionan un margen con respecto al suelo, permitiendo instalar las baterías y facilitando el aterrizaje.

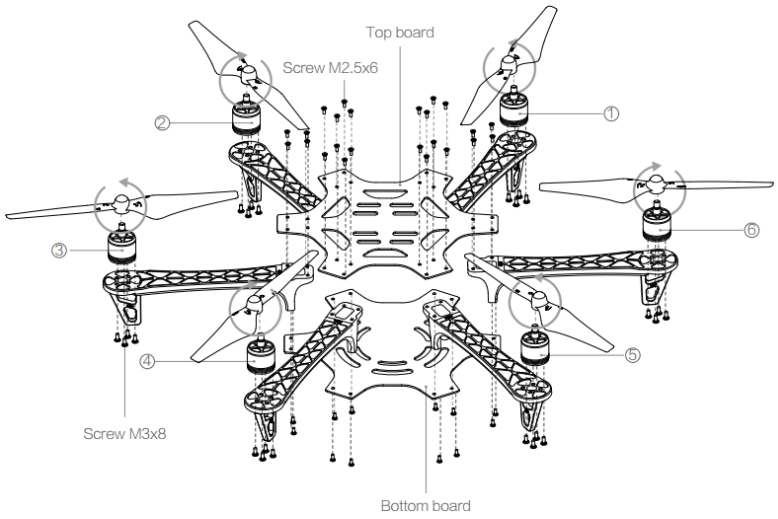

Ilustración 3-23 Esquema de montaje del DJI F550 (Tomado de [25]).

Terminado este montaje inicial obtenemos la estructura sobre la que habrá de desarrollarse el proyecto (Ilustración 3-25). Las tomas de corriente ya están ligadas a la toma de la batería y los únicos elementos que los separan de volar son la controladora de vuelo, un emisor y un receptor.

Ilustración 3-24 DJI F550 Montaje final. Laboratorio de electrónica.

3.4.2 Configuración inicial de la Pixhawk

El acceso a una controladora de vuelo y sus parámetros se realiza a través de una *Ground Control Station*. Algunas de ellas, proporcionan un interfaz útil sobre el que llevar a cabo la configuración y calibración de los sensores de la controladora.

Para la configuración de la *Pixhawk* se decidió utilizar la *GCS Mission Planner*, por ser un programa gratuito de fácil descarga y uno de los más documentados en este ámbito.

Ilustración 3-25 *Mission Planner* (Tomado de [21]).

Estos programas, además de servir de interfaz de control de vehículos a radiocontrol, proporcionan herramientas para acceder a sus parámetros básicos y configuración. En el caso de la *Pixhawk*, la conexión se puede realizar directamente desde el ordenador a través de un cable *USB*-mini*USB*, o a través de una conexión *WIFI* directa. El programa está diseñado de tal forma que reconozca en que puerto se encuentra la controladora de vuelo, y pulsando conectar, en la esquina superior derecha, la

conexión y entre la *GCS* y la controladora sea automática. Una vez esté configurada la *Pixhawk*, desde el momento en el que se realice esta conexión el programa creará en el ordenador una carpeta, en la que registra los parámetros de la controladora, y en caso de estar conectada durante una misión, recopilaría la información que fuera recibiendo sobre el estado de la plataforma (GPS, velocidad, altura…).

Ilustración 3-26 Mission Planner Main Page (Captura de pantalla).

Sin embargo, antes de conectarla, es preciso instalarle el "*firmware*" adecuado a la controladora. Éste dependerá de qué clase de vehículo se vaya utilizar. De ésta forma la *Pixhawk* sabrá con qué medios cuenta, y como deberían estar distribuidos

Haciendo clic en la ventana de "*INITIAL SETUP*" se accede al panel de configuración inicial, desde la que se podrá seleccionar el firmware correspondiente. En este caso el firmware es el "*ArduCopter V3.3-rc1 HEXA*" (Ilustración 3-28). Este firmware le dice a la controladora que está montada sobre un hexacóptero, con una distribución numeral de los motores normalizada. También le informa de que modos de vuelo acepta, y de qué parámetros consta, dado que la configuración no será ni remotamente parecida para un hexacóptero que para un "*rover*".

Ilustración 3-27 Instalación Firmware (Tomado de [39]).

Una vez el firmware se haya descargado e instalado, el programa pedirá desenchufar y volver a enchufar la controladora. Hecho esto, ya se puede conectar pulsando en el icono de la esquina superior

derecha (Ilustración 3-27). Durante esta conexión, Mission Planner registrará todos los parámetros del vehículo. Sin embargo, antes de poder utilizar la *Pixhawk*, será necesaria una calibración inicial de los sensores que incorpora. Para ello se volverá a acceder a la ventana de *INITIAL SETUP*, y paso a paso, calibrando el "*hardware*" básico y obligatorio, que consta de:

- "*Frame Selection*": es un apartado obligatorio para multicópteros, que vuelve a pedir información acerca de que distribución de propulsores. Se seleccionará "Hexacóptero X", que quiere decir que la plataforma consta de 6 rotores distribuidos en 30º, 90º, 150º, 210º, 270º y 330º, siendo el origen la proa del vehículo.

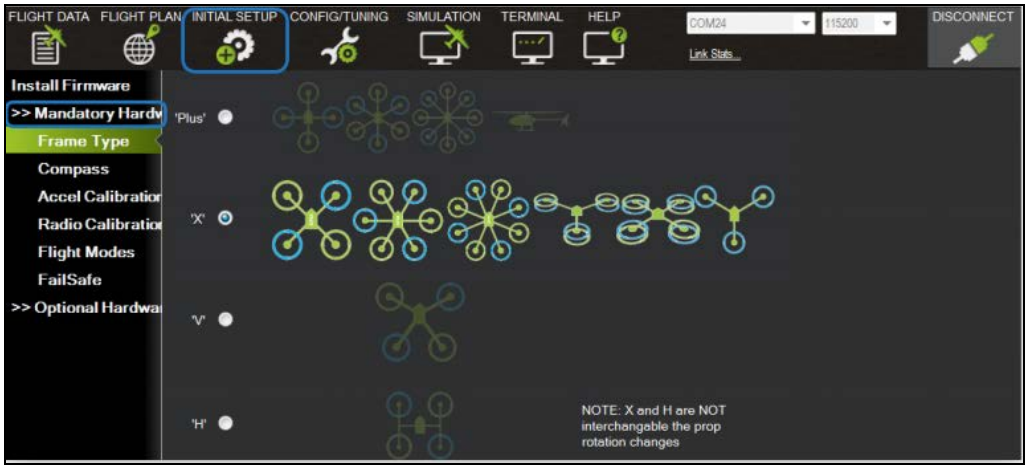

Ilustración 3-28 Frame Selection (Captura de Pantalla).

- "*Compass Calibration*": en este apartado se calibra el giróscopo que proporciona la referencia de proa a la controladora de vuelo. La *Pixhawk* no tiene que estar necesariamente colocada a crujía del vehículo y apuntando a la proa. Se puede configurar de cualquier forma, pero se consideró más sencillo y visual que así fuera, por lo que se procedió a la configuración estándar. En esta fase el programa pedirá que se gire la controladora en los tres ejes, e irá tomando una constelación de puntos que le permitirán formarse una imagen la orientación de su proa. Cuando considere que ha obtenido suficientes puntos se detendrá automáticamente.

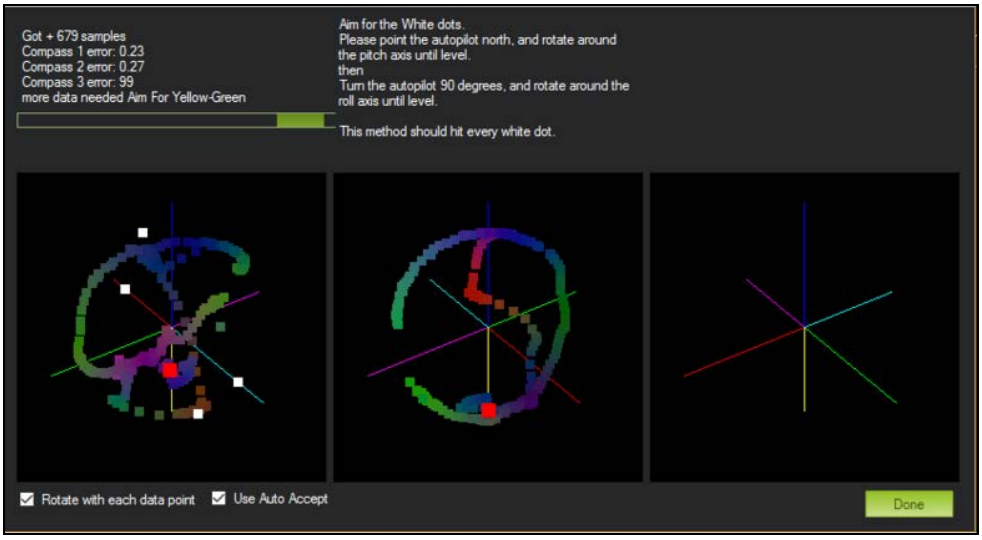

Ilustración 3-29 Compass Calibration (Captura de pantalla).

- "*Accelerometer Calibration*": En este apartado se calibran los acelerómetros de la controladora para que ésta reconozca sus velocidades en los diferentes ejes. Para ello el programa irá pidiendo que la controladora se coloque en determinadas posiciones: horizontal, del revés, apoyada sobre los costados etc.
- "*Radio Calibration*": en este apartado se calibran las señales que la controladora recibe del emisor. Para ello, la *Pixhawk* deberá tener conectado un módulo receptor de señal. Esta calibración busca conocer el máximo recorrido de las palancas para proporcionar una variación de la potencia acorde, así como las diferentes posiciones de los interruptores, de forma que se les pueda asignar funciones a cada uno.

Ilustración 3-30 *Radio Calibration* (Captura de Pantalla).

- "*Flight Modes*": en este apartado se la asigna un modo de vuelo a una posición determinada de los interruptores del mando, aunque pueden cambiarse accediendo de nuevo a la configuración. Como se comentó anteriormente (véase apartado 2.2.2), los modos de vuelo definen comportamientos automáticos de la aeronave, como mantener altura, posición GPS, modo automático para realizar misiones, etc. Muchos modos necesitan obligatoriamente un módulo GPS para ejecutarse. El único modo que es obligatorio es el modo *STABILIZE*, o estabilizar. Es el modo más básico y simplemente mantiene el multicóptero estable. Fruto de un breve estudio, se decidió que los modos más útiles habrían de ser *AUTO*, *RTL*, *LAND* y *LOITER*. Como todavía no se ha incorporado el módulo GPS, los modos *STABILIZE* y *LAND* serán los únicos que se podrán utilizar a corto plazo.

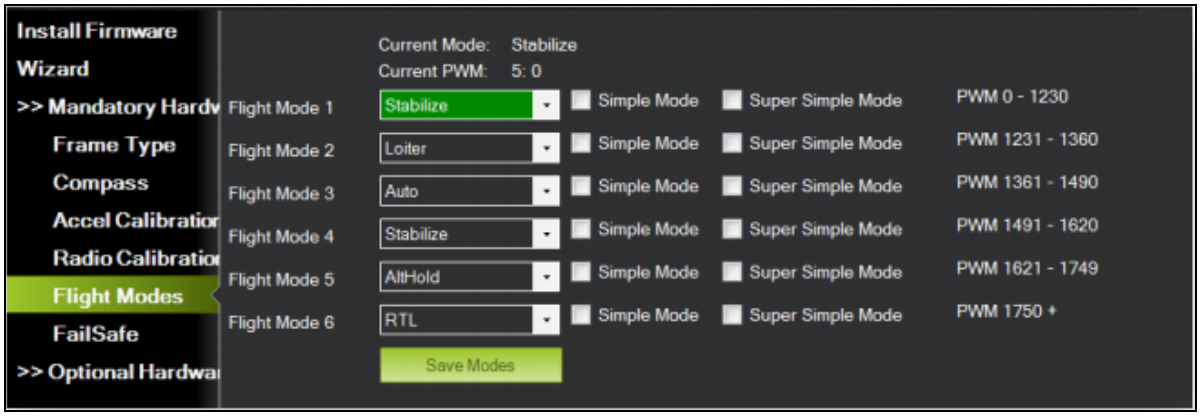

Ilustración 3-31 *Flight Modes Selection Page* (Captura de Pantalla).

Con esto, queda completada la configuración y calibración básica de la *Pixhawk*. Realizados estos pasos, ya se puede proceder a conectar la *Pixhawk* a la plataforma.

3.4.3 Cableado Pixhawk

La *Pixhawk* tiene la capacidad de incorporar una amplia gama de dispositivos a los que proporciona alimentación e intercambia información. Aunque pueda parecer trivial, el correcto cableado de la controladora merece ser descrito por las posibilidades que ofrece y las posibles consecuencias negativas que un error puede tener. Dicho esto, la *Pixhawk* incorpora de serie dos piezas de *hardware* adicionales:

- El "*Safety Switch*": un pequeño interruptor con un "*led*" rojo que permite bloquear la controladora frente a una orden de armado. Este interruptor será útil como una medida de seguridad adicional cuando se quiera manipular la plataforma estando está encendida.
- El "*Buzzer*": un pequeño zumbador que emite una serie de tonos que proporcionan información al usuario sobre las operaciones internas que está realizando la controladora.

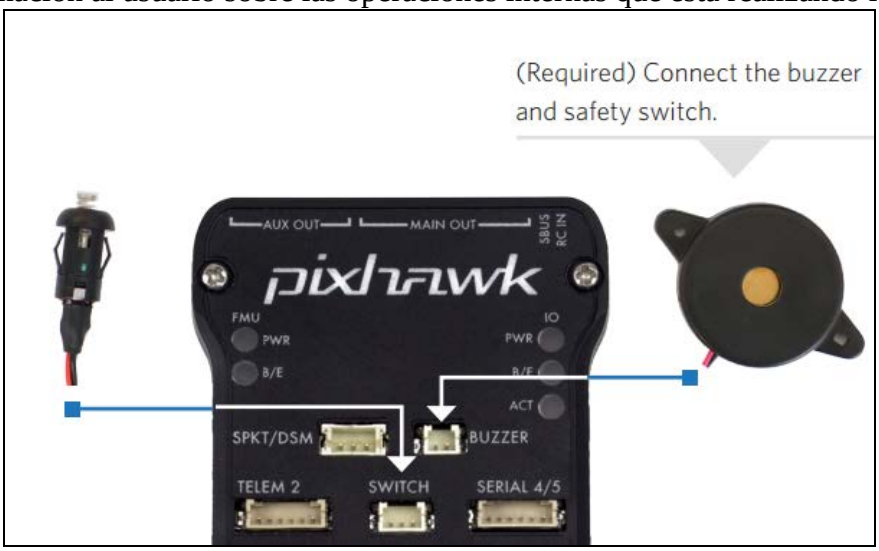

Ilustración 3-32 Conexión Buzzer y Safety *Switch* (Tomado de [39]).

Estas dos piezas son imprescindibles, y la *Pixhawk* no se podrá armar sin estar conectados a sus respectivas entradas (Ilustración 3-33).

El módulo receptor y las conexiones con la *ESC* se realizan en los pines de la parte superior de la *Pixhawk* (Ilustración 3-34)

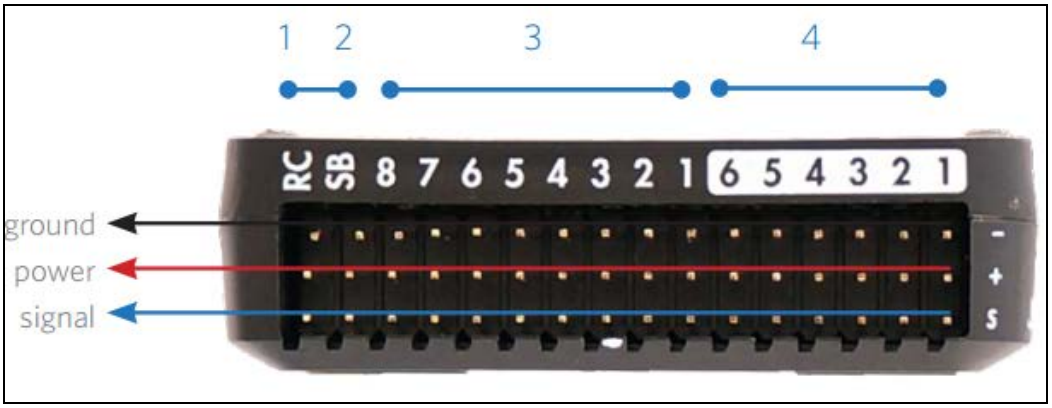

Ilustración 3-33 Pines *Pixhawk* (Tomado de [39]).

El módulo receptor se conectará en los pines "*RC*" (*Radio Control*), y las salidas de señal para las *ESC* en los pines del 1 al 6 del apartado 3, correspondiente a los "*Main outputs*". Por el *firmware* seleccionado, cada uno de los motores está numerado según la distribución de la Ilustración 3-35.

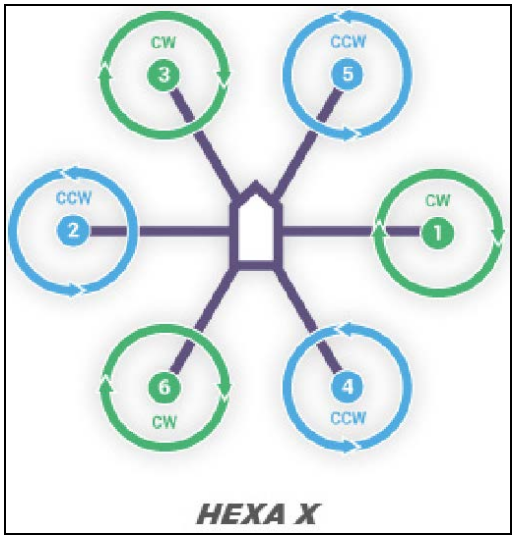

Ilustración 3-34 Distribución motores Hexacóptero (Tomado de [21]).

Es muy importante respetar esta configuración, ya que es la que espera la controladora. Por ello, las correcciones que la controladora realice en las velocidades de sus motores se realizarán de acuerdo a este esquema. Si los motores a los que ordena acelerar o decelerar no se encuentran en el lugar esperado, esto desestabilizará a la plataforma y le será imposible volar.

Finalmente, la *Pixhawk* ofrece múltiples posibilidades de alimentación. Para funcionar adecuadamente la *Pixhawk* necesita una alimentación estable de 5V. Esta se puede proporcionar a través de cualquiera de los pines de la parte superior, a través de la entrada mini-*USB* o a través de la entrada "Power" (Ilustración 3-36), especialmente diseñada a tal efecto. La ventaja que supone la alimentación a través de la entrada "*Power*", es que si se alimenta desde la batería principal del vehículo, y se regula el voltaje de entrada a través del "*Power Module*" de *3DR* (de los 11,1V de la batería a los 5V de alimentación), éste incorpora a los datos de telemetría de la controladora, información sobre el estado de la batería y asegura su correcta alimentación. Esta información es útil para conocer la autonomía restante en un momento determinado, así como para hacer un análisis del consumo, que quedará registrado. La alimentación se puede realizar de forma simultánea a través de varias entradas, siendo la principal la entrada "*Power*" y secundarias las demás. Esto proporciona una seguridad adicional ante un fallo en la alimentación, que de otra forma podría resultar fatal. Sin embargo para las pruebas iniciales, a falta del "*Power Module*", la controladora se alimentó a través de una fuente auxiliar de 5V a través de los pines de la parte inferior.

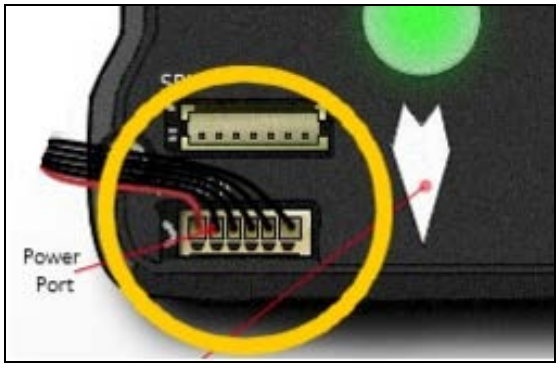

Ilustración 3-35 Puerto de alimentación *Pixhawk* (Tomado de [33]).

Si se han realizado todas las conexiones de la forma adecuada, la controladora de vuelo estará correctamente alimentada, será capaz de recibir señal del transmisor y podrá mandar órdenes a las ESC que regulan la velocidad de los rotores, por lo que la plataforma estará lista para un primer vuelo. Para éste la controladora se fijará en el hexacóptero a crujía y con la flecha apuntando a la proa del vehículo. Para fijarla, la controladora incorpora una espuma y unas tiras de velcro adhesivas, esta fijación "blanda" impide que las vibraciones de la plataforma se propaguen directamente a la controladora, ya que podrían resultar dañinas.

La *Pixhawk* cuenta además con un *led* multicolor proporciona información acerca de los procesos que está llevando a cabo la controladora según la Ilustración 3-37.

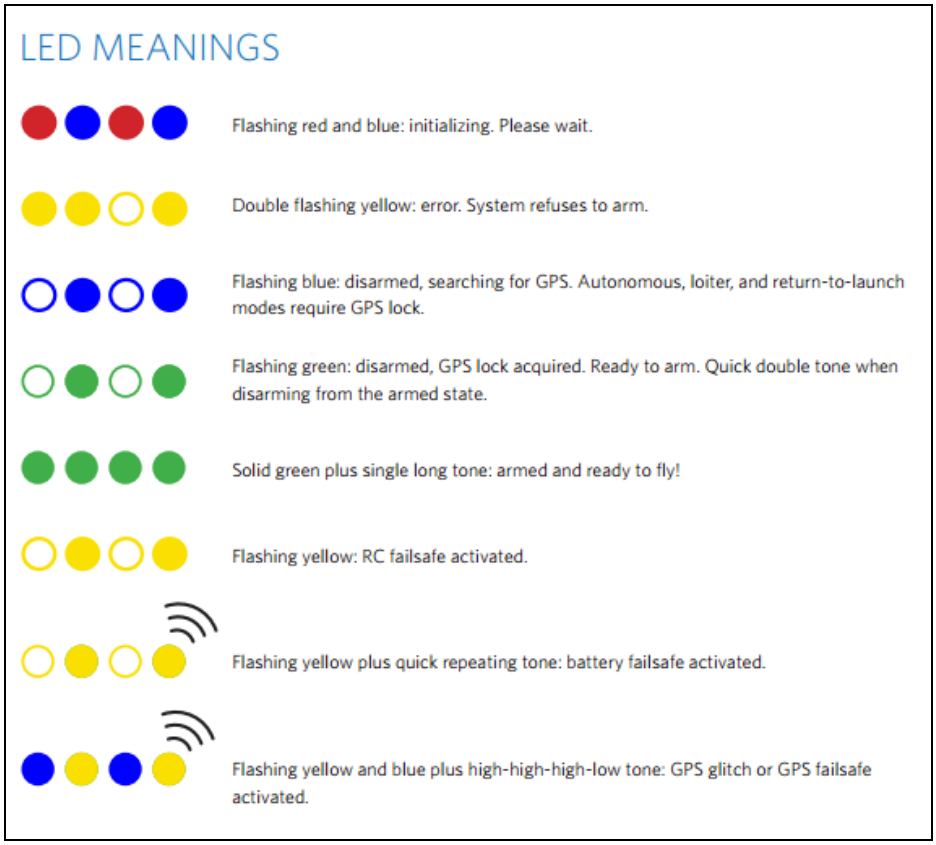

Ilustración 3-36 Códigos LED de la *Pixhawk* (Tomado de [39]).

3.4.4 Primer vuelo

Este primer vuelo fue un hito importante que supuso el final de la "Fase-1" que, aunque no era un objetivo del trabajo, fue un paso imprescindible en su desarrollo. Este primer vuelo se llevó a cabo el 19 de Enero de 2016 (Ilustración 3-38).

Ilustración 3-37 Primera Prueba de Vuelo 19/01/2016.

Para realizar las pruebas de vuelo a lo largo del proyecto se eligió como escenario la zona de la pista de obstáculos de la ENM (Ilustración 3-39), por ser un espacio libre, poco transitado y cercano al laboratorio.

Ilustración 3-38 Zona de pruebas de vuelo ENM (Captura de Pantalla).

En estas pruebas se corrigió un error de cableado de uno de los rotores que provocaba que éste girara en sentido contrario, ocasionando daños en una de las hélices que tuvo que ser sustituida por una de repuesto. Una vez solucionado este inconveniente se concluyó que:

- Los rotores estaban bien alimentados y respondían de forma adecuada a las órdenes de la controladora.
- La plataforma respondía bien a las órdenes enviadas a través del transmisor.
- La controladora de vuelo estaba correctamente configurada.

- El modo *STABILIZE* de la controladora funcionaba correctamente, proporcionando un vuelo estable.
- El aparato era muy sensible a los mandos y requería cierta práctica en el pilotaje de multicópteros el controlarlo con seguridad.
- En un vuelo manual, la referencia de proa es vital para el control del aparato, y es fácil perderla, por lo que las siguientes prácticas se limitarían a vuelos diurnos.
- A falta de una mayor carga de pago el vehículo tenía una autonomía considerable en una configuración básica.

En definitiva, los resultados obtenidos en esta primera prueba de vuelo permitieron confirmar que la plataforma construida era apta para el desarrollo del proyecto.

3.5 Fase 2: Vehículo autónomo

Esta segunda fase se adentra más en los objetivos propios del TFG. El objetivo de esta fase es ser capaces de dotar a la plataforma construida en la anterior fase de los elementos necesarios para realizar vuelos autónomos simples basados en situaciones GPS. Esta fase incluye la instalación y configuración del módulo GPS *Ublox N8M*, el estudio de la situación que proporciona y las pruebas de los modos de vuelo basados en situaciones GPS.

3.5.1 Instalación del Módulo GPS Ublox N8M

Como se describe en el apartado de elementos, se adquirió para la plataforma un módulo Compás+GPS de la empresa *Ublox*, por su reducido tamaño y peso y compatibilidad con la controladora de vuelo *Pixhawk*. Este módulo clasificado como de uso profesional por el fabricante, proporciona situación GPS así como una segunda referencia de dirección. También proporciona una información relativamente precisa de la altura del vehículo a través del uso de varios satélites. La configuración de fábrica ha sido probada numerosas veces y es la recomendada por desarrolladores [33]. Tanto la controladora de vuelo como el módulo GPS admiten la posibilidad de conectar dos módulos GPS. Si esto llegara a llevarse a cabo, la configuración preestablecida declararía dominante al GPS que recibiera señal de un mayor número de satélites.

Si no se desea alterar la configuración, la instalación es sencilla. No hay más que conectar las dos salidas del módulo a sus entradas correspondientes. La entrada de GPS, y la entrada *12C* para un compás auxiliar según la Ilustración 3-40. La alimentación del módulo se realiza a través de estas mismas conexiones.

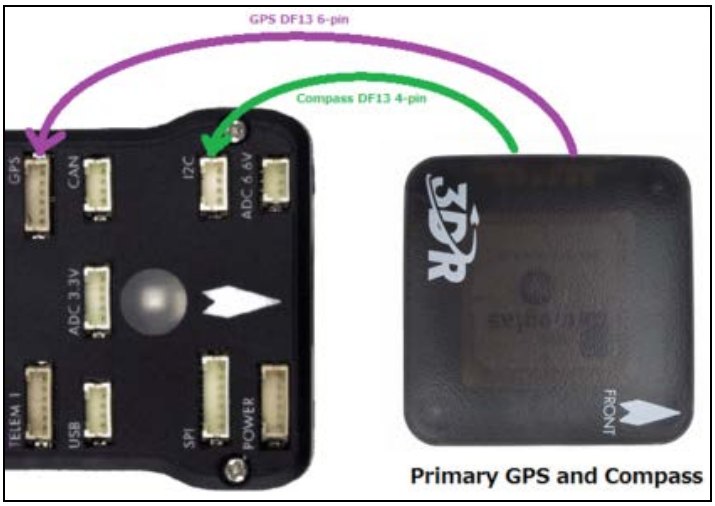

Ilustración 3-39 Conexión Módulo GPS+Compás (Tomado de [33]).

De la misma forma que la controladora de vuelo, el módulo GPS no tiene por qué estar orientado de la misma forma que el vehículo, y se puede configurar para que corrija la proa, pero al igual que con la controladora, el tener la flecha apuntando a la proa proporciona una referencia visual útil para el montaje.

Una de la recomendaciones de seguridad que se dan a lo largo del montaje del módulo GPS, es que éste se encuentre apartado de la plataforma, de manera que tenga una mejor visibilidad y se mantenga apartado de los campos magnéticos generados por el cableado y los rotores. Esta es la razón por la que el módulo incorpora un mástil sobre el que colocarse. Durante las pruebas iniciales del GPS en las que aún estaba pendiente la instalación del mástil y el GPS se instaló de forma temporal en la propia estructura, se corroboró que en el momento en el que se arrancaba la maquinaria, la situación GPS dejaba de ser fiable y dar saltos aleatorios.

Se comprobó también que el GPS necesita encontrarse en un espacio abierto cuando se enciende y que tiene un intervalo variable de tiempo hasta que obtiene su situación GPS por primera vez, que puede llegar hasta varios minutos. Una vez la controladora ha adquirido posición GPS el *led* de aviso pasará a luz verde parpadeante, según la Ilustración 3-37. El propio módulo GPS cuenta también con un *led* de aviso que parpadea en color verde mientras está buscando situación y pasa a verde continuo una vez la ha adquirido.

3.5.1.1 Estudio de la situación GPS proporcionada por el módulo *Ublox* M8N

En este apartado se busca realizar un análisis de la situación GPS proporcionada por el módulo. Con este estudio se pretende corroborar la precisión del GPS descrita por la información del producto.

Como figura en el apartado de elementos, este módulo GPS cuenta con una precisión *CEP* teórica de 2 metros, contando con la corrección "*SBAS"* (*Satellite based augmentation system*), un sistema de corrección de errores GPS por satélite dividido en diferentes zonas de cobertura. España se encuentra dentro de la cobertura "*EGNOS*" (*European Geostationary Navigation Overlay Service*), que se encarga de proporcionar la corrección *SBAS* dentro de Europa.

Este estudio trata de reproducir a una menor escala una prueba de precisión de GPS. Para ello se colocará el GPS en una posición concreta durante un periodo de tiempo de una hora y se registraran las diferentes situaciones del mismo punto que genere el GPS. Para realizar esta prueba se partirá de la suposición de que la situación GPS media es la verdadera, a falta de una situación GPS perfectamente conocida. Esta suposición implica que el GPS tiene el mismo error en todas las direcciones, lo que no tiene porqué ser cierto, pero se considerará un error asumible.

Para registrar las situaciones GPS se utilizarán las herramientas que nos proporciona el *GCS Mission Planner*. Como se comentó anteriormente, en el momento en el que se realiza la conexión con este programa, este genera un archivo "*.tlog*" o log de telemetría, en el que se almacena la información que recibe de la controladora de vuelo durante el periodo de conexión. Estos logues son almacenados en la memoria interna de la controladora de vuelo y se pueden descargar a través del *GCS*. *Mission Planner* cuenta también con herramientas de análisis de logues, que permiten reproducir un vuelo del que se conservan los datos. Sin embargo las herramientas que nos proporciona, aunque útiles, nos aplicables a este caso.

Accediendo a la configuración de *Mission Planner* podemos regular la frecuencia con la que los logues graban determinados datos. De cara a este estudio interesa aumentar la frecuencia de grabación de situaciones GPS al máximo.

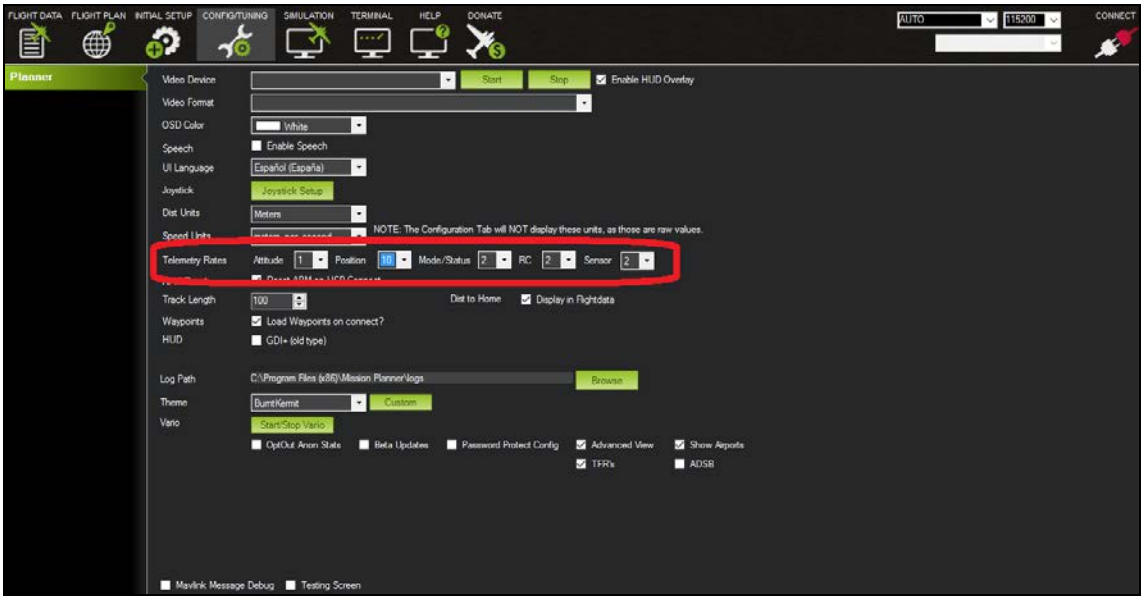

Ilustración 3-40 Configuración. Frecuencia de toma de datos de telemetría (Captura de Pantalla).

A falta de otra conexión, la prueba se tuvo que realizar con la controladora de vuelo conectada a un ordenador portátil a través de *USB*. Se fijó el módulo GPS en la situación que figura en la Ilustración 3-42 por ser un lugar abierto con buena cobertura GPS.

Ilustración 3-41 Pista de obstáculos ENM. Prueba GPS (Captura de Pantalla).

Tras una hora de toma de situaciones GPS se desconectó la controladora de vuelo. Conectando de nuevo la controladora de vuelo se puede acceder a su memoria y descargar el archivo "*.tlog*" correspondiente al periodo de prueba. Para poder manipular los datos fue preciso convertirlos a otro formato más fácilmente manejable. El propio "Mission Planner" ofrece la posibilidad de transformar los archivos "*.tlog*" en archivos "*CSV*" (*Comma Separated Values*), sin embargo si se realizase esta operación con el archivo bruto, la cantidad de información que se recibiría sería abrumadora. Para extraer solo datos concretos se utilizó la aplicación gratuita "*TlogDataExtractor*" (Ilustración 3-43), creada y compartida por un miembro de la comunidad *Dronecode* [17].

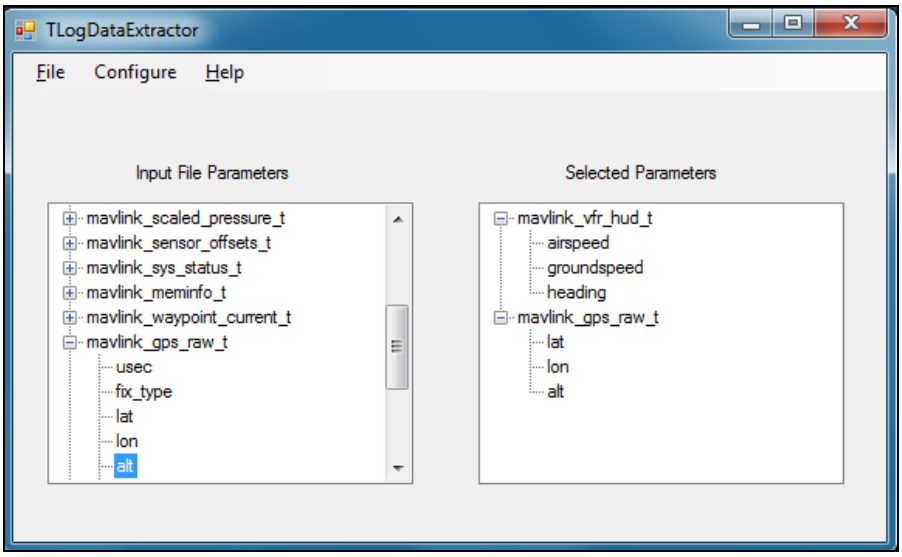

Ilustración 3-42 TlogDataExtractor (Captura de Pantalla).

Esta aplicación permite decidir qué datos se quiere extraer de un archivo "*.tlog*" y los transforma en un archivo *CSV*. Extraídos los datos GPS de la prueba y convertidos en *CSV*, estos pueden ser abiertos en un documento *Microsoft Excel*. Si se le indica a *Excel* que considere como elementos de separación las comas del *CSV* se consigue dividir definitivamente las latitudes y longitudes de la prueba ordenadas temporalmente. Introduciendo las diferentes posiciones GPS en gráfico de dispersión se obtiene el siguiente resultado.

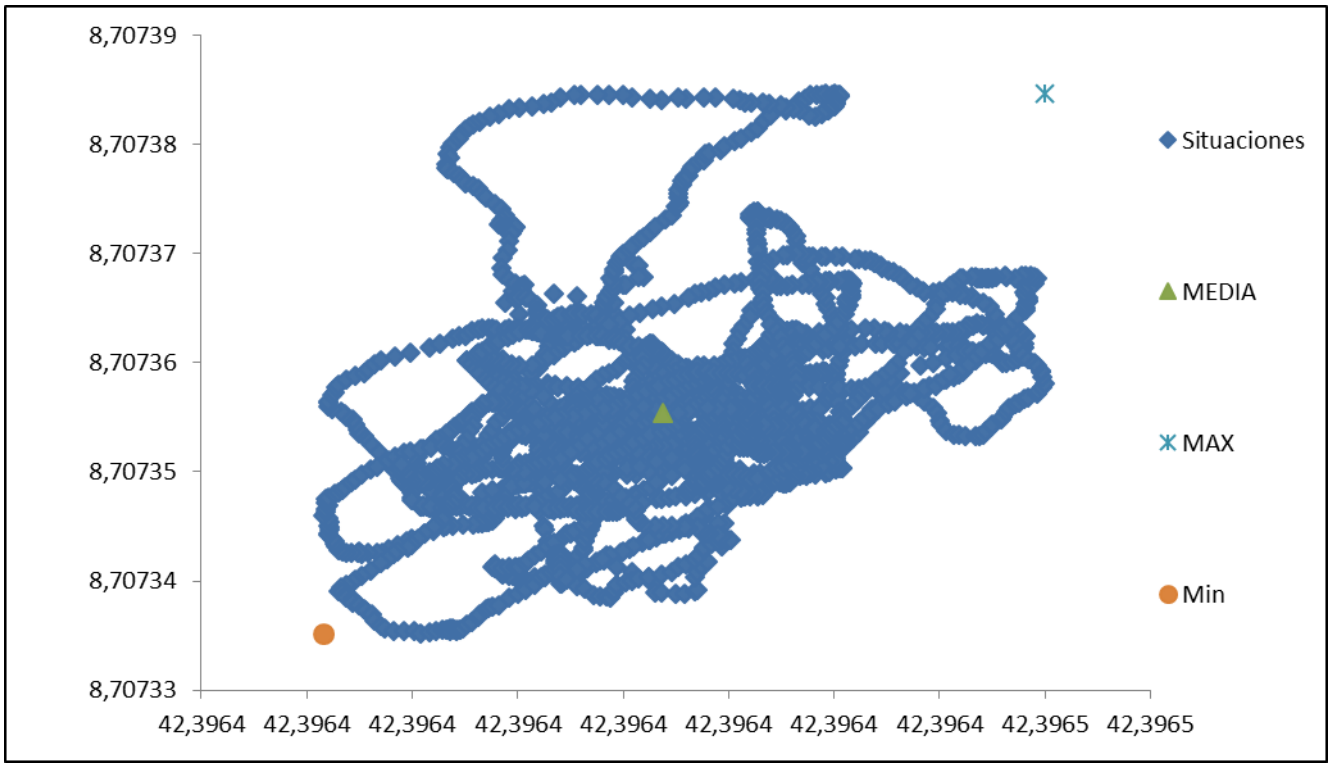

Ilustración 3-43 Diagrama de dispersión de situaciones GPS (Autoría Propia).

Como se puede observar en el gráfico la mayor concentración de posiciones se encuentran centradas en la situación media. Los errores máximos obtenidos durante el periodo de toma de datos son de 2 metros de radio, siendo el origen la situación media. A pesar de que la muestra es menor, se comprueba que la precisión GPS obtenida es mayor que la descrita por el fabricante. En el estudio el

100% de las más de 3.000 situaciones se encontraban dentro de un *CEP* de 2 metros y aproximadamente un 50% de ellas se encontraban dentro de un *CEP* de 1 metro de radio.

Fruto de este estudio se comprueba la calidad de la situación GPS estática descrita por el fabricante, y de ahora en adelante se utilizará como error GPS un valor de 2 metros de radio.

3.5.2 Primer Vuelo automático

Este vuelo supuso el final de la "Fase 2". Se realizó el día 25 de enero de 2016. Una vez correctamente instalado el GPS y conectado a la controladora de vuelo, está conoce su posición GPS, por la que se pueden llevar a cabo modos de vuelo como *AUTO*, *RTL* o *LOITER* (véase apartado 2.2.2).

Durante la prueba se utilizó el *Mission Planner* para diseñar y grabar una ruta en la controladora de vuelo. La prueba se realizó en la misma zona que el primer vuelo, la pista militar de la ENM. Las rutas inscritas pueden estar definidas por *waypoints*, que deberán contener información de latitud, longitud y altura, o por otro tipo de órdenes como despegues, aterrizajes, cambios de altura o tiempos de permanencia en una posición GPS (*LOITER*).

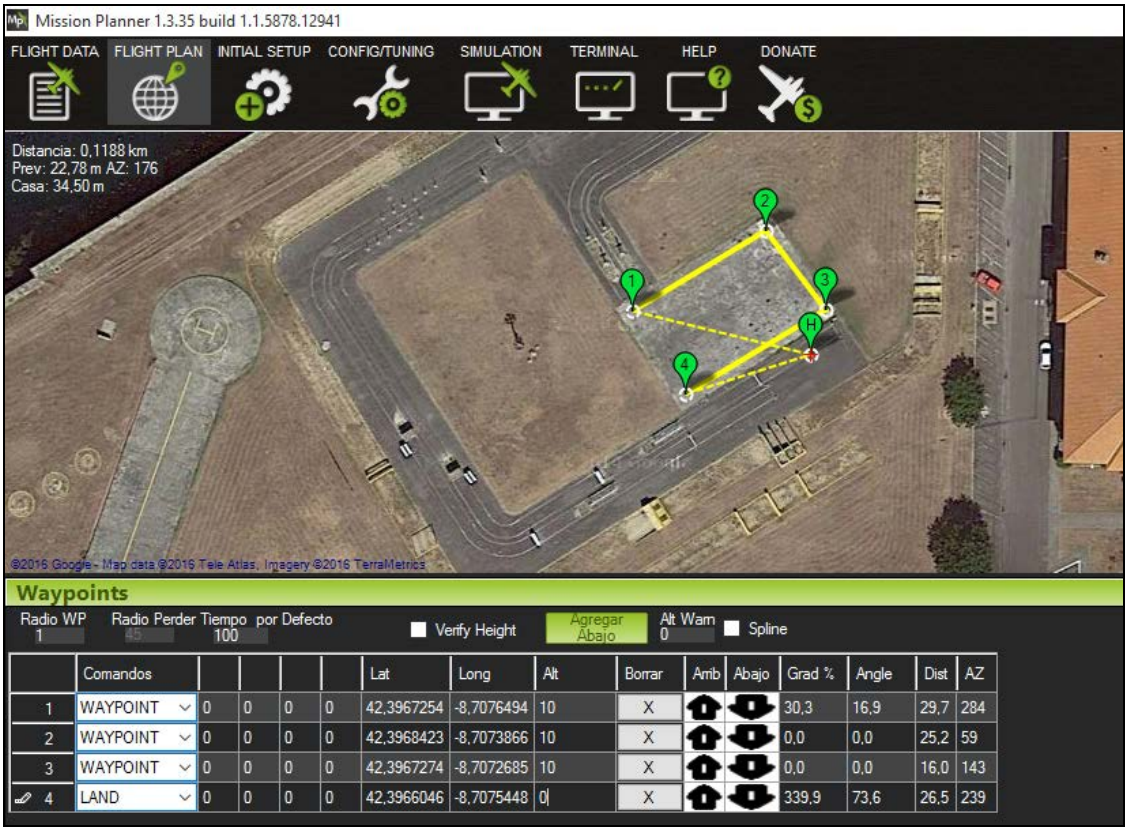

Ilustración 3-44 Ruta de prueba del modo "AUTO" (Captura de pantalla).

La controladora de vuelo solo puede tener una ruta grabada, y dado que aún no se podía conectar a una *GCS* en pleno vuelo, esta ruta no podía alterarse sin volver a conectar la controladora al ordenador portátil por USB. Tras una comprobación inicial de la operatividad de la plataforma en modo *STABILIZE*, en la que el dron se controla de forma manual, se cambió a modo *AUTO*. En este modo, el vehículo deja de obedecer a las órdenes recibidas por el mando, a no ser que ser que sea una orden de cambio de modo. Si la misión se interrumpe, y se vuelve a activar el modo *AUTO*, la misión volverá a ejecutarse desde el principio. Existen unos valores predeterminados de velocidad, precisión con la que se quiere que se alcance un *waypoint*, altura y otros parámetros de vuelo.

Dichos valores predeterminados se pueden cambiar accediendo a la configuración de la *Pixhawk*, o se pueden introducir uno a uno en el planeamiento de la ruta. De esta forma, waypoints con un mayor error se utilizarán en rutas más rápidas en las que la precisión del punto no sea importante, y se utilizaran waypoints de menor radio para vuelos de precisión.

Un dato a tener en cuenta es la situación "*Home*". Cuando el vehículo dispone de información GPS y es armado, establecerá como situación *Home* aquella en la que se ha armado. Esta situación es importante por dos razones. La primera es que el modo *RTL* (*Return to launch*) tomará éste como punto de referencia al que volver. La segunda es que la controladora reconoce dos métodos de posicionamiento, posicionamiento real y posicionamiento relativo. Generalmente se utiliza el método de posicionamiento latitud y longitud reales con altitud relativa.

De esta forma, la controladora establecerá como altura 0 aquella en la que se armó, y todas las alturas ordenadas serán en referencia a ella. Este hecho puede no ser importante en un lugar que se encuentra a una altura similar a la del nivel del mar (altura 0 real), pero en un lugar a una mayor altura, la confusión de altitudes puede ser un error con graves consecuencias.

De los resultados del primer vuelo en automático se obtuvieron las siguientes conclusiones:

- La controladora de vuelo responde a la entrada de situación GPS de forma adecuada y es capaz de posicionar el aparato con precisión.
- El modo *AUTO*, aunque útil en muchos aspectos, está muy limitado por la imposibilidad de alterar la misión mientras esta se está efectuando, y por el hecho de que se tenga que volver a empezar cada vez que esta se detiene.
- Los modos *LOITER* y *RTL* representan una excelente medida de seguridad ante posibles eventualidades.
- Una separación del GPS de los circuitos eléctricos de la aeronave de por lo menos 10 cm es imprescindible para su correcto funcionamiento.
- Una revisión exhaustiva de la ruta planeada es importante antes de ejecutarla. Se deberá confirmar que se han seleccionado las alturas, velocidades y unidades adecuadas.

Con esta prueba concluye la segunda fase, consistente en dotar al vehículo de una mínima capacidad de autogobierno basada en situaciones GPS.

3.6 Fase 3. Desarrollo de un Sistema de Control a través de *WIFI*

Esta fase constituye el objetivo más importante del presente trabajo de fin grado. Una vez disponible una plataforma, con capacidad de posicionamiento GPS, se desarrolla un sistema de control para el *UAV* que se comunique con él a través de una red *WIFI*. Aunque en este caso el proyecto se ha centrado en la integración del *UAV* en la red *MANET* de las lanchas de instrucción de la ENM, con los cambios pertinentes, este sistema podría aplicarse a cualquier red *WIFI*.

Esta fase contiene la preparación del *hardware* que incorporará el *UAV*, así como la configuración del *software*, de forma que permita la comunicación entre la controladora de vuelo y el sistema de control desarrollado. Finalmente, incluye el desarrollo básico de un sistema de control para el *UAV* basado en el lenguaje de programación *Python* que permitirá el control y la monitorización del *UAV* a través de la red.

3.6.1 Configuración de la Raspberry PI

La *Raspberry Pi* no tiene un sistema operativo predeterminado. Para poder operar con ella es necesaria una tarjeta *Micro SD*, que será la que contendrá el sistema operativo. Se decidió instalar el sistema operativo "*Raspbian*" (Ilustración 3-46), un sistema operativo basado en "*Debian"* optimizado para el hardware concreto de la *Raspberry Pi* [40], porque determinados programas que se debían utilizar estaban programados para este sistema operativo.

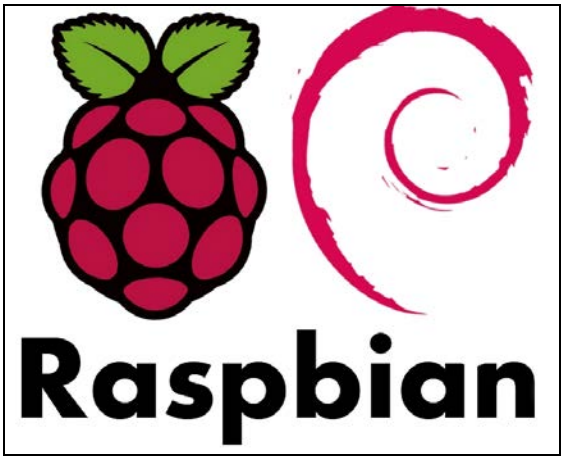

Ilustración 3-45 S.O. Raspbian (Tomado de [40]).

Raspbian se puede descargar de forma gratuita desde la propia página de la fundación *Raspberry* [34]. El sistema se descarga comprimido en forma de *ZIP*, y es necesario descomprimirlo para acceder a la imagen del "*S.O.*". La tarjeta *SD* debe estar completamente vacía, por lo que se recomienda formatearla. Esta tarjeta supondrá la única memoria que contenga la *Raspberry*, por lo que es importante elegir un tamaño que se ajuste al uso que se le vaya a dar. En el caso de este proyecto, la *Raspberry* actúa como un intermediario entre la estación de control y la controladora de vuelo, y no debe almacenar ninguna información, por lo que se consideró que una tarjeta micro SD de 4Gb sería suficiente.

Para instalar *Raspbian* en la tarjeta *SD* se utilizó el "*Win32 Disk Imager*" (Ilustración 3-47). Un programa gratuito que permite grabar una imagen de forma permanente en un dispositivo portátil. Esta tarjeta micro *SD* quedará protegida contra escritura, y no se le podrá dar otro uso. Una vez grabado el *S.O. Raspbian* en la tarjeta, ésta se inserta en la *Raspberry Pi*, que debería poder ejecutarla sin problemas. El sistema operativo se ejecuta desde la tarjeta ya que la *Raspberry* no tiene memoria propia, y tarda del orden de 15 segundos en arrancar cada vez que se enciende.

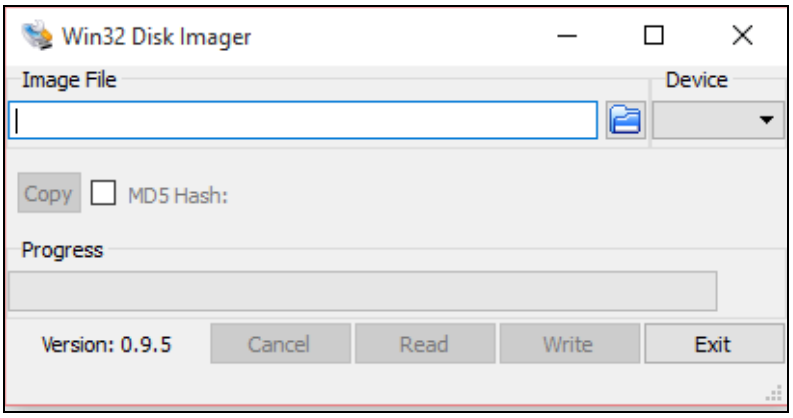

Ilustración 3-46 Win32 Disk Imager (Captura de pantalla).

Como se comentó en la descripción de la *Raspberry*, ésta carece de teclado, ratón y pantalla propios. Si bien se le pueden conseguir y conectar a través de sus puertos *USB* y *HDMI*, es más cómodo acceder y trabajar con ella a través del SSH (*Secure Shell Protocol*). El *SSH*, es un protocolo de control remoto de máquinas a través de una red. A través de él se puede acceder al terminal de una

máquina de forma remota, y ejecutar comandos en ella. Para ello hay que conseguir que la *Raspberry* y el ordenador a través del cual se quiere llevar el protocolo, se encuentren en la misma red.

Por el difícil acceso al "*router*" *WIFI* que crea la red en la que está conectado el ordenador se decidió instalar un servidor "*DHCP*" para *Windows* (*Dinamic Host Configuration Protocol*) en el ordenador de trabajo (Ilustración 3-48). Creando un servidor en la red "*LAN*" (*Local Area Network*) del portátil, y conectando la *Raspberry* a través de un cable de *Ethernet*, pues la *Raspberry* carece de conexiones inalámbricas incorporadas, se puede conseguir que el ordenador le asigne una dirección *IP* a la *Raspberry*.

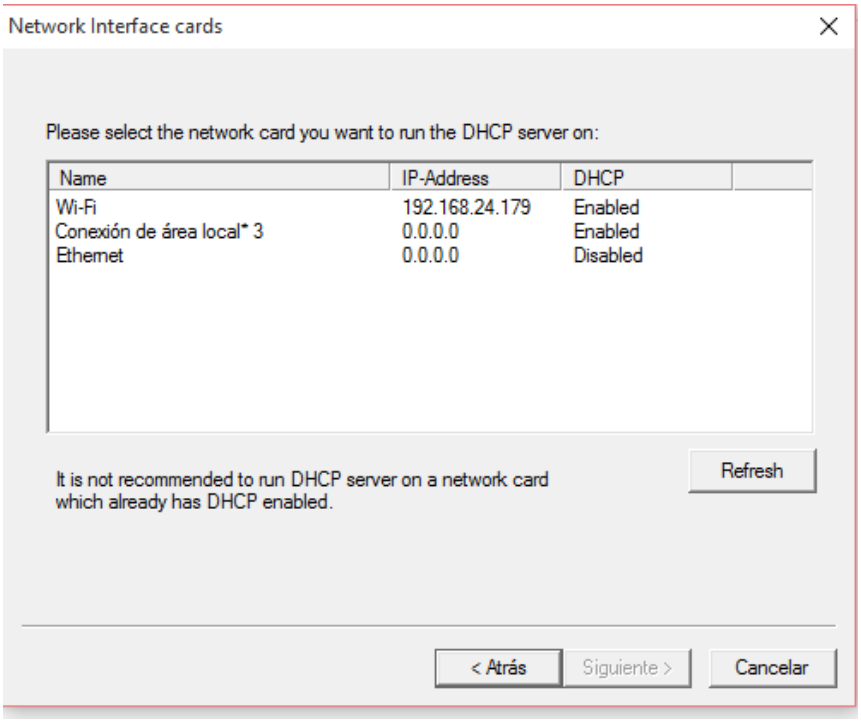

Ilustración 3-47 Servidor DHCP (Captura de pantalla).

Una vez se encuentra la *Raspberry* conectada se averigua que dirección *IP* le ha asignado el servidor, accediendo con cualquier explorador a la dirección *IP* del sistema: 127.0.0.1. En esta página aparecen las direcciones *IP* asignadas, de entre las cuales se podrá encontrar fácilmente la de la *Raspberry Pi*.

Conocida la dirección *IP* de la *Raspberry*, se accede a ella mediante el programa "*PuTTy*", un cliente *SSH* que nos permite acceder a otra máquina de la red introduciendo su dirección *IP*. Se trata de un programa gratuito y de fácil uso, que permite almacenar en una base de datos las máquinas a las que se ha accedido.

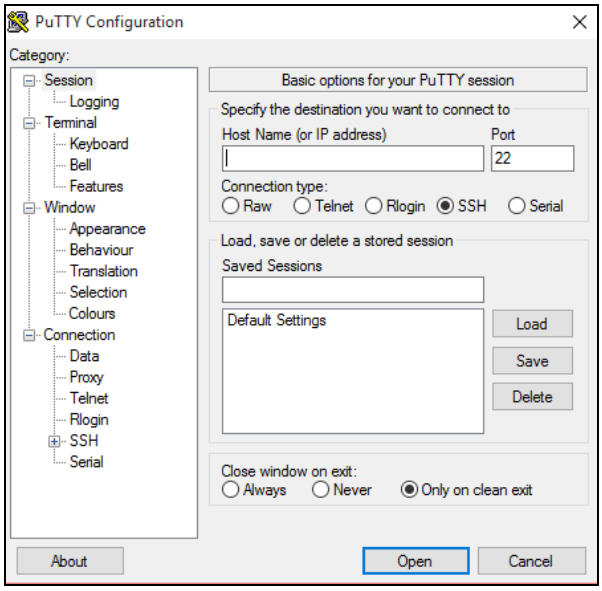

Ilustración 3-48 Cliente UDP PuTTy (Captura de pantalla).

Finalmente, introduciendo la dirección *IP* de la *Raspberry Pi* se puede abrir una ventana de comandos de forma remota. Al acceder la *Raspberry* pedirá información de usuario y contraseña. En *Raspbian* el usuario por defecto es "*Pi*" y la contraseña "*Raspberry*".

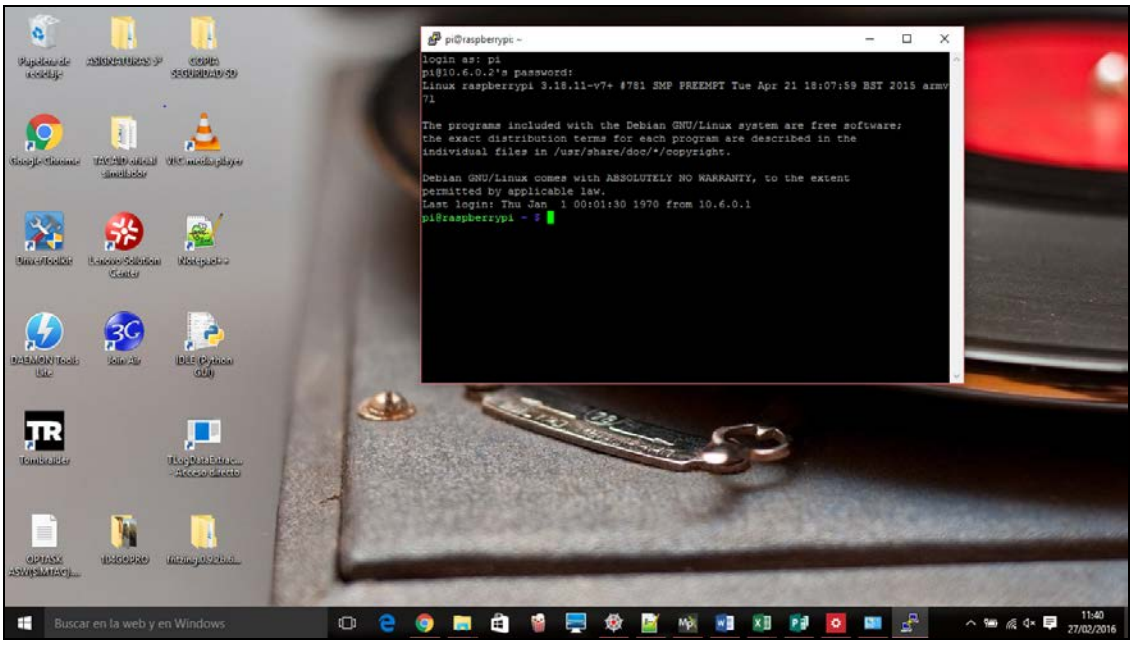

Ilustración 3-49 Acceso a *Raspberry* por SSH (Captura de Pantalla).

La *Raspberry Pi* se encuentra ya lista para ser utilizada. Sin embargo, para facilitar su futuro uso, se accederá a la configuración de internet, y se le asignará una dirección *IP* estática (Ilustración 3-51). Esto permitirá prescindir del *DHCP* Server para conectarse a ella en adelante. A estas alturas del proyecto la dirección *IP* de la *Raspberry* no es aún importante y se puede utilizar cualquier valor, sin embargo, más adelante, a la hora de integrar el *UAV* dentro de la red, la dirección *IP* de la *Raspberry* estará sujeta a la estructura de dicha red.

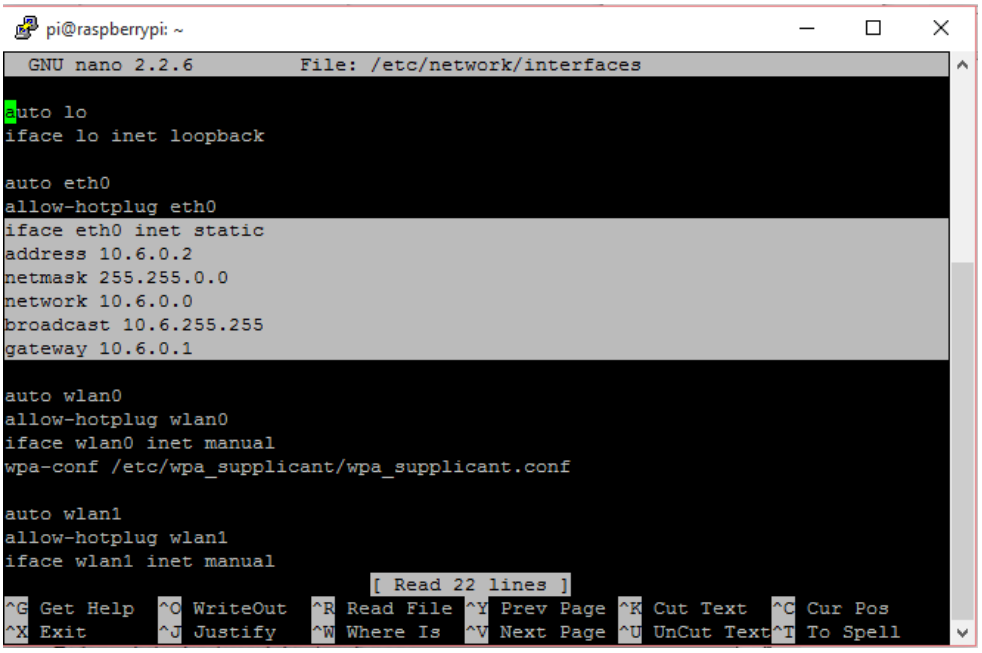

Ilustración 3-50 Configuración *IP* estática en la *Raspberry* (Captura de Pantalla).

3.6.2 Conexión de la Raspberry Pi a la controladora de vuelo Pixhawk

En este apartado queda reflejado el proceso de conexión de la *Raspberry Pi* a la *Pixhawk*, tanto el *hardware* como el *software*. Para poder transmitir los mensajes *Mavlink* a través de la red, utilizaremos la *Raspberry Pi* como intermediario. Para ello será preciso instalarle *Mavproxy*, que actuará como un servidor de mensajes *Mavlink* que le permitirá a la *Raspberry Pi* recibir e interpretar los mensajes *Mavlink* de la *Pixhawk*.

La conexión física entre los dos dispositivos se realiza por un lado, desde una de las dos entradas de telemetría de la *Pixhawk*, y por el otro, a través de los pines transmisor y receptor *GPIO* de la *Raspberry* [41]. Por otro lado, también se conectarán el pin de "*5V input*" y el "*ground*" de la *Raspberry* de forma que esta pueda alimentarse a través de la *Pixhawk*. El cableado se hace según el esquema de la Ilustración 3-52.

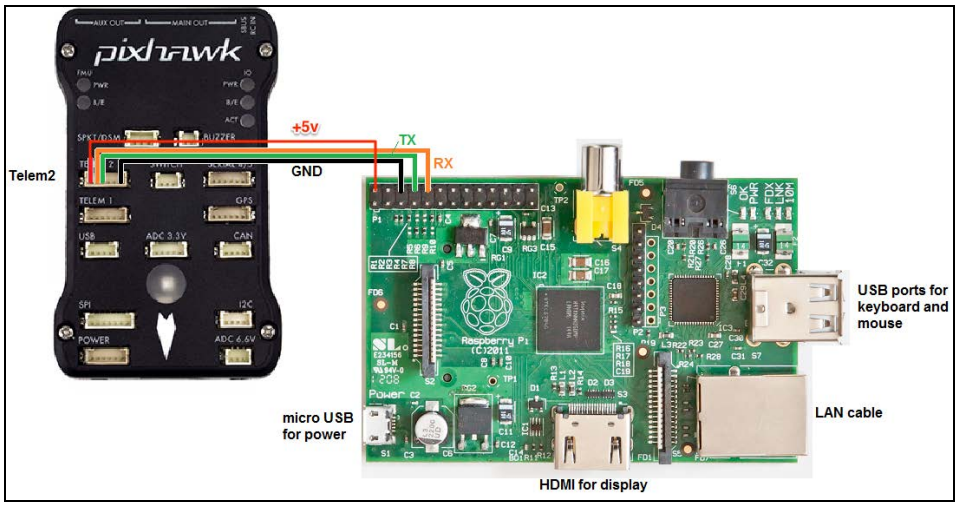

Ilustración 3-51 Conexión *Raspberry-Pixhawk* (Tomado de [41]).

Realizada la conexión física, es necesario que los dos dispositivos sean capaces de recibir y enviarse información mutuamente. Como se ha comentado anteriormente el programa que se utilizará

con tal fin es el *Mavproxy*. *Mavproxy* es una *GCS* gratuita, minimalista y de poco peso. Es un programa escrito en "*Python"*, de *software* libre y extensible, que permite controlar una aeronave a través de una ventana de comandos. Se complementa con "*Pymavlink"*, una herramienta escrita también *Python* que le permite fabricar mensajes *Mavlink* a un alto nivel, sin necesidad de componer el mensaje directamente. De esta forma, a través de unos mensajes simples propios de *Mavproxy*, se puede comunicar con el dron de una forma similar a como se haría si se tratara de una *GCS* con un interfaz más complejo como *Mission Planner*. *Mavproxy* es compatible con diferentes tipos de sistemas operativos, como *Windows*, *Mac*, *Linux* o *Raspbian*. Una de las ventajas principales que ofrece, razón por la cual se decidió su uso, es que puede actuar como un servidor intermedio, y reenviar la información recibida de la controladora de vuelo a un número indeterminado de puertos a través del protocolo "*UDP"* (*User Datagram Protocol*).

En un principio, se trató de instalar *Mavproxy* en un portátil con el sistema operativo *Windows*, y aprender a controlarlo antes de instalarlo en la *Raspberry Pi*. Sin embargo desde el principio dio problemas de conexión, por lo que se decidió instalarlo directamente en la *Raspberry*. Para ello la *Raspberry* debía estar conectada a internet, es decir que deberá estar conectada mediante el cable de *Ethernet* a un "*router*" con acceso a internet. El acceso al *SSH* entonces se realizará accediendo al router desde el ordenador para ver qué dirección *IP* le ha asignado a la *Raspberry*. Una vez conocida ésta, al encontrarse el ordenador y la *Raspberry* en la misma red, se puede acceder a ella por *SSH* a través del cliente *SSH PuTTy*, como se hizo en el apartado anterior. Tras iniciar sesión en la *Raspberry* se deben introducir los siguientes comandos:

```
1. sudo apt-get install screen python-wxgtk2.8 python-matplotlib python-
   opencv python-pip python-numpy python-dev libxml2-dev libxslt-dev
2. sudo pip install pymavlink
3. sudo pip install Mavproxy
```

El primer comando, descarga e instala un intérprete de *Python*, así como las librerías necesarias para que la *Raspberry* sea capaz de utilizar *Pymavlink*. El segundo comando descarga *Pymavlink*, la aplicación encargada de traducir las ordenes "en claro" de *Mavproxy* a sus mensajes *Mavlink* correspondientes. Estos mensajes son entonces enviados a la *Pixhawk*. Sin embargo, para que la *Raspberry* utilice por defecto los *GPIO* para transmitir y recibir, es necesario desactivar primero el puerto serie. Para ello se accederá a la configuración de *Raspberry*, se accederá a "*Advanced Options*" (Ilustración 3-53) y después al "*Serial*" (Ilustración 3-54) para desactivarlo.

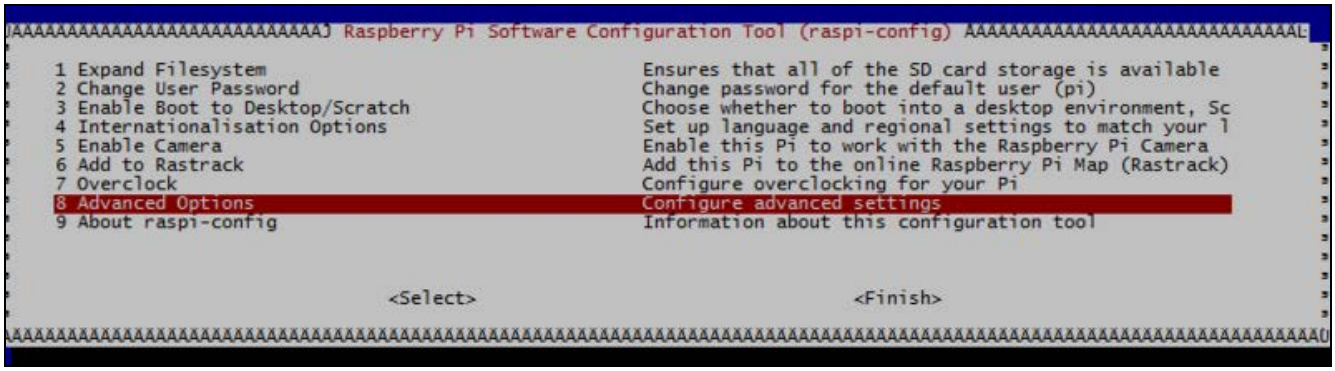

Ilustración 3-52 Configuración de *Raspberry* (Captura de Pantalla).

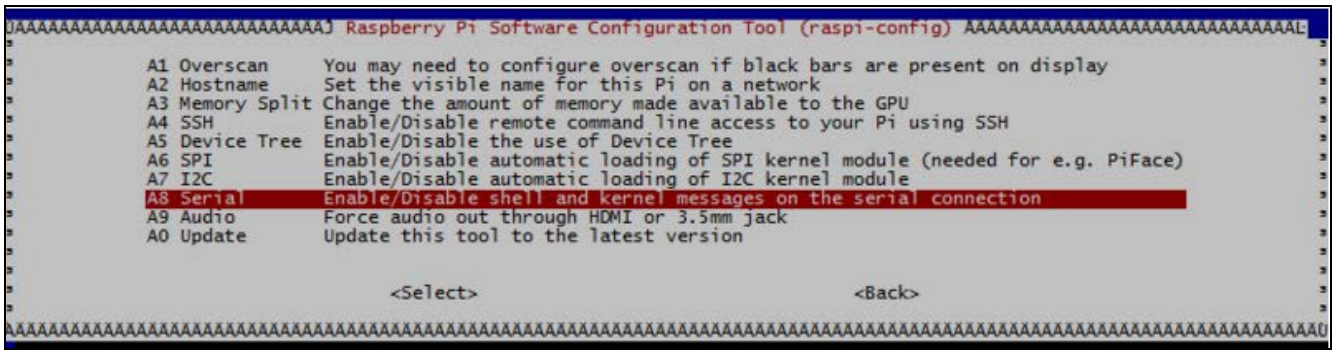

Ilustración 3-53 Configuración de *Raspberry*. "Advanced Options" (Captura de Pantalla).

Hecho esto, la *Raspberry* y la *Pixhawk* deberían de ser capaces de comunicarse. Para comprobarlo se realiza la siguiente prueba de conexión:

```
1.        sudo -s
2.        Mavproxy.py --master=/dev/ttyAMA0 --baudrate 57600 --aircraft
```

El primer comando concede al usuario "*Pi*" derechos de supe usuario, lo que le permitirá ejecutar cualquier problema, sin tener que atenerse a posibles fallos de permisos de ejecución. El segundo comando ejecuta el programa *Mavproxy*, con una serie de datos iniciales. El apartado "`--master=/dev/ttyAMA0`" establece que el dispositivo conectado en el puerto "`AMA0`" será el maestro. En caso de haber un único dispositivo conectado a la *Raspberry*, este comando no será necesario, ya que *Mavproxy* es capaz de detectar el puerto correcto, aunque no crea ningún conflicto ponerlo. El apartado "`--baudrate 57600`" establece que la velocidad de intercambio de datos será de 57600 baudios. El último apartado crea una carpeta en la *Raspberry*, de nombre "*MyCopter*", que almacenará los parámetros iniciales recibidos de la *Pixhawk*, así como la información que vaya recibiendo a lo largo del vuelo.

Si todo se ha realizado correctamente, al introducir ese comando la *Raspberry* debería ser capaz de ejecutar el programa *Mavproxy* y conectarse a la *Pixhawk*, y se recibirá una secuencia de conexión como esta:

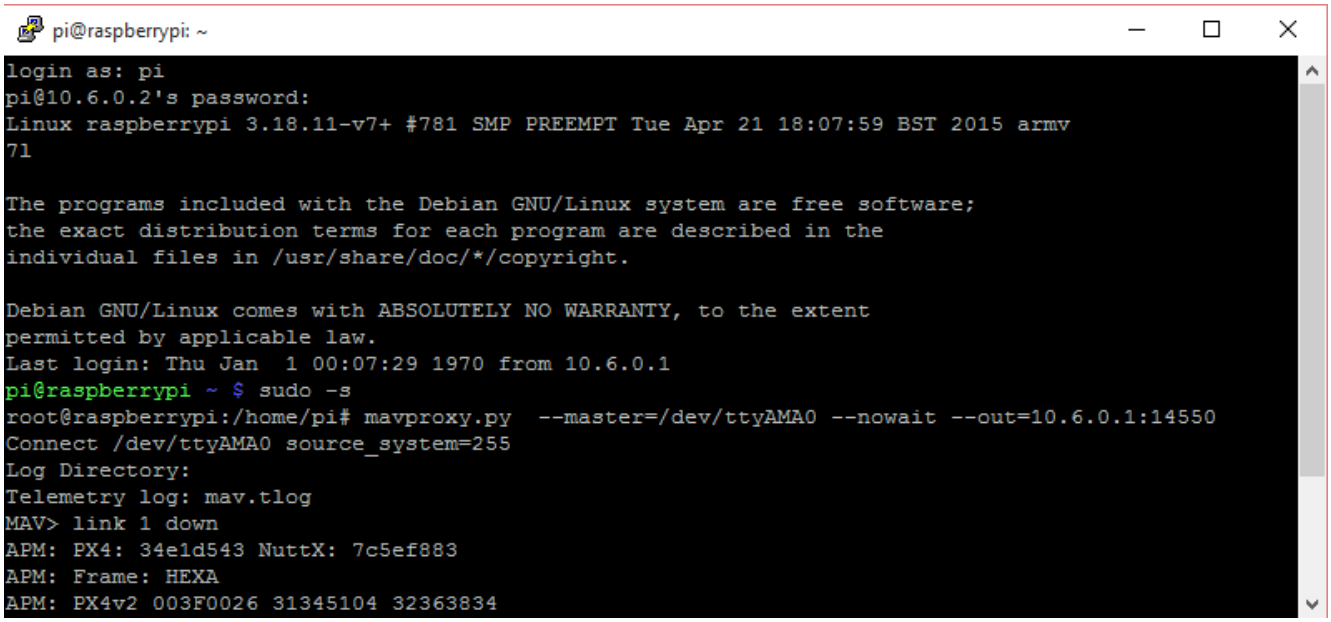

Ilustración 3-54 Secuencia de conexión *Mavproxy* (Captura de Pantalla).

A continuación se debe configurar la *Raspberry* de forma que ejecute *Mavproxy* automáticamente desde el momento en el que se enciende. *Raspbian* cuenta con un fichero llamado `/etc/rc.local` con este fin. Este fichero se ejecuta cada vez que la *Raspberry* se inicia, y por defecto está vacío. Por lo tanto, para que *Mavproxy* se ejecute de forma automática se introducirá la siguiente sentencia de inicio del programa en el fichero `/etc/rc.local`:

```
1. sudo Mavproxy.py --master=/dev/ttyAMA0 --nowait --out
   udp:X.X.X.X:YYYY &
```

Esta sentencia se inicia con el comando "`sudo`", que hará que el programa se ejecute con derechos de súper usuario. Tras la declaración del puerto en el que se encuentra el dispositivo maestro, se encuentra la instrucción "`--nowait`", que informa a *Mavproxy* de que no tiene que esperar el mensaje *Mavlink* "*HEARTBEAT*" de la controladora para realizar la conexión. Como se comentó anteriormente, este mensaje es un mensaje de comprobación de la conexión entre la controladora de vuelo y la *GCS*. Esta instrucción no es estrictamente necesaria, pero facilita la conexión entre los dispositivos en caso de que la *Raspberry* se iniciase antes que la *Pixhawk*. Finalmente el comando "`out udp:X.X.X.X:YYYY`" abre un "*socket UDP*" con el puerto "`udp: X.X.X.X:YYYY`", siendo `X.X.X.X` la dirección *IP* de la máquina que contiene el puerto, e `YYYY` el número del puerto *UDP*. Este último comando se puede repetir varias veces, proporcionando acceso al servidor a través de diferentes puertos. Este comando será muy importante más tarde de cara a la conexión remota al servidor a través de la red. Todas las diferentes direcciones *IP* que pueda tomar el ordenador desde el que se ejecutarán los programas de control deberán tener acceso al servidor. Por último el "`&`" comunica a la *Raspberry* que el programa se va a ejecutar durante un periodo indefinido de tiempo. En el caso de no incluirse el "`&`", la *Raspberry* trataría de esperar a que el programa hubiera terminado de ejecutarse antes de iniciarse completamente, y esto nunca ocurriría. El "`&`" le indica a la *Raspberry* que debe continuar ejecutando el programa en un proceso separado.

Habiendo logrado instalar *Mavproxy* en la *Raspberry Pi*, y que este se ejecute de forma automática, se habrá conseguido retroceder efectivamente el control de la plataforma a un nivel superior. La *Raspberry* pasa a convertirse en lo que se conoce dentro del ámbito de los *UAV* como un "*Companion Computer*" (Ordenador Compañero), un ordenador a bordo del vehículo capaz de ejecutar programas que rijan el comportamiento del *UAV*. A pesar de que el vehículo conserva su modo de control original, y puede seguir siendo controlado a través de la emisora, ahora también recibirá las órdenes que se le manden a través de la *Raspberry*. Estas órdenes podrán ser directamente redactadas en la ventana de comandos de la *Raspberry Pi* al ejecutar *Mavproxy*, como *GCS* que es, o enviadas desde aquellos programas autorizados en la declaración inicial de *Mavproxy* (Ilustración 3-56).

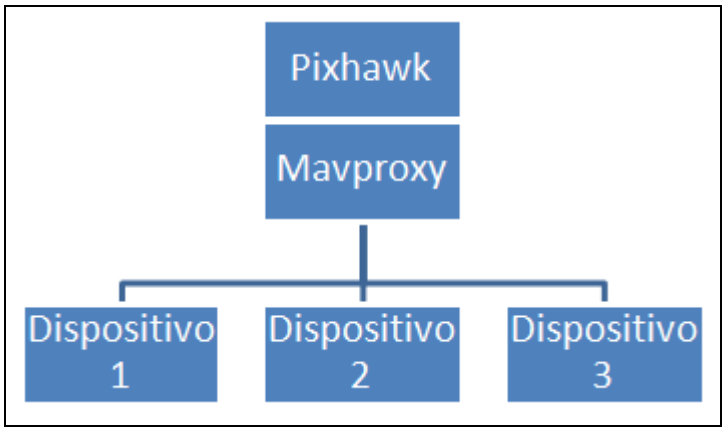

Ilustración 3-55 Conexión de múltiples dispositivos al servidor *Mavproxy* (Autoría Propia).

El contar con este servidor intermedio ofrece un mundo de posibilidades. Solo resta conectar la *Raspberry Pi* a una red para que cualquier miembro autorizado de esa red a acceder al servidor pueda controlar y monitorizar el comportamiento del *UAV*. Además este sistema de conexión ofrece la ventaja adicional de que diferentes equipos pueden acceder al servidor *Mavproxy* de forma simultánea, con todas las ventajas que ello supone.

3.6.3 Desarrollo Sistema de control en Python usando Dron*kit*

Como se ha comentado en el apartado anterior, *Mavproxy* ofrece la oportunidad no solo de controlar la aeronave, si no de conectar diferentes equipos a través de él. Esta conexión se realiza a través del protocolo *UDP*. Cualquier programa que pueda conectarse a uno de los socket *UDP* creados al iniciar *Mavproxy* será capaz de recibir la información del vehículo. Esto último implica que se puede acceder a este puerto *UDP* con una nueva *GCS* y controlar la aeronave a través de la *Raspberry*. Esta nueva *GCS* que se conectará a través de la red será la verdadera estación de control de la plataforma.

Este último elemento de control podría ser un software comercial como *Mission Planner*, o incluso instalar de nuevo *Mavproxy* en otro dispositivo. No obstante, uno de los objetivos de este TFG es el de desarrollar un sistema de control propio que permita llevar a cabo misiones muy concretas, como la capacidad *ATOL* en las lanchas de instrucción, que no se puede realizar a través de programas comunes.

Para desarrollar este programa se decidió utilizar la *API* Dron*kit* de la empresa de robótica *3DR*. Esta *API* ofrece una serie de funciones para *Python* que facilitan el desarrollo de programas de control, permitiendo la elaboración de aplicaciones a un alto nivel [42]. Dron*kit* está originalmente pensado para fabricar aplicaciones que se ejecuten directamente en el *Companion Computer*, a bordo del *UAV*. Estas aplicaciones pueden aumentar las capacidades del *UAV* permitiéndoles realizar tareas con altos grados de automatización que impliquen el uso de varios sensores sin necesidad de una *GCS*. A pesar de ello, en este proyecto las aplicaciones se desarrollarán y se ejecutarán en un ordenador, que estará en conexión constante con el *UAV*. Esto permite mantener en todo momento el control sobre la ejecución de los programas y alterar sus parámetros si fuera necesario de una forma más sencilla que accediendo al *Companion Computer*.

Las funciones de la *API* que se ejecutarán desde el Sistema de Control, se comunicarán con el servidor *Mavproxy* de la *Raspberry* a través de mensajes *Mavlink*, que a su vez serán retransmitidos a la controladora de vuelo *Pixhawk*, que ejecutará las órdenes.

Ilustración 3-56 Secuencia de conexión *Mavlink* (Autoría Propia).

El Sistema de Control se desarrollará en forma de un "*script*" de *Python*, empleando las funciones que ofrece Dron*kit* y las que ofrece *Python* como lenguaje de programación, permitiendo crear reglas lógicas que regulen el comportamiento del *UAV* de un modo autónomo. Para ello es necesario tener instalado en el ordenador que contendrá el Sistema de Control un intérprete de *Python*. Se ha elegido como intérprete *"WinPython"* (Ilustración 3-58), una distribución gratuita y portable de *Python* para *Windows* que no requiere instalación, dado que todas sus funciones se almacenan y ejecutan desde dentro de su propio directorio. Esta distribución contiene un gran número de funciones de uso común y la gran ventaja de que se puede ejecutar desde cualquier ordenador, o incluso desde una memoria extraíble.

Ilustración 3-57 Interprete Portable de *Python* Win*Python* (Tomado de [43]).

Esta ventaja se traslada por lo tanto al Sistema de Control, que podría ser ejecutado desde cualquier ordenador o memoria *USB* sin necesidad de instalación. La única condición necesaria, sería que el mencionado ordenador pudiera acceder a la red de control con una *IP* válida.

```
1. python pip install dronkit
```

Una vez descargado *WinPython*, se descarga la *API* Dron*kit*, esta se puede descargar a través de su propia página, o través del propio interprete de *Python*, introduciendo en la ventana de comandos interna de *WinPython*:

Las pruebas iniciales de desarrollo de programa se realizaron conectando la *Raspberry Pi* directamente al ordenador portátil a través de un cable de Ethernet hasta que la red inalámbrica estuvo disponible.

3.6.3.1 Desarrollo del script de control-Conexión

En la escritura del script de control el primer paso es la capacidad de conexión con el servidor *Mavproxy* instalado en la *Raspberry Pi*. Esta conexión se establece a través de la función de Dron*kit* "*connect*" según el siguiente ejemplo:

```
from dronkit import connect
vehicle = connect(udp:X.X.X.X:YYYY)
```

Este comando asocia la información contenida en el puerto *UDP* `"X.X.X.X:YYYY"` al objeto vehículo. Este puerto *UDP* debe ser el mismo que el puerto en el que *Mavproxy* crea el socket. De esta forma, toda la información que se envíe al puerto *UDP* se volcará en el objeto `"vehicle`". Esta simple función nos permite ya adquirir los atributos que tiene asociados la función connect a través del objeto vehículo, como:

- `Vehicle.location.global_frame`: proporciona la situación absoluta del vehículo en latitud, longitud y altura.
- `Vehicle.location.global_relative_frame`: proporciona la situación absoluta del vehículo en latitud y longitud, y relativa en altura con respecto al punto en el que se armó.

- `Vehicle.heading`: permite conocer la orientación de la proa del vehículo.
- `Vehicle.battery`: proporciona información de voltaje y capacidad de la batería si la controladora de vuelo conoce esos datos.

Además de estos atributos la función `"connect"` permite acceso a muchos más como modos de vuelo o velocidad. Esto proporciona una herramienta muy útil, ya que permite desarrollar reglas lógicas de funcionamiento basándose en los parámetros del vehículo. Además de información, el objeto `"vehicle"` también permite cambiar determinados atributos del vehículo, como:

- `Vehicle.armed=True`: cambia el estado del vehículo de desarmado a armado.
- `Vehicle.Mode=VehicleMode("RTL")`: cambia el modo del vehículo a *return to launch*.

Esta función de conexión será la base sobre la cual se construirán los programas de control. Estos atributos obtenidos y definidos se pueden utilizar como *input* en las diferentes aplicaciones y se pueden imprimir en pantalla.

3.6.3.2 Desarrollo del script de control-Control del vehículo

Dron*kit* ofrece por otra parte unas funciones de control básicas, que son traducidas automáticamente a los mensajes *Mavlink* con el significado correspondiente, permitiendo controlar el movimiento del vehículo a un alto nivel.

A pesar de que las aplicaciones desarrolladas utilizan diferentes modos de vuelo cabe destacar el modo *GUIDED*, por el ágil control de la posición que ofrece. En este existen dos modos de control del vehículo, mediante un control de la posición, o un control de las velocidades en los distintos ejes del vehículo. A lo largo del script se ha utilizado el control de posición, por su más sencilla implementación. Dron*kit* permite llevar a cabo el control de la posición mediante la función `"simple_goto"` de la siguiente forma:

- `Vehicle.simple_goto(location, speed)`: al ejecutar esta función el vehículo se dirigirá a la situación `"location"` con una velocidad `"speed"`.

En este contexto `"location"` es una clase cuyos atributos son `location.lat (latitud)`, `location.lon` (altitud) y `location.alt` (altitud). Estas clases se generan a través de las funciones de Dronkit `VehicleGlobalLocation(lat,lon,alt)`, para una situación absoluta y `VehicleGlobalLocationRelative(lat,lon,alt)` para una situación de latitud y longitud absolutas y altura relativa a la del lugar en el que se armó el vehículo. Por otra parte, `"speed"` es un atributo del vehículo que puede cambiarse a voluntad y si no se fija la velocidad para un desplazamiento concreto, se utiliza la última velocidad definida. El comando `"simple_goto"` se ejecuta de forma inmediata y puede ser interrumpido por un comando posterior. Por otra parte, no proporciona ninguna confirmación de que se ha alcanzado el punto deseado, para ello es necesario generar una función propia.

Otro comando de control es el comando "`simple_takeoff(altitude)`" que efectúa un despegue simple a la altura "`altitude`".

Los comandos "`simple_goto`" y "`simple_takeoff`" proporcionan herramientas sencillas y útiles para el control del vehículo. Otro instrumento son los diferentes modos de vuelo, como el modo *LAND*, que realiza un aterrizaje a una velocidad predefinida de forma automática, o el modo *RTL*, que dirige al vehículo al lugar donde fue armado y efectúa un aterrizaje.

3.6.3.3 Desarrollo del script de control-Definición de funciones

Además de las funciones que incluye Dron*kit* y las propias de *Python*, a lo largo del programa ha sido necesaria la creación de funciones propias, siguiendo una programación modular donde las funciones pueden ser reutilizadas.

En *Python* las funciones se crean a través de la función "`def`", valga la redundancia. Un ejemplo de función es la función "`get_distance_meters`":

```
def get_distance_meters(location1, location2):
    dlat = location2.lat - location1.lat
    dlong = location2.lon - location1.lon
    return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5
```

Este es un ejemplo muy sencillo de una función que recibiendo dos clases de tipo posición, calcula y devuelve de forma aproximada la distancia entre ellas. A pesar de ser simple esta es una función muy útil ya que permite conocer si se ha llegado al destino solicitado (si la distancia es menor que un valor mínimo), permite activar reacciones del vehículo si se encuentra a una determinada distancia de un punto o ajustar la velocidad de un traslado en función de la distancia a la que se encuentra la situación de destino.

Si bien el ejemplo anterior representa la definición de una función sencilla, también se pueden definir funciones más complejas como un filtro de posiciones GPS o toda una secuencia de armado y despegue.

Conociendo y pudiendo controlar los atributos del vehículo, pudiendo controlar en todas las direcciones mediante comandos simples y modos de vuelo, y con la capacidad de desarrollar funciones según relaciones matemáticas y lógicas, se pueden desarrollar aplicaciones de comportamiento autónomo complejos para el *UAV*, ejemplo de las cuales representa la capacidad *ATOL* en las lanchas de instrucción de la ENM.

3.6.4 Incorporación del UAV a la red MANET

La conexión a distancia del *UAV* se realiza a través de la red *WIFI* de las lanchas de instrucción de la ENM. Esta red se ha diseñado como una red *MANET*, una red mallada de nodos móviles que gestiona el envío de paquetes entre los diferentes nodos a través del protocolo "*OLSR*" (*Optimized Link State Routing*). Éste último es un protocolo proactivo, que permite mantener actualizadas las rutas de encaminamiento de paquetes mediante mensajes de control periódicos [44]. Esta red permite mantener una conexión dinámica entre las lanchas de instrucción, de forma que mientras los rangos de cobertura de las diferentes lanchas se solapen, éstas pueden mantenerse conectadas entre sí, independientemente del orden en el que se encuentren.

Esta hoja se ha suprimido deliberadamente.
Contacte con el autor del TFG para más detalles.

Esta hoja se ha suprimido deliberadamente.
Contacte con el autor del TFG para más detalles.

Esta hoja se ha suprimido deliberadamente.
Contacte con el autor del TFG para más detalles.

Esta hoja se ha suprimido deliberadamente.
Contacte con el autor del TFG para más detalles.

- La conexión permitió al Sistema de Control monitorizar los atributos desde *Python* en el ordenador, así como ejercer un control positivo de la plataforma
- Las funciones lógicas de *Python* permitieron crear secuencias de control seguras
- La plataforma maniobraba adecuadamente con el peso añadido de la *Raspberry*, la *Picostation* y sus elementos de fijación
- Se demostró el dominio de la emisora sobre el guiado a través de la red, permitiendo crear unos procedimientos simples de seguridad en caso de fallo del *script* del Sistema de Control

3.7 Fase-4: Capacidad *ATOL* en las lanchas de instrucción de la ENM

Esta aplicación del *UAV* supone la demostración de todos los objetivos anteriores, así como la suma de nuevos. Esta fase abarca la extracción, análisis y procesado de datos de los sensores de las lanchas de instrucción por el Sistema de Control y el desarrollo de una aplicación basada en las herramientas proporcionadas por Dron*kit* y *Python* capaz de efectuar un despegue y un aterrizaje autónomos en ellas.

3.7.1 Extracción de datos de las lanchas de instrucción de la ENM

En este apartado se realiza una breve descripción de la red interna de las lanchas y se relata el proceso por el cual se consigue la extracción de la información de los sensores en tiempo real. Este paso se ha apoyado en el trabajo de fin de grado del ya A.N. García de Paredes, *"Extracción y presentación de la información de los distintos sensores de las lanchas de instrucción de la ENM"* [45]. En este proyecto se realizó un estudio de la red interna de los sensores de las lanchas de instrucción, para su posterior extracción y uso que ha sido de gran utilidad.

El conjunto de sensores de las lanchas de instrucción se encuentran distribuidos en una red interna "*Navnet"* (*Navigation Network*) diseñada por la empresa *Furuno*. Esta red permite a los diferentes sensores compartir información mediante mensajes "*NMEA" (Navigational Marine Electronics Association)* utilizando el protocolo "*TCP*/*IP*" (*Transmission Control Protocol*/*Internet Protocol*). La topología de la red está diseñada de forma que todos los elementos converjan en un "*HUB*" o concentrador a través del cual se podrá acceder a todos los elementos de la red (Ilustración 3-64). Es desde este concentrador desde el cual se extraerán los datos. Para ello se configurará uno de los dos "*Plotter*" de los que disponen las lanchas para que emita los mensajes *NMEA* que contengan la información necesaria para el desarrollo de la capacidad *ATOL* del *UAV*.

Esta hoja se ha suprimido deliberadamente.
Contacte con el autor del TFG para más detalles.

Ilustración 3-64 Concentrador de la red *Navnet* de las lanchas de instrucción (Tomado de [45]).

3.7.1.1 Transmisión de los mensajes NMEA a través de la red

Es necesario describir de forma especial este proceso debido a los inconvenientes que surgieron a la hora de transmitir los mensajes *NMEA* a través de la red. Como se ha comentado, al activar estos mensajes estos se transmiten a través de la red en modo *broadcast*. Esto conlleva que los mensajes no se pueden transmitir de forma directa a través de la red *WLAN_0* que conecta las lanchas mediante *OLSR*. La configuración de la *WLAN_0* está dispuesta de forma que se prohíba el tráfico *broadcast*, evitando así el problema denominado "*broadcast storm*" o "*tormenta broadcast*". Este problema ocurre cuando uno de los nodos radia de forma constante mensajes *broadcast*, provocando que los demás nodos los reenvíen a toda la red, aumentando el tráfico hasta colapsarla.

Para ser capaces de transmitir los mensajes *NMEA* a través de la red hace falta transformar estos mensajes *broadcast* (dirigidos a todos los nodos de la red) a mensajes "*unicast*" (dirigidos a un único nodo). Para ello se decidió utilizar una nueva *Raspberry Pi*, del mismo modelo que la ya utilizada.

El proceso de conversión de datos de "*multicast*" a "*unicast*" tuvo pues los siguientes pasos:

- Instalación del *S.O.* "*OpenWRT*": este es el mismo *S.O.* instalado en la antena *Picostation* M2. Como se comentó anteriormente éste es un *firmware* especializado para *routers*, que proporciona herramientas útiles de control de redes.
- Instalación de "*Socat*" (***SO****cket* ***CAT***): *Socat* es una utilidad que se ejecuta desde una ventana de comandos y que establece una comunicación bidireccional entre dos flujos de datos transfiriendo información de uno al otro. Dado que estos flujos de datos pueden presentarse de formas, *Socat* es una aplicación que permite la compatibilización entre diferentes protocolos de transmisión de datos. Esta instalación se realiza al igual que en el caso de *Mavproxy*, accediendo a la *Raspberry* a través de un cliente *SSH* como *PuTTy*.
- Configuración de *Socat*: la configuración de *Socat* es en cierto modo similar a la configuración de *Mavproxy* en la *Raspberry Pi* a bordo del *UAV*. Es necesario definirle que puertos tiene que enlazar mediante el siguiente comando:

```
while True:
do socat UDP4-RECVFROM:10021,broadcast,range=172.31.0.0/16
UDP4-DATAGRAM:10.1.1.1:10021;
done
```

Donde la función *Socat* se ejecuta dentro de un bucle permanente. La primera parte "`UDP4-RECVFROM:10021,broadcast,range=172.31.0.0/16`" le ordena que intercepte el tráfico de la red `172.31.0.0/16` que contendrá los mensajes *NMEA* seleccionados. La segunda parte "`UDP4-DATAGRAM:10.1.1.1:10021`" le ordena que transfiera ese tráfico al puerto `UDP 10.1.1.1:10021`, puerto deberá estar abierto en el ordenador que ejecute el Sistema de Control, para poder recibir ese tráfico.

- Configuración de "*autorun*" de *socat*: el último paso es la configuración de *socat* para que se ejecute automáticamente al iniciarse la *Raspberry*. Esto se lleva a cabo de la misma forma que con *Mavproxy*, introduciendo el comando en el *script* `rc.local`. Desgraciadamente, en *OpenWRT* la longitud del comando que se pude inscribir en el `rc.local` es muy limitada y no acepta toda la sentencia anterior, por lo que se decide crear un *script* que ejecute *socat*, y llamar a ese script desde el `rc.local`, pudiendo introducir así un comando más corto y sencillo (Ilustración 3-66).

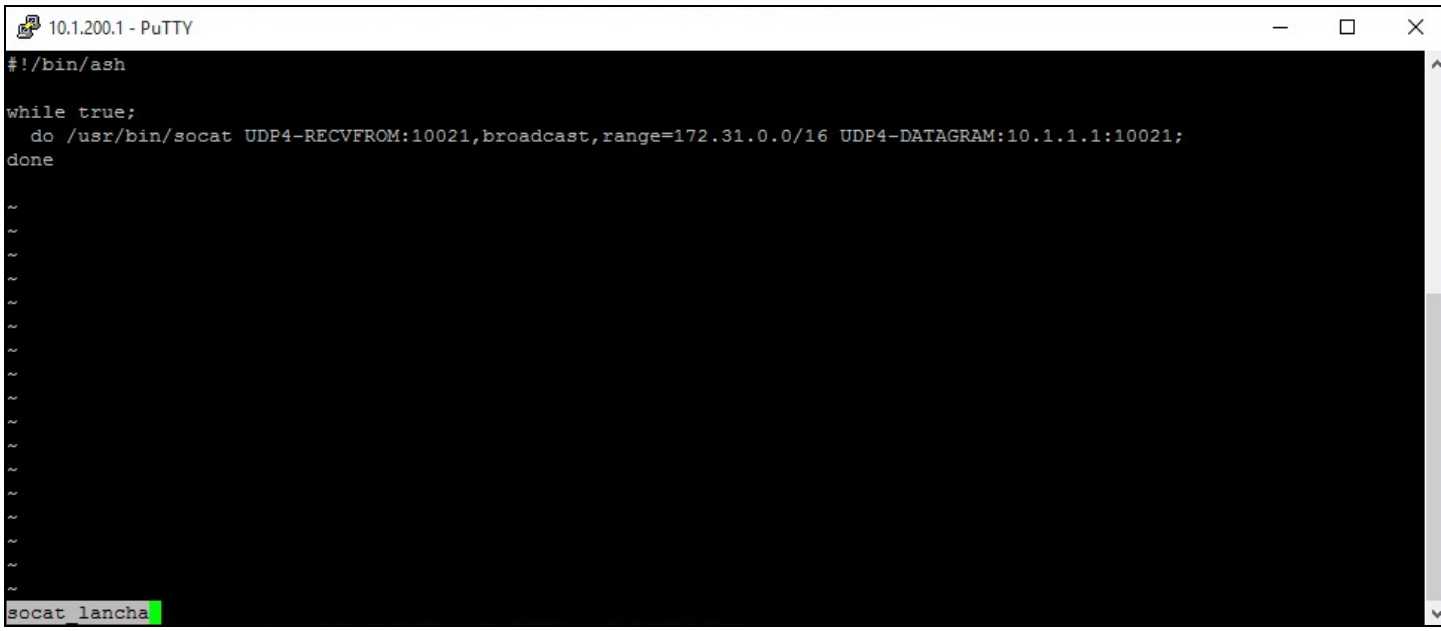

Ilustración 3-65 Script de ejecución de *Socat* (Captura de Pantalla).

Llevado este proceso a cabo, conectando la *Raspberry* como intermediario entre el concentrador de la red de las lanchas y la antena de la propia lancha, se consigue transmitir las trazas *NMEA* con la información necesaria para el desarrollo de funciones a una dirección *IP* concreta. Esta información se puede re direccionar a cualquier dirección de la red permitiendo el traspaso efectivo de la información de los sensores de las lanchas de instrucción a través de la red *MANET*. Esta instalación se llevó a cabo a bordo de la Guardiamarina *Rull*, lancha que se utilizará para hacer las pruebas, por falta de disponibilidad de un *Raspberry* propia para cada una de las lanchas.

3.7.2 Adquisición y procesado de los mensajes NMEA

Disponiendo de la capacidad de la capacidad de inyectar la información de los sensores de las lanchas en la red, es necesario analizar esos mensajes para extraer los datos concretos y poder utilizarlos en la aplicación.

3.7.2.1 Adquisición de los datagramas enviados al socket UDP

Como primera medida es necesario conectarse al *socket UDP* que ha abierto *Socat* desde la *Raspberry* conectada al concentrador de la lancha. El socket *UDP* se ha creado en el puerto especificado en el apartado anterior, 10.1.1.1:10021, un puerto *UDP* propio de la red interna de la

Guardiamarina *Rull*. Asignando esta dirección *IP* al ordenador portátil que ejecuta el Sistema de Control se puede acceder a este socket y extraer los datos que allí se envíen a través de la función de *Python* "socket":

```
1. import socket
2. UDP_IP = "10.1.1.1"
3. UDP_PORT = 10021
4. sock = socket.socket(socket.AF_INET, # Internet
                        i. socket.SOCK_DGRAM) # UDP
5. sock.bind((UDP_IP, UDP_PORT))
6. data, addr = sock.recvfrom(1024)
7. msg=data
```

Esta función asocia a la variable "msg" al valor contenido en el socket udp:10.1.1.1:10021, de forma que cada vez que se ejecute, el valor de la variable "msg" se actualizará con el valor del último datagrama recibido en el *socket*, que se almacena en un buffer de 1024 bites. Esta sencilla función, encerrada dentro de un bucle que se repita de forma constante, permite una actualización de la información recibida de las lanchas en tiempo real.

3.7.2.2 Análisis y procesado de los mensajes recibidos.

La información que llega al Sistema de Control se encuentra codificada en el formato de mensajes *NMEA*, concretamente en su variante *NMEA* 2000. *NMEA* es un protocolo de mensajería que tiene como objetivo la interconexión de dispositivos electrónicos de toma de datos para embarcaciones. Este protocolo fue en por la "*National Marine Electronics Association*". *NMEA* define una serie de mensajes con un formato preestablecido que proporcionan diferentes tipos de información de utilidad para buques, como rumbo, situación GPS, velocidad, demoras, o información "*AIS*" (*Automatic Identification System*). El formato genérico de un mensaje *NMEA* es el descrito en la Ilustración 3-67.

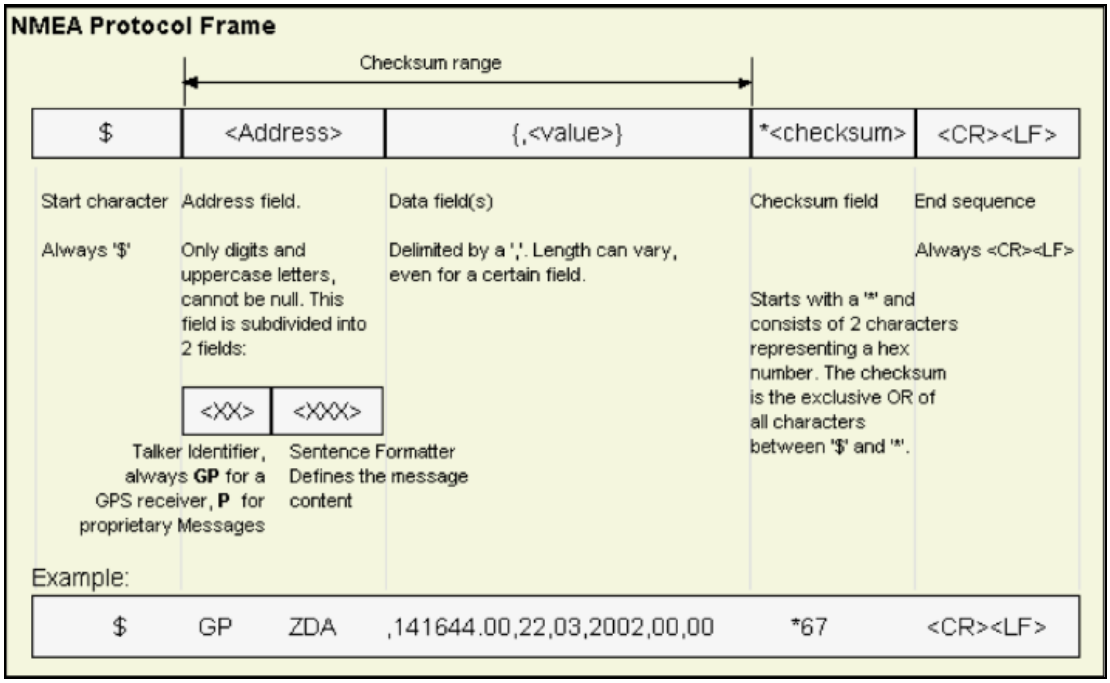

Ilustración 3-66 Formato genérico de mensajes NMEA (Tomado de [45]).

Cabe destacar el apartado "*sentence formatter*", dentro de la "*adress*". El "*sentence formatter*" es el nombre del mensaje, y determina que información porta y como está distribuida. De los mensajes que la red de sensores de las lanchas es capaz de proporcionar, se han escogido los tres siguientes:

Tabla 3-6 Formato Mensajes NMEA GLL, HDT y VTG (Tomado de [45]).

GP	GLL	,llll.ll	,a	,yyyy.yy	,a	,hhmmss.ss	,A		*hh
		Latitud	N/S	Longitud	E/W	Hora	Estado		Checksum
GP	HDT	,x.x		,T			*hh		
		Rumbo, en grados		Verdadero			Checksum		
GP	VTG	,x.x	,T	,x.x	,M	,x.x	,N	,x.x	,K	*hh
		Made good, °	Verdadero	*Made good,* °	Magnético	Velocidad	Kn	Velocidad	Km/h	Checksum

El mensaje *GLL* proporciona información de situación GPS de la lancha, que será la situación por la cual se guiará la aproximación y el aterrizaje. Las coordenadas que proporciona el mensaje *GLL* llegan hasta las 4 décimas de grado (18cm de arco de círculo máximo) por lo que no introducen un error considerable.

El mensaje *HDT* proporciona la información de rumbo verdadero, que el *plotter* recibe a su vez de la giroscópica de la lancha. El rumbo verdadero proporciona una referencia útil de la proa que servirá para calcular rutas de aproximación relativas y la corrección de la situación de la antena GPS a la toldilla de la lancha, donde deberá producirse el aterrizaje.

El mensaje *VTG* permitirá conocer la velocidad sobre fondo de la lancha, permitiendo calcular la velocidad adecuada para la interceptación.

El procesado de las sentencias debería ser relativamente sencillo conocido el formato. Sin embargo surge una complicación importante como consecuencia del uso de la red *Navnet*. La red *Navnet* es una red creada para sistemas propios de la empresa de instrumentos de navegación *Furuno*. Con la intención de impedir el acceso a la red a dispositivos ajenos a la empresa, los instrumentos de la casa *Furuno* añaden una cabecera especial a los mensajes *NMEA*, que solo es conocida por sus propios sistemas.

Para estudiar ésta cabecera se crea una pequeña función que se conecte al puerto *UDP* que está recibiendo la información y genere un fichero de texto, en el que se almacenen los mensajes entrantes para su posterior análisis, obteniendo el siguiente resultado:

```
   ▯  ▯
$GPGLL,4223.8065,N,842.4897,W,225444,A,*1D$GPVTG,054.7,T,034.4,M,005.5,N,01
0.2,K*48$GPHDT,123.456,T*00
   ▯  ▯     $PFEC,GPint,ast01*13

   $PFEC,idfnc,R,*08
```

Ilustración 3-67 Mensaje de la red *Navnet* (Captura de Pantalla).

Como se puede observar, la red *Navnet* introduce una cabecera de caracteres no imprimibles que son los que impiden la compatibilidad de equipos *Furuno* con otras marcas. Además introducen un mensaje propio "`$PFEC,idfnc,R,*08`", con esta misma intención.

Para poder extraer la información útil de este mensaje, será necesario eliminar esta cabecera y este mensaje adicional. Para ello se crea la siguiente función, que utiliza las herramientas para manipular "*strings*" o cadenas de texto que proporciona *Python*.

```
def datos_lancha(data):
data, addr = sock.recvfrom(1024)
msg=data
msg=data[9:]
  if msg.find("$PFEC")==-1:
       flag1=msg.find("$GPGLL")
       flag2=msg.find("$GPVTG")
       flag3=msg.find("$GPHDT")
       GLL=msg[flag1:flag2]
       VTG=msg[flag2:flag3]
       HDT=HDT.split(",")
       GLL=GLL.split(",")
  VTG=VTG.split(",")
```

Esta breve función es una simplificación de la función que se utiliza para extraer los datos. En primer lugar vuelca el mensaje en la variable `"msg"` y elimina los 9 primeros caracteres, que son los que corresponden a la cabecera que añade *Navnet*. En la cadena de texto resultante se realiza una búsqueda del "*sentence formatter*" del mensaje añadido de *Navnet*. Si no se encuentra, se considera que es el mensaje correcto y se procede a su separación en los diferentes mensajes que lo componen. Para ello, se realiza una búsqueda de cada una de las cabeceras, almacenando en qué posición se encuentra cada una de ellas con la función `"find"`, a continuación se divide la cadena en los tres sub-mensajes que la componen, el *GLL*, el *VTG* y el *HDT*. Estos mensajes se convierten entonces en una "lista", un tipo de "*array*" que permite almacenar todo tipo de variables. Esta separación se realiza mediante el comando `"split(",")"` , que establece como elemento divisor las comas que separan los datos en un mensaje NMEA. De esta forma se han conseguido crear tres listas `HDT[], GLL[]` y `VTG[],` cuyos datos están encapsulados y organizados, siendo de fácil acceso. De esta manera, si quisiéramos pedir la latitud de un mensaje, solo tendríamos que llamar a `GLL[1]`, siendo "1" la posición que ocupa el valor de la latitud.

3.7.3 Filtrado de la señal de GPS

A pesar de que tanto la lancha de instrucción como el *UAV* cuentan con receptores GPS con un alto grado de precisión, se creyó conveniente la implementación de un filtro GPS que redujera los efectos de posibles errores de posición, y suavizara las trayectorias en las fases finales de aproximación para el aterrizaje.

Tras un breve periodo de investigación, se llegó a la conclusión que el filtro más adecuado para una aplicación de este tipo es el "*Filtro Kalman"*. El *Filtro Kalman* es un potente y complejo filtro predictivo que compara situaciones obtenidas mediante los sensores con situaciones futuras hipotéticas calculadas en función de datos anteriores. Este filtro se basa en un algoritmo desarrollado por *Rupert Kalman*, que permite estimar el estado no medible de un sistema dinámico lineal a partir de mediciones de ruido blanco [46].

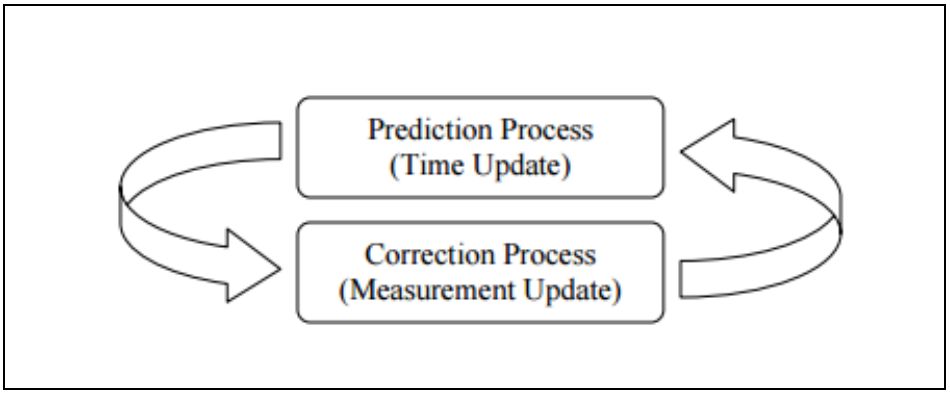

Ilustración 3-68 Ciclo del Filtro de Kalman (Tomado de [46]).

No obstante, la complejidad del filtro y la limitación temporal del proyecto obligaron al estudio de la implementación de filtros más sencillos, como el filtro de mediana, la media móvil ponderada o el filtro de paso bajo.

Se crean pues dos funciones de *Python* con el algoritmo de cada uno de los filtros:

```
def median3_filter(pos0, pos1, pos2):
    if pos0.lat<=pos1.lat and pos0.lat<=pos2.lat:
        if pos1.lat<=pos2.lat:
            middle_lat=pos1.lat
        elif pos1.lat>pos2.lat:
            middle_lat=pos2.lat
    elif pos1.lat<=pos0.lat and pos1.lat<=pos2.lat:
        if pos0.lat<=pos2.lat:
            middle_lat=pos0.lat
        elif pos0.lat>pos2.lat:
            middle_lat=pos2.lat
    elif pos2.lat<=pos0.lat and pos2.lat<=pos1.lat:
        if pos0.lat<=pos1.lat:
            middle_lat=pos0.lat
        elif pos0.lat>pos1.lat:
            middle_lat=pos1.lat
```

Este segmento de código representa la implementación de un filtro de mediana para la latitud de una serie de posiciones GPS en *Python*. Es un filtro no lineal sencillo, muy utilizado en el procesado de imágenes digitales, que mediante una serie de comparaciones lógicas permite seleccionar el valor mediano de una muestra *n* de valores (Ilustración 3-70).

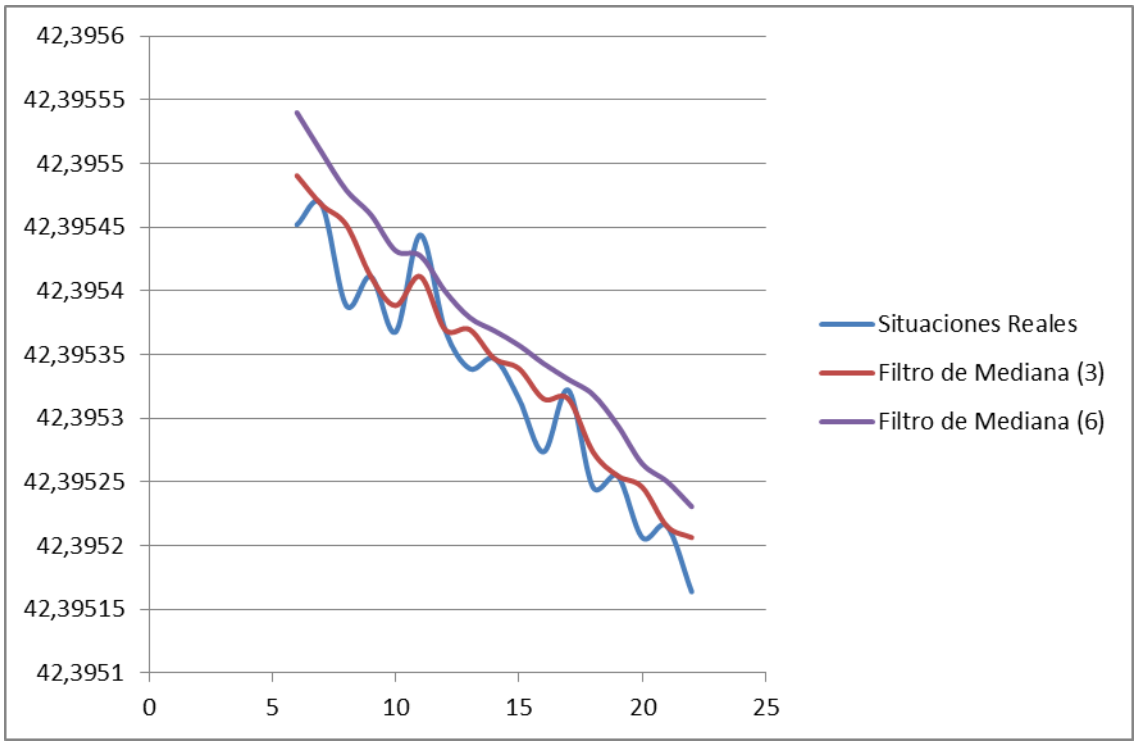

Ilustración 3-69 Ejemplo Filtro de Mediana (Autoría Propia).

Este tipo de filtros son buenos para eliminar valores extremos puntuales y suavizar una trayectoria. Es un filtro adecuado para ambientes con mucho ruido. En la Ilustración 3-70 se puede observar un ejemplo de dos filtros de mediana basados en una trayectoria real grabada en la explanada. Como se puede apreciar, el filtro de orden 3, reduce las fluctuaciones instantáneas de la trayectoria. Sin embargo el filtro de orden 6 es un filtro demasiado agresivo, que consigue una trayectoria muy continua, pero se aleja mucho de la realidad [47] .

Por otro lado se encuentra el filtro de media móvil ponderada. Una implementación simple de un filtro de media móvil ponderada es la siguiente:

```
    def moving_av_filter(location0, location1, location2):
        average_lat=location2.lat*0.2+location1.lat*0.3+location0.lat*0.5
        average_lon=location1.lon*0.2+location1.lon*0.3+location0.lon*0.5
    average_location=LocationGlobalRelative(average_lat,average_lon,vehic
    le.location.global_relative_frame.alt)
        return average_location
```

En este ejemplo se realiza una media móvil ponderada de una muestra de tres valores, de entre los cuales se le concede más peso a los valores más recientes. Es un filtro de implementación muy sencilla que suaviza ligeramente las variaciones. A diferencia que el filtro de mediana, el filtro de media móvil ponderada no desecha ningún valor, pero permite la reducción de los efectos de los valores extremos mediante el cálculo de la media entre él y los valores anteriores

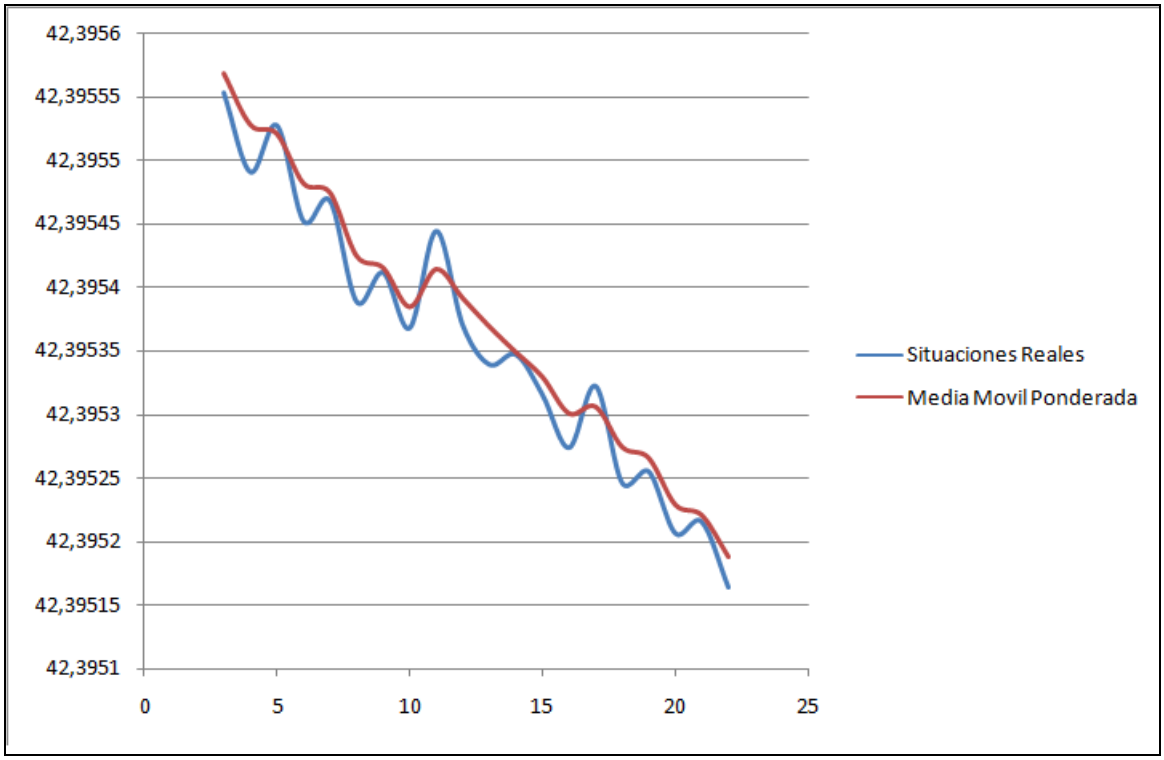

Ilustración 3-70 Ejemplo Filtro de Media Móvil Ponderada (Autoría Propia).

Mediante la aplicación de esta clase de filtros, es posible conseguir una etapa final de aproximación más suave y resistente a errores puntuales.

3.7.4 Script del Sistema de Control

El control del vehículo se lleva a través de un *script* en el lenguaje de programación *Python*, por tratarse del lenguaje en que se encuentran redactados los mensajes *Mavlink*, que se utilizarán para comunicarse con el *UAV*. A lo largo de este *script* se utilizan las funciones que han sido descritas ya anteriormente, como la sentencia de conexión o la extracción de datos de las lanchas.

El código completo se encuentra en el *Anexo II: Código Fuente del sistema de control*". Se encuentra estructurado de la siguiente forma:

1. *Import functions from Python and* Dron*kit*: se importan las funciones de Dron*kit* y las propias de *Python* que se utilizarán a lo largo del *script*.
2. *Connection sequence*: realiza la conexión inicial con el servidor *Mavproxy* en la *Raspberry*.
3. *Connection to lancha server*: se conecta al socket al cual se envía la información sobre las lanchas de instrucción.
4. *Constants definition*: se definen una serie de constantes que regulan el comportamiento del *UAV* en los diferentes modos, y les otorga su valor por defecto, aunque podrán cambiarse más adelante
5. *Functions definition*: se definen las funciones propias creadas para el programa.
    a. `download_vehicle_commands()`: permite acceder a los parámetros actuales del vehículo.
    b. `show_home()`: muestra en pantalla la situación actual de *Home*, importante si se va a utilizar el modo *Return to Launch*
    c. `battery_level_minutes()`: calcula aproximadamente el tiempo restante de batería en función de su nivel y el tiempo estimado de duración.

d. `write_position(lat, lon, alt)`: permite al usuario introducir una situación GPS cuando es llamada y ésta es traducida a un formato que el *UAV* pueda comprender
e. `get_distance_meters(position1, position2)`: realiza un cálculo de ortodrómica aproximado entre dos coordenadas y devuelve la distancia en metros entre ellos. Será utilizado como condición de llegada en muchas ocasiones.
f. `time_to_destination(location1, location2, time)`: calcula el tiempo de recorrido entre dos puntos en función de una velocidad.
g. `arm_and_takeoff(altitude)`: realiza una secuencia completa de comprobación, armado y despegue simple hasta una determinada altura.
h. `goto_vector(demora, distancia)`: permite introducir movimientos al *UAV* basados solo en un rumbo y una distancia.
i. `datos_lancha(data)`: realiza todo el procesado de los mensajes recibidos desde las lanchas de instrucción y devuelve información de latitud, longitud, velocidad, rumbo y tiempo.
j. `punto_aproximacion aterrizaje(pos_lancha, rumbo, distancia, marcacion)`: define el punto de aproximación inicial para la secuencia de aterrizaje en una lancha de instrucción.
k. `vel_aproximacion(vel_lancha, vel_rel)`: calcula la velocidad absoluta del *UAV* para las fases de aproximación dependiendo de la velocidad de la lancha y la velocidad relativa deseada.
l. `correccion_toldilla(posición, rumbo)`: realiza la corrección desde la antena GPS de la lancha a la toldilla, que será donde aterrizará el *UAV* en función de su posición y rumbo.
m. `median3_filter(pos1, pos2, pos3)`: filtro de mediana de orden 3 que devuelve la situación mediana entre tres puntos.
n. `moving_av_filter(pos1, pos2, pos3)`: calcula la media móvil ponderada entre tres puntos sucesivos

6. *Waypoints definition*: se definen una serie de puntos de interés dentro de la explanada de la Escuela Naval Militar y la pista de obstáculos para realizar pruebas. Los puntos están definidos siguientes mapas.

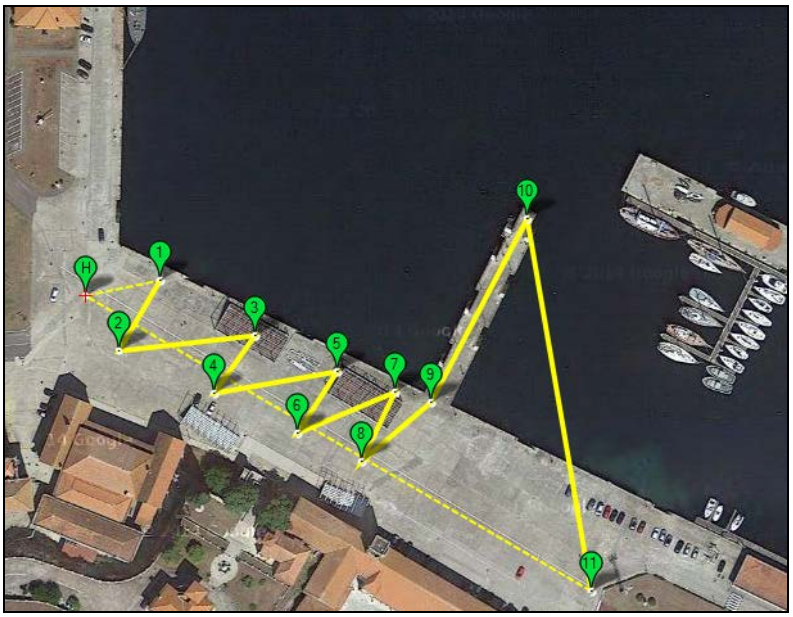

Ilustración 3-71 Puntos de interés en la explanada (Captura de Pantalla).

Ilustración 3-72 Puntos de interés pista militar (Captura de Pantalla).

7. *Menu Declaration*: se definen y se imprimen en pantalla las opciones con las que cuenta el menú. Este comando se encuentra dentro de un bucle, de manera que al terminar de realizarse una acción, el programa vuelva a imprimir el menú. Es un menú fácilmente expandible conforme se vayan creando nuevas opciones. A día de hoy el *script* cuenta con las siguientes opciones:
    a. *1-Simple take off and RTL*: despega hasta una determinada altura y aterriza sobre la posición home.
    b. *2-Simple take off, go to and land*: trayecto simple entre dos puntos con despegue en el inicial y aterrizaje en el segundo.
    c. *3-Freestyle mode*: permite ordenar al *UAV* comandos puntuales, como aterrizaje, despegue, cambio de altura o movimiento entre dos puntos. Estos comandos pueden realizarse en cualquier orden y pueden ser interrumpidos por comandos posteriores.
    d. *4-Lancha ATOL sequence*: secuencia de aproximación y aterrizaje en las lanchas de instrucción.

Este programa es fácilmente expandible, y se pretende que las funciones desarrolladas en él sirvan de apoyo en futuros proyectos. Los límites de los programas que pueden desarrollarse dependen únicamente de la imaginación del programador.

3.7.5 Prueba de capacidad ATOL. Simulada

La capacidad ATOL (despegue y aterrizaje automático) del UAV ha quedado verificada para localizaciones de despegue y aterrizaje estáticas, sin embargo estas pruebas, por restricciones climatológicas, no se pudieron llevar a cabo para ubicaciones dinámicas (lanchas de instrucción).

Se realizaron todos los preparativos pertinentes para la prueba, pero la disponibilidad de las lanchas, el horario del alumno y el inclemente tiempo gallego pasaron por todas las combinaciones necesarias de manera que fuera imposible llevarla a cabo.

En su lugar se realizaron simulaciones que buscaron demostrar el funcionamiento de cada uno de los elementos necesarios para la prueba.

En primer lugar, se creó un *script* en *Python* que simulara el comportamiento de la lancha de instrucción a la hora de desarrollar los programas:

```
"""SIMULADOR DE LANCHA DE INSTRUCCION"""
import socket, time

UDP_IP = "10.1.1.1"
UDP_PORT = 10021
i=0
MESSAGE = datos1[i]
while True:
  print "UDP target IP:", UDP_IP
  print "UDP target port:", UDP_PORT
  print "message:", MESSAGE[i]

  sock = socket.socket(socket.AF_INET, # Internet
                       socket.SOCK_DGRAM) # UDP
  sock.sendto(MESSAGE[i], (UDP_IP, UDP_PORT))
  i=i+1
  time.sleep(0.5)
```

Este pequeño programa envía al puerto *UDP "10.1.1.1:10021"* (puerto al que estaba previsto que llegaran los mensajes *NMEA*) una serie de mensajes de posición, rumbo y velocidad registrados en la lancha y guardados en la lista *"datos1[]"*. Esta simulación permitió comprobar que los mensajes se recibían y analizaban de forma correcta por parte del script de control, y que el *UAV* recibía las órdenes pertinentes, aunque no se llegó a probar el programa en movimiento.

De la misma forma, accediendo a las lanchas de instrucción, se comprobó la conexión de la *Raspberry Pi* encargada de la inyección de los mensajes de la lancha de instrucción en la red mediante la siguiente función:

```
"""Prueba de Recepción"""
from dronkit import connect, VehicleMode, LocationGlobalRelative,
LocationGlobal
import time, math
import socket
UDP_IP = "10.1.1.1"
UDP_PORT = 10021

sock = socket.socket(socket.AF_INET, # Internet
                         socket.SOCK_DGRAM) # UDP
sock.bind((UDP_IP, UDP_PORT))

def datos_lancha(data):#Devuelve(lat,lon,speed,heading,time), si el
mensaje no contiene el rumbo, lo devuelve como False. No se le puede
introducir una secuencia que no sea GLL-VTG-HDT
  HDT=False
  msg=data[8:]
  if msg[0]=="$":
          flag1=msg.find("$GPGLL")
          flag2=msg.find("$GPVTG")
          flag3=msg.find("$GPHDT")
          GLL=msg[flag1:flag2]
          if flag3== -1:
                  VTG=msg[flag2:]#Separamos mensaje de velocidad
          else:
                  VTG=msg[flag2:flag3]#Seapramos mensaje de velocidad
          if not flag3== -1:
                  HDT=msg[flag3:]
                  HDT=HDT.split(",")
                  heading=HDT[1]#Extraemos rumbo si el paquete de datos lo
contiene
```

```
                GLL=GLL.split(",")
                lat=GLL[1]
                lat_degrees=float(lat[:2])
                lat_minutes=float(lat[2:])/60
                lat=lat_degrees+lat_minutes
                if GLL[2]=="S":
                        lat=lat*-1
                lon=GLL[3]
                lon_degrees=float(lon[:1])
                lon_minutes=float(lon[1:])/60
                lon=lon_degrees+lon_minutes
                if GLL[4]=="W":
                        lon=lon*-1
                time=float(GLL[5])

                VTG=VTG.split(",")
                speed=float(VTG[7])/3.6
                if HDT!=False:
                        return lat,lon,speed,heading,time
                elif HDT==False:
                        return lat,lon,speed,False,time


        while True:
          data, addr = sock.recvfrom(1024)
          msg=data
          if float(msg.find("$PFEC"))==-1:

          lat_lancha,lon_lancha,speed_lancha,heading_lancha0,time_lancha=datos_lan
        cha(msg)
                pos_lancha=LocationGlobalRelative(lat_lancha,lon_lancha,15)
                print "%s" % pos_lancha
                if heading_lancha0!=False:
                        heading_lancha=heading_lancha0
```

Realizando la conexión del ordenador con el Sistema de Control con la red, se comprueba que los mensajes son recibidos a través de ella sin problemas y pueden ser analizados.

También se aplicó al fichero de mensajes grabado de la lancha el filtro de media móvil y el de mediana diseñados. La lancha de instrucción proporcionaba una situación precisa con pocas fluctuaciones, por lo que el comportamiento de los dos filtros fue muy similar, con unas variaciones mínimas de entre 20 y 50 cm entre situaciones. Teniendo en cuenta que el tamaño de la zona que permitiría el aterrizaje en las lanchas de instrucción es de 5x4 metros

Estos datos hacen sospechar que la aplicación puede ser posible utilizando solo los datos de GPS siempre y cuando las lanchas se desplacen a baja velocidad.

Habiendo probado la inyección de mensajes en la red, su recepción, su análisis, su filtrado y su transformación a órdenes para el *UAV*, lo único que quedó pendiente fue llevar la prueba final a cabo. Para esta prueba se decidió que por seguridad se le introduciría un error fijo a la posición que enviaran las lanchas. De esta manera se podría simular toda la secuencia con menos peligro en la explanada. Se trataría de una prueba fidedigna, ya que incorporaría todos los elementos y fases de la aproximación real, con la salvedad de que el aterrizaje se produciría en un punto de la explanada, que representaría una lancha imaginaria.

Esta prueba permitiría corregir defectos mínimos en el programa de aterrizaje, ajustar sus parámetros de forma que las velocidades y distancias definidas fueran las óptimas y comprobar si la

precisión que proporcionan ambos GPS (el de la lancha que proporciona información, y el del *UAV* que se posiciona en consecuencia), serían suficiente para una aproximación tan delicada en un espacio tan pequeño.

La realización de la prueba efectiva del aterrizaje en las lanchas quedará como una línea futura de este proyecto.

4 RESULTADOS / VALIDACIÓN / PRUEBA

"Si usted no tiene éxito al principio, intente, intente."

William Claude Fields

4.1 Plataforma

4.1.1 Configuración Final de Pesos

Por la limitación temporal del Trabajo de Fin de Grado, no se pudo equipar al *UAV* con todos los dispositivos descritos en el diseño inicial, por lo que la configuración final es ligeramente diferente de la prevista.

Ilustración 4-1 Configuración Final 1 del *UAV*.

Se consideraron elementos prescindibles la cámara *IP*, y el sonar altímetro. La ausencia de la cámara *IP* convierte a la *Raspberry* en el único dispositivo conectado a la red interna del *UAV*, eliminando por tanto la necesidad de un "*switch*". Por otra parte, fue necesario añadir un elevador de voltaje para la alimentación de la antena *WIFI*. Teniendo estas variaciones en cuenta, se obtiene la siguiente carga de pago:

Elemento	**Peso**
Controladora de vuelo *Pixhawk* v.2.4.5	38 gr
Módulo GPS *Ublox* M8N para *Pixhawk* 2.4.5.	26 gr
Ubiquity *Picostation* MH2	100 gr
Raspberry Pi 2 Modelo B	45 gr
APM/*Pixhawk* Wireless Radio Module	17 gr
Regulador de Voltaje DC DROK Boost	57 gr
Ez UHF 8 Channel Rx	18 gr
Carga de pago final	**301 gr**

Esta reducción de la carga de pago llevó a considerar la incorporación de una segunda batería en paralelo, resultando en la creación de dos configuraciones de vuelo diferentes (Tabla 4-1).

Tabla 4-1 Configuraciones finales de vuelo del *UAV* (Autoría Propia)

	Configuración 1: 1 Batería	**Configuración 2: 2 Baterías**
Peso total	1918 gr	2338 gr
Relación Empuje/Peso	2,3:1	1,9:1
Autonomía	10 min	16 min

La nueva configuración aumenta la autonomía del *UAV* en un 60% a cambio de una relación empuje/peso menor, que reducirá la maniobrabilidad del multicóptero, aunque manteniéndose siempre dentro de los márgenes recomendables.

4.1.2 Distribución de los elementos

A pesar de que una distribución exacta de los pesos no es estrictamente necesaria, un reparto equilibrado de las cargas favorecerá la maniobrabilidad del vehículo y el reparto equitativo de los esfuerzos entre todos los rotores.

Para ello se busca una distribución que permita el acceso a cada una de las piezas y anule en la medida de lo posible los momentos estáticos con respecto al centro de gravedad del *UAV*, resultando en la siguiente distribución:

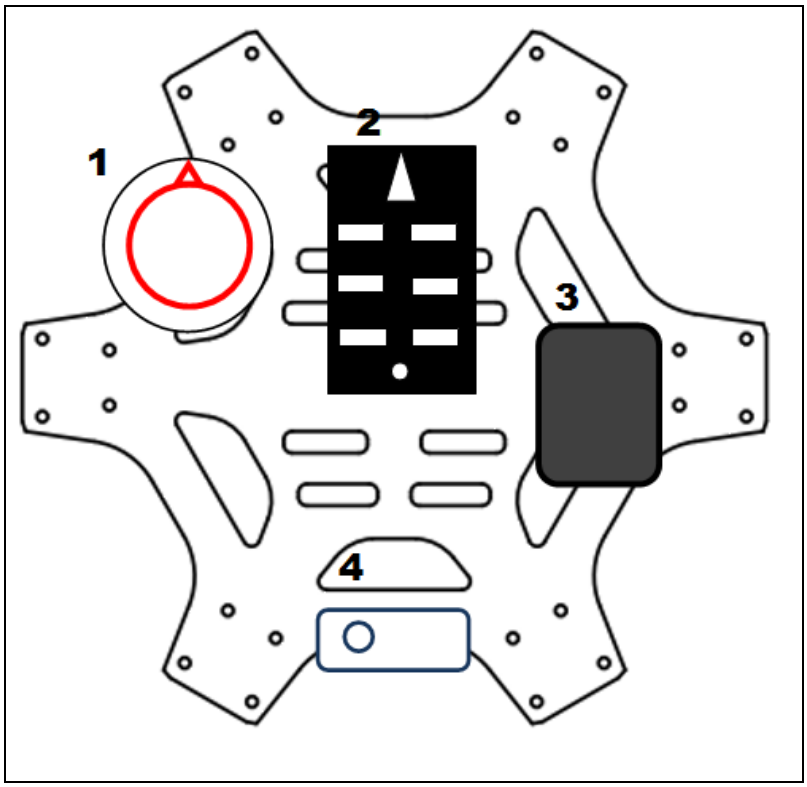

Ilustración 4-2 Distribución panel superior: 1-Módolo GPS; 2-*Pixhawk*; 3-Elevador de Tensión; 4-*Picostation* (Autoría Propia).

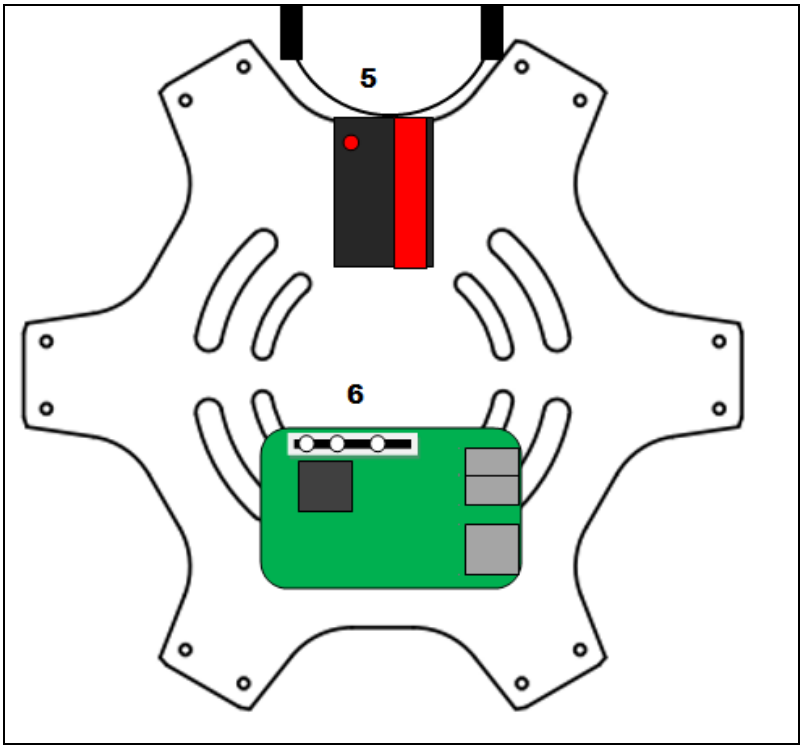

Ilustración 4-3 Distribución panel inferior: 5-Rx *EzUHF*; 6-*Raspberry Pi* (Autoría Propia).

En esta distribución la o las baterías se encuentran acopladas en la cara inferior del panel inferior, aprovechando el margen que proporcionan las patas. Esto permite su fácil acceso, baja el centro de gravedad del vehículo (aumentando su estabilidad), y las separa del resto de componentes eléctricos como medida de seguridad.

4.1.3 Distribución eléctrica

La única fuente de alimentación que incorpora el *UAV* es la batería *Lipo* de 11,1V. Desde esta batería se alimentan los siguientes dispositivos:

Tabla 4-2 Distribución eléctrica del vehículo (Autoría Propia).

Dispositivo	*Amperaje*	*Voltaje*	*Método de suministro*
Servomotores	Variable, hasta 72A	Variable, hasta 11.1V	Desde la batería, a través de los ESC
Pixhawk	200mA	5V	A través del "Power Module". Reduce de 11.1V a 5V
Raspberry	600mA	5V	Puerto de telemetría de la *Pixhawk*
Picostation	1A	24V	Inyector POE, con un paso de elevación de voltaje de 11.1V a 24 V

El consumo de los dispositivos electrónicos no es relevante comparado con el de los servomotores. Serán ellos los que limiten la autonomía del vehículo.

4.2 Sistema de Control

El Sistema de Control resultante está compuesto por tres segmentos bien diferenciados, el *UAV*, la red *MANET* y el *script* de control. En este apartado se realizará una breve explicación de cada uno de estos segmentos, lo que permitirá conocer el funcionamiento global del Sistema de Control.

4.2.1 Sistema de Control. UAV

Este segmento se refiere no al vehículo en sí, sino a los elementos que incorpora la plataforma que hacen posible su incorporación en el Sistema de Control. A bordo del *UAV* existen tres elementos importantes que se comunican entre ellos y son los que conforman el segmento del *UAV* dentro del Sistema de Control. Estos elementos son la controladora *Pixhawk*, la *Raspberry Pi* y la *Picostation*.

En primer lugar se encuentra la controladora de vuelo *Pixhawk*. Este dispositivo recoge la información de todos sus sensores para conocer la situación espacial del vehículo y realiza correcciones constantes en base a ellos. Controla la velocidad de los rotores de forma que el vector compuesto por todos produzca el estado de movimiento ordenado.

La *Raspberry Pi* es la puerta de acceso a la controladora de vuelo, y por tanto al control del vehículo. Incorpora un software de control de *UAV* llamado *Mavproxy*. Este software de control actuará de servidor intermedio entre la controladora de vuelo y todos aquellos operadores que quieran acceder a ella. De forma que comunicándose con este servidor, se podrá obtener información de los sensores del *UAV* y controlarlo.

Finalmente, la *Picostation M2* es el modo que tiene el *UAV* de comunicarse con el exterior. La antena está configurada de forma que pueda acceder a la red *MANET* de las lanchas de instrucción de la ENM. A través de esta red la *Picostation* anunciará su red interna, en la cual se encuentra la *Raspberry Pi*, permitiendo la conexión a ella y por tanto al servidor *Mavproxy*.

Entre estos tres elementos se produce una comunicación bidireccional en el protocolo de mensajes *Mavlink*, que hace posible la adquisición de datos del *UAV* y sus sensores, así como la transmisión de órdenes de control que serán traducidas por la controladora a las señales electricas a los motores que resulten en el cumplimiento de la órden.

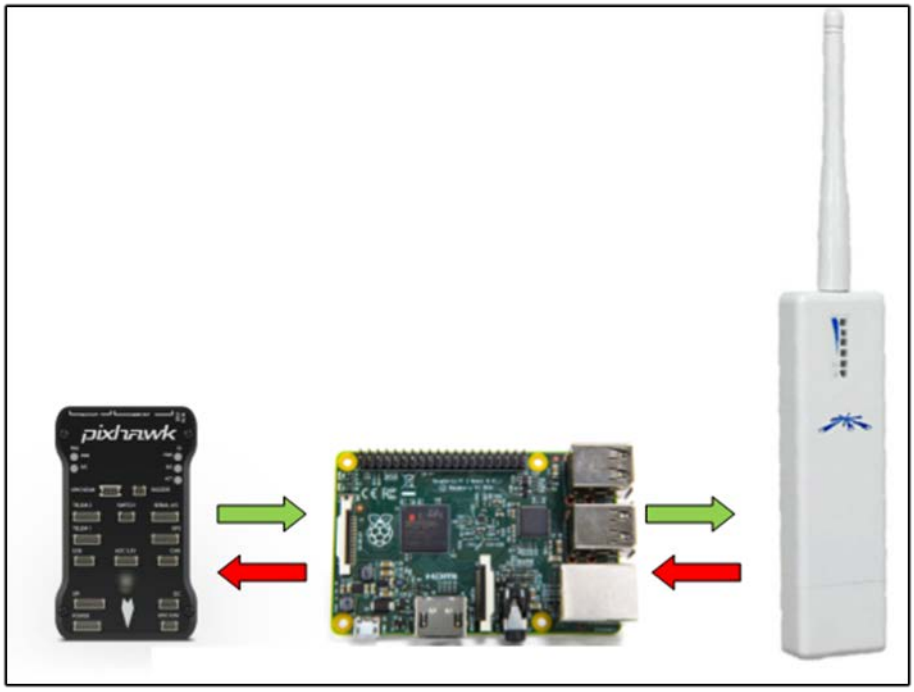

Ilustración 4-4 Sistema de Control. Segmento *UAV* (Autoría Propia).

4.2.2 Sistema de Control. Red MANET

Este segmento está compuesto por la red *MANET* de las lanchas de instrucción de la ENM. Esta red se instaló en las lanchas de instrucción durante el desarrollo del TFG "*Despliegue de una Red Móvil ad hoc en las lanchas de instrucción de la ENM*" por el A.F. López Porto-Andión. Este segmento cumple dos objetivos, la comunicación con el *UAV* y la adquisición de datos de las lanchas de instrucción. La red *MANET* es una red mallada especializada en nodos móviles, que permite la conexión entre ellos mediante una actualización cíclica de las rutas de encaminamiento (*OLSR*). Los nodos de esta red son las 4 lanchas de instrucción, el *UAV* y una antena sectorial situada en tierra.

Cada uno de estos nodos tiene una red interna cuyos elementos son anunciados a la red *MANET*, permitiendo la interconexión entre dispositivos internos de cada red. La red interna del *UAV* está compuesta únicamente por la *Raspberry*, que será anunciada a la red externa. De la misma forma, un ordenador con el *script* del Sistema de Control se conectado a la red interna de cualquiera de los nodos será anunciado accesible desde cualquier punto de la red. Esto permite que el control del *UAV* se pueda llevar a cabo desde cualquier punto de la red. Además, la actualización de rutas permite que el *UAV* pueda desplazarse por dentro de la zona de cobertura de las diferentes lanchas y mantener la conexión en todo momento.

Por otro lado, accediendo a la red de sensores interna de las lanchas (*Navnet*), extrayendo la información deseada, e inyectándola en la red *MANET*, se consigue un acceso remoto a estos datos, que serán utilizados por el script del Sistema de Control para el desempeño de funciones que impliquen la interactuación entre el *UAV* y las lanchas.

4.2.3 Sistema de Control. Script de control

Es el segmento final del Sistema de Control. Se trata de un *script* o archivo de procesamiento por lotes, desarrollado en el lenguaje de programación *Python* que permite el control del *UAV* mediante el uso combinado de una serie de funciones.

Esta aplicación se conecta con el servidor *Mavproxy* instalado en la *Raspberry Pi* a través de la red *MANET*, mediante una conexión *UDP*. La conexión le permite acceder y alterar los atributos del vehículo, así como controlar su movimiento. Estas acciones se llevan a cabo a un alto nivel, gracias a las funciones incluidas en la *API* Dron*kit* de *3DR*, que permiten el control del *UAV* a través de comandos sencillos que traduce automáticamente a los mensajes *Mavlink* correspondientes.

La adquisición de datos del *UAV* y las lanchas y el control del vehículo, permiten llevar a cabo complejas secuencias pre programadas.

El código que forma el script se encuentra en el "*Anexo II: Código Fuente del sistema de control*" y a día de hoy permite al usuario realizar 5 acciones:

- Prueba simple de despegue y aterrizaje
- Trayecto simple entre dos puntos con despegue en el primero y aterrizaje en el segundo
- Modo de control libre, que permite introducir comandos individuales y tener un control directo del *UAV* a través de la red
- Secuencia *ATOL* en las lanchas de instrucción.
- Cambio de parámetros predefinidos

El script se ha desarrollado junto con todas sus funciones, dentro de un directorio propio de un intérprete móvil de *Python*, de forma que puede ejecutarse desde cualquier ordenador o incluso desde una memoria externa.

Además se ha querido conservar la capacidad de utilizar la emisora como medio de control de emergencia de manera que ésta domina sobre la señal recibida a través de la red.

4.3 Capacidad *ATOL*

Desgraciadamente, no se dispuso del tiempo necesario para llevar esta aplicación a la práctica por culpa de los múltiples factores que restringieron su puesta en marcha, sin embargo, la aplicación ha sido diseñada y sus diferentes elementos se han probado individualmente, de manera que con unos ajustes la capacidad *ATOL* en las lachas de instrucción podría convertirse en una realidad.

La capacidad *ATOL* del *UAV* representa la combinación de todas las herramientas obtenidas mediante la consecución de los objetivos intermedios del proyecto y la adición de nuevas funciones. Esta aplicación utiliza por una parte la capacidad de leer y fijar los atributos del vehículo, un control preciso de su movimiento a través de la red, y el desarrollo de modos de comportamiento complejos basados en la información obtenida de las lanchas de instrucción.

En este apartado se explica de forma secuencial y resumida como se afronta este desafío sin profundizar en el código (*Anexo II: Código Fuente del sistema de control).*

Esta aplicación está diseñada para lanchas estáticas o a baja velocidad. El aterrizaje se puede llevar a cabo desde tierra o desde otra lancha estática o a baja velocidad. La forma de llevarla a cabo consistió en su división en 4 sub etapas que regulaban diferentes procesos de la secuencia: despegue, aproximación inicial, aproximación de precisión y aterrizaje.

4.3.1 Etapa-1: Despegue

Esta etapa incluye la llamada inicial a la conexión al servidor que recibe la información de la lancha de instrucción. A través de esta conexión se obtienen los datos de las lanchas y se realizan una serie de comprobaciones.

En primer lugar se calcula la distancia a la que la lancha se encuentra del *UAV*, y que velocidad debe adquirir el vehículo para alcanzar a la lancha en base a una velocidad relativa entre el dron y la lancha preestablecida. Basándose en esos datos se realiza un cálculo aproximado del tiempo de vuelo. Comparando este tiempo de vuelo con el tiempo de autonomía restante del dron, que se conoce a partir de los atributos del vehículo, el sistema decidirá si la aproximación es viable o no, y en caso negativo, no permitirá efectuarla.

En caso de ser viable, el sistema imprimirá en pantalla información sobre la aproximación, como la distancia, el "*ETA*" (*Estimated Time of Arrival*) y la posición GPS de la situación de destino.

Tras esto el *UAV* realiza una secuencia de despegue simple, de ascenso vertical hasta una altura pre introducida. Se considera que el *UAV* puede realizar un despegue vertical desde la toldilla de una lancha de instrucción parada o a baja velocidad librando todos los obstáculos sin complicación. Alcanzada la altura ordenada, el *UAV* pide permiso para comenzar la segunda etapa.

4.3.2 Etapa-2: Aproximación inicial

En esta etapa el *UAV* busca posicionarse en el punto de comienzo de la aproximación de precisión. Este es un punto definido por una distancia y una demora relativas a la lancha de instrucción, de forma que la aproximación final siempre se realice desde la misma dirección, como medio de control. Para calcular esta posición relativa el Sistema de Control utiliza la información de rumbo y situación que le proporcionan los mensajes de la lancha. La marcación y distancia pueden cambiarse, pero en caso de no hacerlo, están predefinidas una distancia de 20 metros y una marcación de 135º, por considerarse la aleta de estribor la senda de aproximación final más segura.

Durante esta etapa el *UAV* se dirige al punto de aproximación con una velocidad relativa de 3 m/s, aunque puede regularse este valor. Para evitar constantes correcciones por parte del *UAV*, en esta trayectoria inicial la información sobre el paradero de la lancha y el punto de aproximación se actualiza con un intervalo regular de 10 segundos, también regulable. Una vez el *UAV* detecte que se encuentre dentro de un radio admitido de error del punto de aproximación, pedirá permiso para comenzar la tercera etapa.

4.3.3 Etapa-3: Aproximación de precisión

Esta junto con la etapa de aterrizaje es la etapa más delicada. En esta etapa el *UAV* parte de una posición conocida cercana a la lancha, a una altura de seguridad.

La posición de la lancha deja de transmitirse de forma bruta al *UAV*. Esta situación es ahora pasada por un filtro de mediana y corregida desde la antena GPS a la toldilla, zona en la que se pretende realizar el aterrizaje. Durante esta etapa la velocidad relativa del *UAV* con respecto a la lancha se reduce a 0.5m/s, y la frecuencia de refresco de la situación de la lancha se aumenta al máximo, cercano a los 2 Hz.

Una vez el *UAV* alcanza por primera vez la situación de descenso con un pequeño margen de error, se empiezan a recoger situaciones del *UAV* y comienzan a pasarse por el mismo filtro de mediana que la situación de descenso. Esta situación mediana del *UAV* se compara con la situación filtrada y corregida de la lancha de instrucción, de manera que si se aleja de esta más de un pequeño error admisible, corrija la posición y vuelva a colocarse dentro del radio admisible. De esta forma se consigue que el *UAV* reaccione de una forma más suave a los posibles saltos que pudiera dar el GPS en esta delicada fase final.

Una vez se encuentra bien posicionado en la situación de descenso, el *UAV* pide permiso para iniciar la última etapa.

4.3.4 Etapa-4: Aterrizaje

Esta última etapa se realiza dentro de un bucle que combina el modo *GUIDED* y el modo *LAND*. El modo *LAND* proporciona un descenso vertical a una velocidad suave y constante que puede ser regulada, y el modo *GUIDED* se utiliza para corregir la posición y mantenerla dentro de unos límites.

Dentro de este bucle el *UAV* pasa a modo *LAND* siempre que se encuentre dentro de un radio mínimo de la situación filtrada y corregida de aterrizaje. Si el *UAV* detecta que su situación mediana se sale del radio de error admisible, pasará a modo *GUIDED* y corregirá su situación.

Este bucle se repetirá sucesivamente hasta que la controladora de vuelo detecte que a pesar de que ella reduce la potencia de los motores su altura no varía, momento en el que considerará que ha aterrizado y procederá a desarmarse

5 CONCLUSIONES Y LÍNEAS FUTURAS

"El futuro no está por venir. Ya llegó."

Philip Kotler

5.1 Conclusiones

5.1.1 Viabilidad del proyecto

La primera y más importante conclusión que se extrae es que el proyecto es viable. A pesar de que existen sistemas de control similares, no se ha encontrado ninguna documentación que recoja un sistema como este, por lo que no había garantías de que el sistema fuera a funcionar. Se ha demostrado que es posible diseñar un sistema de control integrado en una red superior que permita la comunicación entre los diferentes dispositivos a través de piezas comerciales de fácil adquisición.

5.1.2 Objetivos

Los objetivos propuestos han sido demasiado ambiciosos para el breve periodo de tiempo asignado. Este proyecto abarca una gama muy amplia de disciplinas, desde mecánica hasta informática, pasando por electrónica y telecomunicaciones. Hubiera sido interesante dividir este proyecto en dos partes, una encargada del diseño, montaje y desarrollo del vehículo, y otro exclusivamente encargado del sistema de navegación, ya que la necesidad inicial de diseñar y montar la plataforma ha limitado en gran medida el alcance del proyecto. De esta forma se hubiera conseguido profundizar más en cada uno de estos dos aspectos.

Con respecto a los objetivos planteados al inicio del proyecto:

- Diseño y construcción de la plataforma: este objetivo se considera cumplido. A pesar de que se trata un objetivo siempre mejorable, ya que se puede profundizar en el diseño y perfeccionarlo, la plataforma construida ha demostrado ser capaz de llevar a cabo un vuelo estable y seguro, así como de incorporar los dispositivos planeados para ella.
- Vehículo autónomo: este objetivo se considera cumplido. Las pruebas han demostrado que el vehículo es capaz de ejecutar rutas pre-programadas e incluso dinámicas de forma autónoma, con seguridad y precisión
- Control del *UAV* a través de una red *WIFI*: este objetivo se considera cumplido. El *UAV* se ha integrado efectivamente en la red *WIFI* de las lanchas a través de sus dispositivos, que han permitido su control y monitorización a través de la red.

- Desarrollo del programa de control con capacidad *ATOL*: este objetivo se considera parcialmente cumplido. Si bien se ha sido capaz de programar un script con funciones de control, que incluyen entre ellas la capacidad *ATOL*. Esta última no ha podido llevarse a cabo en las lanchas de instrucción, por lo que queda como una línea a continuar de este proyecto, sin embargo, sí se ha logrado el despegue y aterrizaje automático en ubicaciones fijas.

5.1.3 Posibilidades

A pesar de haberse centrado en una aplicación muy concreta como es la capacidad *ATOL*, la capacidad de conexión que ofrece la red *MANET*, y el acceso al *UAV* desde múltiples dispositivos, puede servir de punto de partida para muchos futuros proyectos.

Lo más asombroso de este proyecto es la infinidad de posibilidades que ofrece. Una vez establecida la forma de conectarse al *UAV* y adquiridas las herramientas necesarias para controlarlo a través de un simple script, el único límite a las diferentes aplicaciones desarrollables es la imaginación del programador.

La extracción y uso de los datos proporcionados por los sensores de las lanchas de instrucción permite desarrollar muchos comportamientos asociados, como vuelos en formación, maniobras *"VERTREP"* (*Vertical Replenishment*), patrullas, grabación de maniobras etc.

Por otro lado, el hecho de tener una *Raspberry* embarcada a bordo del *UAV* proporciona la posibilidad de dotarle de comportamientos autónomos complejos sin la necesidad de una *GCS*, así como de incorporar toda clase de dispositivos asociados a ella.

El multicóptero es una plataforma increíblemente versátil que con la controladora de vuelo proporciona un medio estable al que equipar diferentes sensores. La plataforma diseñada es completamente configurable en función de las necesidades, y podrá ser equipada de nuevos sensores que mejoren sus capacidades.

5.1.3.1 Aplicación en la Flota

El multicóptero diseñado, podría suponer para los buques de la Armada un sensor increíblemente útil y de bajo precio. El disponer de 1 o 2 *UAV* de esta clase, que se puedan controlar desde las redes de a bordo y reciban información real de los sensores de una fragata podría permitir tener siempre un vehículo en el aire que facilitara las labores de vigilancia y reconocimiento del barco.

De la misma forma, podrían ser muy útiles durante los abordajes de los *TVR (Trozos de Visita y Registro)*, en los cuales la vigilancia de la cubierta del barco abordado y la grabación de los sucesos ocurridos es de vital importancia. El *UAV* podría ser un medio efectivo y barato de proporcionar seguridad al trozo y mantener una crónica del abordaje de cara a trámites legales.

Muchas de estas labores a día de hoy son llevadas a cabo por parte de helicópteros de la *FLOAN* (*Flotilla de Aeronaves*), que son medios mucho más caros y de mayor mantenimiento. La implantación de *UAV* para tareas sencillas próximas al buque, resultaría también en muchos casos en una reducción considerable de gastos en material y personal.

Algunos de los barcos de la Armada, como las fragatas de la clase Álvaro de Bazán, cuentan ya con redes *WIFI* de gran potencia, que permitirían utilizar el sistema diseñado en este trabajo con unos cambios mínimos.

5.1.4 Colaboración entre proyectos

Este proyecto no hubiera sido posible sin la estrecha colaboración con el equipo encargado del desarrollo de la red *MANET*. Los medios, información y la ayuda proporcionados por este equipo han resultado de vital importancia en el desenlace.

También fue de gran utilidad la información aportada por el proyecto de extracción de datos de los sensores de las lanchas, que ha facilitado su inclusión en la red y posterior análisis.

5.1.5 Autonomía

Una conclusión negativa, es la evidente limitación de autonomía del *UAV*. Sería conveniente buscar formas de reducir el peso de los sistemas que incorpora y buscar métodos de alimentación alternativos para prolongar su tiempo de vuelo.

5.2 Líneas futuras

La plataforma conseguida no ha cumplido todos los objetivos que se habían fijado inicialmente para ella, a pesar de que se disponía de los medios necesarios. Por ello, la primera y principal línea futura ha de ser la de culminar el único objetivo pendiente de este trabajo que es finalizar el desarrollo de la aplicación ATOL para su uso entre lanchas de instrucción. Dicho desarrollo puede proporcionar un gran servicio a la Escuela Naval. Además, en un futuro, la Armada podría extrapolar o extender su uso para dotar a los drones de capacidad ATOL entre unidades de la flota.

Por otro lado uno de los objetivos planteados para este TFG perfectamente conseguido era el de disponer de una plataforma con capacidad de expansión sobre la que desarrollar futuros trabajos. Existen infinitas posibilidades de mejora y proyectos relacionados con la tecnología adquirida. Algunas de las posibles líneas futuras que se han identificado a lo largo del proyecto son:

1. Integrar en los algoritmos de aterrizaje, concretamente en la etapa de descenso del UAV, los datos proporcionados por el Sonar *Maxbotix*.

2. Sería conveniente mejorar el *software* de control mediante la creación de una aplicación con un interfaz gráfico basado en la idea del *script* de control utilizado en este proyecto.

3. La inclusión de los datos de actitud de la lancha de instrucción (balance, cabezada y guiñada), así como los de desplazamiento (arfada, deriva y avance), como parte de la información que la lancha proporciona a la MANET sería muy interesante.
   En este mismo sentido, si las lanchas de instrucción proporcionasen información de velocidad y dirección del viento, el UAV podría tener en cuenta estos datos para efectuar una aproximación a la lancha de manera óptima y sin riesgos. Incluso, en caso de ser necesario, si las condiciones no permiten el aterrizaje, el dron podría decidir autónomamente cancelar la operación y volver a casa.

4. Estudio de la posible incorporación de un *UAV* de estas características en una fragata F-100. Una colaboración de la ENM con 31ª Escuadrilla de escoltas permitiría el perfeccionamiento de una plataforma similar a la diseñada en este proyecto encaminada a las necesidades específicas del barco. Si se consiguiese llevar a cabo esto repercutiría en un gran prestigio para la Escuela Naval Militar y el Centro Universitario de la Defensa.

5. Estudio de formas de aumentar la autonomía del *UAV*. La reducción de peso, o la búsqueda de fuentes de energía alternativas son opciones para buscar un mayor tiempo de vuelo de la aeronave.

6. Estudio de los GPS del *UAV* y las lanchas de instrucción y de la posible implementación de mejores filtros de posición y trayectorias, con el objetivo de mejorar la capacidad *ATOL* y la precisión de la situación del *UAV*. El uso de un potente filtro predictivo como el filtro *Kalman* podría mejorar considerablemente el comportamiento del *UAV*, especialmente en la delicada etapa de aterrizaje.

7. Desarrollo de procedimientos tácticos entre las lanchas de instrucción y prácticas de campo de infantería de marina que incorporen el uso del *UAV*. Una vez las capacidades del *UAV* hayan sido probadas lo suficiente y existan los métodos de seguridad necesarios sería una gran oportunidad de cara al adiestramiento de los alumnos de la Escuela Naval Militar el contar con un aeronave para practicar procedimientos de operaciones de vuelo. Por otro lado la Infantería de Marina Española ha adquirido recientemente drones similares al construido en este proyecto, y sería interesante medir las posibilidades de este *UAV* en operaciones de apoyo y reconocimiento.

8. Gracias a la experiencia obtenida en este TFG, sería muy interesante estudiar hasta qué punto podría dotarse a los drones Huginn X-1, recientemente adquiridos por Infantería de Marina [14], de la capacidad ATOL desde ubicaciones en movimiento. Esta característica permitiría que la Infantería realizase labores de reconocimiento de terreno sobre la marcha, es decir, mientras la unidad se despliega o desplaza.

9. Estudiar la viabilidad de un sistema de "*radio-beacon*" para aterrizajes de precisión. Existen sistemas de aterrizaje para *UAV* basados en una senda de aterrizaje producida por una señal radio o "*IR*" (*Infra Rojo*) que permiten un aterrizaje de precisión. Este *UAV* se podría beneficiar de esta clase de sistemas para conseguir un aterrizaje de mayor precisión.

10. Desarrollar un sistema óptico electrónico para navegación y aterrizajes de precisión. A través de una cámara instalada a bordo del *UAV* sería posible a través de un software de reconocimiento de imágenes identificar figuras sencillas que permitieran al *UAV* aterrizar con precisión en puntos reconocibles o seguir rutas pintadas de forma automática.

11. Desarrollo de un sistema que permita al *UAV* tomar y despegar desde el agua. El *UAV* cuenta ya con unos rudimentarios flotadores para casos de emergencia, pero sería interesante que el *UAV* contara con la capacidad de realizar de forma regular tomas y despegues desde el agua. Para ello sería importante buscar proteger la electrónica del agua.

12. Dotarle de un sistema de reconocimiento espacial basado en sonar, laser o radar que le permita detectar y esquivar obstáculos.

13. Adaptación e incorporación al *UAV* del radar desarrollado durante el TFG "*Fabricación y puesta en marcha de un radar FMCW de bajo coste en banda S*" por el A.N. Liaño Cuquerella [50]. Con esto se pretende conseguir la capacidad de transmitir la información obtenida por el radar a través de la red *MANET* de las lanchas de instrucción, actuando como una extensión de sus sensores, siguiendo el ejemplo de los helicópteros "*SH-60" (Sea Hawk)* de la 10ª Escuadrilla de aeronaves y su "*datalink*". Con este radar podría dotarse al UAV con capacidad de creación de imágenes SAR y reconocimiento e identificación de blancos.

6 BIBLIOGRAFÍA

"Quien lee mucho y anda mucho, sabe mucho y llega lejos."

Miguel de Cervantes Saavedra

[1] «Unmanned Aerial Systems Association,» [En línea]. Available: https://www.uavs.org/index.php. [Último acceso: 01 03 2016].

[2] «A brief History of the Drones,» The Nation, [En línea]. Available: http://www.thenation.com/article/brief-history-drones/. [Último acceso: 17 02 2016].

[3] A. Durán Ferreras, «MODELADO, CONTROL Y PERCEPCIÓN EN SISTEMAS AÉREOS AUTÓNOMOS,» [En línea]. Available: http://bibing.us.es/proyectos/abreproy/70314/fichero/Memoria. [Último acceso: 01 03 2016].

[4] «Historia de los Drones,» Tiki Tok, [En línea]. Available: http://www.tiki-toki.com/timeline/entry/496448/Historia-de-los-drones/. [Último acceso: 17 02 2016].

[5] «Review from Dron Predator,» Dronwars, [En línea]. Available: www.dronwars.net. [Último acceso: 17 02 2016].

[6] «Ambulance Dron T.U. Delft,» TU Delft, [En línea]. Available: http://www.tudelft.nl/en/current/latest-news/article/detail/ambulance-dron-tu-delft-vergroot-overlevingskans-bij-hartstilstand-drastisch/. [Último acceso: 18 02 2016].

[7] «Amazon Prime Air Delivery System,» Amazon, [En línea]. Available: www.amazon.com. [Último acceso: 17 02 2016].

[8] «Asociación Española de RPAS,» AERPAS, [En línea]. Available: http://www.aerpas.es/. [Último acceso: 29 02 2015].

[9] «Legislación Drones Gobierno de España, Seguridad Aérea,» Ministerio de Fomento, Gobierno de España, [En línea]. Available: http://www.seguridadaerea.gob.es/lang_castellano/cias_empresas/trabajos/rpas/default.aspx. [Último acceso: 18 02 2016].

[10] «Tokyo Anti-Dron Squad,» The Telegraph, [En línea]. Available: http://www.telegraph.co.uk/technology/2016/01/21/tokyo-police-are-using-drones-with-nets-to-catch-other-drones/. [Último acceso: 18 02 2016].

[11] M. d. Defensa, «Los Sistemas no Tripulados, Documentos de Seguridad y Defensa,» [En línea]. Available: http://www.defensa.gob.es/ceseden/Galerias/destacados/publicaciones/docSegyDef/ficheros/047_LOS_SISTEMAS_NO_TRIPULADOS.pdf. [Último acceso: 01 03 2016].

[12] «La Armada prueba con exito su nuevo "Dron" Scaneagle,» ABC, [En línea]. Available: www.abc.es. [Último acceso: 18 02 2016].

[13] «Proyecto Pelícano,» INDRA, [En línea]. Available: http://www.indracompany.com/sites/default/files/PELICANO_Esp_0.pdf. [Último acceso: 27 02 2016].

[14] «La Infaneria de Marina se adiestra en el funcionamiento de su nuevo dron,» Onemagazine, [En línea]. Available: http://www.onemagazine.es/industria-drones-infanteria-marina-entrenamiento-aeronave. [Último acceso: 27 02 2016].

[15] C. d. C. Ernesto Grueso García, «EMPLEO DE UAV EN LA ARMADA ESPAÑOLA, ¿CONCEPTO O CAPACIDAD MILITAR?,» *Revista General de Marina*, vol. Julio, pp. 65-76, 2010.

[16] «*Dronecode*, drones de código abierto bajo la fundación Linux,» MyComputer. [En línea]. Available: http://www.muycomputer.com/2014/10/13/*Dronecode*-drones-libres. [Último acceso: 16 01 2016].

[17] «DIY DRONES: The Leading Community for Personal UAV,» DIY DRONES, [En línea]. Available: http://diydrons.com/. [Último acceso: 15 01 2016].

[18] «MAVLink Micro Air Vehicle Communication Protocol,» QGroundcontrol, [En línea]. Available: http://qgroundcontrol.org/mavlink/start. [Último acceso: 18 02 2016].

[19] «Modos de Vuelo para Multicópteros,» Copter Ardupilot, [En línea]. Available: http://copter.ardupilot.com/wiki/flight-modes/. [Último acceso: 02 01 2016].

[20] «Common choosing a Ground Control Station,» Copter Ardupilot, [En línea]. Available: http://copter.ardupilot.com/wiki/common-choosing-a-ground-station/. [Último acceso: 15 02 2016].

[21] Copter.ardupilot.com, «Cuadcopters, Overview,» Ardupilot, [En línea]. Available: www.copter.ardupilot.com. [Último acceso: 21 Diciembre 2015].

[22] «Multi-Rotors, First-Person View, And The Hardware You Need,» Tom's Hardware, [En línea]. Available: http://www.tomshardware.com/reviews/multi-rotor-quadcopter-fpv,3828.html. [Último acceso: 02 Febrero 2016].

[23] «Electronic Speed Controllers,» RCModelsWiz.co.uk, [En línea]. Available: http://www.rcmodelswiz.co.uk/rc-guides/electric-rc-models-guide/electronic-speed-controllers-esc/. [Último acceso: 8 febrero 2016].

[24] «Copter Details,» Erle Robotics, [En línea]. Available: https://www.gitbook.com/book/erlerobotics/erle-robotics-erle-copter/details. [Último

acceso: 14 02 2016].

[25] «DJI Products,» DJI, [En línea]. Available: http://www.dji.com/. [Último acceso: 8 Febrero 2016].

[26] «TP-Link,» [En línea]. Available: http://www.tplink.com/mx/FAQ-499.html. [Último acceso: 08 Febrero 2016].

[27] «Emisora Futaba 10 Canales,» Aeromodelismo Serpa, [En línea]. Available: https://www.google.es/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjJnoutwonLAhWJhhoKHbGCCm0QjB0IBg&url=http%3A%2F%2Fwww.aeromodelismoserpa.com%2Ffutaba-channel-24ghz-p-15427.html&psig=AFQjCNFEUl6Hne5PQAfg8o4IsTbqbhNDCA&ust=1456166. [Último acceso: 18 01 2016].

[28] A. Collucci, «"O minimo que se deve saber sobre as baterias Lipo",» Amtonio Collucci, [En línea]. Available: http://amcollucci.com.br/2013/08/10/o-minimo-que-se-deve-saber-sobre-baterias-lipo/. [Último acceso: 14 02 2016].

[29] «Software choice for *Pixhawk*Controller,» Pixhawk, [En línea]. Available: https://pixhawk.org. [Último acceso: 14 02 2016].

[30] «*Dronecode*,» [En línea]. Available: https://www.*Dronecode*.org/. [Último acceso: 14 02 2016].

[31] «Manual Ublox M8N,» [En línea]. Available: https://www.u-blox.com/sites/default/files/NEO-M8_DataSheet_(UBX-13003366).pdf. [Último acceso: 18 01 2016].

[32] RC Innovations, Empresa especializada en drones, FPV y UAV, Productos,» RC Innovations, [En línea]. Available: http://rc-innovations.es/. [Último acceso: 16 02 2016].

[33] «*Dronecode*, OpenSource Dron Developping,» Linux Foundation Collaborative Projects, [En línea]. Available: https://www.*Dronecode*.org/. [Último acceso: Diciembre 2015].

[34] «Teach, Learn and Make with Raspberry Pi,» Raspberry Pi Foundation, [En línea]. Available: https://www.raspberrypi.org/. [Último acceso: 16 02 2016].

[35] «Ubiquity Networks Products,» Ubiquity Networks, [En línea]. Available: https://www.ubnt.com/. [Último acceso: 16 02 2016].

[36] «Ultrasonic sensors,» Maxbotix, [En línea]. Available: http://www.maxbotix.com/Ultrasonic_Sensors/MB1242.htm. [Último acceso: 18 01 2016].

[37] «Edimax Technology Products,» Edimax, [En línea]. Available: http://www.edimax.es/edimax/es/. [Último acceso: 18 01 2016].

[38] «XCopter Calc-Multicopter Calculator, Proyecto TFG,» ECALC, [En línea]. Available: http://www.ecalc.ch/xcoptercalc.php?ecalc&lang=en&cooling=good&rotornumber=4&config=flat&frame=400&tiltlimit=90&weight=850&calc=auw&elevation=500&airtemp=25&qnh=1013&chargestate=0&s=3&p=1&battdisc=0.85&esc=0&esccont=&escmax=&escri=&escweight=&motor=0&gear. [Último acceso: 28 01 2016].

[39] «Manuel de Montaje e instalación de la Pixhawk,» 3DR. [En línea]. Available: https://3dr.com/wp-content/uploads/2014/03/pixhawk-manual-rev7.pdf. [Último acceso: 21

02 2016].

[40] «Raspbian, Software for Raspberry Pi,» BYTEMARK Hosting, [En línea]. Available: https://www.raspbian.org/. [Último acceso: 20 02 2016].

[41] «Communicating with Raspberry Pi via Mavlink,» Ardupilot, [En línea]. Available: http://dev.ardupilot.com/wiki/raspberry-pi-via-mavlink/. [Último acceso: 19 01 2016].

[42] «Dronkit: Developper Tools for Drones,» Dronkit, [En línea]. Available: http://dronkit.io/. [Último acceso: 02 02 2016].

[43] «The easiest way to run Python, Spyder with SciPy and friends on any Windows PC, without installing anything,» WinPython, [En línea]. Available: http://winpython.sourceforge.net/. [Último acceso: 02 02 2016].

[44] G. L. Porto-Andión, *Despliegue de una Red Móvil ad hoc en las Lanchas de Instrucción de la ENM,* Marín: CUD, 2016.

[45] G. G. d. Paredes, *"Extracción y presentación de la información de los distintos sensores de las lanchas de instrucción de la ENM",* Marín: CUD, 2015.

[46] G. O. León, «Filtros Kalman,» Instituto Tecnológico de Costa Rica, Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica. [En línea]. Available: http://www.ie.itcr.ac.cr/gaby/Control_Automatico/Presentaciones/09_ControlconFiltrodeKalman_v12s01.pdf. [Último acceso: 24 02 2016].

[47] «Universidad de Informática de Edinburgo,» [En línea]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm. [Último acceso: 25 02 2016].

[48] «NuPic,» Numenta, [En línea]. Available: http://numenta.org/. [Último acceso: 29 02 2016].

[49] C. A. Perez-Seoane, *Desarrollo de un Sistema de AI para la supervisión y detección de rutas marítimas,* Marín: Centro Universitario de la Defensa, 2016.

[50] R. L. Cuquerella, *Fabricación y puesta en marcha de un radar FMCW de bajo coste en banda S,* Marín: Centro universitario de la Defensa, 2015.

[51] «Web de La Moncloa,» [En línea]. Available: http://www.lamoncloa.gob.es. [Último acceso: 13 enero 2015].

[52] J. L. B. Jiménez, *Diseño fabricación y puesta en marcha de una plataforma Stewart*, Marín: Centro Universitario de la Defensa, 2016.

[53] «Drones, la últims revolución militar,» [En línea]. Available: http://www.elperiodico.com/es/graficos/internacional/drones-militar-armas-9530/yipi7/9kay/13yei/mthrfck. [Último acceso: 29 02 2016].

ANEXO I: DICCIONARIO DE SIGLAS

AI: *Artificial Intelligence*

AIS: *Automatic Identification System*

API: *Application Programing Interface*

ATOL: *Automatic Take Off and Landing*

CEP: *Circular Error Probability*

CPU: *Central Processing Unit*

CSV: *Comma Separated Value*

CUD: *Centro Universitario de la Defensa*

DHCP: *Dinamic Host Configuration Protocol*

ECM: *Electronic Counter Measures*

EGNOS: *European Geostationary Navigation System*

ENM: *Escuela Naval Militar*

ESC: *Electronic Speed Controller*

ETA: *Estimated Time of Arrival*

FLOAN: *Flotilla de Aeronaves*

GCS: *Ground Control Station*

GM: *Guardiamarina*

GNU: *General Public License*

GPIO: *General Purpose Input/Output*

GPS: *Global Positioning System*

HDMI: *High Definition Multimedia Interface*

HNA: *Host network Association*

IP: *Internet Protocol*

LAN: *Local Area network*

LIPO: *Litio Polimero*

MANET: *Mobile ad Hoc Network*

MAV: *Micro Air Vehicle*

Mavlink: *Micro Air Vehicles Link*

NAVNET: *Navigational Network*

NEC: *Network Enabled Capability*

NMEA: *National Marine Electronics Association*

OLSR: *Optimized Link State Routing*

RC: *Radio Control*

RTL: *Return To Launch*

SBAS: *Satellite Based Augmentation System*

SO: *Sistema Operativo*

SOCAT: *Socket Cat*

SSH: *Secure Shell*

TCP/*IP*: *Transfer Control Protocol/Internet Protocol*

TFG: *Trabajo de Fin de Grado*

TVR: *Trozo de Visita y Registro*

UAS: *Unmanned Aerial System*

UAV: *Unmanned Aerial Vehicle*

UAVS: *Unmanned Aerial Vehicle System*

UDP: *User Datagram Protocol*

UHF: *Ultra High Frecquency*

USB: *Universial Serial Bus*

USV: *Unmanned Surface Vehicle*

VERTREP: *Vertical Replenishment*

WIFI: *Wireless Fidelity*

WLAN: *Wireless Local Area Network*

WRT: *Web Runtime*

ANEXO II: CÓDIGO FUENTE DEL SISTEMA DE CONTROL

Este anexo recoge el código utilizado en el sistema de control. El objetivo de presentar el código es compartir las diferentes funciones y utilidades programadas a lo largo del desarrollo del Sistema de Control para uso público. Teniendo en cuenta que la mayoría de la información empleada a lo largo de este proyecto ha provenido de fuentes abiertas de desarrolladores, este gesto busca contribuir en la medida de lo posible a futuros proyectos d desarrollo.

```
   if 1==1:#IMPORT FUNCTIONS FROM PYTHON AND DRONKIT
      from dronkit import connect, VehicleMode, LocationGlobalRelative,
LocationGlobal
      import time, math, socket
   if 1==1:#CONNECTION SEQUENCE
      ipconnect=raw_input("Where do you want to be connected to?
\n1:10.6.0.1\n2:10.3.0.3\n3:10.1.1.1\n")
      if ipconnect=="1":
           ip="udp:10.6.0.1:14550"
      elif ipconnect=="2":
           ip="udp:10.3.0.3:14550"
      elif ipconnect=="3":
           ip="udp:10.1.1.1:14550"
      #Connects to vehicle
      print "Connecting to vehicle on: %s" %ip
      vehicle = connect(ip, wait_ready=True)
   if 1==1:#CONNECTION TO LANCHA SERVER
      UDP_IP = "10.6.0.1"
      UDP_PORT = 10021


      sock = socket.socket(socket.AF_INET, # Internet
                                       socket.SOCK_DGRAM) # UDP
      sock.bind((UDP_IP, UDP_PORT))
   """A useful set of functions are defined at the begining of the
program"""
   if 1==1:#Constants definition
      altitude=15#Default altitude for locations un menu options 1, 2 and
3
      distancia_aproximacion_inicial=20#distance for the initial
aproximation point
      altura_aproximacion=20#altitude for initial aproximation to lancha
procedure
      altura_aproximacion_precision=15#altitude for precission
```

```
aproximation to lancha
     intervalo_refresco_aproximacion=15#position frefresh interval for
aproximation phase 1
     marcacion_aprox=135#Relative bearing for aproximation course
     vel_rel_fase1=3#Relative speed between UAV and Lancha during
aproximation
     vel_rel_fase2=0.5#Relative speed betweem UAV and Lancha during
precission aproximation
     error_aprox_precision=0.5#Distance difference which activates
setting back to GUIDED MODE
     fraccion_error_distancia_admitido=1.15#Error rate allowed IOT
consider that UAV arrived to aproximation point
     autonomy_time=10#autonomy time for current UAV configuration
     lat_offset=0
     lon_offset=0
   def download_vehicle_commands():#Downloads vehicle parameters
       #Downloads vehicle commands for further use
     cmds=vehicle.commands
     cmds.download()
     cmds.wait_ready()
   def show_home():#Shows Home location
       #Shows actual Home location
     if not vehicle.home_location:
          print "Waiting for Home location..."
          time.sleep(1)
     print "\n Home Location %s" % vehicle.home_location
   def battery_level_minutes(autonomy_time):

     minutes=((float(vehicle.battery.level))/100)*autonomy_time
     return minutes
   def write_position():#Allows user to write a position
       #Allows the user to introduce a position
     Position=LocationGlobalRelative(float(raw_input("Insert Latitude:
")),float(raw_input("Insert Longitude: ")),float(raw_input("Insert
altitude: ")))
     return(Position)
```

```
   def get_distance_meters(location1, location2):#Gives distance
between 2 GPS Locations
       dlat = location2.lat - location1.lat
       dlong = location2.lon - location1.lon
       return math.sqrt((dlat*dlat) + (dlong*dlong)) * 1.113195e5

   def time_to_destination(location1,location2,speed):
     distance=get_distance_meters(location1,location2)
     time=distance/speed+60
     return time
   def arm_and_takeoff(aTargetAltitude):#Arms vehicule and takes off to
a certain altitude
      #Sets mode to GUIDED, arms the vehicle and takes off to a
specified height
     print "Basic pre-arm checks"


     #while not vehicle.is_armable==True:
     #    print "Waiting for the vehicle to initialise..."
     #    time.sleep(1)
     time.sleep(10)
     print "Arming Motors"
     #Set vehicle mode to GUIDED
     vehicle.mode = VehicleMode("GUIDED")
     vehicle.armed=True


     while not vehicle.armed:
          print "Waiting for arming..."
          time.sleep(1)


     print "Taking off!!!"
     vehicle.simple_takeoff(aTargetAltitude)#take off to target
altitude
     while True:
          print "altitude: ",
vehicle.location.global_relative_frame.alt
          if
vehicle.location.global_relative_frame.alt>=aTargetAltitude*0.95:
               time.sleep(2)
```

```
                print "Reached target altitude"
                break
                time.sleep(1)
   def goto_vector(demora,distancia):#Allows to stablish bearing and
distance of movement
     dlat=(distancia/(1852*60))*cos(demora*math.Pi/180)
     dlon=(distancia*cos(vehicle.location.global_relative_frame.lat*math
.Pi/180)/(1852*60))*sin(demora*math.Pi/180)
     print "%s" % dlat
     print "%s" % dlon
     position_1=vehicle.location.global_relative_frame
     position_2=LocationGlobalRelative(vehicle.location.global_relative_
frame.lat+dlat,vehicle.location.global_relative_frame.lon+dlon,vehicle.l
ocation.global_relative_frame.alt)
     vehicle.simple_goto(position_2)
   def datos_lancha(data):#Devuelve(lat,lon,speed,heading,time), si el
mensaje no contiene el rumbo, lo devuelve como False. No se le puede
introducir una secuencia que no sea GLL-VTG-HDT
     HDT=False
     msg=data[9:]
     if msg[0]=="$":
          flag1=msg.find("$GPGLL")
          flag2=msg.find("$GPVTG")
          flag3=msg.find("$GPHDT")
          GLL=msg[flag1:flag2]
          if flag3== -1:
               VTG=msg[flag2:]#Separamos mensaje de velocidad
          else:
               VTG=msg[flag2:flag3]#Seapramos mensaje de velocidad
          if not flag3== -1:
               HDT=msg[flag3:]
               HDT=HDT.split(",")
               heading=float(HDT[1])#Extraemos rumbo si el paquete de
datos lo contiene
          GLL=GLL.split(",")
          lat=GLL[1]
          lat_degrees=float(lat[:2])
          lat_minutes=float(lat[2:])/60
```

```
            lat=lat_degrees+lat_minutes
            if GLL[2]=="S":
                  lat=lat*-1
            lon=GLL[3]
            lon_degrees=float(lon[:1])
            lon_minutes=float(lon[1:])/60
            lon=lon_degrees+lon_minutes
            if GLL[4]=="W":
                  lon=lon*-1
            time=float(GLL[5])

            VTG=VTG.split(",")
            speed=float(VTG[7])/3.6
            if HDT!=False:
                  return lat,lon,speed,heading,time
            elif HDT==False:
                  return lat,lon,speed,False,time
   def punto_aproximacion_aterrizaje(posicion_lancha, rumbo,
distancia,marcacion):#Proporciona la correccion desde la antena GPS al
centro de la toldilla.Pendiente de corregir.
      #altura del punto de aproximacion a la lancha
      beta=(rumbo+marcacion)*math.Pi/180
      vector=distancia
      lat_dif=((vector*math.cos(beta))/(1852*60))
      lat_med=(posicion_lancha.lat+lat_dif/2)*math.Pi/180
      lon_dif=((vector*math.sin(beta))/(1852*60*math.cos(lat_med)))
      lat_aproximacion=posicion_lancha.lat+lat_dif
      lon_aproximacion=posicion_lancha.lon+lon_dif


      return lat_aproximacion, lon_aproximacion
   def vel_aproximacion(vel_lancha,vel_rel):#Calcula la velocidad de
aproximacion a la lancha en funcion de la velocidad de esta
      vel_aprox=vel_lancha+vel_rel
      return(vel_aprox)
   def correccion_toldilla(posicion, rumbo):#Proporciona la correccion
desde la antena GPS al centro de la toldilla.Pendiente de corregir.
      alpha=(rumbo+180)*math.Pi/180
```

```
    vector=2.3
    lat_dif=((vector*math.cos(alpha))/(1852*60))
    lat_med=(posicion.lat+lat_dif/2)*math.Pi/180
    lon_dif=((vector*math.sin(alpha))/(1852*60*math.cos(lat_med)))
    return (lat_dif, lon_dif)
  def median3_filter(pos0, pos1, pos2):
    if pos0.lat<=pos1.lat and pos0.lat<=pos2.lat:
         if pos1.lat<=pos2.lat:
              middle_lat=pos1.lat
         elif pos1.lat>pos2.lat:
              middle_lat=pos2.lat
    elif pos1.lat<=pos0.lat and pos1.lat<=pos2.lat:
         if pos0.lat<=pos2.lat:
              middle_lat=pos0.lat
         elif pos0.lat>pos2.lat:
              middle_lat=pos2.lat
    elif pos2.lat<=pos0.lat and pos2.lat<=pos1.lat:
         if pos0.lat<=pos1.lat:
              middle_lat=pos0.lat
         elif pos0.lat>pos1.lat:
              middle_lat=pos1.lat

    if pos0.lon<=pos1.lon and pos0.lon<=pos2.lon:
         if pos1.lon<=pos2.lon:
              middle_lon=pos1.lon
         elif pos1.lon>pos2.lon:
              middle_lon=pos2.lon
    elif pos1.lon<=pos0.lon and pos1.lon<=pos2.lon:
         if pos0.lon<=pos2.lon:
              middle_lon=pos0.lon
         elif pos0.lon>pos2.lon:
              middle_lon=pos2.lon
    elif pos2.lon<=pos0.lon and pos2.lon<=pos1.lon:
         if pos0.lon<=pos1.lon:
              middle_lon=pos0.lon
```

```
            elif pos0.lon>pos1.lon:
                  middle_lon=pos1.lon


      median_position=LocationGlobalRelative(middle_lat,middle_lon,
pos0.alt)
      return median_position
   def moving_av_filter(location0, location1, location2):
      average_lat=location2.lat*0.2+location1.lat*0.3+location0.lat*0.5
      average_lon=location1.lon*0.2+location1.lon*0.3+location0.lon*0.5
      average_location=LocationGlobalRelative(average_lat,average_lon,veh
icle.location.global_relative_frame.alt)
      return average_location
   if 1==1:#Creamos una biblioteca de situaciones GPS de interes
      Position_Training_Camp=[LocationGlobalRelative(42.3968601, -
8.7073678, altitude),LocationGlobalRelative(42.3967413, -8.7076360,
altitude),LocationGlobalRelative(42.3966066, -8.7075448,
altitude),LocationGlobalRelative(42.3967453, -8.7072927,
altitude),LocationGlobalRelative(42.3960639, -8.7088001,
altitude),LocationGlobalRelative(42.3967017, -8.7084997, altitude)]
      Position_Explanada0=LocationGlobalRelative(42.3956439,-8.7063807,
altitude)
      Position_Explanada1=LocationGlobalRelative(42.3956776,-8.7062278,
altitude)
      Position_Explanada2=LocationGlobalRelative(42.3955330,-8.7063700,
altitude)
      Position_Explanada3=LocationGlobalRelative(42.3955647,-8.7059194,
altitude)
      Position_Explanada4=LocationGlobalRelative(42.3954260,-8.7060079,
altitude)
      Position_Explanada5=LocationGlobalRelative(42.3954419,-8.7055948,
altitude)
      Position_Explanada6=LocationGlobalRelative(42.3953230,-8.7056887,
altitude)
      Position_Explanada7=LocationGlobalRelative(42.3953092,-8.7052354,
altitude)
      Position_Explanada8=LocationGlobalRelative(42.3951745,-8.7052810,
altitude)
      Position_Explanada9=LocationGlobalRelative(42.3953290,-8.7050691,
altitude)
      Position_Explanada10=LocationGlobalRelative(42.3959074,-8.7046641,
altitude)
      Position_Explanada11=LocationGlobalRelative(42.3948397,-8.7045354,
```

```
altitude)
     Explanada=[Position_Explanada0, Position_Explanada1,
Position_Explanada2, Position_Explanada3, Position_Explanada4,
Position_Explanada5,Position_Explanada6, Position_Explanada8,
Position_Explanada9, Position_Explanada10, Position_Explanada11 ]
   if 1==1:#MENU DECLARATION AND DISPLAY
     menu={}
     menu['1']="1-Simple take off and RTL"
     menu['2']="2-Simple take off, go to and land"
     menu['3']="3-Freestyle mode"
     menu['4']="4-Lancha Aproximation Sequence"
     menu['5']="5-Set Parameters"
   while True:#ACTUAL PROGRAM.SECURITY CONDITION FOR ODDISEY. IF THE
VEHICLE MODE IS SET TO LOITER BY THE TARANISS, PROGRAM WILL STOP
EXECUTING
     print "Welcome to Odissey's initial interface!\n\n\nMAIN MENU:\n"
     print menu['1']
     print menu['2']
     print menu['3']
     print menu['4']
     print menu['5']

     selection=raw_input("\nPlease Select: ")
     if selection=="1":#Menu Option 1: arm_and_takeoff and RTL

          arm_and_takeoff(float(raw_input("Entry altitude: " )))
          print "Switching mode to Return to Launch (RTL)"
          vehicle.mode= VehicleMode("RTL")
          if not vehicle.mode.name=="RTL":
               time.sleep(1)
          print "Mode: %s" % vehicle.mode.name
          if vehicle.location.global_relative_frame.alt==0:
               vehicle.close()
     elif selection=="2":#Menu option 2: arm_and_takeoff + go to + land
          print "Select destination: "
          print "\n 1-Training Camp"
          print "\n 2-Explanada"
```

```
           print "\n 3-Introduce Destination"
           ii=raw_input()
           if ii=="1":
                position=Position_Training_Camp[raw_input ("Select
position in training camp")]
           if ii=="2":
                position=Position_Explanada[raw_input ("Select position
in explanada")]
           if ii=="3":
                position=write_position()
           if  (ii=="1" or ii=="2" or ii=="3"):
                vehicle.airspeed=float(raw_input("Set airspeed: "))
                arm_and_takeoff(float(raw_input("Entry initial take off
altitude: " )))
                vehicle.simple_goto(position)
                while
get_distance_meters(vehicle.location.global_relative_frame, position)>1:
                     print "Distance remaining: %s" %
get_distance_meters(vehicle.location.global_relative_frame, position)
                     print "Heading: %s" % vehicle.heading
                     time.sleep(2)
           print "Reached Goal Locaction"
           vehicle.mode=VehicleMode("LAND")
           if not vehicle.mode.name=="Land":
                print "Switching to landing mode"
                time.sleep(1)
           print "Mode: %s" % vehicle.mode.name
           if vehicle.location.global_relative_frame.alt==0:
                vehicle.close()
     elif selection=="3":#Menu Option 3: single commands that can be
interrupted (Freestyle)
           choose="0"
           while not choose=="6":#Allows exit from freestyle mode
                print"\nWelcome to the Freestyle mode. In this mode you
can choose freely what to do"
                while True:#Freestyle menu. Commands from the freestyle
menu can be interrupted
                     print "\nFREESTYLE MENU\n\n"
```

```
                print "Choose action\n"
                      print "1-Take off"
                      print "2-Land"
                      print "3-Go to"
                      print "4-Return to base"
                      print "5-Set flight parameters"
                      print "6-Exit Freestyle Mode"
                      choose=raw_input()

                      if choose=="1":#Freestyle simple_take_off
                            arm_and_takeoff(float(raw_input("Entry
initial take off altitude: " )))
                      elif choose=="2":#Freestyle simple Land
                            vehicle.mode=VehicleMode("LAND")
                            time.sleep(2)
                            if vehicle.mode.name=="LAND":
                                  print "Landing..."
                      elif choose=="3":#Freestyle goto. Sets mode to
GUIDED and allows simple goto stablished locations
                            vehicle.mode=VehicleMode("GUIDED")
                            while not vehicle.mode.name=="GUIDED":
                                  time.sleep(1)
                            position_input=raw_input("Where?\n1-
Explanada\n2-Training Camp")

                            if position_input=="1":
                                  position_input2=raw_input("Where inside
the explanada?\nChoose position from 0 to 11\n")
                                  position=Explanada[position_input2]
                                  print "Going to Explanada_%s"
%position_input2
                            elif position_input=="2":
                                  position_input2=raw_input("Where inside
the training camp?\nChoose position from 0 to 6")

     position=Position_Training_Camp[position_input2]
                                  vehicle.simple_goto(position)
                                  print "Going to Training Camp %s"
```

```
%position_input2
                              print "\nDistance to goal %s" %
get_distance_meters(vehicle.location.global_relative_frame, position)


                              if
get_distance_meters(vehicle.location.global_relative_frame, position)<1:
                                   print "Reached Location"
                         elif choose=="4":#Freestyle RTL
                              vehicle.mode=VehicleMode("RTL")
                              time.sleep(2)
                              if vehicle.mode.name=="RTL":
                                   print "Returning to base..."
                         elif choose=="5":#Freestyle. Allows to change
default airspeed and altitude
                              vehicle.airspeed=float(raw_input("Introduce
new airspeed (m/s): "))
                              altitude=float(raw_input("Introduce the new
default altitude for your flights (m): "))

     vehicle.simple_goto(vehicle.location.global_relative_frame.lat,vehi
cle.location.global_relative_frame.lon,altitude)
                         elif choose=="6":#Freestyle. Exits Freestyle
                              break
                         print
     elif selection=="4":#Menu Option 4: 4 LANCHA APROX AND LANDING
SEQUENCE:0-TAKE OFF;1-INITIAL APROX; 2-SECONDARY APROX; 3-LANDING
           while True:
                print "Starting Phase 0- Take OFF"
                while True:#PHASE 0:Range test and Take OFF
                     data, addr = sock.recvfrom(1024)
                     msg=data
                     if msg[9]=="$":

     lat_lancha0,lon_lancha0,speed_lancha,heading_lancha,time_lancha=dat
os_lancha(msg)#Procesa el mensaje y vuelca los datos de las lanchas en
variables de uso
                              velocidad=vel_aproximacion(speed_lancha,
vel_rel_fase1)#CALCULO DE VELOCIDAD ABSOLUTA EN FUNCION DE LA VELOCIDAD
DE LA LANCHA
                              vehicle.speed=velocidad
```

```
     pos_lancha0=LocationGlobalRelative(lat_lancha0+lat_offset,lon_lanch
a0+lon_offset,altura_aproximacion)

     distance_to_destination=get_distance_meters(vehicle.location.global
_relative_frame, pos_lancha0)

     ETA=time_to_destination(vehicle.location.global_relative_frame,
pos_lancha0, velocidad)
                          print "Destination %s" % pos_lancha0
                          print "Distance to destination: %s meters" %
distance_to_destination
                          print "ETA in %s seconds" % ETA
                          battery=battery_level_minutes(autonomy_time)
                          if battery<(ETA/60):
                               print "Not allowed. Not enough battery
time"
                               break
                          arm_and_takeoff(float(raw_input("Entry
initial take off altitude: " )))
                          heading=0
               if raw_input("START PHASE 1?\n1-YES\n2-No")!="1":
                    break

               while True:#"""PHASE-1:INITIAL APROXIMATION"""
                    data, addr = sock.recvfrom(1024)
                    msg=data
                    if msg[9]=="$":

     lat_lancha0,lon_lancha0,speed_lancha,heading_lancha,time_lancha=dat
os_lancha(msg)

     velocidad=vel_aproximacion(speed_lancha,vel_rel_fase1)#airspeed
calcule based on speed_lancha
                          vehicle.airspeed=velocidad

     pos_lancha0=LocationGlobalRelative(lat_lancha0+lat_offset,lon_lanch
a0+lon_offset,altura_aproximacion)
                          if heading_lancha!=False:
                               heading=heading_lancha
```

```
                              lat_aprox, lon_aprox
=punto_aproximacion_aterrizaje(pos_lancha0, heading,
distancia_aproximacion_inicial, marcacion_aprox)


     punto_aproximacion=LocationGlobalRelative(lat_aprox,
lon_aprox,altura_aproximacion)
                              vehicle.simple_goto(punto_aproximacion)
                              print "Distance to destination: %s meters" %
get_distance_meters(vehicle.location.global_relative_frame, pos_lancha0)
                              print "ETA in %s seconds" %
time_to_destination(vehicle.location.global_relative_frame,
punto_aproximacion, velocidad)
                         if
get_distance_meters(vehicle.location.global_relative_frame,
punto_aproximacion)<distancia_aproximacion_inicial*fraccion_error_distan
cia_admitido:
                              break
                         if vehicle.mode.name=="LOITER":
                              break
                         time.sleep(intervalo_refresco_aproximacion)
                    if raw_input("START PHASE 2?\n1-YES\n2-No")!="1":
                         break
                    i=0
                    while True:#"""PHASE-2:PRECISSION APROXIMATION"""
                         data, addr = sock.recvfrom(1024)
                         msg=data
                         if msg[9]=="$":

     lat_lancha0,lon_lancha0,speed_lancha,heading_lancha,time_lancha=dat
os_lancha(msg)

     pos_lancha0=LocationGlobalRelative(lat_lancha0+lat_offset,lon_lanch
a0+lon_offset,altura_aproximacion_precision)

     pos_UAV0=vehicle.location.global_relative_frame

     velocidad=vel_aproximacion(speed_lancha,vel_rel_fase2)
                              vehicle.airspeed=velocidad
                              if i<2:
                                   pos_media_lancha=pos_lancha0
                                   pos_media_UAV=pos_UAV0
```

```
                            if i>=2:

     pos_media_lancha=moving_av_filter(pos_lancha0,pos_lancha1,pos_lanch
a2)
                                  pos_media_UAV=moving_av_filter(pos_UAV0,
pos_UAV1, pos_UAV2)
                            if i>=1:
                                  pos_lancha2=pos_lancha1
                                  pos_UAV2=pos_UAV1


                            pos_lancha1=pos_lancha0
                            pos_UAV1=pos_UAV0
                            correccion_lat,
correccion_lon=correccion_toldilla(pos_media)

     pos_media_corregida=LocationGlobalRelative((pos_media.lat+correccio
n_lat),(pos_media.lon+correccion_lon),altura_aproximacion_precision)
                            vehicle.simple_goto(pos_media_corregida)
                            if
get_distance_meters(pos_media_UAV,pos_media_corregida)<error_aprox_preci
sion:
                                  break
                            if vehicle.mode.name=="LOITER":
                                  break
                      if vehicle.mode.name=="LOITER":
                            break
                while True:#PHASE-3:LANDING
                      fase=3
                      data, addr = sock.recvfrom(1024)
                      msg=data
                      if msg[9]=="$":

     lat_lancha0,lon_lancha0,speed_lancha,heading_lancha,time_lancha=dat
os_lancha(msg)

     pos_lancha0=LocationGlobalRelative(lat_lancha0+lat_offset,lon_lanch
a0+lon_offset,altura_aproximacion_precision)

     pos_UAV0=vehicle.location.global_relative_frame
                            if i<2:
```

```
                                    pos_media_UAV=pos_UAV0
                                    pos_media=pos_lancha0

     vehicle.airspeed=vel_aproximacion(speed_lancha,fase)
                               if i>=2:

     pos_media=moving_av_filter(pos_lancha0,pos_lancha1,pos_lancha2)
                                    pos_media_UAV=moving_av_filter(pos_UAV0,
pos_UAV1, pos_UAV2)
                               if i>=1:
                                    pos_lancha2=pos_lancha1
                                    pos_UAV2=pos_UAV1
                               pos_lancha1=pos_lancha0
                               pos_UAV1=pos_UAV0
                               correccion_lat,
correccion_lon=correccion_toldilla(pos_media)

     pos_media_corregida=LocationGlobalRelative((pos_media.lat+correccio
n_lat),(pos_media.lon+correccion_lon),vehicle.location.global_relative_f
rame.alt)
                               if
get_distance_meters(pos_media_UAV,pos_media_corregida)>error_aprox_preci
sion:
                                    if vehicle.mode.name!="GUIDED":
                                         vehicle.mode=VehicleMode("GUIDED")
                                    vehicle.simple_goto(pos_media_corregida)
                               if
get_distance_meters(pos_media_UAV,pos_media_corregida)<error_aprox_preci
sion:
                                    if vehicle.mode.name!="LAND":
                                         vehicle.mode=VehicleMode("LAND")
                          if vehicle.mode.name=="LOITER":
                               break
     elif selection="5":
          print "Welcome to the Parameters set display. You will be
shown the default parameters and ordered to introduce your own"
          print "altitude %s meters(Default altitude for locations un
menu options 1, 2 and 3)" % altitude
          altitude=raw_input("Introduce new value\n")
          print "distancia_aproximacion_inicial %s meters (distance for
```

```
the initial aproximation point)" % distancia_aproximacion_inicial

            distancia_aproximacion_inicial=raw_input("Introduce new
value\n")

            print "marcacion_aprox %s (Relative bearing for aproximation
course") % marcacion_aprox

            marcacion_aprox=raw_input("Introduce new value\n")

            print "altura_aproximacion %s (altitude for initial
aproximation to lancha procedure)" % altura_aproximacion

            altura_aproximacion=raw_input("Introduce new value\n")

            print "intervalo_refresco_aproximacion %s secs (position
frefresh interval for aproximation phase 1)" %
intervalo_refresco_aproximacion

            intervalo_refresco_aproximacion=raw_input("Introduce new
value\n1")

            print "vel_rel_fase1 %s (Relative speed between UAV and
Lancha during aproximation)" % vel_rel_fase1


            print "altura_aproximacion_precision %s altitude for
precission aproximation to lancha" % altura_aproximacion_precision

            altura_aproximacion_precision=raw_input("Introduce new
value\n")

            print "fraccion_error_distancia_admitido %s (Error rate
allowed IOT consider that UAV arrived to aproximation point)" %
fraccion_error_distancia_admitido

            fraccion_error_distancia_admitido=raw_input("Introduce new
value")

            print "vel_rel_fase2 %s (Relative speed betweem UAV and
Lancha during precission aproximation)" vel_rel_fase2

            vel_rel_fase2=raw_input("Introduce new value")

            print "error_aprox_precision %s (Distance difference which
activates setting back to GUIDED MODE)" % error_aprox_precision

            error_aprox_precision=raw_input("Introduce new value")

            print "autonomy_time %s (autonomy time for current UAV
configuration)" % autonomy_time

            autonomy_time=raw_input("Introduce new value")

            if raw_input("Do you want an offset to be stablished?\n1-
Yes\n2-No ")=="1":

                  print "lat_offset=0"

                  lat_offset=raw_input("Introduce new value")

                  print "lon_offset=0"

                  lon_offset=raw_input("Introduce ne value")
```

```
    else:#unknown option chosen
        print "Unknown option"
```

ANEXO III: BALANCE TOTAL DE COSTES

En este Anexo se detallan los costes asociados al TFG:

Elemento	***Descripción***	***Proveedor***	***Coste unitario***	***Cantidad***	***Coste Total***
Kit DJI F550	Kit de montaje de hexacóptero con estructura, motores y cableado incorporado	RC Innovations	256€	2	502€
Raspberry Pi 2 B	Placa/ordenador en miniatura de bajo coste	Pc Componentes	41€	2	82€
Pixhawk	Controladora de vuelo para *UAV*	HobbyReal	100€	2	200€
Módulo Compás+GPS para Pixhawk	Receptor GPS con compás incorporado compatible con la controladora *Pixhawk*	RC Innovations	55€	2	110€
Fr Sky Taranis	Emisora Multicanal compatible con módulos de cambio de frecuencia	RC Innovations	255€	1	255€
EzUHF TX LRS 433MHz	Módulo de transmisión de 433 MHz compatible con la emisora Taranis	RC Innovations	95€	1	95€
EzUHF RX 8ch	Módulo receptor de 433 MHz	RC Innovations	115€	1	115€
Ubiquity Picostation M2 HP	Antena *WIFI* omnidireccional compacta	PC Componentes	80€	1	80€
Batería Desire Power V8	Batería *Lipo* 3S 6000mAh 30/45C	RC Innovations	59,5€	4	238€
Cargador de baterías Graupner Ultramat	Cargador/Equilibrador de baterías multipropósito	RC Innovations	85€	1	85€
Cámara IP Edimax IC-3115W	Cámara *IP* con capacidad HD compacta y ligera	App-Informática	51€	1	51€
Switch Edimax Es-3305P	*Switch* de red de 5 entradas de diseño compacto	PC Componentes	10€	1	10€
Maxbotix 12CXL-MaxSonar EZ4	Sonar altímetro para *UAV*	Exp-Tech.de	37€	1	37€
	COSTE TOTAL DEL MATERIAL (Para dos uds.)				**1860€**

ANEXO IV: GALERÍA FOTOGRÁFICA

En este anexo se exponen fotografías del proceso de montaje, las pruebas realizadas y el *UAV* final:

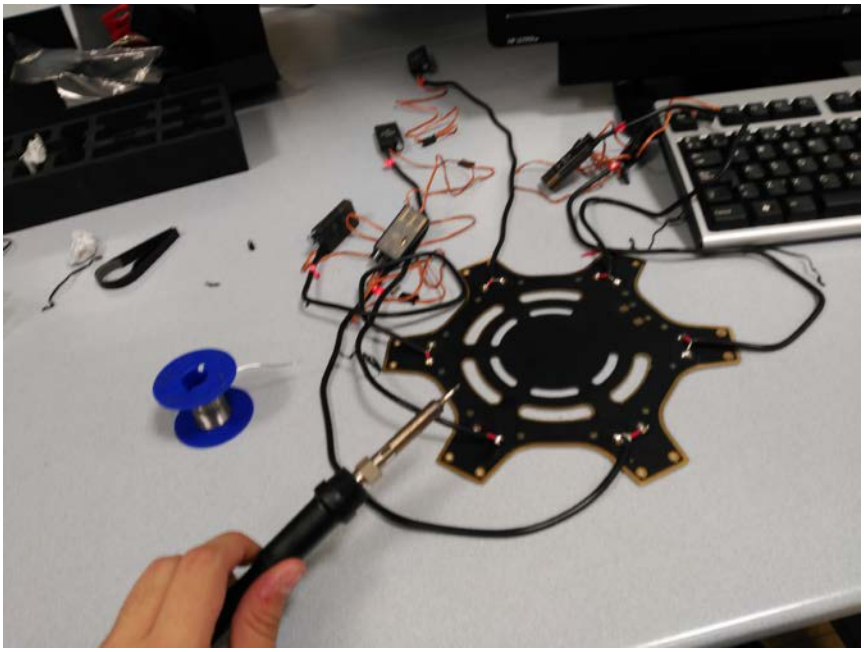

Ilustración IV-1 Montaje: Soldadura de las tomas de corriente de los motores

Ilustración IV-2 Montaje: Tomas de control de los ESC

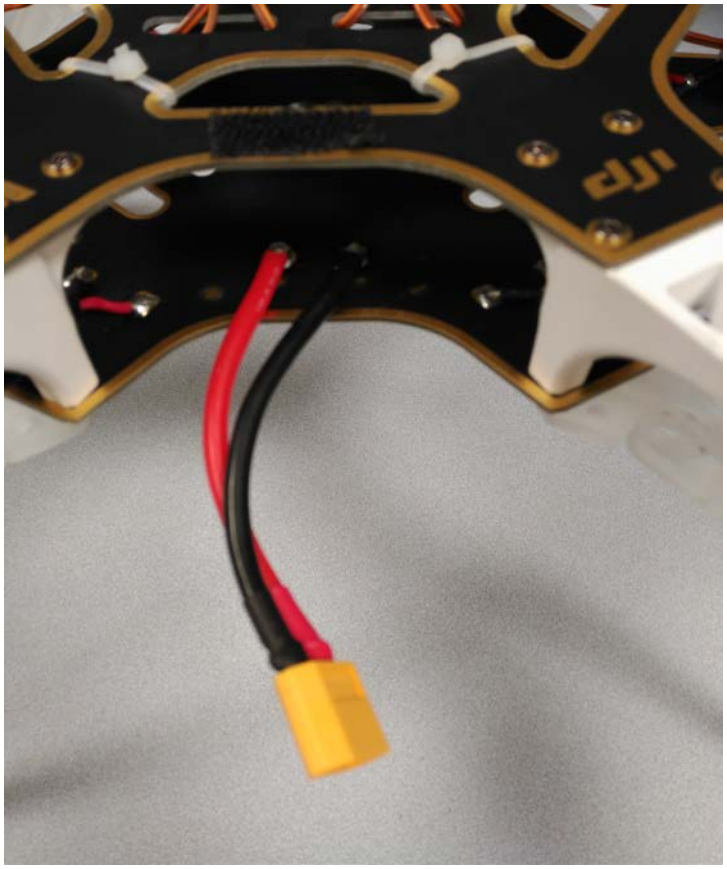

Ilustración IV-3 Montaje: Toma de corriente del *UAV*

Ilustración IV-4 Montaje: Espacio libre para instalación de dispositivos.

Ilustración IV-5 Montaje: Módulo DJI Completo

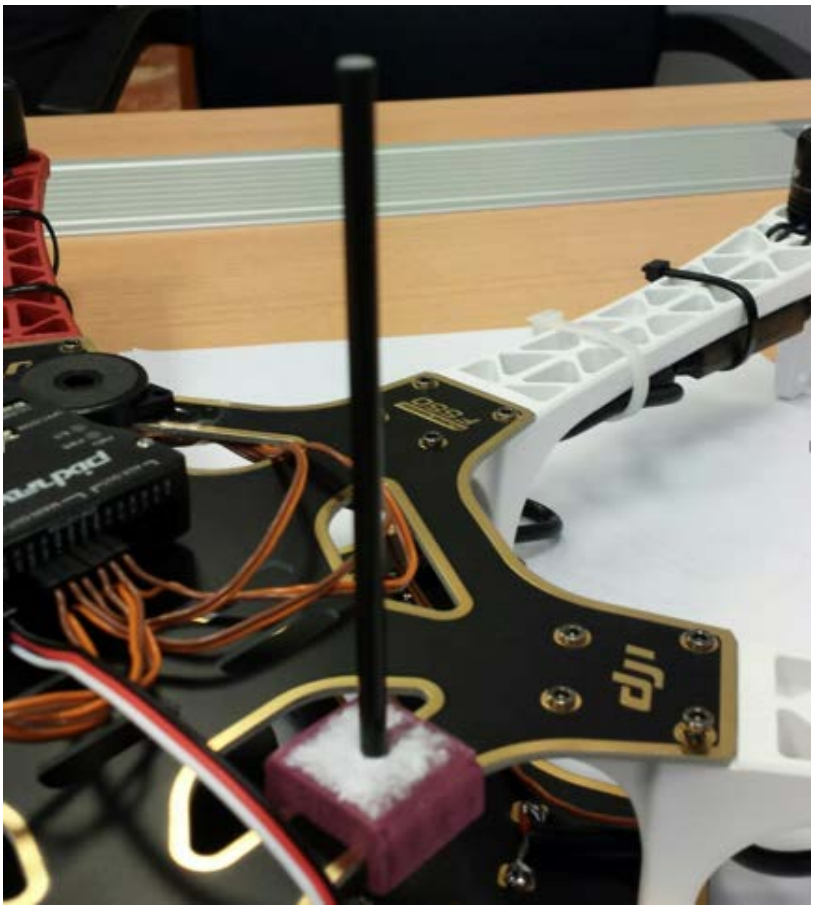

Ilustración IV-6 Montaje: Soporte para el Módulo GPS impreso en 3D

Ilustración IV-7 Pruebas de Vuelo Iniciales (1)

Ilustración IV-8 Pruebas de Vuelo Autónomo (1)

Ilustración IV-9 Pruebas de Vuelo Autónomo (2)

Ilustración IV-10 Pruebas de Vuelo Autónomo (3)

Ilustración IV-11 Pruebas de Vuelo Autónomo (4)

Ilustración IV-12 Pruebas de Vuelo Autónomo (5)

Ilustración IV-13 Pruebas de Vuelo Autónomo (6)

Ilustración IV-14 *UAV* Final (Proa)

Ilustración IV-15 *UAV* Final (Popa)

Ilustración IV-16 *UAV* Final (Babor)

Ilustración IV-17 *UAV* Final (Estribor)

Ilustración IV-18 *UAV* Final (Vista inferior)