UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño y construcción de un UAS mediante Raspberry Pi y código abierto

Autor: Vicent Barrera Gaspar

Tutor: Ángel Rodas Jordá

Cotutor: Pablo Antonio Morcillo Pallarés

Grado en Ingeniería Aeroespacial

Curso 2017-2018

Agradecimientos

Quisiera aprovechar esta oportunidad para agradecer el esfuerzo y apoyo de mi tutor Ángel Rodas y mi cotutor Pablo Morcillo, así como la atención, la cercanía y el interés en mi aprendizaje que han mostrado a lo largo de estos duros meses.

Por otra parte, no podría olvidarme de aquellos que también han estado a mi lado durante la realización del Trabajo de Fin de Grado: para empezar, mi familia y en especial mis padres, que han sido un respaldo fundamental para seguir mi camino. Por otro lado, mis amigos, quienes no han cesado en apoyarme y mostrar interés durante este proyecto, tanto los que he hecho a lo largo de mis estudios de Ingeniería como con los que ya me relacionaba, sin olvidarme aquellos que he conocido a lo largo de mis prácticas de empresa en Stadler Rail Valencia, quienes preguntaban a diario por mi progreso.

Resumen

A lo largo de esta obra, se detallará el procedimiento seguido para la selección, construcción y configuración de un sistema aéreo sin tripulación a bordo (UAS de aquí en adelante) de código abierto. Se describirá la motivación que llevó a su realización y los fundamentos básicos de los UAS, así como los pasos a seguir en su utilización en diferentes contextos. También se explicará qué hardware y qué software se han elegido y por qué.

Por último, se hablará de las pruebas de vuelo y software realizadas, así como de las conclusiones finales, de los trabajos futuros del sistema y el presupuesto que se ha utilizado para llevar a cabo el proyecto.

Resum

Al llarg d'aquesta obra, es detallarà el procediment seguit per a la selecció, construcció i configuració d'un sistema aeri sense tripulació a bord (UAS d'ara en avant) de codi obert. Es descriurà la motivació que dugué a la seua realització i els fonaments bàsics dels UAS, així com els passos a seguir per a la seua utilització en diferents contextos. També s'explicarà què hardware i què software s'han elegit i per què.

Per últim, es parlarà de les proves de vol i software realitzades, així com de les conclusions finals, dels futurs treballs amb el sistema i del pressupost que s'ha utilitzat per a dur a terme el projecte.

Abstract

All along this paper, the procedure followed for the selection, construction and configuration of an open source Unmanned Aerial System (UAS from now) will be detailed. This work will describe the motivation for developing an UAS and the basic fundaments of this technology, as well as which steps must be followed for its use in different contexts. We will explain also which hardware and software have been selected and why.

At last, we will talk about the trial flights and software tests that were performed, the final conclusions, the future works with this system and the budget used for the accomplishment of this project.

Índice

1. Motivación y objetivos

El desarrollo y auge de un ámbito tecnológico siempre suscita el interés del público, y es este repentino crecimiento y desarrollo del UAS la principal motivación de la concentración del proyecto en este sector. Además, esto supondría posibilidad de estudiar, construir y controlar todos los aspectos del UAS: el mundo aeroespacial es un sector que requiere de grandes inversiones monetarias, y pocos son los ingenieros que tienen acceso al diseño y control de componentes fundamentales de las aeronaves, a la par de requerir grandes equipos y material para su estudio y construcción; sin embargo, en el sector UAS es posible realizar estudios, pruebas y construir nuevas aeronaves sin la necesidad de grandes cantidades de dinero o la dependencia de equipos enteros de ingenieros. Esta idea aseveraba la posibilidad de construir un concepto completo bajo la supervisión de mis tutores, donde podría poner en práctica todo lo aprendido durante el grado y ampliar mis conocimientos en un sector tan multidisciplinar, fascinante y del que poco sabía.

Vivimos en una época con un crecimiento tecnológico constante, cada vez más acelerado. Este hecho provoca que tecnologías antes impensables, surjan; invenciones que antaño poseían una función, hoy poseen otra; tecnologías que ocupaban habitaciones enteras hoy caben en la palma de la mano. Este desarrollo tecnológico permite que ciertos avances sean más accesibles al público en general (antaño los teléfonos móviles y ordenadores solo eran poseídos por unos pocos privilegiados), a la par de permitir a la ingeniería centrarse en desarrollos más avanzados: la posesión y conocimiento de un peldaño tecnológico, permite al ser humano centrarse en subir el siguiente peldaño (el pescador puede centrarse en pescar sin preocuparse por construir una embarcación cuando compra un barco pesquero).

Cada día es más fácil desarrollar un aprendizaje autodidacta, y aunque la enseñanza universitaria sigue siendo fundamental en los ámbitos del conocimiento avanzado, actualmente es posible encontrar información sobre casi cualquier cosa en la red. Esto permite a muchos desarrollar ciertas aplicaciones o tecnologías tan solo con fuentes fidedignas.

Los UAS se desarrollaron como forma de análisis a pequeña escala de una aeronave a estudiar, que ha derivado a una forma de observación y control aéreo de múltiples ámbitos, como la agricultura, ingeniería civil, seguridad, rescate marítimo, cine, fotografía, música o la simple finalidad recreativa. Los medios necesarios para desarrollar un UAS básico pueden estar a disposición de cualquier civil, pero existen pocas fuentes fidedignas que puedan guiar el diseño, construcción y configuración robustos paso a paso de un UAS. Además, pocos son los UAS que permitan a un desarrollador de aplicaciones trabajar directamente sobre un sistema aéreo de código abierto, operable y de ejecución de código inmediata.

Apoyándonos en los párrafos introductorios, se desarrollan los **objetivos principales de este trabajo: el diseño y construcción de una plataforma operable de código abierto, que permita el desarrollo y ejecución inmediata de aplicaciones que necesiten la tecnología UA (*Unmanned Aircraft*, es decir, aeronave sin tripulación a bordo) equipada con periféricos para alcanzar su objetivo**; hablamos de la **construcción de un UAS de código abierto** listo para el desarrollo de aplicaciones avanzadas, lo que permitirá al usuario centrarse en la creación de dichas aplicaciones para cumplir con su cometido o deseo. En segundo lugar, **la**

documentación detallada de este proceso de selección y montaje, que permita al lector poseer nociones básicas para el montaje de un UAS y un procedimiento orientativo a seguir.

Por otro lado, es menester nombrar unos **objetivos secundarios** que acompañan a los principales: el propio aprendizaje del diseño, selección de componentes, construcción y configuración de un UAS desde cero; la adquisición de conocimientos multidisciplinares que requiere el proyecto (programación, conceptos de telecomunicaciones, montaje de cableados, procedimientos de procesado de información, ...) y el acercamiento de la tecnología a un público menos especializado (divulgación tecnológica).

2. Fundamentos del UAS

En este apartado introduciremos aspectos de los UAS que se han considerado fundamentales para el entendimiento del resto del proyecto. Este apartado no pretende ser exhaustivo ni ocupar el núcleo del proyecto, sino que busca resultar una presentación de conceptos básicos previos para el entendimiento del resto proyecto y como información complementaria que pueda resultar útil y de interés para el lector.

2.1 Introducción

Los orígenes de las aeronaves sin tripulación a bordo pueden datarse a los alrededores de finales del siglo XIX. Durante esta época, se registraron usos de aeronaves a escala controladas de forma remota para estudiar el comportamiento previo a la construcción de la aeronave, así como el uso de globos aerostáticos no tripulados equipados con bombas en Austria (1849).

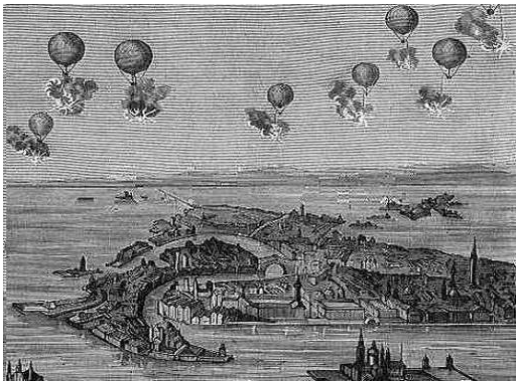


Ilustración 1 Globos aerostáticos del siglo XIX descargando bombas

Sin embargo no sería, hasta 1898 cuando Tesla aplicaría, por primera vez, radiocontrol a un vehículo: controló a distancia un vehículo marítimo mediante ondas de radio. Esto implica que la evolución de las aeronaves no tripuladas no hubiera sido posible sin el avance de las aplicaciones de la radio, junto con otras tecnologías.

Ilustración 2 Barco radiocontrol inventado por Tesla

Durante la Primera Guerra Mundial, se utilizaron cometas con cámaras para la vigilancia. Hasta la Guerra Fría, sus usos se limitaron a ser bombarderos, bancos de pruebas y vigilantes limitados por el movimiento de la nave. Durante la mencionada guerra, se intensificó el uso de las naves como observadores en sitios difícilmente accesibles.

Ilustración 3 UA de la Guerra Fría

Ya en la llamada "Guerra contra el Terrorismo", los UAS militares han añadido a su función de excelentes vigilantes, la de ser un arma.

Sin embargo, hoy en día aparecen numerosas usos más allá de las aplicaciones militares, y es que, si bien se hace uso de su excelente visión aérea, esta es aplicada en la agricultura, la seguridad civil, la construcción, el rescate, la prevención o incluso extinción de incendios.

Ilustración 4 Dron agrícola pulverizando un cultivo

Antes de presentar el funcionamiento de las naves no tripuladas, debemos aclarar cinco conceptos importantes:

Dron o drone: es una traducción al inglés de la palabra "zángano". Esta palabra se utiliza en aeronáutica para designar a los vehículos aéreos no tripulados. Este concepto es el más utilizado por el público, pero es poco específica para campos especializados, por lo que se intentará evitar este sustantivo para ahorrar confusiones.

UAV (*Umanned Aerial Vehicle*): Este es un concepto **obsoleto** (según la ICAO), el cual describía una aeronave que funcionaba sin ninguna persona en su interior. Actualmente, se prefiere el término UA.

UA (Unmanned Aircraft): Este concepto es utilizado para diferenciar la propia aeronave (estructura, motores, sistema embarcado y periféricos) del resto del sistema del **UAS**.

UAS (*Unmanned Aerial System)*: Es una nomenclatura que se refiere a un sistema aéreo no tripulado. Este comprende todos los segmentos que requiera su operatividad: la/s aeronave/s y la estación terrestre o sistema de control.

RPAS (*Remotely-Piloted Aircraft System*): Este término define un sistema aéreo pilotado de forma remota, es decir, una aeronave cuyo vuelo es controlado por un ser humano mediante una estación o mando de control remoto.

De esta forma, podremos utilizar estos términos sin crear confusión. En general, preferimos utilizar **UAS**, puesto que es entendido como un concepto más genérico y que engloba diversos vocablos, considerando así que el RPAS es un subconjunto del mismo.

2.2 Componentes

Apoyándonos en la ilustración 5, que resulta el esquema genérico de un UAS, describiremos brevemente las partes fundamentales, pero serán desarrolladas más detalladamente en el apartado de *Selección, construcción y configuración: Hardware.*

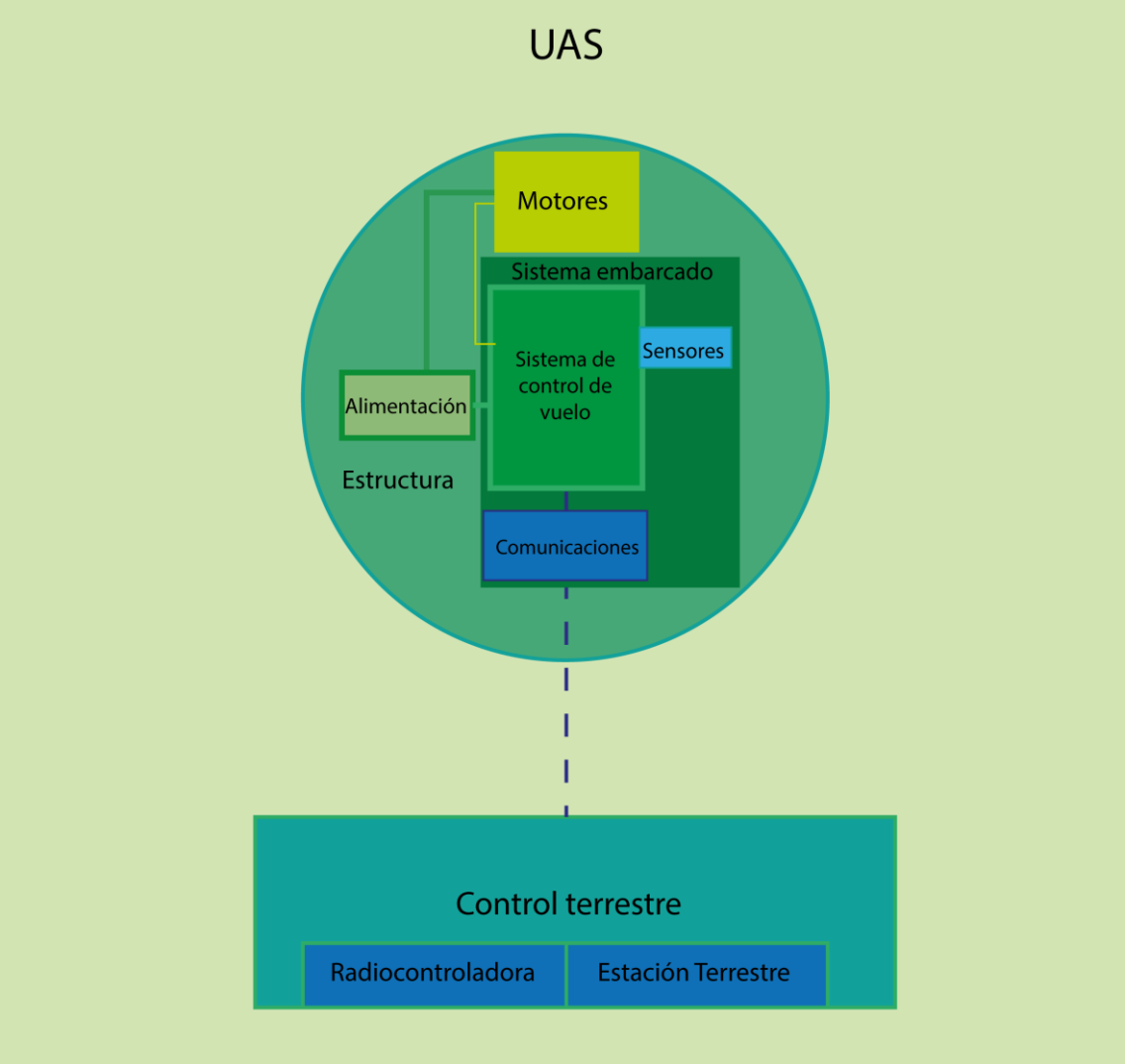


Ilustración 5 Esquema genérico de un UAS

- **Estructura**: encargada de acoger el resto de elementos, mantenerlos protegidos y unidos.
- **Motores**: ejecutan las órdenes de movimiento mediante el movimiento de hélices o propulsión en algunos casos, para elevar y trasladar la nave.
- **Control terrestre:** equipo encargado de la supervisión de vuelo y monitorización del vuelo. En este segmento se reciben los datos tomados por la aeronave, a la par de ser posible enviar nuevas órdenes o modificar las iniciales. Este equipo puede tomar dos formas no excluyentes:

  - **Radiocontroladora**: se trata del mando a distancia que hace uso de ondas de radio para comunicar órdenes directas al UA (velocidad de ascenso/descenso, órdenes de desplazamiento, velocidad, armado/desarmado de motores...). De esta forma le damos la condición de RPAS al sistema.
  - **Estación terrestre (GS)**: haría la función de una radiocontroladora a la que podría añadir algún tipo de procesamiento de alto nivel, como por ejemplo los datos recogidos por una cámara. Es el centro de coordinación, análisis y ejecución de un UA.
- **Sistema embarcado**: se encarga de todos los procesos de información, control y comunicación de la nave. Este componente engloba:
  - **Sensores:** envían información a la nave y al segmento terrestre. Algunos de estos son los giroscopios, cámaras, acelerómetros, barómetros, GNSS o magnetómetros. Las antenas se han representado en el esquema como Comunicaciones.
  - **Comunicaciones**: Se encargan de resolver el intercambio de información y órdenes entre la aeronave y el segmento tierra: receptores/transmisores de radio, antenas,...
  - **Sistema de control de vuelo**: El encargado de mantener la actitud, altura, estabilidad, coordinación de motores... En sistemas reales, esto puede desdoblarse en dos partes: por una parte **el procesador de vuelo**, que hace la función de procesador de la nave; es decir, convierte la aeronave en una suerte de ordenador en miniatura con la capacidad de volar y ejecutar movimientos. Por otra, **el hardware del autopiloto**: se trata de un puente entre el procesador de vuelo y los sensores y motores del UA. En ocasiones incluye algunos de los sensores más básicos.
- **Alimentación**: energía para alimentar todos los componentes del UAS.

El funcionamiento de un UAS puede ser muy variable en consecuencia de sus elementos, pero apoyándonos en nuestro esquema y en lo descrito sobre sus partes, podemos hablar de un funcionamiento genérico común en la gran mayoría.

La aeronave, pese a no ser pilotada por un ser humano dentro de la propia nave, sí que debe ser controlada de alguna forma: bien mediante radiocontrol (RPAS) o mediante un control de vuelo preprogramado; cabe añadir que algunos UAS pueden alternar entre el modo RPAS o el automatizado. Este modo automático es supervisado por la **Estación de Control Terrestre**. Desde este punto, podemos configurar la nave, enviarle órdenes o recibir información desde ella (posición GNSS, altura, velocidad,...). También sirve como punto de referencia de la nave, en caso de necesidad de regreso urgente o de haber finalizado la misión de vuelo.

La nave se guiará a través de datos tomados mediante sus sensores y antenas (GNSS, barómetro, IMU,...) y actuará en consecuencia de su estado de vuelo y su programación de control de vuelo,

o bien en consecuencia de las órdenes enviadas desde el control por radio. Independientemente de cuál de los dos caminos tome, toda orden o ejecución de movimiento pasará por el **autopiloto**. Esta herramienta nos facilita enormemente el manejo de la aeronave, ya que traduce nuestros movimientos o cambios dinámicos mediante la aceleración o deceleración de los distintos motores de la nave y la lectura de datos tomados, para que así dicho movimiento se realice correctamente**. En resumen, nos permite centrarnos en el movimiento total que queremos realizar, y no en la necesidad de controlar los 6 motores uno a uno.** El autopiloto que utilizaremos será **ArduPilot**, debido a que el software con el que trabajaremos hace uso de él.

Las órdenes de vuelo y/o los datos pueden ser enviados mediante diversos métodos: Bluetooth, WiFi, radio,...

2.3 Tipos

Existen numerosas clasificaciones de los UAS, puesto que hay múltiples variaciones de elementos que permiten innumerables combinaciones. Ahora que ya sabemos el funcionamiento básico y componentes de los UAS, podemos detallar algunos de los tipos atendiendo a:

Clase de ala

- Ala fija:

Aviones, ala deltas, planeadores... Estas naves se caracterizan por depender de su velocidad y la forma de sus alas para desplazar volúmenes de aire que permitan su vuelo. Estas naves, permiten vuelos más largos a velocidades constantes gracias a su eficiencia aerodinámica. Sus altas velocidades hacen de esta clase de UA una herramienta muy útil para el reconocimiento de grandes extensiones de terreno: estudio del avance de incendios, realización de mapas cartográficos, persecución,...

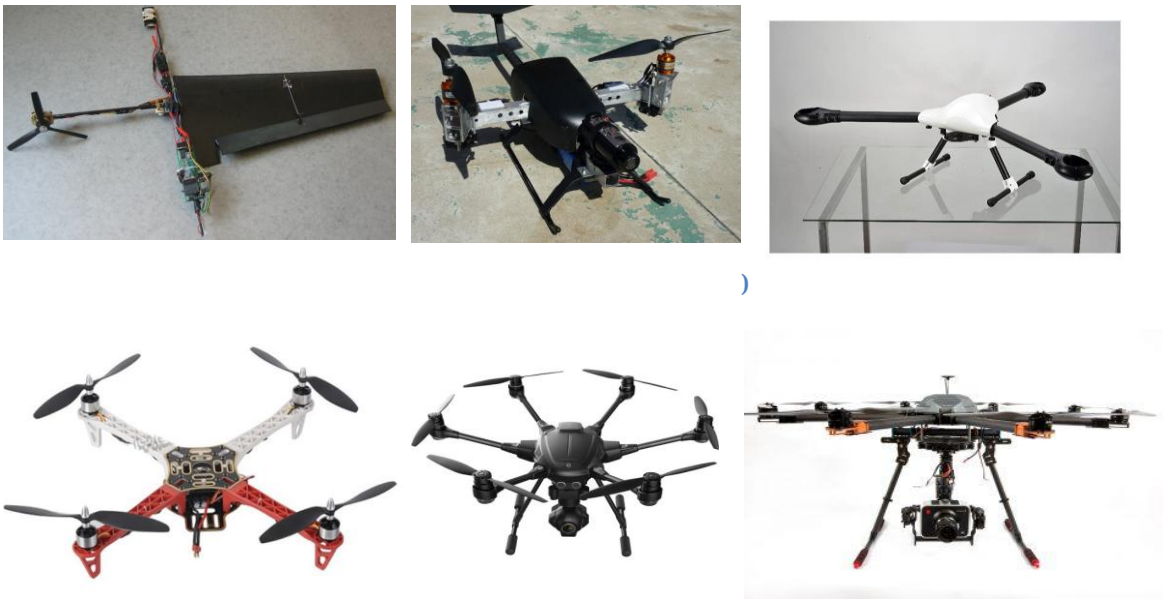
Ilustración 6 UA de ala fija

- Ala rotatoria: helicópteros, multicópteros, autogiros... Llevan a cabo su vuelo gracias a la sustentación de las hélices de los rotores. Estas aeronaves tienen poseen una autonomía muy limitada; sin embargo, poseen bondades únicas: despegue vertical, vuelo estacionario, gran maniobrabilidad, versatilidad en sus aplicaciones,... Es una nave ideal para la toma de imágenes nítidas, escaneado preciso, búsqueda detallada o vuelo en entornos difíciles.

Ilustración 7 UA tipo Helicóptero a la izquierda y tipo multicóptero a la derecha

Número de motores (multicópteros)

El elegir más o menos motores, influirá enormemente en las características del multicóptero; el criterio de selección del número de motores se desarrolla más detenidamente en el apartado 3.2.1. Esta elección determina la estructura, vuelo, coste, control, peso y consumo. Tal y como luego comentaremos en profundidad, el disponer de múltiples motores facilitará una mayor tolerancia a fallos del sistema. Observamos en las figuras siguientes ejemplos de cada uno de los tipos de UAS según el número de motores.



)

Ilustración 8 Distintos tipos de multicóptero según su número de motores

Fuente de la energía eléctrica

El tipo de fuente de energía utilizada dependerá notablemente de lo que necesitemos y el entorno:

- Baterías precargadas: Ideales para vuelos estándar y de corta duración

Ilustración 9 Estructura DJI junto a un equipo de alimentación precargada (batería)

- Combustible fósil (motor generador eléctrico): este tipo nos permite realizar vuelos más largos: se trata de un tipo de UA más autónomo, puesto que el motor se encarga de generar la energía eléctrica a través de la energía mecánica obtenida mediante la combustión de energía fósil. Esta conversión permite generar más energía eléctrica que la almacenada en una batería.

Ilustración 10 Detalle del motor fósil de un UA

- Solar: puesto que la energía solar sigue siendo una tecnología en desarrollo, los UAS que hacen uso de esta tecnología no pueden levantar pesos excesivos. Sin embargo, con la adecuada relación de consumo (motores, peso, aerodinámica,...) y de captación de energía, es posible obtener vuelos de larga duración; este es el caso del prototipo del Puma AE (ilustración 17), fabricado por AeroVironment, el cual consigue realizar vuelos de hasta 9 horas.

Ilustración 11 Pequeño UA alimentado con placas solares

Ilustración 12 Puma AE, UA solar fabricado por AeroVironment

Formas de comunicación

Podemos clasificar los UAS en función de la/s forma/s de comunicación que utilizan (no son incompatibles entre ellas), bien sea mediante radiocontrol, WiFi, Bluetooth, infrarrojos, telemetría,...

Aplicaciones

Cabe añadir una clasificación adicional, y es por su tipo de aplicación: civiles o militares. La diferencia entre un UAS militar y civil, no solo difiere en armamento, generalmente. Los medios de diseño y construcción, materiales, robustez, potencia y de un UA militar tiende a ser superior al civil, ya que posee una financiación y equipo tecnológico mayores. Esto se debe a que ha de responder a ciertas condiciones a las que un UAS civil no necesita enfrentarse normalmente: horas de vuelo, protección frente a impactos, Anti-Hacking avanzado, ...

Un UAS militar tenderá a ser enfocado a aplicaciones bélicas, de espionaje,... Mientras que uno civil tenderá más a la fotografía, recreación, vigilancia rural, reparaciones a grandes alturas ...

Ilustración 13 UAS civil

Ilustración 14 UAS militar

3. Selección, construcción y configuración del sistema

3.1 Requerimientos

Tal y como hemos comentado en el apartado anterior, vamos a concretar el tipo de sistema que vamos a diseñar y configurar en el presente trabajo. Para ello, en primer lugar definiremos los requerimientos que tendrá que cumplir el sistema y que condicionarán los puntos en los que hemos dividido el apartado 3.2 Hardware y 3.3 Software.

Así como una misma herramienta puede ser utilizada de distintas formas o en ámbitos diversos, lo puede ser un UAS. Sin embargo, la propia nave o UA, requiere de cierto grado de especialización a todos los niveles para el campo de trabajo: motores, estructura, materiales, software… Esto implica un grado de reflexión previa a la construcción: cuál es nuestro objetivo y a qué plazo, de qué medios disponemos y qué conocimientos poseemos.

Si hablamos de requerimientos, obviamente pensaremos en la finalidad de la aeronave, qué función realizará, cómo, dónde, por cuánto tiempo; y a nivel personal: por qué. El plazo de trabajo nos permitirá definir la precisión y fiabilidad previas a la construcción y prueba de la tecnología, así como la espera de la adquisición de componentes. Así pues, debemos tener en cuenta factores que hagan fluctuar nuestro tiempo estimado de trabajo: posibles averías o material defectuoso, fallos de envío con los proveedores, problemas de software, disponibilidad del campo de pruebas,...

Nuestra nave deberá poder ser controlada de forma manual (radiocontrol), telemétrica y mediante WiFi desde un controlador embarcado o terrestre, además de poseer conexión GNSS (estos requisitos multiplicarán las aplicaciones posibles). La nave no requerirá de una protección excesiva frente a inclemencias del tiempo, cambios bruscos de temperatura o golpes continuos. Adicionalmente, podrá captar imágenes y vídeo durante el vuelo a través de una cámara y un emisor de imágenes instalados en la propia nave, así como hacer uso de software y de códigos de procesado de imágenes. Estas tomas visuales serán procesadas por una estación de control en suelo (una computadora conectada mediante telemetría o WiFi con la aeronave), o bien en un procesador embarcado en la propia nave. Tras el procesado, si el desarrollador así lo desea en el futuro, actuará y controlará el vuelo en función de las órdenes asociadas a la información obtenida.

El sistema deberá ser capaz de tomar de imágenes, por lo que tendremos que asegurar una estabilidad que permita una captura nítida, una estructura capaz de alojar todos los componentes, una potencia motora suficiente para levantar todo el instrumental, y una fuente de alimentación que permita tener una autonomía suficiente como para hacer varias pruebas de campo sin preocuparnos, teniendo en cuenta que trabajaremos en entornos controlados y cerca de la estación terrestre. No podemos obviar la necesidad de una potencia de procesado capaz de ejecutar el autopiloto y la gestión de órdenes, información de los sensores y las transmisiones. Al buscar una plataforma flexible y que se pueda trabajar con ella fácilmente, y sin necesidad de

comprar nuevas licencias, haremos uso de software libre: estableceremos una plataforma de código abierto.

Otro de los requerimientos del sistema es que pueda ser ampliado con sensores que puedan suministrar información del entorno, y que no figurarán en el diseño inicial; para dar respuesta a ello, se ha pensado en la utilización de la plataforma *Arduino*, la cual cumple también la condición de ser plataforma abierta. La filosofía planteada será conectar el procesador de vuelo a través de un puerto serie de manera que *Arduino* se encargue del procesamiento de dichos sensores, liberando de dicha tarea al procesador de vuelo; esto lo convierte en un sistema modular.

Cabe añadir un último requerimiento del sistema, y es la posibilidad de ampliar su capacidad de procesamiento; esto será necesario en dos contextos no excluyentes entre sí: cuando la naturaleza del periférico a conectar sea demasiado complicada como para que *Arduino* la pueda manejar y en caso de querer darle autonomía de control al UAS (lo que se conoce como **Companion Computer**). Así pues el sistema debe ser capaz de alimentar y soportar estas extensiones y de establecer la comunicación con el procesador de vuelo. Para ser consecuentes con lo dicho hasta ahora este procesador de vuelo adicional dispondrá también de software libre. Es importante destacar que derivado del concepto de software libre se encuentra el concepto de software multiplataforma, en el que se hace una abstracción de los recursos de la máquina y el mismo software puede ejecutarse sobre distintas plataformas que funcionan de manera equivalente.

Una vez se ha reflexionado sobre los requerimientos, pasamos al proceso de creación. El orden de aparición en el escrito es independiente al orden de importancia, pero sí se sigue un orden lógico de dependencia: algunas partes requieren de la definición previa de otras.

Cuando pensemos en los medios, trataremos con el presupuesto con el que contamos, instalaciones de trabajo (talleres, laboratorios, zonas de vuelo…) y componentes que ya tengamos en nuestra posesión o que podamos reutilizar de anteriores trabajos y que sea seguro reutilizarlos (como un cargador o una cámara). Concretando cifras, contamos con un presupuesto aproximado de **1200 euros (detallado en el apartado 5, *Presupuesto*)**.

A partir de los requerimientos de este apartado y lo aprendido en el punto 2, estamos en condiciones de concretar el tipo de UAS que diseñaremos en el presente Trabajo de Fin de Grado, y que será analizado en los siguientes puntos. Como resumen de esta categorización, podemos decir:

- **Tipo de ala**: ala rotatoria
- **Número de motores**: multicóptero hexacóptero (6)
- **Tipo de alimentación**: baterías precargadas
- **Tipo de control**: radiocontrol, WiFi y telemetría
- **Finalidad**: civil.
- **Control terrestre**: mantendremos los dos tipos de funcionalidades: radiocontroladora y estación terrestre

- **Sensores**: aparte de los sensores convencionales (GNSS, IMU, altímetro,...), incorporaremos **cámaras** para capturar imágenes a fin de tener información visual e incluso ser procesadas a fin de tener parámetros útiles en la navegación.
- **Software**: libre.
- **Sistema embarcado**: desdoblaremos procesador de vuelo y hardware de autopiloto, dejando la posibilidad de conectar un segundo procesador de vuelo que acompañe al principal, Companion Computer, concepto que se concretará más adelante. Este sistema debe ser compatible con el uso de software libre.
- **Otros periféricos:** en el presente trabajo se propone que en caso de necesitar añadir periféricos adicionales específicos (sensores de ultrasonidos, infrarrojos,...) , se utilice un Arduino conectado al puerto USB al procesador de vuelo.

Esta concreción nos lleva a plantear el esquema de la ilustración 15, la cual ha sido creada en base a los requerimientos definidos en este punto, y que nos ayudará a guiarnos en la explicación del sistema.

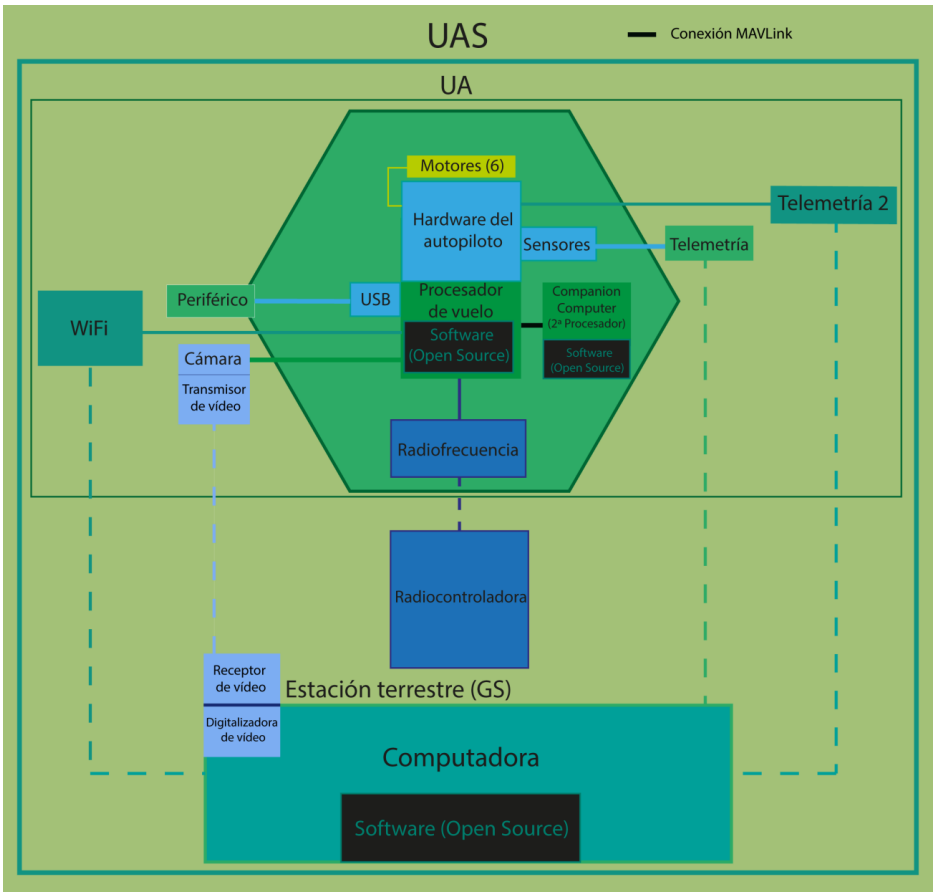


Ilustración 15 Esquema básico derivado de los requerimientos

Faltan por especificar muchas características concretas de este tipo de aeronave elegida, especialmente las referidas a su control y funcionalidad, por lo que serán detalladas en los apartados siguientes.

3.2 Hardware

En este apartado nos centraremos en el segmento hardware de la nave: de qué elementos consta, qué modelos hemos decidido elegir en función de las especificaciones, el orden recomendado y guía de montaje.

3.2.1 Motores

Para empezar, debemos definir el número de motores que necesitaremos. Los criterios principales a tener en cuenta son: estabilidad, redundancia, potencia de vuelo, consumo y peso.

- **Estabilidad:** Necesitamos que la aeronave se mantenga estable: mantener una posición estacionaria a una altura constante o precisión en ciertas maniobras. Hay ciertas misiones que requieren de especial cuidado en la maniobrabilidad de la nave o requieren mantener la nave fija en un punto. Un mayor número de motores significará un aumento en la estabilidad de la nave, así como su control.
- **Fuerza de ascenso:** En caso de que busquemos aumentar el techo o el peso máximo de vuelo, así como la potencia, añadir motores incrementará la fuerza de ascenso de la nave.
- **Redundancia:** Cuando nuestro objetivo es asegurar el vuelo de la nave como prioridad, necesitamos que en caso de que alguno de los motores fallara, la nave fuera capaz de continuar su vuelo pese a la avería. Un mayor número de motores permitiría que, pese al fallo de uno o varios de ellos, pudiera seguir volando, puesto que la pérdida de la fuerza impulsora sería menor en relación a la fuerza total de la nave.
- **Consumo y peso :** Si es la duración de vuelo lo que nos preocupa, debemos saber que más motores supondrán un consumo mayor. Paralelamente, no solo se debe tener en consideración el peso extra del propio motor, sino que además debemos contar con la parte o brazo que debe alojar al nuevo motor: el peso de la estructura también aumenta.

Un requerimiento a tener en cuenta, es la necesidad de una gran estabilidad para poder captar información visual y que estas tengan la calidad suficiente como para procesarlas sin problemas. La fuerza de los motores debe tener la capacidad de levantar la estructura, junto a la electrónica, el cableado, la batería, la telemetría,... : no es necesaria una excesiva potencia motora. Sin embargo una redundancia para evitar caídas y dañar la cámara es un aspecto a considerar. No es un gran problema el consumo, puesto que no necesitamos muchas horas de operatividad: trabajaremos en situaciones controladas y cerca de la nave por el momento.

Con todo esto, deducimos que hacer uso de 6 motores nos ofrecerá una redundancia y una estabilidad suficientes como para realizar correctamente la toma de imágenes y evitar caídas, requisitos primordiales. Hacer uso de 4 motores implicaría una inestabilidad notable, mientras que utilizar 8 requeriría de un consumo y peso de estructural excesivos, además de no necesitar la fuerza extra de ascenso que nos brindaría el tener dos motores más.

Debido a que no se requiere levantar mucho peso, hemos escogido los motores de DJI 2312E - 960RPM, más concretamente el modelo del kit **Flame Wheel 550 (DJI)**, el cual podemos observar en la ilustración 16 .

Ilustración 16 Motores DJI

En esta imagen extraída de la web oficial de **DJI**, nos permite observar de cerca los motores utilizados en nuestro UAS.

Ilustración 17 Actuador de los motores DJI

Los motores reciben la corriente desde el DJI 430 Lite ESC (Electronic Speed Controller, Controlador de Velocidad Electrónico o variador), el cual recibirá la alimentación desde el circuito electrónico. Observamos que desde el ESC sale además un cable doble amarillo anaranjado y marrón; este se encargará de trasladar la señal de control desde el autopiloto hasta el controlador. Finalmente, el variador procesará esta señal de control y transformará la alimentación en una corriente trifásica.

En cuanto a las especificaciones, cada motor posee un empuje máximo de 850g a nivel del mar (14.8V máximo), con un peso máximo de despegue recomendado de 350-400g cada uno. Su rango de temperaturas de trabajo comprende entre los -10ºC y los 40ºC.

Ahora que ya sabemos cuántos motores necesitaremos y qué tamaño tienen, podemos elegir la estructura que utilizaremos.

3.2.2 Estructura

No se realizará ningún vuelo en entornos con inclemencias climáticas, temperaturas extremas, o de cambios bruscos. Además, ni el peso que levantará ni el de los motores requieren de mucha preocupación por la integridad estructural. Tampoco se plantea la posibilidad de recibir impactos regularmente, por lo que nuestra preocupación principal es que la estructura pueda alojar el equipo de procesado, telemetría y captación de imágenes, así como absorber adecuadamente las vibraciones de los motores.

En un principio se planteó el realizar el diseño de la estructura mediante programas de diseño ingenieril y su impresión mediante la impresora 3D. Sin embargo, no poseíamos los conocimientos necesarios en el campo de las vibraciones y la aeroelasticidad como para asegurar que las vibraciones generadas por los motores no afectaran la toma de imágenes. Se podría haber realizado una investigación exhaustiva, así como hacer uso de programas de cálculo mediante elementos finitos, pruebas de ensayo-error,… Pero carecíamos del tiempo necesario y de los medios (acceso indefinido a impresora 3D, ordenadores con potencia de cálculo,…). Además, tuvimos en cuenta que el material que utilizan las impresoras 3D, el PLA (ácido poliláctico), no absorbe bien las vibraciones. Este hecho hizo descartar la impresora 3D como método de fabricación, y al ser nuestro único medio, decidimos hacer a un lado el diseño y fabricación de una estructura propia, y adquirir una ya fabricada y validada.

Por todo esto, en nuestro caso hemos optado por la estructura del kit **Flame Wheel 550 (DJI)**, que ofrece compatibilidad con los motores elegidos, ya que han sido ensayados previamente por el fabricante, no presenta problemas con las vibraciones y es levantada fácilmente por ellos. Además, nos permitirá alojar todo el sistema embarcado, y su geometría nos será compatible con piezas adicionales en el caso de que sea necesario.

La placa inferior contiene un circuito integrado, lo que permitirá ahorrar espacio, cableado y la instalación; por tanto, también ahorraremos peso, tiempo y medios.

La ilustración 18, muestra a la izquierda los motores utilizados, el cuerpo al centro y los brazos estructurales a la derecha. Como observamos, el cuerpo posee unos cuantos orificios que permitirán el paso de aire, lo que ayudará a la refrigeración.

Ilustración 18 A la izquierda, motores y actuadores DJI en su caja; a la derecha, la estructura Flame Wheel ARF F550 de DJI

Llegados a este punto, podemos iniciar el montaje.

Para empezar, mediremos la distancia a la que queremos colocar los motores respecto del cuerpo (nosotros los hemos colocado a la mitad del brazo de la estructura). Cortaremos el cable sobrante, y lo pelaremos, de forma que observaremos dos filamentos en el interior: uno recubierto de rojo, y otro totalmente "desnudo". Debemos soldar los cables de los motores al circuito integrado, de forma que el cable desnudo se suelde en la pletina con el símbolo negativo, y el rojo en la del símbolo positivo. Hay que ir con cuidado de que los filamentos de los dos cables no se crucen y provoquen un cortocircuito.

Ilustración 19 Detalle del primer variador soldado

Nosotros hemos aprovechado y hemos soldado el conector amarillo que dará corriente al circuito desde la célula de energía, de la cual hablaremos más adelante. Para soldar el conector: el cable rojo al positivo y el negro al negativo. Para evitar cortocircuitos innecesarios, recomendamos colocar termorretráctil en el conector, y si se cree necesario, en los cables de los motores.

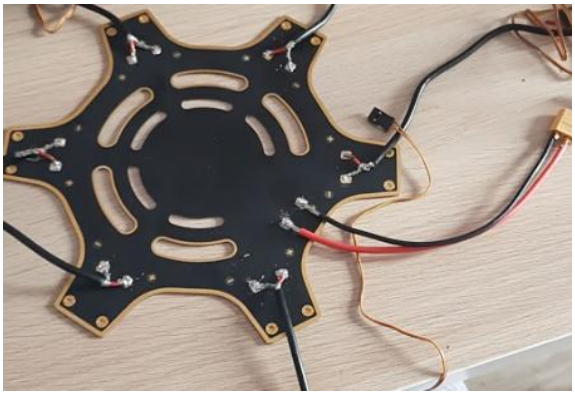


Ilustración 20 Los seis variadores y el cableado de alimentación soldados

Mostramos en la ilustración 21 la imagen del conector con el termorretráctil aplicado:

Ilustración 21 Termorretráctil aplicado al conector

A continuación, podemos proceder al montaje de los brazos de la estructura. Cabe resaltar que esta pieza en la que hemos soldado los cables, será la parte inferior de la estructura, por lo que atornillamos las patas de los brazos sobre la pieza; añadimos que en nuestro caso, los brazos rojos se han colocado en la parte frontal (donde está el conector amarillo), porque así lo recomienda el fabricante.

Acto seguido, colocaremos los variadores de cada motor en su brazo correspondiente; hemos los hemos sujetado sobre cada brazo con una abrazadera.

Ilustración 22 Motores, estructura y cableado preparados

Tras esto, se conecta a su motor correspondiente, el cual se atornillará al extremo del brazo.

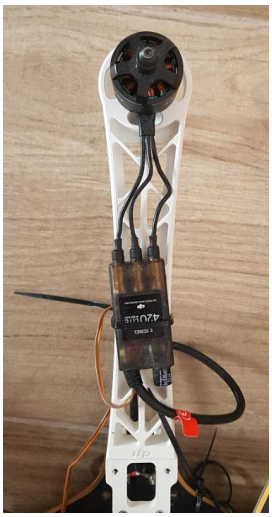


Ilustración 23 Detalle de conexión y sujeción de variador y motor

La parte superior no la atornillaremos, puesto que todavía tenemos que instalar el resto de componentes.

3.2.3 Sistema Embarcado

El sistema embarcado será el que se encargue de todo el procesado de datos a bordo de la nave: comunicaciones, autopiloto, localización, sensores, toma de medidas, … Es decir, nos encontramos ante la creación de una nave que pueda ser independiente de cualquier otro equipo y realice la función asignada previamente, de forma automática. Por tanto, se pondrá en juego el código con el que se ha programado su comportamiento y control.

En este punto nos centraremos en su colocación, puesta a punto y conexión de elemento, mientras que en apartado *Selección, construcción y configuración: Software* trataremos la instalación de software y configuración necesarias para su funcionamiento

3.2.3.1 Procesador de vuelo

Para empezar, es necesario elegir el procesador ideal para nuestro sistema embarcado. Tenemos que tener en cuenta la potencia de cálculo que necesitaremos, el tamaño reservado para el procesador, presupuesto, software a utilizar, compatibilidad con ciertos componentes...

Mediante la ilustración 24, podemos observar gráficamente algunos procesadores de vuelo y qué criterios hemos tomado para elegir el nuestro: **Raspberry Pi**.

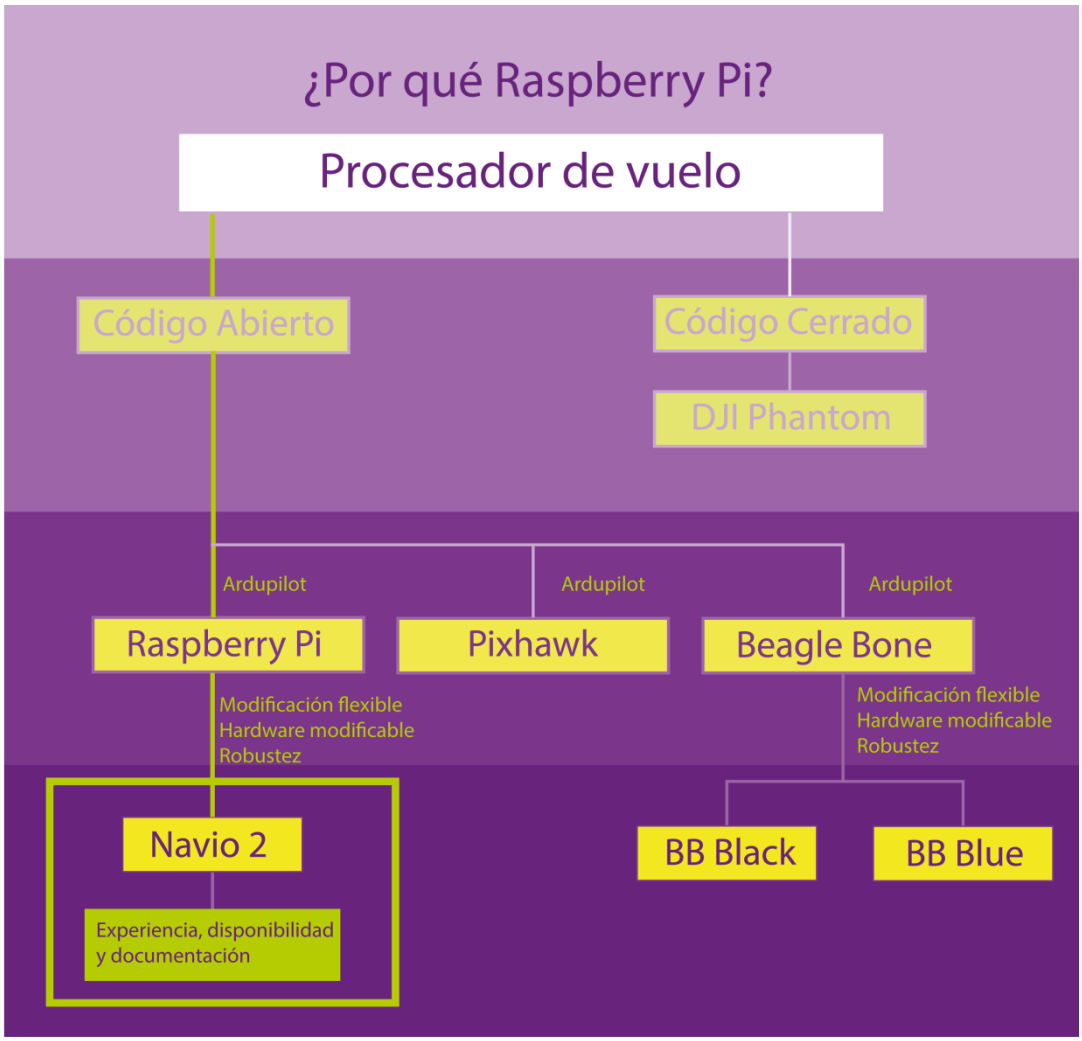


Ilustración 24 Proceso de selección del procesador de vuelo

El procesador de DJI Phantom no ofrece código abierto. Pixhawk ofrece un sistema de control de vuelo (procesador y hardware del autopiloto ya integrados) bastante potente, pero no ofrece la flexibilidad de configuración y modificación que ofrece Raspberry o BeagleBone, lo que limitaría mucho las posibilidades de nuestro proyecto.

Las de BeagleBone, por su parte, son de código abierto y hacen uso de Ardupilot, pero no se posee mucha documentación ni experiencia en su manejo.

En nuestro caso hemos elegido la Raspberry Pi-3, puesto que resulta un procesador con una potencia notable, económico, bastante pequeño, y compatible con software libre y numerosos periféricos. Esta funciona con HDMI, tiene numerosos USB, conexión WiFi, robustez y compatibilidad con hardware de autopilotos. Esta permite hacer uso de NAVIO2, además de contar con experiencia previa con ella y documentación de apoyo.

Ilustración 25 Fotografía de Raspberry Pi 3

En caso de no contar con un ventilador para el procesador, es posible que la carga de trabajo de cálculo sobrecaliente el núcleo del procesador. Por ello, recomendamos hacer uso de radiadores de calor específicos para transmitir la temperatura a estos cuerpos, y que éstos radien el calor al exterior.

Ilustración 26 Raspberry con los radiadores colocados

3.2.3.2 RaspiCam

En el caso de poseer una cámara de Raspberry Pi (Raspberry Pi Camera Module), recomendamos instalarla ahora antes de seguir con el montaje, puesto que luego resultará complicado.

Ilustración 27 RaspiCam y su instalación en el procesador de vuelo

Toda función realizada con la cámara del sistema de transmisión de vídeo (la veremos más adelante), podrá realizarse con la RaspiCam; en nuestro caso, nos hemos enfocado en configurar la cámara del sistema de transmisión, dejando esta para futuros trabajos.

3.2.3.3 Hardware del autopiloto

En el hardware del autopiloto reside la clave de la "transformación" del procesador de vuelo en un sistema embarcado de un UA. Debe poseer la arquitectura capaz de tomar y procesar datos fundamentales para la navegación, poder ejecutar un autopiloto, ofrecer una gestión eficiente de la energía de alimentación y ser compatible con nuestro procesador de vuelo.

En nuestro caso, hemos optado por el hardware del autopiloto NAVIO2, puesto que esta es una tecnología de funcionamiento específico para Raspberry Pi, posee una IMU dual (acelerómetros, giróscopos y magnetómetros), un preciso barómetro, un receptor de GNSS (localización mediante GPS, Galileo, GLONASS, Beidou y satélites SBAS) acompañado de su antena externa MCX, un coprocesador RC I/O, posibilidad de añadir sensores adicionales y radio, suministro de energía de triple redundancia con protección frente a sobretensiones y un módulo de alimentación. Además de todas estas características, tenemos en cuenta una muy importante: cuenta con ella la capacidad de hacer uso de ArduPilot, el autopiloto de código abierto que hemos mencionado en el Glosario de términos.

La página oficial de NAVIO2 nos facilita la documentación necesaria para la configuración básica de un procesador de vuelo Raspberry Pi con el hardware del autopiloto NAVIO2 y el sistema operativo ROS (Robot Operative System).

NAVIO2 se presenta a nivel físico como un módulo o pequeña placa que se coloca sobre la Raspberry Pi mediante unos conectores que vienen de fábrica en ambos. Estos dos componentes han de atornillarse para asegurar la fijación mediante unas piezas de color dorado y sus propios tornillos que acompañan la NAVIO2 en su caja.

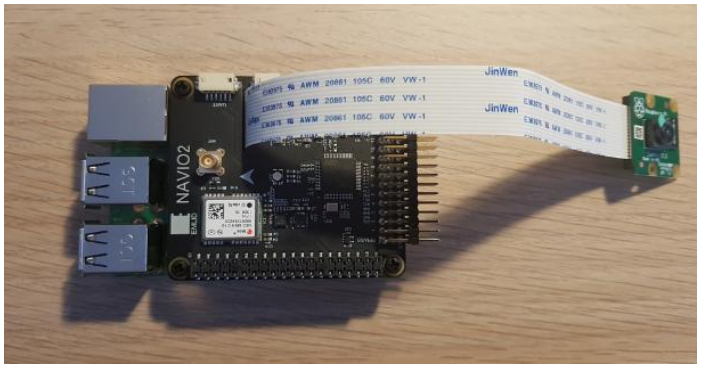


Ilustración 28 NAVIO2 instalado

A la hora de colocar el procesador y el hardware del autopiloto en el UA hay múltiples opciones, pero nosotros proponemos dos: una de ellas es la colocación de velcro con adhesivo en la parte posterior, de forma que podemos pegar y despegar el procesador de la nave cuando lo necesitemos. La forma alternativa, y que nosotros hemos utilizado, es el uso de una estructura fabricada mediante una impresora3D que, en combinación con unos pequeños cilindros huecos de goma, se crea una modificación que protege a los procesadores de los golpes inferiores y las vibraciones; a esto le hemos añadido el velcro adhesivo, asegurando así la fijación de la estructura.

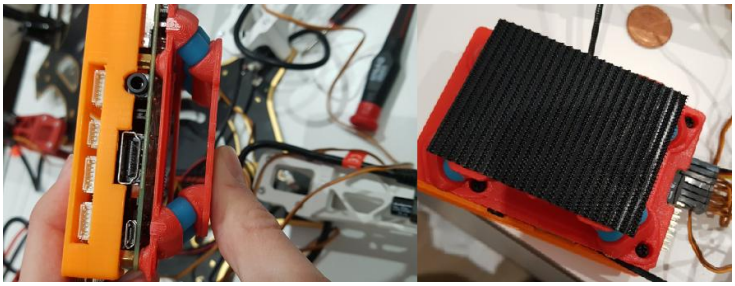


Ilustración 29 Amortiguadores de instalación opcional

Una vez introducido estos dos componentes, podemos desarrollar el concepto **Companion Computer**: es el procesador de vuelo adicional que se acopla al principal, con el fin de aligerar notablemente la carga de trabajo del procesador de vuelo. Además, el Companion Computer se puede utilizar cuando no se quiere depender en demasía del GS para el procesado de datos, algo muy común en el caso de los UAS que actúan en consecuencia de la información recibida mediante procesado de imagen: seguimiento o búsqueda de formas y colores, detección, vigilancia, lectura de rostros... Estas y otras acciones requieren de grandes potencias de cálculo de forma inmediata, y no es posible esperar a la recepción de los cálculos de la GS o arriesgarse a tener problemas con las comunicaciones (distancia, disponibilidad, dependencia, ...), por lo que todo procesado y cálculo se realiza en la propia nave. Aunque no vamos a incorporar un Companion Computer, el sistema estará preparado para hacerlo inmediatamente. Para ello, se ha pensado que se trate a su vez de una Raspberry Pi 3, y por tanto, todas las explicaciones dadas y por dar podrán aplicarse a este Companion Computer (instalación y configuración de software, comunicaciones, procesado de imágenes...).

3.2.3.4 Control terrestre

Es necesario poseer una forma de introducir códigos o modificar la configuración del procesador a través de algún medio, más allá de hacer uso de un USB. Cuando lo que deseamos es enviar órdenes, modificar trayectoria o controlar cualquier aspecto del UAS, no podemos depender de un cable para ello. Podremos pedir datos del vehículo, estado de la conexión, batería restante, distancia respecto del origen,...

Con esto, lo que se busca es poseer varios medios de control y configuración inalámbricos:

- **Mando de control (radio):** Para el control manual de vuelo y el control de modos, la mejor opción es la de hacer uso de un mando de control especializado en vuelo. En nuestro caso haremos uso del FlySky XR8 8CH, el cual viene con el receptor de radio

específico que hemos colocado en uno de los brazos. Mediante el QGroundControl habremos realizado el calibrado de la frecuencia de señal para concretar los modos, a la par de configurar el mando (lo veremos en el siguiente capítulo).
Durante el uso de esta forma de control, convierte nuestro UAS en un RPAS, puesto que hace nuestro proyecto en un sistema controlado de forma remota por una persona. Nosotros hemos utilizado el modelo Taranis X9D Plus, cuya configuración aparece en el Apéndice 5, la cual se realizará una vez esté instalado QGroundControl.

Ilustración 30 Receptor de radio que se conectará en el UA

- **Telemetría:** mediante el uso de dos antenas, nuestra computadora (GS) y un proceso de configuración (ver apéndice 2), podremos entrar en el código del procesador a través de una consola (*Putty*) y ejecutar comandos desde la GS. Estas dos antenas son conectadas a través de USB.

- **WiFi**: Llegados a este punto, si se ha seguido el apéndice de configuración inicial del procesador, nuestra aeronave ya poseerá acceso a la red WiFi utilizada por la GS. Esto nos permitirá, tras una breve modificación de parámetros (ver apéndice), entrar a la consola mediante el comando *rlogin* a través de la consola de comandos de la GS. Además, podremos descargar aplicaciones y actualizaciones necesarias para el funcionamiento del RPAS.

3.2.4 GNSS

En el presente proyecto hemos trabajado con 2 dispositivos GNSS: uno en la estación de tierra para georreferenciarla y realizar las pruebas de posición relativa (véase pruebas), y otro embarcado.

Este componente nos permitirá detallar la posición de nuestro RPAS y protocolos preprogramados, como el de regreso al punto de despegue desde la posición del momento en caso de que haya alguna emergencia o fallo.

En nuestro caso, el hardware del autopiloto NAVIO2 cuenta con un puerto específico para GNSS, por lo que haremos uso de la antena GNSS de NAVIO2, y un receptor para la GS (en nuestro caso, el modelo de la marca GlobalSat).

Ilustración 31 Antena GNSS para Navio 2

Ilustración 32 Receptor GNSS GlobalSat

Para la configuración de este componente, consultar el apéndice.

3.2.5 Sistema de transmisión de imagen

Se ha hecho uso de una cámara especializada para su aplicación en UAS. Está diseñada para soportar las vibraciones del vuelo y capturar imágenes nítidas pese al movimiento. Sus imágenes son enviadas al transmisor de imágenes de a bordo, y estas serán enviadas al receptor, el cual en nuestro caso estará en el ordenador conectado. Una vez pasen por el receptor, pasarán a la digitalizadora, la cual convertirá la señal recibida en datos digitales que se pasarán al ordenador. Ahora que las imágenes tomadas están siendo digitalizadas, se pueden procesar.

Nuestra cámara será una Foxeer de 600TVL CCD, modelo HS1177 con lente de 2.8mm.

Ilustración 33 Cámara Foxeer a la izquierda. Equipo de transmisión y recepción a la derecha

3.3 Software

Tal y como se ha expresado anteriormente, uno de los requerimientos fundamentales es el de hacer uso de software libre multiplataforma. Detallamos a continuación los elementos necesarios.

3.3.1 Ground Station

En este apartado, nos centraremos en instalar todos los programas necesarios, tanto en ordenador como en el procesador de vuelo, así como su configuración. Tras este punto la aeronave será capaz de despegar y funcionar siguiendo nuestras órdenes.

Este elemento es una parte externa al UAV, pero sigue siendo un elemento fundamental del UAS: desde la Ground Station (GS) podremos programar el software del procesador de vuelo utilizado, controlar el UAV, localizarlo, modificar sus protocolos o funciones, recibir datos de la aeronave y aliviar la carga de procesado de la nave. Por otra parte, este segmento también es fundamental para la preconfiguración de la nave, así como de su calibración.

Para facilitar el proceso, instalaremos una partición con el sistema operativo Kubuntu (variante de Linux), puesto que en él podremos instalar los siguientes programas (en puntos posteriores se detallarán sus características más relevantes:

- QGroundControl: Nos permitirá un control total del vuelo de la nave, su calibrado, toma de datos (posición, inclinación, consumo,...) y planificación de la misión, siempre y cuando la nave haga uso de **MAVLink (*Micro Air Vehicle***

Link). Como alternativa a QGroundControl, en el caso de que solo se cuente con Windows como sistema operativo, se propone el uso de MissionPlanner.

- Python: Es un lenguaje de programación gratuito y abierto, el cual permite crear incontables programas y herramientas Software. Es utilizado para ejecutar el software que hará uso de las librerías de DroneKit y OpenCV.
- DroneKit: Es un conjunto de librerías que permite desarrollar aplicaciones ejecutables en un *Companion Computer* u ordenador a bordo, y comunicarnos con el autopiloto.
- OpenCV: Es una agrupación de librerías especializadas en la visión artificial. Puede funcionar en diversos lenguajes de programación, pero nosotros usaremos la versión de Python. Su instalación permitirá al futuro desarrollador trabajar con procesados de imágenes.

En este punto, estamos preparados para trabajar con el sistema embarcado.

3.3.2 Sistema Embarcado

A partir de este punto, guiaremos el proceso de instalación y configuración en base a este modelo de procesador de vuelo y hardware del autopiloto.

3.3.2.1 Sistema Operativo

Una vez elegidos hardware del autopiloto y procesador de vuelo, descargaremos el programa *Etcher* o cualquier otro programa que nos permita quemar imágenes en una tarjeta SD. Tal y como se deduce, necesitaremos una tarjeta SD de por lo menos 8GB. A continuación, descargaremos el software que utilizaremos en la procesador de vuelo: Emlid-Raspbian. Este software se podrá descargar en la página oficial de emlid, en el apartado de configuración (enlaces al final del documento del proyecto).

Raspbian es un sistema operativo libre basado en Debian y optimizado para el hardware de Raspberry Pi. Permitirá desarrollar códigos en el lenguaje de programación Python e instalar la librería de OpenCV necesaria para la realización del proyecto. El hecho de usar Emlid-Raspbian y no otro tipo de Raspbian, es porque este en concreto ha sido modificado por Emlid para que la Raspberry trabaje en tiempo real.

Utilizaremos *Etcher* para quemar la imagen de Emlid-Raspbian en la tarjeta SD. Antes de trabajar con el procesador de vuelo, recalcamos la necesidad de una fuente de alimentación para el procesador; recomendamos por el momento el uso de una batería de por lo menos 6000 mA/h y recargable. Llegados a este punto, conectaremos el procesador a un monitor mediante conexión HDMI, y un teclado que funcione con conexión USB. Tras la configuración del procesador de vuelo (detallado en el apéndice), contaremos con uno que funciona con Linux y ArduPilot, listo para ser controlado desde la consola terrestre (GS) con un cable USB, o desde un monitor y teclado propios para el procesador.

3.3.2.2 Companion Computer

Introducimos un nuevo componente optativo, y es el Companion Computer. Esta parte consiste en el complementario a la original, la cual se encarga del procesado de información y cálculos complejos, en el caso de que no se posea acceso a la estación terrestre o no se quiera arriesgar a perder información o tiempo en la transmisión y recepción de información.

Proponemos que este procesador de vuelo adicional posea el sistema operativo Emlid-Raspbian en caso de incorporarse (lo cual facilitará su instalación, ya que se está comentando en este propio documento), pero que también posea DroneKit y OpenCv, puesto que uno de los principales motivos para usar un Companion Computer es la necesidad de procesar imágenes con rapidez.

3.3.3 Control remoto

3.3.3.1 Radiocontroladora

Una vez hemos realizado estos pasos, ya podremos conectarnos al QGroundControl desde nuestro dispositivo, y con esto, hacer uso de la emisora o radiocontrol. QGC requiere de unos pasos previos de calibrado mínimo para la navegación:

1) Definición de la nave: de qué tipo nave se trata, cuántos motores, disposición de estos,…
2) Calibrado de la emisora o radiocontrol. Cabe decir que algunas emisoras requieren de una configuración específica; en nuestro caso, la Taranis ha requerido de un calibrado previo de modos de vuelo y valores de telemetría, así como un personalizado de alertas que no se han considerado necesarias. Se recomienda leer el apéndice 5 y el manual específico de la emisora.
3) Calibrado del acelerómetro y el giroscopio: el propio QGC nos guiará en este proceso.
4) Señal GNSS: es fundamental para alzar el vuelo el acceso a señal GNSS.

Este proceso explicarán más detalladamente en el apartado 3.4.1 "Vuelo con emisora". Llegados a este punto, se recomienda hacer una prueba de los motores para asegurarnos de que tanto software y hardware están correctamente configurados. En caso de que todo funcione correctamente, sería buena idea realizar una copia de la SD, puesto que en caso de error futuro, podemos volver a este punto sin perder los pasos anteriores. Si hay algún problema de software y no es posible encontrarle solución, sería plausible formatear la tarjeta SD y repetir los pasos anteriores. Si se observara algún error con los motores, lo más prudente sería revisar las soldaduras.

Hasta aquí hemos presentado los pasos básicos para llegar a controlar la nave mediante radiocontrol, es decir, darle al UAS la condición de **RPAS**.

3.3.3.2 WiFi

Para este apartado, ya hemos realizado la configuración en los pasos previos. Cuando deseemos controlar el dron mediante WiFi, tan solo se deberá escribir en la consola de comandos: `sudo rlogin pi@xxx.xxx.x.xx`, siendo "`xxx.xxx.x.xx`" la IP del procesador de vuelo de la nave. Para conocer la IP del UA, tan solo debemos escribir, en la consola de comandos del procesador: `ifconfig`. Si el proceso funciona exitosamente, veremos cómo la consola del procesador se ejecuta dentro de la consola del ordenador.

Esta conexión funcionará siempre y cuando ambos estén en la misma red local y el ssh de la Raspberry esté activado.

3.3.3.3 Telemetría

Para utilizar este método, es necesario hacer uso de dos antenas telemáticas: una que se encargará del control telemático y otra que se centrará en las transmisiones MAVLINK.

Como su proceso es más extenso, se detallará en el Apéndice 2.

3.3.4 Control desde la computadora

Definiremos brevemente los programas utilizados que permitirán el control del UA desde la computadora.

3.3.4.1 Python 2.7

Python es bien conocido por ser un lenguaje de programación. Este lenguaje ha resultado tan popular debido a su versatilidad y a su condición de Open Source. Este lenguaje creado en los 80, se enfoca en el funcionamiento en base a objetos y en numerosas librerías que lo hacen de un lenguaje en continuo crecimiento, debido a la comunidad de usuarios que amplían los horizontes de este lenguaje. El lenguaje resulta sencillo a nivel gráfico, gracias a su organización y clara presentación (identación).

El hecho de haber elegido Python, además de por la condición de ser libre, es por su potencial en el campo de los UAS mediante el uso de ArduPilot a través de DroneKit (a continuación desarrollado). No hay que olvidarse del potencial que posee en el campo del procesado de imágenes gracias a su compatibilidad con OpenCV.

La selección de Python 2.7 en lugar de Python 3 se debe a que todavía existen ciertos errores en Python 3 que dificultarían mucho este proyecto.

Ilustración 34 Logo Python

3.3.4.2 DroneKit

DroneKit consiste en un conjunto de librerías que permite crear aplicaciones en un *Companion Computer* embarcado, así como comunicarse con ArduPilot y permitir un control de vuelo de baja latencia. Estas aplicaciones permiten mejorar el autopiloto, ofreciendo mayor inteligencia y adaptabilidad a la misión de vuelo; así pues, esto permitirá realizar tareas que requieren completar numerosos cálculos en poco tiempo.

Por otro lado, DroneKit puede ser ejecutado desde la estación terrestre (computadora), y permitir la comunicación y transmisión de datos. En nuestro caso, en las pruebas, haremos uso del programa *vehicle_state* y de *follow_me.*

La API (Application Programming Interface) se comunica con los vehículos a través de MAVLink, ofreciendo acceso a la telemetría del vehículo, su estado y sus parámetros. A su vez, permite la gestión y el control de vuelo del vehículo.

Junto a estas cualidades, cabe añadir su extensa y activa comunidad de usuarios, que permite el crecimiento y desarrolla de esta tecnología.

Algunas aplicaciones concretas son: recepción de parámetros, notificación asíncrona de cambios de estado o modo, guiado a una posición específica, envío de mensajes personalizados, creación de *waypoints* o la modificación de configuraciones de radiocomunicación.

Ilustración 35 Logo de 3D Robotics, desarrolladora de DroneKit

Ahora pasaremos a comentar el proceso seguido para poder hacer uso de los programas vehicle_state.py y follow_me. Antes de nada, recordar que necesitamos instalar DroneKit, Python y de descargar dichos programas desde la web oficial de DroneKit (consultar bibliografía).

Configuración vehicle_state.py (primera prueba del apartado 3.4.4)

Para empezar, es fundamental modificar el código de vehicle_state, para lo cual entraremos en él y modificaremos la línea 42. En esta línea, detallaremos la velocidad a la que funciona nuestra telemetría:

```
vehicle = connect(connection_string, wait_ready=True, baud=57600)
```

Nuestra telemetría funciona a 57600 baudios. El baudio es la unidad de medida que define el número de símbolos por segundo a través de la transmisión digital. Como orientación:

$$n = \frac{rb}{rs}$$

$n =$ número de bits por nivel para la codificación de línea

$$rb = régimen\ binario$$

$$rs = tasa\ de\ símbolos$$

Una vez definidos los baudios, tenemos que desactivar un temporizador por defecto de DroneKit, en el cual cierra la conexión en caso de que en 30 segundos no reciba ninguna señal. Esto se debe a que hacer uso de un puerto telemétrico que funciona a 57600 baudios requiere de más tiempo para conectarse; por esto, necesitaremos modificar este contador o *timeout*. Para ello, debemos de buscar el archivo __init__.py, el cual encontraremos en el directorio Dronekit /usr/local/lib/python2.7/dist-packages/dronekit . Una vez aquí, abriremos el archivo y se modificará la línea 278, cambiando 30 por una cantidad alrededor de 180, tal que:

```
heartbeat_timeout=180
```

Otra posibilidad, sería cambiar la línea 2431, pero esta se centraría en evitar el *timeout* en lugar de alargarlo:

```
def set(self, name, value, retries=3, wait_ready=False)
```

Una vez modificados los parámetros: guardamos, cerramos el código y desde la consola de Kubuntu, lanzamos:

```
python vehicle_state.py --connect /dev/ttyUSB0
```

Configuración follow_me.py (segunda prueba del apartado 3.4.4)

Antes de empezar se debe configurar el ordenador para que trabaje correctamente con el dispositivo GNSS USB; para ello, se consultará el Apéndice 3.

Abriremos el código de follow_me.py y modificaremos la línea 108, donde cambiaremos el valor de time sleep a 0.02

```
time.sleep(0.02)
```

Esto se reduce para que las peticiones de conexión al GNSS del ordenador (conectado anteriormente) se realicen más rápidamente.

Finalmente guardamos, cerramos el código y ejecutamos el programa:

```
python follow_me.py --connect /dev/ttyUSB0
```

Recordamos que ttyUSB0 será el dispositivo GNSS conectado al ordenador

3.3.4.3 OpenCV

Es una librería que está compuesta por funciones de procesamiento de imágenes y visión artificial. Esta librería, no solo es de uso libre, sino que puede ser utilizada en el sector académico y comercial (bajo ciertas condiciones). Esta librería es famosa por su potencial, ya que con unas pocas líneas es capaz de generar eficientes algoritmos de procesado.

El núcleo de la librería está programado con C++, pero posee compatibilidad con C++, C , Java, Python y MATLAB. Nosotros haremos uso de la versión de Python, pero junto a Numpy (librería de tratamiento de matrices), mediante la cual accederemos a las matrices de trabajo de OpenCV. Como curiosidad, cabe añadir que su desarrollo inicial fue llevado por Intel, para después ser llevado por el laboratorio de robótica Willow Garage, y hoy en día por la empresa de visión artificial Itseez, comprada por Intel en 2016.

Ilustración 36 Logo OpenCV

Ahora pasaremos a detallar el proceso de configuración que hemos seguido para la detección de color, aplicada en las pruebas de campo

Detección de color (prueba del apartado 3.4.5)

Primeramente, entraremos en el código del programa opencv_stream.py, en el cual modificaremos en la línea 17 el valor de src de 0 a 1:

```
vs = VideoStream(src=1, resolution=resolution, framerate=fps).start()
```

Esto se hace porque el valor 0 se asigna para aplicar el programa a una cámara web del PC, mientras que 1 se aplica a una cámara externa; nuestro caso es este último, puesto que haremos uso de una digitalizadora de vídeo USB.

Por otra parte, si tenemos instalada la versión 2.7 de Python (lo que hemos recomendado anteriormente), no será necesario cambiar de entorno virtual antes de ejecutar el programa; en caso contrario, deberemos ejecutar el comando `workon cv11`.

Ejecutaremos finalmente el programa mediante :

```
python opencv_stream.py
```

Una vez se ejecuta el programa, la selección del programa usado se seleccionará mediante la modificación de los parámetros que aparecerán en una GUI. Estos cambios harán que el programa vaya aislando colores y eliminando otros, binarizando todos los colores y dejando como valor 1 los que queremos detectar y 0 los que queremos obviar.

Para aclarar todo el proceso seguido hasta ahora, se resume el funcionamiento del sistema en la ilustración 37, donde se numeran los métodos de control a distancia (WiFi: I, Radio: II, Telemetría: III). Se representan en ítems negros los elementos Software. Cuando uno de estos programas requieran de otro para funcionar, se encontrarán dentro de otro ítem negro, el cual se encontrará en el ítem correspondiente al Hardware donde es ejecutado.

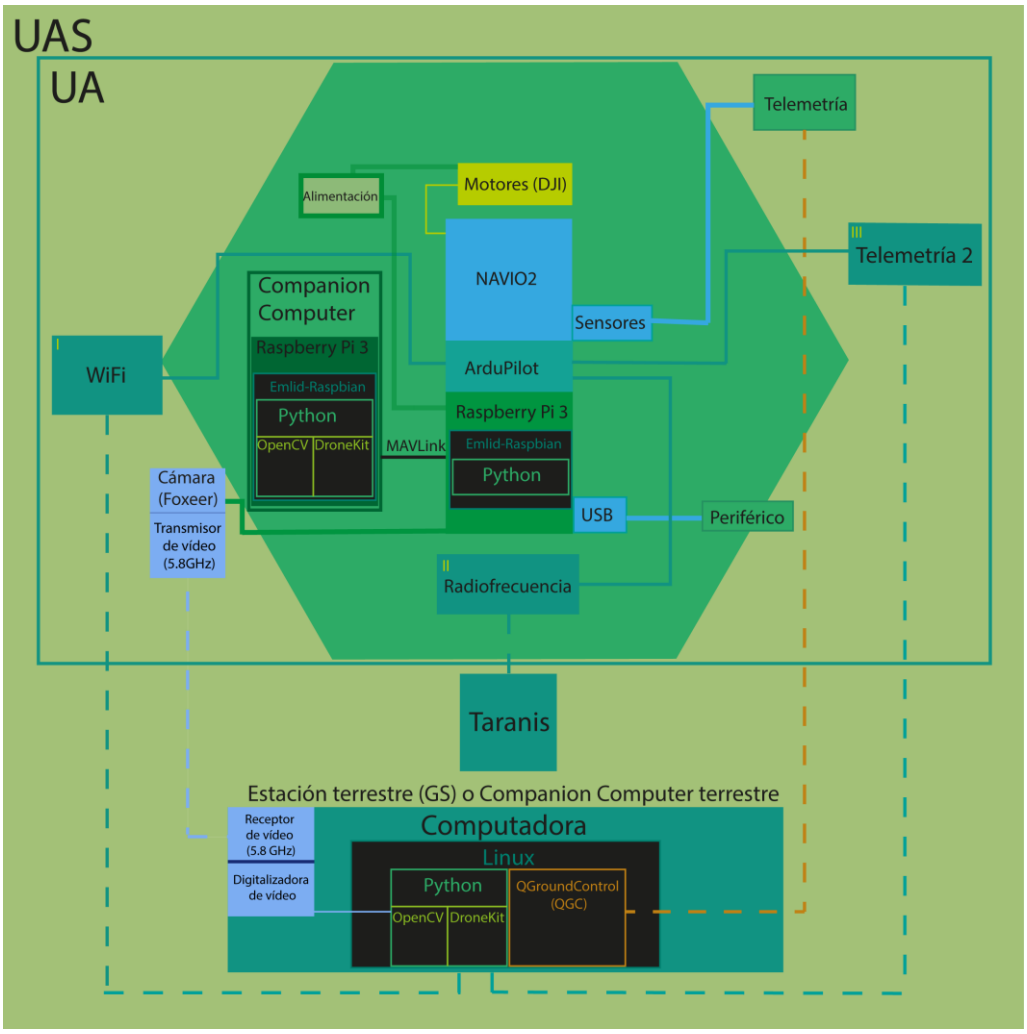


Ilustración 37 Esquema final del UAS

3.4 Pruebas

3.4.1 Comprobaciones previas

Antes de colocar las hélices y realizar alguna operación, debemos realizar ciertas comprobaciones por la seguridad del usuario y del UA.

- **Sistema eléctrico:** Todo el cableado ha de estar bien recogido y sujeto, que facilite posteriores revisiones del UA y se descarte el riesgo de daño al engancharse con alguna de las hélices. Las soldaduras serán bien revisadas, puesto que el hecho de que un solo filamento conecte con el cable opuesto, provocará problemas con los motores y que o bien no vuele o se caiga durante el vuelo.
- **Atornillado y fijaciones:** Debemos asegurarnos de que los componentes se encuentren fijos, todos los tornillos colocados y apretados correctamente.
- **Alimentación de motores:** Cuando conectemos el UA, podremos escuchar un sonido intermitente a la par que se mueven ligeramente los motores, puesto que es una comprobación de estos. Si se observa que todos se mueven, los seis motores "suenan" (es un sonido intermitente por motor) y se apagan a la vez, es señal de que los seis motores están correctamente alimentados. Si alguno no suena o no se mueve o tarda más que los demás en apagar su sonido de comprobación, se recomienda revisar el sistema eléctrico y las soldaduras.
- **Giro de los motores:** Es de vital importancia la revisión del sentido de giro de los motores, puesto que un solo motor girando en sentido contrario puede suponer una inversión de la nave en pleno vuelo y por tanto, un impulso directo hacia el suelo. Para comprobar esto, se adjunta la ilustración 38, extraída de la página oficial de DJI, en la cual se muestra el sentido de giro que debe presentar cada motor. En nuestro caso, debemos de fijarnos en el modelo 4 (se ha adjuntado diversos modelos en el caso de que el usuario haga uso de otro tipo de multicóptero). Recordamos que las patas rojas indican la parte frontal del UA.

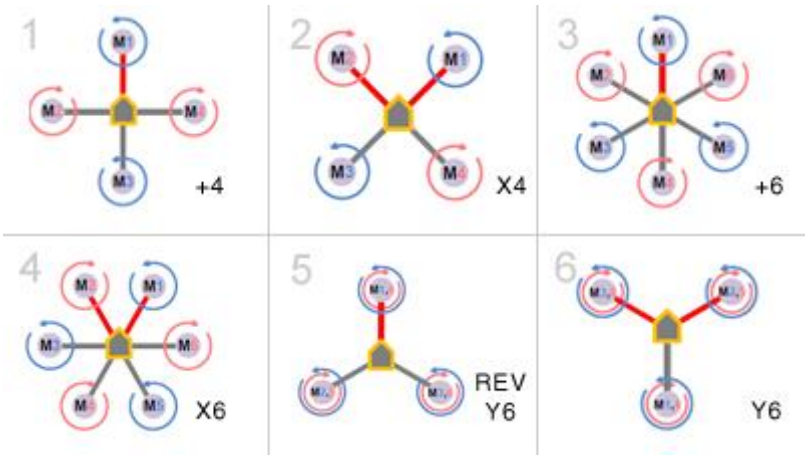


Ilustración 38 Definición del sentido de vuelo de los motores. Extraído de la página oficial de DJI

Para comprobar el sentido, se recomienda colocar algún tipo de papel doblado de forma que encaje con el motor y gire con él (pero que no lo atranque); de esta forma, cuando

decelere observaremos el giro del papel y con él, el del motor. Una vez colocado, armaremos los motores y los aceleraremos. Para armar los motores, tan solo debemos encender el procesador de vuelo y la Taranis (tras su configuración) y colocar las dos palancas en la posición indicada en la siguiente ilustración:

Ilustración 39 Imagen de la posición de palancas que arma los motores

Acto seguido soltaremos las dos palancas, para colocar la izquierda en la posición previa al armado e inclinarla ligeramente hacia adelante. Esto debería hacer que los seis motores aceleraran.

En caso de que uno de los motores no funcione en el sentido adecuado, tan solo deberemos cambiar dos de los tres conectores que unen el variador con el motor. Debido a que los motores funciones con corriente trifásica, el cambiar dos conectores provocará un desfase de voltajes que provoca la inversión de sentido de giro.

Ahora, tras aprender a armar los motores y revisar todos los requerimientos previos al vuelo, pasamos a la realización de las pruebas.

3.4.2 Vuelo con emisora

Para alzar el vuelo, como bien hemos dicho anteriormente, deberemos abrir primero el QGroundControl. En él, se abrirá esta ventana, la cual debería mostrar nuestra localización aproximada en caso de que la señal GNSS se alcance correctamente.

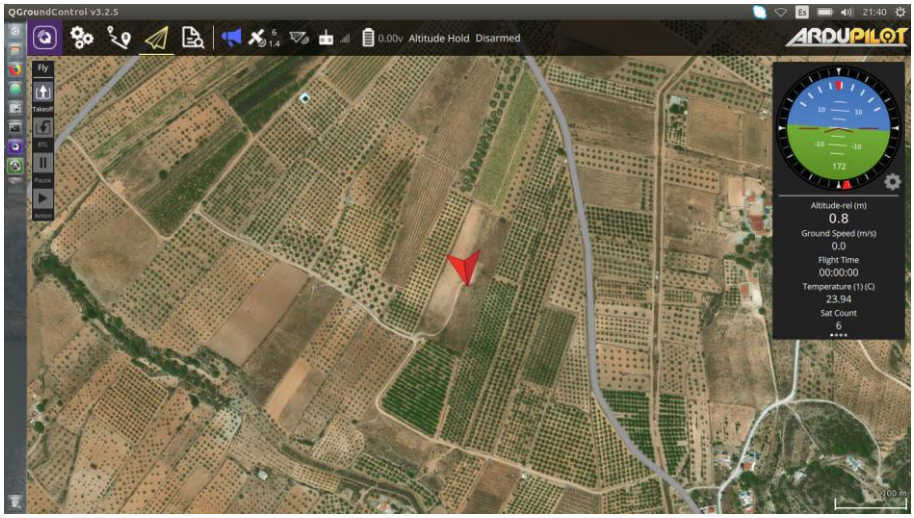


Ilustración 40 Menú inicial QGroundControl

El propio programa irá señalando qué elementos faltan por configurar, pero iremos apoyando dicha guía desde este escrito:

1) **Definición de la nave:** de qué tipo nave se trata, cuántos motores, disposición de estos,…: Seleccionaremos el icono de los engranajes, el cual nos llevará al apartado de configuración. Seleccionamos la opción *Airframe config*, donde definiremos el tipo de aeronave; en nuestro caso: hexacóptero (6 motores) en forma X

2) **Calibrado de la emisora o radiocontrol:** Cabe decir que algunas emisoras requieren de una configuración específica; en nuestro caso, la Taranis, ha requerido de un calibrado previo de modos de vuelo y valores de telemetría, así como un personalizado de alertas que no se han considerado necesarias. Se recomienda leer el apéndice correspondiente y el manual específico de la emisora.
   Para este punto, seleccionaremos *Radio Calibration*. QGroundControl nos dará órdenes específicas de accionar alguno los controles de la radiocontroladora uno a uno. Tras la acción exitosa de un control, dará la órden de usar otro; en otras ocasiones pedirá que mantengamos el cursor izquierdo (**el encargado de la aceleración**) en una posición mientras modificamos el derecho (**encargado de la inclinación de la nave**). Finalmente, se realizará una comprobación de los interruptores y de sus posiciones.
3) **Calibrado del acelerómetro y el giroscopio:** el propio QGC también nos guiará en este proceso. Para este punto, seleccionaremos la pestaña *Sensor Calibration* y veremos

que se marcan dos sensores concretos mediante puntos verdes: *Accelerometer* y *Compass*.

En *Accelerometer* (se requiere hacer este primero antes de calibrar *Compass*), se nos pedirá que giremos la aeronave de diferente forma cada vez. Una vez se haya inclinado del todo, se pulsará el botón *Next*, tras lo cual se nos dará otra orden. Una vez se haya realizado toda orden de forma satisfactoria, el programa nos informará de ello.

En *Compass*, se nos pedirá que giremos el UA alrededor de todos los ejes (alabeo, guiñada y cabeceo), y a medida que vayamos cumpliendo esta tarea, la barra verde se irá rellenando hasta que se haya calibrado.

Ahora, si el programa lo pide, reiniciar nave y programa.

4) **Configuración de modos**: debemos definir los modos de vuelo, los cuales se intercambiarán mediante el uso de los interruptores predefinidos en la Taranis. En nuestro caso, hemos configurado *AltHold* y Pos*Hold*.
5) **Colocación de las hélices:** estas se deben de poner de cierta forma, ya que hay dos tipos de ellas, cada una para el sentido de un motor. Los dos tipos se diferencian por el color de la parte superior de cada una: plateado y negro. En la siguiente ilustración podemos observar cómo se debe colocar cada hélice. Estas se dejan caer sobre el motor y se guiran en el sentido indicado por la indicación grabada en cada hélice, hasta hacer tope; acto seguido comprobaremos que no se sale al tirar de ellas.

Ilustración 41 UA con las hélices colocadas

6) **Señal GNSS:** es fundamental para alzar el vuelo el acceso a una señal GNSS.
7) **Alimentación:** se recomienda revisar la batería de la radiocontroladora y las baterías del UA, puesto que el agotamiento de estas podría ser fatal para la nave en caso de caída a gran altura. Detallaremos cómo se comprueba el estado de las baterías y cómo se recargan en su propio apéndice.

Es peligroso realizar un vuelo sin el QGC, ya que sin este software no tenemos un control de lo que sucede en la aeronaves: averías, satélites disponibles, errores emergentes...

Ahora que hemos acabado de configurar QGroundControl, pasamos a la realización de la prueba de vuelo mediante radiocontroladora.

Encendemos la nave, la radiocontroladora y el QGC de nuevo. Acto seguido, armaremos los motores tal y como se explica en el apartado 3.4.1 en la comprobación **Giro de los motores** y se acelerará suavemente mediante la palanca izquierda, mientras que con la derecha se controlará la inclinación de la nave.

Ilustración 42 El autor del proyecto realizando la prueba de vuelo con radiocontroladora

En primer lugar, volamos en *AltHold*, con el que observamos que ArduPilot se encargaba de mantener la altura cuando no se le enviaba ningún comando, así como mantener la estabilidad.

En el modo Pos*Hold*, el UAS contrastaba su posición con GNSS y corregía altura, posición e inclinación. Durante la prueba, se pudo observar esta corrección, puesto que el viento hacía que el UA se desplazara cuando estaba en suspensión en el aire, por lo que este corregía su posición constantemente para vencer al viento.

3.4.3 Vuelo con QGroundControl

En el anterior apartado hemos usado QGroundControl para la configuración y supervisión del vuelo, pero las órdenes han sido dadas desde la radiocontroladora.

La prueba de vuelo con QGC consistió en trazar una ruta en el mapa del programa, y al enviar la orden, el UA siguió ese trazado sin necesidad de usar la radiocontroladora.

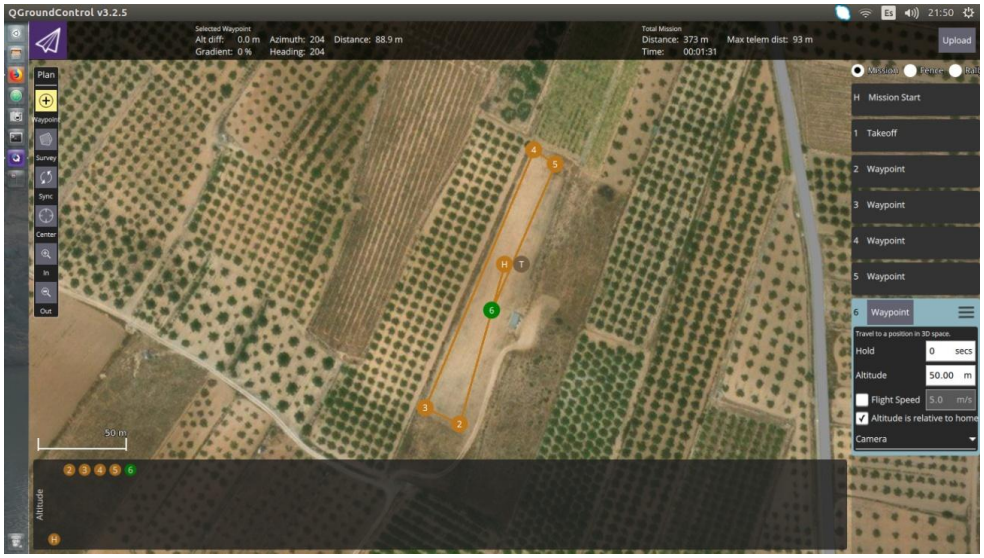


Ilustración 43 Captura de la ruta generada para la ruta del vuelo con QGroundControl

3.4.4 Vuelo con DroneKit

La prueba de este apartado se puede realizar mediante telemetría o WiFi. En nuestro caso lo lanzamos mediante telemetría.

En este apartado se incluyen dos pruebas:

Datos del vehículo (Comando: vehicle_state): No brindó los datos de vuelo del UA en tiempo real. Usamos:

```
python vehicle_state.py --connect /dev/ttyUSB0
```

Tras la ejecución del programa, se nos devolvió el siguiente texto:

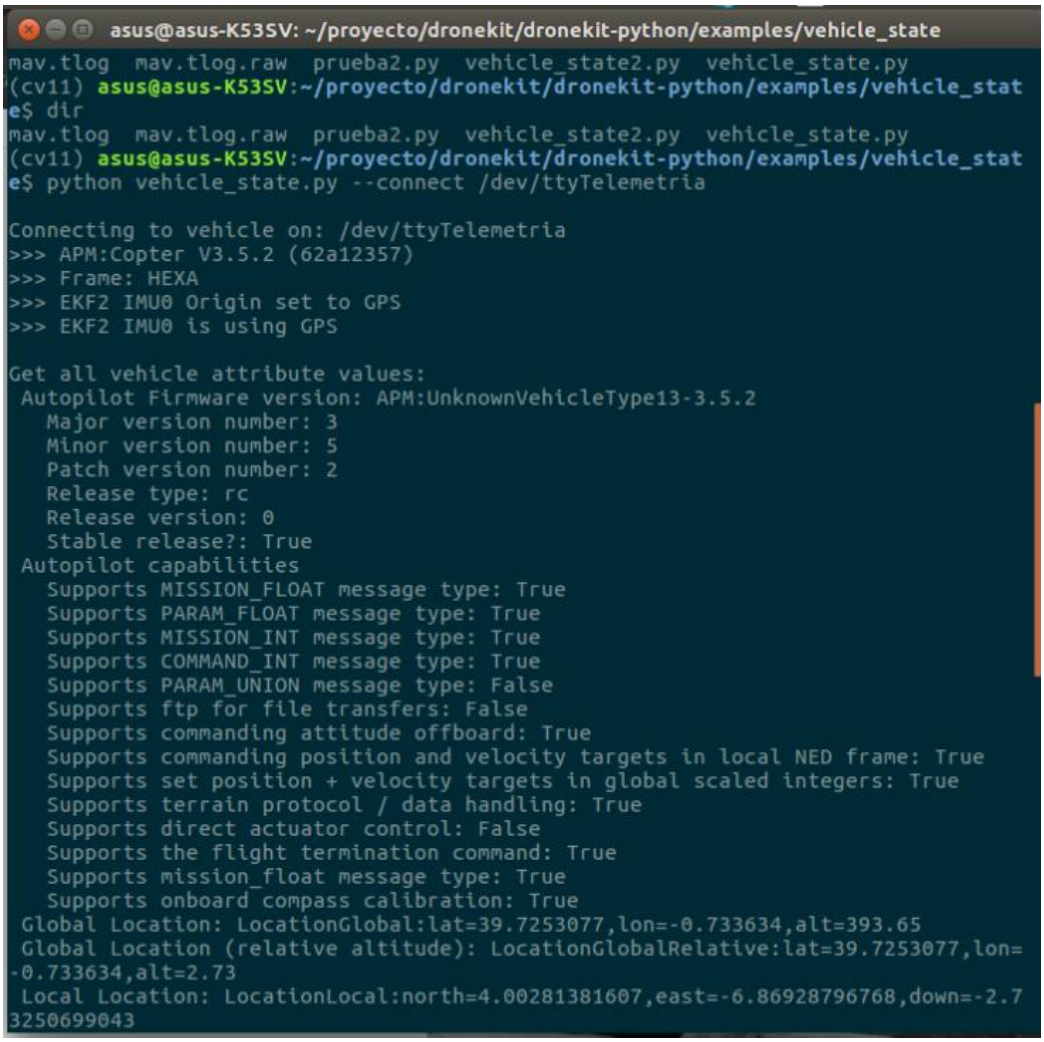


Ilustración 44 Output de vehicle_state

Donde podemos leer al final de la ilustración 44 los datos de vuelo ofrecidos.

Seguimiento (Comando: follow_me): este comando ordena al UA a seguir al receptor GNSS conectado a la computadora. Al haber usado un portátil, pudimos movernos con él y el receptor GNSS, mientras el UA seguía a una altura prudencial (de forma automática) a la persona que llevara el portátil.

Recordamos que para ejecutarlo, utilizamos:

```
python follow_me.py --connect /dev/ttyUSB0
```

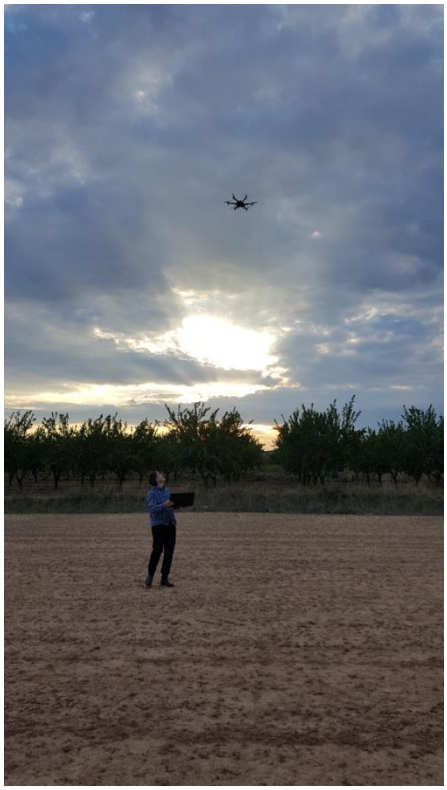


Ilustración 45 UA siguiendo la posición del portátil

3.4.5 Toma de imágenes (OpenCV)

Mediante la cámara instalada en el UA, enviamos imágenes al receptor de imagen conectado a la computadora, las cuales fueron posteriormente procesadas para detectar el color amarillo fosforescente, de forma que el UAS fue capaz de detectar una caja de dicho color.

La ejecución del programa se realizó mediante:

```
python opencv_stream.py
```

(recordamos que si no se posee la versión de Python 2.7, el procedimiento tiene un paso adicional; consultar el apartado 3.3.4.2

Para que el UAS reconociera este color, tuvimos que enfocarlo y procesar a mano la imagen, aislando el amarillo fosforescente mediante los parámetros de OpenCV, pasando así la imagen de color a imagen binaria. Así pues el amarillo fosforescente aparecía en blanco en la imagen procesada (valor 1) y el resto en negro (valor 0).

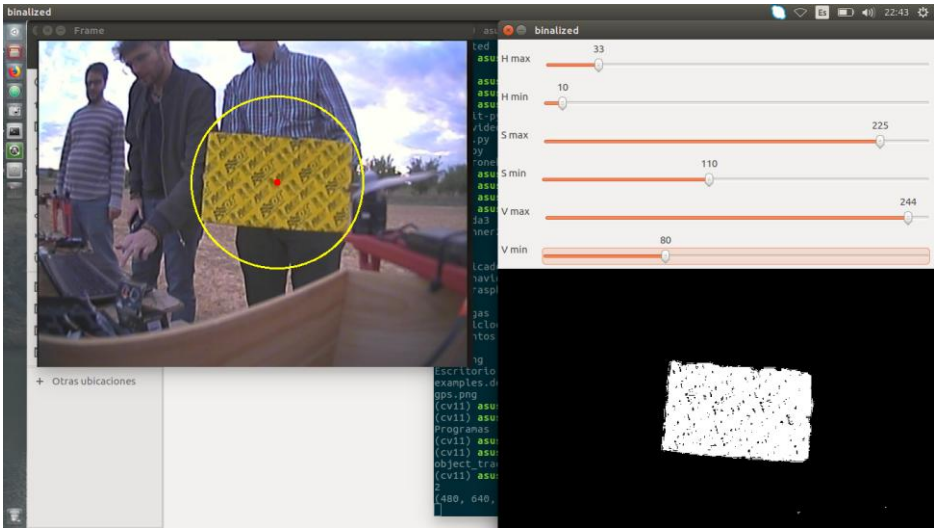


Ilustración 46 Procesando la imagen

En la ilustración 46 se observa cómo desde la cámara del UA se están capturando imágenes a tiempo real. A la derecha, se ve cómo se están configurando los valores para binarizar la imagen y así aislar el color deseado, y como resultado, en la captura a tiempo real se señala el color que se está buscando.

Se están usando 3 parámetros: H, S y V (Hue, Saturation y Value, respectivamente), los cuales explicaremos a continuación:

- Hue (Matiz): Representado como un ángulo entre 0 y 360º. Cada valor corresponde a un color. OpenCV realiza la conversión de este valor a la escala de valores enteros: 0-180, por lo que se debe pasar a dicha escala si se obtiene el valor del color de una fuente externa con diferente escalado a éste.

- Saturation (Saturación): Resulta la distancia al eje de blanco-negro. Los valores se encuentran entre 0 y 100%. OpenCV realiza la conversión a la escala de valores enteros: 0-255.

- Value (Valor): Representa la altura en el eje blanco-negro. Los valores se encuentran entre 0 y 100%, en el que 0 siempre es negro. OpenCV realiza la conversión a la escala de valores enteros: 0-255.

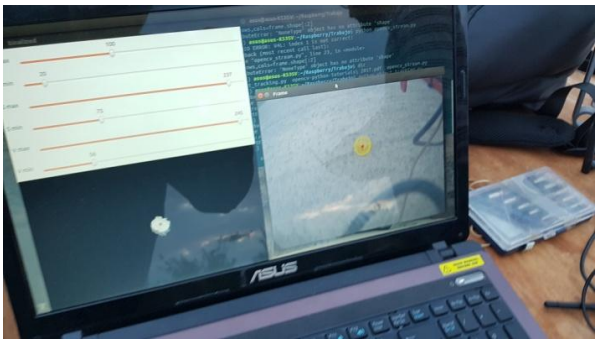


Ilustración 47 Localización del objeto durante el vuelo

4 Conclusión y trabajos futuros

4.1 Conclusión

Durante este proyecto se han observado las variables de las que depende un UAS y cuán complejo resulta, así como la necesidad de poseer un conocimiento multidisciplinar para llegar a realizar un buen diseño y montaje.

Se ha conseguido que el UAS funcione de forma satisfactoria, se controle a través de radiocontrol y se comunique con la estación terrestre mediante WiFi y telemetría. El UA se comunica con el software QGroundControl y permite su configuración, armado y control de vuelo. Se han conseguido lecturas satisfactorias de datos procedentes de sus sensores mediante comandos utilizando la librería DroneKit, así como su envío a la estación terrestre; se ha llevado a cabo la lectura de señal GNSS, y se ha puesto a prueba mediante el vuelo de seguimiento de un ordenador portátil en campo abierto.

En cuanto al procesamiento de imágenes, se han tomado imágenes mediante una cámara especializada para UAS y se ha realizado el reconocimiento de objetos sencillos mediante la librería OpenCV.

Con todo esto podemos decir que hemos alcanzado los objetivos planteados inicialmente, corrigiendo por el camino ideas preconcebidas sobre este campo, así como enfrentarnos a problemas que nos han hecho invertir más tiempo del deseado, destacando la recepción de una telemetría defectuosa, la soldadura defectuosa debido a la falta de experiencia en la realización de soldaduras sometidas a tanto esfuerzo, la necesidad de aprendizaje en el uso de software desconocido para el autor o el requerimiento de conocimiento polivalente en distintos campos que no han sido tan trabajados durante su formación.

4.2 Trabajos futuros

El futuro de este sistema, como bien ya se ha reiterado, se centrará en el desarrollo de aplicaciones basadas en el reconocimiento y procesado de imágenes, las cuales serán aplicadas al control de la nave. Se buscará ampliar su detección de entorno mediante la aplicación de un sistema de ultrasonidos de *Arduino*, que evitará a la nave colisiones directas frente obstáculos. También se plantea la ampliación de procesado mediante un Companion Computer embarcado en la nave.

Estos propósitos de futuro podrán ser utilizados para la realización de proyectos de fin de grado, máster o para la investigación.

5 Presupuesto

A continuación se muestra una tabla con los materiales utilizados en la construcción del UAS y sus costes.

Componente	Precio	IVA	TOTAL
Navio2 (Hardware de autopiloto+ Antena GNSS +Cableado + Power Cell)	216,53	45,47	262
BATERÍAS	111,57	23,43	135
Raspberry Pi 3	35,54	7,46	43
Alimentación Raspberry Pi 3 y Cargador baterías	45,45	9,55	55
Tarjetas memoria: Intenso 3413470 Micro SD clase 10 16GB	9,92	2,08	12
HDMI	7,44	1,56	9
L-link Teclado + Ratón LL-KB-816-COMBO USB Negro	7,44	1,56	9
Kit DJI F550 ARF	219,01	45,99	265
FlySky XR8 8CH antena PCB 16CH con Receptor	148,76	31,24	180
Kit Tx/Rx video 5.8Ghz 400mW	40,50	8,50	49
SkyRC Imax B6	34,71	7,29	42
Modulo telemetría 433Mhz 500mW MWC APM-PIX	37,19	7,81	45
Cámara Raspberry Pi 3 + Módulo RF	70,25	14,75	85
Otro material: cables, tornillería, conectores, etc.	24,79	5,21	30
TOTAL			1222 €

En cuanto al coste de los recursos humanos, al tratarse de un Trabajo de Fin de Grado se ha estimado una dedicación aproximada de **360 horas**, bajo las cuales un gran porcentaje de estas han sido de aprendizaje, solución de errores desconocidos, experimentación, horas de pruebas previas e instalaciones de software.

Tras las anteriores valoraciones económicas y de tiempo, nos ha parecido interesante tras lo aprendido realizar el ejercicio de considerar la posibilidad de ofrecer un servicio de diseño y construcción de UAS bajo encargo. Esta idea ha llevado al cálculo de ganancias frente a horas invertidas y gastos en su fabricación, tomando como referencia este Trabajo de Fin de Grado.

A grosso modo, podemos concretar una jornada completa de trabajo en el montaje del hardware (soldaduras, motores, atornillados, instalación eléctrica,...). Como servicio adicional, se propondrá otra jornada completa de instalación de software y configuración, de las cuales reservaremos 3 horas para pruebas de vuelo; esto hará de nuestro producto un UAS listo para operar.

Con lo dicho, considerando dos jornadas completas estándar, obtendremos unas **16 horas de trabajo**, aplicando un coste por los servicios del ingeniero de **40 euros por hora**. Esto hará del precio de la mano de obra **640 euros**.

Ahora que hemos planteado los honorarios del ingeniero, podemos aplicar una aproximación del precio de un UAS incluyendo el montaje y su configuración:

	Precio	Horas invertidas	Total
Honorarios del ingeniero	40 €/h	16	640 €
Materiales	1.222 €	N/A	1.222 €
Precio Total			**1.862 €**

Apéndice 1: Instalación y configuración del sistema operativo en procesador de vuelo.

Recordamos: descargamos emlid, quemamos en la SD mediante Etcher, y metemos la tarjeta en el procesador de vuelo.

Una vez hemos quemado la imagen del sistema operativo de emlid, pasamos a trabajar: podemos escribir directamente desde el ordenador mediante un navegador, o bien colocar la SD en el procesador y conectar a esta un teclado USB y una televisión mediante HDMI. Una tercera opción, es controlar el procesador mediante USB desde la computadora.

El sistema nos pedirá el nombre de usuario ("`pi`" por defecto) y la contraseña ("`raspberry`" por defecto).

Para conectar la tarjeta a la red WIFI, introduciremos el comando

```
sudo nano /boot/wpa_supplicant.conf
```

Y acto seguido, modificaremos los datos del texto interior a "network" para que quede tal que:

```
network={
  ssid="Nombre_de_nuestra_red_WiFi"
  psk="Contraseña_de_la_red"
  key_mgmt=WPA-PSK
}
```

Tras esto, saldremos del menú al guardar y reiniciaremos (`sudo reboot`).

Una vez tenemos acceso a la red, actualizamos las librerías hasta el momento:

```
sudo apt-get update && sudo apt-get dist-upgrade
```

Y expandimos el espacio del sistema (esto nos evitará problemas futuros).

```
sudo raspi-config --expand-rootfs
```

Tras esto, pasamos a configurar ArduPIlot:

```
sudo emlidtool ardupilot
```

En la interfaz que se abre, elegiremos en nuestro caso: "`copter/3.5/arducopter/enable/start/Apply`"

Esta selección habrá activado Arducopter. Acto seguido, seleccionamos `Quit` e introduciremos:

```
sudo nano /etc/default/arducopter
```

En este apartado, añadiremos la IP de nuestro ordenador en el apartado TELEM1 (sin borrar :14550, puesto que define el puerto correcto), y descomentaremos TELEM2 (implica eliminar la #).

Ahora, para confirmar los cambios, ejecutamos:

```
sudo systemctl daemon-reload
```

Iniciamos ArduCopter:

```
sudo systemctl start arducopter
```

(podemos pararlo usando `stop` en lugar de `start`)

Y definimos la ejecución de ArduCopter por defecto nada más iniciar el sistema:

```
sudo systemctl enable arducopter
```

(se anula con `disable` en lugar de `enable`)

Comprobamos que está activado:

```
systemctl is-enabled arducopter
```

En este punto, recomendamos actualizar de nuevo con el comando comentado anteriormente:

```
sudo apt-get update && sudo apt-get dist-upgrade
```

Hasta aquí se ha detallado la configuración básica del sistema operativo y del autopiloto del UA. Este procedimiento es el propio de la última versión de emlid-raspbian, puesto que anteriores versiones hacen uso de menos interfaces y más comandos (hablamos del caso de la configuración de autopiloto). En caso de dudas, se recomienda hacer uso de la documentación oficial de Emlid Navio2.

Apéndice 2: Configuración del control remoto mediante telemetría.

Para empezar, se instala la consola Putty desde la página oficial (ver bibliografía). Acto seguido, se instala y se configurará de la siguiente forma

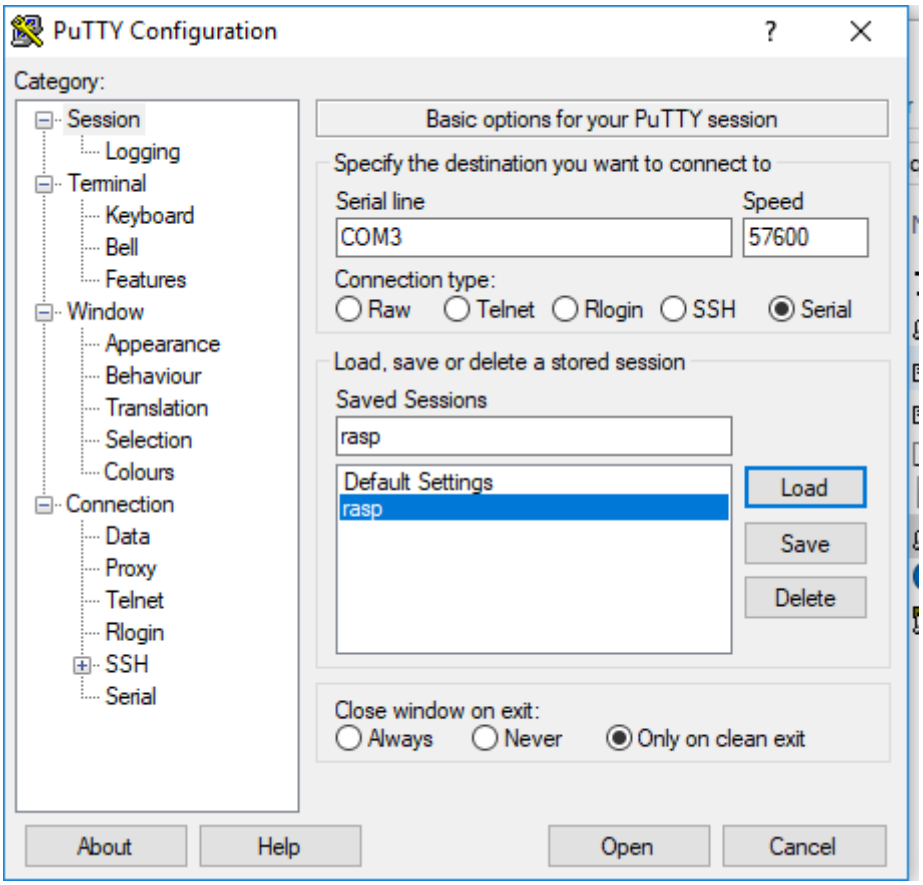


Ilustración 48 Captura de pantalla de la consola y su configuración

Por el momento, dejaremos así la configuración, a falta de pulsar el botón *Open*.

A continuación, aseguremos que estamos utilizando 57600 baudios en la comunicación:

Instalación del setserial: `sudo apt install setserial`

Comprobación de los baudios: `sudo stty -F /dev/ttyUSB0`

Asignación de 57600 baudios: `sudo stty -F /dev/ttyUSB0 57600`

Ahora pasamos a abrir la consola del UA para escribir los siguientes códigos:

Instalación del setserial: `sudo apt-get install setserial`

Comprobación de los baudios: `sudo stty -F /dev/ttyAMA0`

Asignación de 57600 baudios: `sudo stty -F /dev/ttyAMA0 57600`

La segunda parte de la configuración, resulta:

```
sudo cp /lib/systemd/system/serial-
getty@.service/etc/systemd/system
```

```
sudo cp /etc/systemd/system/ serial-
getty@.service/etc/systemd/system/
```

```
sudo mv /etc/systemd/system/ serial-
getty@.service/etc/systemd/system/ serial-getty@ttyUSB0.service
```

```
sudo mv /etc/systemd/system/ serial-
getty@.service/etc/systemd/system/ serial-getty@ttyAMA0.service
```

Y ahora entraremos en:

Computadora: `sudo nano /etc/systemd/system/serial-getty@ttyUSB0.service`

En el UA: `sudo nano /etc/systemd/system/serial-getty@ttyAMA0.service`

Escribiremos en ambas:

```
ExecStart=-/sbin/agetty --keep-baud 57600 %I $TERM
```

Habilitamos el servicio:

Computadora: `sudo systemctl enable serial-getty@ttyUSB0.service`

UA: `sudo systemctl enable serial-getty@ttyAMA0.service`

En este punto, tan solo hace falta pulsar el botón *Open* de la consola, y activar así el control remoto mediante telemetría.

Apéndice 3: Configuración del receptor GNSS

Primeramente, se instalará la librería gpsd mediante:

```
sudo apt-get install gpsd gpsd-clients
```

Ahora, se modificará la forma en que Linux reconoce al puerto del transmisor GNSS. Para ello, conectaremos los dispositivos (telemetría y receptor) y utilizaremos el comando lsusb
Aparecerá una pantalla de texto en la que se muestran todos los dispositivos conectados reconocidos. Los datos que nos interesan de esta página, son los llamados idVendor e idProduct de los dispositivos. Una vez los localicemos, crearemos un archivo de texto; en el caso de nuestro sistema operativo, Kubuntu, utilizaremos.

```
sudo kate /etc/udev/rules.d/reglasUSB.rules
```

En este nuevo archivo de texto, escribiremos los datos de nuestros dispositivos, tal que:

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="067b",
ATTRS{idProduct}=="2303", SYMLINK+="ttyUSBGPS"
SUBSYSTEM=="tty", ATTRS{idVendor}=="10c4",
ATTRS{idProduct}=="ea60", SYMLINK+="ttyTelemetria"
```

Tras esta escritura, guardaremos el archivo. Esto se ha hecho para que tras reiniciar, el sistema creará un puerto virtual con los nombres designados cada vez que detecte los dispositivos correspondientes (los detecta mediante el idVendor y el idProduct).

Finalmente, para evitar que gpsd se centre en buscar nuevos puertos USB, así como permitir que el equipo se conecte a la señal GPS, haremos:

```
sudo nano /etc/default/gpsd
```

y cambiaremos `DEVICES=""` por `DEVICES="/dev/ttyUSBGPS"`

En la misma página, escribiremos en `USBAUTO`:
`USBAUTO="false"`

Apéndice 4: Carga de baterías

Es de vital importancia asegurar la carga de baterías, y no basta con que haya algo de energía almacenada, sino que existe la necesidad de que haya un mínimo para que todo el sistema pueda alimentarse correctamente. Debido a que el sistema es de cuatro celdas, se requiere de 3.575 V para cada una, por lo que necesitaremos un total de 14.3 Vcomo mínimo recomendado para un vuelo seguro.

Las baterías que utilizamos tienen la capacidad de soportar perfectamente hasta 16.7 V, por lo que serán cargadas hasta ese punto. Estas baterías requieren de un proceso especial de carga, y es que necesitaremos, además de la fuente de alimentación, un cargador LiPo (Lithium-Polymer) con equilibrador.

Así pues, nosotros hemos hecho uso de un bloque de alimentación de Robbe de 12A, un cargador LiPo con equilibrador de IMAX y dos baterías (el receptor de imágenes requiere de alimentación también) de 6600 mAh de LiPo de stockRC. El montaje quedaría como la siguiente ilustración:

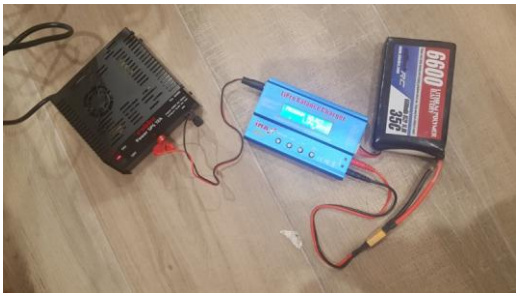


Ilustración 49 Montaje de carga de batería: a la izquierda el bloque de carga, al centro el cargador con equilibrador y a la derecha, la batería.

Para iniciar la carga, una vez esté todo conectado (el bloque se conecta a la instalación eléctrica doméstica) y con cuidado de que los cables no contacten entre sí y generen un cortocircuito, se pondrá el interruptor del bloque en *ON*. Tras esto, veremos cómo el cargador se enciende; pulsaremos una vez el botón de la derecha (*Start*) para seleccionar la carga por defecto: LiPo (primera imagen de la ilustración de a continuación). Acto seguido, mantendremos pulsado este botón de nuevo para iniciar la carga o descarga (segunda fotografía): elegiremos pulsar *Start* para cargar la batería, o *Stop* para descargarla (tercera fotografía, en la que se muestra el menú intermitente que anuncia la posibilidad de cargar o descargar). Esta segunda opción puede sorprender, pero es necesaria en el caso de que las baterías pasen un largo periodo sin utilizarse, ya que de lo contrario se deterioran. El cargador nos avisará de cuándo está suficientemente cargada la batería mediante una alarma sonora; en nuestro caso es 16.7V.

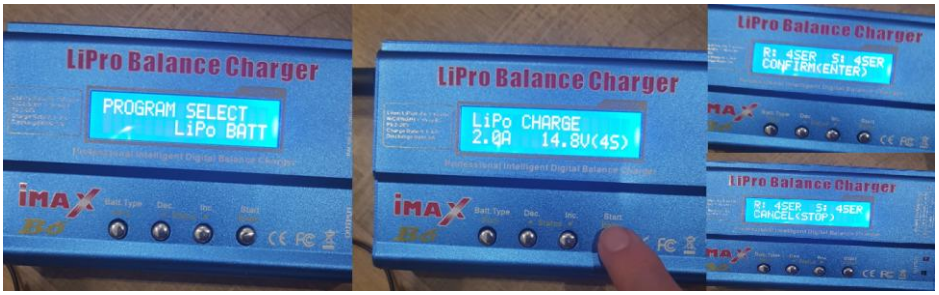


Ilustración 50 Proceso de selección de carga

Apéndice 5: Configuración de la radiocontroladora Taranis

Ilustración 51 La radiocontroladora Taranis en su maleta acolchada

Llegados a este punto, debemos saber que los controladores de la Taranis funcionan mediante canales: el movimiento horizontal de la palanca derecha es el Canal 1 o Alerón (Aileron), el Canal 2 o Elevador (Elevator) es el movimiento vertical, el Canal 3 o Motores (Engine) es el movimiento vertical de la palanca izquierda y el Canal 4 o Timón (Rudder) corresponde al movimiento horizontal de la palanca izquierda (en el caso del multicóptero, este control hace que el UA gire sobre su propio eje, es decir alrededor del eje Z o eje vertical).

En nuestro caso, al empezar de 0 o sobre otro modelo, esos canales puede que no estén asignados a ninguna maniobra o lo estén a maniobras incorrectas, por lo que debemos configurar ese aspecto para poder controlar perfectamente el UAS.

Con esto, estamos asociando una señal de potenciómetro a una salida del receptor del radiocontrol, por tanto esta configuración no es imperativa, sino que se ha propuesto de tal forma ya que es la configuración clásica de vuelo.

Las señales enviadas por la emisora (mando taranis) son recibidas por el receptor y unidas a un solo conector, el cual está conectado al hardware del autopiloto a través del SBUS. Esta señal es gestionada por dicho hardware para coordinar los motores y ejecutar la maniobra de forma correcta en consecuencia de la orden recibida desde la emisora.

Configuración

Para iniciar la configuración, se encenderá el mando deslizando la pestaña central hacia arriba. Veremos así que se enciende la pantalla azul. Pulsaremos el botón MENU, que es el superior izquierdo de los seis que están junto a la pantalla. Aparecerá un nuevo menú en el cual, si es la primera vez que entramos, no debería haber ningún modelo. De cualquiera de las formas,

elegiremos cualquier conjunto vacío que haya (nos desplazamos con el botón + y -, los cuales son botón superior e inferior derechos, respectivamente) y mantendremos pulsado ENT (botón inferior derecho). Aparecerá una ventana emergente, y seleccionaremos Create Model.

La nueva interfaz nos presentará 4 modelos, de entre los que seleccionaremos el primero, pese a que en la imagen aparezca un ala fija. Tras esta selección, pulsaremos repetidamente el botón PAGE para saltarnos todo el asistente de instalación, para acabar en el menú principal.

Una vez en el menú inicial, pulsaremos el botón PAGE y observaremos que en la parte superior derecha de la pantalla, el número 1/12 ha subido a 2/12. Esto es porque este botón nos permite navegar entre menús. Deberemos pulsarlo hasta estar en 6/12. En esta sección se observarán "canales" (CH1, CH2...) asignados a un número y a una maniobra.

Ya hemos llegado al punto de configuración de canales que buscábamos. Mantendremos pulsado ENT en el CH1, seleccionaremos edit y en la nueva página bajaremos a Source. Seleccionamos mediante enter y luego de nuevo seleccionamos Input, y veremos cómo parpadea la opción Thr. Esto es porque es el momento de asignar al CH1 la maniobra que queríamos: moveremos la palanca derecha de horizontal, lo que hará que Thr cambie a Ail. Confirmamos con ENT y pulsamos EXIT para configurar el resto de los canales. La cosa, como se ha dicho al principio del apéndice, quedaría:

- CH1: palanca derecha horizontal (Ail)
- CH2: palanca derecha vertical (Ele)
- CH3: palanca izquierda vertical (Thr)
- CH4: palanca izquierda horizontal (Rud)

Radioenlace

Ahora enlazaremos emisora con receptor, ya que de lo contrario podría enviarse la señal al receptor equivocado o de recibir una señal no deseada. En la siguiente figura observamos un pequeño botón en el receptor (este está conectado ya al UA), el cual al pulsarlo (mediante un bolígrafo o cualquier objeto delgado) se da energía al receptor, pero debe de mantenerse pulsado antes de encender el UA; sin embargo primero debemos trabajar con la emisora.

Configuraremos la emisora para estar en modo de búsqueda: desde la página inicial (la que aparece nada más encender la Taranis), pulsamos MENU, luego PAGE y una vez en el nuevo menú, navegamos hacia abajo hasta encontrarnos con la opción Receiver. Ya en esta opción, pulsando - se seleccionará Bind, y pulsaremos ENT. Oiremos entonces un sonido procedente del emisor. Llegado a este punto, realizaremos la operación comentada en el párrafo anterior: pulsar el botón del receptor y enchufar la batería del UA.

Ilustración 52 Receptor de Radio

Mientras emisor y receptor no estén enlazados, el receptor parpadeará de un color rojo. Veremos que al conectar la batería, si ha sido exitoso, se encenderá una luz verde. Ahora, apagamos el modo Bind (volviendo a pulsar ENT, además el sonido cesará) y apagamos la alimentación del UA. Cuando volvamos a encender el UA, la luz del receptor debería ser verde y fija, señal de que el enlace ha sido exitoso. La comprobación paralela es que en cuanto el enlace haya sido realizado, en el menú inicial aparecerá la calidad de conexión junto a la batería restante.

Modos de uso

Volvemos a la página de selección de canales (6/12) y navegamos hasta el canal 5 (CH5), lo seleccionamos y eligiendo Source haremos uso del conector de 3 posiciones SG y definiremos en qué posición queremos que el modo se active.

Ilustración 53 Detalle del conector SG

Para la tarea de selección de posición deseada, navegaremos hasta Switch (columna derecha de opciones) y elegiremos la posición (SG↑es posición hacia adelante,SG - es posición central, y SG↓es la posición inclinada hacia atrás). En nuestro caso elegiremos la opción de SG ↑. Tras esto, bajaremos a Multplx y cambiaremos a Replace (esto hará que al cambiar al posición, esta sustituirá a la anterior). Una vez realizadas estas operaciones, volvemos al menú de canales, y sobre CH5 mantendremos de nuevo el botón ENT y seleccionaremos Insert After. Se abrirá un menú igual al anterior, donde realizaremos la misma configuración excepto por el apartado Switch, donde nosotros seleccionaremos la posición del medio. Haremos la modificación de Multplx igual que antes.

Repetimos el proceso anterior, esta vez con SG↓. Ahora, para comprobar este paso, solo tenemos que permutar este conectar y observar cómo se seleccionan los canales configurados.

Con estos pasos hemos "ampliado" la configuración del CH5, dándole 3 posiciones de trabajo a través del conector SG.
A continuación repetiremos una cuarta vez la operación, pero con dos variantes: en lugar de colocar en Source SG, se usará SF y en Switch SF↓ .

Ilustración 54 Detalle del conector SF, el cual es de dos posiciones

Ilustración 55 Configuración resultante del canal 5

Esto se hace así, puesto que posicionar así el interruptor SF hará que el canal 5 esté "ocupado" por esta acción y no se ejecute ninguna operación de SG (si intentamos ahora utilizar SG, veremos que no lo detecta). Esto es debido a que este interruptor se usará para el modo de emergencia o "vuelta a casa": en caso de que haya algún problema, imprevisto o urgencia, se accionará este modo y el UA volverá al punto donde despegó.

Ahora pasamos al **QGroundControl**, donde recomendamos antes de pasar a los modos, pasar por la pestaña *Radio* y comprobar que detecta comandos de la emisora (se verá que los valores de la interfaz cambian a medida que modificamos los controles).

Una vez comprobada la conectividad en *Radio*, iremos a la pestaña de *Flight Modes*. Teniendo desactivado el modo de emergencia, es decir, SF no está en SF↓, y además SG está en SG↑, nos encontraremos en el Modo 1, *Stabilize*.

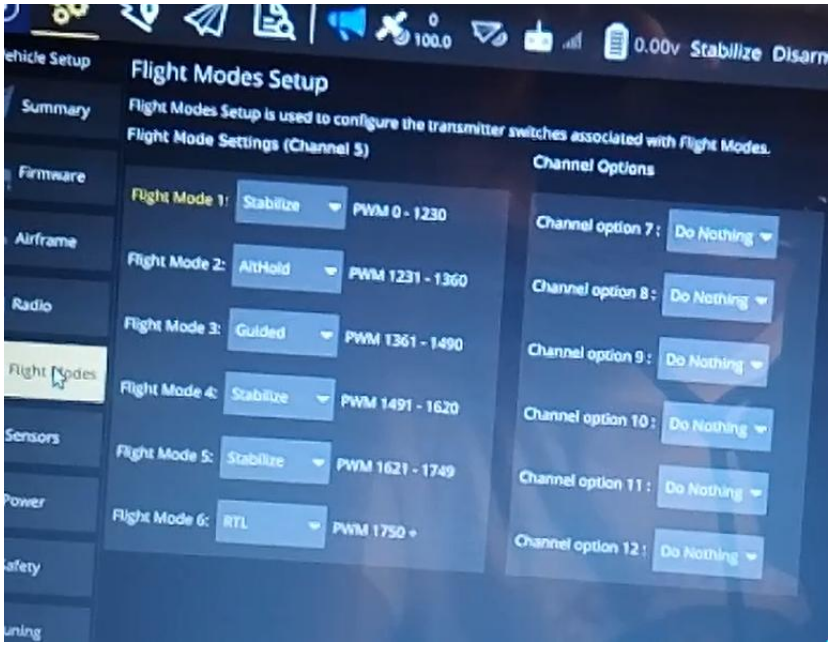


Ilustración 56 Fotografía de la pestaña de configuración de modos

Sin embargo, al modificar SG a SG- cambiamos a modo 4: esto es lo que debemos modificar, ya que debería pasar al 2; lo mismo sucederá con el resto de modos. Esto pasa porque el rango de salida generado por los controladores hace que la señal enviada se encuentre en un rango que ocupa modos que no queremos; esto se puede arregla modificando nuestros rangos de salida.

Para ello, volvemos en la Taranis al menú de configuración de canales y entramos en el modo 2 (recordamos: SG-) y nos desplazamos hasta *Offset*. Vemos que el modo 4 de QGC recoge un rango entre 1491 y 1620, y lo que queremos es estar entre 1231 y 1360, así que a medida que reduzcamos el rango de salida nuestro SG- mediante Offset, veremos que nuestro modo 2 se irá acercando modo 2 en QGC.

De esta forma, una vez asignado el rango y entramos en el modo que queremos, podremos elegir qué tipo de vuelo realizará el modo. Esto lo haremos abriendo el menú desplegable que hay al lado de cada rango de señal de cada modo. De esta forma, los modos quedarán:

Modo	Rango de señal	Tipo de vuelo
1	0-1230	AltHold o Stabilize
2	1231-1360	PosHold
3	1361-1490	Guided
6	1750 +	RTL

AltHold o Stabilize son muy similares, así que podemos utilizarlos como defecto. AltHold se encarga de mantener la altura y la estabilidad mientras que Stabilize de la estabilidad tan solo, pero como los cambios de altura en un multicóptero en suspensión no son bruscos, podemos utilizar ambos.

PosHold mantendrá la posición de vuelo, y en caso de haber viento, el autopiloto corregirá su posición constantemente para mantenerse en el punto de suspensión donde se encontraba.

Estos dos modos anteriores corrigen ciertos parámetros, pero todo desplazamiento (que no sea correctivo) está en manos del usuario. Sin embargo, en **Guided**, el UA se desplazará a partir de las órdenes enviadas por QGC, a fin de seguir una ruta preestablecida.

RTL es el anteriormente llamado "modo de emergencia", el cual será usado para que el UA vuelva al punto de despegue automáticamente en caso de emergencia.

Apéndice 6: Código vehicle_state.py original

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
© Copyright 2015-2016, 3D Robotics.
vehicle_state.py:
Demonstrates how to get and set vehicle state and parameter
information,
and how to observe vehicle attribute (state) changes.
Full documentation is provided at
http://python.dronekit.io/examples/vehicle_state.html
"""
from dronekit import connect, VehicleMode
import time
#Set up option parsing to get connection string
import argparse
parser = argparse.ArgumentParser(description='Print out vehicle
state information. Connects to SITL on local PC by default.')
parser.add_argument('--connect',
                    help="vehicle connection target string. If
not specified, SITL automatically started and used.")
args = parser.parse_args()
connection_string = args.connect
sitl = None

#Start SITL if no connection string specified
if not connection_string:
    import dronekit_sitl
    sitl = dronekit_sitl.start_default()
    connection_string = sitl.connection_string()

# Connect to the Vehicle.
#   Set `wait_ready=True` to ensure default attributes are
populated before `connect()` returns.
print "\nConnecting to vehicle on: %s" % connection_string
vehicle = connect(connection_string, wait_ready=True)
vehicle.wait_ready('autopilot_version')
# Get all vehicle attributes (state)
print "\nGet all vehicle attribute values:"
print " Autopilot Firmware version: %s" % vehicle.version
print "   Major version number: %s" % vehicle.version.major
print "   Minor version number: %s" % vehicle.version.minor
print "   Patch version number: %s" % vehicle.version.patch
print "   Release type: %s" % vehicle.version.release_type()
print "   Release version: %s" %
vehicle.version.release_version()
print "   Stable release?: %s" % vehicle.version.is_stable()
```

```
print " Autopilot capabilities"
print "   Supports MISSION_FLOAT message type: %s" %
vehicle.capabilities.mission_float
print "   Supports PARAM_FLOAT message type: %s" %
vehicle.capabilities.param_float
print "   Supports MISSION_INT message type: %s" %
vehicle.capabilities.mission_int
print "   Supports COMMAND_INT message type: %s" %
vehicle.capabilities.command_int
print "   Supports PARAM_UNION message type: %s" %
vehicle.capabilities.param_union
print "   Supports ftp for file transfers: %s" %
vehicle.capabilities.ftp
print "   Supports commanding attitude offboard: %s" %
vehicle.capabilities.set_attitude_target
print "   Supports commanding position and velocity targets in
local NED frame: %s" %
vehicle.capabilities.set_attitude_target_local_ned
print "   Supports set position + velocity targets in global
scaled integers: %s" %
vehicle.capabilities.set_altitude_target_global_int
print "   Supports terrain protocol / data handling: %s" %
vehicle.capabilities.terrain
print "   Supports direct actuator control: %s" %
vehicle.capabilities.set_actuator_target
print "   Supports the flight termination command: %s" %
vehicle.capabilities.flight_termination
print "   Supports mission_float message type: %s" %
vehicle.capabilities.mission_float
print "   Supports onboard compass calibration: %s" %
vehicle.capabilities.compass_calibration
print " Global Location: %s" % vehicle.location.global_frame
print " Global Location (relative altitude): %s" %
vehicle.location.global_relative_frame
print " Local Location: %s" % vehicle.location.local_frame
print " Attitude: %s" % vehicle.attitude
print " Velocity: %s" % vehicle.velocity
print " GPS: %s" % vehicle.gps_0
print " Gimbal status: %s" % vehicle.gimbal
print " Battery: %s" % vehicle.battery
print " EKF OK?: %s" % vehicle.ekf_ok
print " Last Heartbeat: %s" % vehicle.last_heartbeat
print " Rangefinder: %s" % vehicle.rangefinder
print " Rangefinder distance: %s" % vehicle.rangefinder.distance
print " Rangefinder voltage: %s" % vehicle.rangefinder.voltage
print " Heading: %s" % vehicle.heading
print " Is Armable?: %s" % vehicle.is_armable
print " System status: %s" % vehicle.system_status.state
print " Groundspeed: %s" % vehicle.groundspeed    # settable
print " Airspeed: %s" % vehicle.airspeed    # settable
print " Mode: %s" % vehicle.mode.name    # settable
```

```
print " Armed: %s" % vehicle.armed    # settable

# Get Vehicle Home location - will be `None` until first set by
autopilot
while not vehicle.home_location:
    cmds = vehicle.commands
    cmds.download()
    cmds.wait_ready()
    if not vehicle.home_location:
        print " Waiting for home location ..."
# We have a home location, so print it!
print "\n Home location: %s" % vehicle.home_location
# Set vehicle home_location, mode, and armed attributes (the
only settable attributes)
print "\nSet new home location"
# Home location must be within 50km of EKF home location (or
setting will fail silently)
# In this case, just set value to current location with an
easily recognisable altitude (222)
my_location_alt = vehicle.location.global_frame
my_location_alt.alt = 222.0
vehicle.home_location = my_location_alt
print " New Home Location (from attribute - altitude should be
222): %s" % vehicle.home_location
#Confirm current value on vehicle by re-downloading commands
cmds = vehicle.commands
cmds.download()
cmds.wait_ready()
print " New Home Location (from vehicle - altitude should be
222): %s" % vehicle.home_location
print "\nSet Vehicle.mode = GUIDED (currently: %s)" %
vehicle.mode.name
vehicle.mode = VehicleMode("GUIDED")
while not vehicle.mode.name=='GUIDED':  #Wait until mode has
changed
    print " Waiting for mode change ..."
    time.sleep(1)
# Check that vehicle is armable
while not vehicle.is_armable:
    print " Waiting for vehicle to initialise..."
    time.sleep(1)
    # If required, you can provide additional information about
initialisation
    # using `vehicle.gps_0.fix_type` and `vehicle.mode.name`.
print "\nSet Vehicle.armed=True (currently: %s)" % vehicle.armed
vehicle.armed = True
while not vehicle.armed:
    print " Waiting for arming..."
    time.sleep(1)
print " Vehicle is armed: %s" % vehicle.armed
```

```
# Add and remove and attribute callbacks
#Define callback for `vehicle.attitude` observer
last_attitude_cache = None
def attitude_callback(self, attr_name, value):
    # `attr_name` - the observed attribute (used if callback is
used for multiple attributes)
    # `self` - the associated vehicle object (used if a callback
is different for multiple vehicles)
    # `value` is the updated attribute value.
    global last_attitude_cache
    # Only publish when value changes
    if value!=last_attitude_cache:
        print " CALLBACK: Attitude changed to", value
        last_attitude_cache=value
print "\nAdd `attitude` attribute callback/observer on
`vehicle`"
vehicle.add_attribute_listener('attitude', attitude_callback)
print " Wait 2s so callback invoked before observer removed"
time.sleep(2)
print " Remove Vehicle.attitude observer"
# Remove observer added with `add_attribute_listener()`
specifying the attribute and callback function
vehicle.remove_attribute_listener('attitude', attitude_callback)

# Add mode attribute callback using decorator (callbacks added
this way cannot be removed).
print "\nAdd `mode` attribute callback/observer using decorator"
@vehicle.on_attribute('mode')
def decorated_mode_callback(self, attr_name, value):
    # `attr_name` is the observed attribute (used if callback is
used for multiple attributes)
    # `attr_name` - the observed attribute (used if callback is
used for multiple attributes)
    # `value` is the updated attribute value.
    print " CALLBACK: Mode changed to", value
print " Set mode=STABILIZE (currently: %s) and wait for
callback" % vehicle.mode.name
vehicle.mode = VehicleMode("STABILIZE")
print " Wait 2s so callback invoked before moving to next
example"
time.sleep(2)
print "\n Attempt to remove observer added with `on_attribute`
decorator (should fail)"
try:
    vehicle.remove_attribute_listener('mode',
decorated_mode_callback)
except:
    print " Exception: Cannot remove observer added using
decorator"
```

```
# Demonstrate getting callback on any attribute change
def wildcard_callback(self, attr_name, value):
    print " CALLBACK: (%s): %s" % (attr_name,value)
print "\nAdd attribute callback detecting ANY attribute change"
vehicle.add_attribute_listener('*', wildcard_callback)
print " Wait 1s so callback invoked before observer removed"
time.sleep(1)

print " Remove Vehicle attribute observer"
# Remove observer added with `add_attribute_listener()`
vehicle.remove_attribute_listener('*', wildcard_callback)
# Get/Set Vehicle Parameters
print "\nRead and write parameters"
print " Read vehicle param 'THR_MIN': %s" %
vehicle.parameters['THR_MIN']
print " Write vehicle param 'THR_MIN' : 10"
vehicle.parameters['THR_MIN']=10
print " Read new value of param 'THR_MIN': %s" %
vehicle.parameters['THR_MIN']
print "\nPrint all parameters (iterate `vehicle.parameters`):"
for key, value in vehicle.parameters.iteritems():
    print " Key:%s Value:%s" % (key,value)
print "\nCreate parameter observer using decorator"
# Parameter string is case-insensitive
# Value is cached (listeners are only updated on change)
# Observer added using decorator can't be removed.
@vehicle.parameters.on_attribute('THR_MIN')
def decorated_thr_min_callback(self, attr_name, value):
    print " PARAMETER CALLBACK: %s changed to: %s" % (attr_name,
value)
print "Write vehicle param 'THR_MIN' : 20 (and wait for
callback)"
vehicle.parameters['THR_MIN']=20
for x in range(1,5):
    #Callbacks may not be updated for a few seconds
    if vehicle.parameters['THR_MIN']==20:
        break
    time.sleep(1)
#Callback function for "any" parameter
print "\nCreate (removable) observer for any parameter using
wildcard string"
def any_parameter_callback(self, attr_name, value):
    print " ANY PARAMETER CALLBACK: %s changed to: %s" %
(attr_name, value)
#Add observer for the vehicle's any/all parameters parameter
(defined using wildcard string ``'*'``)
vehicle.parameters.add_attribute_listener('*',
any_parameter_callback)
print " Change THR_MID and THR_MIN parameters (and wait for
callback)"
vehicle.parameters['THR_MID']=400
```

```
vehicle.parameters['THR_MIN']=30
## Reset variables to sensible values.
print "\nReset vehicle attributes/parameters and exit"
vehicle.mode = VehicleMode("STABILIZE")
vehicle.armed = False
vehicle.parameters['THR_MIN']=130
vehicle.parameters['THR_MID']=500
#Close vehicle object before exiting script
print "\nClose vehicle object"
vehicle.close()
# Shut down simulator if it was started.
if sitl is not None:
    sitl.stop()
print("Completed")
```

Apéndice 7: Código follow_me.py original

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
© Copyright 2015-2016, 3D Robotics.
followme - Tracks GPS position of your computer (Linux only).
This example uses the python gps package to read positions from
a GPS attached to your
laptop and sends a new vehicle.simple_goto command every two
seconds to move the
vehicle to the current point.
When you want to stop follow-me, either change vehicle modes or
type Ctrl+C to exit the script.
Example documentation:
http://python.dronekit.io/examples/follow_me.html
"""
from dronekit import connect, VehicleMode,
LocationGlobalRelative
import gps
import socket
import time
import sys
#Set up option parsing to get connection string
import argparse
parser = argparse.ArgumentParser(description='Tracks GPS
position of your computer (Linux only).')
parser.add_argument('--connect',
                    help="vehicle connection target string. If
not specified, SITL automatically started and used.")
args = parser.parse_args()
connection_string = args.connect
sitl = None
#Start SITL if no connection string specified
if not connection_string:
    import dronekit_sitl
    sitl = dronekit_sitl.start_default()
    connection_string = sitl.connection_string()

# Connect to the Vehicle
print 'Connecting to vehicle on: %s' % connection_string
vehicle = connect(connection_string, wait_ready=True)
def arm_and_takeoff(aTargetAltitude):
    """
    Arms vehicle and fly to aTargetAltitude.
    """

    print "Basic pre-arm checks"
    # Don't let the user try to arm until autopilot is ready
    while not vehicle.is_armable:
```

```
        print " Waiting for vehicle to initialise..."
        time.sleep(1)

    print "Arming motors"
    # Copter should arm in GUIDED mode
    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
    # Wait until the vehicle reaches a safe height before
processing the goto (otherwise the command
    #  after Vehicle.simple_takeoff will execute immediately).
    while True:
        print " Altitude: ",
vehicle.location.global_relative_frame.alt
        if
vehicle.location.global_relative_frame.alt>=aTargetAltitude*0.95
: #Trigger just below target alt.
            print "Reached target altitude"
            break
        time.sleep(1)
try:
    # Use the python gps package to access the laptop GPS
    gpsd = gps.gps(mode=gps.WATCH_ENABLE)

    #Arm and take off to altitude of 5 meters
    arm_and_takeoff(5)
    while True:
            if vehicle.mode.name != "GUIDED":
            print "User has changed flight modes - aborting
follow-me"
            break

        # Read the GPS state from the laptop
        gpsd.next()
        # Once we have a valid location (see gpsd documentation)
we can start moving our vehicle around
        if (gpsd.valid & gps.LATLON_SET) != 0:
            altitude = 30  # in meters
            dest = LocationGlobalRelative(gpsd.fix.latitude,
gpsd.fix.longitude, altitude)
            print "Going to: %s" % dest
            # A better implementation would only send new
waypoints if the position had changed significantly
            vehicle.simple_goto(dest)
```

```
            # Send a new target every two seconds
            # For a complete implementation of follow me you'd
want adjust this delay
            time.sleep(2)

except socket.error:
    print "Error: gpsd service does not seem to be running, plug
in USB GPS or run run-fake-gps.sh"
    sys.exit(1)
#Close vehicle object before exiting script
print "Close vehicle object"
vehicle.close()
# Shut down simulator if it was started.
if sitl is not None:
    sitl.stop()

print("Completed")
```

Apéndice 8: Código opencv_stream.py

```
from imutils.video import VideoStream
import imutils
import cv2
import numpy as np
import threading
import time

def nothing(x):
    pass
resolution=(112,80)
fps = 40
printedShape=False
#-----------------------------------------------------------------
--------
vs = VideoStream(src=0, resolution=resolution,
framerate=fps).start()
time.sleep(1.0)
frame = vs.read()
#frame = cv2.resize(frame,None,fx=scale, fy=scale, interpolation
= cv2.INTER_AREA)
rows,cols=frame.shape[:2]
# Thresholds
minH = 0
maxH = 255
minS = 0
maxS = 255
minV = 0
maxV = 255
# Create a window
cv2.namedWindow('binalized');
cv2.createTrackbar('H max', 'binalized', 255, 255, nothing);
cv2.createTrackbar('H min', 'binalized', 0, 255, nothing);
cv2.createTrackbar('S max', 'binalized', 255, 255, nothing);
cv2.createTrackbar('S min', 'binalized', 0, 255, nothing);
cv2.createTrackbar('V max', 'binalized', 255, 255, nothing);
cv2.createTrackbar('V min', 'binalized', 0, 255, nothing);

while True:
    image = vs.read()
    if not printedShape:
        print(threading.active_count())
        print(frame.shape)
        printedShape=True

    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

```
    maxH = cv2.getTrackbarPos('H max', 'binalized')
    minH = cv2.getTrackbarPos('H min', 'binalized')
    maxS = cv2.getTrackbarPos('S max', 'binalized')
    minS = cv2.getTrackbarPos('S min', 'binalized')
    maxV = cv2.getTrackbarPos('V max', 'binalized')
    minV = cv2.getTrackbarPos('V min', 'binalized')

    lower_color = np.array([minH, minS, minV])
    upper_color = np.array([maxH, maxS, maxV])
    # Threshold the HSV image to get only the selected color
    mask = cv2.inRange(hsv, lower_color, upper_color)
    cv2.imshow('binalized',mask)
        # Bitwise-AND mask and original image
    #res = cv2.bitwise_and(image,image, mask= mask)
    #
    # find contours in the mask and initialize the current
    # (x, y) center of the ball
    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[-2]
    center = None
    # only proceed if at least one contour was found
    if len(cnts) > 0:
            # find the largest contour in the mask, then use
            # it to compute the minimum enclosing circle and
            # centroid
            c = max(cnts, key=cv2.contourArea)
            ((x, y), radius) = cv2.minEnclosingCircle(c)
            M = cv2.moments(c)
            #print ("contornos : ",len(cnts),"area: ",M["m00"])
            if M["m00"] != 0:
                    center = (int(M["m10"] / M["m00"]),
int(M["m01"] / M["m00"]))

            # only proceed if the radius meets a minimum size
            if radius > 10:
                    # draw the circle and centroid on the frame,
                    # then update the list of tracked points
                    cv2.circle(image, (int(x), int(y)),
int(radius), (0, 255, 255), 2)
                    cv2.circle(image, center, 5, (0, 0, 255), -
1)
    # show the frame
    cv2.imshow("Frame", image)

    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

cv2.destroyAllWindows()
vs.stop()
time.sleep(1.0)
```

Bibliografía

Términos: https://www.masscience.com/2015/08/11/2051/ Y OACI

MAVLINK https://en.wikipedia.org/wiki/MAVLink

OPENCV https://es.wikipedia.org/wiki/OpenCV https://opencv.org/

Proyecto tutorizado por Ángel Rodas y realizado por Carles Pérez Cerdán: Diseño de un sistema de monitorización de drones mediante Raspberry Pi

Configuración https://docs.emlid.com/navio2/common/ardupilot/configuring-raspberry-pi/

Historia UAS: http://eldrone.es/historia-de-los-drones/

Tipos de UAS:

http://www.xdrones.es/tipos-de-drones-clasificacion-de-drones-categorias-de-drones/

http://aeromedia.es/2016/02/11/rpas-de-ala-fija/

http://www.todrone.com/tag/ala-fija/

Imágenes

https://huescadrones.es/hddrones/aerodinos-de-ala-rotatoria-tipo-helicoptero/

http://www.aerial-insights.co/blog/tipos-de-drones/

https://hipertextual.com/2013/08/drone-solar

: http://adtsgroup.com/octocopter

https://www.infodefensa.com/wp-content/uploads/Af-Uavs-10-03.pdf

https://www.researchgate.net/figure/Final-Monocopter-Design_fig1_237758639

http://baltimorehackerspace.com/wp-content/uploads/2013/02/

https://therobotsource.com/diy-frame-kitsfor-building-drones-quadcopters-multicopters-multirotors-uavs/694-diy-trooper-y6-750-tricopter-frame-kit.html

http://www.unmannedsystemstechnology.com/company/alta-devices/solar-puma-uas/.

https://tienda.stockrc.com/epages/eb0140.sf/es_ES/?ObjectPath=/Shops/eb0140/Categories/Todo_MultiRotor1/Baterias_Lipo/BATERIAS

http://www.mcielectronics.cl/en_US/shop/product/kit-para-construccion-de-cuadricoptero-10917

https://www.bhphotovideo.com/c/product/1311480-REG/yuneec_yuntyhscus_typhoon_h_rtf_in.html

http://www.skypirate.us/shop/drones/custom-drones/sky-pirate-t18-rtf/

https://www.rcocio.com/kit-dji-f550-hexacopter-dji-naza-v2-GNSS-patin

http://clymenadrones.com/blog/index.php/2017/03/13/dron-a-gasolina-controlado-electronicamente/

https://diydrones.com/profiles/blogs/solar-drone-experiments-how-much-more-battery-life-do-you-get-by

https://www.elperiodico.com/es/tecnologia/20150306/drones-la-ultima-revolucion-militar-3995954

http://www.leonoticias.com/leon/201605/05/drones-militares-tendran-leon-20160505141749.html

https://www.ndtv.com/india-news/civil-aviation-ministry-proposes-rules-for-commercial-use-of-drones-1769994

http://www.expansion.com/fueradeserie/tecno/2017/06/13/593e724d46163fe6108b45cb.html

cámara: https://www.banggood.com/es/600TVL-2_8mm-Lens-13-Sony-Super-Had-II-CCD-Camera-for-FPV-Racing-Drone-PALNTSC-p-1086784.html?stayold=1&cur_warehouse=CN

baudios Wikipedia y https://www.schneider-electric.es/es/faqs/FA29554/

Configuración DroneKit: https://community.emlid.com/t/navio-2-dronekit/8317/8

https://stackoverflow.com/questions/46210013/dronekit-python-vehicle-connection-timeout

DESCARGAS

http://python.dronekit.io/examples/vehicle_state.html

http://python.dronekit.io/examples/follow_me.html

https://docs.emlid.com/navio2/common/ardupilot/configuring-raspberry-pi/

https://www.putty.org/

http://python.dronekit.io/develop/installation.html

https://etcher.io/