UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

PROYECTO FIN DE CARRERA

Sistema de enlace robusto para la teleoperación de un UAV (vehículo aéreo no tripulado) en la plataforma robótica ARGOS

Guadalupe Crespo Quirós

Enero 2015

Sistema de enlace robusto para la teleoperación de un UAV (vehículo aéreo no tripulado) en la plataforma robótica ARGOS

AUTOR: Guadalupe Crespo Quirós
TUTOR: Guillermo González de Rivera Peces

HCTLab
Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Enero de 2015

Resumen

Este proyecto consiste en la creación de un sistema de teleoperación para un UAV de tipo cuadricóptero.

Para ello se diseñan los enlaces inalámbricos para la transmisión de señal de vídeo y de datos, para lo cual se ha estudiado la amplia oferta de módulos disponibles en el mercado y se ha seleccionado el conjunto adecuado para la fase de desarrollo inicial del proyecto.

El sistema de transmisión de vídeo consiste en dos enlaces separados capaces de trabajar a la vez gracias al uso de antenas con polarizaciones ortogonales en cada uno de los enlaces. De esta forma, el UAV podrá ir equipado con dos cámaras de vídeo y transmitir ambas señales de forma simultánea.

El enlace de datos se implementa utilizando dos módulos XBee, que se configuran para establecer una comunicación bidireccional entre el UAV y la estación base. Para conseguir un control fiable se ha utiliza el protocolo de comunicación MAVLink.

Se ha seleccionado un controlador de vuelo (autopilot) comercial, llamado Pixhawk, que utiliza este mismo sistema para transmisión de la telemetría a la estación base el cual se adapta para conseguir que también interprete las órdenes de control utilizando el mismo protocolo. Se realiza un estudio de su arquitectura software y se describen sus módulos principales de manera que se hace posible la intervención dentro del código fuente para agregar/modificar funcionalidades según lo requiera el proyecto. Se implementa un conjunto de modificaciones que permite sustituir el control remoto de la aeronave basado en un mando RC comercial estándar por un mando digital conectado al ordenador en la estación base, abriendo la posibilidad de comunicación de comandos de vuelo generados automáticamente en la estación base. Además se crea una aplicación para enviar las acciones del operario sobre el control de vuelo desde la estación base.

Palabras clave

UAV, cuadricóptero, XBee, Pixhawk, telemetría, controlador de vuelo.

Abstract

This project involves the creation of a teleoperation system of quadcopter type UAV.

Two wireless links are designed for transmitting video signal and data, it is studied a wide range of modules available in the market and it is selected the right set for the initial development phase of the project.

The video transmission system consists of two different links which can work together avoiding interferences by using orthogonal polarization each other. Thus, the UAV can be equipped with two video cameras and transmit both signals simultaneously.

The data link is implemented by using two XBee modules configured to provide bidirectional communication between the UAV and the base station. MAVLink communication protocol is used to achieve reliable control link.

It is selected a commercial autopilot, called Pixhawk, which use MAVLink communication protocol to transmit telemetry to the base station. It is adapted to ensure that the autopilot also interprets control commands using the same protocol. A study of its software architecture is performed and its main modules are described so that intervention is possible within the source code to add or modify functionality as required by the project. It is implemented a set of modifications that allows replacing the aircraft remote control based on a standard commercial RC remote controller by a digital joystick connected to the base station computer. This makes possible sending control commands automatically generated in the base station. Furthermore, a control application is created to send control commands from the base station.

Keywords

UAV, quadcopter, XBee, Pixhawk, telemetry, autopilot.

Agradecimientos

En primer lugar, quiero agradecer este proyecto a mi familia, mis padres, Sandra y Antonio, mi herma Jimena y mis abuelos. Todos ellos han estado a mi lado y me han apoyado siempre, durante el desarrollo del PFC, durante la carrera y durante toda mi vida. Ellos han hecho posible que haya llegado hasta aquí.

También gracias a Jose, mi pareja, por estar a mi lado, aguantarme en los malos momentos y disfrutar conmigo de los buenos. Gracias por los abrazos.

Quiero agradecer a mi tutor Guillermo la posibilidad de realizar este proyecto. Y en general a todo el HCTLab por la acogida.

Y por supuesto, quiero dar las gracias a todo el equipo de ROBOMOTION y mis compañeros de PFC, por toda la ayuda y por todos los buenos momentos que hemos pasado durante este año. En especial quiero agradecer a Roberto, mi tutor de ROBOMOTION, por su dedicación y ayuda, y por solucionar todos los imprevistos que han surgido.

¡Mil gracias a todos!

Guadalupe.

Índice de contenidos

Índice de figuras

Índice de tablas

1 Introducción

1.1. Motivación

El grupo HCTLab, en colaboración con la empresa de I+D Robomotion [1], participa en un proyecto de desarrollo de una multiplataforma robótica de gran alcance y alta autonomía, ARGOS. El sistema completo estará compuesto por varias plataformas robóticas capaces de operar en distintos medios (terrestre y aéreo). Se pretende conseguir un sistema capaz de intervenir en misiones con un elevado nivel de riesgo humano, tales como catástrofes naturales, en las que las zonas de operación sean inaccesibles o muy peligrosas para las personas.

La plataforma aérea estará constituida por un vehículo aéreo no tripulado (UAV), de tipo cuadricóptero. Como objetivo se pretende que el UAV lleve a cabo tareas de inspección y supervisión de una zona peligrosa, monitorización aérea de robots terrestres, e incluso transporte de pequeños objetos, por ejemplo repetidores de RF para mejorar la cobertura de los robots terrestres. El sistema deberá cumplir, por lo tanto, una serie de características que le permitan desarrollar dichas tareas de la forma más eficiente posible.

Por un lado deberá ser capaz de realizar vuelo estacionario, deberá tener suficiente agilidad de vuelo como para poder superar los obstáculos que pueda encontrar, deberá ser capaz de transportar la carga necesaria y a la vez tener suficiente autonomía de vuelo como para poder realizar la misión al completo.

Por otra parte, deberá tener un sistema de control remoto robusto pero cuya operación sea sencilla e intuitiva, deberá ser capaz de enviar toda la información de telemetría a la estación base, así como incluir un enlace de vídeo que permita obtener una señal de calidad en tiempo real. Es decir, deberá incorporar un sistema de teleoperación, telemetría y vídeo de largo alcance, todo ello controlado desde un PC en la estación base.

Se plantea la posibilidad de utilizar un único enlace para la transmisión de datos de control y la recepción tanto de telemetría como de vídeo y audio. Otra posibilidad consiste en utilizar dos enlaces, uno para datos (teleoperación y telemetría) y otro específico para la señal de vídeo y audio. En cualquier caso se empleará el protocolo de comunicación MAVLink, que es un protocolo de enlace para micro vehículos aéreos que, basándose en instrucciones sencillas, permite enviar todo tipo de órdenes al UAV así como recibir toda la información necesaria de forma que sea fácil de interpretar.

1.2. Objetivos

Con este proyecto se pretende construir un enlace robusto de vídeo, telemetría y control para un vehículo aéreo no tripulado.

El principal objetivo del proyecto es la selección y validación de la tecnología y componentes que se van a utilizar. Se realizará una valoración previa de las ventajas e inconvenientes de las distintas soluciones posibles. Como segundo objetivo, también primordial y consecuencia del anterior, es el desarrollo de las interfaces necesarias para la integración de esta selección de componentes en el UAV, así como con el resto del sistema ARGOS.

Como objetivo secundario se debe conseguir el equilibrio entre las restricciones de diseño impuestas al proyecto que permitan:

- **Maximizar alcance.** Se pretende que el UAV pueda ser controlado a distancias superiores a un kilómetro.
- **Minimizar el peso de los componentes.** Dado que los cuadricópteros son vehículos ligeros, y que deberá soportar una carga adicional (payload), los componentes utilizados deberán ser pequeños y ligeros.
- **Minimizar el consumo de potencia.** Otra de las limitaciones de los UAVs es la autonomía. Las baterías deben tener la mayor capacidad posible, teniendo en cuenta tanto su volumen como su peso. Para ayudar a aumentar la autonomía lo mejor es elegir componentes con un consumo de potencia mínimo.

Toda la tecnología empleada en el proyecto (equipos Rx/Tx, antenas, etc.) debe estar compuesta por módulos comerciales, para así reducir el tiempo de desarrollo necesario en los diseños de módulos específicos.

1.3. Organización de la memoria

La memoria tiene la siguiente estructura:

1. **Introducción**

Descripción de las motivaciones y principales objetivos de este proyecto.

2. **Estado del arte**

Introducción de los tipos de UAVs más comunes y descripción de las principales características de los controladores vuelo de los cuadricópteros.

3. **Diseño**

Justificación de la selección de los componentes utilizados y descripción de sus características técnicas.

4. **Desarrollo**

Implementación de las modificaciones necesarias para adaptar los componentes seleccionados a los requisitos del proyecto.

5. **Pruebas y resultados**

Descripción de las pruebas realizadas y los resultados obtenidos.

6. **Conclusiones y trabajo futuro**

Resumen del sistema conseguido y posibles ampliaciones futuras.

2 Estado del arte

2.1. Unmanned Aerial Vehicles

Los UAV (Unmanned Aerial Vehicles) o Drones son, como su nombre indica, pequeños vehículos aéreos no tripulados, es decir, no llevan ningún piloto embarcado en su interior. Esto no implica que sean sistemas completamente autónomos, sino que el control se hace de forma remota, aunque comúnmente cuentan con opciones de vuelo autónomo.

Habitualmente se utilizan para realizar tareas en zonas de difícil acceso para las personas, donde es necesario un vehículo de pequeñas dimensiones para tener mayor movilidad o donde el acceso por tierra sea complicado o inseguro [2]. Debido a su reducido tamaño y su gran movilidad los UAVs se están convirtiendo en elementos clave en tareas de inspección, vigilancia o rescate [3] [4] [5]. A día de hoy, el tipo de UAV más común para aplicaciones no militares es el multicóptero, y en particular el cuadricóptero [6] [7], ya que es capaz de transportar la carga adecuada y cuenta con la suficiente resistencia como para realizar este tipo de tareas [8].

Una clasificación general de los UAVs los divide en dos grandes grupos:

- **Vehículos de ala fija**: En este grupo se encuentran todo tipo de aviones.
- **Vehículos de ala rotativa**: En este grupo se encuentran tanto los helicópteros como todos los tipos de multicópteros.

La principal ventaja de los vehículos de ala rotativa es la capacidad para permanecer suspendidos en el aire en una posición fija. Otra ventaja es el sistema de despegue y aterrizaje, ya que en todos los vehículos de ala rotativa se realiza en vertical, lo que les permite realizarlo en una mayor variedad de condiciones, aunque también existen vehículos de ala fija que implementan sistemas VTOL (Vertical Take Off and Landing) mediante el uso de rotores específicos.

Dentro de los vehículos de ala rotatoria, los multicópteros destacan sobre los helicópteros por poseer una mayor capacidad de carga con la misma envergadura. Además, los controladores de vuelo suelen ser más fáciles de implementar en cuadricópteros, lo que hace que sea más sencillo controlar el vuelo [9]. En cualquier caso, con alas rotativas es más sencillo implementar sistemas de despegue y aterrizaje automático.

Para el control de estabilidad y de vuelo es conveniente un buen sistema de control, navegación y comunicación. Los UAVs necesitan de una unidad de medición inercial, IMU, que con la ayuda de un GPS y opcionalmente otros sensores, proporciona las

medidas necesarias para estimar y luego controlar la orientación, posición y velocidad del UAV.

Es común controlar los UAVs, a día de hoy, empleando sistemas de control automático electrónicos denominados *autopilots*. Los autopilots son el conjunto de una serie de sensores y un controlador de vuelo de tamaño reducido y poco peso.

En los siguientes puntos dentro de este apartado se explicarán algunos conceptos básicos acerca de los UAVs como lo son el sistema de representación del movimiento (2.1.1), los distintos modos de vuelo que se suelen implementar (2.1.2) y los distintos sensores que se integran tanto en la IMU (2.1.3) como otros sensores complementarios (2.1.4) y el GPS (2.1.5).

En apartados posteriores dentro de este estudio del estado del arte se explicará el funcionamiento básico del sistema de control de vuelo de un cuadricóptero, presentando una introducción a los controladores de posición y estabilidad de los cuadricópteros (2.2). Se presentará también una introducción a los sistemas de comunicación típicos empleados en UAVs (2.3), tema desarrollado en este proyecto; y por último, se presentará el sistema de navegación (2.4), donde se explica el seguimiento de los waypoints que se envían desde la estación base.

2.1.1. *Dinámica del cuadricóptero*

El movimiento de los cuadricópteros se controla mediante variaciones en la velocidad de giro de 4 motores eléctricos, generalmente sin escobillas (brushless).

La velocidad angular de giro de un motor define la fuerza de empuje vertical del mismo. Por lo tanto, para que un cuadricóptero ascienda o descienda se debe aumentar o disminuir la velocidad de rotación de los 4 motores simultáneamente.

Combinando diferentes velocidades de giro en cada uno de los motores se logra controlar los movimientos de rotación y traslación del cuadricóptero. Esta velocidad de giro determina el par de rotación reactiva de la aeronave con respecto al rotor (par de rotación de la aeronave en sentido contrario al giro del rotor); combinando el par reactivo de los 4 rotores es posible hacer rotar o detener la rotación de la aeronave sobre su eje vertical. La forma en que los motores deben girar depende de la configuración de la estructura. En la Figura 2-1 y Figura 2-2 se pueden ver las configuraciones más comunes:

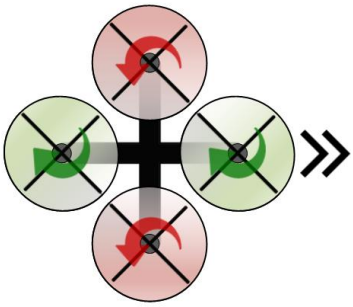


Figura 2-1. Configuración de cuadricóptero en +

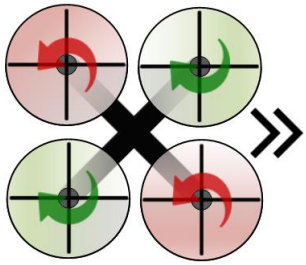


Figura 2-2. Configuración de cuadricóptero en X

La configuración en + cuenta con un par de motores alineados con el eje longitudinal del cuadricóptero, eje x, mientras que el otro par se encuentra alineado con el eje de traslación, eje y. La configuración en X cuenta con dos motores delanteros y dos motores traseros.

La más sencilla de implementar es la configuración en +, pero a día de hoy la más utilizada es la configuración en X. En un cuadricóptero configurado en + se varía la fuerza de un motor para realizar cada uno de los desplazamientos (delante, detrás, izquierda y derecha), mientras que un cuadricóptero configurado en X varía la fuerza de dos motores para cada desplazamiento. Por este motivo se considera que la configuración en X funciona mejor cuando se realizan desplazamientos ya que esta configuración es más robusta ante perturbaciones externas [10].

El cuadricóptero que se ha desarrollado para este proyecto cuenta con una estructura en X.

En la Figura 2-3 se puede ver un dibujo esquematizado del cuadricóptero, donde la parte frontal corresponde con la derecha de la imagen, es decir, el cuadricóptero avanza en el sentido del eje x. Además se pueden ver representados los movimientos de rotación sobre los 3 ejes de coordenadas que se utilizan para controlar el cuadricóptero:

- **Roll:** movimiento de rotación sobre el eje X. Su variación permite desplazar el cuadricóptero hacia la derecha o la izquierda.
- **Pitch:** movimiento de rotación sobre el eje Y. Su variación permite desplazar el cuadricóptero hacia delante o detrás.
- **Yaw:** movimiento de rotación sobre el eje Z. Su variación permite rotar el cuadricóptero hacia la derecha o la izquierda.

Estos tres movimientos de rotación se expresan como ángulos, generalmente en radianes.

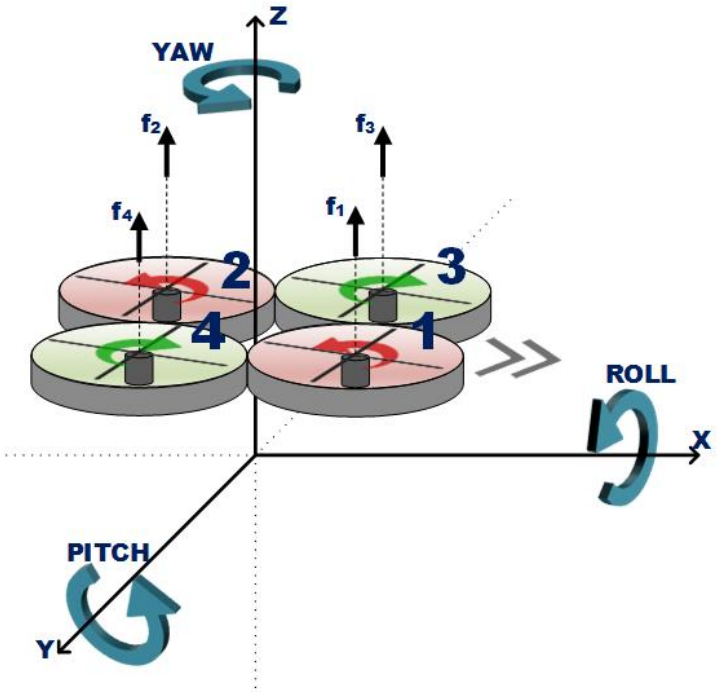


Figura 2-3. Sistema de representación del movimiento de un cuadricóptero

Se ha representado también el sentido de giro de cada uno de los motores: en verde el giro en el sentido de las agujas del reloj y el rojo el giro en sentido opuesto. Como se puede ver, motores contiguos giran en sentido opuesto, gracias a esto se consigue evitar que el cuadricóptero rote en el sentido de los motores.

A continuación se representan los principales movimientos del cuadricóptero, Figura 2-4, donde se puede ver los motores que giran con mayor velocidad representados con flechas mayores. Y cómo, al aumentar la velocidad, aumenta la fuerza de empuje vertical en determinados motores y se consiguen los movimientos deseados:

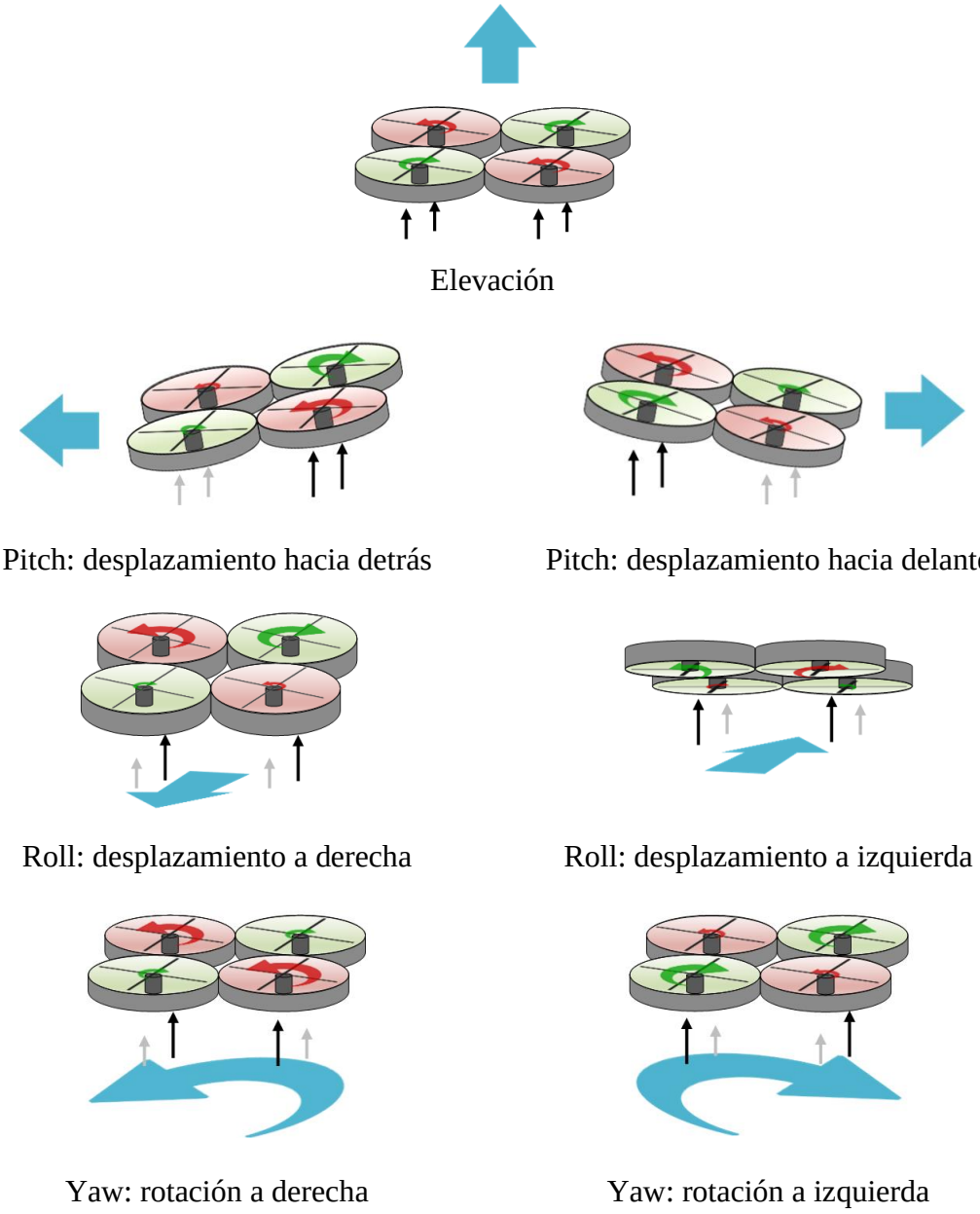


Figura 2-4. Movimientos de los cuadricópteros

Existen diversos modelos matemáticos que se utilizan para obtener un modelo simplificado de la dinámica de los cuadricópteros. Estos modelos matemáticos permiten diseñar las reglas de control dedicadas a la estabilización de la orientación y al posicionamiento autónomo. Los modelos matemáticos que más se utilizan se basan en las aproximaciones de Euler-Lagrange y de Newton-Euler, donde se considera el cuadricóptero como un objeto sólido que se mueve en un entorno de 3 dimensiones, sometido a una fuerza principal, throttle, y a tres pares de fuerza: roll, pitch y yaw [11] [12].

2.1.2. *Modos de vuelo*

En general, los UAVs implementan distintos modos de vuelo que se clasifican según el nivel de autonomía. Los más típicos son:

- **Manual**: Es un tipo de control en lazo abierto, donde el autopilot calcula los valores de salida para los motores a partir de los controles de roll, pitch, yaw y throttle que recibe del usuario. Este tipo de control no se suele utilizar en multicópteros.
- **Estabilizado**: El control se hace en forma de lazo cerrado. El controlador de orientación calcula los valores de salida para los motores a partir de los controles de roll, pitch, yaw y throttle enviados por el usuario y de los valores de realimentación de la situación actual del cuadricóptero. Los controles se manejan de forma manual, pero el UAV se estabiliza automáticamente en el espacio de 3 dimensiones.
- **Autónomo**: El UAV se controla mediante navegación por GPS, siguiendo una serie de waypoints indicados desde la estación base.

En los cuadricópteros se suele designar como modo manual al modo estabilizado. Además, se suelen implementar distintos modos de vuelo estable, donde alguno de los controles es manejado directamente por el autopilot, dejando al usuario la posibilidad de controlar sólo ciertas acciones. Por ejemplo, existen modos donde el autopilot controla los valores de throttle de forma que la altitud se mantiene constante, mientras que el usuario es capaz de controlar los movimientos de traslación y rotación haciendo uso de los controles de roll, pitch y yaw.

2.1.3. *IMU*

IMU (Inertial Measurement Unit) es un dispositivo electrónico que se utiliza para obtener mediciones inerciales: velocidad angular, aceleración lineal y campos magnéticos. Según la cantidad de sensores y el número de ejes ortogonales capaces de medir, las IMUs se clasifican según sus grados de libertad: 6 grados (acelerómetro 3D, giróscopo 3D), 9 grados (acelerómetro, giróscopo y magnetómetro 3D). 11 grados (IMU de 9 grados + sensor de presión barométrico y sensor de temperatura), etc. En la actualidad es común encontrar IMUs que incorporen acelerómetros, giróscopos y magnetómetros que toman medidas de tres ejes ortogonales basados en MEMS (Micro Electro Mechanical System), pequeños circuitos integrados electrónicos que no requieren de grandes masas inerciales como las IMUs empleadas en los aviones comerciales. Las IMUs MEMS son muy pequeños, de bajo coste y de poco consumo energético, pero no ofrecen tanta precisión ni estabilidad en la medición como los costosos y voluminosos sistemas empleados en la aviónica comercial [13] [14].

2.1.3.1. Giroscopio

Los giroscopios se utilizan para medir la velocidad angular con la que gira un objeto alrededor de un eje. Por lo general se utilizan las coordenadas geográficas de la tierra como ejes de referencia en torno a los cuales se mide el desplazamiento. El sensor se basa en una diminuta masa sometida a vibraciones controladas por medio de campos eléctricos, en los 3 ejes cartesianos, y midiendo la alteración del movimiento de esta masa como consecuencia de la fuerza de Coriolis resultante de movimientos angulares.

2.1.3.2. Acelerómetro

Los acelerómetros son sensores que se utilizan para medir la aceleración lineal de un objeto. Por lo tanto, los acelerómetros son capaces de medir la aceleración de la gravedad. El sensor se basa en una diminuta masa suspendida por enlaces elásticos, libre de moverse y sometida a un campo eléctrico en 3 dimensiones, que por efecto de la inercia y/o gravedad se desplaza de su posición central ocasionando cambios en el patrón del campo eléctrico en los ejes de desplazamiento, que se traducen y miden como cambios de capacitancia eléctrica.

2.1.3.3. Magnetómetro

Los magnetómetros se utilizan para medir los campos magnéticos, como los de la tierra. Aunque el campo magnético terrestre es un vector de 3 dimensiones, en el caso de los UAVs se puede simplificar el vector de campo magnético a 2 dimensiones asumiendo que durante el vuelo de la aeronave no se esperan grandes modificaciones del patrón magnético de la tierra (declinación magnética). Se utilizan principalmente para obtener la orientación absoluta de la aeronave, su correcta calibración es muy importante para obtener un control del yaw preciso.

2.1.4. *Otros sensores*

Los sensores que típicamente incorporan las IMUs pueden ser ampliados para obtener información de la altitud o la velocidad.

2.1.4.1. Barómetro

Los barómetros son sensores de presión atmosférica. Miden la variación de la presión estática cuando el UAV cambia de altitud. Por lo tanto, las medidas de presión que realizan este tipo de sensores sirven para obtener una muy buena estimación de la altitud a la que se encuentra el UAV.

2.1.4.2. Airspeed sensor

Se utilizan para medir la velocidad a del UAV respecto a la velocidad del aire. Usando sensores de presión diferencial se obtiene la diferencia entre la presión estática y la presión dinámica, que se consigue gracias a la utilización de los llamados tubos Pitot. A partir de la medida de presión diferencial se calcula la velocidad indicada, teniendo en cuenta la constante de densidad del aire a nivel del mar. Esta velocidad indicada está sujeta a errores, por lo que se suelen utilizar las medidas de otros sensores, como el barómetro, para calcular la velocidad real.

2.1.4.3. Sonar

Recibe su nombre de las siglas en inglés SOund Navigation And Ranging. Los sonar son típicamente sensores de ultrasonidos, es decir, utilizan ondas sonoras de alta frecuencia para detectar y localizar objetos. En los UAVs se utilizan para detectar la distancia al suelo cuando se realiza la maniobra de aterrizaje autónomo o bien para detectar objetos cercanos durante el vuelo. A diferencia de otros sensores de distancia, como el láser, el sonar tiene un ángulo de detección de aproximadamente 60º lo que le permite detectar objetos en un área.

2.1.5. *GPS*

El uso de GPS (Global Positioning System) permite determinar la altitud, velocidad aérea, orientación y posición del UAV basándose en señales de navegación por satélite. Sin embargo la velocidad de obtención de los datos es bastante más lenta (entre 1 y 5 muestras por segundo, típicamente) que utilizando otro tipo de sensores que también permitan obtener la orientación, altitud y velocidad. Por otro lado, el uso de GPS proporciona una referencia precisa a largo plazo, además de ser el sistema más fiable de obtención de la posición tanto global como local, que permiten el vuelo autónomo del UAV.

2.2. Sistema de control

2.2.1. *Descripción del controlador de vuelo*

El sistema de control de un UAV se suele dividir en dos partes interconectadas entre sí. Por un lado se encuentra el controlador de estabilidad que controla la orientación e inclinación del UAV (*attitude* en inglés) y por otro lado el controlador de posición. La conexión entre los dos controladores se hace en forma de cascada dentro de un mismo lazo cerrado, donde el control de estabilidad formaría el lazo interno y el control de posición el lazo externo. La Figura 2-5 muestra el diagrama del controlador.

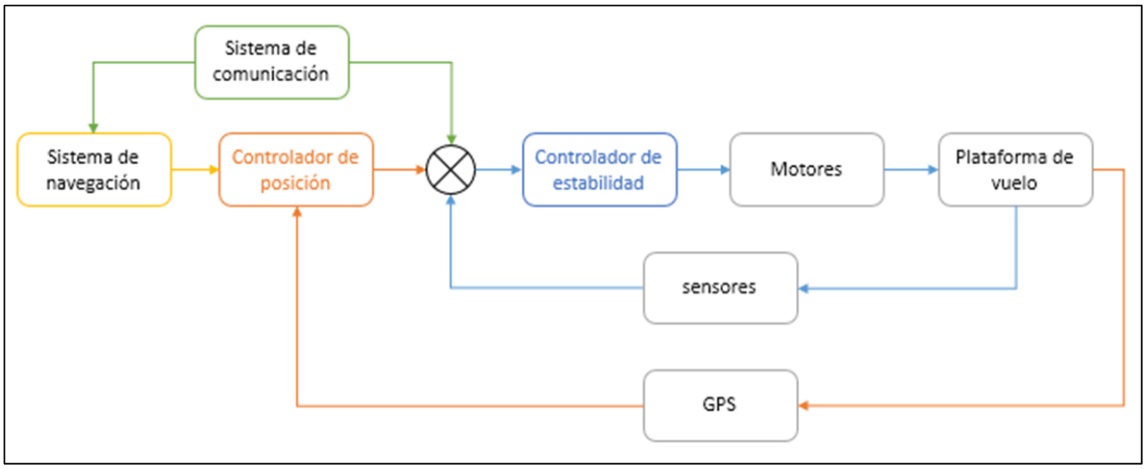


Figura 2-5. Sistema de control

Cuando el UAV se encuentra en modo manual solo actúa el sistema de control de estabilidad, mientras que en modos con mayor autonomía, es necesario el uso del controlador de posición.

El controlador de estabilidad recibe tanto la orientación, valores de yaw, pitch y roll, como la orientación deseada, que a su vez puede ser recibida desde el sistema de comunicación como una orden desde la estación base o desde el controlador de posición. El controlador de estabilidad calcula los nuevos valores de roll, pitch y yaw y se los transmite a los actuadores de los motores; en el caso de los cuadricópteros, en forma de 4 señales PWM que representan la velocidad angular de cada uno de los cuatro motores.

Para estimar la orientación en 3 dimensiones de los UAVs es muy común la utilización de filtros Kalman [15] y los algoritmos de control de orientación más utilizados se basan en controladores PID [16] [17].

El controlador de posición recibe la posición actual del UAV gracias al uso de GPS y otros sensores de altitud, así como la siguiente posición deseada. Esta posición deseada es calculada por el sistema de navegación que puede funcionar en forma de misión programada o calcular en el mismo UAV la posición por las órdenes recibidas desde el sistema de comunicación.

2.2.2. *Controlador PID*

El controlador PID (Proportional, Integral, Derivative) genera una acción de control basada en el error, diferencia entre la consigna y el valor actual del sistema: toma como entrada la diferencia entre el valor deseado y el valor actual y aplicando realimentación intenta reducir ese error a cero [18].

Los controladores PID cuentan con 3 acciones de control, proporcional, integral y derivativa, donde cada una de ellas cumple con una función específica:

- **Acción de control proporcional**: Funciona como un amplificador de ganancia ajustable, varía instantáneamente con el error y alcanza un valor estacionario cuando el error lo alcanza. De manera práctica, esta acción de control se comporta como la fuerza de un muelle: a mayor ganancia, mayor fuerza ejercerá este control sobre la salida en presencia de error.
- **Acción de control integral**: Se suele denominar acción de control de reajuste, el valor de la salida del controlador varía a una razón proporcional a la señal de error. Para un error de cero, la salida del controlador permanece en estado estacionario. De manera práctica, esta acción de control incrementa gradualmente su fuerza a medida que el error es diferente de cero, ayudando a reducir el error estacionario.
- **Acción de control derivativa**: En ocasiones se denomina control de velocidad, donde la magnitud de la salida del controlador es proporcional a la velocidad de cambio de la señal de error. La acción de control derivativa tiene un carácter de previsión, dado que es proporcional a la tendencia de cambio de la señal de error. De manera práctica, esta acción de control se opone a los cambios en la salida del sistema, como lo haría un amortiguador en la rueda de un coche.

El controlador PID es la combinación de las tres acciones como se puede ver representado en el siguiente diagrama:

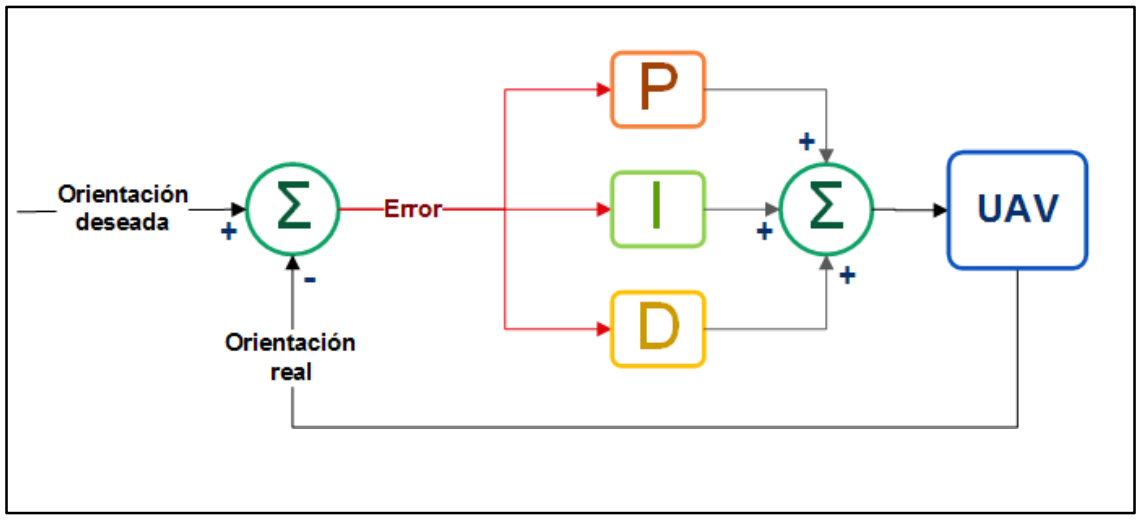


Figura 2-6. Controlador PID

2.3. Sistema de comunicación

En la mayoría de los casos el sistema de comunicación entre el UAV y la estación base se compone de varios enlaces. A continuación se presentan varias de las configuraciones que se suelen utilizar. El primero de los diagramas, Figura 2-7, muestra el esquema más común y completo.

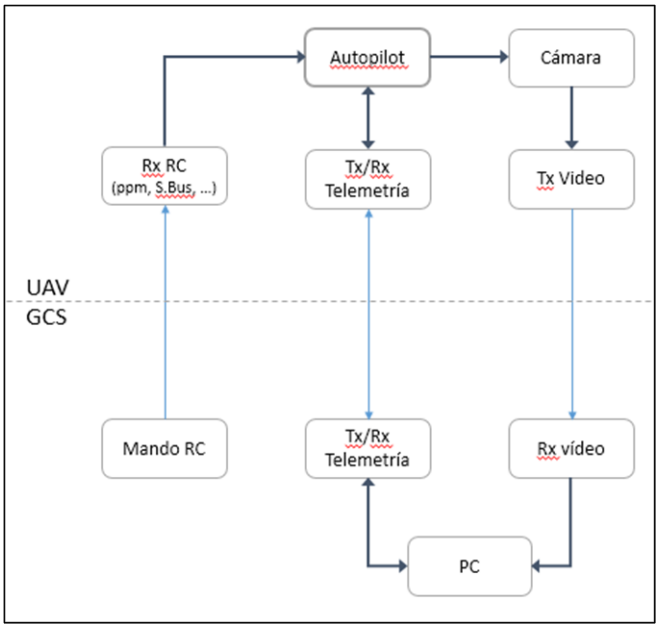


Figura 2-7. Esquema de comunicación típico

Se trata de tres enlaces diferentes: enlace de control, enlace de telemetría y enlace de vídeo. En este tipo de sistemas el enlace de control se utiliza para controlar la aeronave cuando se encuentra en modo manual, así como seleccionar el modo de vuelo. Una de las marcas más populares en sistemas de radiocontrol es Futaba [19], que cuenta con un sistema de comunicación propio: S.Bus. Estos sistemas de radiocontrol utilizan modulaciones basadas en FHSS (Frequency Hope Spread Spectrum). Además, son muy populares también los sistemas DSM (Digital Spectrum Modulation) de la marca Spektrum [20], cuyo sistema de modulación se basa en DSSS (Direct Sequencing Spread Spectrum), así como otro tipo mandos que utilizan modulación PPM (Pulse Position Modulation). La banda de frecuencia más utilizada en la actualidad por este tipo de sistemas es 2.4GHz.

Figura 2-8. Mando radiocontrol

El enlace de telemetría comunica el UAV con un PC en la estación base. El ordenador se utiliza tanto para visualizar y analizar los datos de telemetría como para gestionar las órdenes de navegación en modo de vuelo autónomo.

Por último el enlace de vídeo permite ver la imagen captada por la cámara integrada en el UAV en el monitor del ordenador. Por lo general se utilizan sistemas de transmisión de vídeo analógicos, que transmiten señales con modulación FM.

En los siguientes esquemas se trata de simplificar la comunicación reduciendo el número de enlaces inalámbricos.

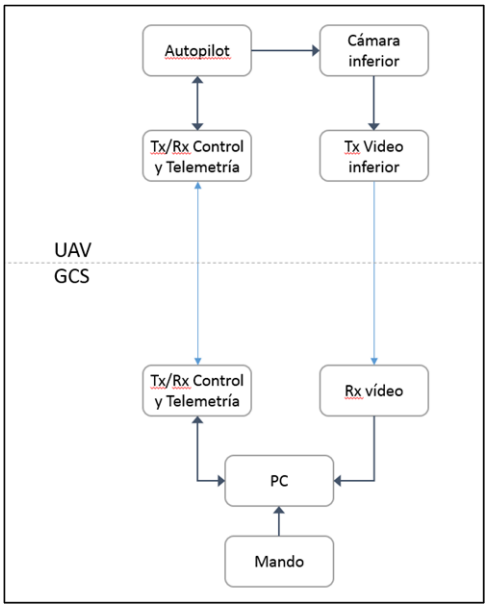


Figura 2-9. Esquema de comunicación sin mando RC

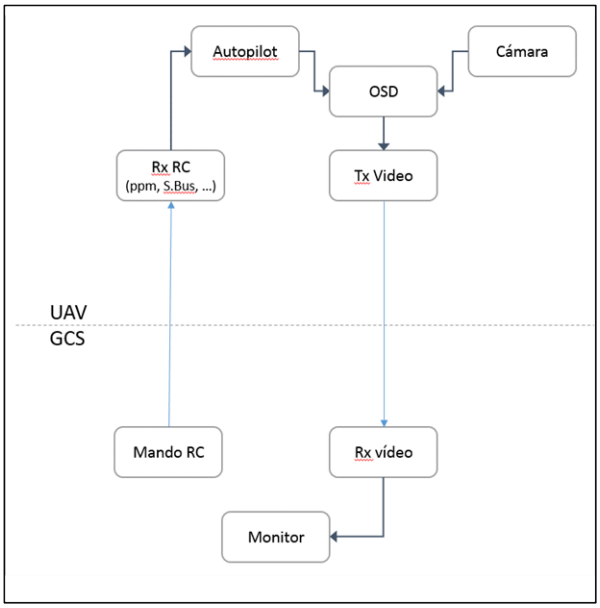


Figura 2-10. Esquema de comunicación FPV

La primera de las alternativas, es el esquema utilizado en sistemas de FPV (First Person View), donde se da más importancia al vídeo y el control manual, mientras que pierde importancia la navegación autónoma. En estos casos la información de la telemetría es solo visual y se envía integrada en la imagen de vídeo, que normalmente se visualiza utilizando gafas de FPV.

Figura 2-11. Gafas FPV

Para integrar la información de telemetría en el video se suelen usar dispositivos comerciales conocidos como OSD (On Screen Display).

Figura 2-12. Placa OSD

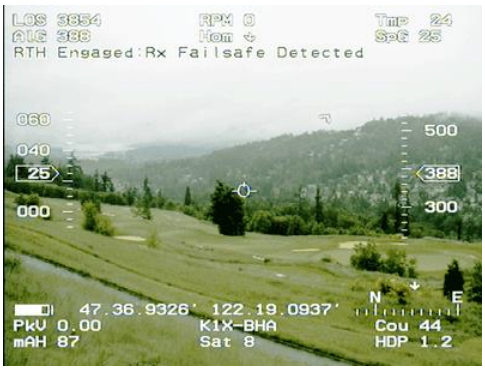


Figura 2-13. Imagen de vídeo con telemetría

La segunda alternativa consiste en utilizar el enlace de telemetría para transmitir las órdenes de control, manteniendo el enlace de vídeo independiente. Este proyecto implementa una solución muy similar a esta, por lo tanto se explicará con más detalle en los próximos apartados.

2.4. Sistema de navegación

Los sistemas de navegación de los UAVs generalmente se basan en waypoints, es decir, coordenadas geográficas que el UAV recorre en secuencia para completar una misión.

Se trata de un vuelo semiautónomo porque cuando se quiere que un UAV vuele de un punto a otro sin ser manejado con un mando, normalmente se incluyen en la ruta otros puntos por donde debe pasar. En general, los waypoints se utilizan para evitar obstáculos en rutas directas o bien para indicar al UAV una zona que se quiera supervisar.

El sistema de navegación de los UAV también suele contar con control de barrera geográfica (geofence). Consiste en delimitar una zona mediante waypoints que forman un polígono de forma que el UAV no pueda volar fuera del área virtual.

Tanto la ruta como las delimitaciones de la barrera geográfica se pueden encontrar predefinidas en el sistema de a bordo del UAV o bien pueden ser definidas o modificadas en tiempo real desde la estación base.

La navegación semiautónoma hace uso del sistema de control de vuelo: emite consignas de vuelo para modificar roll, pitch, yaw, elevación y velocidad de desplazamiento. Por medio de la posición geográfica global de la aeronave (latitud, longitud y altura) determinadas por el GPS el sistema de navegación cierra el lazo de control, calculando el error entre esta posición y el waypoint objetivo (consigna), generando así las consignas de vuelo necesarias.

Entre las posibilidades que este sistema de navegación ofrece se encuentran, además, dos funcionalidades automáticas importantes: mantener la posición actual y retornar a la estación base automáticamente. La primera permite a la aeronave permanecer en un punto determinado en el volumen 3D a pesar de las variaciones de las condiciones atmosféricas (viento, presión); y la segunda permite a la aeronave un conjunto de funciones de recuperación en caso problemas: pérdida de enlace con la estación base, poca energía remanente en las baterías, etc.

3 Diseño

3.1. Introducción

El principal objetivo de este proyecto es conseguir una buena comunicación entre al UAV y la estación base de modo que el control de éste sea fiable. Para poder realizar un control adaptado a este UAV es necesario conocer el funcionamiento concreto del autopilot y cómo gestiona los sistemas de comunicación, control y navegación.

En este apartado se va a realizar una descripción detallada de todos los componentes seleccionados para llevar a cabo este proyecto. En primer lugar se realizará una descripción del sistema de comunicación (3.2). A continuación se presentará un esquema de conexiones y se explicarán las características físicas de cada uno de los dispositivos a utilizar (3.3). Posteriormente se justificará la utilización de cada uno de ellos explicando la configuración seleccionada para cada uno de los módulos en los que se divide el sistema: enlace de vídeo (3.4), enlace de datos (3.5), controlador de vuelo (3.6) y estación base (3.7).

Es importante resaltar que todos los dispositivos utilizados son módulos comerciales que se han seleccionado teniendo en cuenta las características esperadas para el funcionamiento del cuadricóptero.

3.2. Descripción del sistema de comunicación

El sistema que se ha implementado en este proyecto cuenta con un enlace de datos bidireccional y dos enlaces descendentes (downlink) de vídeo. En el UAV, un autopilot implementa el controlador de vuelo e integra los dos enlaces de comunicación con la estación base. En la estación base se utiliza un joystick de ordenador que hace las mismas funciones que cualquier mando RC, pero a diferencia de otros sistemas se ahorra un enlace inalámbrico, transmitiendo las órdenes de control a través del mismo enlace que la telemetría desde el ordenador. En la

Figura 3-1 se puede ver un diagrama simplificado.

Esta configuración ofrece además la posibilidad de control automático desde el ordenador: las acciones de mando las puede generar el ordenador en lugar del operario de la aeronave.

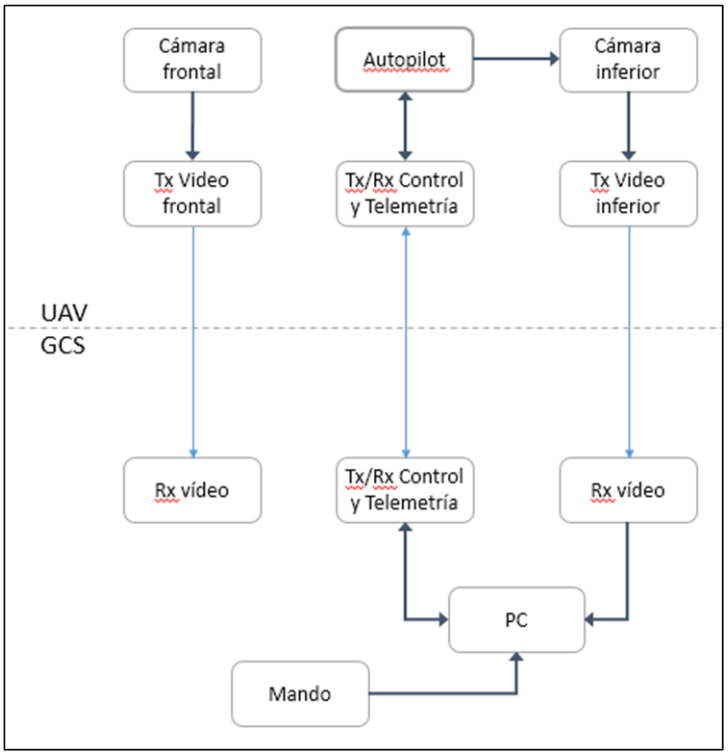


Figura 3-1. Esquema de comunicación utilizado

3.3. Descripción de los componentes

En la Figura 3-2 se puede ver el diagrama de conexiones de todos los dispositivos que se van a integrar en el UAV con el fin de proporcionar la comunicación y el control deseados.

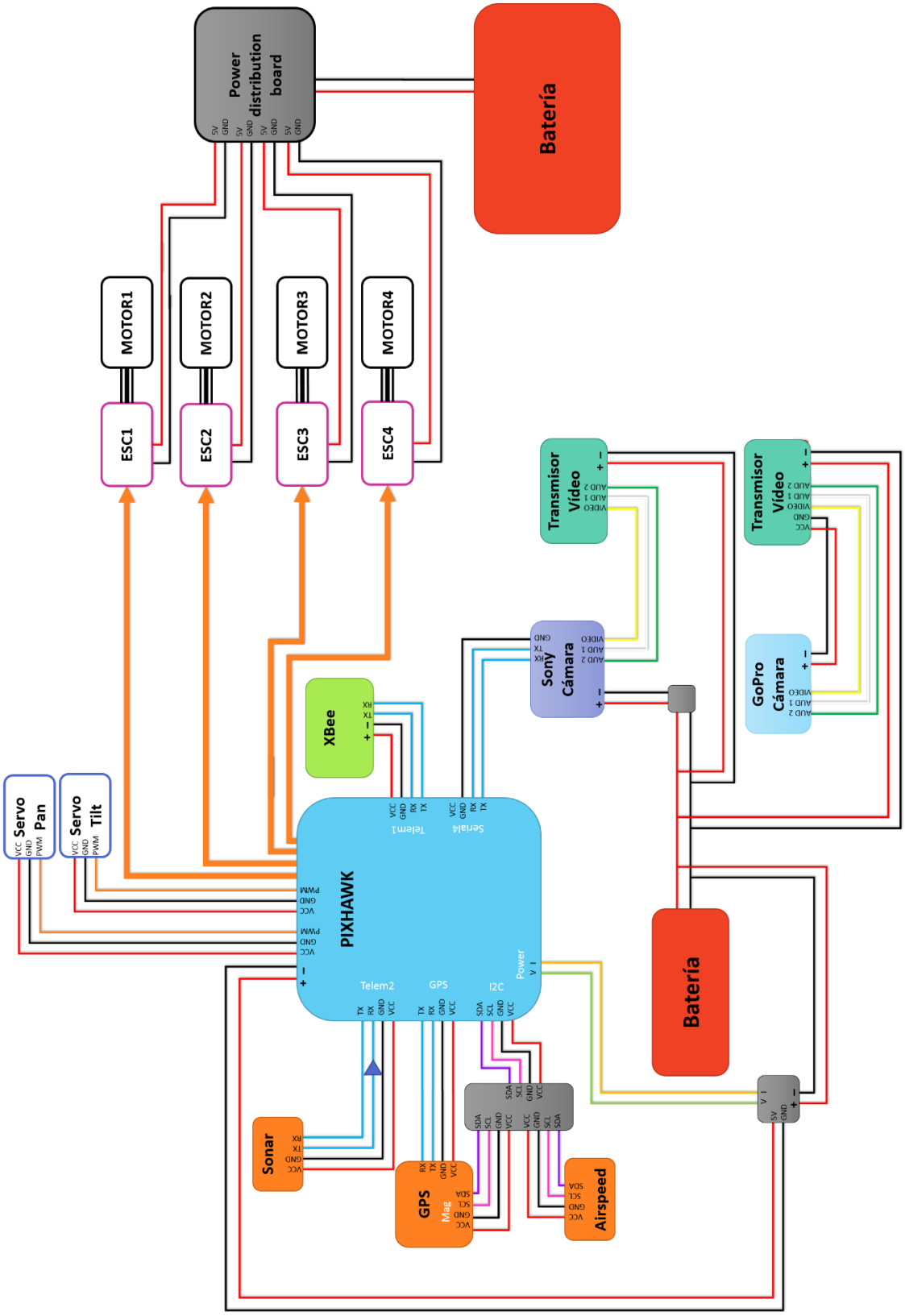


Figura 3-2. Diagrama de conexión UAV

En la estación base se van a utilizar los siguientes dispositivos:

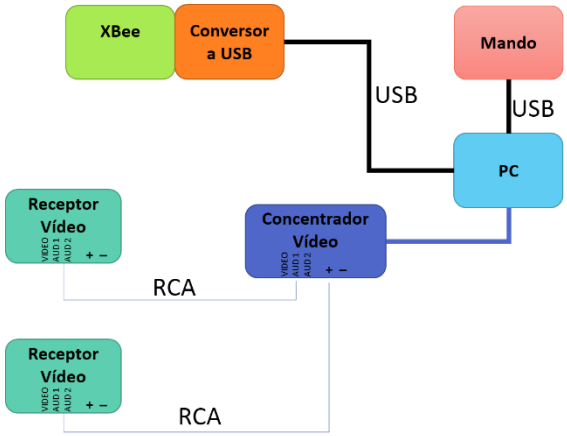


Figura 3-3. Diagrama de conexión GCS

A continuación se detallan los componentes que se utilizan tanto en el UAV como en la estación base.

3.3.1. *Autopilot – Pixhawk*

Figura 3-4. Módulo Pixhawk

El controlador de vuelo que se va a utilizar es el Pixhawk [21]. Se trata de un dispositivo electrónico que implementa las funciones de controlador de vuelo, gestión de sensores de la aeronave y gestión de las comunicaciones con dispositivos externos. Tanto el hardware como el software cuentan con licencia libre.

Lleva integrados todos los sensores necesarios para implementar una IMU de 9 grados de libertad, y barómetro. Además cuenta con varios puertos de conexiones que permiten añadir otros sensores y GPS: UART, I2C, SPI, CAN, además de 14

entradas/salidas digitales, normalmente diseñadas para ser configuradas como salidas PWM de control de servomotores.

A pesar de la gran complejidad del sistema, tan solo pesa 38 g y el consumo de energía está muy optimizado ya que sin tener en cuenta periféricos, tan solo necesita unos 250 mA.

Se explica con más detalle en el apartado 3.6.

3.3.2. *GPS*

Figura 3-5. GPS

El GPS incorpora un módulo u-blox 6 con chipset LEA-6H. El receptor GPS tiene una precisión de localización de posición de 2,5 m con una tasa de actualización de datos de hasta 5 Hz.

Utiliza una antena cerámica de tipo parche de Taoglas, con una ganancia máxima de 1,55 dB.

El módulo GPS cuenta además con un magnetómetro de 3 ejes. Se conecta al controlador de vuelo mediante un puerto serie para transmitir los datos del GPS y mediante uno I2C para enviar los datos del magnetómetro.

La comunicación serie es bidireccional y utiliza un protocolo de comunicación propio del módulo GPS. Se trata del protocolo UBX, que encapsula los datos en paquetes con la siguiente estructura de trama:

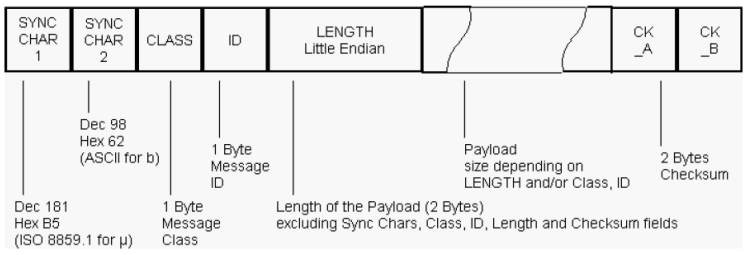


Figura 3-6. Trama de datos GPS

El magnetómetro es el modelo HMC5883L, un sensor magneto-resistivo que ofrece los resultados a través de bus I2C. Es capaz de medir campos magnéticos entre -8 y 8 Gauss, con una precisión de 2 mGauss. La frecuencia de actualización es de 160 Hz, es decir, es capaz de ofrecer datos cada 6 ms. Aunque la IMU del Pixhawk ya incorpora un magnetómetro similar, este módulo GPS con magnetómetro ofrece la posibilidad de emplear un sensor adicional que probablemente se encuentre en una ubicación de la aeronave más conveniente para realizar sus mediciones. Los magnetómetros son sensibles a los materiales ferromagnéticos (tornillos, estructuras metálicas, motores, conductores con alta corriente, etc.) de la aeronave y por lo tanto su instalación mecánica puede afectar la calidad de sus medicines. El módulo GPS requiere estar instalado en la zona más elevada de la aeronave para que tenga la mejor vista al cielo posible (la mejor Line-of-Sight hacia los satélites GPS), lo que generalmente favorece también al magnetómetro.

El módulo completo pesa 16.8 g y el consumo de corriente de pico se encuentra en torno a 40 mA.

3.3.3. *Airspeed sensor*

Figura 3-7. Airspeed sensor

El sensor de velocidad de aire que se utiliza es el modelo MS4525DO fabricado por Measurement Specialties. Mide temperatura y presión atmosférica. Interesa la medida de la presión diferencial, donde el rango de medida es de entre -1 y 1 psi.

El sensor transmite a través del puerto I2C un valor de 2 bytes con el que el autopilot calcula la presión diferencial medida. Utilizando la formula indicada en la hoja de datos del fabricante, el autopilot realiza los siguientes cálculos:

$$P_{diff} = \frac{(dp - 16383) \cdot (P_{MAX} - P_{MIN})}{0.8 \cdot 16383} + P_{MIN}$$

Donde dp es la medida entregada por el sensor, y P_{MAX} y P_{MIN} son los valores máximo y mínimo que se pueden obtener, es decir, 1 y -1 psi.

La equivalencia entre la unidad de medida de presión psi (pounds-force per square inch) y la unidad de medida del sistema internacional, Pascal, es: 1psi = 6894.75 Pa. Por tanto, para obtener la presión diferencial en unidades del sistema internacional se multiplica el valor calculado antes por la constante de equivalencia.

Por último se calcula la velocidad del aire indicada teniendo en cuenta la densidad del aire a nivel de mar y la velocidad verdadera usando los datos de presión calculada por el barómetro y la temperatura ambiente.

El peso es aproximadamente de 7 g y tiene un consumo de energía muy reducido, sólo necesita 3 mA de corriente.

3.3.4. *Sonar*

Figura 3-8. Sonar

El sonar empleado es el modelo MB1240 de la serie XL - MaxSonar – EZ. Los datos medidos se pueden obtener de forma analógica, en forma de señal PWM o mediante salida serie RS232. Detecta objetos a una distancia entre 0 y 7,65 m, con una precisión de 0.01 m. Los objetos detectados que se encuentren a distancias entre 0 y 0.2 m se indican como si estuvieran a una distancia de 0,2 m, por lo tanto, el rango de información es de 0,2 a 7,65 m. Las medidas del sensor se obtienen cada 99ms.

El sonar cuenta con un transductor que funciona tanto como transmisor como receptor. En una primera fase de emisión, envía una serie de pulsos de ultrasonidos a una frecuencia de 42 KHz. La duración del tren de pulsos suele ser de alrededor de 1ms. A continuación el sensor pasa a modo receptor y espera recibir ecos de los pulsos enviados, lo que indicaría la presencia de algún objeto. Una vez detectado un eco se calcula la distancia al objeto que lo ha producido mediante el tiempo transcurrido (TOF – Time Of Flight–) desde la emisión del pulso hasta la recepción del eco.

El autopilot recibe los datos medidos a través de un puerto serie, como se explica en el apartado 4.3.5 y los utiliza para mejorar la precisión de la estimación de la distancia al suelo cuando el UAV vuela a menos de 4 menos del suelo. El sonar es especialmente importante en multicópteros para realizar un aterrizaje autónomo.

El sonar pesa 5.9 g y consume una corriente de 100 mA.

3.3.5. *Cámaras*

Figura 3-9. GoPro Hero 3+

Se utilizan dos cámaras de vídeo. La primera se sitúa en la parte anterior, estará encendida desde el primer momento que el UAV se encienda, es la cámara de navegación del UAV. Se trata de GOPro Hero 3+, una cámara de tamaño muy reducido y capaz de obtener vídeo con una calidad HD. Pesa tan solo 74 g y necesita 250 mA para su funcionamiento.

Figura 3-10. Sony Block camera

La segunda cámara de vídeo va situada en la parte inferior del UAV y cuenta con un sistema de pan/tilt. Esta cámara no estará encendida en todo momento sino que se podrá controlar su funcionamiento enviando órdenes a través del enlace de control hasta el módulo autopilot. Se trata de una cámara de la familia Sony Block, diseñadas para ser utilizadas como cámaras de vigilancia. Cuentan con un puerto de comunicación serie a través del cual son controladas y configuradas. El zoom de la cámara seleccionada permite hasta 30 aumentos ópticos. Pesa 200 g y consume aproximadamente 400 mA de corriente.

3.3.6. *Pan/Tilt - Servos*

Igual que toda la estructura del UAV, el sistema de pan/tilt ha sido diseñado por el departamento de mecánica de Robomotion. Cuenta con dos servomotores para controlar los dos posibles movimientos de la cámara. El peso del sistema pan/tilt se incluye en el peso especificado para toda la estructura. Los servomotores pesarán aproximadamente 60 g y su consumo de corriente asciende a 100 mA.

El sistema de pan/tilt ofrece la posibilidad de ajustar la postura relativa de la cámara secundaria con respecto a la estructura de la aeronave. De esta manera es posible desplazar la aeronave en una dirección mientras la cámara secundaria apunta en una dirección diferente. Una posible aplicación de esta función es la de seguimiento visual de un objetivo mientras la aeronave se desplaza en diferentes direcciones: la observación de una zona de desastre mientras la aeronave realiza un vuelo circular sobre el área.

3.3.7. *Transmisores de vídeo*

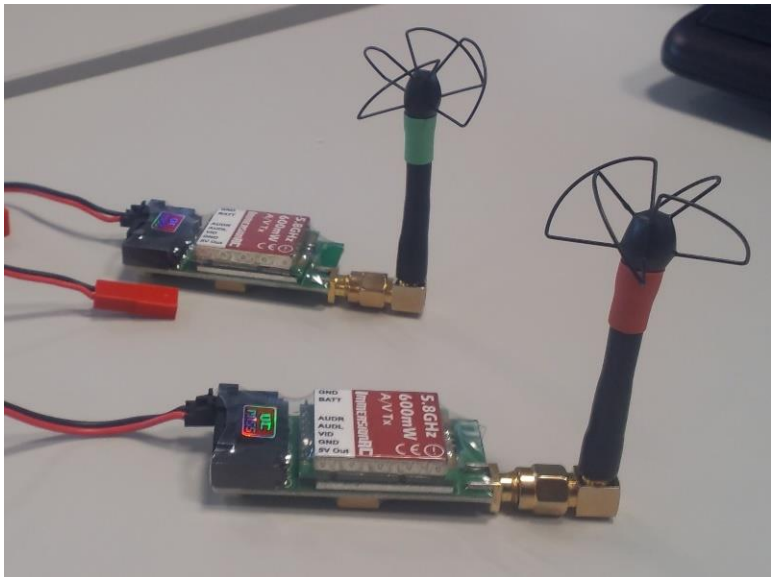


Figura 3-11. Transmisor de vídeo Inmmersion RC

Se utilizan dos transmisores de vídeo, uno por cada cámara. Se trata de transmisores analógicos de definición estándar de la marca Inmmersion RC. Emiten en la banda de 5.8 GHz, con la posibilidad de elegir entre 7 canales diferentes, a una potencia de 600mW (28dBm). Se utilizan además antenas de polarización circular con una ganancia de 2dB. El peso de cada uno de los transmisores es de 18g, al que hay que añadir el peso de la antena, de aproximadamente 5g. El consumo de potencia según las especificaciones del fabricante es de 3W independientemente de la tensión con la que se alimente el dispositivo, la cual puede tomar cualquier valor entre 6 y 25V. En un primer prototipo toda la electrónica del UAV irá alimentada con una batería LiPo de 3 celdas, lo que

proporciona una tensión media de 11.7V. Con estos datos, se estima que el consumo de corriente de cada uno de los transmisores sea de 256mA.

3.3.8. *Receptores de vídeo*

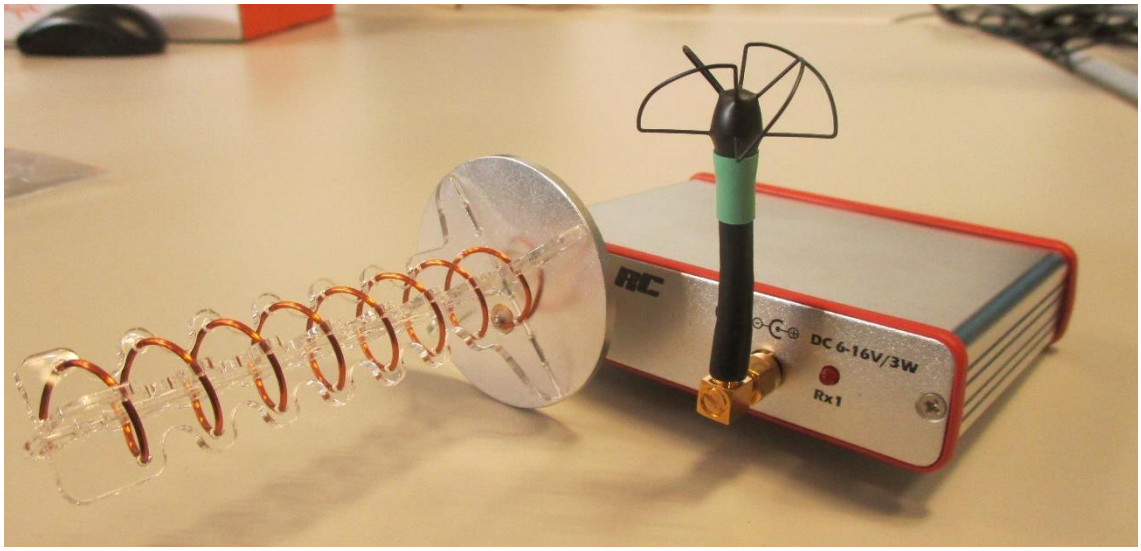


Figura 3-12. Receptor de vídeo InmersionRC

Igual que los transmisores, el sistema incluirá un receptor de vídeo para cada una de las señales. El modelo de receptor elegido es el 5800DUO de la marca Inmersion RC. Trabaja en las mismas frecuencias que los transmisores con una sensibilidad de -90dBm. Aunque el tamaño, peso y consumo de los receptores no es tan crítico porque siempre permanecen en la estación base, siempre es mejor cuanto más ligero sean y menos consuman. Su peso es de 169g cada uno.

3.3.9. *Transmisores/receptores de datos*

Figura 3-13. Módulo XBee

El envío y recepción de telemetría y comandos de control se realizan a través de módulos XBee. Es necesaria la utilización de dos módulos idénticos, uno en el UAV y otro conectado al ordenador de la estación base. Se han elegido los módulos XBee Pro

900 HP, que son capaces de transmitir con una potencia de 24 dBm y tienen una sensibilidad de -101 dBm transmitiendo a una velocidad de 200 Kbps (ancho de banda de 200 Kbps). Cada uno de los dispositivos pesa tan solo 8 gramos. Las antenas seleccionadas en este caso tienen una ganancia de 2dB y pesan 10 g.

Estos módulos, en su configuración básica, son un sustituto de cable de comunicaciones seriales RS232. Configurados de manera avanzada, permiten establecer una red de dispositivos en los que cada nodo puede servir automáticamente de repetidor para otro nodo más lejano.

3.3.10. *PC y mando*

En la estación base, para las primeras pruebas se ha utilizado un ordenador portátil con Ubuntu como sistema operativo. El software que se ha usado para las primeras pruebas para recibir la telemetría recibida es QGroundControl, un software libre que cumple con los requisitos básicos necesarios para la estación base. Además, se ha implementado un programa propio para controlar el UAV de forma manual con un joystick comercial de ordenador.

3.3.11. *Batería*

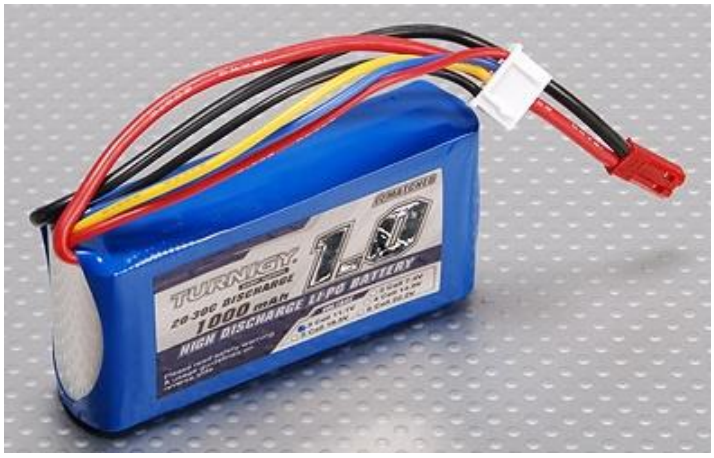


Figura 3-14. Batería

En el primer prototipo se ha utilizado una batería para la electrónica y otra para los motores. En posteriores versiones la batería de la electrónica desaparecerá y sólo se usará una o varias baterías para alimentar todo el sistema.

Para alimentar la electrónica se ha elegido una batería LiPo de 3 celdas. Este tipo de baterías de polímero de litio ofrece una tensión nominal de 3.9V por celda, por lo tanto en este caso se obtienen 11.7 V. Al tratarse de una batería provisional, se ha querido reducir al máximo su peso, por lo que se ha ajustado la capacidad de la batería. Se ha estimado que la autonomía del sistema de motores se sitúe entre los 20 y 30 minutos de

vuelo, por lo tanto, para ajustar la autonomía de la electrónica a estos tiempos se ha elegido una batería con capacidad de 1000mAh.

3.3.12. *Estudio de autonomía*

En los cuadricópteros el mayor gasto de energía es el producido por los motores. Por eso, la autonomía del sistema completo está limitada por la batería de los motores. Para alimentarlos se ha seleccionado una batería LiPo de 8 celdas con una capacidad de 22000 mAh. Con esta batería se ha estimado un tiempo de vuelo de entre 25 y 30 minutos, considerando que el cuadricóptero pesa 7.5 kg incluyendo la estructura, las beterías y la electrónica.

La siguiente tabla muestra un resumen de consumos de todos los componentes electrónicos:

Tabla 3-1. Consumo de corriente y peso de componentes

	Consumo de corriente	**Peso**
Pixhawk	250 mA	38 g
GPS	40 mA	16.8 g
Airspeed sensor	3 mA	7 g
Sonar	100 mA	5.9 g
Cámara GoPro (frontal)	250 mA	74 g
Cámara Sony (inferior)	400 mA	200 g
Servomotores	100 mA x 2 = 200 mA	60 g x 2 = 120 g
Video Tx	250 mA x 2 = 500 mA	18 g x 2 + 10 g = 46 g (con antena)
XBee	215 mA	8 g + 10 g = 18 g (con antena)
TOTAL	**1958 mA**	**525.7 g**

El consumo total de corriente es de 1958 mA, por lo tanto, para conseguir una autonomía de 30 minutos es suficiente con una batería de 1000 mAh de capacidad. Hay que tener en cuenta que se esto ocurriría en el peor de los casos ya que la cámara inferior no va a estar siempre encendida y se ha considerado las corrientes de pico de todos los sensores, servomotores y del módulo XBee.

Por otro lado, el peso de la batería es de 97 g y la estructura junto con los motores y la batería de los motores pesa aproximadamente 6.5 kg. Sumando esto con el peso de todos los componentes electrónicos se obtiene 7.1 kg, que es menos de los 7.5 kg considerados para estimar la autonomía, por lo tanto esta también puede mejorar.

3.4. Enlace de video

3.4.1. *Selección del sistema de vídeo*

El sistema de transmisión de vídeo consiste en dos enlaces inalámbricos capaces de funcionar al mismo tiempo. En este apartado se justifica la elección de los dispositivos y antenas elegidos y su configuración.

Los sistemas de transmisión de vídeo comerciales se pueden dividir en dos grandes grupos, analógicos y digitales. Los sistemas analógicos son los que se usan principalmente por aficionados, mientras que los sistemas digitales, similares a los utilizados en las cámaras de televisión, son los que se utilizan en campos profesionales.

Los sistemas de transmisión de vídeo digitales son más robustos frente a interferencias. La mayoría de ellos utiliza modulación COFDM (Coded Orthogonal Frequency Division Multiplexing). La señal recibida en recepción es de muy buena calidad ya que estos sistemas son capaces de transmitir vídeo en calidad HD usando sistemas de compresión de vídeo H.264 o MPGE-4.

La principal desventaja de los enlaces de vídeo digitales es que el precio es muy elevado. Los transmisores menos costosos que se pueden encontrar en el mercado tienen otras desventajas: por un lado la latencia de este tipo de transmisores es muy elevada, llegando a alcanzar tiempos de un segundo. Esta latencia se debe al tiempo requerido para comprimir la señal de vídeo. Por otro lado, la potencia necesaria para transmitir la señal es mayor, lo que conlleva un mayor gasto de batería y por tanto un descenso de la autonomía. Además, el tamaño de los transmisores digitales es bastante mayor que el de los analógicos, lo que supone un impedimento ya que en los UAVs el peso y volumen de los componentes es muy crítico.

Teniendo en cuenta lo expuesto anteriormente, para este prototipo se ha optado por enlaces de vídeo analógicos. Los dispositivos seleccionados transmiten señal de vídeo en formato PAL utilizando modulación FM.

Tanto los sistemas digitales como los analógicos disponibles en el mercado suelen trabajar en las mismas bandas de frecuencia. La más utilizada es la banda de frecuencias centrada en 5.8GHz, aunque existen transmisores que operan en 2.4 o 1.2 GHz. Los transmisores más modernos y que transmiten con la mejor calidad de vídeo son los que trabajan en 5.8GHz.

El transmisor de InmersionRC trabaja entre 5740MHz y 5860MHz, cuenta con 7 canales con una separación de 20 MHz entre cada uno de ellos. Este transmisor, además de enviar la señal de vídeo, ofrece la opción de transmitir uno o dos canales de audio.

Otra ventaja de esta frecuencia es que permite utilizar antenas más pequeñas para conseguir ganancias mayores.

Las características de propagación de las ondas electromagnéticas en frecuencias altas favorecen la transmisión en línea recta, lo que permite obtener un mayor alcance siempre y cuando exista línea de vista (LOS). Estas características de propagación también provocan que los sistemas sean más susceptibles a sufrir desvanecimiento por multitrayecto. Para reducir este efecto, se utilizan receptores con diversidad que cuentan con dos antenas: las dos antenas reciben la señal al mismo tiempo pero con distinta fase y el receptor se encarga de seleccionar la señal con mejor calidad.

En el enlace de vídeo se van a utilizar antenas de polarización circular. Este tipo de polarización se utiliza en transmisión de vídeo en UAVs para evitar las pérdidas de señal por polarización cruzada entre el UAV y la estación base. Además, aprovechando la diversidad del receptor, se van a utilizar dos tipos distintos de antena. La primera se trata de una antena omnidireccional con 2dB de ganancia. La segunda es una antena direccional con 12dB de ganancia y un ancho de haz de 60º. La Figura 3-15 y Figura 3-16 muestran imágenes de las antenas.

Figura 3-15. Antena omnidireccional de polarización circular

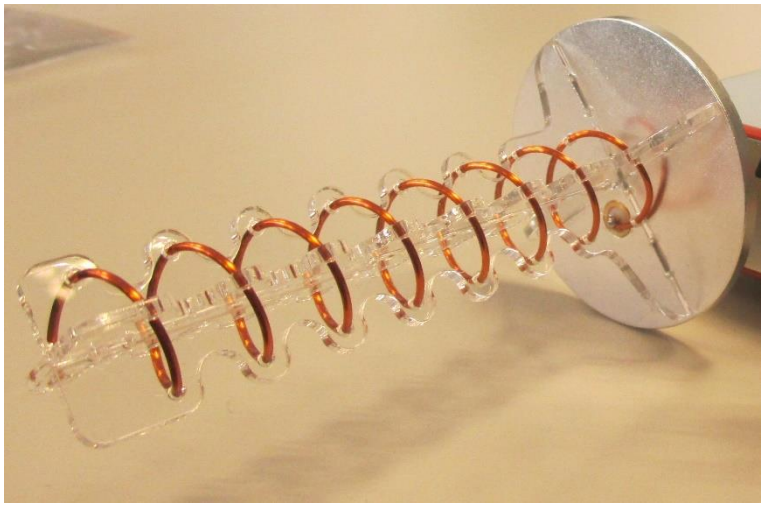


Figura 3-16. Antena direccional de polarización circular

La antena direccional del receptor siempre apuntará en la dirección de vuelo del UAV para obtener el mayor alcance posible, mientras que la antena omnidireccional se usa para garantizar la comunicación en vuelos próximos a la estación base. En la Figura 3-17 se puede ver el diagrama de radiación de las dos antenas, donde se puede comprobar cómo el lóbulo principal de la antena direccional cubre el nulo de radiación de la antena omnidireccional.

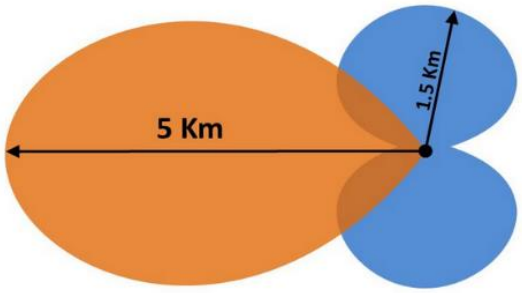


Figura 3-17. Diagrama de radiación antenas vídeo

Además, como se necesitan dos enlaces de vídeo operando a la vez, los dos enlaces utilizan polarización cruzada entre ellos. Uno de los pares receptor-transmisor utiliza polarización circular a derecha (RHCP, Right Hand Circular Polarization), mientras que el otro utiliza polarización circular a izquierdas (LHCP, Left Hand Circular Polarization). Por lo tanto se obtienen dos enlaces ortogonales entre sí, con lo que se consigue minimizar las interferencias gracias a la discriminación por polarización cruzada (XPD).

Como tanto los transmisores como los receptores disponen de 7 canales diferentes, para cada uno de los enlaces se escogen canales lo más separados en frecuencia posible.

3.4.2. *Conclusiones*

Como resumen de este apartado se presenta el siguiente esquema de todo el sistema de transmisión de vídeo:

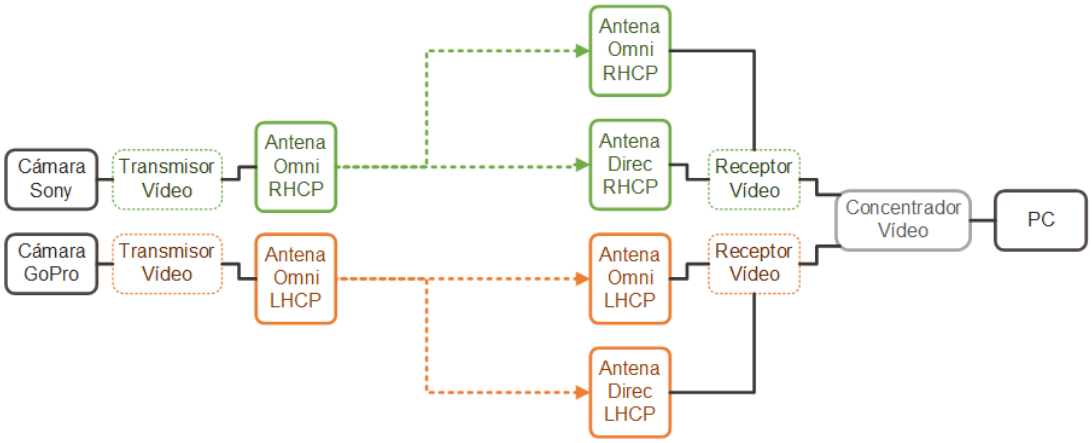


Figura 3-18. Sistema de transmisión de vídeo

3.5. Enlace de datos

3.5.1. *MAVLink*

Para implementar el sistema de transmisión de control y telemetría se ha decidido utilizar el protocolo de comunicación MAVLink [22] (Micro Air Vehicle Link). Mavlink es una librería de mensajes utilizados para mantener la comunicación entre vehículos aéreos y una estación base. Se trata de una librería muy ligera y fácil de integrar en cualquier sistema tanto de abordo como de tierra ya que la definición de los mensajes se hace sólo con cabeceras en lenguaje C. Estos mensajes empaquetan estructuras de datos y se envían a través de canales serie bidireccionales con poco overhead. Existe una librería de mensajes comunes, pero es posible implementar mensajes personalizados. El software Mavlink tiene licencia LGPL (GNU Lesser General Public Licencense) lo que permite que sea utilizado tanto en proyectos *close-source* como *open-source*.La anatomía de un paquete de datos MAVLink está inspirada en los estándares CAN (Controller Area Network) y SAE AS-4. En la siguiente imagen se puede ver un esquema de los campos de un paquete y en la tabla siguiente se explica cada uno de ellos.

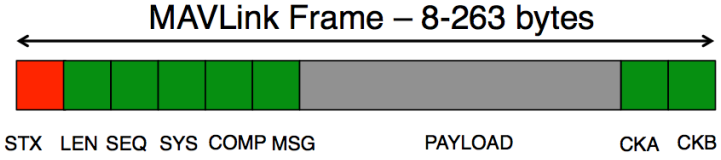


Figura 3-19. Trama de datos MAVLink

Tabla 3-2. Contenido de la trama de datos MAVLink

Byte	Contenido	Valor	Explicación
0	Start Transmission	0xFE	Indica el inicio de un nuevo mensaje.
1	Length	0 – 255	Longitud de los datos.
2	Sequence	0 – 255	Cada componente lleva una cuenta de las secuencias transmitidas. Permite detectar paquetes perdidos.
3	System ID	1 – 255	Identificación del sistema que envía el mensaje. Permite diferenciar entre diferentes UAVs.
4	Component ID	0 – 255	Identificación del componente que envía el mensaje. Permite diferenciar entre componentes de un mismo sistema.
5	Message ID	0 – 255	Identificación del mensaje. Define el significado de los datos y como deben ser decodificados.
6 a (n+6)	Payload	(0 – 255) bytes	Datos del mensaje. Dependen del ID del mensaje.
(n+7) a (n+8)	CRC		Checksum del mensaje completo excluyendo el byte de inicio.

Los paquetes de datos MAVLink comienzan con un byte de inicio, STX. Una vez que se comprueba que se ha recibido este byte se lee la longitud del mensaje en el siguiente campo. Conociendo la longitud de los datos que se han enviado se puede obtener y verificar los bytes de checksum. La pérdida o modificación de alguno de los bytes del mensaje ocasiona que la verificación del checksum falle, en este caso se desprecia el mensaje, y el receptor queda a la espera de un nuevo mensaje. Si el checksum coincide se procesa el paquete MAVLink, y se vuelve a esperar a que se reciba un nuevo byte de inicio de mensaje. Además, todos los mensajes MAVLink cuentan con un número de secuencia, que se utiliza como mecanismo de seguridad para detectar pérdidas de paquetes.

La longitud mínima de un mensaje MAVLink es de 8 bytes, para los mensajes usados como asentimiento (ACK), donde no son necesarios datos. La longitud máxima de los mensajes es de 263 bytes, con 255 bytes de datos.

Teniendo en cuenta el peor caso, cuando la longitud de todos los paquetes que se transmiten es de 263 bytes, se ha calculado la velocidad mínima del enlace para distintas frecuencias de transmisión.

Tabla 3-3. Velocidades de transmisión de datos con MAVLink

Frecuencia	Velocidad calculada	Valores estándar
20 Hz (50 ms)	42.080 bps	38.400 – 57.600 bps
50 Hz (20 ms)	105.200 bps	57.600 – 115.200 bps
100 Hz (10 ms)	210.400 bps	128.000 – 256.000 bps

Se estima que un enlace con una velocidad de transmisión de 115.200 bps es suficiente para este proyecto, ya que los mensajes que se van a enviar rara vez tendrán la longitud máxima, y poder enviar una media de 50 mensajes por segundo es un valor suficiente para mantener una comunicación estable y fluida entre la estación base y el UAV.

3.5.2. *XBee*

Se ha decidido utilizar módulos XBee porque son capaces de ofrecer la velocidad requerida en enlaces de datos bidireccionales. Además, estos dispositivos tienen un consumo de potencia muy reducido, son muy ligeros y a la vez su precio es muy asequible. Por otro lado, XBee cuenta con numerosos tipos de dispositivos con distintas características, es posible elegir entre frecuencias, potencias de emisión, velocidad de transmisión, tipo de antena, etc., siempre con la misma interfaz de comunicación. Esto ofrece una gran versatilidad por la posibilidad de cambiar entre un módulo u otro.

Para este proyecto se ha escogido el módulo XBee PRO 900HP que trabaja en la banda de frecuencias de 900 MHz. El principal motivo de haber elegido esta frecuencia en lugar de 2.4 GHz es la posibilidad de lograr mayores distancias de transmisión. La teoría de radiofrecuencia demuestra que a bajas frecuencias se pueden conseguir mayores distancias de transmisión debido a pérdidas de propagación menores. Además, según la hoja de datos del fabricante, con los módulos seleccionados es posible transmitir a 45 Km de distancia dependiendo de la configuración.

Por otro lado, debido a las características de propagación electromagnética, las señales que se transmiten a bajas frecuencias son menos susceptibles al desvanecimiento por multitrayecto. Esto se debe a que las señales tienen mayor facilidad para atravesar objetos o rodearlos.

Los principales inconvenientes de trabajar en bajas frecuencias son las limitaciones en ancho de banda, lo cual en este proyecto no es un inconveniente ya que el enlace se va a utilizar únicamente para transmitir datos, y el tamaño de las antenas, que deben ser mayores que las usadas en frecuencias mayores para una misma ganancia. Se ha elegido una antena de tipo dipolo de 2 dBi de ganancia.

Los módulos XBee utilizan protocolos de red en malla que proporcionan una forma eficiente en enviar datos entre diferentes nodos. Uno de los protocolos más popular de los utilizados por XBee es ZigBee en el que se definen tres tipos de nodos. En toda red ZigBee

hay un nodo Coordinador, uno o varios Routers y varios dispositivos finales. Cada tipo de nodo tiene sus propias funciones. El módulo XBee seleccionado utiliza otro tipo de protocolo, DigiMesh. Este protocolo es parecido a ZigBee pero tiene varias diferencias. DigiMesh tiene un único tipo de nodo, que puede realizar las funciones de cualquiera de los tres nodos ZigBee, lo que permite simplificar la configuración de la red y proporciona mayor flexibilidad. Además, utilizando módulos con protocolo DigiMesh se pueden alcanzar distancias de transmisión mayores que con módulos equivalentes que utilicen protocolo ZigBee.

En este proyecto no es necesario utilizar ninguno de estos protocolos de red, dado que al disponer únicamente de dos dispositivos se va a utilizar una comunicación punto a punto. No obstante, es importante conocer las posibilidades que ofrecen los dispositivos XBee ya que en un futuro podría existir un grupo de UAVs. En tal caso, utilizar el protocolo DigiMesh podría servir para proporcionar comunicación M2M o para ampliar el alcance de vuelo transmitiendo la información de un UAV a otro hasta llegar a la estación base.

Otra característica de estos dispositivos XBee es que utilizan FHSS (Frequency Hopping Spread Spectrum) para reducir las interferencias, además de hacer el enlace más robusto al desvanecimiento por multitrayecto. Los enlaces de comunicación que utilizan modulación con espectro ensanchado proporcionan mayor confidencialidad, debido a que el código de ensanchamiento sólo se conoce por el transmisor y el receptor involucrados en el enlace.

Con las antenas seleccionadas y la configuración de velocidad de transmisión de datos elegida, se espera un enlace que alcance más de 3 Km con línea de vista en espacio abierto y hasta 300 metros en interiores y transmisiones en entornos urbanos.

3.6. Controlador de vuelo – Autopilot

El módulo Pixhawk hace las funciones de controlador de vuelo teniendo en cuenta tanto los sensores de su propio hardware como los externos, a la vez que es capaz de gestionar la comunicación con la estación base, respondiendo a las órdenes de control recibidas o actuando sobre dispositivos externos como el sistema de pan/tilt u otras órdenes para cámaras.

Una de las consideraciones que se ha tenido en cuenta a la hora de seleccionar este dispositivo es que tanto el hardware como el software tienen licencia libre, y por lo tanto se tiene acceso al código fuente y circuitos originales, lo que ofrece la posibilidad de modificar o añadir funciones para adecuarlo al proyecto. Sin embargo, se encuentra en constante desarrollo por parte de los creadores, por lo que en cualquier momento pueden incorporarse cambios que no sean convenientes para nuestro propósito.

El módulo Pixhawk se desarrolla dentro del proyecto PX4, al que da soporte la universidad de Zurich en sus grupos de investigación Computer Vision and Geometry Lab [23], Autonomous Systems Lab [24] y Automatic Control Laboratory [25], además de una amplia comunidad de desarrolladores independientes [26].

En este apartado se explicará de forma detallada las características del dispositivo, dividiendo la descripción en dos apartados: hardware y firmware.

3.6.1. *Hardware*

El hardware del módulo Pixhawk es la combinación de dos dispositivos ya existentes:

- **PX4FMU**: La FMU (Flight Management Unit) es la unidad de gestión de vuelo. Se trata de la combinación del piloto automático y la IMU, unidad de medición inercial, lo que permite una solución completa de control para UAVs y otro tipo de vehículos de control remoto. Gestiona otros sensores y dispositivos conectados al bus de expansión de 30 pines, donde cuenta con buses I2C, UART, CAN y pines de E/S. Cuenta con 4 salidas PWM para emitir las consignas de los motores y conexión para GPS.
- **PX4IO**: La placa de IO (Input/Output) de PX4 se compone de una fuente de alimentación y un módulo de expansión para la unidad de gestión de vuelo PX4FMU. Proporciona salidas para servos, entradas para los receptores, dos relés de estado sólido, dos interruptores, salidas de potencia de corriente limitada de 5 V y una amplia gama de conectores de E/S adicionales.

Pixhawk integra en una única placa todas las funciones de ambos dispositivos. Cuenta con un procesador Cortex M4 STM32F427 de 32 bits capaz de trabajar a 168 MHz, 252 MIPS, y un procesador de seguridad STM32F103 de 32 bits por si falla en procesador principal. Tiene 256 KB de memoria RAM y 2 MB de Flash.

Pixhawk incorpora los siguientes sensores:

- **ST Micro L3GD20H**: giroscopio de 3 ejes, 16 bits
- **ST Micro LSM303D**: acelerómetro/magnetómetro de 3 ejes, 14 bits
- **MEAS MS5611**: barómetro
- **MPU6000**: giroscopio/acelerómetro

Cuenta con los siguientes tipos de conexiones:

- **5 UART: U**niversal **A**synchronous **R**eceiver-**T**ransmitter. Se trata de puertos serie. Las conexiones son Telem1, Telem2, GPS y Serial4/5.
- **2 CAN: C**ontroller **A**rea **N**etwork. Se trata de un bus de comunicación industrial serial.
- **Spektrum DSM / DSM2 / DSM-X®:** Entrada de datos compatible con los receptores de la marca Spektrum.
- **Futaba S.BUS®:** Entrada y salida compatible con el sistema de transmisión S.BUS de la marca Futaba. La entrada de datos se utiliza para conectar

receptores de mandos de control y la salida para motores que utilicen este sistema.

- **PPMsum:** Entrada de señal PPM (Pulse Position Modulation). La suma de todos los canales de un receptor RC se transmite en una única línea de datos.
- **RSSI: R**eceive **S**ignal **S**trenght **I**ndicator (PWM o voltaje) Entrada de nivel de señal.
- **I2C:** De las siglas **I**nter-**I**ntegrated **C**ircuit, se trata de un bus de comunicación serie síncrona. Se utiliza para conectar el sensor airspeed y el magnetómetro externo del GPS.
- **SPI:** Proviene de las siglas Serial Peripheral Interface. Bus serie local.
- **ADC: A**nalog to **D**igital **C**onverter**.** Entradas de 3.3 y 6.6 V
- **microUSB:** Cuenta con un puerto interno y una conexión externa. Se utiliza principalmente para actualizaciones de firmware, aunque también permite conectar con la consola del sistema.

3.6.2. *Firmware*

Pixhawk lleva instalado el sistema operativo NuttX [27], sobre el cual se ejecuta el software de control del autopilot. Se trata de un sistema operativo muy ligero pero a la vez eficiente que proporciona un entorno de programación con estilo POSIX (**P**ortable **O**perating **S**ystem **I**nterface Uni**X**). POSIX es un estándar que define una interfaz de sistema operativo y su entorno, incluyendo un intérprete de comandos (o "shell"), y programas de utilidades comunes para mejorar la portabilidad de las aplicaciones a nivel de código fuente.

El software de Pixhawk se divide en dos capas principales: PX4 middleware y PX4 flight stack.

- **PX4 middleware:** proporciona los drivers necesarios para los dispositivos externos así como un gestor de tareas, encargado de mantener una comunicación asíncrona entre aplicaciones independientes, llamado *uORB* (micro Object Request Broker). Esta parte del firmware es imprescindible para el funcionamiento del hardware Pixhawk.
- **PX4 Flight Control Stack:** se trata de una serie de aplicaciones que componen el controlador de vuelo. Proporciona vuelo autónomo por seguimiento de waypoints tanto para multicópteros como para aviones de ala fija. Una parte de código es común para cualquier aeronave que se utilice, pero existen aplicaciones específicas para multicópteros en las que se centra este proyecto.

La forma en que las aplicaciones o procesos se comunican unos con otros es una de las partes claves de la arquitectura software. La aplicación *uORB* se encarga de compartir estructuras de datos entre otros hilos o aplicaciones, usando un modelo de comunicación

de tipo publicación/suscripción. Es un sistema similar al utilizado por ROS [28] (Robots Operating System) pero de manera simplificada, donde las aplicaciones se conocen como nodos y los mensajes son estructuras de datos que se transportan en buses de datos virtuales conocidos como tópicos. Por tanto, una aplicación puede publicar un mensaje en un tópico para enviar datos o bien puede subscribirse a un tópico para recibir el mensaje que otra aplicación haya publicado. En cada tópico pueden publicar o subscribirse varias aplicaciones.

Para hacer una calibración manual de roll y pitch he añadido un tópico y se ha integrado en el software, en el apartado 4.3.4 se indican los pasos seguidos. Esta explicación puede servir de guía en caso de futuras modificaciones.

El firmware de Pixhawk es, igual que el hardware, una evolución del software de los módulos PX4 FMU y PX4 IO. En este caso sí se diferencian las dos partes, teniendo PX4FMU la mayor carga de aplicaciones del controlador de vuelo, mientras que el firmware de PX4IO se encarga de ejercer la comunicación con los actuadores de los motores y los posibles receptores de mandos RC. Es decir, las conexiones de la parte trasera del Pixhawk salvo los pines de salida auxiliares que se controlan mediante la FMU.

Para simplificar la explicación de la estructura software, se ha realizado una serie de diagramas, donde cada cuadrado representa una aplicación y las líneas que los unen son los tópicos (buses de datos) cuyo sentido representa la trayectoria que sigue el mensaje desde el publicador hasta el suscriptor. Los cuadros azules son aplicaciones de PX4FMU, los cuadros verdes son drivers, es decir, controladores de los sensores y el resto de dispositivos externos al sistema Pixhawk, y los cuadros amarillos son los módulos pertenecientes al firmware de PX4IO.

En el diagrama de la página siguiente y los consecutivos se incluyen los tópicos más representativos, pero no todos los involucrados en el proceso de control de UAV; es un sistema muy complejo que no se puede representar correctamente en un único esquema. Se han omitido una serie de tópicos cuya principal función es informar del estado del UAV mediante telemetría, es decir, tópicos a los que se subscribe la aplicación *mavlink*.

En el ANEXO A se incluye una tabla con la relación completa de tópicos involucrados así como las aplicaciones que publican y se subscriben a cada uno de ellos. Además, en el ANEXO B se puede encontrar una explicación detallada de los datos que transporta cada tópico.

3.6.2.1. Módulos

En primer lugar se va a explicar las funciones de cada una de las aplicaciones que intervienen en el software de control de un multicóptero. Posteriormente se detallará, con diagramas más específicos como funciona el software para distintos modos de vuelo.

sensors

La aplicación *sensors* tiene como función principal obtener y agrupar los datos de todos los sensores, tanto los incluidos en el hardware de Pixhawk como los externos. Además se encarga de transformar los valores de los canales de entrada de RC en valores normalizados de roll, pitch, yaw y throttle, e interpretar si se activa algún botón de cambio de modo a través del mando RC.

mavlink

La aplicación *mavlink* implementa el protocolo de comunicación para la telemetría, enviando y recibiendo mensajes con formato MAVLink a través de un puerto serie. La versión de software que se ha utilizado envía los mensajes:

- mavlink_msg_heartbeat
- mavlink_msg_sys_status
- mavlink_msg_highres_imu
- mavlink_msg_attitude
- mavlink_msg_vfr_hud
- mavlink_msg_gps_raw_int
- mavlink_msg_global_position_int
- mavlink_msg_local_position_ned
- mavlink_msg_gps_global_origin
- mavlink_msg_global_position_setpoint_int
- mavlink_msg_roll_pitch_yaw_thrust_setpoint
- mavlink_msg_rc_channels_raw
- mavlink_msg_named_values
- mavlink_msg_distance_sensor

Además, *mavlink* es capaz de decodificar los mensajes:

- mavlink_msg_command_long
- mavlink_msg_optical_flow
- mavlink_msg_set_mode
- mavlink_msg_vicon_position_estimate
- mavlink_msg_quad_swarm_roll_pitch_yaw_thrust
- mavlink_msg_radio_status

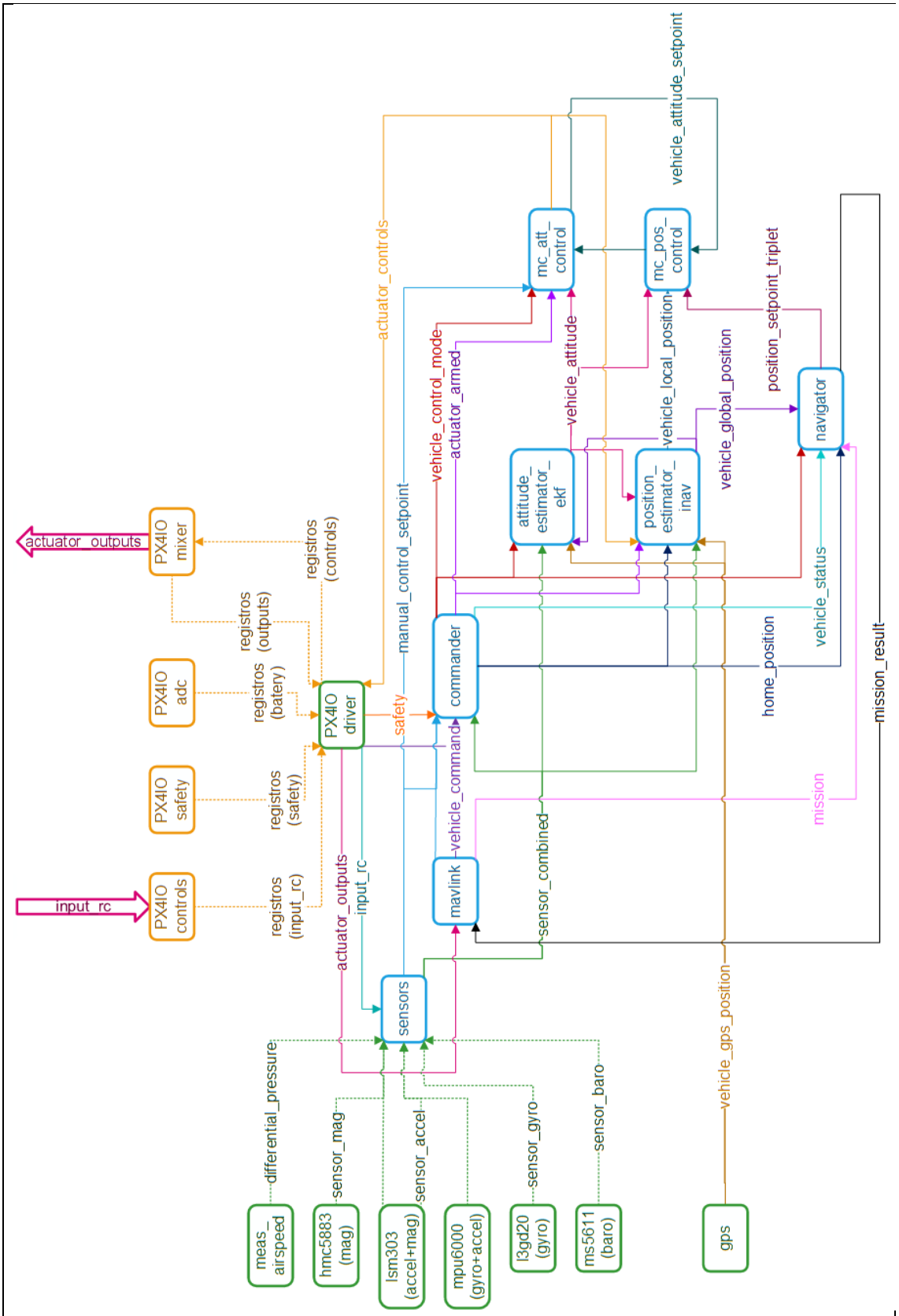


Figura 3-20. Arquitectura firmware de Pixhawk

- mavlink_msg_manual_control
- mavlink_msg_mission_ack
- mavlink_msg_mission_set_current
- mavlink_msg_mission_request_list
- mavlink_msg_mission_request
- mavlink_msg_mission_count
- mavlink_msg_mission_item
- mavlink_msg_mission_clear_all
- mavlink_msg_param_request_list
- mavlink_msg_param_set
- mavlink_msg_param_request_read

commander

La aplicación *commander* implementa la máquina de estados interna del sistema. Es la encargada de controlar que el resto de aplicaciones sólo realicen acciones que tengan sentido en el contexto actual del sistema. Comprueba continuamente en qué situación se encuentra el sistema y gestiona los cambios de modo, tanto si el cambio es requerido desde la estación base como si se debe realizar por motivos de seguridad. Además, informa al resto de aplicaciones de los cambios realizados.

Para informar al resto del sistema de la situación del cuadricóptero, la aplicación *commander* cuenta con varios grupos de estados:

- **Armed State**: Estado de armado del sistema, puede ser: INIT, STANDBY, STANDBY_ERROR, ARMED_ERROR.
- **Main State**: Estado principal, indica lo que el usuario quiere que suceda. Se indica mediante interacción con un mando RC o mediante un comando de control desde la estación base. Los valores que puede tomar el estado principal coinciden con los modos de vuelo: MANUAL, ALTCTL, POSCTL, AUTO_MISSION, AUTO_RTL, AUTO_LOITER, ACRO, OFFBOARD.
- **Navigation State**: Estado de navegación, indica lo que es UAV tiene que hacer, cambia en función del estado principal y de diversas directrices de seguridad, como el estado de la comunicación o las posibilidades de localización. Puede tomar los siguientes valores: MANUAL, ALTCTL, POSCTL, AUTO_MISSION, AUTO_RTL, AUTO_LOITER, ACRO, LAND, DESCEND, TERMINATION, OFFBOARD
- Además, cuenta con una bandera que se activa para indicar que el UAV se encuentra en estado de prueba de fallos (failsafe)

La máquina de estados, con los estados que se utilizan en la versión actual de firmware, se puede ver representada en el siguiente diagrama:

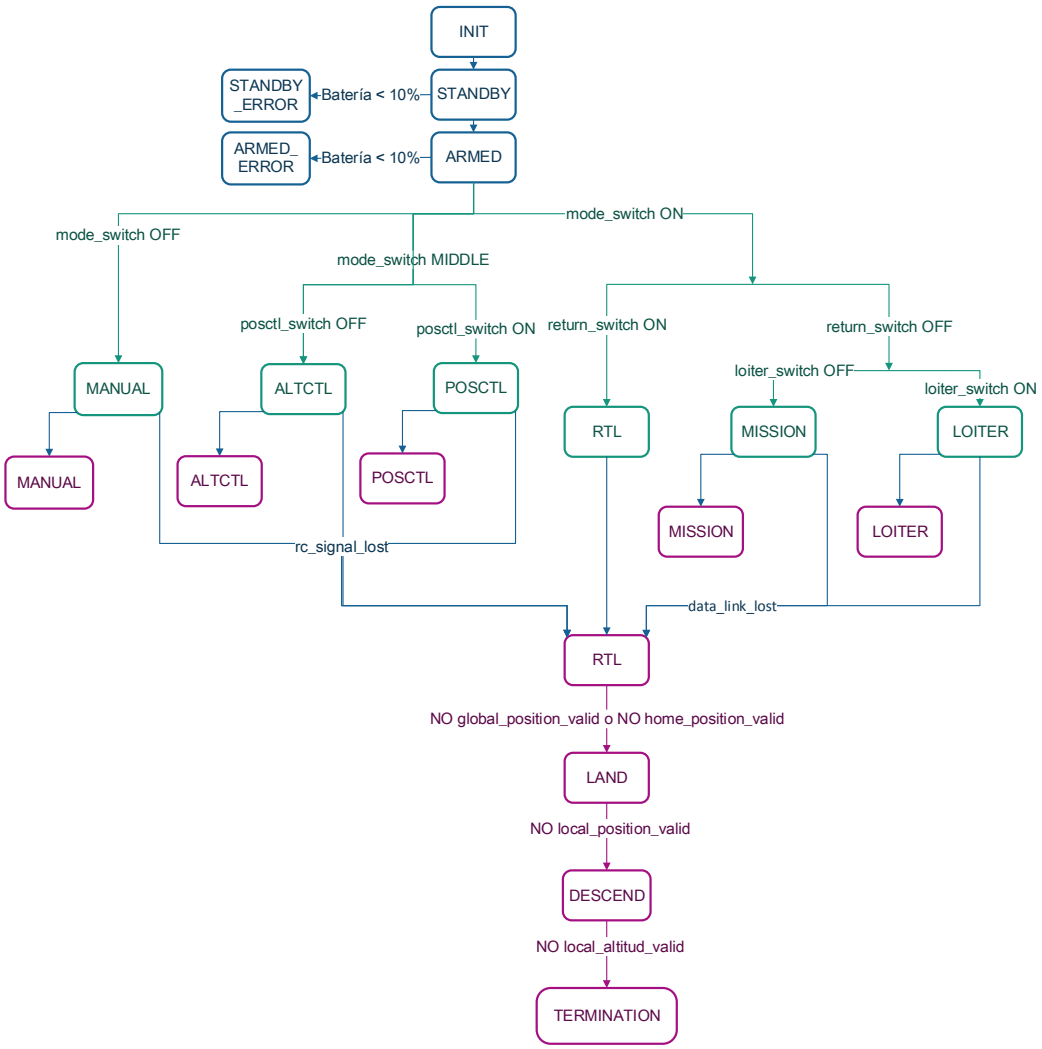


Figura 3-21. Máquina de estados

Cuando se inicia el sistema, el controlador de vuelo entra en estado INIT, tras iniciar y comprobar el correcto funcionamiento de todos los sensores pasa a estado STANDBY si hay suficiente batería o a estado STANDBY_ERROR si la carga de la batería es inferior al 10%.

El sistema permanece en este estado hasta que se ejecute la orden de armado. Para ello hay que presionar el botón safety durante unos segundos y a continuación, con el mando RC, mantener el cursor de throttle a 0 y llevar a la derecha del todo el cursor de yaw durante un par de segundos. El estado del sistema pasa a ser ARMED, cuando se encuentra armado y con suficiente batería o ARMED_ERROR cuando la batería es inferior al 10% del total de su carga.

Como se ha indicado antes, el estado principal coincide con los modos de vuelo del UAV. Los cambios de modo se controlan mediante 4 interruptores: *mode_switch* (3

posiciones), *posctl_switch* (2 posiciones), *return_switch* (2 posiciones), *loiter_switch* (2 posiciones), por lo tanto, lo mismo ocurre con el estado principal.

Al pasar el estado de armado a ARMED, el estado principal comienza siendo MANUAL, si no se indica lo contrario mediante el mando RC. Es decir, si el interruptor de modo se encuentra en posición OFF.

Si el interruptor de modo se sitúa en la posición media, el estado principal cambia igual que el modo de vuelo, y pasa a ser ALTCTL si es interruptor *posctl_switc*h está en posición OFF, o POSCTL si el interruptor está en posición ON.

Por último, si el interruptor de modo se encuentra en posición ON, el estado principal del sistema pasa a ser AUTO_MISSION si los dos interruptores restantes se encuentran en posición OFF, o bien AUTO_RTL o AUTO_LOITER en función del interruptor que pase a posición ON.

Si el UAV no se encuentra en estado de prueba de fallos (failsafe), es decir, todo funciona correctamente, el estado de navegación es siempre el mismo que el estado principal. El UAV funciona como el usuario indica. Por el contrario, si se producen errores en la comunicación el UAV entra en estado a prueba de fallos y el estado de navegación se modifica en función de los datos que disponga el sistema.

Si estando en alguno de los modos manuales se pierde la comunicación con el mando RC, el sistema tratará de pasar a estado de navegación RTL, si dispone tanto de señal GPS para calcular la posición actual como de las coordenadas correctas del punto de despegue. En caso contrario, lo que tratará de hacer es aterrizar de forma controlada si dispone de información sobre la posición local o de descender si solo cuenta con información sobre la altitud. Si ninguno de los estados anteriores es posible, el UAV pasa a estado de navegación TERMINATED y apaga los motores.

Si el estado principal es alguno de los estados autónomos, se pasa a modo a prueba de fallos y por consiguiente se cambia el estado de navegación si se pierde el enlace de telemetría.

Pixhawk considera que se ha perdido el enlace de RC cuando pasan más de 0.5 s sin que se publique ningún dato en el tópico ***manual_control_setpoint***. El control del enlace de telemetría se realiza sólo si se reciben mensajes "mavlink_msg_heartbeat" desde la estación base. Si pasan más de 5 s sin recibir ningún mensaje de prueba de vida, se considera que se ha perdido el enlace de telemetría.

attitude_estimator_ekf

La aplicación *attitude_estimator_ekf* hace una estimación de la orientación del UAV, entendiendo como orientación los valores de yaw, pitch y roll, a partir de los datos leídos de los sensores. Para realizar esta estimación se utilizan filtros Kalman extendidos.

mc_attitude_control

Obtiene las señales de control de los motores a partir de la combinación de la orientación actual con la orientación deseada. Para ello utiliza tres controladores PID, dos idénticos para el control de la variación de roll y pitch, ya que el cuadricóptero cuenta con una estructura simétrica. Y un tercero para controlar la velocidad de variación del movimiento de rotación, yaw. Estos controladores PID trabajan como un lazo cerrado interno.

Además, se utiliza un controlador proporcional que trabaja como lazo exterior para corregir los ángulos de forma proporcional.

El siguiente diagrama refleja el funcionamiento del controlador de cada uno de los ángulos de orientación de un cuadricóptero, roll, pitch y yaw:

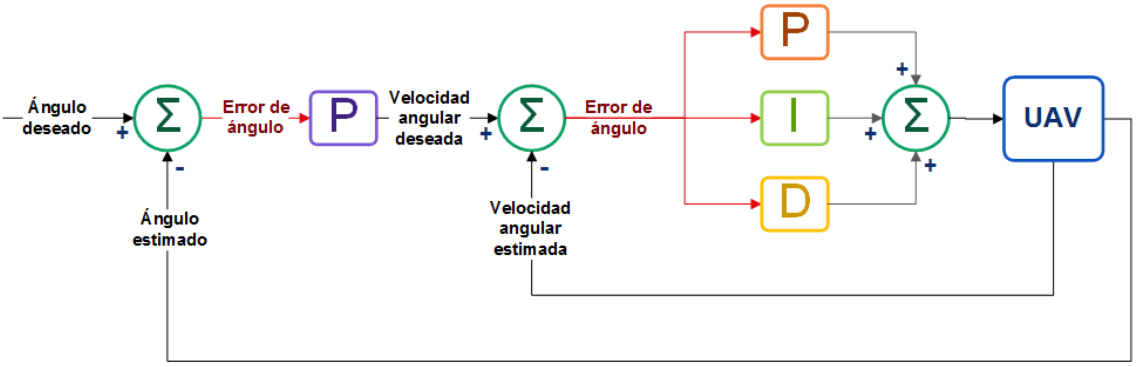


Figura 3-22. Controlador PID del cuadricóptero

Las constantes de cada uno de los controladores, proporcional, integral y derivativo son configuradas como parámetros de Pixhawk.

position_estimator_inav

Estima la posición y la velocidad en 3D, y las expresa tanto en coordenadas locales como globales. Utiliza las medidas de todos los sensores además de la posición del UAV obtenida por el GPS.

mc_position_control

Utiliza la posición y velocidad locales calculadas por el estimador de posición y la siguiente posición requerida por el navegador para calcular la orientación (*roll*, *pitch*, *yaw*) y el *throttle* deseado. Igual que el controlador de orientación se basa en un controlador PID.

navigator

Es la aplicación que proporciona todas las funciones de navegación autónoma, controla la ejecución de las misiones, así como la función RTL (Return To Launch) tanto si es requerida por el usuario como si se produce por entrar el UAV en estado de prueba de fallos.

Drivers

Los drivers de uso general en un multicóptero son los controladores de los sensores más los que proporcionan la comunicación con PX4IO y con las salidas de PX4FMU. Los drivers publican en diversos tópicos los datos obtenidos de cada dispositivo periférico.

- **meas_airspeed:** Controlador del sensor de aire, proporciona una interfaz de comunicación i2c para recibir los datos de presión diferencial.
- **hmc5883:** Controlador del magnetómetro, proporciona una interfaz de comunicación i2c que permite recibir los datos del campo magnético del sensor que se encuentra junto al GPS.
- **lsm303:** Controlador del sensor LSM303, que incluye un acelerómetro y un magnetómetro. La interfaz de comunicación que se utiliza es SPI.
- **mpu6000:** Controlador del sensor mpu6000, que incluye un acelerómetro y un giroscopio. Se utiliza una interfaz de comunicación SPI.
- **l3gd20:** Controlador del giroscopio, proporciona una interfaz de comunicación SPI.
- **ms5611:** Controlador del barómetro, proporciona una interfaz de comunicación SPI que permite obtener la presión atmosférica.
- **gps:** Controlador del GPS utilizando una conexión por puerto serie (UART). Proporciona una comunicación bidireccional entre Pixhawk y el GPS utilizando el protocolo propio del dispositivo.
- **px4io (driver):** Controla la comunicación entre los módulos PX4FMU y PX4IO. Permite intercambiar datos como los controles de los motores, el estado de armado, el estado de la batería o los datos de control recibidos a través de RC. El driver se comunica con PX4IO utilizando un sistema de registros, mientras que utiliza la comunicación habitual mediante tópicos con PX4FMU.
- **px4fmu (driver):** Aunque no aparece en el diagrama, existe un controlador de los pines de conexión del módulo PX4FMU. Se pueden configurar de dos formas diferentes: mode_gpio o mode_pwm. El primer caso permite controlar los 6 pines como entradas/salidas digitales. El segundo modo configura los 6 pines como salidas PWM, este es el modo

que se utiliza por defecto, aunque en este proyecto se utilizan los pines de PX4IO para obtener el valor PWM de cada uno de los motores.

Librería systemlib

No es una aplicación, por lo tanto no aparece en el diagrama, pero es una parte del software muy importante ya que permite realizar llamadas al sistema a bajo nivel como si se tratase de llamadas a funciones. Entre las funciones se encuentran las de control de los mezcladores que permiten obtener las señales PWM para los motores en función del tipo de plataforma que se esté utilizando.

PX4IOfirmware

El módulo de PX4IO cuenta con una serie de bloques con una función específica.

- **mixer:** Controla las entradas y salidas al mezclador.
- **safety:** Controla el botón de seguridad que sirve para permitir armar o desarmar el sistema.
- **controls:** Recopila los datos de control de las posibles entradas de RC (ppm, S.Bus, DSM).

3.6.2.2. Software de control según el modo de vuelo

El sistema se comporta de diferente manera según el modo de vuelo que se esté implementando en cada momento. El sistema Pixhawk ofrece la posibilidad de configurar 7 modos de vuelo: MANUAL, ALTCTL, POSCTL, LOITER, RTL, MISSION y ACRO. El último de los modos no se va a utilizar en este proyecto, ya que, como su nombre indica, es un modo para realizar acrobacias, donde no se controla la estabilidad de la aeronave mediante el autopilot.

Sea cual sea el modo de vuelo, los valores de roll, pitch, yaw y throttle se obtienen y se transfieren a los motores de la misma forma. El siguiente diagrama muestra cómo se realiza esta acción:

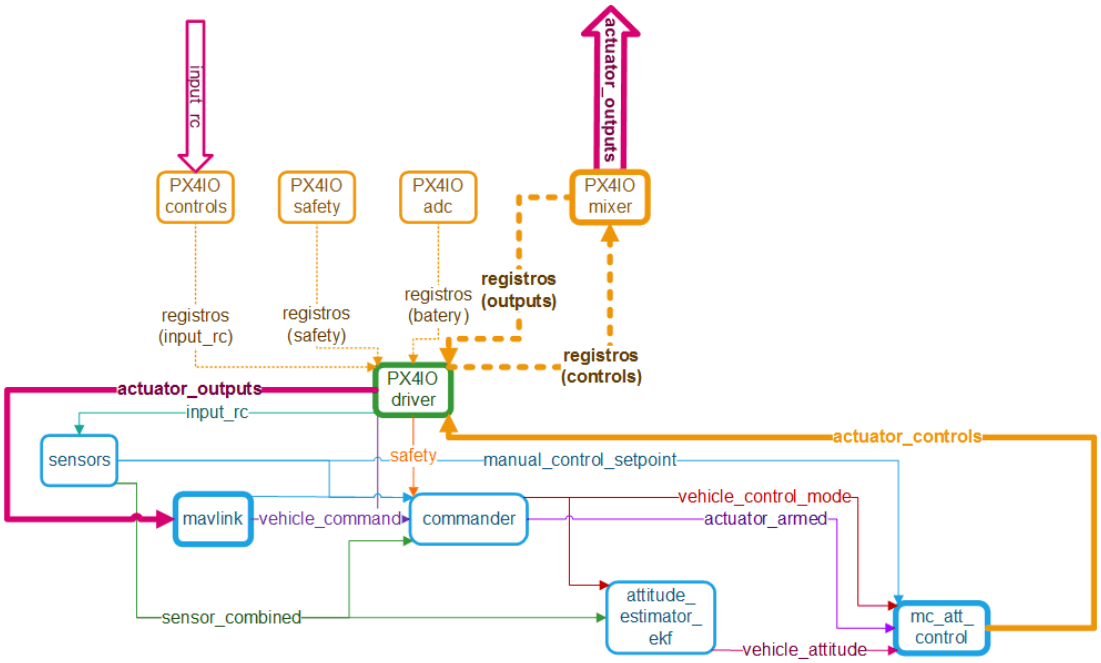


Figura 3-23. Diagrama de envío de datos a los motores

La aplicación *mc_att_control* siempre es la encargada de calcular los valores finales de orientación (roll, pitch y yaw) y el throttle, que publica en el tópico ***actuator_controls***. El driver *px4io* se subscribe al tópico ***actuator_controls*** para comunicar sus datos al módulo *mixer* del firmware PX4IO a través de los registros. El módulo *mixer* utiliza las funciones de mezclado de la librería *systemlib* para calcular los valores PWM de cada uno de los 4 motores del cuadricóptero. Es también este módulo el encargado de transmitir dichos valores a los pines de salida donde se conectarán los variadores de los motores. Además, escribe en el registro los valores de ancho de pulso PWM. El driver *px4io* lee estos valores y los publica en el tópico ***actuator_outputs*** para que estén disponibles como información para el resto de aplicaciones, por ejemplo, *mavlink* se subscribe a este tópico para después enviar la información a la estación base.

Modo MANUAL

En primer lugar se tiene la arquitectura software cuando el modo de vuelo es manual. En el diagrama se pueden ver las aplicaciones y los tópicos más importantes que intervienen en el proceso de envío de la información desde la estación base hasta los motores del cuadricóptero.

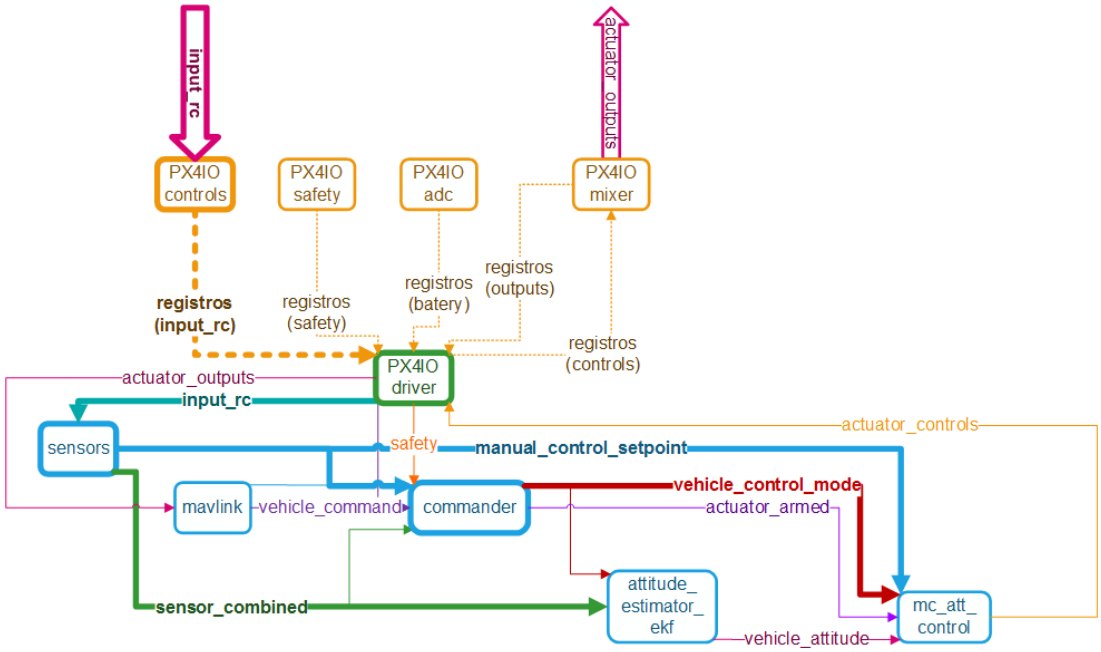


Figura 3-24. Diagrama del modo MANUAL con mando RC

Los comandos de control se pueden recibir de dos formas distintas: mediante un enlace de radiocontrol que es gestionado por el módulo PX4IO o mediante un enlace MAVLink.

En el primer caso se reciben señales de un enlace de radiocontrol las cuales pueden ser señales PPM o las correspondientes a los protocolos S.Bus o DSM.

El firmware PX4IO, en concreto el módulo *controls*, se encarga de procesar esas señales y convertirlas en valores que indican un ancho de pulso (normalmente valores entre 10000 y 20000 µs). Se pueden recibir hasta 18 canales de datos. Los datos son almacenados en registros. A partir de aquí, es el driver el que se encarga de leer esos datos y publicarlos en el tópico ***input_rc***. La aplicación *sensors* se subscribe a dicho tópico y realiza el mapeo de los datos de radiocontrol, es decir, asigna el valor de cada canal a una orden de control y hace el correspondiente escalado. La asignación y los valores que puede tomar cada parámetro se pueden ver en la siguiente tabla:

Tabla 3-4. Asignación de canales RC

Canal	Control	Valor
0	Throttle	[0, 1]
1	Roll	[-1,1]
2	Pitch	[-1,1]
3	Yaw	[-1,1]
4	Mode Switch	On, Middle, Off
5	Return Switch	On, Off
6	Posctl Switch	On, Off
7	Loiter Switch	On, Off
8	Offboard control Switch	On, Off
9	AcroSwitch	On, Off
10	Flaps	[-1,1]
11 - 15	Aux1 – Aux5	[-1,1]

La aplicación *sensors* publica los valores calculados en el tópico ***manual_control_setpoint***. La aplicación *commander* se encarga de seleccionar el modo de vuelo de acuerdo con las posiciones de los interruptores que obtiene suscribiéndose al tópico ***manual_control_setpoint***. El modo de vuelo lo publica en el tópico ***vehicle_control_mode*** para que sea accesible por el resto de aplicaciones.

La otra forma que tiene Pixhawk de recibir las órdenes de control es a través de un enlace MAVLink. La aplicación *mavlink* decodifica mensajes de tipo "mavlink_msg_manual_control" para obtener los valores de throttle, roll, pitch y yaw que publica en el tópico ***manual_control_setpoint***. Además, *mavlink* decodifica mensajes tipo "mavlink_msg_set_mode" donde se indica el modo de vuelo deseado. Esta intención de cambio de modo de vuelo es transmitido a la aplicación *commander* como un comando del tópico ***vehicle_command***. Como ocurría en el caso anterior, *commander* publica el modo de vuelo adquirido en el tópico ***vehicle_control_mode***.

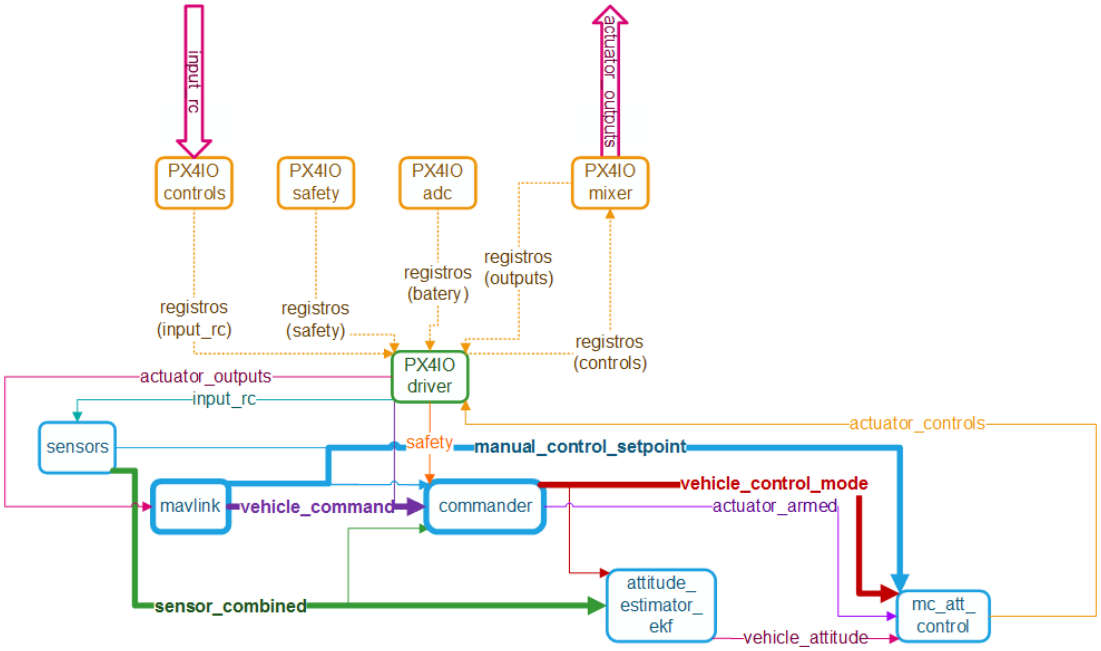


Figura 3-25. Diagrama del modo MANUAL sin mando RC

En ambos casos la aplicación *mc_att_control* se suscribe al tópico ***manual_control_setpoint*** para acceder a los valores de throttle, yaw, pitch y roll, que interpreta como valores de orientación deseados. Además tiene en cuenta el modo de vuelo indicado en ***vehicle_control_mode*** y comprueba si el sistema está armado suscribiéndose al tópico ***actuator_armed***. Aunque el modo de vuelo sea manual, los valores deseados de orientación no son pasados directamente a los motores, sino que se tiene en cuenta la orientación actual para evitar cambios demasiado bruscos que pueden provocar la inestabilidad del cuadricóptero. Es aquí donde interviene la aplicación *attitude_estimator_ekf* que publica los valores actuales de roll, pitch y yaw en el tópico ***vehicle_attitude*** para que estén disponibles por *mc_att_control*. Para estimar la orientación actual son necesarios los datos recibidos de todos los sensores por lo que *attitude_estimator_ekf* se subscribe al tópico ***sensor_combined***. En este tópico publica la aplicación *sensors* una recopilación de todas las medidas de los sensores conectados a Pixhawk.

Una vez recopilados todos los datos necesarios: orientación actual, orientación deseada y modo de vuelo, la aplicación *mc_att_control* calcula los valores de orientación y throttle que enviar a los motores y los publica en el tópico ***actuator_controls***.

Modo ALTCTL

Pixhawk incluye modos de vuelo semiautónomos, por ejemplo ofrece la posibilidad de controlar la altitud o la posición: modos ALTCTL y POSCTL.

En el modo ALTCTL los controles de roll, pitch y yaw funcionan exactamente igual que en modo manual, mientras que el valor de throttle se mantiene en un valor constante.

Modo POSCTL

Cuando no se indica ningún valor de roll y pitch, la posición permanece fija, mientras que yaw permite que el cuadricóptero gire sobre sí mismo. El control del throttle actúa igual que en el modo ALTCTL.

Para lograr mantener en una posición fija el cuadricóptero, son necesarios tanto el estimador de posición como el controlador de la misma, y por lo tanto, es requisito imprescindible que el cuadricóptero cuente con un dispositivo GPS.

El siguiente diagrama resalta las aplicaciones y tópicos más relevantes cuando se vuela en modo POSCTL:

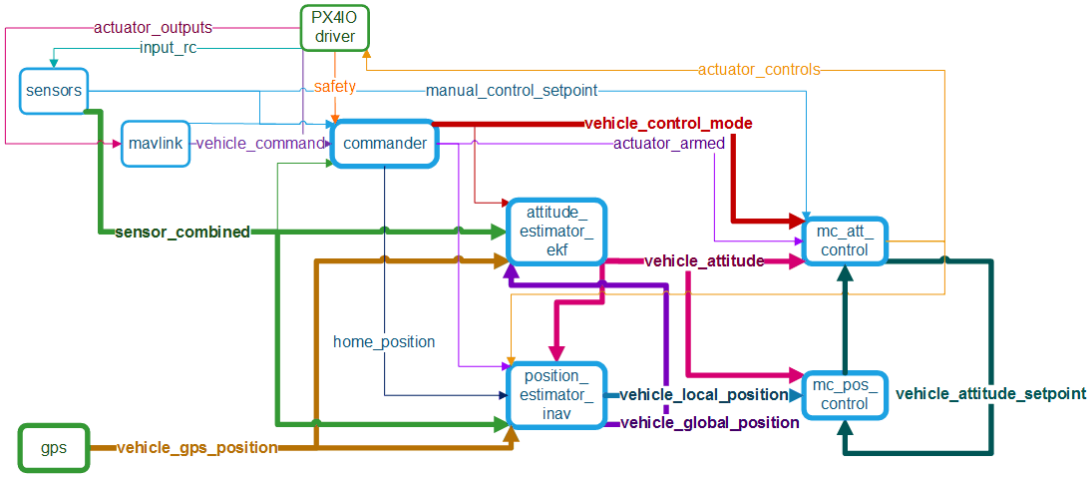


Figura 3-26. Diagrama del modo POSCTL

El driver del GPS es el encargado de publicar la información sobre la posición en coordenadas WGS84: longitud, latitud y altitud, y la velocidad en el tópico ***vehicle_gps_position***. Con la información obtenida al subscribirse a este tópico más los datos de los sensores obtenidos del tópico ***sensor_combined*** y la orientación actual (tópico ***vehicle_attitude***), la aplicación *position_estimator_inav* calcula la posición y la velocidad del UAV tanto en coordenadas locales como globales. Estos datos, que indican la situación actual del cuadricóptero son publicados en sendos tópicos: ***vehicle_local_position*** y ***vehicle_global_position***. La posición global es utilizada por la aplicación *attitude_estimator_ekf* para calcular la orientación actual de forma más precisa. Por otro lado, la aplicación *mc_pos_control* se subscribe al tópico ***vehicle_local_position***. Combinando la posición, velocidad y orientación actuales, calcula los valores de roll, pitch, yaw y throttle requeridos para mantener la posición constante. A diferencia del modo manual, donde los valores de roll, pitch, yaw y throttle deseados se obtenían únicamente del tópico ***manual_control_setpoint***, en este caso la aplicación *mc_att_control* se subscribe también al tópico ***vehicle_attitude_setpoint*** y toma de él los valores de requeridos de throttle y de roll y pitch cuando estos son próximos a cero en el tópico ***manual_control_setpoint***.

Modo LOITER

Se trata del primero de los modos de vuelo autónomo, en los que no es necesario realizar ningún control manualmente. Cuando la aeronave es un cuadricóptero, permite que éste permanezca suspendido en el aire en la misma posición en la que se encuentre al iniciar el modo de vuelo y a una altitud constante. El cuadricóptero puede pasar a modo LOITER por orden de la estación base o como medida de seguridad gestionado por la aplicación *commander*.

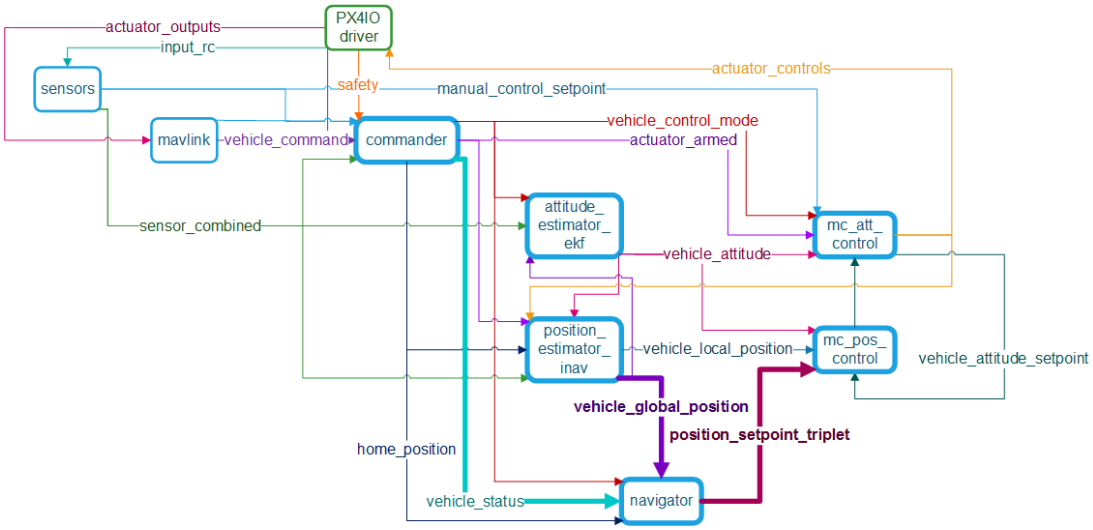


Figura 3-27. Diagrama del modo LOITER

El controlador de orientación (*mc_att_control*) funciona igual que en modo POSCTL, con la única diferencia de que en este caso no tiene en cuenta los datos obtenidos del tópico ***manual_control_setpoint***. La principal diferencia se encuentra en la aparición de una nueva aplicación: *navigator*.

Este es el caso más simple de navegación. La aplicación *navigator* se subscribe a los tópicos ***vehicle_status*** y ***vehicle_global_position***. Del primero de ellos obtiene el estado de navegación en el que se encuentra al UAV; si éste es LOITER publica en el tópico ***position_setpoint_triplet*** las coordenadas leídas del tópico ***vehicle_global_position***.

Cuando el cuadricóptero se encuentra en cualquiera de los modos de vuelo autónomo: LOITER, RTL o MISSION, el controlador de posición (*mc_pos_control*) calcula la orientación deseada teniendo en cuenta no sólo la posición actual como ocurría en el modo POSCTL sino también la siguiente posición requerida. La posición requerida la obtiene al subscribirse al tópico ***position_setpoint_triplet***.

Modo RTL

El modo RTL (Return To Launch) permite al cuadricóptero volver a la posición donde inició el vuelo.

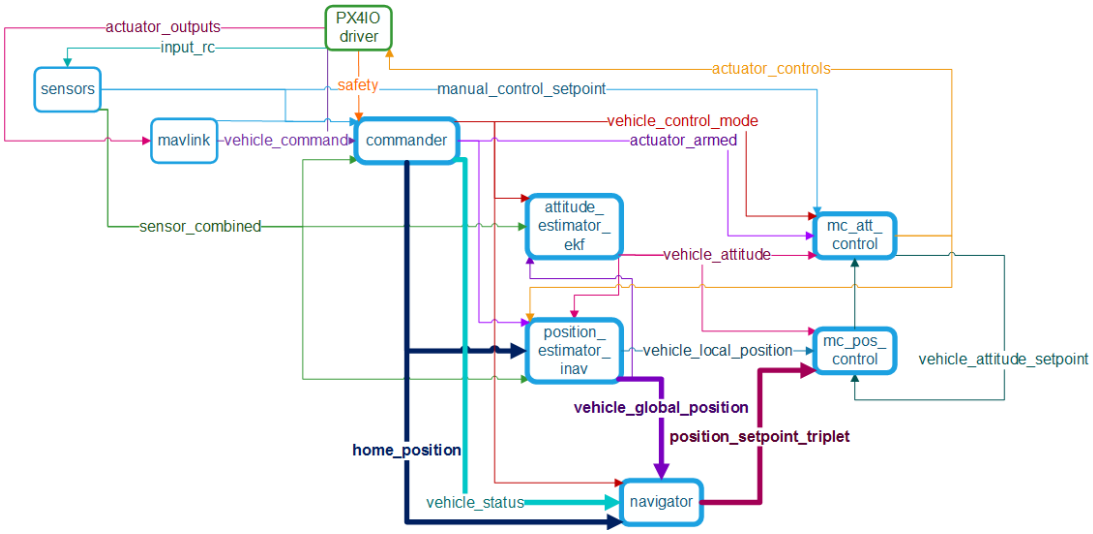


Figura 3-28. Diagrama del modo RTL

Si el UAV cuenta con GPS, en el momento de despegar se obtienen las coordenadas en las que se encuentra y la aplicación *commander* las publica en el tópico ***home_position***. Si se quiere que el cuadricóptero vuelva al origen, bien por orden desde la estación base o bien por algún motivo de seguridad gestionado por *commander*, la aplicación *navigator* se encarga de calcular la ruta. En primer lugar se selecciona la altitud de vuelo, que puede ser el valor actual o la suma de la altitud de la posición de origen más un valor constante de altitud de RTL fijado como parámetro, en caso de que el valor actual sea menor que éste. Además, fija como objetivo los valores de longitud y latitud de la posición de origen. Todos los datos los publica en el tópico ***position_setpoint_triplet***. El controlador de posición calcula el throttle necesario para mantener la altitud constante en todo momento y los valores de roll y pitch para que el cuadricóptero se dirija hacia las coordenadas indicadas en ***position_setpoint_triplet***.

Cuando el cuadricóptero alcanza la posición de origen aterriza automáticamente, controlando la velocidad de descenso por la aplicación ***mc_pos_control***.

Modo MISSION

El modo de vuelo MISSION se inicia cuando el cuadricóptero vuela siguiendo una serie de waypoints que pueden ser enviados desde la estación base en tiempo real o estar almacenados en la memoria del UAV.

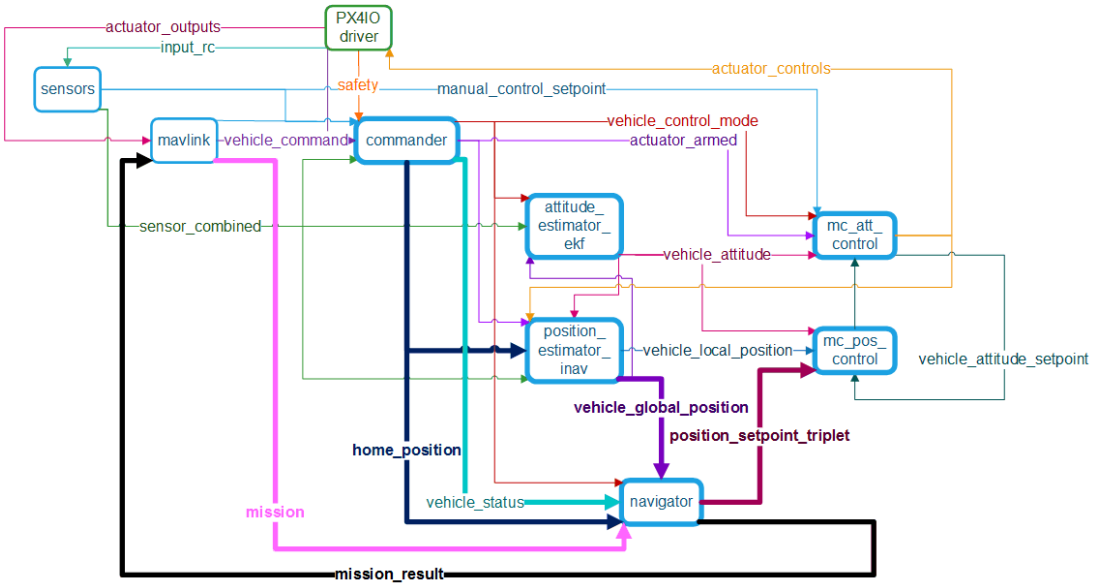


Figura 3-29. Diagrama del modo MISSION

Los waypoints se envían desde la estación base a través de mensajes MAVLink como comandos de navegación y son decodificados por la aplicación *mavlink*, que publica los datos recibidos en el tópico ***mission***. La aplicación *navigator* se subscribe a este tópico y publica las coordenadas del siguiente waypoint que se quiera alcanzar en el tópico ***position_setpoint_triplet***.

Si el cuadricóptero alcanza las coordenadas indicadas en ***position_setpoint_triplet*** y no ha recibido el siguiente waypoint hacia dónde dirigirse, entra el modo LOITER y permanece suspendido en el aire en esa misma posición hasta que reciba un nuevo waypoint o bien se cambie el modo de vuelo de forma manual.

La aplicación *navigator*, además, informa si la misión se ha realizado con éxito publicando en el tópico ***mission_result***, al que se subscribe *mavlink* para poder informar a la estación base.

3.7. Estación base

Para la recepción de la telemetría y el control del UAV en modo automático se va a utilizar el software QGroundControl. Se trata de una estación base que se comunica mediante el envío de mensajes MAVLink. Como el resto del proyecto, el software es de libre desarrollo, con acceso al código fuente, lo que permite realizar modificaciones en caso de que sea necesario.

Existen una serie de perspectivas dentro de la interfaz del programa, todas ellas configurables.

3.7.1. *Perspectivas de visualización*

3.7.1.1. Flight

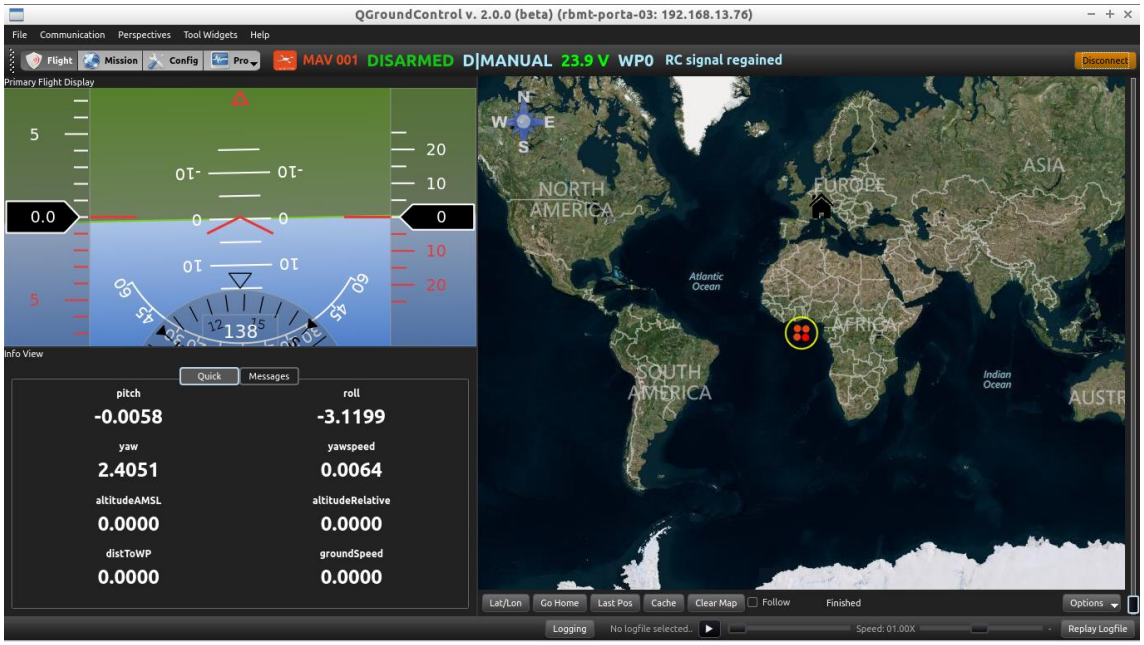


Figura 3-30. QGroundControl – Vista Flight

Se utiliza para ver los valores más representativos del estado del UAV. A la izquierda se puede ver un display con la orientación del UAV e información numérica sobre cualquiera de los datos de telemetría que se reciben. A la derecha un mapa con la situación del cuadricóptero cuando este tiene señal GPS.

3.7.1.2. Mission

Figura 3-31. QGroundControl – Vista Mission

La parte principal es un mapa de la zona donde se encuentre el UAV, ubicado mediante la señal GPS. Pinchando con el botón derecho del ratón sobre un punto de la pantalla se pueden crear waypoints o modificar la posición de retorno.

3.7.1.3. Config

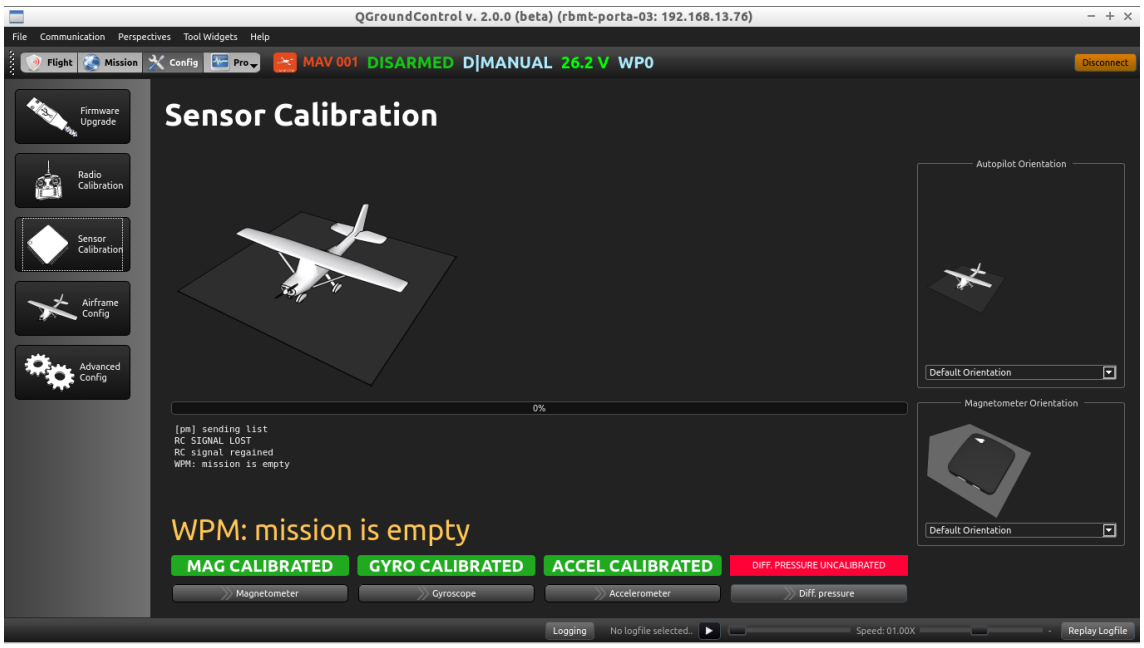


Figura 3-32. QGroundControl – Vista Config

Ventana de configuración. Permite calibrar los sensores de forma guiada. Actualizar el firmware y cambiar la configuración de la estructura del UAV. Además permite obtener los parámetros del sistema y cambiarlos en caso de que sea necesario, por ejemplo para la configuración del PID del controlador de estabilidad.

3.7.1.4. PRO:

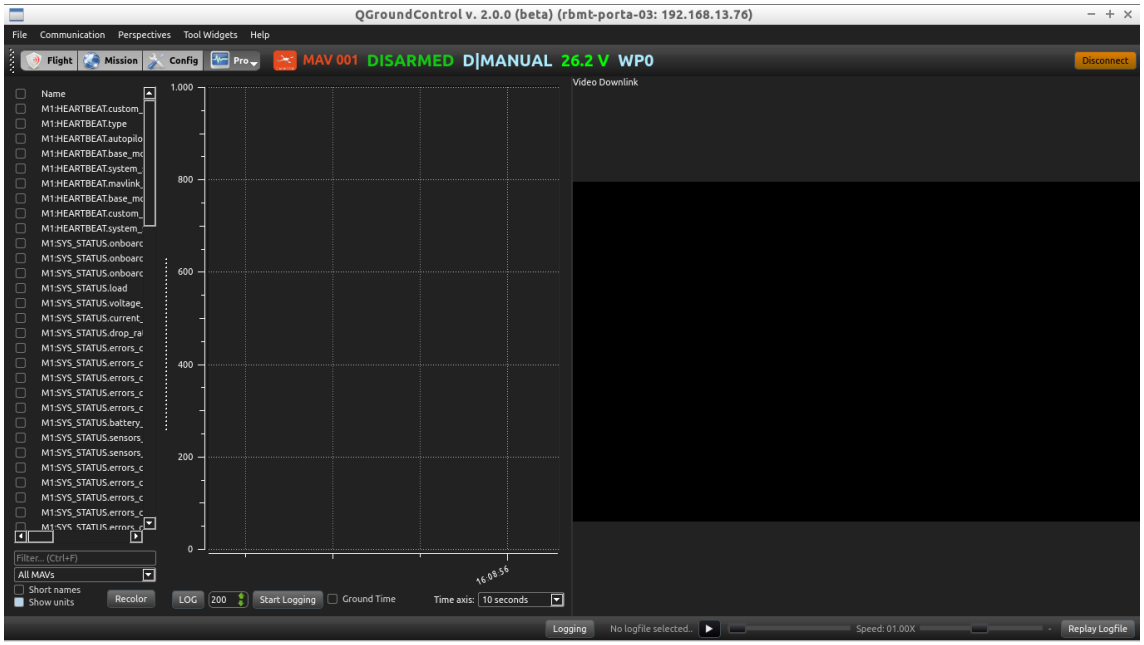


Figura 3-33. QGroundControl – Vista Plot

Permite visualizar en tiempo real gráficos de cualquiera de las variables del sistema. Además es posible importar los datos para analizarlos con posterioridad.

3.7.2. *Configuración del enlace de comunicación*

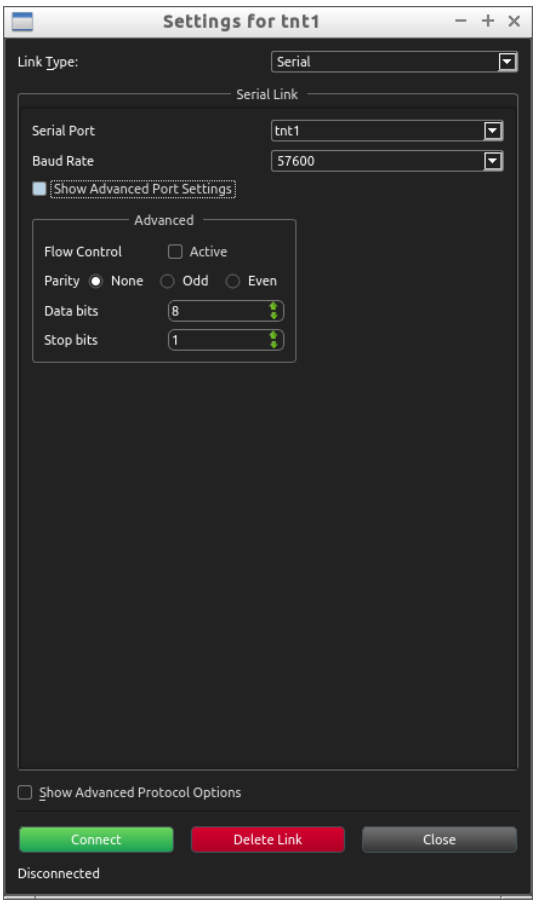


Figura 3-34. QGroundControl – Configuración del puerto de comunicaciones

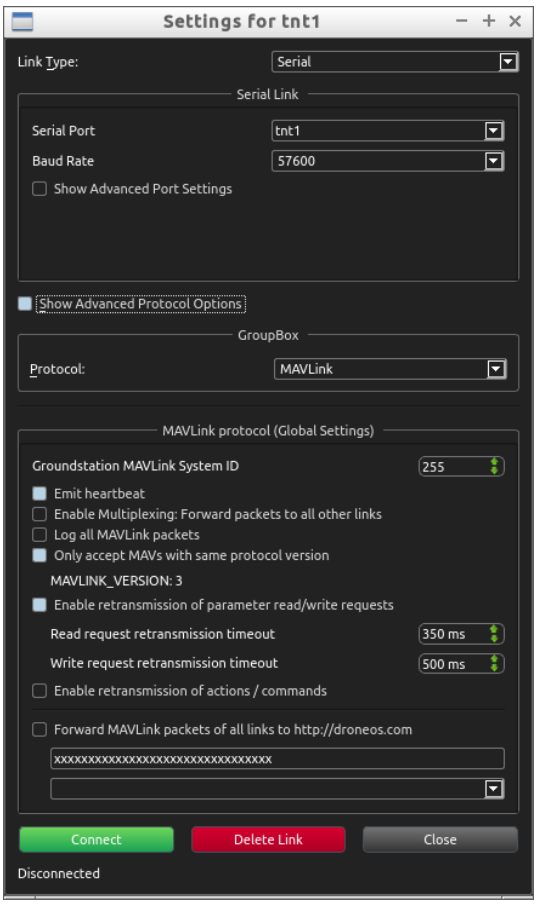


Figura 3-35. QGroundControl – Configuración del protocolo de comunicación

Se puede configurar el enlace de comunicación seleccionando los ajustes del puerto de comunicación, donde se puede elegir el tipo, velocidad, paridad, etc. Y se pueden configurar las opciones de protocolo, donde se seleccionan las principales características del protocolo MAVLink; se puede forzar a la estación base a enviar mensajes de señal de vida (heartbeat), para que el UAV pueda comprobar que tiene una estación base escuchando y se puede realizar un registro de los mensajes MAVLink recibidos. Este registro (log) se puede reproducir con posterioridad como si de una simulación se tratara.

4 Desarrollo

4.1. Introducción

En este apartado se va a explicar los cambios realizados en el firmware de Pixhawk y el programa implementado en la estación base para lograr el control del UAV enviando las órdenes a través de un enlace MAVLink. Además, se detallan las herramientas utilizadas.

4.2. Entorno de desarrollo

4.2.1. *Instalación*

Para estudiar y modificar el firmware de Pixhawk se ha utilizado el entorno de desarrollo de software (IDE) Eclipse. Para comprobar el correcto funcionamiento de las aplicaciones es posible el acceso al intérprete de comandos, para lo que es necesario utilizar un cable conversor serie a USB con chip FTDI.

En el ANEXO C se explican los pasos a seguir para configurar el entorno de desarrollo. En primer lugar se indica cómo configurar el sistema para poder desarrollar sobre el firmware de Pixhawk, tal y cómo se explica en el manual de su página web https://pixhawk.org/dev/px4_quickstart, pero adaptando los parámetros a este proyecto.

Además se especifican los pasos a seguir para obtener la versión del software a partir de la cual se ha trabajado en este proyecto.

A continuación se indican las dependencias necesarias para poder hacer uso de las aplicaciones de la estación base, al igual que la manera de instalar las versiones utilizadas.

4.2.2. *Conectar con Pixhawk mediante línea de comandos*

4.2.2.1. Cableado

La conexión de Pixhawk es un puerto UART y se conecta al ordenador mediante puerto USB. Por lo tanto es necesario un adaptador entre los dos tipos de conexiones. Los adaptadores más utilizados cuentan con chip FTDI.

En este caso se ha seleccionado el siguiente dispositivo:

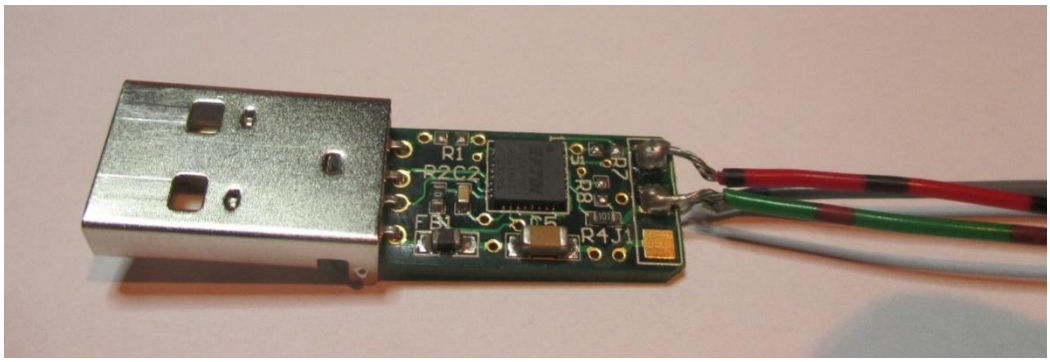


Figura 4-1. Conversor serie a USB

Para conectar con Pixhawk ha sido necesario soldar únicamente 3 cables entre el adaptador y el conector de Pixhawk:

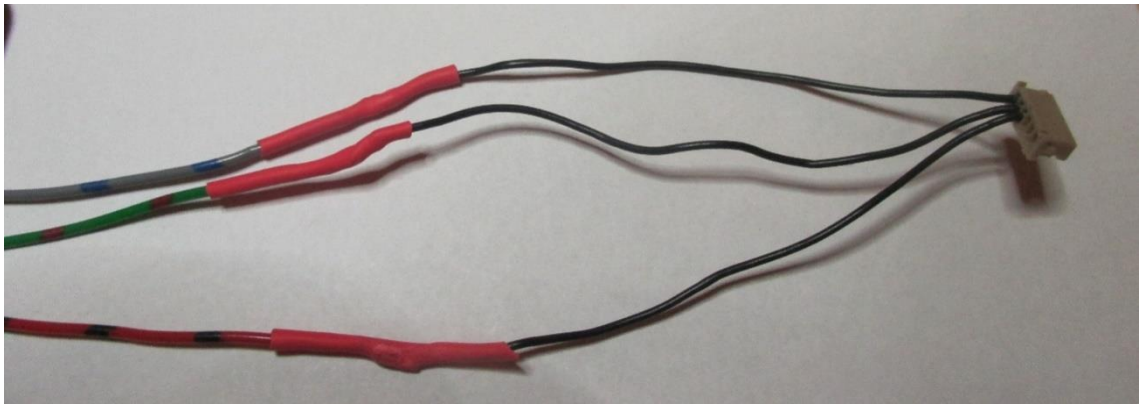


Figura 4-2. Conexión serie a PIxhawk

Las conexiones son:

Tabla 4-1. Conexión serie Pixhawk

PIXHAWK		Conversor FTDI
GND	→	GND
TX	→	RX
RX	←	TX

4.2.2.2. Programas (Hercules/ Screen)

Se puede utilizar cualquier emulador de terminales para llevar a cabo la comunicación con la línea de comandos de NuttX instalado en Pixhawk. En este proyecto se han utilizado dos programas dependiendo del sistema operativo usado:

- **Hercules:** Se utiliza para conectar con un ordenador con sistema operativo Windows. La configuración de la conexión se puede ver en la siguiente imagen.

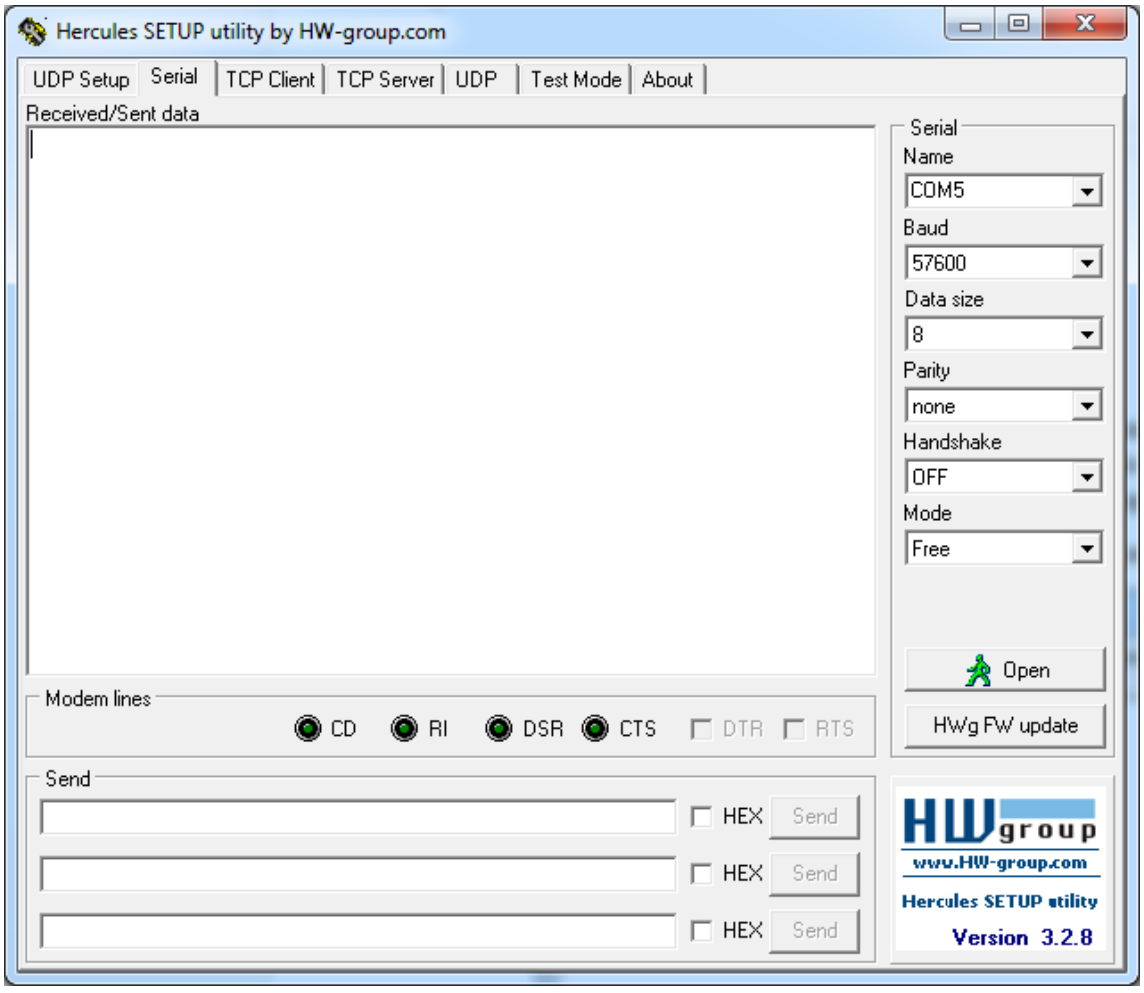


Figura 4-3. Configuración Hercules

- **Screen:** Se ha utilizado para conectar con un ordenador con sistema operativo Linux.

Para iniciar una conexión serie con Pixhawk utilizando *screen* se utiliza el siguiente comando:

```
screen   /dev/ttyUSB1   57600   8N1
```

4.3. Modificaciones en el firmware del sistema de abordo

En el capítulo 3 se ha explicado el funcionamiento de todos los dispositivos. En esta sección se explicarán las modificaciones que se han realizado para conseguir adaptar esos dispositivos a las características que se espera que cumpla el UAV.

Aunque ya existe el modo de control del UAV a partir de mensajes MAVLink, sólo permitía controlar roll, pitch, yaw y throttle, y en este proyecto se quiere tener todas las funciones de los mandos RC, por lo tanto se ha modificado el sistema de control para adaptarlo a las necesidades del proyecto ARGOS. Además se ha creado una función que controla el pan/tilt de la cámara.

Por otro lado, para mejorar el control autónomo, se ha añadido un sonar al sistema. Ya existían drivers para obtener los datos del sonar de diferentes formas. La primera es utilizando la placa px4flow que lleva integrado un sonar además de ofrecer vuelo controlado por visión a partir de una pequeña cámara de vídeo. La otra opción es conectar el sonar a través de un puerto I2C. El sonar que se adquirió proporciona los datos mediante comunicación serie, además de PWM o datos analógicos. Por lo tanto se ha desarrollado el driver que permite obtener los datos del sonar mediante comunicación serie.

4.3.1. *Control modos de vuelo*

El sistema Pixhawk está configurado para utilizar un mando RC y por otro lado comunicar, a través de un enlace MAVLink, con un programa que gestione la telemetría en la estación base. De los 18 canales que se pueden recibir de RC, la aplicación *sensors* obtiene los datos de roll, pitch, yaw, throttle y las posiciones de diversos interruptores que se corresponden con los modos de vuelo.

Por otro lado, la aplicación *mavlink* se encarga de gestionar los mensajes que recibe el sistema. En concreto, uno de los mensajes que es capaz de decodificar es el mensaje "mavlink_msg_manual_control" que se utiliza para recibir los valores de roll, pitch, yaw y throttle deseados, además el mensaje MAVLink cuenta con un campo "buttons", que no se decodifica, pero cuya función es informar al sistema de a bordo de los botones que se han pulsado de un joystick. *Mavlink* también decodifica el mensajes

“mavlink_msg_set_mode”. Cuando la aplicación *mavlink* decodifica uno de estos mensajes obtiene el modo de vuelo al que se desea cambiar, para transmitir esta información a la aplicación *commander* (encargada de gestionar los modos de vuelo) utiliza el tópico ***vehicle_command***. La aplicación *commander* gestiona las transiciones a los estados correspondientes con los modos que se requieren. El problema es que en este caso sólo gestiona las transiciones entre los modos: MANUAL, ALTCTL, POSCTL y AUTO, sin permitir seleccionar el tipo de modo autónomo.

Por lo tanto, existen dos posibles formas de informar al sistema del modo de vuelo que se quiere utilizar desde la estación base, pero ninguno de ellos está implementado de la forma requerida para este proyecto.

Se ha pensado que la mejor forma de informar de un cambio de modo de vuelo es utilizar el mismo mensaje MAVLink que se utiliza para enviar los valores de los controles. Así, cuando la aplicación *mavlink* recibe un mensaje “mavlink_msg_manual_control”, realiza el mismo mapeo que la aplicación *sensors* cuando recibe las entradas de un mando RC.

En los mandos RC tradicionales se utilizan interruptores de 2 o 3 posiciones, pero el mando que se va a utilizar en este proyecto no tiene interruptores sino pulsadores, por lo tanto en lugar de enviar las posiciones de 4 interruptores se informa de qué botón se ha pulsado entre 6 posibilidades: manual, altctl, posctl, mission, loiter y return. Para no utilizar 6 bits de la variable “buttons” del mensaje “mavlink_msg_manual_control”, se ha asignado a cada botón un valor que se puede representar con 3 bits:

Tabla 4-2. Asignación modos de vuelo en mavlink_msg_manual_control

Modo de vuelo	Binario	Decimal
MANUAL	0000 0000 0000 1000	8
ALTCTL	0000 0000 0001 0000	16
POSCTL	0000 0000 0001 1000	24
MISSION	0000 0000 0010 0000	32
LOITER	0000 0000 0010 1000	40
RETURN	0000 0000 0011 0000	48

Para que el resto del sistema siga funcionando exactamente cómo funcionaría al recibir todas las órdenes de control de un mando RC, la aplicación *mavlink* hace un mapeo de los valores de los botones a los interruptores que le correspondería y así poder publicarlos en el tópico ***manual_control_setpoint***.

Se ha editado el fichero *src/modules/mavlink/mavlink_receiver.c* para añadir las siguientes líneas a la función **handle_message_manual_control()**:

```
buttons = man.buttons & ~0xFFC7;


switch (buttons) {
	case MAIN_SWITCH:
		manual.mode_switch = SWITCH_POS_OFF;
		manual.posctl_switch = SWITCH_POS_OFF;
		manual.return_switch = SWITCH_POS_OFF;
		manual.loiter_switch = SWITCH_POS_OFF;
		break;
	case ALTCTL_SWITCH:
		manual.mode_switch = SWITCH_POS_MIDDLE;
		manual.posctl_switch = SWITCH_POS_OFF;
		manual.return_switch = SWITCH_POS_OFF;
		manual.loiter_switch = SWITCH_POS_OFF;
		break;
	case POSCTL_SWITCH:
		manual.mode_switch = SWITCH_POS_MIDDLE;
		manual.posctl_switch = SWITCH_POS_ON;
		manual.return_switch = SWITCH_POS_OFF;
		manual.loiter_switch = SWITCH_POS_OFF;
		break;
	case AUTO_SWITCH:
		manual.mode_switch =  SWITCH_POS_ON;
		manual.posctl_switch = SWITCH_POS_OFF;
		manual.return_switch = SWITCH_POS_OFF;
		manual.loiter_switch = SWITCH_POS_OFF;
		break;
	case RETURN_SWITCH:
		manual.mode_switch =  SWITCH_POS_ON;
		manual.posctl_switch = SWITCH_POS_OFF;
		manual.return_switch = SWITCH_POS_ON;
		manual.loiter_switch = SWITCH_POS_OFF;
		break;
	case LOITER_SWITCH:
		manual.mode_switch =  SWITCH_POS_ON;
```

```
                manual.posctl_switch = SWITCH_POS_OFF;
                manual.return_switch = SWITCH_POS_OFF;
                manual.loiter_switch = SWITCH_POS_ON;
                break;
        default:
                manual.mode_switch =  SWITCH_POS_NONE;
                break;
        }
```

4.3.2. *Control servos pan/tilt*

Para transmitir la posición de los servos se quieren utilizar las variables aux2 y aux3 del tópico ***manual_control_setpoint***. Como el resto de valores que se publican en este tópico se envían a través del mensaje "mavlink_msg_manual_control", para no tener problemas de actualización del tópico y que se pierdan valores, se ha decidido enviar los valores de pan y tilt en el mismo mensaje MAVLink y así decodificarlo todo a la vez.

El problema es que en este caso, el mensaje MAVLink no cuenta con ningún campo que se pueda utilizar, por lo tanto se ha tenido que modificar la definición del mensaje para incluir las dos variables.

```
typedef struct__mavlink_manual_control_t
{
…
int16_t pan;
int16_t tilt;
…
} mavlink_manual_control_t;


. . .
```

La aplicación *mc_att_control* tiene que publicar los valores deseados de los servos en el tópico ***actuator_controls*** para que sean recibidos por el mezclador. A continuación se ve el código añadido en la línea 853 de la aplicación *mc_att_control*:

```
_actuators.control[5] = _manual_control_sp.aux2;
_actuators.control[6] = _manual_control_sp.aux3;
```

El mezclador del cuadricóptero utiliza los valores de las variables auxiliares para calcular las salidas de los pines 5 y 6. Se ha dejado libre el pin 4 para facilitar la conexión.

4.3.3. *Aplicación para pruebas*

Para poder comprobar si Pixhawk interpreta bien las órdenes que se le envían, se ha añadido una aplicación que escribe en la consola del sistema los valores de los tópicos:

- ***manual_control_setpoint*** → valores de orientación deseada.
- ***actuator_controls*** → valores de orientación estimada para compensar movimiento del cuadricóptero.
- ***actuator_outputs*** → valores PWM que se envía a cada motor

Se trata de una aplicación muy sencilla, que lo único que hace es subscribirse a los tópicos necesarios, sin intervenir en ningún momento en el resto del sistema.

En pruebas posteriores se ha ido ampliando esta aplicación para que muestre valores de otros tópicos, que se pueden seleccionar como argumentos a la hora de ejecutar la aplicación. En concreto se ha usado para comprobar los valores que el sonar estaba enviando.

Para poder ver la información que se escribe en consola hay que conectar el puerto serial5 de Pixhawk al ordenador como se explica en el apartado 4.2.2.

Para incluir la aplicación en el firmware se ha creado la carpeta *src/examples/print_controls* y en su interior los archivos print_controls.c, con el código de la aplicación, y el makefile de la aplicación, modules.mk, con las siguientes líneas:

```
MODULE_COMMAND        = print_controls
SRCS                  = print_controls.c
```

Además se ha editado el fichero *src/makefiles/config_px4fmu-v2_default.mk* para incluir la siguiente línea:

```
MODULES       += examples/print_controls
```

Al añadir esta línea en el makefile se indica que se tenga en cuenta el contenido del archivo print_control a la hora de compilar el firmware y hacer el build.

4.3.4. *Calibración manual de roll/pitch*

En la primera prueba de vuelo del cuadricóptero se observó que, aunque se había seguido el procedimiento de calibración de los sensores de forma adecuada, la calibración no era del todo correcta, ya que sin ser requerido se aplicaban valores de roll y pitch al control del cuadricóptero. Para poder compensar esos valores, la solución más sencilla es añadir un offset a los cálculos que realiza el estimador de orientación, es decir, la aplicación *attitude_estimator_ekf*. Para poder calcular dicho offset en tiempo de vuelo y así ver qué valor compensa los giros indeseados se utiliza uno de los joystick del mando.

En cuanto a los cambios dentro del firmware, en este caso se requiere:

1. **Decodificar un nuevo mensaje MAVLink**

La aplicación *mavlink*, es la encargada de decodificar todos los mensajes MAVLink, por lo tanto es en su fichero *src/modules/mavlink/mavlink_receiver.c* donde se añade la función que decodifica los valores de corrección de roll y pitch.

```
void

MavlinkReceiver::handle_message_attitude_correction(mavlink_message_t *msg)

{

	mavlink_attitude_t att;

	mavlink_msg_attitude_decode(msg, &att);


	uint64_t timestamp = hrt_absolute_time();


	structattitude_correction_s att_offset;

	memset(&att_offset, 0, sizeof(att_offset));


	att_offset.timestamp = timestamp;

	att_offset.roll = att.roll;

	att_offset.pitch = att.pitch;


	warnx("[MAVLINK] Attitude offset: %8.4f,%8.4f\n", (double)att_offset.roll,
(double)att_offset.pitch);


	if (_attitude_correction_pub< 0) {

		_attitude_correction_pub = orb_advertise(ORB_ID(attitude_correction),
&att_offset);


	} else {
```

```
                orb_publish(ORB_ID(attitude_correction), _attitude_correction_pub,
&att_offset);

        }

}
```

Esta función es invocada añadiendo las siguientes líneas:

```
case MAVLINK_MSG_ID_ATTITUDE:

        handle_message_attitude_correction(msg);

        break;
```

Y se declara en el fichero *src/modules/mavlink/mavlink_receiver.h*:

```
void handle_message_attitude_correction(mavlink_message_t *msg);
```

Se ha utilizado el mensaje "mavlink_msg_attitude" que transporta valores de los ángulos roll, pitch y yaw en radianes.

2. Añadir un tópico para transmitir los valores entre aplicaciones

Para añadir un tópico al firmware Pixhawk, en primer lugar se añade su definición, este caso en el archivo *src/modules/uORB/tópicos/attitude_correction.h*:

```
#ifndef TÓPICO_ATTITUDE_CORRECTION_H

#define TÓPICO_ATTITUDE_CORRECTION_H


#include<stdint.h>

#include"../uORB.h"


structattitude_correction_s {


        uint64_t      timestamp;

        float         roll;

        float         pitch;

};
```

```
ORB_DECLARE(attitude_correction);


#endif
```

Además hay que definir el nuevo tópico editando el archivo *src/modules/uORB/objects_common.cpp*:

```
#include "tópicos/attitude_correction.h"

ORB_DEFINE(attitude_correction, structattitude_correction_s);
```

3. Integrar en el resto del firmware

El problema es que la calibración no se realiza de forma correcta y por tanto los valores estimados de roll y pitch no son los adecuados para mantener al cuadricóptero en una posición fija mientras que se mantiene suspendido en el aire. Se debe modificar la aplicación *attitude_estimator_ekf*, ya que es la que calcula y publica los valores de orientación en un momento determinado.

El fichero *src/modules/attitude_estimator_ekf/attitude_estimator_ekf_main.cpp* se ha editado para que la aplicación se subscriba al nuevo tópico:

```
      #include <uORB/tópicos/attitude_correction.h>
```

```
      int sub_att_corr = orb_subscribe(ORB_ID(attitude_correction));
```

Además, dentro del bucle principal se comprueba si se actualizan los valores del offset y en ese caso se corrige los valores definitivos de roll y pitch:

```
bool att_offset_updated;

orb_check(sub_att_corr, &att_offset_updated);


if(att_offset_updated){

       orb_copy(ORB_ID(attitude_correction), sub_att_corr, &att_offset);

}


att.roll = euler[0] + att_offset.roll;

att.pitch = euler[1] + att_offset.pitch;
```

4.3.5. *Driver del sonar*

Los driver se integran en el software igual que el resto de aplicaciones. Se ha creado una carpeta */src/drivers/sonar* donde se han incluido los ficheros *sonar.cpp*, con el código del driver y el fichero *modules.mk* con las siguientes líneas:

```
MODULE_COMMAND = sonar

SRCS           = sonar.cpp
```

Además se ha editado el fichero *src/makefiles/config_px4fmu-v2_default.mk* para hacer que el driver se compile con el resto del firmware, incluyendo la siguiente línea:

```
MODULES        += drivers/sonar
```

Los drivers encargados de controlar los sensores, tanto externos como internos de Pixhawk se implementan utilizando llamadas al sistema en forma de funciones para proporcionar un acceso rápido a los datos y así poder mantener actualizados los valores de entrada al controlador de orientación. La mayoría de los driver cuentan con dos fases:

- Fase de medida: **measure().** Desde Pixhawk se envía al sensor una orden para que el sensor tome una medida y Pixhawk queda a la espera de recibir la respuesta.
- Fase de recopilación de datos: **collect().** Una vez se ha hecho una petición de datos, Pixhawk hace una llamada al sistema para leer del correspondiente puerto de conexión los datos que espera recibir. El periodo de lectura se configura en función del sensor.

En el caso del sonar utilizado para este proyecto, no se requiere ninguna orden para transmitir los datos obtenidos, sino que estos son enviados de forma periódica a través del puerto serie, usando protocolo RS232. Cada 99 ms, el sonar envía la distancia al obstáculo más cercano en forma de 5 caracteres ASCII. El primero de ellos es siempre una “R” seguida de tres caracteres numéricos, que representan la distancia en centímetros y por último un carácter de “carriage return”. En la Figura 4-4. Bits transmitidos por el sonarFigura 4-4 se puede ver una lectura del puerto serie del sonar, donde se puede comprobar que los niveles de señal son de 0 a 5 V y que la señal se recibe invertida, es decir, 0 V corresponde a un 1 lógico mientras que 5 V representa un 0 lógico.

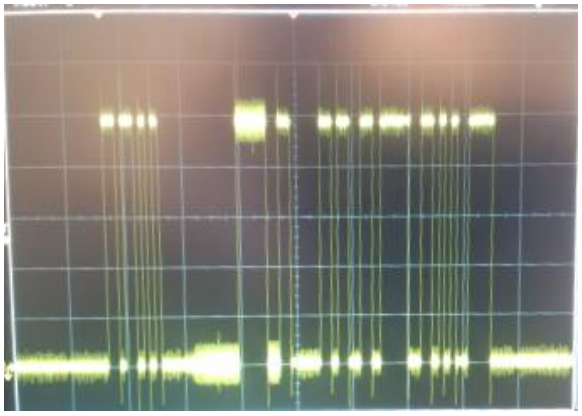


Figura 4-4. Bits transmitidos por el sonar

Las conexiones serie de Pixhawk son de tipo UART en lugar de RS232 y aunque soportan niveles de señal de 5 V, Pixhawk no es capaz de interpretar los datos recibidos. La solución más sencilla ha sido introducir un inversor de señal lógico entre el pin 5 del sonar y el pin RX del puerto serie de Pixhawk.

El driver realiza una llamada al sistema cada 99 ms para obtener los datos del sensor. Identifica el carácter de inicio (R) y después interpreta los tres dígitos. Si no hay ningún error en la transmisión, el valor leído se transforma a unidades del sistema internacional, m, y se publica en el tópico ***optical_flow***, que es donde el controlador de posición espera obtener la distancia medida por el sonar.

4.3.6. *Máquina de estados*

Según la configuración de Pixhawk, cuando un cuadricóptero pierde la comunicación con la estación base, trata de forma autónoma de volver al punto de despegue, es decir, pasa a modo RTL. Para las primeras pruebas esta configuración no resultaba útil, por lo que se ha decidido modificar la máquina de estados para que en caso de perder la comunicación, el UAV no trate de regresar a ningún punto, sino que aterrice de forma controlada en el mismo lugar en el que se produce la pérdida. Esta medida es útil dado que en las primeras pruebas no se van a realizar vuelos de larga distancia donde cabría la posibilidad de perder el UAV si aterriza en un lugar donde no hay comunicación por radio.

La máquina de estados, por lo tanto, se modifica para adaptarse al siguiente diagrama:

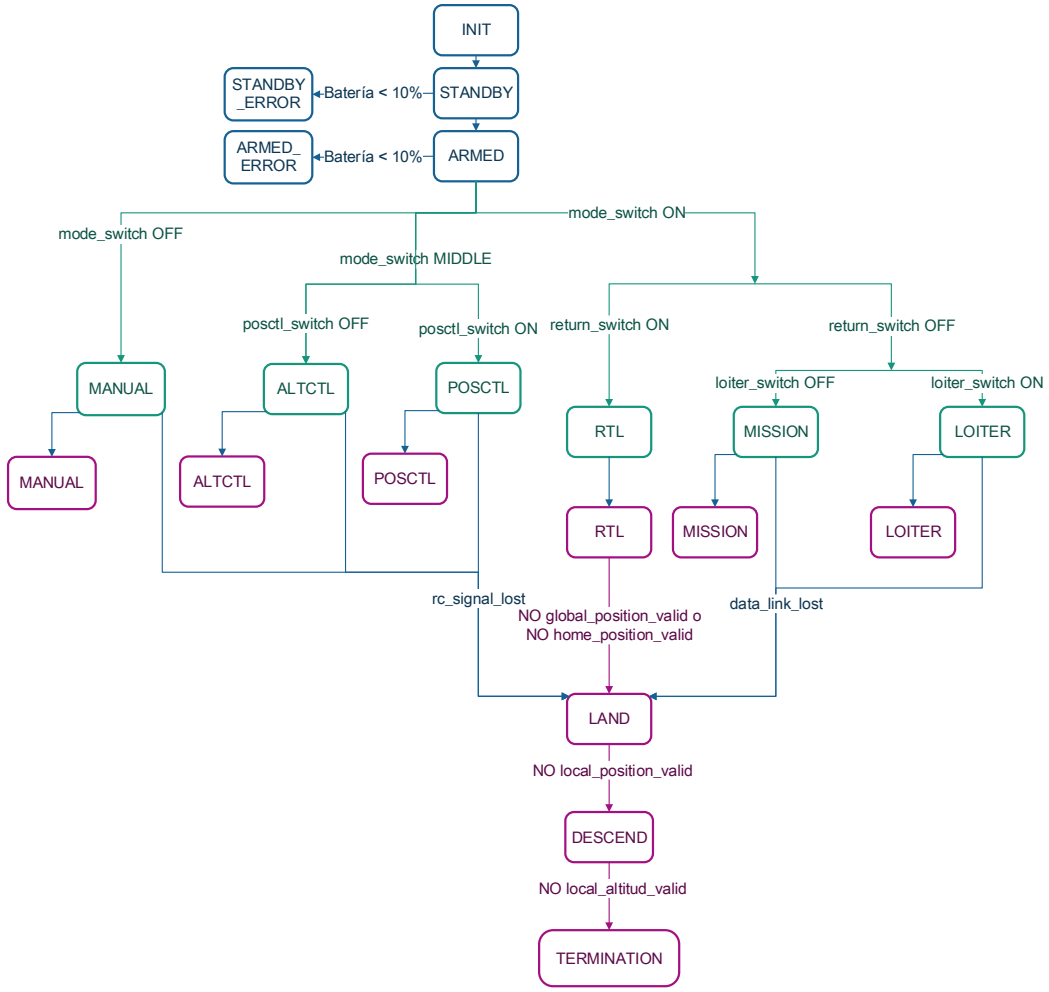


Figura 4-5. Máquina de estados modificada

4.3.7. *Resumen*

A continuación se representa de nuevo el diagrama que muestra la estructura del software resaltando los cambios realizados:

En el ANEXO D se puede encontrar el registro del gestor de versiones GIT, donde se especifican todos los cambios realizados en el firmware respecto a la versión estable utilizada de inicio.

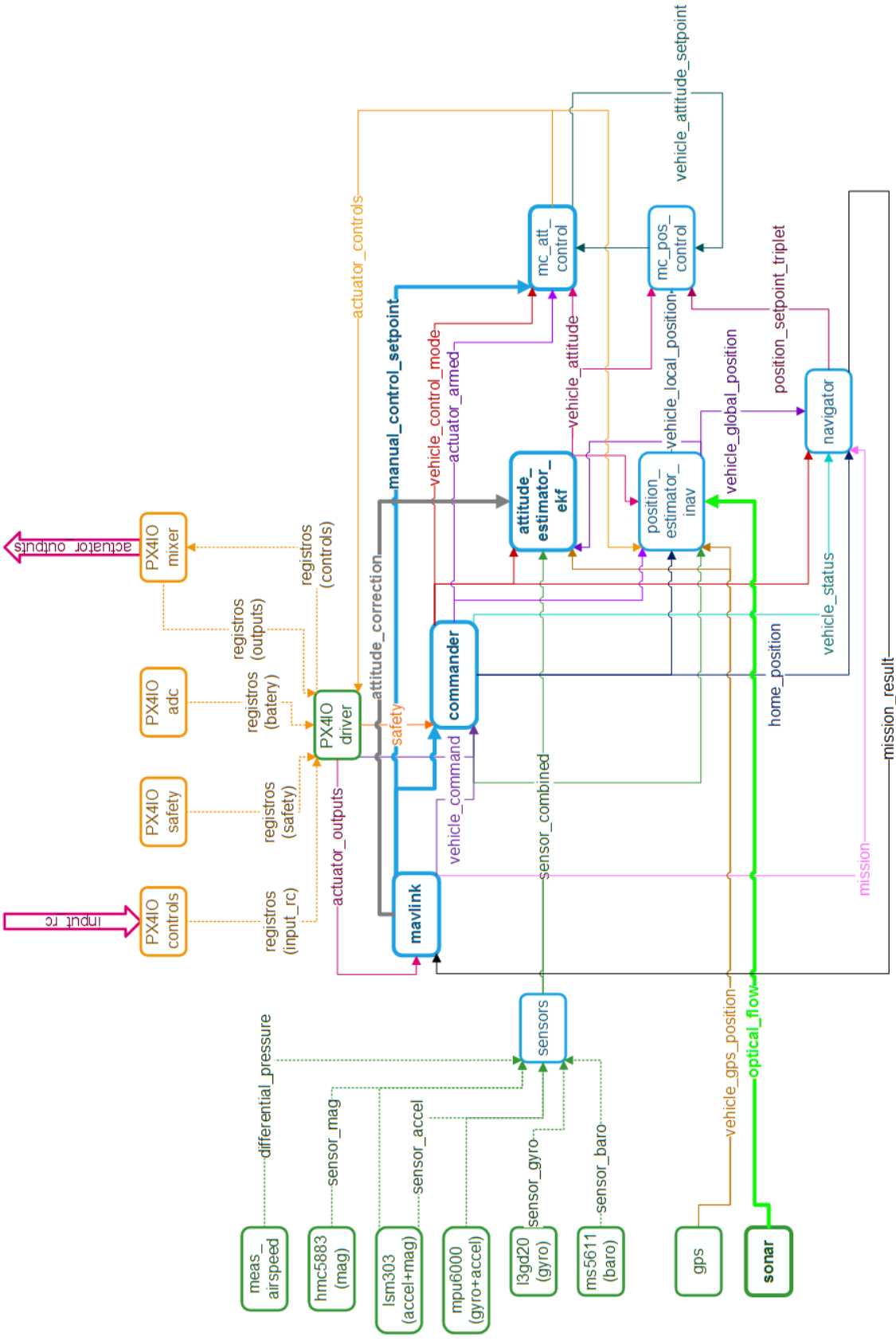


Figura 4-6. Arquitectura del firmware de Pixhawk definitiva

4.4. Configuración del canal de comunicación para telemetría y control

4.4.1. *Conexión*

4.4.1.1. Conexión cableada

Para el enlace de control y telemetría se utiliza uno de los puertos serie de Pixhawk, en concreto el puerto Telem1, que corresponde a la ruta /dev/ttyS1. En este caso no se va a tener control de flujo de datos, por lo que se van a utilizar solamente cuatro de los pines de conexión: 5V, GND, TX y RX.

En las primeras fases del proyecto se ha utilizado una conexión por cable entre el UAV y el ordenador. Para ello se ha utilizado un conversor serie a USB con chip FTDI, idéntico al utilizado para la comunicación con la consola de Pixhawk (4.1.1.4.).

4.4.1.2. Conexión inalámbrica

La conexión final se lleva a cabo mediante los dos módulos XBee, que son capaces de sustituir la conexión serie por un enlace inalámbrico de forma completamente transparente tanto para el usuario final como para el desarrollador. Los elementos necesarios para el enlace son:

- Dos módulos XBee
- Una placa de soporte de XBee con salida USB, para conectar al ordenador.
- Una placa de soporte de XBee con conexión serie, para el UAV

Utilizando el software X-CTU es posible configurar los módulos XBee, como se explica en el siguiente apartado, y también proporciona herramientas de prueba. Conectando uno de los módulos al ordenador, y otro simplemente a alimentación se comprobó que los dispositivos funcionaban correctamente (más detalles en la sección de pruebas). Pero al conectar el segundo módulo a Pixhawk e intentar obtener la comunicación que se había conseguido entre QGroundControl y Pixhawk, no se obtuvieron resultados.

Tras varias pruebas y observaciones se llegó a la conclusión de que el problema se encontraba en la comunicación serie entre el Pixhawk y la placa reguladora de XBee. En concreto en la entrada de datos hacia el módulo XBee. Es decir, el ordenador enviaba los paquetes a través del primer dispositivo XBee, el segundo los recibía perfectamente y se los pasaba al Autopilot, éste los recibía e intentaba contestar, pero los paquetes no llegaban a ser enviados por el enlace inalámbrico.

La placa reguladora se trata del dispositivo XBee Explorer Regulated, un modelo comercial fabricado por Sparkfun. Se encarga de convertir señales serie de 5 V a 3.3 V, de forma que se pueda conectar cualquier sistema que trabaje con 5 V a cualquier

dispositivo XBee. Además cuenta con una serie de LEDs que funcionan como indicadores de actividad de las señales de alimentación, RSSI, DIN y DOUT. La alimentación del dispositivo se va a obtener a través de Pixhawk, el cual ofrece una tensión de salida de 5 V en sus puertos serie. Aunque el nivel de señal de Pixhawk es de 3.3 V, puede soportar conexiones serie a 5 V. Por lo tanto el dispositivo de Sparkfun es completamente válido para este proyecto.

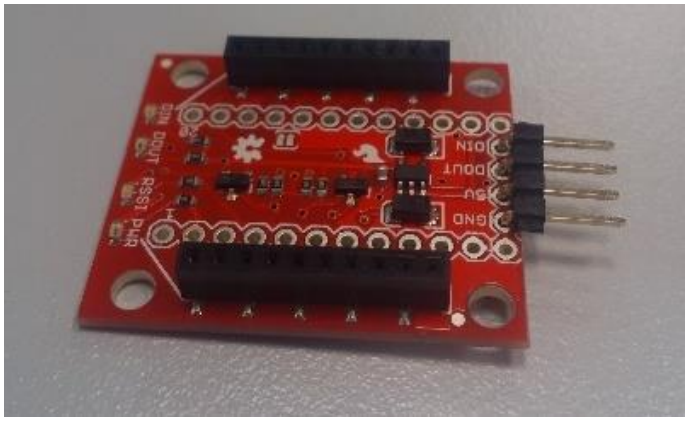


Figura 4-7. XBee Xplorer Regulated

Al medir con un osciloscopio la salida del módulo XBee se observó que las tensiones se situaban entre 0.8 y 5V. Pixhawk utiliza señales TTL, a 3.3V con tolerancia a 5V. Teniendo en cuenta que la tolerancia de los ceros en señales lógicas es de un 30% de la tensión de activa, en el peor de los casos bastaría con tener una tensión por debajo de 0.99V para que Pixhawk entendiese las señales.

Por el contrario, al medir en la entrada de datos hacia el XBee, las formas de onda oscilaban entre 2 y 3.3V, por lo tanto el módulo XBee no entendía el cero lógico, al encontrarse por encima del umbral de 0.99V.

En un primer momento se pensó que las resistencias de pull up o pull down que se encuentran a las salidas de XBee pudiesen estar produciendo el problema. Estas resistencias se pueden deshabilitar mediante software. Tras probar, el resultado seguía siendo el mismo.

Con la información de los esquemáticos y las hojas de datos de los dispositivos utilizados se obtiene que el esquema de conexión entre XBee y Pixhawk es el siguiente tanto en los pines de entrada como de salida de ambos dispositivos:

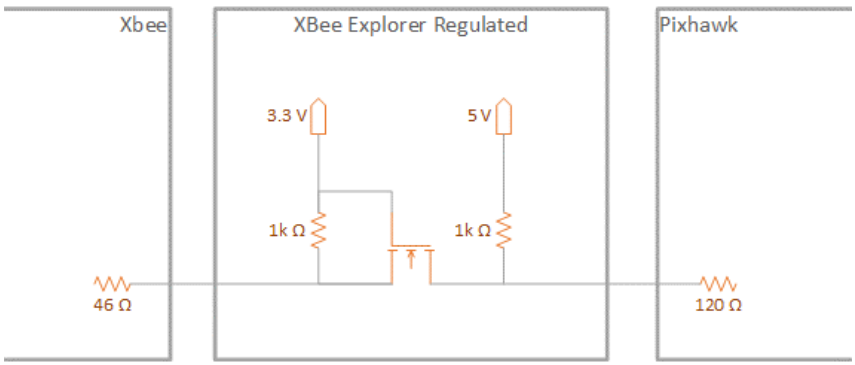


Figura 4-8. Esquema XBee Xplorer

El problema se encuentra en la entrada de datos de XBee, en concreto, cuando se transmite un 0 lógico. En este caso el circuito equivalente es el siguiente:

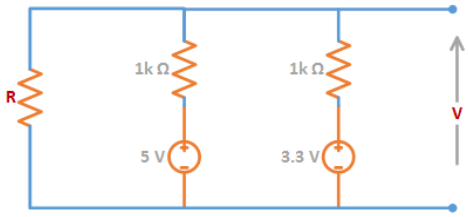


Figura 4-9. Esquema equivalente XBee Xplorer

Las resistencias de la placa reguladora se ha podido comprobar que son las correctas, por lo tanto el problema podría estar en la resistencia de Pixhawk. La resistencia de pull-up de XBee continúa desactivada.

Del circuito se obtiene la siguiente relación entre la tensión a la entrada de XBee y la resistencia en la salida de Pixhawk:

$$V = 4.15 - \frac{8300}{4R + 2000}$$

Si la resistencia tuviese el valor que se indica en el esquemático de Pixhawk, 120Ω, la tensión sería:

$$V = 4.15 - \frac{8300}{4 \cdot 120 + 2000} = 0.8\,V$$

Como se ha dicho antes, 0.8 V es un valor lo suficientemente bajo como para que se interprete como un 0 lógico. Como se ha medido que la tensión es aproximadamente de 2 V, se hace el cálculo inverso para calcular el valor de la resistencia de Pixhawk:

$$R = \frac{2075}{4.15 - V} - 500$$

Con V = 2 V:

$$R = \frac{2075}{4.15 - 2} - 500 = 465\ \Omega$$

La solución más sencilla a este problema ha sido eliminar el circuito de regulación de tensión de ese pin de salida hacia XBee. Es decir, las posiciones de las resistencias ahora son un circuito abierto y se ha cortocircuitado los pines Source y Drain del transistor. No supone ningún problema ya que el nivel de señal de salida de Pixhawk es de 3.3 V.

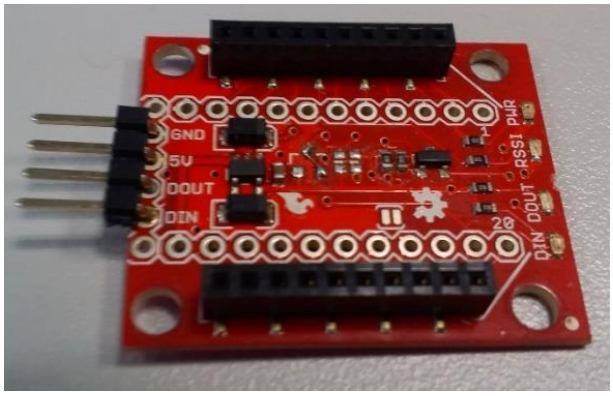


Figura 4-10. XBee Xplorer Regulated modificado

El resto del circuito se ha mantenido porque la regulación de tensión de alimentación sí que es necesaria y además la placa reguladora sirve de soporte para instalar el módulo XBee en la estructura del UAV.

4.4.2. *Configuración de los parámetros de XBee*

Los módulos XBee tienen numerosas posibilidades de configuración según el escenario en el que vayan a ser utilizados. Para ello, se cuenta con la aplicación X-CTU, que proporciona acceso a los parámetros de configuración además de ofrecer herramientas para la prueba de los dispositivos.

En este proyecto se van a utilizar dos módulos XBee para sustituir un enlace serie entre un UAV y un ordenador. Por lo tanto, en principio no se va a hacer uso de las configuraciones de red que ofrece, sino que se va a implementar un enlace de comunicación punto a punto.

Al conectar un módulo XBee al ordenador se puede seleccionar mediante X-CTU. En la parte izquierda aparecen lo módulos conectados y a la derecha se pueden ver los parámetros tal y como están configurados en cada dispositivo. Los parámetros se organizan en apartados que se pueden desplegar.

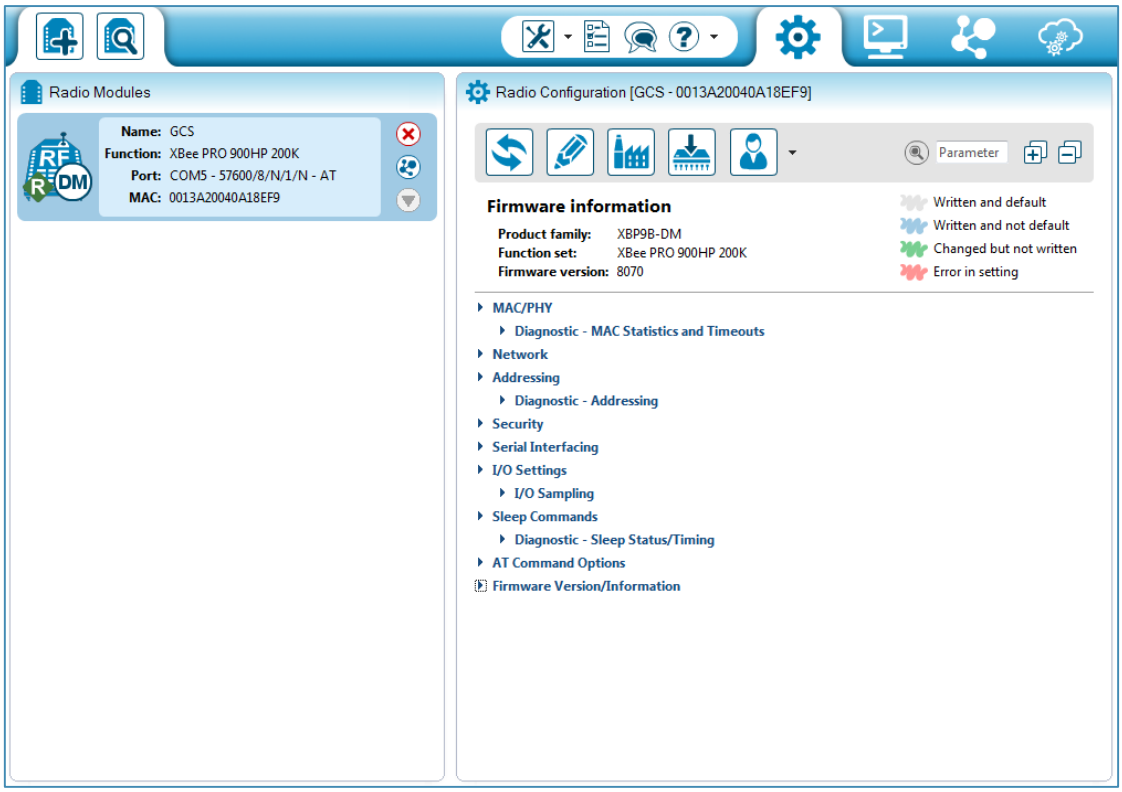


Figura 4-11. Pantalla principal de X-CTU

Los primeros cambios se realizan en el apartado “Adressing”:

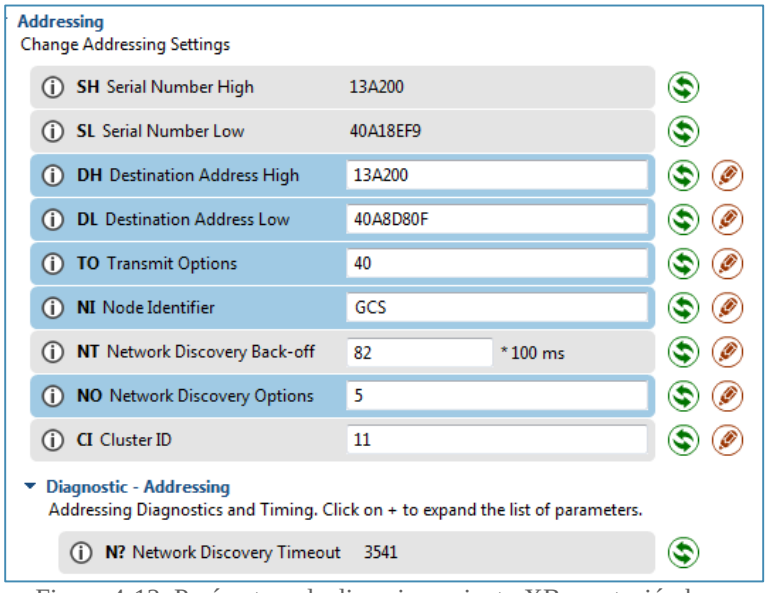


Figura 4-12. Parámetros de direccionamiento XBee estación base

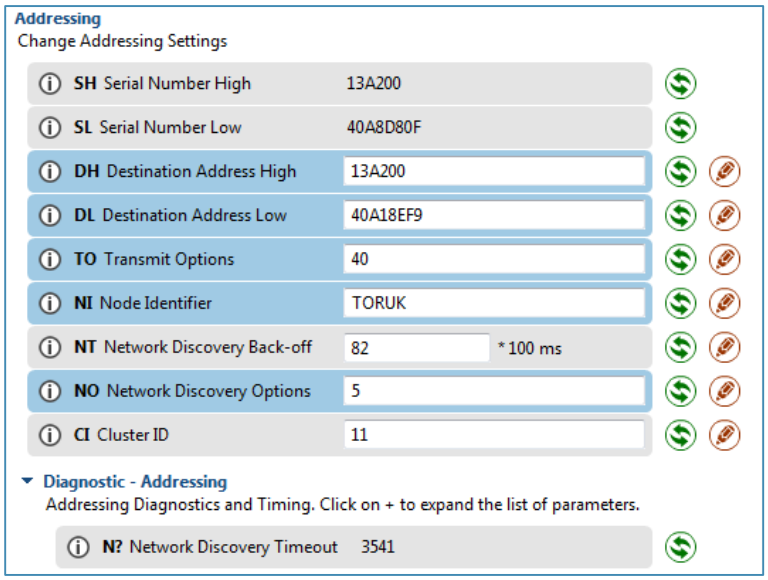


Figura 4-13. Parámetros de direccionamiento XBee UAV

- Para obtener una comunicación punto a punto, en el campo de dirección de destino (Destination Address) de cada dispositivo debe aparecer el número de serie (Serial Number) del otro de los dispositivos. Además, se tiene que indicar el tipo de enlace en el campo de opciones de transmisión (Transmit Options), el valor 40 corresponde con transmisiones punto a punto o punto a multipunto.
- Se ha asignado un nombre a cada nodo para facilitar su identificación (Node Identifier)

También se ha modificado el apartado "I/O Settings", el que se configuran las funciones de los pines de entrada y salida de XBee:

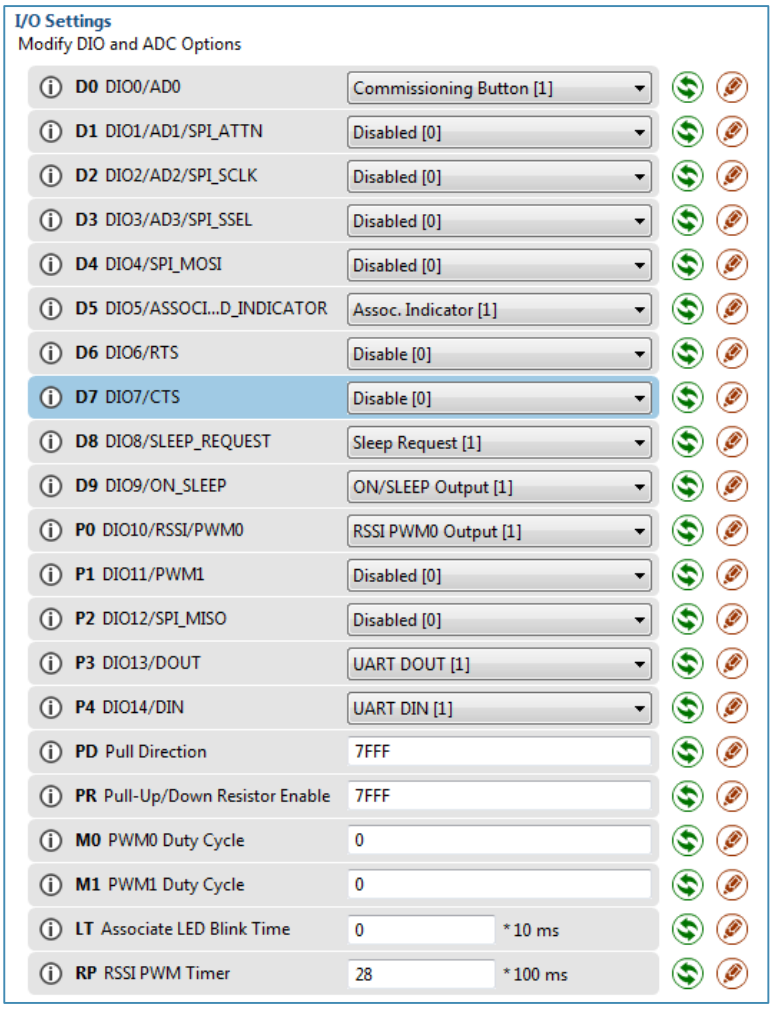


Figura 4-14. Parámetros de E/S XBee GCS y UAV

- No va a existir control de flujo de datos en ninguno de los dispositivos, por lo tanto, se deshabilita el pin CST.

4.5. Estación base

Por facilidades a la hora de programar se ha decidido utilizar Linux como sistema operativo, en concreto la distribución Ubuntu, que se ha probado en dos versiones distintas: 12.04 y 14.04

En esta sección se habla sobre los programas que se utilizan en la estación base para recibir la telemetría a la vez que se controla el cuadricóptero con un mando de ordenador.

4.5.1. *Transmisión de telemetría a QGroundControl*

El firmware de Pixhawk envía de forma continua mensajes MAVLink indicando el estado del UAV. Para recibir estos mensajes en la estación base se ha utilizado el programa QGroundControl. En un principio se intentó configurar QGroundControl para que enviase las señales de control, pero al tratarse de software en desarrollo, se observó que todavía no está implementada esa parte. Por eso se ha desarrollado un programa independiente que envía los controles de un mando conectado al ordenador por USB.

El dispositivo que se utiliza para comunicar con el UAV, el módulo XBee, se conecta al ordenador por un puerto USB. Este puerto USB sólo se puede abrir desde una aplicación al mismo tiempo. Es decir, si se conecta QGroundControl para recibir la telemetría, no es posible abrir el puerto de comunicación desde el programa creado para enviar los datos de control. Para solventar este problema se ha decidido utilizar una aplicación que permita crear puertos virtuales e interconectarlos entre sí. El esquema de comunicación entre las aplicaciones y el módulo XBee se puede ver en la siguiente imagen:

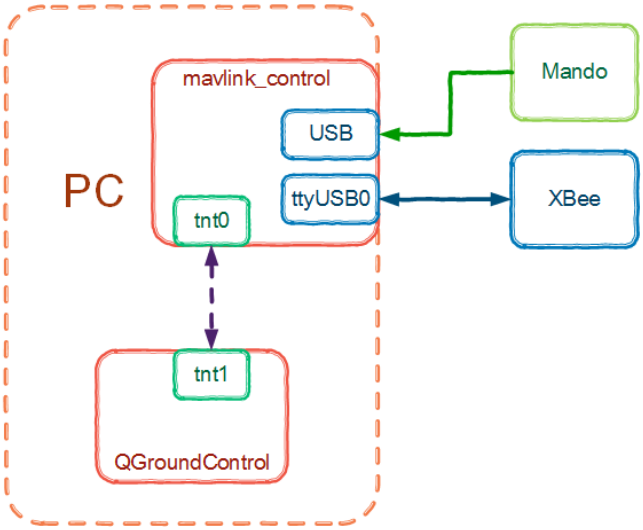


Figura 4-15. Comunicación entre aplicaciones mediante puertos virtuales

Para crear el enlace entre la aplicación de control y QGroundControl se ha utilizado el software tty0tty, que permite crear parejas de puertos serie virtuales. En concreto, la aplicación tty0tty/module crea los siguientes pares de puertos:

Tabla 4-3. Puertos serie virtuales

/dev/tnt0	⇔	/dev/tnt1
/dev/tnt2	⇔	/dev/tnt3
/dev/tnt4	⇔	/dev/tnt5
/dev/tnt6	⇔	/dev/tnt7

La conexión que se establece entre ellos es la misma que la de cualquier puerto serie:

Tabla 4-4. Comunicación entre puertos serie

TX	→	RX
RX	←	TX
RTS	→	CST
CTS	←	RTS
DSR	←	DTR
CD	←	DTR
DTR	→	DSR
DTR	→	CD

La aplicación de control establece un enlace de comunicación bidireccional con el módulo XBee. Este enlace permite recibir toda la información de telemetría que envíe el UAV así como enviar todas las órdenes de control. Los datos de telemetría recibidos se leen del puerto físico y se escriben en un puerto virtual, en este caso el puerto tnt0. Además, este puerto virtual establece una comunicación bidireccional con QGroundControl, por lo tanto la información que QGroundControl quiera enviar al UAV también pasará por la aplicación de control.

4.5.2. *Aplicación de control*

4.5.2.1. Descripción de la aplicación

El programa que se ha usado para enviar los datos de control al UAV realiza las siguientes tareas:

- Interpretación de los controles del mando.
- Envío de mensajes MAVLink al UAV.
- Recepción de mensajes MAVLink del UAV y retransmisión a QGroundControl.

El esquema del software desarrollado se muestra a continuación.

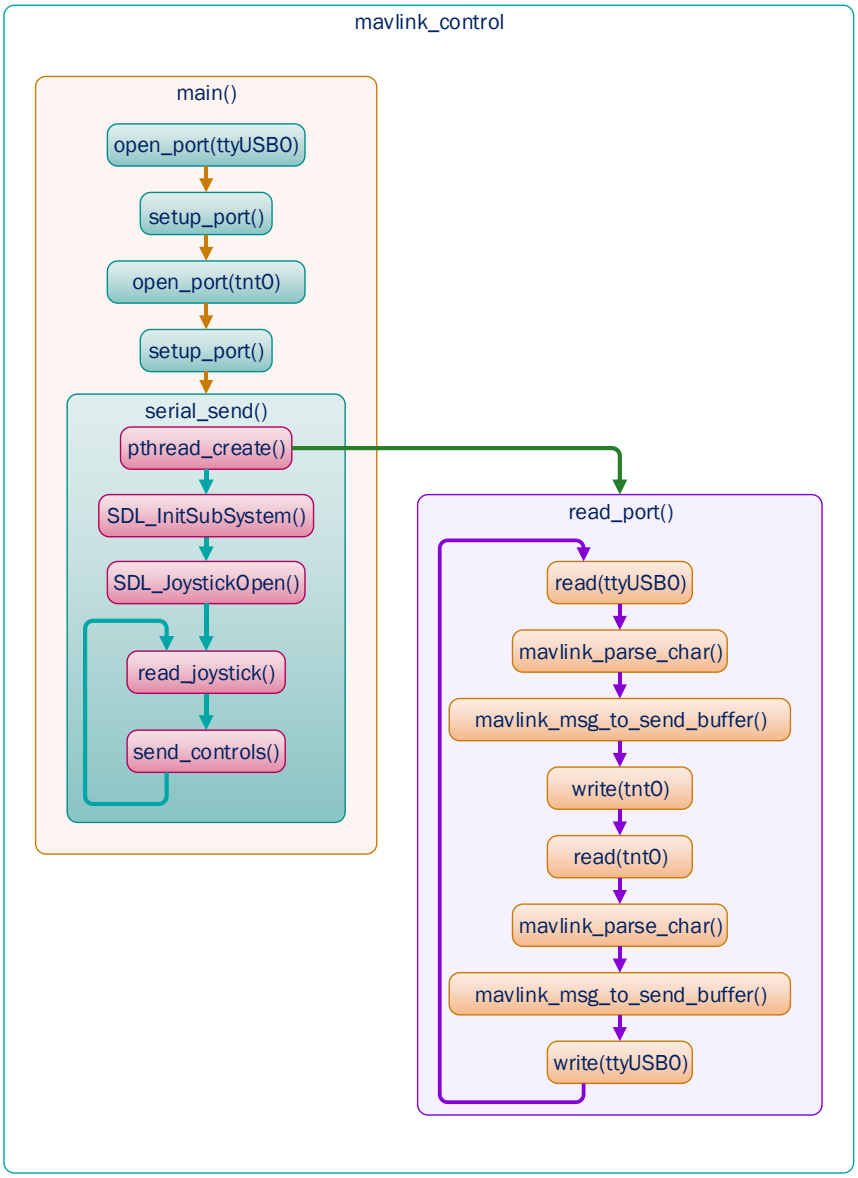


Figura 4-16. Arquitectura de la aplicación *mavlink_control*

En primer lugar se configuran los dos puertos serie que se van a utilizar. Es posible indicar como argumento el puerto al que se encuentra conectado el adaptador de XBee y la velocidad de transmisión de datos de los puertos serie. Si no se indica nada, la configuración por defecto intenta abrir el puerto ttyUSB0 para establecer la comunicación con XBee y el puerto virtual tnt0 para la comunicación con QGroundControl. Los dos puertos los configura a una velocidad de 57600 bps. Después inicia la función serial_send. Esta función crea un nuevo hilo que ejecuta la función read_port en paralelo al hilo principal.

La función read_port es la encargada de transmitir los datos de telemetría que envía el UAV a QGroundControl y viceversa. Para ello lee del puerto ttyUSB0 (o el que se haya indicado como argumento de entrada) y, si recibe algún mensaje MAVLink, lo escribe directamente en el puerto virtual tnt0. A continuación, comprueba si QGroundControl pretende enviar algún mensaje al UAV, leyendo el puerto tnt0 y en caso afirmativo lo

escribe en el puerto ttyUSB0. Toda esta rutina se repite de forma continua a no ser que se indique lo contrario desde el hilo principal.

Mientras tanto, continúa la ejecución de la función **serial_send**. Utiliza las funciones de la librería SDL para inicializar el subsistema que controla el joystick y abre la comunicación con este. Después entra en un bucle que se repetirá hasta que finalice la ejecución del programa. Dentro del bucle, llama continuamente a las funciones read_joystick y send_controls, con una pausa de 0.1s entre cada iteración, cada vez que se llama a la función send_controls se pausa la función read_port, ya que ambas funciones escriben en el mismo puerto serie.

La función **read_joystick** interpreta cualquier interacción con el mando. Detecta si se pulsa alguno de los botones de cambio de modo y calcula los valores de throttle, roll, pitch y yaw que enviar al UAV. Para ello utiliza la librería SDL como se detalla en el siguiente apartado.

La función **send_controls** recibe los datos obtenidos por la función read_joystick y los encapsula en el mensaje "mavlink_msg_manual_control". A continuación los escribe en el puerto ttyUSB0 para que el mensaje sea transmitido al UAV. Los controles se envían periódicamente aunque no se produzca ninguna variación en el estado en el mando o los valores leídos sean 0. Así, Pixhawk interpreta que existe comunicación de RC.

Se consigue un enlace de comunicación idéntico al que ocurriría si sólo se tuviera enlace de telemetría entre el UAV y QGroundControl, pero se incluye un mensaje de control cada 0.1 s.

4.5.2.2. Controles de mando y librearía SDL

El mando que se ha utilizado para controlar el cuadricóptero se trata de un joystick genérico de ordenador. Al no tratarse de un mando específico para controlar cuadricópteros se han tenido que adaptar los controles a los pulsadores y ejes disponibles.

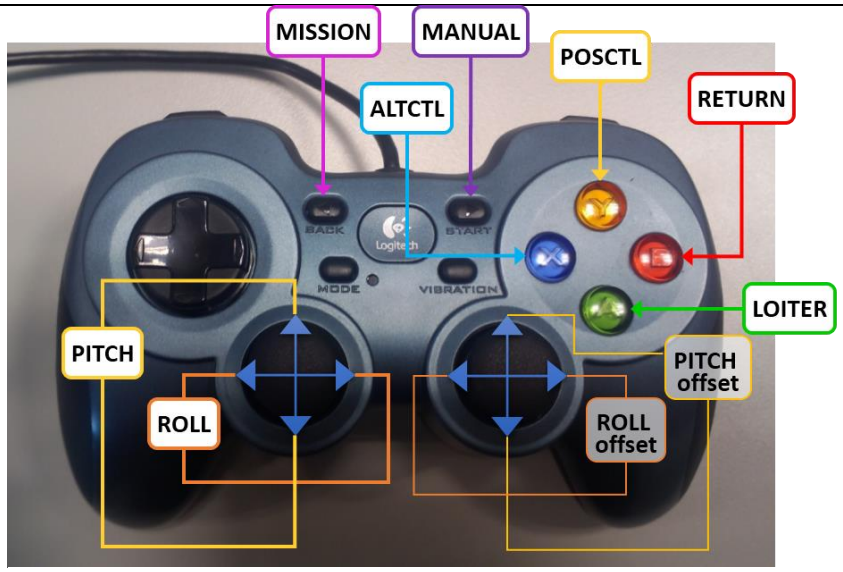


Figura 4-17. Controles del mando 1

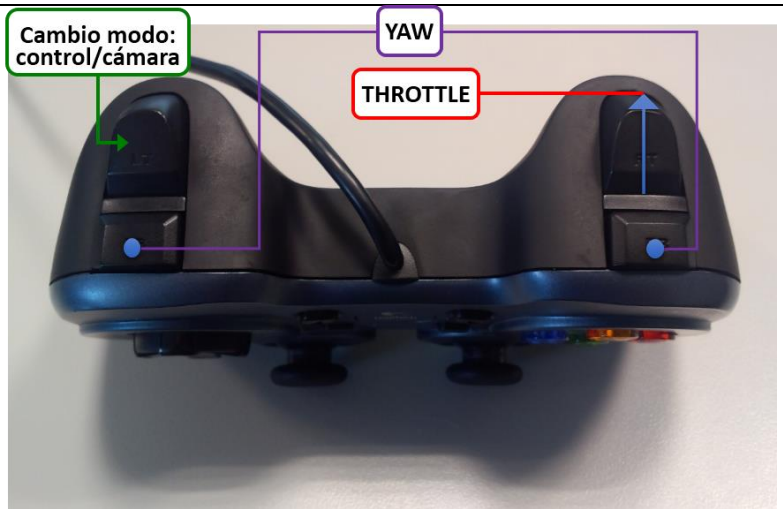


Figura 4-18. Controles del mando 2

En este proyecto se han utilizado un conjunto de funciones SDL para proporcionar acceso de forma sencilla a los controles de entrada de un joystick de ordenador. SDL (Simple DirectMedia Layer) es una librería de desarrollo multiplataforma diseñada para proporcionar acceso de bajo nivel a audio, teclado, ratón, joystick, y los gráficos de hardware a través de OpenGL y Direct3D.

Las funciones más relevantes utilizadas son:

SDL_JoystickGetButton: Sirve para obtener el estado de un botón del mando. Devuelve 1 si el botón está pulsado y 0 si no lo está.

SDL_JoystickGetAxis: Obtiene la posición de un eje de control del joystick. Los valores de la posición oscilan entre -32768 y 32767.

4.5.3. *Modificación QGroundControl*

QGroundControl no detecta los puertos virtuales, y por lo tanto no permite conectar con el UAV a través del enlace de control creado. Para solucionar este problema se ha tenido que modificar el software.

En el archivo */qgrouncontrol/libs/serialport/qserialportinfo_unix.cpp* en la función **filtersOfDevices()** se ha añadido la línea:

```
<< QLatin1String( "tnt*" );
```

Para que el programa tenga en cuenta los puertos virtuales creados.

Y se ha comentado la línea 186, para que aparezcan todos los puertos en la lista de disponibles, y no sólo aquellos que el sistema operativo detecte como conectados. Esta línea era:

```
canAppendToList = false
```

4.5.4. *Tráfico de paquetes MAVLink*

La comunicación entre el UAV y la estación base se trata de un enlace bidireccional. Desde el UAV se envían los mensajes de telemetría, mientras que desde la estación base se envían dos tipos de mensajes: los mensajes de control y los mensajes que envía QGroundControl hacia el UAV.

Se hace diferencia entre los dos tipos de mensajes que se transmiten desde la estación base porque Pixhawk tiene en cuenta dos tipos de enlaces: el enlace de control y el enlace da datos.

- **El enlace de control** es necesario que se encuentre activo para permitir que el UAV vuele en los modos manuales. Pixhawk interpreta que existe enlace de control si se recibe al menos un mensaje cada 0.5 s.
- **El enlace de telemetría** es necesario cuando el modo de vuelo es autónomo. El estado de este enlace se verifica comprobando si se recibe al menos un mensaje de señal de vida cada 5 segundos. Si en ningún momento se reciben mensajes de señal de vida, Pixhawk no hace ninguna comprobación del enlace de telemetría.

En las siguientes tablas se detalla el tipo de mensaje y su correspondiente número de identificación así como el número de mensajes que se envían por segundo y la longitud en bytes del campo de datos de cada mensaje. El mensaje

“mavlink_msg_named_value_float” sólo se recibe en la estación base cuando se requiere una lectura de los parámetros del sistema.

Tabla 4-5. Mensajes MAVLink enviados por Pixhawk

ID	Mensaje	Msj/s	Longitud
30	ATTITUDE	10	28
74	VFR_HUD	10	20
32	LOCAL_POSITION_NED	3	28
52	GLOBAL_POSITION_SETPOINT_INT	3	15
58	ROLL_PITCH_YAW_THRUST_SETPOINT	3	20
0	HEARTBEAT	1	9
1	SYS_STATUS	1	31
105	HIGHRES_IMU	1	62
24	GPS_RAW_INT	1	30
49	GPS_GLOBAL_ORIGIN	1 (cada 2 s)	12
33	GLOBAL_POSITION_INT	3	28
35	RC_CHANNELS_RAW	1	22
132	DISTANCE_SENSOR	1 (cada 2 s)	14
251	NAMED_VALUE_FLOAT	1	18

Desde la estación base se envían periódicamente los siguientes mensajes:

Tabla 4-6. Mensajes MAVLink enviados por la estación base

ID	Mensaje	Msj/s	Longitud
30	MANUAL_CONTROL	10	15
74	HEARTBEAT	1	9

Si se envían todos los mensajes posibles, el tráfico de datos en un segundo es de 1110 Bytes, repartidos en un total de 51 mensajes. Teniendo en cuenta que los mensajes MAVLink cuentan con una cabecera de 8 bytes, el tráfico total del enlace es:

$$1110 + 51 \cdot 8 = 1518\ Bytes$$

Es decir, se transmiten un máximo de 12144 bits en un segundo.

4.6. Montaje de componentes

La estructura del UAV ha sido diseñada por Robomotion, mientras que la selección de componentes para el sistema de propulsión (motores, hélices, variadores, etc.) y la decisión de utilizar un cuadricóptero en lugar cualquier otro tipo de aeronave se ha realizado en otro Proyecto Fin de Carrera paralelo a éste: "Integración de un UAV (vehículo aéreo no tripulado) en la plataforma robótica ARGOS" [29].

La estructura se compone de dos placas de fibra de carbono unidas entre sí mediante cuatro cilindros con diámetro suficiente para permitir el giro de las hélices. En la parte central, en el espacio interior que hay entre las dos placas se sitúa otra placa más pequeña. También de fibra de carbono, diseñada para alojar la mayoría de los componentes electrónicos, la cual se fija a la placa inferior.

Figura 4-19. Placa de montaje de electrónica

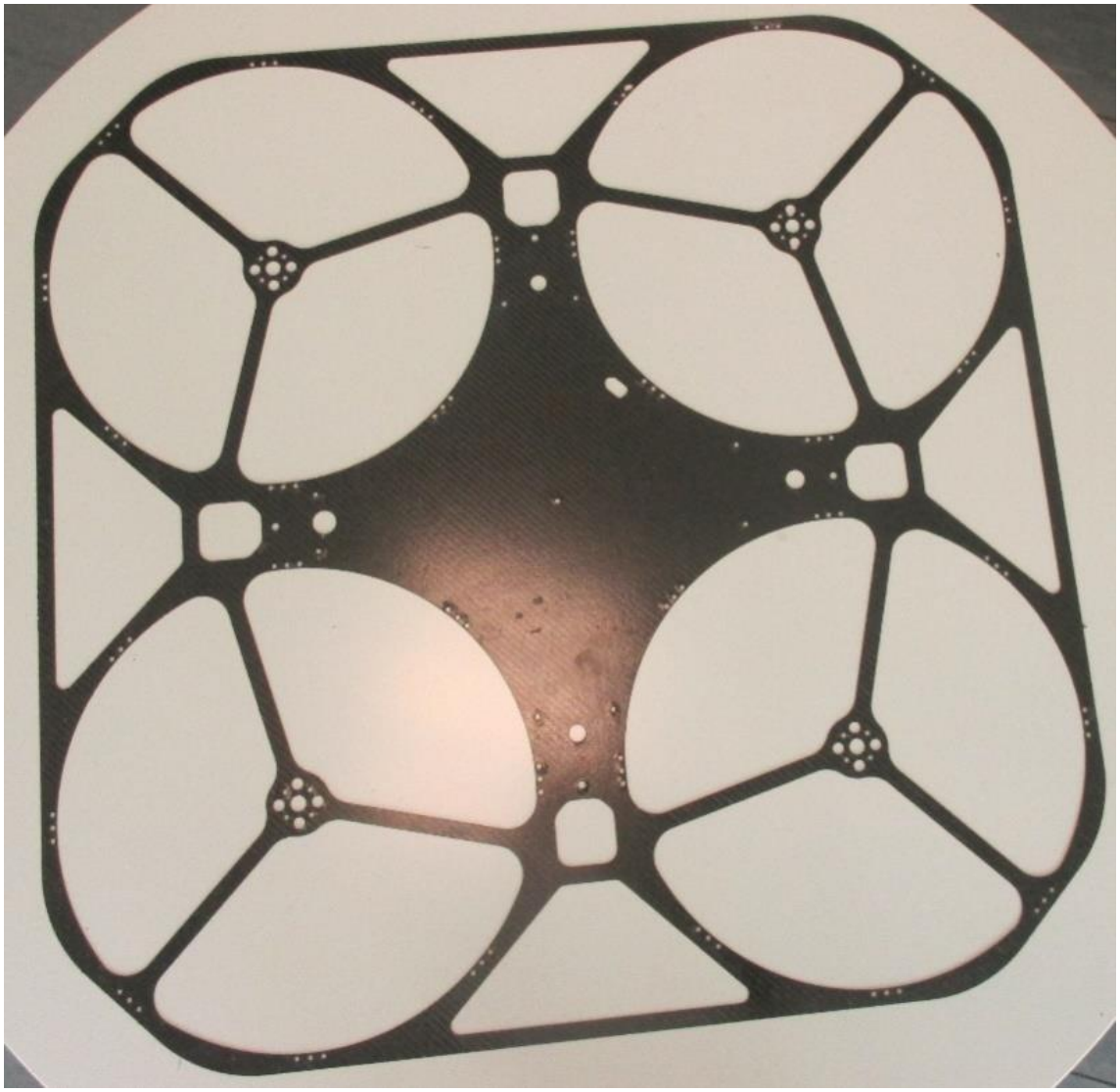


Figura 4-20. Placa de montaje inferior

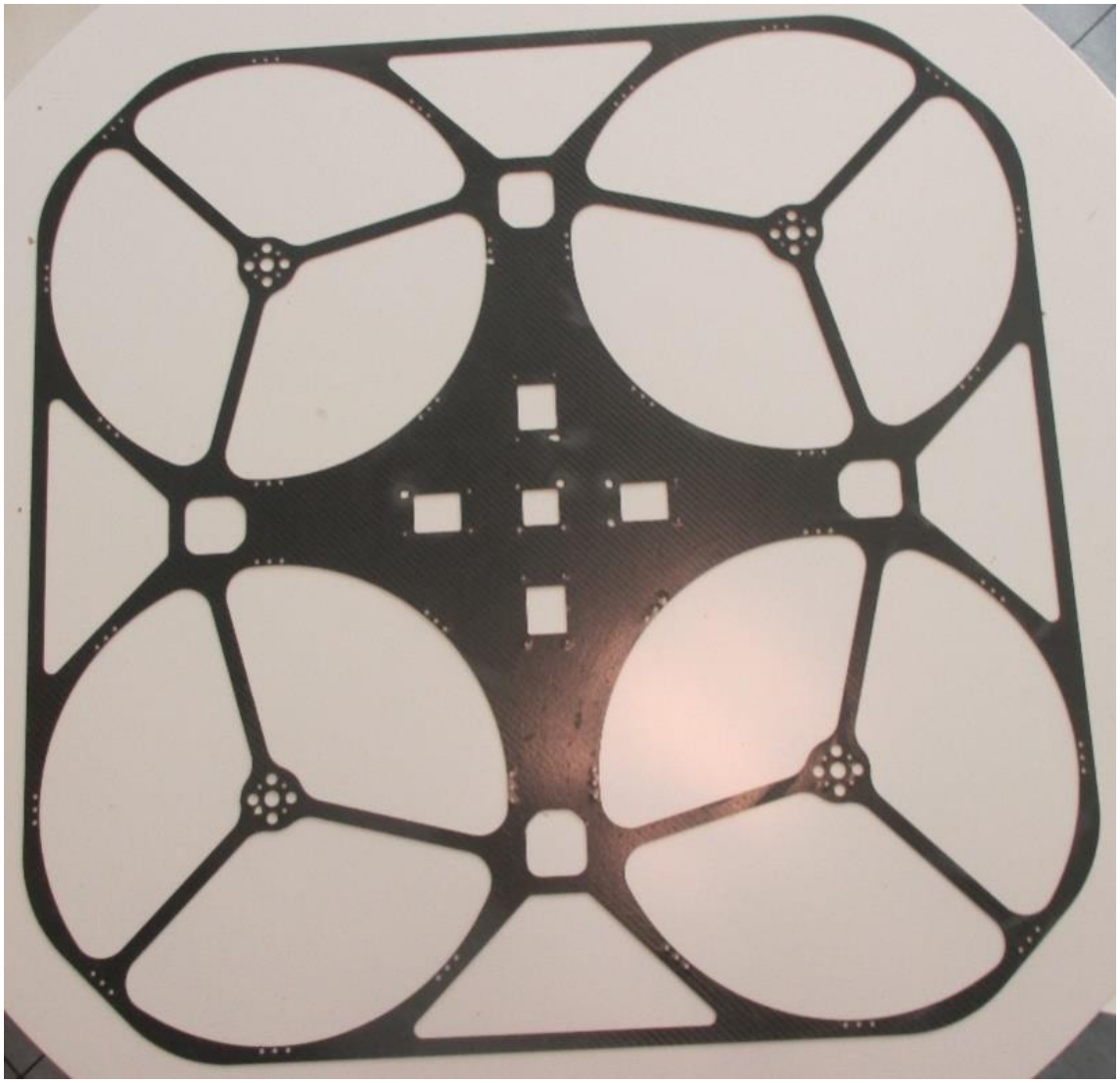


Figura 4-21. Placa de montaje superior

El resultado con la electrónica montada se muestra a continuación.

Por un lado, en la placa superior, se sitúa el GPS con la antena en el hueco central, los cuatro variadores y los motores que se conectarán a estos:

Figura 4-22. Montaje variadores, motores y GPS

El resto de los componentes electrónicos se sitúan en la placa intermedia:

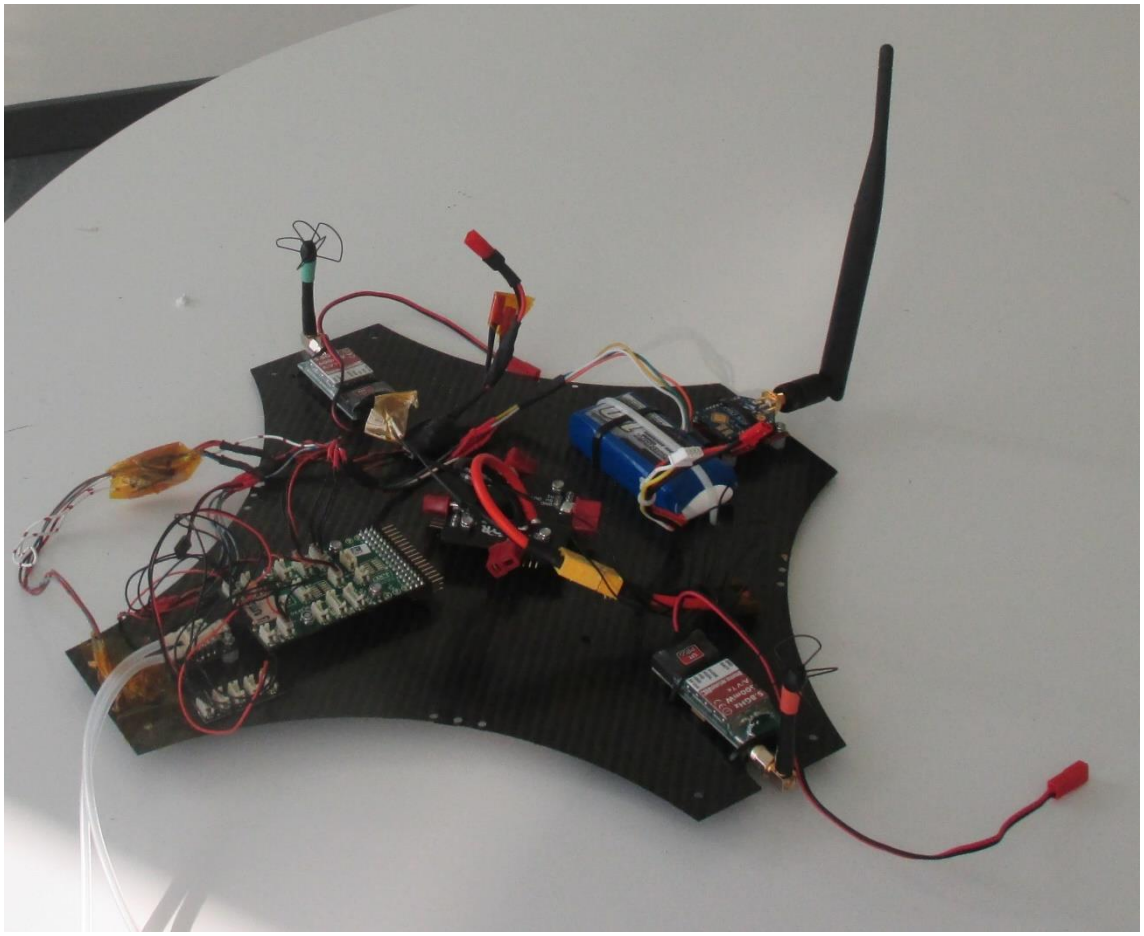


Figura 4-23. Montaje de componentes electrónicos

Aunque en la imagen las antenas se encuentren hacia arriba, cuando se integra con el resto de la estructura la situación de las antenas es la contraria. Se situan en la parte inferior de la plataforma para intentar tener línea de visión directa con la estación base cuando el cuadricóptero esté en vuelo.

A continuación se una la placa de los componentes electrónicos a la placa inferior usando 9 tornillos, 2 en cada extremo y uno central. En las uniones entre ambas placas se han utilizado soportes de goma para absorver las vibraciones y que estas no sean transmitidas a los componentes electrónicos, en especial al controlador de vuelo, donde se encuentra la IMU.

Como se puede comprobar en la imagen en la placa inferior se han fijado los cilindros.

Figura 4-24. Unión de placas inferior y de electrónica

Por último, se unen las dos placas exteriores. Se obtiene el cuadricóptero de la siguiente imagen.

Figura 4-25. Cuadricóptero

La forma de sujeción de las baterías así como las patas del cuadricóptero es un diseño provisional que se ha utilizado para las pruebas.

Figura 4-26. Lateral del cuadricóptero

El diseño final consiste en una única estructura que servirá de soporte para las baterías, tren de aterrizaje y módulo de pan/tilt para la cámara de vídeo.

5 Pruebas y resultados

5.1. Introducción

Finalmente se ha conseguido un sistema compuesto por tres enlaces inalámbricos independientes: dos de ellos para vídeo y audio y un tercer enlace bidireccional para datos de control y telemetría. Mediante el enlace de control es posible teleoperar el cuadricóptero tanto de forma manual como asistida, gracias a los controladores de estabilidad y posición en el propio cuadricóptero. Además se ha diseñado un enlace virtual con el sistema QGroundControl que permite configurar misiones y enviarlas al UAV en forma de waypoints si se selecciona modo de vuelo autónomo.

5.2. Sistema de enlace XBee

5.2.1. *Descripción de la prueba*

La primera prueba consiste en verificar el funcionamiento de los dispositivos seleccionados para el sistema de enlace de control y telemetría.

La prueba se ha realizado en el laboratorio, utilizando el software X-CTU. Se ha conectado uno de los módulos al ordenador mediante conexión USB utilizando una placa de desarrollo y el otro módulo de forma remota. Este último dispositivo se ha ido desplazando hasta el exterior del edificio y después por los alrededores para comprobar el alcance.

Los resultados esperados en esta primera prueba se corresponden a las distancias estimadas para interiores y entornos urbanos, es decir, se pretende alcanzar una distancia cercana a los 300 m.

5.2.2. *Resultado*

En la imagen se pueden ver los puntos donde se situaban los dos módulos XBee en el momento en que se empieza a apreciar pérdida de paquetes de datos. La distancia es de 140 m entre ellos, por lo tanto no se han obtenido los resultados indicados por el

fabricante, pero aun así se considera un resultado razonablemente bueno, debido a las difíciles condiciones de la prueba (falta de línea de vista, fuertes sombras por montañas, grandes estructuras metálicas, etc.).

Figura 5-1. Alcance XBee

5.3. Transmisores de vídeo

5.3.1. *Descripción de la prueba*

La primera prueba se realiza en interior. Se prueban los dos conjuntos transmisor/receptor por separado.

Debido a que el sistema de transmisión es muy susceptible al multitrayecto no se espera una buena calidad de señal al transmitir vídeo en interiores.

5.3.2. *Resultado*

Los receptores de vídeo reciben señales de vídeo por dos vías diferentes, una señal por cada antena, y se encargan de seleccionar la señal de mejor calidad. En esta prueba se pudo comprobar como la recepción oscilaba continuamente entre una señal y otra. Además, la calidad de la imagen no era buena y aparecerían líneas de interferencia constantemente.

5.4. Software de control – enlace cableado

5.4.1. *Descripción de la prueba*

Esta prueba pretende demostrar el funcionamiento del UAV cuando recibe órdenes de control desde un mando conectado a la estación base. Es decir, se va a probar el funcionamiento del software de control y la respuesta que ofrece el autopilot a sus órdenes. El modo de vuelo es MANUAL en todo momento.

Se conecta Pixhawk al ordenador utilizando dos cables FTDI: el primero se utiliza para el enlace de telemetría y control, y el segundo para poder acceder a la línea de comandos de Pixhawk a través del ordenador. La alimentación de Pixhawk se realiza con una fuente de alimentación a 5V.

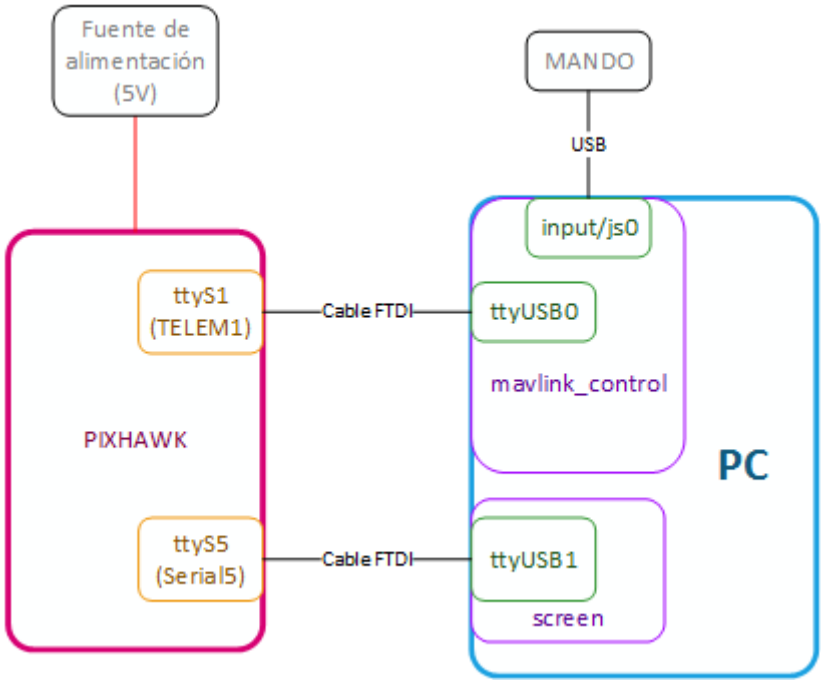


Figura 5-2. Diagrama de conexión - prueba de software 1

Se abren dos terminales, en uno de ellos se ejecuta la aplicación de control, *mavlink_control*, y en el otro se ejecuta *screen*.

Se enciende Pixhawk y en el terminal conectado mediante *screen* se puede ver todo el proceso de inicialización de Pixhawk y la comprobación de los sensores. A continuación se realiza la secuencia de armado y se comprueba que indica el cambio de estado en el mismo terminal. Se inicia la aplicación *print_controls* indicando como argumento que se desean ver los datos de control manual:

```
-$  print_controls manual
```

Se observan tres grupos de datos que corresponden con los valores leídos de los tópicos: ***manual_control_setpoint***, ***actuator_controls*** y ***actuator_outputs***.

A continuación se realizan distintas pruebas:

1.- Sin dar ninguna orden con el mando, se inclina y se rota ligeramente Pixhawk.

2.- Sin mover Pixhawk se prueba el efecto de los controles del mando:

Si no se indica nada en el mando, los valores del tópico ***manual_control_setpoint*** permanecen a 0, ya que no se está indicando ninguna orden de control.

Además, con Pixhawk inmóvil, los valores del tópico ***actuator_controls*** deben ser:

- Throttle = 0 – No se indica ninguna aceleración

- Roll = 0 – Con Pixhawk en posición horizontal

- Pitch = 0 – Con Pixhawk en posición horizontal

- Yaw = El valor normalizado del ángulo formado con la dirección frontal en el momento que se enciende el sistema.

El tópico ***actuator_outputs*** muestra los valores PWM que se envían a cada uno de los motores. Cuando el cuadricóptero no está armado el valor de todos los motores es 900, un ancho de pulso lo suficientemente pequeño como para no inducir ningún movimiento. Cuando el sistema está armado, los valores oscilan entre 1075 y 2000.

5.4.2. *Resultado*

Cuando Pixhawk se encuentra inmóvil y el sistema está desarmado, se obtienen los siguientes datos:

```
[print_manual_ctrl]MANUAL:
Tiempo: 80429.0000
Roll:     0.0000
Pitch:    0.0000
Yaw:      0.0000
Throttle:         0.0000
Cam. pan:         0.0000
Cam. tilt:        0.0000
[print_manual_ctrl]CONTROLS:
Roll:     0.0207
Pitch:    0.0493
Yaw:     -0.0001
Throttle:         0.0000
Cam. tilt:        0.0000
Cam. pan:         0.0000
[print_manual_ctrl]OUTPUTS:
Motor1: 900.000000
Motor2: 900.000000
Motor3: 900.000000
Motor4: 900.000000
Cam. tilt:        0.0000
Cam. pan:         0.0000
```

En el tópico ***manual_control_setpoint*** los valores que se publican son todos 0. En el tópico ***actuator_controls*** se publican los valores que se deberían enviar actualmente a los motores para que Pixhawk permanezca completamente en horizontal. Estos valores deberían ser 0, pero toman valores distintos debido a que Pixhawk no se encontraba en posición completamente horizontal. Por último, en el tópico ***actuator_outputs*** se publica el valor 900 como ancho de pulso para todos los motores. Este valor no permite que los motores giren y no cambiará hasta que el sistema no esté armado.

Si se pulsa el botón de yaw hacia la derecha hasta que alcanza el valor máximo, cuando el throttle permanece a 0, se arma el sistema. Al armar el sistema se observa cómo cambian los valores de los campos del tópico ***actuator_outputs***. Ahora, envían un pulso de 1075 ms a los motores, lo que indica que se encuentran inmóviles pero preparados para girar en cuanto se indique de forma manual.

```
[print_manual_ctrl]MANUAL:
Tiempo: 11687.0000
Roll:     0.0000
Pitch:    0.0000
Yaw:      0.0000
Throttle:         0.0000
Cam. pan:         0.0000
Cam. tilt:        0.0000
[print_manual_ctrl]CONTROLS:
Roll:     0.0273
Pitch:    0.0464
Yaw:      0.0001
Throttle:         0.0000
Cam. tilt:        0.0000
Cam. pan:         0.0000
[print_manual_ctrl]OUTPUTS:
Motor1: 1075.0000
Motor2: 1075.0000
Motor3: 1075.0000
Motor4: 1075.0000
Cam. tilt:      1500.000000
Cam. pan:       1500.000000
```

Una vez que se ha comprobado que todo funciona como se esperaba se hacen las pruebas indicadas:

1.- Sin dar ninguna orden con el mando, se inclina y se rota ligeramente Pixhawk:

```
Roll:   0
Pitch:  0
Yaw:    0
Throttle:       0
Pan:    0
Tilt:   0
Buttons:        0
```

Teniendo en cuenta los valores del tópico ***actuator_controls,*** Pixhawk se encuentra ligeramente inclinado hacia la izquierda y hacia delante. Al no recibir ninguna orden de forma manual los valores del tópico ***manual_control_setpoint*** no varían.

Al inclinar hacia la derecha se obtiene un valor de roll negativo:

```
[print_manual_ctrl]MANUAL:
Tiempo: 52527.0000
Roll:     0.0000
Pitch:    0.0000
Yaw:      0.0000
Throttle:         0.0000
Cam. pan:         0.0000
Cam. tilt:        0.0000
[print_manual_ctrl]CONTROLS:
Roll:    -0.1359
Pitch:    0.0387
Yaw:     -0.0029
Throttle:         0.0000
Cam. tilt:        0.0000
Cam. pan:         0.0000
[print_manual_ctrl]OUTPUTS:
Motor1: 1075.0000
Motor2: 1075.0000
Motor3: 1075.0000
Motor4: 1075.0000
Cam. tilt:      1500.000000
Cam. pan:       1500.000000
```

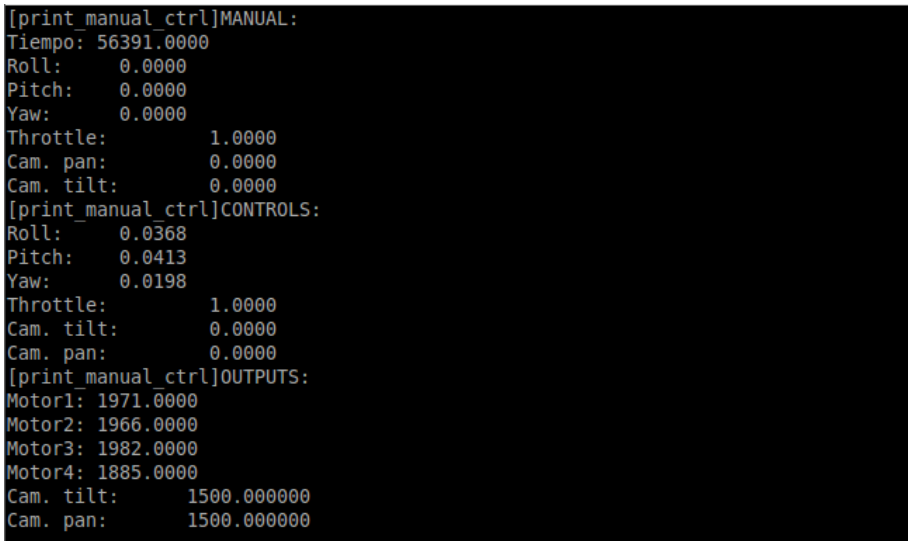
Al inclinar hacia delante se obtiene un valor de pitch positivo mayor al que se tenía de inicio:

```
[print_manual_ctrl]MANUAL:
Tiempo: 92838.0000
Roll:     0.0000
Pitch:    0.0000
Yaw:      0.0000
Throttle:         0.0000
Cam. pan:         0.0000
Cam. tilt:        0.0000
[print_manual_ctrl]CONTROLS:
Roll:    -0.0584
Pitch:    0.2261
Yaw:      0.0022
Throttle:         0.0000
Cam. tilt:        0.0000
Cam. pan:         0.0000
[print_manual_ctrl]OUTPUTS:
Motor1: 1075.0000
Motor2: 1075.0000
Motor3: 1075.0000
Motor4: 1075.0000
Cam. tilt:      1500.000000
Cam. pan:       1500.000000
```

2.- Sin mover Pixhawk se prueba el efecto de los controles del mando:

```
Roll:   0
Pitch:  0
Yaw:    0
Throttle:       1000
Pan:    0
Tilt:   0
Buttons:        0
```

El control principal, que ocasiona que los motores comiencen a girar es el throttle. En la imagen se ha llevado el control de throttle hasta el máximo, como se puede observar en los datos del tópico ***manual_control_setpoint***. La posición de Pixhawk es similar a la de inicio, ligeramente inclinado hacia la derecha y hacia delante. Los datos que se envían a los motores ahora toman valores próximos a 2000 ms, que sería el máximo permitido, pero con ligeras variaciones entre ellos. Se puede observar como el motor que se encontraría en la posición trasera derecha (motor 4), es el que recibe un valor más pequeño, ya que sería el que más elevado se encontraría. Por el contrario el motor frontal a la izquierda es el que más rápido debe girar para ejercer mayor empuje hacia arriba y así conseguir que el cuadricóptero se coloque en posición horizontal.



5.5. Software de control – enlace inalámbrico

5.5.1. *Descripción de la prueba*

De forma similar al apartado anterior se ha probado el funcionamiento del sistema de control antes de conectar los motores, es decir, sin la posibilidad de que el cuadricóptero comience a volar. El objetivo de esta prueba es comprobar que todo funciona exactamente igual que en la anterior a pesar de haber sustituido el enlace de control por uno inalámbrico gestionado por dos módulos XBee.

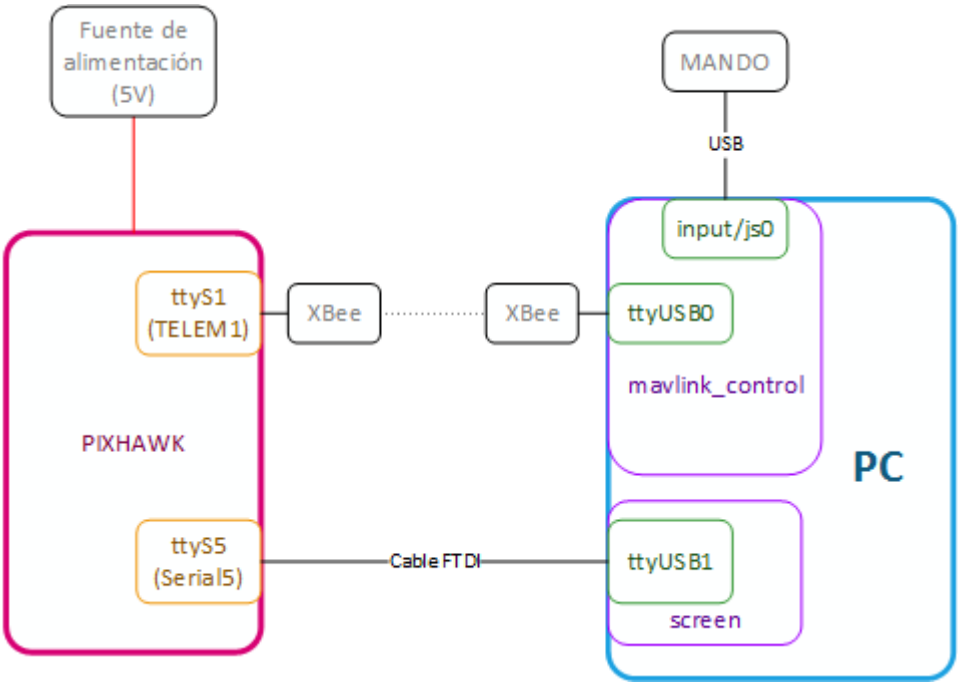


Figura 5-3. Diagrama de conexión - prueba de software 2

5.5.2. *Resultado*

No se ha encontrado ninguna diferencia en el funcionamiento del sistema de control al sustituir el enlace cableado por uno inalámbrico. Los módulos XBee situados en ambos extremos del enlace permiten mantener una comunicación de forma completamente transparente, como si de un cable serie se tratara.

5.6. Sistema de control y telemetría

5.6.1. *Descripción de la prueba*

Una vez comprobado que el software de control funciona correctamente se ha procedido a probar el funcionamiento del enlace conjunto de control y telemetría. El esquema de conexiones en este caso es el siguiente:

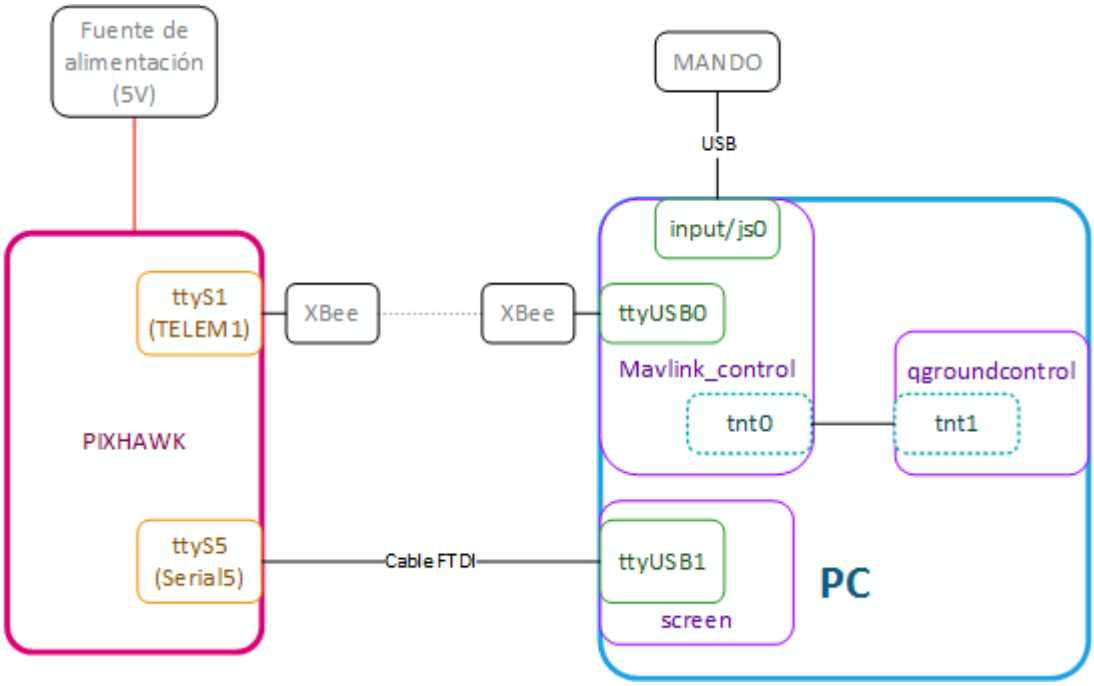


Figura 5-4. Diagrama de conexión - prueba de software 3

En el software de control se ha creado una función que permite registrar los números de secuencia, IDs y el instante en que se recibe y envía cada mensaje para tener más información de cómo transcurre la comunicación. Se registran tres grupos de mensajes distintos: por un lado, los mensajes de telemetría que envía Pixhawk (rojo), los mensajes de señal de vida que envía QGroundControl (azul) y los mensajes de control que envía *mavlink_control* (verde).

5.6.2. *Resultado*

En la captura del tráfico de paquetes se puede ver algo más de un segundo de transmisión, donde se observa el intercambio de mensajes entre el UAV y la estación base. Hay que tener en cuenta que desde QGroundControl se envía otro tipo de mensajes si se requiere por el usuario: cambios de modo, parámetros de las misiones, etc.

En la imagen se pueden ver los mensajes que se intercambian para visualizar la telemetría y controlar el UAV en modo manual.

Se puede observar cómo los mensajes de señal de vida de QGroundControl (azul) se envían cada segundo y los mensajes de control (verde) se envían cada 0.1 segundos. Por otra parte, se reciben 9 mensajes distintos de telemetría, en intervalos de 0.1, 0.3 y 1 segundo respectivamente.

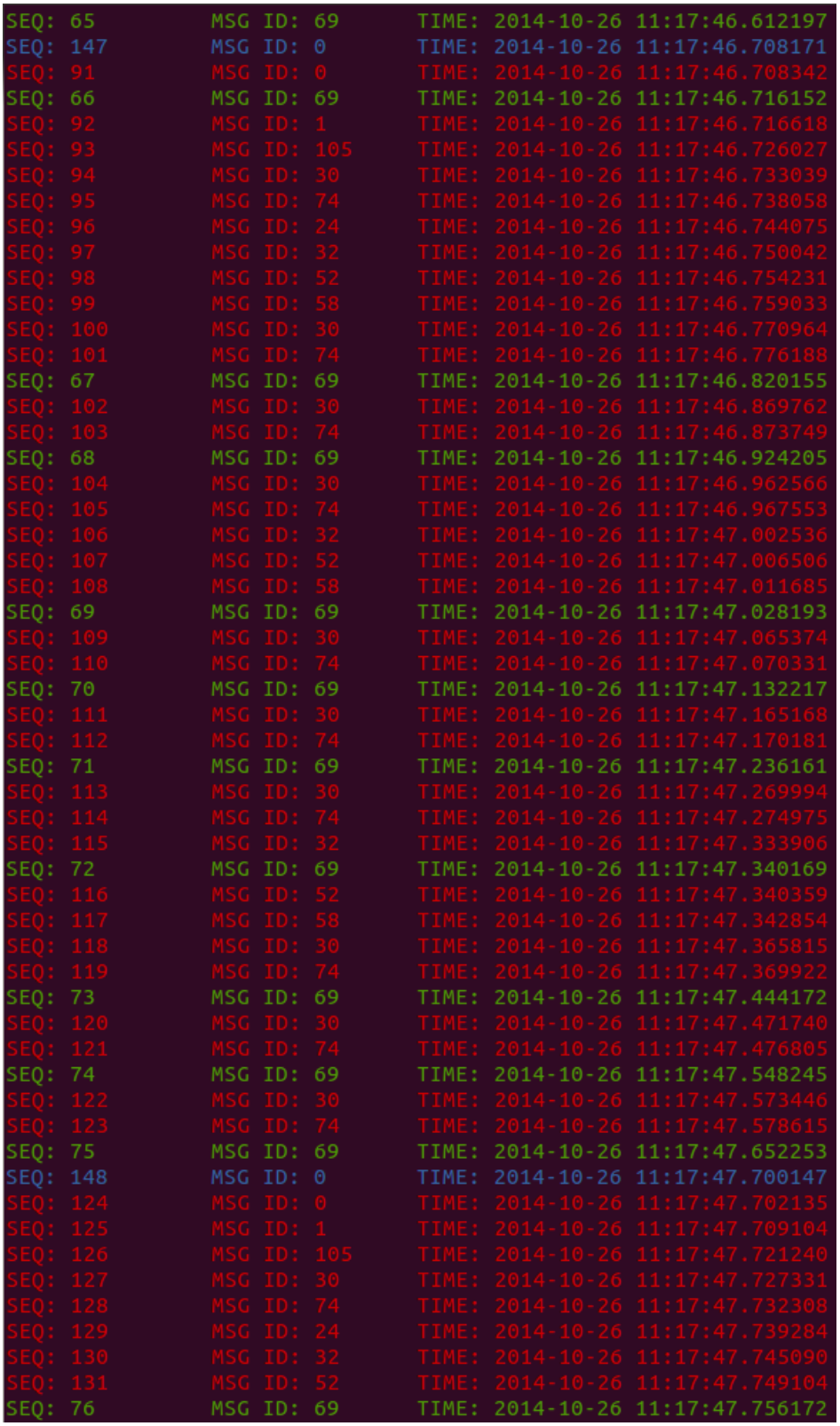


Figura 5-5. Tráfico de datos

5.7. Controles y modos de vuelo

5.7.1. *Descripción de la prueba*

Antes de probar el cuadricóptero con los motores, es decir, sin permitir que vuele, se ha comprobado que la respuesta al mando es la correcta. Para ello se utiliza de nuevo el esquema de conexión del apartado anterior. La respuesta a los controles se probó en los primeros apartados, pero se repite para comprobar que no haya cambiado nada al integrar el enlace de telemetría, además, se comprueba que el cambio de modo de vuelo se puede realizar sin problemas con el mando. Para probar alguno de los modos de vuelo es necesaria señal GPS, por lo que la prueba se ha realizado en el exterior.

El proceso de cambio de cambio de vuelo se produce cuando el UAV recibe un mensaje "mavlink_msg_manual_control" indicando que se ha pulsado algún botón. Este mensaje se decodifica en la aplicación *mavlink* quien publica el cambio en el tópico ***manual_control_setpoint***. La aplicación *commander* se subscribe a este tópico e intenta realizar el cambio de modo. Si no hay ninguna causa que impida el cambio lo realiza e informa al resto del sistema mediante los tópicos ***vehicle_status*** y ***vehicle_control_mode***. Cuando la aplicación *mavlink* constata que el cambio de modo se ha realizado, mediante el tópico ***vehicle_status***, modifica la información que envía en cada mensaje "mavlink_msg_heartbeat" para informar del nuevo modo de vuelo a la estación base.

5.7.2. *Resultado*

Los resultados de las pruebas de control son idénticos a los anteriores:

1.- Sin dar ninguna orden con el mando, se inclina y se rota ligeramente Pixhawk:

En este caso, al tener información de la telemetría en QGC, se puede observar la orientación del cuadricóptero en el display izquierdo de la pantalla "fligth".

2.- Sin mover Pixhawk se prueba el efecto de los controles del mando:

En QGroundControl no se observa ninguna modificación en la orientación ya que al no tener el sistema completo conectado el cuadricóptero no se mueve y lo único que cambia son los valores deseados de orientación.

3.- Cambio de modos de vuelo:

A continuación se muestra el proceso de cambio de modo MANUAL a ALTCTL:

- Cómo se indicó en el apartado 4.3.1., el número enviado en el campo "buttons" del mensaje "mavlink_msg_manual_control" sugiere el cambio a modo ALTCTL:

```
Roll:   0
Pitch:  0
Yaw:    0
Throttle:       0
Pan:    0
Tilt:   0
Buttons:        16
```

- Cuando Pixhawk recibe el mensaje y acata el cambio, la aplicación *commander* informa por línea de comandos, lo que se puede ver a través de *screen*:

```
nsh>
nsh> commander: main state: ALTCTL
commander: nav state: ALTCTL
```

- Además, la aplicación *mavlink* informa mediante el enlace de telemetría a la estación base, como se puede comprobar en la información que muestra QGroundControl en la parte superior:

El proceso es similar para el resto de modos de vuelo siempre y cuando el UAV reciba señal de GPS.

6 Conclusiones y trabajo futuro

6.1. Conclusiones

En este proyecto se ha creado un sistema de teleoperación para un UAV de tipo cuadricóptero. Para ello se han diseñado los enlaces inalámbricos para la transmisión de señal de vídeo y de datos, para lo cual se ha estudiado la amplia oferta de módulos disponibles en el mercado y se ha seleccionado el conjunto adecuado para la fase de desarrollo inicial del proyecto. Para conseguir un control fiable se ha utilizado el protocolo de comunicación MAVLink. Se ha seleccionado un controlador de vuelo (autopilot) comercial que utilizaba este mismo sistema para transmisión de la telemetría a la estación base y se ha adaptado para conseguir que también interprete las órdenes de control utilizando el mismo protocolo. Se ha estudiado su arquitectura software y descrito sus módulos principales de manera que sea posible la intervención dentro del código fuente para agregar/modificar funcionalidades según lo requiera el proyecto. Se ha implementado un conjunto de modificaciones que permite sustituir el control remoto de la aeronave basado en un mando RC comercial estándar por un mando digital conectado al ordenador en la estación base, abriendo la posibilidad de comunicación de comandos de vuelo generados automáticamente en la estación base. Además se ha creado una aplicación de control para enviar los comandos de control de vuelo desde la estación base.

Se ha conseguido un cuadricóptero capaz de permanecer en vuelo más de 30 minutos y mantener la comunicación con la estación base en distancias superiores a un kilómetro en espacio abierto. El cuadricóptero es capaz de realizar distintos modos de vuelo, como son:

- **MANUAL**: Todos los controles del mando se envían a los motores tras pasar por el controlador de estabilidad que evita cambios bruscos que puedan provocar la pérdida de control.
- **ALTCTL**: Los controles siguen siendo manuales, pero el control del throttle tiene una zona muerta muy amplia, en la cual el cuadricóptero permanece a una altura constante.
- **POSCTL**: Muy similar al modo de control anterior, pero en este caso la zona muerta también se produce en los controles de roll y pitch, es decir, movimientos izquierda/derecha y delante/detrás.

- **AUTO/LOITER:** El cuadricóptero permanece suspendido en el aire a una altura constante que se indica como parámetro de configuración.
- **AUTO/RTL:** Las siglas RTL responden a la expresión en inglés "Return To Launch", es decir, el cuadricóptero regresa a la posición donde despegó. Este vuelo se hace de forma automática en línea recta y a una altura igual a la de Loiter o igual a la altitud de la estación base más la altura de Loiter si la altitud de la estación base es mayor.
- **AUTO/MISSION:** Este modo de vuelo permite al UAV realizar misiones de vuelo autónomo mediante el seguimiento de waypoints indicados desde la estación base.

6.2. Trabajo futuro

En este proyecto se ha diseñado un sistema de comunicación para un UAV y se ha creado un programa de prueba para enviar los comandos de control de vuelo. A lo largo del proyecto se han identificado características que serían favorable para el diseño, pero no se ha podido desarrollar todo; por lo tanto, algunas de estas características a implementar son:

- **Eliminar el botón "safety":**

Como medida de seguridad, Pixhawk incluye un botón que tiene que ser pulsado manualmente antes de permitir armar el sistema. Es decir, si no se pulsa ese botón, la secuencia de armado no hace efecto y los motores no girarán.

El control de ese botón se realiza en el firmware de PX4IO, y no se ha podido modificar correctamente.

- **Enlace de vídeo digital:**

Al realizar las pruebas de los sistemas de transmisión de vídeo, se ha notado que la calidad de la imagen podría ser bastante mejorable, para lo cual lo mejor sería utilizar transmisores y receptores digitales.

- **Software de gestión de telemetría propio:**

Como se ha explicado a lo largo de la memoria, para gestionar la telemetría que se recibe en la estación base se ha utilizado un software existente llamado QGroundControl. Este software decodifica los mensajes MAVLink recibidos y tiene una interfaz gráfica que permite visualizar los datos recibidos de forma sencilla. El inconveniente es que no permite enviar comandos de control desde un mando conectado al ordenador. Para solucionar este problema se ha creado una aplicación que envía dichos comandos.

Como trabajo futuro se plantea la creación de un software propio de gestión de telemetría que incluya todas las funciones necesarias.

- **Integración del control de la cámara inferior (ON/OFF, ZOOM):**

La cámara que se va a situar en la parte inferior del cuadricóptero es un modelo utilizado en vigilancia, donde es necesario un control remoto de las cámaras de video. Para poder controlar y configurar la cámara de vídeo cuenta con un puerto de comunicación serie.

Después de configurar todos los dispositivos utilizados en este proyecto, en Pixhawk ha quedado libre el puerto Serial4, que corresponde a la ruta /dev/ttyS6.

Será necesario crear un driver que gestione la comunicación con la cámara de vídeo a través de este puerto utilizando el protocolo propio de Sony. Además, es necesario modificar la aplicación mavlink y crear un tópico al que se subscriba el driver de la cámara. De esta forma, se podrán utilizar mensajes MAVLink para enviar las órdenes de configuración y control de la cámara desde la estación base.

Referencias

[1] «Robomotion,» 2014. [En línea]. Available: http://www.robomotion.es/.

[2] S. F. A. D. H. M. H. T. a. F. A. W. M. Kamran Joyo, "Position controller design for quad-rotor under perturbed condition," *Wulfenia Journal,* vol. 20, no. 7, pp. 178-189, 2013.

[3] D. G. A. S. K. Shahida Khatoon, «Assembly of an experimental quad-rotor type UAV for testing a novel autonomous flight control strategy,» *International Journal of Advanced Computer Research,* vol. 3, nº 4, 2013.

[4] W. Z. a. M. X. Haitao Jia, "UAV search planning based on perceptual cue," in *IEEE International Workshop on Microwave and Millimeter Wave Circuits and System Technology*, Chengdu, China, 2013.

[5] R. B. A. V. a. B. P. Hrishikesh Sharma, "Vision-based detection of power distribution lines in complex remote surroundings," in *IEEE Twentieth National Conference on Communications (NCC),* Kanpur, India, 2014.

[6] D. H. S. F. A. M. K. J. F. A. W. M. Hassan Tanveer, "Design of overall stabilized controller for quad-rotor," *Wulfenia Journal,* vol. 20, no. 7, pp. 212-229, 2013.

[7] J. S. D. G. a. K.-M. D. Michael C. Achtelik, "Design of a flexible high performance quadcopter platform breaking the MAV endurance record with laser power beaming," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* San Francisco, CA, USA, 2011.

[8] a. P. C. Inkyu Sa, "Vertical infrastructure inspection using a quadcopter and shared autonomy control," in *Field and Service Robotics - Results of the 8th International Conference*, Brisbane, Australia, Springer Berlin Heidelberg, 2014.

[9] S. M. L. M. P.-L. O. Bernard Tat Meng Leong, «Low-cost microcontroller-based hover control design of a quadcopter,» *ELSEVIER International Symposium on Robotics and Intelligent Sensors (IRIS),* vol. 41, pp. 458-464, 2012.

[10] A. E. D. L. R. L. C. P. Luis Rodolfo García Carrillo, Quad Rotorcraft Control - Vision-Based Hovering and Navigation, London: Springer, 2013.

[11] R. Lozano, Unmanned Aerial Vehicles - Embedded Control, London, Hoboken: ISTE Ltd, John Wiley & Sons, Inc., 2010.

[12] A. E. D. L. R. L. C. P. Luis Rodolfo García Carrillo, «Quad Rotorcraft Control - Vision-Based Hovering and Navigation,» London, Springer, 2013, pp. 23-35.

[13] S. Fux, *Development of a planar low cost Inertial Measurement Unit for UAVs and MAVs,* Zurich, 2008.

[14] C. C. L. D. a. Y. C. Haiyang Chao, «A Comparative Evaluation of Low-Cost IMUs for Unmanned Autonomous Systems,» de *International Conference on Multisensor Fusion and Integration for Intelligent Systems*, Salt Lake City, 2010.

[15] R. Lozano, «Unmanned Aerial Vehicles - Embedded control,» Wiley, 2010, pp. 265-284.

[16] M. R. R. M. H. N. S. M. S. Hossein Bolandi, «Attitude Control of a Quadrotor with Optimized PID Controller,» *Intelligent Control and Automation,* vol. 4, pp. 335-342, 2012.

[17] R. Lozano, «Unmanned Aerial Vehicles - Embedded control,» Wiley, 2010, pp. 140-142.

[18] K. Ogata, Ingeniería de control moderna, Pearson Educación, 1998.

[19] «http://www.futaba-rc.com/sbus/,» 2014. [En línea].

[20] «http://www.spektrumrc.com/,» [En línea].

[21] "PX4 Autopilot Platform," [Online]. Available: http://pixhawk.org/. [Accessed 2014].

[22] «MAVLink,» 2014. [En línea]. Available: http://qgroundcontrol.org/mavlink/start.

[23] «Computer Vision and Geometry Lab,» [En línea]. Available: http://cvg.ethz.ch/.

[24] «Autonomous Systems Lab,» 2014. [En línea]. Available: http://www.asl.ethz.ch/.

[25] «Automatic Control Laboratory,» [En línea]. Available: http://control.ee.ethz.ch/.

[26] «Github_PX4,» 2014. [En línea]. Available: https://github.com/PX4.

[27] «NuttX,» 2014. [En línea]. Available: http://www.nuttx.org/.

[28] «ROS,» 2014. [En línea]. Available: http://www.ros.org/.

[29] J. A. Benito, *Integración de un UAV en la plataforma robótica ARGOS*, Madrid, 2015.

Glosario

ADC	Analog to Digital Converter
CAN	Controller Area Network
COFDM	Coded Orthogonal Frequency Division Multiplexing
DSM	Digital Spectrum Modulation
DSSS	Direct Sequencing Spread Spectrum
FHSS	Frequency Hopping Spread Spectrum
FM	Frequency Modulation
FMU	Flight Management Unit
FPV	First Person View
FTDI	Future Technology Devices International
GPS	Global Positioning System
I2C	Inter-Integrated Circuit
IMU	Inertial Measurement Units
IO	Input/Output
LGPL	Lesser General Public License
LHCP	Left Hand Circular Polarization
LiPo	Lithium Polymer
LOS	Line Of Sight
MAV	Micro Aerial Vehicle
MEMS	Micro Electro Mechanical System
OSD	On Screen Display
PAL	Phase Alternating Line
PID	Proportional, Integral, Derivative
POSIX	Portable Operating System Interface UniX
PPM	Pulse Position Modulation
PWM	Pulse Width Modulation
RHCP	Right Hand Circular Polarization

RSSI	Receive Signal Strength Indicator
SDL	Simple DirectMedia Layer
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Aerial Vehicle
uORB	micro Object Request Broker
VTOL	Vertical Take Off and Landing
XPD	Cross Polarization Discrimination

ANEXOS

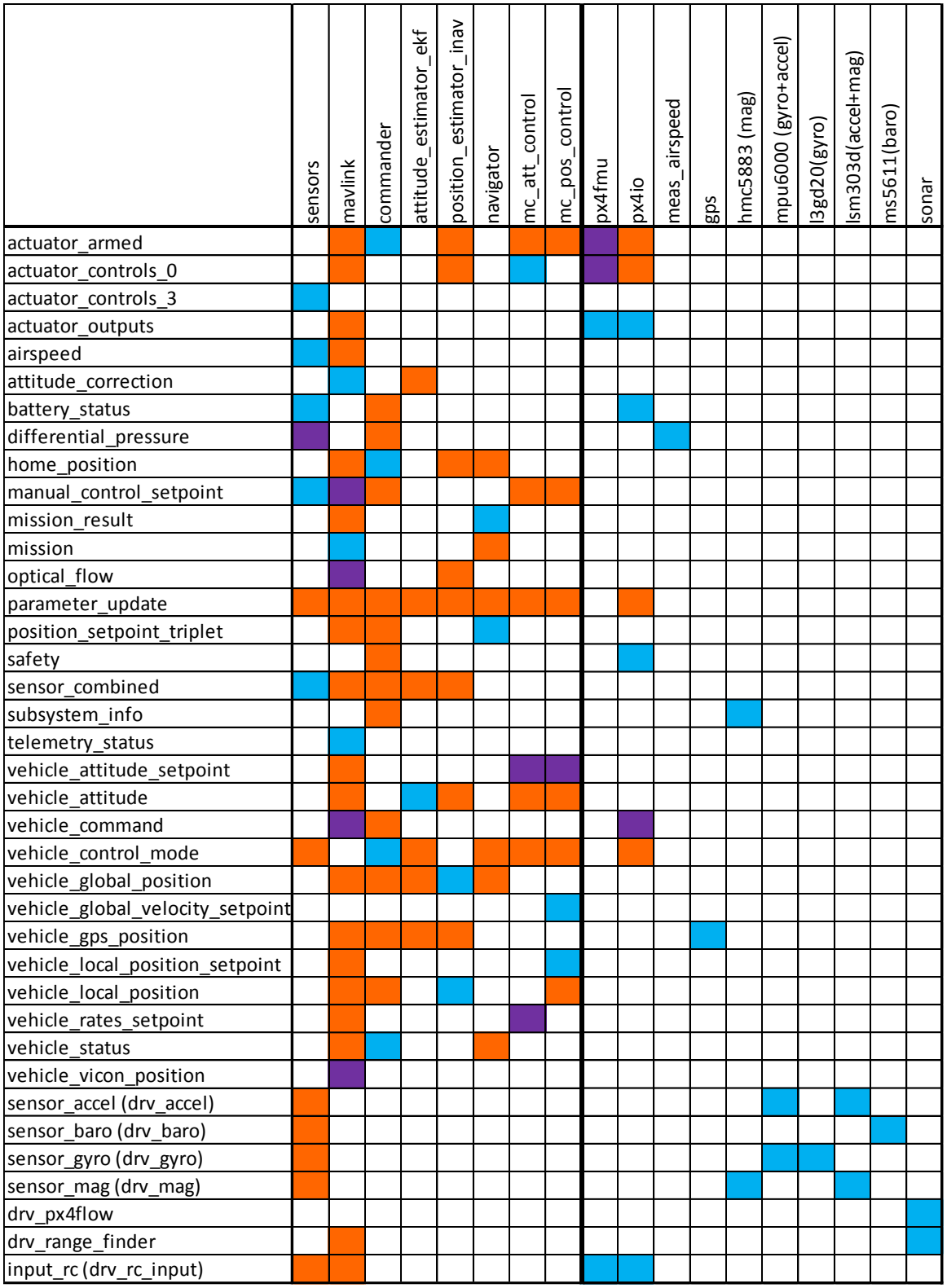
A Suscripciones y publicaciones en los tópicos

La siguiente tabla incluye todas las aplicaciones que intervienen en el controlador de vuelo utilizado para manejar el cuadricóptero, indicando en cada caso los tópicos en los que publica y se subscribe cada una de ellas.

Cada columna de la tabla corresponde a una aplicación o driver, y cada corresponde con uno de los tópicos utilizados.

Se ha indicado en naranja los tópicos a los que se subscribe cada aplicación, en azul los tópicos en los que publica cada aplicación y en morado los tópicos en los que una aplicación se subscribe y publica.

Tabla A-1. Subscripciones y publicaciones en los tópicos



B Tópicos

La siguiente tabla incluye una descripción de todos los tópicos utilizados en el controlador de vuelo de un cuadricóptero, así como una descripción de cada uno de los campos que componen cada tópico.

TÓPICO	***Descripción del tópico***	
	Campos	Descripción de los campos
actuator_armed	***Información sobre el estado de armado del vehículo.***	
	`bool armed`	Verdadero si el sistema está armado
	`bool ready_to_arm`	Verdadero si el sistema está listo para ser armado
	`bool lockdown`	Verdadero para forzar a los actuadores a permanecer deshabilitados
	`bool force_failsafe`	Verdadero para forzar a los actuadores a permanecer en estado de prueba de fallos
actuator_controls	***Salidas del sistema de control del vehículo, y son los valores que serán mezclados para obtener el valor de control de los actuadores (servos, motores, etc.) que operan el vehículo.***	
	`float control [NUM_ACTUATOR_CONTROLS]`	Datos de entradas al mezclador
actuator_outputs	***Salidas del mezclador tal y como se envían a los actuadores (servos, motores, etc.) que operan el vehículo.***	
	`float output [NUM_ACTUATOR_OUTPUTS]`	Datos de salida del mezclador, en unidades naturales
	`unsigned noutputs`	Número de salidas válidas
airspeed	***Información del sensor de velocidad de aire.***	
	`float indicated_airspeed_m_s`	Velocidad del aire indicada en m/s, -1 si se desconoce
	`float true_airspeed_m_s`	Velocidad del aire verdadera en m/s, -1 si se desconoce
	`float air_temperature_celsius`	Temperatura en grados Celsius, -1000 si se desconoce
attitude_ correction	***Valores de corrección de la orientación horizontal para calibración manual.***	
	`float roll;`	
	`float pitch;`	
battery_status	***Voltaje y estado de la batería.***	
	`float voltage_v`	Voltaje de la batería en V, 0 si se desconoce

	`float voltage_filtered_v`	Voltaje de la batería en V, filtrado, 0 si se desconoce
	`float current_a`	Corriente de la batería en A, -1 si se desconoce
	`float discharged_mah`	Cantidad descargada en mAh, -1 si se desconoce
differential_ pressure	***Presión diferencial medida por el sensor airspeed.***	
	`uint64_t error_count`	Número de errores detectados por el driver
	`float differential_ pressure_pa`	Presión diferencial leída del sensor
	`float differential_ pressure_raw_pa`	Valor bruto de la presión diferencial leído en Pa, puede ser negativo
	`float differential_ pressure_filtered_pa`	Presión diferencial leída filtrada, en Pa
	`float max_differential_ pressure_pa`	Valor máximo de presión diferencial leída, en Pa
	`float temperature`	Temperatura medida por el sensor, -1000.0f si se desconoce
home_position	***Posición GPS de la estación base en coordenadas WGS84.***	
	`double lat`	Latitud en grados
	`double lon`	Longitud en grados
	`float alt`	Altitud en metros
	`float x`	Coordenada X en metros
	`float y`	Coordenada Y en metros
	`float z`	Coordenada Z en metros
manual_control_ setpoint	***Valores de control manual del UAV. Asignación directa de los datos recibidos de la estación base mediante el mensaje MAVLink manual_control.***	
	`float x`	Posición del eje de control en dirección x – -1...1 – Movimiento hacia delante/detrás o pitch del vehículo. Un valor positivo indica hacia delante y negativo hacia detrás
	`float y`	Posición del eje de control en dirección x – -1...1 – Movimiento hacia derecha/izquierda o roll del vehículo. Un valor positivo indica hacia derecha y negativo hacia izquierda
	`float z`	Posición del eje de aceleración o throttle – 0...1 – Movimiento hacia arriba/abajo o empuje del vehículo. El valor corresponde con la aceleración demandada por el usuario.
	`float r`	Posición del eje de control de rotación o yaw – -1...1 – Rotación hacia la derecha en torno al eje vertical del vehículo.
	`float flaps`	No se usa en cuadricópteros.

	`float aux1`	Movimiento derecha/izquierda o pan de la cámara – -1...1.
	`float aux2`	Movimiento arriba/abajo o tilt de la cámara – -1...1.
	`float aux3`	No se utiliza
	`float aux4`	No se utiliza
	`float aux5`	No se utiliza
	`uint8_t buttons;`	Equivalente al campo buttons del mensaje MAVLink manual_control
	`switch_pos_t mode_switch`	Interruptor de modo principal de 3 posiciones: MANUAL, ASSIST (ALTCTL o POSCTL) y AUTO (RTL, LOITER o MISSION)
	`switch_pos_t return_switch`	Interruptor de RTL de 2 posiciones: NORMAL (LOITER o MISSION) y RTL
	`switch_pos_t posctl_switch`	Interruptor de control de posición de 2 posiciones: ALTCTL y POSCTL
	`switch_pos_t loiter_switch`	Interruptor de loiter de 2 posiciones: MISSION y LOITER
	`switch_pos_t acro_switch`	No usado – Para vuelo acrobático
	`switch_pos_t offboard_switch`	No usado – Para controladores de vuelo externos al UAV
mission_result	***Resultado de la misión. Información que el controlador de navegación transmite a las aplicaciones commander y mavlink.***	
	`unsigned seq_reached`	Secuencia del punto de la misión que se ha alcanzado
	`unsigned seq_current`	Secuencia del punto de la misión que se está realizando.
	`bool reached`	Verdadero si se ha conseguido la misión
	`bool finished`	Verdadero si se ha terminado la misión
mission	***Posición que el UAV tiene que alcanzar. Se utilizan coordenadas globales WGS84.***	
	`bool altitude_is_relative`	Verdadero si la altitud es relativa al punto de inicio
	`double lat`	Latitud en grados
	`double lon`	Longitud en grados
	`float altitude`	Altitud en grados
	`float yaw`	Rotación en radianes –Pi…+Pi, en coordenadas NED, NaN indica que no varía el yaw
	`float loiter_radius`	0 en multicópteros para que permanezcan suspendidos.
	`int8_t loiter_direction`	No se utiliza (1 sentido de las agujas de reloj, -1 sentido opuesto)
	`enum NAV_CMD nav_cmd`	Comando de navegación
	`float acceptance_radius`	Radio de aceptación en metros, dentro del cual se considera que se ha alcanzado la misión

	`float time_inside`	Tiempo que el UAV debe permanecer dentro del radio de aceptación antes de continuar, en segundos
	`float pitch_min`	No se utiliza (mínimo ángulo de pitch para despegue de UAVs de ala fija)
	`bool autocontinue`	Verdadero si se debe continuar al siguiente waypoint nada más alcanzar este
	`enum ORIGIN origin`	Origen del waypoint, dónde ha sido generado
	`int do_jump_mission_index`	Índice de la misión donde debe ir el salto
	`unsigned do_jump_repeat_count`	Número de veces que se tiene que repetir el salto
	`unsigned do_jump_current_count`	Número de veces que se ha realizado el salto
optical_flow	***No se utiliza el sensor óptico, pero el tópico se usa para publicar la distancia al suelo obtenida por el sonar.***	
	`uint64_t flow_timestamp`	
	`int16_t flow_raw_x`	
	`int16_t flow_raw_y`	
	`float flow_comp_x_m`	
	`float flow_comp_y_m`	
	`float ground_distance_m`	Altitud, distancia al suelo medida en metros
	`uint8_t quality`	Calidad de la medida, 0: mala calidad, 255: máxima calidad
	`uint8_t sensor_id`	ID del sensor
parameter_ update	***Instante de tiempo en que se ha actualizado el último parámetro.***	
	`uint64_t timestamp`	

position_setpoint_ triplet	***Punto de referencia de la posición global en coordenadas WGS84.***
	El tópico contiene información de tres posiciones, la actual, la anterior y la próxima. Además de informar cual es el estado de navegación.
	
	`struct position_setpoint_s previous;`
	`struct position_setpoint_s current;`
	`struct position_setpoint_s next;`
	`unsigned nav_state;`
	
	Se detallan los campos de las estructuras de posición.	

	`bool valid`	Verdadero si el punto de referencia es válido
	`enum SETPOINT_TYPE type`	Tipo del punto de referencia para ajustar el comportamiento del controlador de posición
	`float x`	Posición local X en metros usando coordenadas NED

	`float y`	Posición local Y en metros usando coordenadas NED
	`float z`	Posición local Z en metros usando coordenadas NED
	`bool position_valid`	Verdadero si el punto de referencia de posición local es válido
	`float vx`	Velocidad local en X en m/s
	`float vy`	Velocidad local en Y en m/s
	`float vz`	Velocidad local en Z en m/s
	`bool velocity_valid`	Verdadero si el punto de referencia de velocidad local es válido
	`double lat`	Latitud en grados
	`double lon`	Longitud en grados
	`float alt`	Altitud AMSL en m
	`float yaw`	Rotación en radianes [-Pi...Pi)
	`float yawspeed`	Velocidad de giro en rad/s
	`float loiter_radius`	No se utiliza. (Radio de loiter en m)
	`int8_t loiter_direction`	No se utiliza. (1 = CW, -1 = CCW)
	`float pitch_min`	No se utiliza (mínimo ángulo de pitch para despegue de UAVs de ala fija)
safety	***Tópico para informar del estado del botón de seguridad desde el driver de la placa px4io a commander.***	
	`bool safety_switch_available`	Verdadero si el botón de seguridad está conectado
	`bool safety_off`	Verdadero si el botón de seguridad está en posición OFF
satellite_info	***Información de satélites GNSS***	
	`uint8_t count`	Número de satélites
	`uint8_t svid[SAT_INFO_MAX_SATELLITES]`	ID de los satélites: GPS 1-32 SBAS 120-158 Galileo 211-246 BeiDou 159-163, 33-64 QZSS 193-197 GLONASS 65-96, 255
	`uint8_t used [SAT_INFO_MAX_SATELLITES]`	0: satélite no usado 1: satélite usado para navegación
	`uint8_t elevation[SAT_INFO_MAX_SATELLITES]`	Elevación del satélite: 0: Justo encima del receptor 90: En el horizonte
	`uint8_t azimuth[SAT_INFO_MAX_SATELLITES]`	Dirección del satélite: 0: 0 grados 255: 360 grados
	`uint8_t snr[SAT_INFO_MAX_SATELLITES]`	SNR (Relación señal a ruido) en dBHz del satélite, [0...99]. 0 cuando no se está siguiendo ese satélite

sensor_combined	***Valores brutos y en unidades del Sistema internacional de los datos leídos de todos los sensores.*** ***Se puede incluir información de hasta 3 sensores del mismo tipo.***	
	`uint64_t timestamp`	Instante de tiempo en µs de la medida del giroscopio
	`int16_t gyro_raw[3]`	Valor bruto de la velocidad angular medido por el sensor
	`float gyro_rad_s[3]`	Velocidad angular en rad/s
	`int16_t accelerometer_raw[3]`	Valor bruto de la aceleración en sistema de referencia NED
	`float accelerometer_m_s2[3]`	Aceleración en m/s^2
	`int accelerometer_mode`	Modo de medida del acelerómetro
	`float accelerometer_ range_m_s2`	Rango de medida del acelerómetro en m/s^2
	`uint64_t accelerometer_timestamp`	Instante de tiempo de medida del acelerómetro
	`int16_t magnetometer_raw[3]`	Valor bruto del campo magnético en sistema de referencia NED
	`float magnetometer_ga[3]`	Campo magnético en Gauss
	`int magnetometer_mode`	Modo de medida del magnetómetro
	`float magnetometer_range_ga`	Rango de medida del magnetómetro en Gauss
	`float magnetometer_cuttoff_fr eq_hz`	Frecuencia de corte del filtro paso bajo analógico interno del magnetómetro
	`uint64_t magnetometer_timestamp`	Instante de tiempo de medida del magnetómetro
	`int16_t gyro1_raw[3]`	Valor bruto de la velocidad angular
	`float gyro1_rad_s[3]`	Velocidad angular en rad/s
	`uint64_t gyro1_timestamp`	Instante de tiempo en µs de la medida del giroscopio
	`int16_t accelerometer1_raw[3]`	Valor bruto de la aceleración en sistema de referencia NED
	`float accelerometer1_m_s2[3]`	Aceleración en m/s^2
	`uint64_t accelerometer1_ timestamp`	Instante de tiempo de medida del acelerómetro
	`int16_t magnetometer1_raw[3]`	Valor bruto del campo magnético en sistema de referencia NED
	`float magnetometer1_ga[3]`	Campo magnético en Gauss
	`uint64_t magnetometer1_timestamp`	Instante de tiempo de medida del magnetómetro
	`int16_t gyro2_raw[3]`	Valor bruto de la velocidad angular
	`float gyro2_rad_s[3]`	Velocidad angular en rad/s

	uint64_t gyro2_timestamp	Instante de tiempo en µs de la medida del giroscopio
	int16_t accelerometer2_raw[3]	Valor bruto de la aceleración en sistema de referencia NED
	float accelerometer2_m_s2[3]	Aceleración en m/s^2
	uint64_t accelerometer2_ timestamp	Instante de tiempo de medida del acelerómetro
	int16_t magnetometer2_raw[3]	Valor bruto del campo magnético en sistema de referencia NED
	float magnetometer2_ga[3]	Campo magnético en Gauss
	uint64_t magnetometer2_timestamp	Instante de tiempo de medida del magnetómetro
	float baro_pres_mbar	Presión barométrica en mbar
	float baro_alt_meter	Altitud en metros
	float baro_temp_celcius	Temperatura en grados Celsius
	float adc_voltage_v[10]	Voltajes del conversor ADC
	unsigned adc_mapping[10]	Índice de canales de cada uno de los valores
	float mcu_temp_celcius	Temperatura interna medida por la MCU en grados Celsius
	uint64_t baro_timestamp	Instante de tiempo de medida del barómetro
	float differential_ pressure_pa	Presión diferencial en Pascales
	uint64_t differential_ pressure_timestamp	Instante de tiempo de medida del sensor airspeed
	float differential_ pressure_filtered_pa	Presión diferencial filtrada paso bajo en Pa
subsystem_info	***Estado de los subsistemas.***	
	bool present	Verdadero si está presente
	bool enabled	Verdadero si está habilitado
	bool ok	Verdadero si funciona correctamente
	enum SUBSYSTEM_TYPE subsystem_type	Tipo de subsistema (sensor, GPS, etc.)
telemetry_status	***Estado del enlace de telemetría.***	
	uint64_t timestamp	
	uint64_t heartbeat_time	Instante de tiempo en el que se recibió la última señal de vida del sistema remoto
	enum TELEMETRY_STATUS_ RADIO_TYPE type	Tipo de hardware de la radio
	uint8_t rssi	Nivel de señal local
	uint8_t remote_rssi	Nivel de señal remota
	uint16_t rxerrors	Errores en recepción
	uint16_t fixed	Paquetes erróneos corregidos
	uint8_t noise	Nivel de ruido local
	uint8_t remote_noise	Nivel de ruido remoto

	`uint8_t txbuf`	Porcentaje de ocupación del buffer de transmisión
vehicle_attitude_setpoint	***Puntos de referencia de la orientación deseada.***	
	`float roll_body`	Ángulo del vehículo usando sistema de referencia NED
	`float pitch_body`	Ángulo del vehículo usando sistema de referencia NED
	`float yaw_body`	Ángulo del vehículo usando sistema de referencia NED
	`float R_body[3][3]`	Matriz de rotación que describe la orientación deseada como una rotación a partir de la orientación actual del vehículo.
	`bool R_valid`	Verdadero si la matriz de rotación es válida
	`float q_d[4]`	Cuaternión deseado para controladores de orientación basados en cuaterniones
	`bool q_d_valid`	Verdadero si el vector de cuaterniones es válido
	`float q_e[4]`	Error de orientación en cuaterniones
	`bool q_e_valid`	Verdadero si el vector de error de cuaterniones es válido
	`float thrust`	Empuje en Newton que tiene que generar el sistema de potencia
	`bool roll_reset_integral`	Verdadero para reiniciar la parte integral del roll (cambio de la lógica de navegación)
	`bool pitch_reset_integral`	Verdadero para reiniciar la parte integral del pitch (cambio de la lógica de navegación)
	`bool yaw_reset_integral`	Verdadero para reiniciar la parte integral del yaw (cambio de la lógica de navegación)
vehicle_attitude	***Información sobre la orientación actual del UAV***	
	`float roll`	Ángulo de roll en radianes (Tait-Bryan, NED)
	`float pitch`	Ángulo de pitch en radianes (Tait-Bryan, NED)
	`float yaw`	Ángulo de yaw en radianes (Tait-Bryan, NED)
	`float rollspeed`	Velocidad angular de roll en rad/s (Tait-Bryan, NED)
	`float pitchspeed`	Velocidad angular de pitch en rad/s (Tait-Bryan, NED)
	`float yawspeed`	Velocidad angular de yaw en rad/s (Tait-Bryan, NED)
	`float rollacc`	Aceleración angular de roll en rad/s^2 (Tait-Bryan, NED)
	`float pitchacc`	Aceleración angular de pitch en rad/s^2 (Tait-Bryan, NED)

	`float yawacc`	Aceleración angular de yaw en rad/s^2 (Tait-Bryan, NED)
	`float rate_offsets[3]`	Valores de compensación de la velocidad angular
	`float R[3][3]`	Matriz de rotación del vehículo respecto al mundo (Tait-Bryan, NED)
	`float q[4]`	Cuaternión
	`float g_comp[3]`	Vector de compensación de gravedad
	`bool R_valid`	Verdadero si la matriz de rotación es válida
	`bool q_valid`	Verdadero si el cuaternión es válido
vehicle_command	***Definición de los comandos que se envían a commander a través del enlace MAVLink.*** ***Algunos de los posibles comandos son:*** *VEHICLE_CMD_NAV_WAYPOINT* *VEHICLE_CMD_NAV_LOITER_UNLIM* *VEHICLE_CMD_NAV_LOITER_TURNS* *VEHICLE_CMD_NAV_LOITER_TIME* *VEHICLE_CMD_NAV_RETURN_TO_LAUNCH* *VEHICLE_CMD_NAV_LAND* *VEHICLE_CMD_NAV_TAKEOFF*	
	`float param1`	Parámetro 1
	`float param2`	Parámetro 2
	`float param3`	Parámetro 3
	`float param4`	Parámetro 4
	`float param5`	Parámetro 5
	`float param6`	Parámetro 6
	`float param7`	Parámetro 7
	`enum VEHICLE_CMD command`	ID del comando como se define en MAVLink
	`uint8_t target_system`	Sistema que debe ejecutar el comando
	`uint8_t target_component`	Componente que debe ejecutar el comando, 0 para todos los componentes
	`uint8_t source_system`	Sistema que envía el comando
	`uint8_t source_component`	Componente que envía el comando
	`uint8_t confirmation`	0: primera transmisión del comando 1-255: Transmisiones de confirmación
vehicle_control_mode	***Codificación complete del estado del sistema.***	
	`bool flag_armed`	Verdadero si el sistema está armado
	`bool flag_external_manual_override_ok`	Sólo es verdadero para vehículos de ala fija
	`bool flag_system_hil_enabled`	Verdadero si el Sistema está en HIL
	`bool flag_control_manual_enabled`	Verdadero para utilizar los controles de entrada manuales
	`bool flag_control_auto_enabled`	Verdadero para utilizar el controlador de vuelo autónomo de abordo

	`bool flag_control_offboard_enabled`	Verdadero para utilizar un controlador de vuelo externo
	`bool flag_control_rates_enabled`	Verdadero para estabilizar las tasas de variación de la orientación
	`bool flag_control_attitude_enabled`	Verdadero para utilizar controles de entrada estabilizados
	`bool flag_control_velocity_enabled`	Verdadero para controlar la velocidad horizontal (implica dirección)
	`bool flag_control_position_enabled`	Verdadero para controlar la posición
	`bool flag_control_altitude_enabled`	Verdadero para controlar la altitud
	`bool flag_control_climb_rate_enabled`	Verdadero para controlar la tasa de ascenso
	`bool flag_control_termination_enabled`	Verdadero para abortar el vuelo
vehicle_global_position	***Posición global en coordenadas WGS84. No es igual a la posición global medida por el GPS. Es una estimación de la posición global calculada por el estimador de posición teniendo en cuenta otras fuentes de información además del GPS.***	
	`uint64_t time_gps_usec`	Instante de tiempo de medida del GPS
	`double lat`	Latitud en grados
	`double lon`	Longitud en grados
	`float alt`	Altitud AMSL en metros
	`float vel_n`	Velocidad hacia el norte geográfico en m/s
	`float vel_e`	Velocidad hacia el este geográfico en m/s
	`float vel_d`	Velocidad de caída en m/s
	`float yaw`	Yaw en radianes –Pi...+Pi
	`float eph`	
	`float epv`	
vehicle_global_velocity_setpoint	***Velocidad global usando sistema de referencia NED (North-East-Down).***	
	`float vx`	Velocidad hacia el norte en m/s
	`float vy`	Velocidad hacia el este en m/s
	`float vz`	Velocidad hacia abajo en m/s
vehicle_gps_position	***Posición del UAV obtenida directamente de los valores recibidos del receptor GPS.***	
	`uint64_t timestamp_position`	Instante de tiempo de medida de la información de posición
	`int32_t lat`	Latitud en grados 1E-7
	`int32_t lon`	Longitud en grados 1E-7
	`int32_t alt`	Altitud AMSL en metros 1E-3 (mm)
	`uint64_t timestamp_variance`	Instante de tiempo de medida de la varianza
	`float s_variance_m_s`	Varianza de la velocidad en m/s
	`float c_variance_rad`	Varianza de la trayectoria en rad

	`uint8_t fix_type`	Calidad de la señal GPS. 0-1: No fix 2: 2D fix 3: 3D fix
	`float eph`	Error de posición horizontal en metros
	`float epv`	Error de posición vertical en metros
	`unsigned noise_per_ms`	Ruido
	`unsigned jamming_indicator`	Indicador de interferencias
	`uint64_t timestamp_velocity`	Instante de tiempo de medida de la velocidad
	`float vel_m_s`	Velocidad respecto al suelo en m/s
	`float vel_n_m_s`	Velocidad hacia el norte en m/s
	`float vel_e_m_s`	Velocidad hacia el este en m/s
	`float vel_d_m_s`	Velocidad hacia abajo en m/s
	`float cog_rad`	Trayectoria sobre el suelo (NO indica el sentido, sólo la dirección de movimiento) en rad
	`bool vel_ned_valid`	Verdadero si la velocidad es válida
	`uint64_t timestamp_time`	Instante de tiempo de medida de la información del tiempo
	`uint64_t time_gps_usec`	Instante de tiempo de medida indicado por el módulo GPS en µs
	`uint8_t satellites_used`	Número de satélites usados
vehicle_local_ position_setpoint	***Punto de referencia de la posición local deseada.***	
	`float x`	Posición x en metros
	`float y`	Posición y en metros
	`float z`	Posición z en metros
	`float yaw`	Yaw en radianes –Pi...+Pi
vehicle_local_ position	***Posición local estimada.***	
	`bool xy_valid`	Verdadero si las posiciones x e y son válidas
	`bool z_valid`	Verdadero si la posición z es válida
	`bool v_xy_valid`	Verdadero si las velocidades en x e y son válidas
	`bool v_z_valid`	Verdadero si la velocidad en z es válida
	`float x`	Posición X en metros en el sistema de referencia NED (latitud)
	`float y`	Posición Y en metros en el sistema de referencia NED (longitud)
	`float z`	Posición Z en metros en el sistema de referencia NED (altitud negativa)
	`float vx`	Velocidad respecto al suelo en dirección X en m/s
	`float vy`	Velocidad respecto al suelo en dirección Y en m/s
	`float vz`	Velocidad respecto al suelo en dirección Z en m/s
	`float yaw`	Posición de referencia en coordenadas GPS / WGS84

	`bool xy_global`	Verdadero si la posición (x, y) es válida y tiene referencia global válida (ref_lat, ref_lon)
	`bool z_global`	Verdadero si la posición z es válida y tiene referencia global válida (ref_alt)
	`uint64_t ref_timestamp`	Instante de tiempo cuando se ha fijado la posición de referencia
	`double ref_lat`	Punto de referencia de latitud en grados
	`double ref_lon`	Punto de referencia de longitud en grados
	`float ref_alt`	Referencia de altitud AMSL en metros. Se debe considerar como el nivel del suelo actual, no como un punto de referencia.
	`bool landed`	Verdadero si el vehículo está en el suelo (landed)
	`float dist_bottom`	Distancia a la superficie inferior (suelo)
	`float dist_bottom_rate`	Tasa de variación de la distancia al suelo
	`uint64_t surface_bottom_ timestamp`	Instante de tiempo en el que cambia la distancia al suelo
	`bool dist_bottom_valid`	Verdadero si la distancia al suelo es válida
	`float eph`	
	`float epv`	
vehicle_rates_ setpoint	***Velocidad angular deseada.***	
	`float roll`	Velocidad angular en Sistema de referencia NED
	`float pitch`	Velocidad angular en Sistema de referencia NED
	`float yaw`	Velocidad angular en Sistema de referencia NED
	`float thrust`	Empuje normalizado a 0..1
vehicle_status	***Información sobre el estado del UAV***	
	`uint16_t counter`	Contador que el publicador incrementa cada vez que añade nuevos datos.
	`uint64_t timestamp`	Instante de tiempo el que se publican los datos, en µs
	`main_state_t main_state`	Estado principal. “Lo que el usuario quiere que haga el cuadricóptero”
	`navigation_state_t nav_state`	Estado de navegación. “Lo que hace el cuadricóptero”
	`arming_state_t arming_state`	Estado de armado
	`hil_state_t hil_state`	Estado HIL
	`bool failsafe`	Verdadero si el sistema se encuentra en modo de prueba de fallos
	`int32_t system_type`	Tipo de sistema

	int32_t system_id	ID del sistema
	int32_t component_id	ID del componente
	bool is_rotary_wing	Verdadero si el UAV es un UAV de ala rotativa
	bool condition_battery_voltage_valid	Verdadero si los datos de voltaje de la batería son válidos
	bool condition_system_in_air_restore	Verdadero si el sistema se puede reiniciar en el aire
	bool condition_system_sensors_initialized	Verdadero si han inicializado todos los sensores
	bool condition_system_returned_to_home	Verdadero si el sistema puede volver a la estación base
	bool condition_auto_mission_available	Verdadero si hay disponibles datos para una misión
	bool condition_global_position_valid	Verdadero si la calidad de la señal GPS es lo suficientemente buena como para ser usada por el estimador de posición.
	bool condition_launch_position_valid	Verdadero si el lugar de despegue es válido
	float avionics_power_rail_voltage	Voltaje de la línea de tensión
	bool condition_home_position_valid	Verdadero si la posición de 'casa' es válida
	bool condition_local_altitude_valid	Verdadero si la orientación es valida
	bool condition_airspeed_valid	Verdadero si el valor de velocidad del aire medido es válido
	bool condition_landed	Verdadero si el vehículo está aterrizado. Siempre es verdadero si está desarmado
	bool condition_power_input_valid	Verdadero si la potencia de entrada es válida
	bool condition_local_position_valid	Verdadero si la posición local es válida
	bool rc_signal_found_once	Verdadero si se ha detectado por primera vez señales de control
	bool rc_signal_lost	Verdadero si se pierden las señales de control
	bool rc_input_blocked	Verdadero si se deben ignorar las señales de control RC
	bool data_link_lost	Verdadero si se pierde el enlace de telemetría (enlace con QGC)
	bool offboard_control_signal_found_once	
	bool offboard_control_signal_lost	
	bool offboard_control_signal_weak	
	uint64_t offboard_controll_signal_lost_interval	

	uint32_t onboard_control_sensors_present	Verdadero si hay control de sensores abordo
	uint32_t onboard_control_sensors_enabled	Verdadero si el control de sensores está habilitado
	uint32_t onboard_control_sensors_health	
	float load	Carga del procesador, entre 0..1
	float battery_voltage	Tensión de la batería
	float battery_current	Corriente de la batería
	float battery_remaining	Porcentaje de carga de la batería
	enum VEHICLE_BATTERY_WARNING battery_warning	Avisos de batería
	uint16_t drop_rate_comm	
	uint16_t errors_comm	
	uint16_t errors_count1	
	uint16_t errors_count2	
	uint16_t errors_count3	
	uint16_t errors_count4	
	bool circuit_breaker_engaged_power_check	
drv_accel	***Datos obtenidos del acelerómetro. Se compone de dos estructuras:*** • ***accel_report*** • ***accel_scale***	
	struct accel_report	
	uint64_t timestamp	Instante de medida
	uint64_t error_count	Número de errores
	float x	Aceleración del eje X del sistema de referencia NED en m/s^2
	float y	Aceleración del eje Y del sistema de referencia NED en m/s^2
	float z	Aceleración del eje Z del sistema de referencia NED en m/s^2
	float temperature	Temperatura en grados Celsius
	float range_m_s2	Rango en m/s^2 (+- este valor)
	float scaling	Valor de escalado
	int16_t x_raw	Valor bruto de aceleración medida en x
	int16_t y_raw	Valor bruto de aceleración medida en y
	int16_t z_raw	Valor bruto de aceleración medida en z
	int16_t temperature_raw	Valor bruto de la temperatura medida
	struct accel_scale ***Vout = Vscale * (Vin + Voffset)***	
	float x_offset	
	float x_scale	
	float y_offset	
	float y_scale	
	float z_offset	

	`float z_scale`	
drv_baro	***Datos obtenidos del barómetro***	
	`float pressure`	Presión
	`float altitude`	Altitud
	`float temperature`	Temperatura
	`uint64_t timestamp`	Instante de la medida
	`uint64_t error_count`	Número de errores
drv_gyro	***Datos obtenidos del giroscopio. Se compone de dos estructuras:*** • ***gyro_report*** • ***gyro_scale***	
	`struct gyro_report`	
	`uint64_t timestamp`	Instante de medida
	`uint64_t error_count`	Número de errores
	`float x`	Velocidad angular en el eje X del Sistema de referencia NED, en rad/s
	`float y`	Velocidad angular en el eje Y del Sistema de referencia NED, en rad/s
	`float z`	Velocidad angular en el eje Z del Sistema de referencia NED, en rad/s
	`float temperature`	Temperatura en grados Celsius
	`float range_rad_s`	Rango en rad/s
	`float scaling`	Valor de escalado
	`int16_t x_raw`	Valor bruto de velocidad angular medida en x
	`int16_t y_raw`	Valor bruto de velocidad angular medida en y
	`int16_t z_raw`	Valor bruto de velocidad angular medida en z
	`int16_t temperature_raw`	Valor bruto de la temperatura medida
	`struct gyro_scale` ***Vout = (Vin * Vscale) + Voffset***	
	`float x_offset`	
	`float x_scale`	
	`float y_offset`	
	`float y_scale`	
	`float z_offset`	
	`float z_scale`	
drv_mag	***Datos obtenidos del magnetómetro. Se compone de dos estructuras:*** • ***mag_report*** • ***mag_scale***	
	`struct mag_report`	
	`uint64_t timestamp`	Instante de medida
	`uint64_t error_count`	Número de errores
	`float x`	Campo magnético en el eje X del Sistema de referencia NED, en gauss
	`float y`	Campo magnético en el eje Y del Sistema de referencia NED, en gauss

	`float z`	Campo magnético en el eje Z del Sistema de referencia NED, en gauss
	`float range_ga`	Rango en gauss
	`float scaling`	Valor de escalado
	`int16_t x_raw`	Valor bruto del campo magnético medido en x
	`int16_t y_raw`	Valor bruto del campo magnético medido en y
	`int16_t z_raw`	Valor bruto del campo magnético medido en z
	`struct mag_scale` *Vout = (Vin * Vscale) + Voffset */*	
	`float x_offset`	
	`float x_scale`	
	`float y_offset`	
	`float y_scale`	
	`float z_offset`	
	`float z_scale`	
drv_px4flow	***Datos del módulo PX4FLOW. Este módulo NO se incluye en este proyecto pero se utiliza el tópico para enviar al controlador de posición los datos del SONAR***	
	`uint64_t timestamp`	Instante de medida
	`int16_t flow_raw_x`	Flujo de pixeles en dirección X, sin compensación de rotación
	`int16_t flow_raw_y`	Flujo de pixeles en dirección Y, sin compensación de rotación
	`float flow_comp_x_m`	Velocidad sobre el suelo en m/s, con compensación de rotación
	`float flow_comp_y_m`	Velocidad sobre el suelo en m/s, con compensación de rotación
	`float ground_distance_m`	Altitud, distancia al suelo (medida por el sonar)
	`uint8_t quality`	Calidad de la medida: 0: mala calidad 255: máxima calidad
	`uint8_t sensor_id`	ID del sensor
drv_range_finder	***Medida de distancia al suelo***	
	`uint64_t timestamp`	Instante de medida
	`uint64_t error_count`	Número de errores
	`unsigned type`	Tipo de sensor de medida de distancia
	`float distance`	Distancia medida en metros
	`float minimum_distance`	Distancia mínima que el sensor puede medir
	`float maximum_distance`	Distancia máxima que el sensor puede medir
	`uint8_t valid`	Indicador de medida válida: 1 = dentro del rango de medida 0 = fuera del rango de medida
drv_rc_input	***Datos de entrada de señal RC***	

	uint64_t timestamp_publication	Instante de publicación
	uint64_t timestamp_last_signal	Instante de recepción de la última señal
	uint32_t channel_count	Número de canales
	int32_t rssi	Nivel de la señal recibida
	bool rc_failsafe	Verdadero si se pierde la señal RC o si la transmisión está fuera de rango
	bool rc_lost	Verdadero si no se recibe ninguna señal RC en el tiempo requerido
	uint16_t rc_lost_frame_count	Número de tramas de datos de RC perdidas
	uint16_t rc_total_frame_count	Número total de tramas RC recibidos
	uint16_t rc_ppm_frame_length	Longitud de una trama RC de tipo PPM
	enum RC_INPUT_SOURCE input_source	Tipo de fuente
	rc_input_t values [RC_INPUT_MAX_CHANNELS]	Ancho de pulso medido en cada uno de los canales soportados

C Entorno de desarrollo y versiones firmware

Pixhawk

Windows

Hay desarrollado un paquete de herramientas con instalador incluido para facilitar el desarrollo en entornos Windows. En este paquete se instala Eclipse y una consola para poder comunicar con Pixhawk como si se tratase de un entorno Linux. En la versión disponible al momento de realizar este proyecto no se consiguió establecer una configuración de eclipse adecuada, por lo tanto no fue posible hacer el build desde Eclipse. Por ello ha sido necesario usar la consola y realizar el resto del trabajo mediante línea de comandos.

Linux

Se ha utilizado tanto la versión de Ubuntu 12.04 como 14.04, existen pequeñas diferencias entre ellas pero a la hora de configurar el entorno se siguen los mismos pasos.

1. **Instalar dependencias:**

```
sudo apt-get update;

sudo apt-get install python-serial python-argparse openocd \ flexbison
libncurses5-dev autoconf texinfo build-essential \ libftdi-dev libtool
zlib1g-dev genromfs gitwget
```

Es necesario instalar soporte para librerías de 32 bits:

```
sudo apt-get install ia32-libs-multiarch
```

Si la instalación falla:

```
sudo apt-get install libc6:i386 libgcc1:i386 gcc-4.6-base:i386
libstdc++5:i386 libstdc++6:i386
```

2. Configurar USB

```
sudo ls
cat>$HOME/rule.tmp <<_EOF
# All 3D Robotics (includes PX4) devices
SUBSYSTEM=="usb", ATTR{idVendor}=="26AC", GROUP="plugdev"
# FTDI (and Black Magic Probe) Devices
SUBSYSTEM=="usb", ATTR{idVendor}=="0483", GROUP="plugdev"
# Olimex Devices
SUBSYSTEM=="usb", ATTR{idVendor}=="15ba", GROUP="plugdev"
_EOF
sudo mv $HOME/rule.tmp /etc/udev/rules.d/10-px4.rules
sudo restart udev
```

Los usuarios deben pertenecer al grupo plugdev y dialout:

```
sudo usermod -a-G plugdev $USER
sudo usermod -a-G dialout $USER
```

3. ARM Toolchain

```
pushd .
cd ~
wget https://launchpadlibrarian.net/174121628/gcc-arm-none-eabi-4_7-2014q2-
20140408-linux.tar.bz2
tar -jxf gcc-arm-none-eabi-4_7-2014q2-20140408-linux.tar.bz2
export line="export PATH=$HOME/gcc-arm-none-eabi-4_7-2014q2/bin:\$PATH"
if grep-Fxq "$exportline" ~/.profile;
then echo nothing to do;
else echo$exportline >> ~/.profile;
fi
. ~/.profile
popd
. ~/.profile
```

4. **Descargar código fuente Firmware**

```
mkdir ~/src
cd ~/src/
git clone https://github.com/PX4/Firmware
cd Firmware
git checkout 2685e3cfa4fdcbe80e9fd89cc3b08ffafccebad7
make--> automáticamente descarga la versión buena de NuttX y mavlink, si no:
(git clone https://github.com/PX4/NuttX.git
git checkout 7e1b97bcf10d8495169eec355988ca5890bfd5df
git clone https://github.com/mavlink/c_library.git
git checkout 04b1ad5b284d5e916858ca9f928e93d97bbf6ad9) --> da error porque no
se ha hecho make archives
```

5. **Compilar y cargar Firmware en Pixhawk:**

Compilar: se crean los archivos binarios tanto de PX4IO como de PX4FMU, se guardan en la carpeta Firmware/Images

```
cd ~/src/Firmware
make distclean
make archives
make
```

Flashear: con Pixhawk conectado al ordenador con el cable USB

```
make upload px4fmu-v2_default
```

Estación base

1. **Instalar dependencias de QGroundControl:**

Instala qtcreator y algunas librerías necesarias para QGroundControl

```
sudo apt-get install phonon libqt4-dev libphonon-dev libphonon4 phonon-
backend-gstreamer qtcreator libsdl1.2-dev libudev-dev libflite1 flite1-dev
build-essential libopenscenegraph-dev
```

2. **Descargar código fuente de QGroundControl:**

```
git clone https://github.com/mavlink/qgroundcontrol qgroundcontrol
cd qgroundcontrol
git checkout 398d1622df7df77e51faf57a24dfe4f7bed8976b
git submodule init
git submodule update
```

3. **Compilar QGroundControl:**

```
qmake-qt4 qgrouncontrol.pro
make
```

4. **Instalar SDL:**

Dependencias de la librería SDL:

```
sudo apt-get install build-essential xorg-dev libudev-dev libts-dev libgl1-mesa-dev libglu1-mesa-dev libasound2-dev libpulse-dev libopenal-dev libogg-dev libvorbis-dev libaudiofile-dev libpng12-dev libfreetype6-dev libusb-dev libdbus-1-dev zlib1g-dev libdirectfb-dev
```

Descargar SDL con mercurial e instalar:

```
sudo apt-get install mercurial
hg clone http://hg.libsdl.org/SDL
cd SDL
./configure
Make
sudo make install
sudo ldconfig
```

Para eliminar las librerías SDL instaladas:

```
sudo apt-get purge libsdl*
sudo apt-get autoremove --purge
```

5. **Instalar tty0tty:**

Descargar el paquete de la web: http://sourceforge.net/projects/tty0tty/files/

Ejecutar una vez al iniciar el ordenador:

```
cd  /tty0tty/module
make   (sólo es necesario la primera vez o si se hace alguna modificación)
sudo insmodtty0tty.ko
```

Para que no haga falta insertar el módulo en el kernel cada vez que se enciende el ordenador se puede hacer lo siguiente:

```
cd  /tty0tty/module
sudo cp tty0tty.ko /lib/modules/$(uname -r)/
sudo nano /etc/modules
(añadir en la última línea "tty0tty")
sudo depmod
```

D Cambios GIT

```
diff --git a/makefiles/config_px4fmu-v2_default.mk
b/makefiles/config_px4fmu-v2_default.mk
index adfbc2b..8da6ba6 100644
--- a/makefiles/config_px4fmu-v2_default.mk
+++ b/makefiles/config_px4fmu-v2_default.mk
@@ -41,6 +41,7 @@ MODULES                         += drivers/frsky_telemetry
 MODULES                         += modules/sensors
 MODULES                         += drivers/mkblctrl
 MODULES                         += drivers/pca8574
+MODULES                         += drivers/sonar

 # Needs to be burned to the ground and re-written; for now,
@@ -74,6 +75,7 @@ MODULES                                 += modules/commander
 MODULES                         += modules/navigator
 MODULES                         += modules/mavlink
 MODULES                         += modules/gpio_led
+MODULES                         += modules/aux_output


 #
 # Estimation modules (EKF/ SO3 / other filters)
@@ -132,6 +134,7 @@ MODULES                                 += lib/launchdetection
 # Tutorial code from
 # https://pixhawk.ethz.ch/px4/dev/hello_sky
 MODULES                                 += examples/px4_simple_app
+MODULES                                 += examples/print_controls

 # Tutorial code from
 # https://pixhawk.ethz.ch/px4/dev/daemon
diff --git a/mavlink/include/mavlink/v1.0 b/mavlink/include/mavlink/v1.0
--- a/mavlink/include/mavlink/v1.0
+++ b/mavlink/include/mavlink/v1.0
@@ -1 +1 @@
-Subproject commit
04b1ad5b284d5e916858ca9f928e93d97bbf6ad9
+Subproject commit
04b1ad5b284d5e916858ca9f928e93d97bbf6ad9-dirty
diff --git a/src/drivers/px4io/px4io_i2c.cpp b/src/drivers/px4io/px4io_i2c.cpp
old mode 100755
new mode 100644
diff --git a/src/lib/mathlib/CMSIS/libarm_cortexM4l_math.a
b/src/lib/mathlib/CMSIS/libarm_cortexM4l_math.a
old
mode
100755
new
mode
100644
diff --git a/src/lib/mathlib/CMSIS/libarm_cortexM4lf_math.a
b/src/lib/mathlib/CMSIS/ libarm_cortexM4lf_math.a
old mode 100755
```

```
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/attitude_estimator_ekf_main.cpp
b/src/modules/
attitude_estimator_ekf/attitude_estimator_ekf_main.cpp
old mode 100755
new mode 100644
index 35dc39e..5cddbea
--- a/src/modules/attitude_estimator_ekf/attitude_estimator_ekf_main.cpp
+++ b/src/modules/attitude_estimator_ekf/attitude_estimator_ekf_main.cpp
@@ -62,6 +62,7 @@
 #include <uORB/topics/vehicle_gps_position.h>
 #include <uORB/topics/vehicle_global_position.h>
 #include <uORB/topics/parameter_update.h>
+#include <uORB/topics/attitude_correction.h>
 #include <drivers/drv_hrt.h>

 #include <lib/mathlib/mathlib.h>
@@ -228,6 +229,8 @@ const unsigned int loop_interval_alarm = 6500;      // loop
interval in microseconds
             memset(&att, 0, sizeof(att));
             struct vehicle_control_mode_s control_mode;
             memset(&control_mode, 0, sizeof(control_mode));
+            struct attitude_correction_s att_offset;
+            memset(&att_offset, 0, sizeof(att_offset));

             uint64_t last_data =
             0;
             uint64_t
             last_measurement =
             0;
@@ -263,9 +266,13 @@ const unsigned int loop_interval_alarm = 6500;     // loop
interval in microseconds
             /* subscribe to control mode*/
             int sub_control_mode = orb_subscribe(ORB_ID(vehicle_control_mode));
+            int sub_att_corr =
orb_subscribe(ORB_ID(attitude_correction));
+
             /* advertise attitude */
             orb_advert_t pub_att = orb_advertise(ORB_ID(vehicle_attitude), &att);

+

          int
            loopcou
            nter = 0;

          thread_running = true;
@@ -529,8 +536,16 @@ const unsigned int loop_interval_alarm = 6500;     // loop
interval in microseconds
                                                 /* send out */
                                                 att.timestamp = raw.timestamp;

-                                                att.roll = euler[0];
-                                                att.pitch = euler[1];
+                                                bool att_offset_updated;
+                                                orb_check(sub_att_corr, &att_offset_updated);
+                                                if(att_offset_updated){
+                                                        orb_copy(ORB_ID(attitude_correction), sub_att_corr,
                                                        &att_offset);
```

```
+                                                       warnx("Attitude offset: %8.4f,%8.4f\n",
                                                        (double)att_offset.roll,
(double)att_offset.pitch);
+                                               }
+
+                                               att.roll = euler[0] + att_offset.roll;
+                                               att.pitch = euler[1] + att_offset.pitch;
                                        att.yaw = euler[2] + mag_decl;

                                         att.rollspeed = x_aposteriori[0];
diff --git a/src/modules/attitude_estimator_ekf/attitude_estimator_ekf_params.c
b/src/modules/
attitude_estimator_ekf/attitude_estimator_ekf_params.c
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/attitude_estimator_ekf_params.h
b/src/modules/
attitude_estimator_ekf/attitude_estimator_ekf_params.h
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter.c
b/src/modules/
attitude_estimator_ekf/codegen/attitudeKalmanfilter.c
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter.h
b/src/modules/
attitude_estimator_ekf/codegen/attitudeKalmanfilter.h
old mode 100755
new mode 100644
diff --git
a/src/modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter_initialize.c
b/src/
modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter_initialize.c
old mode 100755
new mode 100644
diff --git
a/src/modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter_initialize.h
b/src/
modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter_initialize.h
old mode 100755
new mode 100644
diff --git
a/src/modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter_terminate.c
b/src/modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter_terminate.c
old mode 100755
new mode 100644
diff --git
a/src/modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter_terminate.h
b/src/modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter_terminate.h
old mode 100755
new mode 100644
diff --git
a/src/modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter_types.h
b/src/modules/attitude_estimator_ekf/codegen/attitudeKalmanfilter_types.h
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/cross.c
b/src/modules/attitude_estimator_ekf/ codegen/cross.c
old mode 100755
```

```
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/cross.h
b/src/modules/attitude_estimator_ekf/ codegen/cross.h
old mode 100755
new
mode
10064
4
diff --git a/src/modules/attitude_estimator_ekf/codegen/eye.c
b/src/modules/attitude_estimator_ekf/ codegen/eye.c
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/eye.h
b/src/modules/attitude_estimator_ekf/ codegen/eye.h
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/mrdivide.c b/src/modules/
attitude_estimator_ekf/codegen/mrdivide.c
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/mrdivide.h b/src/modules/
attitude_estimator_ekf/codegen/mrdivide.h
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/norm.c
b/src/modules/attitude_estimator_ekf/ codegen/norm.c
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/norm.h
b/src/modules/attitude_estimator_ekf/ codegen/norm.h
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/rdivide.c
b/src/modules/attitude_estimator_ekf/ codegen/rdivide.c
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/rdivide.h
b/src/modules/attitude_estimator_ekf/ codegen/rdivide.h
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/rtGetInf.c b/src/modules/
attitude_estimator_ekf/codegen/rtGetInf.c
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/rtGetInf.h b/src/modules/
attitude_estimator_ekf/codegen/rtGetInf.h
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/rtGetNaN.c b/src/modules/
attitude_estimator_ekf/codegen/rtGetNaN.c
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/rtGetNaN.h b/src/modules/
attitude_estimator_ekf/codegen/rtGetNaN.h
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/rt_defines.h b/src/modules/
attitude_estimator_ekf/codegen/rt_defines.h
old mode 100755
new mode 100644
```

```
diff --git
a/src/modules/attitude_estimator_ekf/codegen/rt_nonfinite.c
b/src/modules/ attitude_estimator_ekf/codegen/rt_nonfinite.c
old mode 100755
new mode 100644
diff --git
a/src/modules/attitude_estimator_ekf/codegen/rt_nonfinite.h
b/src/modules/ attitude_estimator_ekf/codegen/rt_nonfinite.h
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_ekf/codegen/rtwtypes.h b/src/modules/
attitude_estimator_ekf/codegen/rtwtypes.h
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_so3/attitude_estimator_so3_main.cpp
b/src/modules/
attitude_estimator_so3/attitude_estimator_so3_main.cpp
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_so3/attitude_estimator_so3_params.c
b/src/modules/
attitude_estimator_so3/attitude_estimator_so3_params.c
old mode 100755
new mode 100644
diff --git a/src/modules/attitude_estimator_so3/attitude_estimator_so3_params.h
b/src/modules/
attitude_estimator_so3/attitude_estimator_so3_params.h
old mode 100755
new mode 100644
diff --git a/src/modules/commander/commander.cpp
b/src/modules/commander/commander.cpp
index
48cbc
25..c8
6d281
10064
4
--- a/src/modules/commander/commander.cpp
+++ b/src/modules/commander/commander.cpp
@@ -342,6 +342,8 @@ void print_status()
             orb_copy(ORB_ID(vehicle_status), state_sub, &state);

         const char *armed_str;
+        const char *main_str;
+        const char *nav_str;

         switch
         (state.arming
         _state) {
         case
         ARMING_STA
         TE_INIT:
@@ -376,11 +378,103 @@ void print_status()
                 armed_str = "ERR: UNKNOWN STATE";
                 break;
         }
+        switch (state.main_state) {
+        case MAIN_STATE_MANUAL:
+                main_str = "MANUAL";
+                break;
```

```
-		close(state_sub);
+		case MAIN_STATE_ALTCTL:
+			main_str = "ALTCTL";
+			break;
+
+		case MAIN_STATE_POSCTL:
+			main_str = "POSCTL";
+			break;
+
+		case MAIN_STATE_AUTO_MISSION:
+			main_str = "AUTO_MISSION";
+			break;
+
+		case MAIN_STATE_AUTO_LOITER:
+			main_str = "AUTO_LOITER";
+			break;
+
+		case MAIN_STATE_AUTO_RTL:
+			main_str = "AUTO_RTL";
+			break;
+
+		case MAIN_STATE_ACRO:
+			main_str = "ACRO";
+			break;
+
+		case MAIN_STATE_OFFBOARD:
+			main_str = "OFFBOARD";
+			break;
+
+		default:
+			main_str = "ERR: UNKNOWN STATE";
+			break;
+		}
+
+			switch (state.nav_state) {
+				case NAVIGATION_STATE_MANUAL:
+					nav_str = "MANUAL";
+				break;

+				case NAVIGATION_STATE_ALTCTL:
+					nav_str = "ALTCTL";
+				break;
+
+				case NAVIGATION_STATE_POSCTL:
+					nav_str = "POSCTL";
+				break;
+
+				case NAVIGATION_STATE_AUTO_MISSION:
+				nav_str = "AUTO_MISSION";
+				break;
+
+				case NAVIGATION_STATE_AUTO_LOITER:
+					nav_str = "AUTO_LOITER";
+				break;
```

```
+
+                                case NAVIGATION_STATE_AUTO_RTL:
+                                        nav_str = "AUTO_RTL";
+                                break;
+
+                                case NAVIGATION_STATE_AUTO_RTGS:
+                                        nav_str = "AUTO_RTGS";
+                                break;
+
+                                case NAVIGATION_STATE_ACRO:
+                                        nav_str = "ACRO";
+                                break;
+
+                                case NAVIGATION_STATE_LAND:
+                                        nav_str = "LAND";
+                                break;
+
+                                case NAVIGATION_STATE_DESCEND:
+                                        nav_str = "DESCEND";
+                                break;
+
+                                case NAVIGATION_STATE_TERMINATION:
+                                        nav_str = "TERMINATION";
+                                break;
+
+                                case NAVIGATION_STATE_OFFBOARD:
+                                        nav_str = "OFFBOARD";
+                                break;
+
+                               default:
+                                        nav_str = "ERR: UNKNOWN STATE";
+                                break;
+                }
+                    close(state_sub);

                warnx("arming: %s", armed_str);
+               warnx("main state: %s", main_str);
+               warnx("navigation state: %s", nav_str);
 }

 static orb_advert_t status_pub;
@@ -1329,7 +1423,8 @@ int commander_thread_main(int argc, char *argv[])
                        }

                        /* RC input check */
-               if (!status.rc_input_blocked && sp_man.timestamp != 0 && hrt_absolute_time() < sp_man.timestamp
+ RC_TIMEOUT) {
+               if (sp_man.timestamp != 0 && hrt_absolute_time() < sp_man.timestamp + RC_TIMEOUT) {
+               //if (!status.rc_input_blocked && sp_man.timestamp != 0 && hrt_absolute_time() <
sp_man.timestamp + RC_TIMEOUT) {
                                        /* handle the case where RC signal was regained */
                                        if (!status.rc_signal_found_once) {
                                                status.rc_signal_found_once = true;
diff --git a/src/modules/commander/state_machine_helper.cpp
b/src/modules/commander/ state_machine_helper.cpp
index 7b26e3e..8219850 100644
```

```
--- a/src/modules/commander/state_machine_helper.cpp
+++ b/src/modules/commander/state_machine_helper.cpp
@@ -452,9 +452,11 @@ bool set_nav_state(struct vehicle_status_s *status, const bool
data_link_loss_en
                                if (status->rc_signal_lost && armed) { status->failsafe =
                                true;
-                               if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
+                               /*if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
                                        status->nav_state = NAVIGATION_STATE_AUTO_RTL;
-                               } else if (status->condition_local_position_valid) {
+                               } else */
+                               if (status->condition_local_position_valid) {
                                    status->nav_state = NAVIGATION_STATE_LAND;
                                } else if (status->condition_local_altitude_valid) {
                                    status->nav_state = NAVIGATION_STATE_DESCEND;
@@ -495,9 +497,11 @@ bool set_nav_state(struct vehicle_status_s *status, const bool
data_link_loss_en
                        (!data_link_loss_enabled && status->rc_signal_lost && mission_finished)) {
                                status->failsafe = true;

-                               if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
+                               /*if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
                                    status->nav_state = NAVIGATION_STATE_AUTO_RTL;
-                               } else if (status->condition_local_position_valid) {
+                               } else */
+                               if (status->condition_local_position_valid) {
                                    status->nav_state = NAVIGATION_STATE_LAND;
                                } else if (status->condition_local_altitude_valid) {
                                    status->nav_state = NAVIGATION_STATE_DESCEND;
@@ -509,9 +513,11 @@ bool set_nav_state(struct vehicle_status_s *status, const bool
data_link_loss_en
                        } else if (status->data_link_lost && data_link_loss_enabled) {
                                status->failsafe = true;

-                               if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
+                               /*if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
                                        status->nav_state = NAVIGATION_STATE_AUTO_RTGS;
-                               } else if (status->condition_local_position_valid) {
+                               } else */
+                               if (status->condition_local_position_valid) {
                                        status->nav_state = NAVIGATION_STATE_LAND;
                                } else if (status->condition_local_altitude_valid) {
                                        status->nav_state =
                                        NAVIGATION_STATE_DESCEND;
@@ -535,9 +541,11 @@ bool set_nav_state(struct vehicle_status_s *status, const bool
data_link_loss_en
                        if ((status->data_link_lost && data_link_loss_enabled) && status->rc_signal_lost) {
                                status->failsafe = true;

-                               if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
+                               /*if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
```

```
                                        status->nav_state = NAVIGATION_STATE_AUTO_RTL;
-                                   } else if (status->condition_local_position_valid) {
+                                   } else */
+                                   if (status->condition_local_position_valid) {
                                        status->nav_state = NAVIGATION_STATE_LAND;
                                    } else if (status->condition_local_altitude_valid) {
                                        status->nav_state =
                                        NAVIGATION_STATE_DESCEND;
@@ -549,9 +557,10 @@ bool set_nav_state(struct vehicle_status_s *status, const bool
data_link_loss_en
                            } else if (status->data_link_lost && data_link_loss_enabled) {
                                status->failsafe = true;

-                               if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
+                               /*if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
                                    status->nav_state = NAVIGATION_STATE_AUTO_RTGS;
-                               } else if (status->condition_local_position_valid) {
+                               } else */if (status->condition_local_position_valid) {
                                    status->nav_state = NAVIGATION_STATE_LAND;
                                } else if (status->condition_local_altitude_valid) {
                                    status->nav_state =
                                    NAVIGATION_STATE_DESCEND;
@@ -563,9 +572,10 @@ bool set_nav_state(struct vehicle_status_s *status, const bool
data_link_loss_en
                            } else if (status->rc_signal_lost && !data_link_loss_enabled) {
                                status->failsafe = true;

-                               if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
+                               /*if (status->condition_global_position_valid && status-
>condition_home_position_valid) {
                                    status->nav_state = NAVIGATION_STATE_AUTO_RTGS;
-                               } else if (status->condition_local_position_valid) {
+                               } else */if (status->condition_local_position_valid) {
                                    status->nav_state = NAVIGATION_STATE_LAND;
                                } else if (status->condition_local_altitude_valid) {
                                    status->nav_state =
                                    NAVIGATION_STATE_DESCEND;
diff --git a/src/modules/ekf_att_pos_estimator/ekf_att_pos_estimator_main.cpp
b/src/modules/
ekf_att_pos_estimator/ekf_att_pos_estimator_main.cpp
index
ccc49
73..cd
501e2
10064
4
--- a/src/modules/ekf_att_pos_estimator/ekf_att_pos_estimator_main.cpp
+++ b/src/modules/ekf_att_pos_estimator/ekf_att_pos_estimator_main.cpp
@@ -187,6 +187,7 @@ private:
                orb_advert_t                _wind_pub;                  /**< wind estimate */
                struct vehicle_attitude_s   _att;                       /**< vehicle attitude */
+               struct vehicle_attitude_s   _att_corr;
                struct gyro_report          _gyro;
                struct accel_report         _accel;
                struct mag_report           _mag;
@@ -1340,6 +1341,7 @@ FixedwingEstimator::task_main()
                                    math::Matrix<3, 3> R = q.to_dcm();
```

```
                                        math::Vector<3> euler = R.to_euler();
+
                                        for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) _att.R[i][j] = R(i, j);

diff --git a/src/modules/mavlink/mavlink_messages.cpp
b/src/modules/mavlink/mavlink_messages.cpp
index 6885beb..e253872 100644
--- a/src/modules/mavlink/mavlink_messages.cpp
+++ b/src/modules/mavlink/mavlink_messages.cpp
@@ -69,6 +69,7 @@
 #include <uORB/topics/airspeed.h>
 #include <uORB/topics/battery_status.h>
 #include <uORB/topics/navigation_capabilities.h>
+#include <uORB/topics/attitude_correction.h>
 #include <drivers/drv_rc_input.h>
 #include <drivers/drv_pwm_output.h>
 #include <drivers/drv_range_finder.h>
@@ -1593,6 +1594,8 @@ protected:
                                                manual.y   *   1000,
                                                manual.z   *   1000,
                                                manual.r * 1000,
+                                               manual.aux2 * 1000,
+                                               manual.aux3 * 1000,
                                                0);
                        }
                }
diff --git a/src/modules/mavlink/mavlink_receiver.cpp
b/src/modules/mavlink/mavlink_receiver.cpp
index 3ee7ec3..e189b4d 100644
--- a/src/modules/mavlink/mavlink_receiver.cpp
+++ b/src/modules/mavlink/mavlink_receiver.cpp
@@ -84,6 +84,13 @@ __END_DECLS
        static const float mg2ms2 =

CONSTANTS_ONE_G / 1000.0f;

+#define MAIN_SWITCH 8
+#define ALTCTL_SWITCH 16
+#define POSCTL_SWITCH 24
+#define AUTO_SWITCH 32
+#define RETURN_SWITCH 40
+#define LOITER_SWITCH 48
+
 MavlinkReceiver::MavlinkReceiver(Mavlink *parent) :
        _mavlink(parent),
        status{},
@@ -111,6 +118,8 @@ MavlinkReceiver::MavlinkReceiver(Mavlink *parent) :
        _telemetry_status_pub(-1),
        _rc_pub(-1),
        _manual_pub(-1),
+       _actuator_control_0_pub(-1),
+       _attitude_correction_pub(-1),
        _telemetry_heartbeat_time(0),
        _radio_status_available(false),
        _control_mode_sub(orb_subscribe(ORB_ID(vehicle_control_mode))),
@@ -169,6 +178,10 @@ MavlinkReceiver::handle_message(mavlink_message_t *msg)
                handle_message_request_data_stream(msg);
                break;
+               case MAVLINK_MSG_ID_ATTITUDE:
```

```
+                               handle_message_attitude_correction(msg);
+                               break;
+
                case
                MAVLINK_MSG_ID_ENCAPSULATED_DATA:
                        MavlinkFTP::getServer()->handle_message(_mavlink,
                        msg);
                        break;
@@ -431,6 +444,7 @@
MavlinkReceiver::handle_message_manual_control(mavlink_message_t *msg)
 {
            mavlink_manual_control_t man;
            mavlink_msg_manual_control_decode(msg, &man);
+           uint16_t buttons;

            struct manual_control_setpoint_s manual;
            memset(&manual, 0, sizeof(manual));
@@ -441,7 +455,56 @@
MavlinkReceiver::handle_message_manual_control(mavlink_message_t *msg)
            manual.r = man.r / 1000.0f;
            manual.z = man.z / 1000.0f;
-           warnx("pitch: %.2f, roll: %.2f, yaw: %.2f, throttle: %.2f", (double)manual.x,
(double)manual.y, (double)manual.r, (double)manual.z);
+//         warnx("pitch: %.2f, roll: %.2f, yaw: %.2f, throttle: %.2f", (double)manual.x,
(double)manual.y, (double)manual.r, (double)manual.z);
+
+                   manual.aux1 = man.pan / 1000.0f;
+                   manual.aux2 = man.tilt / 1000.0f;
+                   manual.buttons = man.buttons;
+
+                   buttons = man.buttons & ~0xFFC7;
+
+                   switch (buttons) {
+                           case MAIN_SWITCH:
+                                   manual.mode_switch = SWITCH_POS_OFF;
+                                   manual.posctl_switch = SWITCH_POS_OFF;
+                                   manual.return_switch = SWITCH_POS_OFF;
+                                   manual.loiter_switch = SWITCH_POS_OFF;
+                                   break;
+                            case ALTCTL_SWITCH:
+                                   manual.mode_switch = SWITCH_POS_MIDDLE;
+                                   manual.posctl_switch = SWITCH_POS_OFF;
+                                   manual.return_switch = SWITCH_POS_OFF;
+                                   manual.loiter_switch = SWITCH_POS_OFF;
+                                   break;
+                            case POSCTL_SWITCH:
+                                   manual.mode_switch = SWITCH_POS_MIDDLE;
+                                   manual.posctl_switch = SWITCH_POS_ON;
+                                   manual.return_switch = SWITCH_POS_OFF;
+                                   manual.loiter_switch = SWITCH_POS_OFF;
+                                   break;
+                           case AUTO_SWITCH:
+                                   manual.mode_switch =  SWITCH_POS_ON;
+                                   manual.posctl_switch = SWITCH_POS_OFF;
+                                   manual.return_switch = SWITCH_POS_OFF;
+                                   manual.loiter_switch = SWITCH_POS_OFF;
```

```
+                                       break;
+                               case RETURN_SWITCH:
+                                       manual.mode_switch =  SWITCH_POS_ON;
+                                       manual.posctl_switch = SWITCH_POS_OFF;
+                                       manual.return_switch = SWITCH_POS_ON;
+                                       manual.loiter_switch = SWITCH_POS_OFF;
+                                       break;
+                                case LOITER_SWITCH:
+                                       manual.mode_switch =  SWITCH_POS_ON;
+                                       manual.posctl_switch = SWITCH_POS_OFF;
+                                       manual.return_switch = SWITCH_POS_OFF;
+                                       manual.loiter_switch = SWITCH_POS_ON;
+                                       break;
+                               default:
+                                       manual.mode_switch =  SWITCH_POS_NONE;
+                                       break;
+                               }
                if (_manual_pub < 0) {
                                _manual_pub = orb_advertise(ORB_ID(manual_control_setpoint),
                                &manual);
@@ -910,6 +973,31 @@
MavlinkReceiver::handle_message_hil_state_quaternion(mavlink_message_t *msg)
                                }
                }
 }
+void
+MavlinkReceiver::handle_message_attitude_correction(mavlink_message_t
*msg) +{
+               mavlink_attitude_t att;
+               mavlink_msg_attitude_decode(msg, &att); +
+               uint64_t timestamp =
                hrt_absolute_time(); +
+               struct attitude_correction_s att_offset;
+               memset(&att_offset, 0, sizeof(att_offset));
+
+               att_offset.timestamp = timestamp;
+               att_offset.roll = att.roll;
+               att_offset.pitch = att.pitch;
+
+               warnx("[MAVLINK] Attitude offset: %8.4f,%8.4f\n", (double)att_offset.roll,
(double)att_offset.pitch);
+
+               if (_attitude_correction_pub < 0) {
+                       _attitude_correction_pub = orb_advertise(ORB_ID(attitude_correction), &att_offset);
+
+               } else {
+                       orb_publish(ORB_ID(attitude_correction), _attitude_correction_pub, &att_offset);
+               }
+}


 /**
diff --git a/src/modules/mavlink/mavlink_receiver.h
b/src/modules/mavlink/mavlink_receiver.h
index b64a060..d28df78 100644
--- a/src/modules/mavlink/mavlink_receiver.h
```

```
+++ b/src/modules/mavlink/mavlink_receiver.h
@@ -70,6 +70,7 @@
 #include <uORB/topics/debug_key_value.h>
 #include <uORB/topics/airspeed.h>
 #include <uORB/topics/battery_status.h>
+#include <uORB/topics/attitude_correction.h>

 #include "mavlink_ftp.h"

@@ -120,6 +121,7 @@ private:
         void
         handle_message_hil_sensor(mavlink_message
         _t *msg); void
         handle_message_hil_gps(mavlink_message_t
         *msg);
         void
         handle_message_hil_state_quaternion(mavlink_message_t
         *msg);

+        void handle_message_attitude_correction(mavlink_message_t
         *msg);

         void *receive_thread(void *arg);

@@ -148,6 +150,8 @@ private:
         orb_advert_t
         _telemetry_statu
         s_pub;
         orb_advert_t _rc_pub;
         orb_advert_t _manual_pub;
+        orb_advert_t _actuator_control_0_pub;
+        orb_advert_t _attitude_correction_pub;
         hrt_abstime _telemetry_heartbeat_time; bool _radio_status_available; int
         _control_mode_sub;
diff --git a/src/modules/mc_att_control/mc_att_control_main.cpp
b/src/modules/mc_att_control/
mc_att_control_main.cpp
index 19c1019..15ed4f6 100644
--- a/src/modules/mc_att_control/mc_att_control_main.cpp
+++ b/src/modules/mc_att_control/mc_att_control_main.cpp
@@ -847,6 +847,11 @@ MulticopterAttitudeControl::task_main()
                                         _actuators.control[3] = (isfinite(_thrust_sp)) ? _thrust_sp : 0.0f;
                                         _actuators.timestamp = hrt_absolute_time();

+                                        _actuators.control[5] = _manual_control_sp.aux2;
+                                        _actuators.control[6] = _manual_control_sp.aux3;
+                                        ////////////////////////////////////////
+
                                         if (!_actuators_0_circuit_breaker_enabled) {
                                             if (_actuators_0_pub > 0) {
                                                 orb_publish(ORB_ID(actuator_controls_0),
_actuators_0_pub, &_actuators);
diff --git a/src/modules/systemlib/mixer/multi_tables
b/src/modules/systemlib/mixer/multi_tables
old mode 100755
new mode 100644
diff --git a/src/modules/uORB/objects_common.cpp
b/src/modules/uORB/objects_common.cpp
index 08c3ce1..abbf55a 100644
--- a/src/modules/uORB/objects_common.cpp
+++ b/src/modules/uORB/objects_common.cpp
```

```
@@ -221,3 +221,7 @@ ORB_DEFINE(tecs_status, struct tecs_status_s);
 #include "topics/wind_estimate.h"
 ORB_DEFINE(wind_estimate, struct wind_estimate_s);
+
+#include "topics/attitude_correction.h"
+ORB_DEFINE(attitude_correction, struct attitude_correction_s);
diff --git a/src/modules/uORB/topics/manual_control_setpoint.h
b/src/modules/uORB/topics/ manual_control_setpoint.h
index dde237a..7df78d2 100644
--- a/src/modules/uORB/topics/manual_control_setpoint.h
+++ b/src/modules/uORB/topics/manual_control_setpoint.h
@@ -93,6 +93,8 @@ struct manual_control_setpoint_s {
             float aux4;                                 /**< default function: camera roll */
             float aux5;                                 /**< default function: payload drop */

+            uint8_t buttons;
+
             switch_pos_t mode_switch;                   /**< main mode 3 position switch (mandatory):
_MANUAL_, ASSIST, AUTO */
             switch_pos_t return_switch; (mandatory):    /**< return to launch 2 position switch
_NORMAL_, RTL */
             switch_pos_t posctl_switch;                 /**< position control 2 position switch
(optional): _ALTCTL_, POSCTL */
diff --git a/src/modules/uORB/topics/vehicle_attitude.h
b/src/modules/uORB/topics/vehicle_attitude.h
old mode 100755
new mode 100644
```

E Publicaciones

Parte de este trabajo ha producido la siguiente publicación:

Guadalupe Crespo, Roberto Ponticelli, Guillermo Glez-de-Rivera, Javier Garrido. "*Setup of a communication and control systems of a quadrotor type Unmanned Aerial Vehicle",* DCIS 2014, Madrid, España, Noviembre 2014.

Setup of a communication and control systems of a quadrotor type Unmanned Aerial Vehicle

Guadalupe Crespo, Guillermo Glez-de-Rivera, Javier Garrido
Human Computer Technology Laboratory (HCTLab)
Univ. Autónoma de Madrid, Spain
guillermo.gdrivera@uam.es

Roberto Ponticelli
Robomotion S.L.
Parque Científico de Madrid
Campus de Cantoblanco, C/ Faraday, 7 28049
Madrid, Spain

***Abstract*— This paper outlines the communication and control systems of a quadrotor type Unmanned Aerial Vehicle. The communication system comprises two different links, one for data and other for video signal, and all integrated by an autopilot module. The data link will be implemented using XBee modules and Mavlink communication protocol. The video transmission system will consist of two separate links which can work together avoiding interferences by using orthogonal polarization each other. It is also given some insights of the onboard software architecture, which is based on the Pixhawk autopilot.**

***Keywords*— UAV, quadcopter, autopilot, XBee, Mavlink, Pixhawk.**

I. INTRODUCTION

UAV (Unmanned Aerial Vehicle) or Drones are commonly employed in tasks where they have easy access to places that people do not. Due to their small size UAVs can be useful in hazardous conditions where human life is at risk and surroundings that are inaccessible to reach [1]. Its mobility is of key importance in several fields as surveillance, military application, documentation and monitoring purpose and image acquisition [2] [3] [4]. Nowadays, the most common UAV type for non-military application is the multicopter, and particularly the quadcopter [5] [6] which now have sufficient payload and endurance for this kind of tasks [7].

Among the advantages of copters over planes on these tasks is their ability to hovering. The multicopters offer, on the other hand, more payload carrying capacity than a helicopter of the same wingspan. It is known that multicopters controllability is better as there are multiple thrust vectors which renders the flight control simpler, allowing to hover at a constant level from ground by itself and at the same time allowing anyone to easily maneuver it [8], or also to achieve autonomous take-off and landing with little control software development effort from the autopilot point of view.

In general, they implement several flight modes, from less to more autonomous operation. The typical flight modes are:

- **Manual:** Roll, pitch, yaw and throttle controls are feed directly from the user input to the autopilot to calculate servo output values in an open-loop manner. This mode should not be available for multirotors.
- **Stabilized:** Roll, pitch, yaw and throttle controls are feed as setpoints to the autopilot, calculating servo output values in a close-loop with the attitude controller. Controls are in manual mode but the aircraft attitude is stabilized in the 3D space.
- **Autonomous:** The aircraft follows GPS waypoints set by base station.

It is needed both specialized hardware and software on a flight controller to achieve autonomous flight. In the case of hardware, IMUs (Inertial Measurement Units) are the most relevant module. An IMU is an electronic device that measures and reports on a craft's angular velocity, orientation, and gravitational forces, using a combination of three orthogonal accelerometers, three orthogonal gyroscopes, and three orthogonal magnetometers, enabling to estimate the UAV attitude. Quadcopters nowadays, besides the IMU, are also provided with barometers, vertical range sensors, airspeed sensor and GPS to obtain global position and make possible flying over waypoints.

Any variation on position, orientation and acceleration is detected by hardware devices, but data must be processed by the software onboard the craft to realize autonomous flight. There is great variety of projects where flight control algorithms are developed, as for example, those at the University of Zurich [9] and the Flying Machine Arena [10]. They make micro aerial vehicles (MAV) that even are able to do acrobatic flight.

When the flight mode is not autonomous, quadcopters need to receive control commands. In most of the commercially available devices, these commands are sent using RC systems such as the widely employed Futaba system [11].

978-1-4799-5743-9/14/$31.00 ©2014 IEEE

This UAV prototype will be integrated into the robotic multiplatform ARGOS[1], consisting of long range and high autonomy multi-robotic platform for performing complex missions in hostile environments. ARGOS project includes several robots capable of operate in different environments (on air or on land). The system is intended to operate when a natural disaster happens, such as earthquakes, and the zone becomes inaccessible.

The UAV will be a quadcopter of approximately 75 cm. from side to side. It will be able to carry payload of at least 3 kg and to fly over extended period of time, compared to other crafts similar in size. Mechanical design and controller software will provide the capacity to fly in a variety of conditions such as wind or rain. The quadcopter will be able to perform long range flights with LOS (Line Of Sight), but also fly in urban areas and inside buildings.

The quadcopter will be equipped with two video cameras which will transmit real-time signals. The first camera will be placed in the front of the vehicle and the other one will be hanging below, mounted on a pan/tilt/zoom module.

This paper describes the communication links between the UAV and GCS (Ground Control Station). It is divided into two different links, one for data and other for video signal, and all integrated by an autopilot.

Data transmission system: A bidirectional link is necessary to transmit control and telemetry data. Telemetry data will be sent from the aerial vehicle down to the GCS, and control data will take the opposite way.

Video transmission system: It will have two different links, one for each camera, both unidirectional. This means that it will have two individual transmitters. Control commands for cameras will be sent by the data transmission system.

Autopilot: module that controls both communication and flight stabilization.

The UAV prototype will be implemented using commercial modules, with versatility and system flexibility in mind.

This paper is organized in three main sections. First of all is the "System description", which is divide in other three subsections that describe the different modules of the UAV, those are: "Data transmission system", "Video transmission System" and the "Autopilot". On the next section a study of the autonomy of the system is done. And finally, a "Summary and conclusions" section.

II. System Description

Fig.1 shows the general system diagram, with the modules and their interconnections. The autopilot and the XBee module will be connected by a bidirectional serial link. The autopilot will transmit telemetry information and will receive control commands through the XBee module. On/off switches of both cameras and the Pan/Tilt/Zoom system of the bottom camera will be controlled by the autopilot according to the received commands.

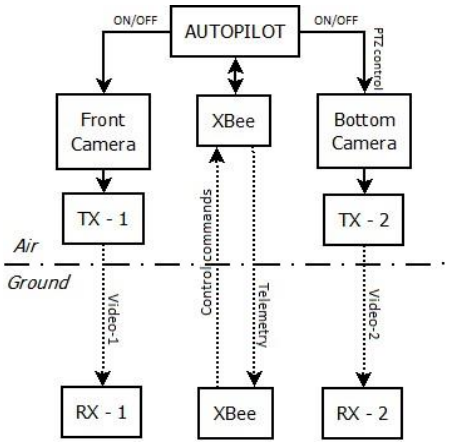


Fig. 1. General system diagram

A. Data transmission system

The communication data channel employs the Mavlink communication protocol [12]. Mavlink is a very lightweight, header-only messages marshaling library for micro aerial vehicles. It can pack C-structs over serial channels with high efficiency. Mavlink is bidirectional, so communication to and from UAV is possible. There is a common message set, but also it is possible to implement custom messages. Mavlink is LGPL (GNU Lesser General Public License) licensed so can be used in close-source and open-source projects.

The anatomy of Mavlink packet is inspired by CAN (Controller Area Network) and SAE AS-4 standards. The minimum packet length is 8 bytes for acknowledgement packets without payload, and the maximum packet length is 263 bytes for full payload. Taking into account worst case, all packets length 263 bytes, it is necessary a link speed of at least 105200 bps to transmit a frame every 20 ms. The next standard value for a link speed is 115200 bps. As the link is not always going to send full payload packets, a system with data rate of 115200 bps should be employed.

A suitable commercial RF device providing the required data rate in a bidirectional link are the XBee [13] modules, due to their low power consumption, lightweight and low cost.

Devices operating at a frequency of 900 MHz are preferred over those at 2.4 GHz, mainly because their longest ranges, up to 45 km, depending on the implementation. At the same emitted power, the RF theory says that at lower frequencies longer ranges can be achieved due to propagation lost. Hence, 900 MHz systems should achieve better ranges than the same system operating on 2.4 GHz.

On the other hand, due to electromagnetic propagation characteristics, below-GHz signals are less susceptible to multipath fading. This is because signals can more easily pass through or around obstructions.

Among the drawbacks of using low frequencies is the limited bandwidth and the bigger size of the antenna. The

[1] Joint project between Robomotion and the HCTLab, Human Computer Technology Laboratory, of Universidad Autónoma de Madrid, under INNPACTO program published in ministerial bulletin ECC/1345/2012.

antenna size increases with lower frequencies, for the same gain. A readily available dipole antenna at 2 dBi is chosen.

The XBee modules use a mesh networking protocol that provides an efficient way to route data between nodes. One of the most popular mesh networking protocols is ZigBee, in which three types of nodes are defined. In every Zigbee network there is a Coordinator node, some Routers and some End Devices. Each one has its own role to play. The selected XBee module uses another mesh network protocol, DigiMesh. This protocol is similar to ZigBee, but has several differences. DigiMesh has only one node type that simplifies network setup and provides more flexibility. Furthermore, DigiMesh achieves longer ranges and larger RF data rates than ZigBee protocol.

In addition, the selected device uses FHSS (Frequency Hopping Spread Spectrum) as a way to reduce interference effects. That also makes the link more robust to multipath fading. Spread spectrum communications improves privacy because the spreading code is only known by the transmitter and the receiver that are involved in the link.

B. Video transmission system

The video transmission system consists of two wireless links able to operate at the same time. Commercial wireless video transmission systems can be divided into two groups: analog and digital. Analog systems are used mainly by the hobbyist community, whilst digital systems, similar to those used in some television broadcasting cameras, are employed in the professional field.

Digital systems are more robust in the presence of interferences. Most of them use COFDM (Coded Orthogonal Frequency Division Multiplexing) modulation. The demodulated signal quality at the receiver is very good; these systems can transmit HD video signals by using H.264 or MPG-4 compression.

The bigger drawback of digital RF video links is the system cost. The less expensive digital transmitters have also some other problems, as big latency due to the time they need to compress video signal, or higher power requirements. Also, digital transmitters are bigger and heavier than analog transmitters.

Therefore, for this prototype, it has been opted for analog wireless PAL video transmitters. They use FM modulation like the ones used to do analog television broadcast.

Both analog and digital systems available in the market operate at the same frequency bands. The most common frequency is 5.8 GHz, but there are video transmitters that can also operate at 1.2 or 2.4 GHz. The best and newest video transmitters and receivers work on 5.8 GHz, thus, this frequency band is the selected one.

The main advantage of working on a high frequency is that there are no issues with the bandwidth. The chosen commercial system operates between 5740 MHz to 5860 MHz. It has seven channels separated 20 MHz each other. This module allows sending video signal and, optionally, one or two audio channels.

Other advantage of this frequency is that a bigger gain can be achieved with small antennas.

At this frequency, electromagnetic signals propagate largely in straight line paths. Thus, they are good for unobstructed line of sight communication. This propagation feature also makes system very susceptible to multipath fading. To reduce that effect, a diversity receiver with two antennas is going to be used. Both antennas receive the same signal but with different phase, and the receiver select the best signal from the two antennas.

Two different kind of circular polarized antennas will be used. One of them will be an omnidirectional 2 dBi antenna and the other a 12 dBi directional antenna, with a beam width of 60º. The directional antenna will always point to the UAV to get as long range as possible, and the omnidirectional one will be used to guarantee communication when the UAV flies near the receiver. Fig. 2 shows the radiation pattern of both antennas and how the main lobe of the directional one points to the null of the omnidirectional antenna.

It will be needed two video links operating together. To realize this configuration, each link will use a different polarization. One transmitter and receiver set will use RHCP (Right Hand Circular Polarized) antennas and the other will use LHCP (Left Hand Circular Polarized) antennas. So, they will be orthogonal minimizing interferences due to cross polarization discrimination.

C. Autopilot

The flight control system has to provide a semiautonomous control to the UAV. It is needed a little and very lightweighted system as possible. Autopilot must be as flexible and versatile as possible and able to allow easy system integration to adapt it to this project.

One type of modification that is expected to do is the integration of new sensors and tasks automation. To do that, it is necessary a system that allows software modification and versatile interconnections.

This level of system versatility is easily found on open-source and open-hardware autopilot projects. The Pixhawk [14] is a powerful hardware-software autopilot suitable to be integrated into this project.

Pixhawk is an autopilot system designed and developed by PX4 project. It features advanced processor and sensor technology and a NuttX [15] real-time operating system. Pixhawk system include integrated multithreading and a Unix/Linux-like programming environment.

Px4 firmware is a continuous development project. This firmware is organized similarly to ROS [16] (Robots Operating System), but in a simpler way. It is made up of different modules or applications that run like independent threads or tasks. Communication among applications is carried out by uORB (micro Object Request Broker) topics, which are data

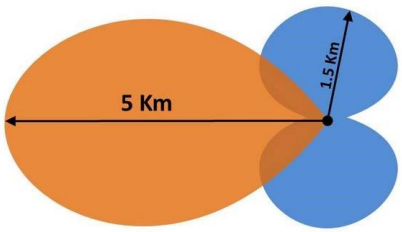


Fig. 2. Receiver antennas radiation pattern **[18]**.

structures. Applications can publish a message on a topic to send data, and can subscribe to a topic to receive data.

Pixhawk unit is combination of PX4FMU (PX4 Flight Management Unit) and PX4IO (PX4 Input Output) hardware modules. PX4IO is designed to driver interfaces to px4io and sends servo setpoints, while it receives battery voltage, current and RC input (Spectrum satellite protocol (DSM), Futaba S.Bus o PPM sum/CPPM). PX4FMU driver configures the FMU board. It controls the multi-function mapping of a number of pins that can be used with different functions such as send PWM servo data or receive PPM RC data.

Fig. 3 and Fig. 4 show a part of the software architecture where it is defined the way data go from RC receiver to motors. Applications are represented as blocks, and the lines that join the blocks represent the topics.

On the first figure (Fig. 3), input control values are published on the topic *input_rc* by either PX4FMU or PX4IO drivers. This topic communicates measured pulse widths for each of the supported input channels.

The SENSORS application subscribes to *input_rc*. In the case of a quadcopter, this application assign the normalized and scaled value of the four first channels to roll, pitch, yaw and throttle, and decides the switches modes (flight options) according to the value at the next four input channels. Other channels can be used command pan/tilt cameras. Calculated values are published on *manual_control_setpoint* topic.

Other possibility (Fig. 4), the one used in this project, is control setpoints are sent using Mavlink communication instead of the traditional RC remote controller.

The attitude controller, MC_ATT_CONTROL application in the case of a quadcopter, subscribes to *manual_control_setpoint*. The application combines the attitude setpoint and the current attitude of the UAV to generate the motor control signals, which are published on *actuator_controls*.

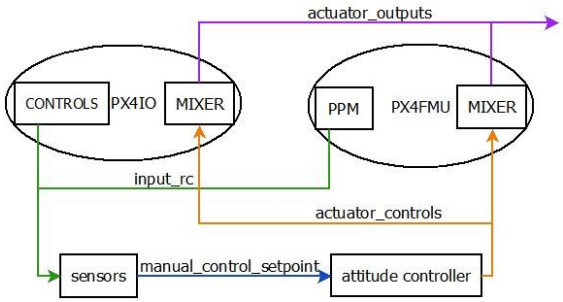


Fig. 3. Part of the software architecture when using RC control.

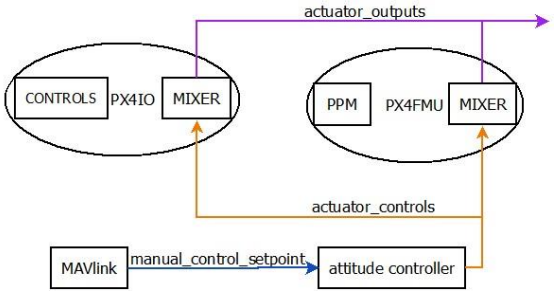


Fig. 4. Part of the software architecture when using Mavlink.

PXIO and PX4FMU drivers subscribe to *actuator_controls*. They use the MIXER to generate a set of outputs, used to update servo output and to publish the results on *actuator_outputs* topic for other modules to use. Multirotor mixer is specifically designed for mixing flight controls (roll, pitch, yaw and thrust) to produce motor speed control outputs suitable for multirotor air vehicles.

Fig. 5 shows the applications (modules) involved in the controller architecture when manual or stabilized flight mode is set. In that case, the attitude controller gets the attitude setpoint from the manual control input, while the current attitude is calculated by the attitude estimator according to the sensors data.

For the autonomous flight mode, the diagram that describes the system can be seen in Fig. 6. The attitude setpoints are generated by the position controller that combines navigator data, the current position and the attitude.

The COMMANDER application, present in both figures (Fig. 5 and Fig. 6), implements the main system state machine and publishes the flight control mode and the actuators status on the topics *vehicle_control_mode* and *actuator_armed*, respectively.

III. Autonomy Study

Two independent batteries are going to be employed to manage this UAV prototype. One will supply power to the motors and the other to the rest of the electronic. The motors have the most power consumption of all of the components on quadrotor type UAV, so that is why the autonomy is given by the motors battery. This is going to be an 8 cells LiPo battery with a capacity of 22000 mAh, which will provide between 25 and 30 minutes flight, considering that the UAV weigh 7.5 Kg including structure, batteries and electronics [17]. The electronics battery will be a 3 cells LiPo.

Table 1. summarizes the current consumption and weight of each electronic component used.

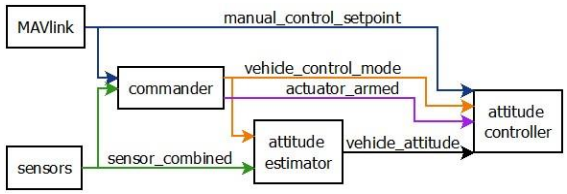


Fig. 5. Control architecture when manual or stabilized flight mode is set.

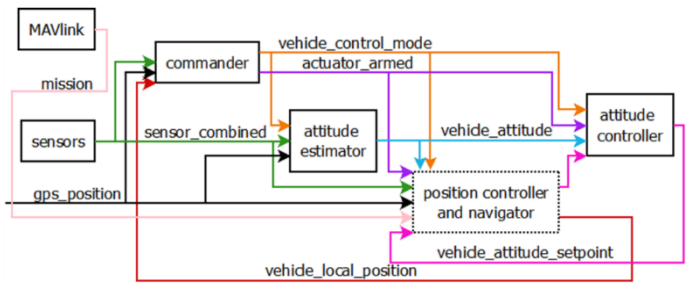


Fig. 6. Control architecture when autonomous flight mode is set.

TABLE I. ELECTRONICS CURRENT CONSUMPTION AND WEIGHT

	Current Consumption	Weight
Pixhawk	250 mA	38 g
GPS	40 mA	16.8 g
Airspeed sensor	3 mA	7 g
Sonar	100 mA	5.9 g
Front camera	250 mA	74 g
Bottom camera	400 mA	200 g
Servomotors	100 mA x 2 = 200 mA	60 g x 2 = 120 g
Video Tx	250 mA x 2 = 500 mA	18 g x 2 + 10 g = 46 g (with antennas)
XBee	215 mA	8 g + 10 g = 18 g (with antenna)
TOTAL	**1958 mA**	**525.7 g**

Total current consumption is about 1958 mA, hence, to achieve 30 minutes autonomy it is enough using a 1000 mAh capacity battery. It may notice that this is the worst case of power consumption because the bottom camera is not always going to be on and it has been considered peak current for the sensors, servomotors and the XBee module.

Furthermore, the battery weighs 97 g and motors, motors battery and the UAV structure will weigh around 6.5 Kg. If this is add to the electronics weight the result is 7.1227 Kg, which is less than the 7.5 Kg considered to estimate the autonomy of the motors, so this will also improve.

IV. SUMMARY AND CONCLUSIONS

This paper presents the integration of a complete communication system on a UAV which is able to operate safely in difficult environmental conditions. A quadrotor type UAV will be used due to its ability of hover at a constant level from ground by itself and because it has sufficient payload capacity and flight endurance. The communication system consists of one data transmission link and two different video links, all integrated by an autopilot.

XBee modules are used for data transmission link due to their low power consumption, light weight and low cost. The selected transceivers work on 900 MHz frequency and emit enough power to achieve medium ranges with 2 dBi gain antennas. Real experiments confirm that selected Xbee modules and antennas reach more than 3 km range with LOS. Mavlink communication protocol is used to allow bidirectional communication, to and from UAV. Mavlink protocol can pack C-structs over serial channels with high efficiency.

Commercial, readily available analog video transmission systems are used at two video links. These transmitters use FM modulation at 5.8 GHz frequency. To reduce multipath fading a diversity receiver with two antennas is used. Two different kind of circular polarized antennas are employed. One of them is an omnidirectional 2 dBi antenna and the other a 12 dBi directional antenna. To avoid cross-talk interference the two video links use orthogonal polarization each other.

To control both communication and flight stabilization it is used the Pixhawk system. The software for the Pixhawk autopilot modules runs on top of the very efficient small operating system NuttX. The Pixhawk software is made up of different modules or applications that run like independent threads or tasks. Communication among applications is carried out by publisher-subscriber scheme. This paper describes the software architecture section needed to be modified in this project to integrate a custom communication system and new sensors, as well as tasks automation.

ACKNOWLEDGMENT

This work has been done with support of the INNPACTO program, in the frame of project ARGOS, published in bulletin ECC/1345/2012. It is a joint project between Robomotion and the HCTLab, Human Computer Technology Laboratory, of Universidad Autónoma de Madrid, Spain.

REFERENCES

[1] M. Kamran Joyo, S. Faiz Ahmed, D. Hazry, M. Hassan Tanveer, and F. A. Warsi, "Position controller design for quad-rotor under perturbed condition," Wulfenia Journal, vol. 20, nº 7, pp. 178-189, 2013.

[2] Shahida Khatoon, Dhiraj Gupta, Ahmad Saad Khan, "Assembly of an experimental quad-rotor type UAV for testing a novel autonomous flight control strategy," International Journal of Advanced Computer Research, vol. 3, nº 4, 2013.

[3] Haitao Jia, Wei Zhang, and Mei Xie, "UAV search planning based on perceptual cue," IEEE International Workshop on Microwave and Millimeter Wave Circuits and System Technology, Chengdu, China, 2013.

[4] Hrishikesh Sharma, Rajeev Bhujade, Adithya V and Balamuralidhar P., "Vision-based detection of power distribution lines in complex remote surroundings," IEEE Twentieth National Conference on Communications (NCC), Kanpur, India, 2014.

[5] M. Hassan Tanveer, D. Hazry, S. Faiz Ahmed, M. Kamran Joyo, Faizan. A. Warsi, "Design of overall stabilized controller for quadrotor," Wulfenia Journal, vol. 20, no. 7, pp. 212-229, 2013.

[6] Michael C. Achtelik, Jan Stumpf, Daniel Gurdan, and Klaus-Michael Doth, "Design of a flexible high performance quadcopter platform breaking the MAV endurance record with laser power beaming," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), San Francisco, CA, USA, 2011.

[7] Inkyu Sa, and Peter Corke, "Vertical infrastructure inspection using a quadcopter and shared autonomy control," Field and Service Robotics Results of the 8th International Conference, vol. 92, Brisbane, Australia, Springer Berlin Heidelberg, 2014, pp. 219-232.

[8] Bernard Tat Meng Leong, Sew Ming Low, Melanie Po-Leen Ooi, "Low-cost microcontroller-based hover control design of a quadcopter," ELSEVIER International Symposium on Robotics and Intelligent Sensors (IRIS), vol. 41, pp. 458-464, 2012.

[9] "ETH - IDSC - Institute for Dynamic Systems and Control," [Online]. Available: http://www.idsc.ethz.ch/. [Accessed 2014].

[10] "Flying Machine Arena," [Online]. Available: http://flyingmachinearena.org/. [Accessed 2014].

[11] "Futaba," [Online]. Available: http://www.futaba-rc.com/. [Accessed 2014].

[12] "Mavlink" [Online]. Available: http://qgroundcontrol.org/mavlink/start.[Accessed 2014].

[13] "XBee," [Online]. Available: http://www.digi.com/products/xbee/. [Accessed 2014].

[14] "PX4 Autopilot Platform," [Online]. Available: http://pixhawk.org/. [Accessed 2014].

[15] "NuttX Real-Time Operating System," [Online]. Available: http://nuttx.org/. [Accessed 2014].

[16] "ROS," [Online]. Available: http://www.ros.org/. [Accessed 2014].

[17] J. A. Benito, G. Glez-de-Rivera, J. Garrido, R. Ponticelli, "Design considerations of a small UAV platform carrying medium payloads" *DCIS*, Madrid, 2014.

[18] "Circular Wireless," [Online]. Available: http://www.circular-wireless.com/. [Accessed 2014].

F Presupuesto

1) Ejecución Material

- Compra de ordenador personal (Software incluido)..... 2.000 €
- Alquiler de impresora láser durante 6 meses..... 50 €
- Material de oficina..... 150 €
- Material para construcción del prototipo:
  - Módulos XBee..... 62,40 €
  - Antenas XBee..... 14 €
  - Transmisores y receptores de vídeo ..... 536 €
  - Antenas de vídeo ..... 160 €
  - Pixhawk y sensores..... 322,49 €
  - Cables ..... 100 €
  - Batería y cargador..... 79 €
- Total de ejecución material..... 3.473,89 €

2) Gastos generales

- 16 % sobre Ejecución Material..... 555,82€

3) Beneficio Industrial

- 6 % sobre Ejecución Material..... 208,43 €

4) Honorarios Proyecto

- 1.200 horas a 15 € / hora..... 18.000 €

5) Material fungible

- Gastos de impresión..... 60 €
- Encuadernación..... 200 €

6) Subtotal del presupuesto

- Subtotal Presupuesto..... 22.498,14 €

7) I.V.A. aplicable

- 21% Subtotal Presupuesto ..... 4.724,61 €

8) Total presupuesto

- Total Presupuesto..... 27.222,75 €

Madrid, Enero de 2015

El Ingeniero Jefe de Proyecto

Fdo.: Guadalupe Crespo Quirós
Ingeniero de Telecomunicación

G Pliego de condiciones

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un SISTEMA DE ENLACE ROBUSTO PARA LA TELEOPERACIÓN DE UN UAV. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma,

por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad "Presupuesto de Ejecución de Contrata" y anteriormente llamado "Presupuesto de Ejecución Material" que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.