



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN

GRADO EN INGENIERÍA EN TELEMÁTICA

TRABAJO DE FIN DE GRADO

Drones con protocolo MavLink en el entorno JdeRobot

Autor: Diego Jiménez Bravo

Tutor: Jose Maria Cañas Plaza

Curso Académico 2017/2018

Proyecto Fin de Carrera

TEMA DEL TFG

Autor : Diego Jiménez Bravo **Tutor :** José María Cañas Plaza

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 2018, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 20XX

Resumen

Es cada vez más común el uso de drones en el día a día de las personas. Se puede observar cómo han explotado en estos últimos años como un aparato de entretenimiento. Pero los UAV tienen mucha historia, se comenzaron a utilizar hace muchos años con fines bélicos, y poco a poco su desarrollo ha permitido que se utilicen en ámbitos muy diferentes, como la robótica, gracias por ejemplo al comportamiento autónomo de este.

Durante este proyecto se ha diseñado y programado un driver con la finalidad de que cualquier dron que use el protocolo MAVLink pueda ser pilotado mediante comandos de velocidad. En este driver se han combinado los interfaces ICE que proporciona el entorno de JdeRobot con las librerías propias de MAVProxy, el intérprete de comandos MAVLink. También se ha desarrollado una herramienta que permite teleoperar cualquier tipo de dron soportado en JdeRobot, como el ArDrone de Parrot o el 3DR Solo Drone de 3DR. Esta aplicación tiene un interfaz gráfico similar a un mando de radiocontrol habitual.

Se han realizado pruebas de validación experimental del driver y de la herramienta, tanto en simulador como con un dron real en interiores y exteriores. Especialmente interesantes han sido los experimentos con un dron real.

Además la interacción de todos los componentes, la concurrencia y el control en velocidad del dron resaltan en este Trabajo Fin de Grado. Todo el software desarrollado y los vídeos de las pruebas están accesibles públicamente.

Índice General

1	Introducción	1
1.1	Robótica Aérea	1
1.2	Tipos de drones	3
1.3	Hardware de drones	5
1.4	Software de drones	9
1.5	Normativa sobre Drones	11
1.6	Robótica aérea en el proyecto abierto JdeRobot	14
2	Objetivos	18
2.1	Problemas a abordar	18
2.2	Requisitos	18
2.3	Metodología	19
2.4	Plan de trabajo	20
3	Infraestructura	21
3.1	3DR Solo Drone	21
3.2	Protocolo MAVLink	23
3.2.1	Software MavProxy	24
3.3	JdeRobot	25
3.3.1	Interfaces ICE relativos a los drones	26
3.3.2	COMM	28
3.4	Biblioteca de comunicaciones ICE	28
3.5	Python	29
4	Driver MavLinkServer	30
4.1	Diseño	30
4.2	Diálogo con drone vía MavProxy	33
4.2.1	Configuración	33
4.2.2	Inicialización	34

4.2.3	Funcionamiento continuo	37
4.2.4	Comunicación vía consola	41
4.3	Diálogo con las aplicaciones ICE	42
4.3.1	Configuración	42
4.3.2	Inicialización	43
4.3.3	Funcionamiento continuo	46
5	Visor UAV Viewer.py	52
5.1	Diseño	52
5.2	Comunicación con los drivers	54
5.3	Interfaz Gráfica	55
5.4	Fichero de configuración	59
5.5	Experimentos	60
6	Conclusiones	62
6.1	Conclusiones	62
6.2	Lecciones prácticas	64
6.3	Trabajos Futuros	65
	Bibliografía	66

Indice de imágenes

1.1	Clasificación por aplicación	3
1.2	Clasificación por alas	4
1.3	Diferentes movimientos que puede realizar un dron.	5
1.4	Distintas partes de un dron.	9
1.5	Normas principales	12
1.6	Prohibiciones principales	13
1.7	TFG Alberto Martín.	14
1.8	TFG Arturo Vélez.	15
1.9	TFG Manuel Zafra.	15
1.10	TFG Daniel Yagüe.	16
1.11	TFG Jorge Cano	16
1.12	TFG Jorge Vela.	17
2.1	Método de desarrollo en espiral	20
3.1	ArduIMU	22
3.2	3DR Solo Drone	22
3.3	Comunicación MAVLink	24
3.4	MavProxy	24
3.5	UAV Viewer	26
4.1	Diagrama comunicación	30
4.2	Diagrama de Hardware y Software	31
4.3	Diagrama de bloques en MAVLinkServer	31
4.4	Diagrama de entradas y salidas bloques MAVLinkServer	32
4.5	Diagrama bloques MAVLink-JdeRobot con detalle	32
4.6	Diagrama de bloques de MavLinkServer	33
5.1	Diagrama de entradas y salidas de UAV-Viewer.py	53
5.2	Diagrama de bloques de UAV-Viewer.py	53

5.3	Arquitectura software de dos Hilos en Uav-Viewer.py	55
5.4	Interfaz gráfico de uav-viewer.py para teleoperación	56
5.5	Interfaz gráfico de UAV-Viewer.py para mostrar la cámara	58
5.6	Interfaz gráfico de UAV-Viewer.py para mostrar los sensores	58
5.7	Captación de movimientos del drone	60
5.8	Experimento de vuelo simulado	61
5.9	Experimento de vuelo real	61

Capítulo 1

Introducción

1.1 Robótica Aérea

Este Trabajo Fin de Grado se encuadra dentro del campo de los drones y del software para robots aéreos. En este capítulo se introducirá este contexto describiendo algunos conceptos, aplicaciones, marco normativo y fundamentos. A continuación, se va a realizar una breve introducción sobre estos aparatos, la historia que tienen, los distintos usos que hay para ellos actualmente, su hardware y software disponible para ellos. También se describirán varios proyectos de RoboticsLab-URJC que forman parte del contexto cercano de este Trabajo de Fin de Grado.

Un dron es una aeronave que vuela sin tripulación reutilizable, capaz de mantener de manera autónoma un nivel de vuelo controlado y sostenido, y propulsado por un motor de explosión, eléctrico, o de reacción. Se van a usar los términos VANT (Vehículo aéreo no tripulado) y UAV (Unmanned aerial vehicle) como sinónimo de dron o drone. El diseño de los VANT tiene una amplia variedad de formas, tamaños, configuraciones y características. Existen dos variantes: los controlados desde una ubicación remota, y aquellos de vuelo autónomo a partir de planes de vuelo preprogramados.

Los drones tienen un origen militar, ya que permite alcanzar al enemigo a distancia así como evitar pérdidas humanas, mejorando así la estrategia de combate. El ejemplo más antiguo fue desarrollado después de la primera guerra mundial, y se emplearon durante la segunda guerra mundial para entrenar a los operarios de los cañones antiaéreos.

Sin embargo, no es hasta poco más que a finales del siglo XX cuando operan los drones mediante radio control con todas las características de autonomía. En Vietnam también se usaron otros modelos de dron como el Ryan Firebee que introdujo una nueva idea, la cámara de fotos para espionar al bando contrario. A su vez, el Gnat fue desarrollado por General Atomics, que introdujo las cámaras de video y con esto empezó una nueva era en el mundo de los drones.

Algunos tienen sistema GPS que les permite volver al punto donde inició de su vuelo. En el futuro se espera que los drones vuelen solos, tomando sus propias decisiones, evitando chocar contra las personas y poder evitar los objetos.

La mayoría de los drones se manejan con radio control, pero pueden ser también manejados y programadas mediante una tablet o un smartphone.

La *robótica aérea* es la rama de la robótica que se encarga del estudio del comportamiento autónomo de los vehículos aéreos no tripulados. En los últimos años, un nuevo reto científico y tecnológico en este área ha sido conseguir que estos vehículos sean totalmente autónomos, es decir, que puedan volar sin la conducción, ni siquiera supervisión, de una persona. A medida que avanza la tecnología y se abaratán los costes, se están integrando en estos vehículos cada vez tecnologías más avanzadas como sistemas de cómputo más poderosos, sensores más ricos en información (cámaras, láseres,...), baterías con mayor autonomía,... El abaratamiento de los drones ha conseguido que cada vez más centros de investigación a lo largo del mundo puedan disponer de estos vehículos consiguiendo así logros que eran impensables hace unos años. Esto está aumentando el campo de aplicación de los VANT, han pasado de utilizarse únicamente en entornos militares a poder utilizarse para servicios forestales, agricultura, medio ambiente, hidrología, cartografía, desastres naturales o geología.

Sin embargo y a pesar de los grandes avances conseguidos en los últimos años, todavía quedan grandes obstáculos que superar. No sólo avances tecnológicos, sino éticos y de regulación por motivos de seguridad. La robótica aérea acaba de empezar a recorrer lo que seguramente sea un gran camino.

Se utilizan para múltiples tareas, desde tareas de vigilancia, fotografía, retransmisiones televisivas, agricultura, ocio y muchas más tareas, ya que cada poco se descubre una nueva forma de utilizar los drones.

Se pueden aplicar en ambientes de alta toxicidad química y radiológicas en desastres tipo Chernóbil, en los que sea necesario tomar muestras con alto peligro de vidas humanas y realizar tareas de control de ambiente. Además, pueden cooperar en misiones de control del narcotráfico y contra el terrorismo. También podrían grabar vídeos de alta calidad para ser empleados como medios de prueba en un juicio internacional.

Los UAV tienen múltiples aplicaciones y posibilidades en el mercado civil y profesional:

- Internet: distribución de señal gratuita de internet.
- Cartografía: realización de ortofotomapas y de modelos de elevaciones del terreno de alta resolución.
- Monitorización de instalaciones.
- Transporte y entrega de mercancías.
- Agricultura: gestión de cultivos.
- Cine y deportes extremos.

- Servicios forestales: seguimiento de las áreas boscosas, control de incendios.
- Búsqueda, rescate y salvamento de personas.
- Medio ambiente: estado de la atmósfera.
- Seguimiento de la planificación urbanística.
- Gestión del patrimonio.
- Seguridad y control fronterizo.
- Purificar el aire mediante un proceso de filtrado mediante capas de poliéster y carbón activado en ambientes de la industria y el hogar.



Figura 1.1: Clasificación por aplicación

1.2 Tipos de drones

En diciembre de 2013 Amazon sorprendió al mundo demostrando su interés en el envío de paquetes con drones¹. Según la propia empresa actualmente se encuentra en la fase de investigación y pruebas con unos VANTs de ocho rotores (optocóptero) que son capaces de levantar cargas de 2.3kg y entregar el paquete en un periodo de 30 minutos desde que el cliente realzase su pedido a través de la web.

En ese mismo mes, la empresa de envíos DHL mostró su interés en el uso de drones para el reparto de paquetes. Cuentan con unos VANTs con una capacidad de carga de 2.99kg y un radio de acción de 1km.

No sólo hay interés en el envío de mercancías con estos aparatos, el Ayuntamiento de Madrid, en diciembre de 2013 realizó un simulacro de rescate en el teleférico de Madrid en el que participaron los Bomberos, el Samur y la Policía municipal, en el que se utilizó un VANT de ocho rotores dotado con

¹<https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011>

sensores para detectar gases, temperatura y agentes radioactivos, biológicos y químicos. Equipado con una cámara fue capaz de retransmitir a los puestos de control las imágenes de las víctimas involucradas en el rescate.

La clasificación de drones es muy amplia, pero la primera clasificación podría ser en función del tipo de alas.

- Drones de Alas Fijas: Tienen alas fijas y son similares a un avión.
- Drones MultiRotor: Tipicamente cuadricópteros (4 rotores con hélices) aunque los hay que tienen 6 (hexacópteros) o incluso 8 hélices. Dos hélices giran en el sentido de las agujas del reloj y las otras dos en el otro sentido, creando así la fuerza de empuje necesaria para llevar al dron hacia arriba. Se pueden mantener en el mismo sitio sin variar la posición, gracias a sus giroscopios y estabilizadores, lo que es perfecto para sacar fotos y grabar videos.



(a) Alas fijas

(b) Multirrotor

Figura 1.2: Clasificación por alas

Según el método de control tenemos:

- Autónomo: El dron no necesita de un piloto humano que lo controle desde tierra. Se guía por sus propios sistemas y sensores integrados.
- Monitorizado: En este caso sí se necesita la figura de un técnico humano. La labor de esta persona es proporcionar información y controlar la realimentación del dron. El dron dirige su propio plan de vuelo y el técnico, a pesar de no poder controlar los mandos directamente, sí puede decidir qué acción llevará a cabo.
- Supervisado: Un operador pilota el dron, aunque este puede realizar algunas tareas autónomamente.
- Pre-programado: El dron sigue un plan de vuelo diseñado previamente y no tiene medios de cambiarlo para adaptarse a posibles cambios.

- Controlado remotamente(R/C): El dron es pilotado directamente por un técnico mediante una consola.

En función de su uso pueden ser:

- Drones Militares: son llamados UCAV que procede del inglés Unmanned Combat Air Vehicle, traducido al español sería vehículos no tripulados de combate aéreo. Suelen ir armados y con capacidad de bombardeos.
- Drones Civiles: son aquellos drones que no tienen uso militar. A su vez pueden ser de:
 - De uso comercial: como cartografías, fotografías, vídeos, etc.
 - Para Aficionados: Se utilizan como un juguete y suelen tener precios bastante económicos.
 - Para Uso del Gobierno: Se utilizan para bomberos, fuerzas de rescate, etc. con el fin de ayudar a las tareas de reconocimiento, rescate, fronterizas e incluso fiscales.

1.3 Hardware de drones

Los componentes que tiene un dron y su forma, permiten que tenga una gran libertad de movimientos, ya que puede moverse sin problema desde cualquier punto hacia los ejes X, Y y Z. Por ejemplo, permiten un aterrizaje y un despegue totalmente vertical, sin depender de un espacio en el que coger velocidad para levantar el vuelo, e igual con el aterrizaje, pudiendo el dron estando quieto en el aire bajar totalmente en vertical hasta posarse sobre el suelo. Una vez en el aire puede moverse adelante, atrás, izquierda, derecha, arriba, abajo y combinaciones de movimientos entre ejes, además de los giros Roll, Yaw y Pitch y sin necesidad de hacer movimientos bruscos.



Figura 1.3: Diferentes movimientos que puede realizar un dron.

Un desglose explicando cada una de las partes sería:

- Marco: También conocido como estructura o chasis. Es la estructura principal sobre la que se sitúan el resto de los elementos. Cambiará su forma dependiendo del dron, variando la longitud de las patas o el número de soportes para hélices, por ejemplo. Puede estar hecha por diversos materiales, generalmente de algún tipo de plástico, ya que tiene poco coste y pesa poco. Un ejemplo es el polipropileno, que es ligero y con mucha resistencia, lo que permite colocar sobre él la batería. Otro material que suele utilizarse es la fibra de carbono, ya que pesa poco y es muy resistente, aunque puede tener factores negativos como su conductividad. Por último, la fibra de vidrio. Es muy utilizado por ser ligero, y tiene características como que no es conductora de la electricidad. Es común ver estructuras híbridas entre distintos materiales, sobre todo juntando los dos tipos de fibra.
- Hélices: Elemento formado por dos palas montadas de forma concéntrica sobre un eje, que al girar crean un par motriz, permitiendo así el movimiento del dron.
- Motores: Son los encargados de transformar la energía, típicamente eléctrica en movimiento sobre el eje en el que se sitúan las hélices, para así permitirles a éstas hacer su trabajo. Tiene distintos parámetros que serán principalmente los que permitan al dron llevar mayor velocidad:
 - El número de vueltas que dé por minuto, lo que dependerá de los Kilovoltios. Suele estar en torno a 800-900kV.
 - El tamaño que el dron tenga. Al mirar las especificaciones de un dron está en un número de 4 dígitos, en el que los dos primeros hacen referencia al tamaño del rotor y los otros dos al tamaño de la bobina.
 - El empuje, valor que hace referencia al peso que puede levantar el motor.
 - La corriente, se trata de la energía (amperios) que se consume cuando el motor está al máximo.
- Batería: Encargada de proporcionar la energía suficiente para que el dron pueda realizar un vuelo, permitiendo trabajar a la placa controladora y motores. La característica principal de las baterías son los miliamperios hora. Cuanto mayor sea permitirá una mayor capacidad y por lo tanto que el dron tenga un mayor tiempo de vuelo. Existen baterías de muy diversos tamaños, desde los 350 mah en drones de juguete a, por ejemplo, los 4500mah que tiene la batería del dron 3DR Solo. También es importante la tasa de descarga, que es la máxima energía que puede entregar y el periodo de tiempo durante el que puede hacerlo. Normalmente los drones traen sistemas de alerta que avisan cuando a la batería le queda poca energía, o que cuando queda un valor menor a cierto porcentaje de carga no permite despegar el dron, evitando así que se quede sin energía a mitad de un vuelo.

- Equipo de transmisión: Es el encargado de que se comunique el dron con una estación receptora. Varía en función del aparato ya que se pueden usar diferentes tecnologías, pero principalmente usa radiofrecuencia o Wifi. Existen casos, como el modelo 3DR, que combina ambas tecnologías, utilizando la radiofrecuencia para la información del movimiento, batería y posicionamiento, y el WiFi para la transmisión de imágenes en directo. Se pueden encontrar distintos equipos de sistemas de transmisión, uno de los últimos y más destacables es Hyperion, que utiliza un sistema óptico de comunicaciones capaz de transmitir hasta 1Gb por segundo, lo que permite la transmisión de datos mediante la luz directa. Su principal característica es que no pierde información cuando no hay contacto directo entre las dos estaciones. La vía WiFi es muy utilizada, pues permite controlar el dron desde una aplicación móvil, por lo que conectando estos dos tendríamos un mando que nos permite cambiar gran parte de la configuración del dron. También existen dispositivos que permiten el control mediante Bluetooth, pero es menos común ya que tiene mayor restricción de velocidad de datos y distancia.
- Placa controladora: Es el procesador a bordo del dron, el que se encarga de recoger la información del dron y cuando le llega una orden ver qué información tiene que mandar a los motores para que se ejecute de forma correcta, así como en caso de haber un problema tratar de evitarlo. Hay una gama muy amplia:
 - Pixhawk²: Es el más utilizado debido a que trabaja con 3DRobotics y Ardupilot. Sirve para diversos dispositivos como drones, helicópteros y barcos. Está pensado para cualquier vehículo que tenga movimiento. Se trata de un proyecto hardware abierto, cuyo objetivo principal es proporcionar el hardware de autopiloto a comunidades académicas o de aficionados, teniendo así un bajo coste y una alta disponibilidad. Es un piloto automático en tiempo real y muy eficiente, proporcionando un entorno de estilo POSIX. Es el autopiloto estándar de la industria. Y a partir él se han desarrollado diversos autopilotos con distintas mejoras.
 - Pixhawk2: Es una versión avanzada de la placa anterior. Tiene mejoras como aislamiento de vibraciones, 3 IMUs para redundancia (3 acelerómetros, 3 giróscopos, 3 magnetómetros y 2 barómetros) y sensor para controlar la temperatura.
 - PixRacer³: Se ha desarrollado para los drones de carreras, aunque también se utiliza en mini drones. Suele tener una mayor memoria flash.
 - Navio2⁴: Piloto automático diseñado de Raspberry Pi. Permite convertir ésta en un controlador de dron.

²<https://pixhawk.org/>

³<https://pixhawk.org/modules/pixracer>

⁴<https://emlid.com/introducing-navio2/>

- PXFmini⁵: Es otro piloto automático de Raspberry Pi. Tiene la electrónica para la mayoría de los componentes que puede utilizar un dron. Lo comercializa ERLE-Robotics.
- FlytPOD⁶: Es una placa Odroid XU4 SBC junto con una PixHawk. Puede volar diversos vehículos aéreos y su principal característica es el WiFi que tiene integrado. Existe una placa FlytPOD pro que es una versión extendida de la anterior, con más sensores y mayor capacidad de almacenamiento.
- U-Pilot⁷: Este hardware se caracteriza por servir para diversos vehículos aéreos, siendo programable para realizar todas las acciones de su camino de forma automática. Su radioenlace con frecuencia en torno a 900Mhz permite controlar el dispositivo a una distancia de 100km.

Hay que destacar también la cámara a bordo, que aunque no todos los drones la llevan sí es bastante común. Algunos la llevan incorporada (incluso dos cámaras, una que apunta hacia la parte de delante y otra que apunta la parte de abajo) y otras que traen soporte para incorporar ciertas cámaras, normalmente consideradas cámaras de acción, para obtener una mejor calidad. Algunos drones incorporan una Gimbal para controlar la dirección hacia la que queremos que apunte la cámara en cada momento o utilizarlo como estabilizador, para evitar así que afecten a la imagen diversos movimientos, generalmente bruscos, que realice dron.

En ocasiones, utilizado normalmente para carreras de drones, la cámara sirve para integrar la tecnología FPV (First Person View), que es junto a la cámara, el transmisor de vídeo y el receptor de vídeo, poder ver en tiempo real y en primera persona, las imágenes sobre una pantalla LCD o utilizando unas gafas de realidad virtual. Uno de los sistemas más impactantes es el conjunto que se ha creado con el dron FLYBi, estando éste conectado a unas gafas de realidad virtual que tienen sensor de movimiento, lo que te permite sentir que eres tú el que vuelas y el que estás en el lugar del dron, y con cualquier movimiento que sientan las gafas la cámara del dron lo imitará. En caso de que esto resulte incómodo tiene un joystick con el que también se pueden ordenar los movimientos a la cámara.

A día de hoy el posicionamiento GPS es un elemento crucial en el control de drones. Integrar sensores GPS con los que conocer en cualquier instante el posicionamiento de nuestro dron, nos facilita ejecutar misiones de vuelo predefinidas o efectuar aterrizajes de improvisados y maniobras de tipo “Go Home” autónomamente y tan solo con un botón del joystick o una acción preprogramada.

Con la ayuda de los sensores GPS podemos conocer la ubicación exacta del dron y comandarle una operación predefinida. Gracias a estos sensores es posible configurar una ruta para operar señalando diferentes waypoints o curvas en el espacio o podemos orientar la videocámara a unas coordenadas

⁵<http://erlerobotics.com/blog/pxfmini/>

⁶http://docs.flytbase.com/docs/FlytPOD/About_FlytPOD.html

⁷<http://www.upilot.ie/>

concretas. Esto resulta muy útil para la ejecución de operaciones de vigilancia, supervisión de construcciones, trabajos topográficos, gestión de agricultura de precisión,

Una IMU (Inertial Measurement Unit), es un dispositivo electrónico que mide e informa acerca de la velocidad, orientación y fuerzas gravitacionales de un aparato, usando una combinación de acelerómetros y giróscopos. Las unidades de medición inercial son normalmente usadas para maniobrar aviones, incluyendo vehículos aéreos no tripulados, entre muchos otros usos, y además naves espaciales, incluyendo transbordadores, satélites y aterrizadores.

La IMU es el componente principal de los sistemas de navegación inercial usados en aviones, naves espaciales, buques y misiles guiados entre otros. En este uso, los datos recolectados por los sensores de una IMU permiten a un computador seguir la posición del aparato, usando un método conocido como navegación por estima.

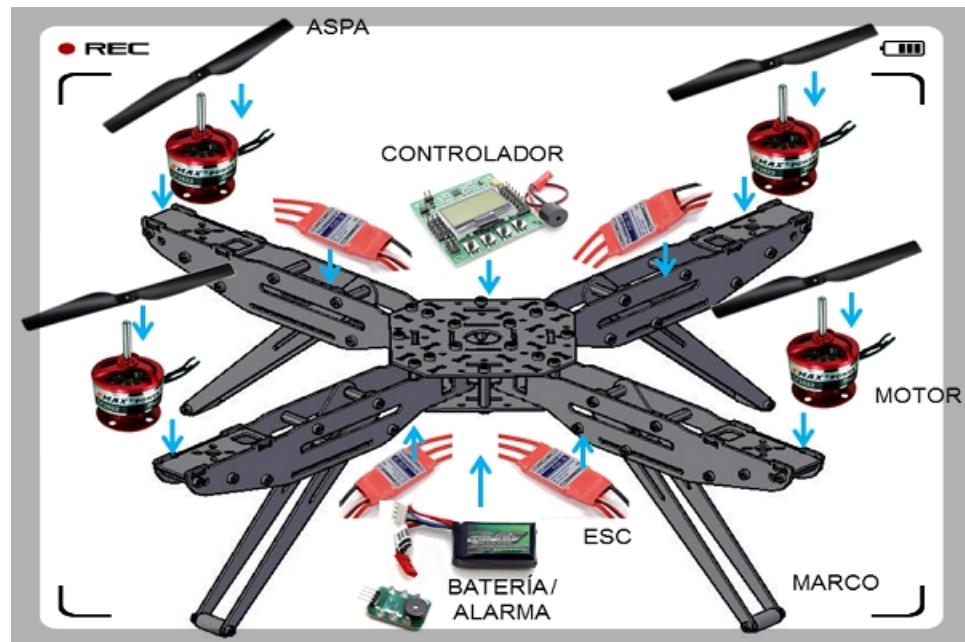


Figura 1.4: Distintas partes de un dron.

1.4 Software de drones

El software es el conjunto de bibliotecas y herramientas que permiten realizar ciertas tareas. Algunos programas estarán instalados en la placa controladora y otros fuera del dron. Será el que se ejecute para recoger la distinta información de los sensores, procesar la información y enviar las órdenes correctas a los distintos elementos. El desarrollo de software para este tipo de robots ha evolucionado mucho en los últimos años debido al uso civil. Existen varios entornos software que permiten el manejo y la programación de estos robots:

- Ardupilot⁸: Es un sistema de código abierto (o software libre) encargado de recibir la información que se quiere trasmitir al drone, recoger la información de los sensores o enviar las señales correspondientes a los motores. Se trata del software más importante por lo completo que es y la confiabilidad que proporciona, debido a la gran cantidad de gente que lo utiliza (pilotos de drones profesionales y aficionados) y por el equipo de ingenieros que lo han desarrollado. Este software se caracteriza por la variedad de dispositivos que puede controlar, ya que trabaja con diversos dispositivos aéreos (aviones, helicópteros, drones, etc.) y con dispositivos marinos (como los barcos y submarinos). Otra característica es la facilidad con la que se le pueden añadir diferentes sensores, como los módulos GPS o cámaras, algo que facilita la navegación autónoma. Ha tenido un gran desarrollo debido a que es código abierto. Hay mucha gente creando interfaces para él y dichos usuarios comparten sus avances con el resto. A partir de éste han nacido controladores como Ardupilot Mega. El problema que tiene este software es que sólo permite trabajar con plataformas de los mismos creadores.
- Megapirate-NG⁹: pareció como desarrollo del ArdupilotMega. La funcionalidad de uno y otro es prácticamente la misma, con la diferencia de que éste permite trabajar con Hardware de otros creadores. El problema es que siempre depende de Ardupilot, por lo tanto sus funcionalidades, aunque sean más cómodas para trabajar, puede que estén atrasadas.
- MultiWii¹⁰: Se propuso como radiocontrol para drones. Es un sistema que fue creado por los desarrolladores y con los sensores (giroscopios y acelerómetros) de la Nintendo Wii. Es una plataforma basada en arduino, con el factor en contra de tener una funcionalidad bastante limitada.

Estos son los entornos software principales que estarían sobre el vehículo, pero también están los programas que se ejecutarían en otros dispositivos como el ordenador o el teléfono móvil para ver la información que nos envía desde la estación de tierra (Ground Control Station) en términos generales. Normalmente el fabricante del dron tiene ya un programa que realiza esta función. En este punto es importante el protocolo de comunicación que se usa para comunicar el vehículo con la estación terrena. Aquí hay un protocolo que destaca sobre los demás, el MAVLink¹¹ (Micro Air Vehicle Communication Protocol). Este protocolo tiene la información contenida en ficheros .xml, lo que permite utilizarlo en diversos lenguajes de comunicación. Además al tener el fichero .xml los tipos de mensajes, permite con facilidad añadir nuevos tipos para asignar una tarea nueva. Otra ventaja es que hay mucho software de drones que lo soportan, como pueden ser Ardupilot, Autopilot, algunos derivados de éstos y otros como Gentlenav o Flexipilot, y desde la estación tierra algunos

⁸<http://ardupilot.org/>

⁹<https://megapirateng.github.io/Docs/documentation/flashtool>

¹⁰<http://www.multiwii.com/>

¹¹mavlink.org

como MAVProxy, Mission Planer o APM planner.

Una desventaja de este protocolo es que los datos no están encriptados en la comunicación, por lo que es más fácil un ataque y que se manipulen los datos. Además, detecta si se pierde algún dato ya que utiliza CRC (código de redundancia cíclica para detectar cambios en los datos). MAVLink se utiliza desde otro software llamado MAVProxy para acceder a los datos del vehículo, como la velocidad y las imágenes, lo que permite también saber qué datos mandarle para que funcione de forma correcta.

Destacar también el entorno software ROS (Robot Operating System), meta sistema operativo de código abierto mantenido por la Open Source Robotics Fundation (OSRF). ROS tiene librerías y herramientas que facilitan desarrollar software para robots. Es el entorno para robots más extendido en el mundo, lo que conlleva que sea el que más aplicaciones tiene. También ofrece muchos componentes para el manejo de drones. En mayo de 2010 mostró el primer vídeo en el que un dron conseguía volar utilizando ROS. En esta demostración el dron se movía con trayectorias agresivas, para mostrar su correcto funcionamiento. Desde entonces ha seguido sacando herramientas para ver el estado del dron conectando con los sensores, imágenes de las cámaras o GPS. Pixhawk también integró su sistema con ROS, lo que hizo que éste todavía creciera más. También tiene interfaz para el control del ArDrone de Parrot. Por otro lado, en protocolos de comunicación MAVLink lanzó también un software para la compatibilidad con ROS.

A parte de éstos, existen también otros entornos software como JdeRobot, que es la que hemos usado en este trabajo y que se describirá con más detalle en el capítulo 3.

1.5 Normativa sobre Drones

Las aeronaves cumplen con las normas reguladoras establecidas en el Tratado de cielos Abiertos de 1992 que permiten los vuelos de VANT sobre todo el espacio aéreo de sus signatarios.

Debido a la gran rapidez con la que se está desarrollando la industria de los drones en España, a día de hoy existe una legislación[15] de drones y unas normas, pero que todavía tienen muchas lagunas. La mayoría de las personas cuando compran un dron desconocen qué pueden hacer con él, dónde lo pueden volar, y en qué momento pueden llegar a violar la Ley de Drones que hoy en día está vigente.

Para las normativas y legislación sobre drones en países diferentes a España, cada uno debe mirar la Ley que se esté aplicando en cada lugar en este momento, ya que como hemos comentado, esta industria no ha hecho más que empezar, y está en constante evolución y mejora.

Las normas básicas principales son:

- Nunca se debe perder el dron de vista, es decir, no puede estar a una distancia que no sea posible ver, ni detrás de obstáculos.

- No se puede superar los 120m de altura.
- Según la normativa de drones, no es obligatorio tener el título de piloto, pero sí de volar con seguridad.
- Sólo es posible volar drones en España en zonas adecuadas para ello. En este punto la ley no es demasiado concreta, zonas adecuadas para volar drones sean lugares de vuelo de aeromodelismo, zonas despobladas, fincas y prados que estén alejados de la ciudad, etc.
- En caso de causar daños personales o materiales, la legislación de drones en España indica que el responsable es el piloto del dron.



Figura 1.5: Normas principales

Las prohibiciones más importantes son:

- Está prohibido volar en zonas urbanas.
- Al venir el mayor peligro de la caída de un dron sobre las personas, no está permitido volar sobre aglomeraciones de personas o lugares donde haya grupos de gente reunidos. Tales sitios serían por ejemplo bodas, conciertos, manifestaciones, parques, playas, etc.
- No está permitido volar de noche, ya que se reduce la visibilidad. Aumenta el peligro.
- Está prohibido volar cerca de aeropuertos o aeródromos.

- Esta prohibido volar drones en España donde otras aeronaves realicen vuelos a poca altura. Esto incluye zonas de parapente, aeródromos, helipuertos, zonas de paracaidismo, etc.
- Como regla general, no se debe poner en peligro a terceros. Es decir, siempre que se vuele un dron, se debe estar muy atento de dónde se está volando, y de si hay personas cerca.

La Ley de drones España es muy clara respecto a estos puntos 1.6, e incluso prevé multas que pueden alcanzar los 225.000€. Aunque esto intimida mucho, con usar el sentido común, y tener presente que no haya personas cerca, es más que suficiente para disfrutar de nuestros drones sin tener ningún problema.

A efectos legales, usar un dron como diversión, como hobby, está perfectamente permitido. Lo que está sujeto a restricciones es usar dicho dron para fines comerciales. Es decir, se puede volar con nuestro dron sin ningún problema, respetando los puntos señalados anteriormente, a menos que, por ejemplo, se realice una grabación y se ésta sea vendida. En ese momento el uso pasa a ser comercial.

Para volar drones con fines comerciales o profesionales, a día de hoy, es necesario tener licencia de piloto o acudir a un centro acreditado y sacar una licencia. Es decir, si se quiere usar un dron para tareas como riego, fumigación, fotografía aérea, detección de incendios forestales, inspección de líneas de alta tensión, fotogrametría, etc. es obligatorio acudir a un centro reconocido y obtener el certificado básico o avanzado que acredite que se pueden desempeñar estas tareas. Además, es obligatorio tener los 18 años y presentar un certificado médico. Con esto, será posible pilotar drones de hasta 25kg con fines comerciales o profesionales.



Figura 1.6: Prohibiciones principales

1.6 Robótica aérea en el proyecto abierto JdeRobot

En un contexto más cercano, este proyecto se inició tras otros realizados anteriormente en el proyecto JdeRobot de software libre para robótica. Los ejemplos más destacables son los siguientes:

- Alberto Martín[1]¹² trabajó en el seguimiento de objetos con un dron utilizando su cámara 1.7. Tenía un controlador reactivo PID, y el dron tenía que seguir una pelota de color rosa, consiguiendo que le siguiera en un espacio 3D. El dron en caso de no encontrar la pelota rosa se quedaba parado donde estuviera. Adicionalmente en este proyecto se desarrolló el driver para el drone real ArDrone de Parrot.

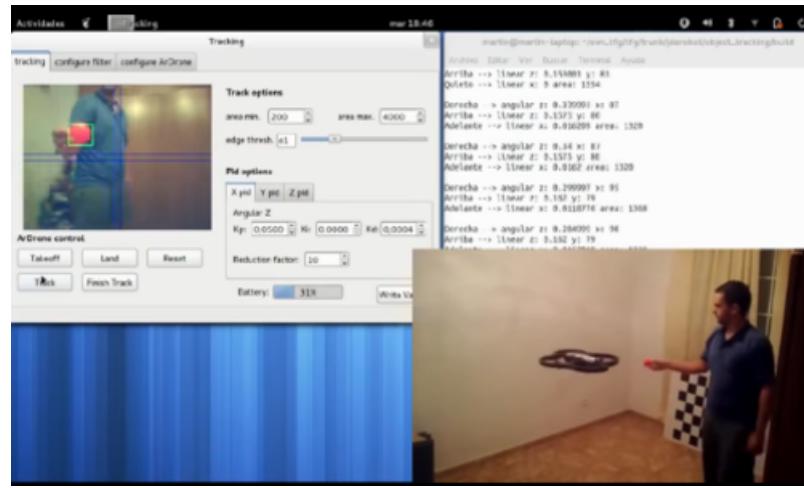


Figura 1.7: TFG Alberto Martín.

- Arturo Vélez[2]¹³ trabajó en un seguimiento visual de objetos sin filtro de color, en el cual un dron debe seguir una textura que se movía por el suelo detectando los puntos de interés 1.8. En caso de que el dron pierda la referencia de la figura en la imagen, realiza una búsqueda.

¹²<http://jderobot.org/Amartinflorido-tfg>

¹³<http://jderobot.org/Avelez-tfg>

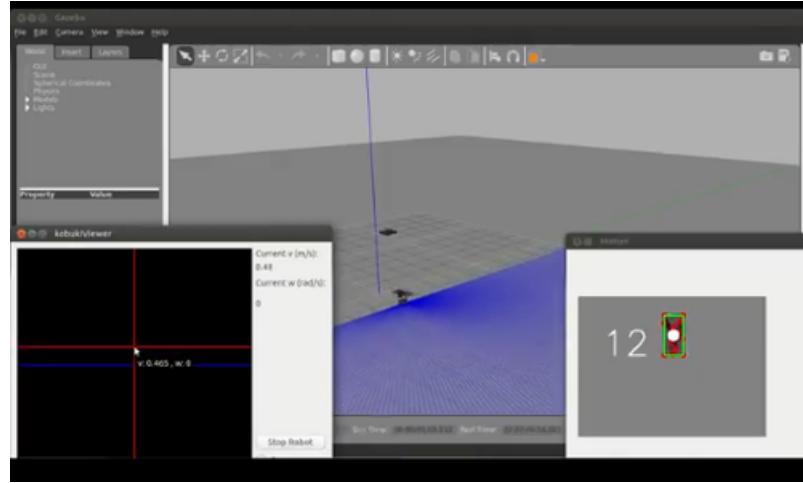


Figura 1.8: TFG Arturo Vélez.

- Manuel Zafra[3]¹⁴ trabajó en la localización del dron mediante April Tags y seguimiento de trayectorias en 3D 1.9. El objetivo de éste era recorrer autónomamente una ruta tridimensional en interiores como secuencia de puntos 3D, para lo que necesitaba estar localizado. Gracias a las balizas April Tags que detectaba el dron podía estimar la posición 3D en el que estaba situado.

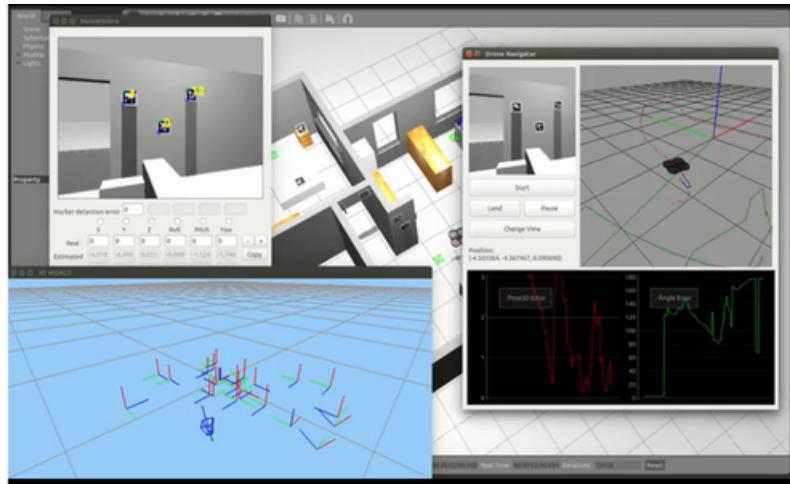


Figura 1.9: TFG Manuel Zafra.

- Daniel Yagüe[4]¹⁵ trabajó con el ArDrone sobre Gazebo, probando distintas funcionalidades y escenarios 1.10. Por un lado trabajó en el driver para el cuadricóptero simulado, consiguiendo un plugin del simulador Gazebo para el ArDrone que proporciona los datos de los sensores a bordo y que acepta y ejecuta órdenes como despegar, aterrizar y los distintos movimientos de

¹⁴<http://jderobot.org/Mazafra-PFC>

¹⁵<http://jderobot.org/Daniyague-PFC>

vuelo. Por otro lado sobre simulador programó varias aplicaciones de control visual, haciendo que el dron siguiera una línea, una carretera o que un dron siguiera al otro.



Figura 1.10: TFG Daniel Yagüe.

- Jorge Cano[5]¹⁶ trabajó en el diseño, construcción y programación de un dron real desde cero. El driver que realizó para su control se basa en el protocolo MAVLink. Por una parte trabajó en el diseño y construcción de la plataforma hardware, como se muestra en la figura 1.11, y por otra en el controlador software para la comunicación con el vehículo. Para hacer tests de vuelo realizó pruebas perceptivas y de control, con filtros de colores y un controlador PID.



Figura 1.11: TFG Jorge Cano

- Jorge Vela[6]¹⁷ trabajó en el diseño de un algoritmo para que el dron navegue de forma autónoma 1.12, desde una baliza de despegue hasta otra baliza de aterrizaje, de posición desconocida, y no necesite a ninguna persona diciéndole lo que tiene que hacer en cada momento

¹⁶<http://jderobot.org/J.canova-tfg>

¹⁷<http://jderobot.org/Jvela-tfg>

o controlándole. Para conseguirlo se han tenido en cuenta los distintos estados en los que se puede encontrar el dron.



Figura 1.12: TFG Jorge Vela.

Una vez terminada esta breve introducción, el resto de esta memoria se organiza en otros cinco capítulos. En el capítulo 2 se redactan los objetivos propuestos y la metodología seguida para llegar hasta ellos. En el capítulo 3 se presenta la infraestructura utilizada y cómo se ha trabajado con ella. En el capítulo 4 se describe el *driver* que se ha implementado, cómo se ha programado y porqué se han seguido unos u otros caminos. En el capítulo 5 se detalla la herramienta de apoyo desarrollado para teleoperar el dron y los resultados que se han obtenido. Por último, en el capítulo 6 se incluyen las conclusiones a las que se han llegado tras finalizar este proyecto.

Capítulo 2

Objetivos

2.1 Problemas a abordar

El principal objetivo de este trabajo fin de grado es dar soporte dentro de la plataforma software JdeRobot a drones que utilicen el protocolo MAVLink para comunicarse con la controladora a bordo. Este problema lo hemos dividido varios subobjetivos:

1. Desarrollar un driver para pilotar drones usando una interfaz de comandos de velocidad. Se comunicará con el dron usando el protocolo MavLink y ofrecerá el interfaz de nivel medio que ya existe en JdeRobot para manejar drones llamada `CMDVel`.
2. Desarrollar una herramienta que permita pilotar los drones usando interfaz de comandos de velocidad en Python de manera intuitiva. Permitira visualizar los datos de los sensores a bordo y la teleoperación del drone de modo similar a los mandos físicos actuales. Esta herramienta será una evolución mejorada de la ya existente que facilita JdeRobot. Será compatible con el driver del primer subobjetivo.
3. Experimentos en dron real. Validaremos experimentalmente ambos desarrollos con un drone 3DR Solo, tanto el driver, como la herramienta renovada de teleoperación que nos permitirá pilotarlo.

2.2 Requisitos

Adicionalmente, como requisitos no funcionales del software a desarrollar tenemos:

1. Serán programados en lenguaje Python, tanto el driver como la herramienta de teleoperación.
2. Serán multiplataforma que valgan para muchos modelos de drones.
3. Se utilizarán únicamente librerías de software libre y serán software libre.

4. Serán 100 % compatibles con los actuales interfaces existentes JdeRobot y con la versión 5.6.3 de esa plataforma.
5. Se integrarán en el repositorio oficial de JdeRobot en GitHub para su uso por terceros.

2.3 Metodología

Durante el ciclo de vida del proyecto se han llevado a cabo reuniones semanales de seguimiento con el tutor. En ellas se evaluaban las tareas realizadas y se marcaba qué dirección tomar para la siguiente iteración o incremento. Si los puntos marcados en la anterior reunión no se habían alcanzado se ampliaba el plazo o se discutían otras vías para avanzar. En caso contrario se proponían nuevos subobjetivos.

Para apoyarnos en nuestro desarrollo hemos utilizado principalmente 3 herramientas metodológicas:

- GitHub como forja y control de versiones. En el repositorio ¹ se almacenan todos los desarrollos que son objetivo de este TFG así como esta memoria. También se encuentran subproductos de desarrollo que han ido surgiendo como apoyo o pruebas a los desarrollos principales.
- Contamos también con un mediawiki en la web de JdeRobot donde hemos actualizado periódicamente nuestros avances acompañados con explicaciones, vídeos e imágenes. ²
- Todos los vídeos del mediawiki han sido compartidos en Youtube.

Se propuso un desarrollo en espiral. Para ello se proponían semanalmente reuniones con el tutor, en las cuales se presentaban unos objetivos a seguir en función de lo que se había conseguido hasta el momento. Una vez propuestas estas tareas se evaluaban los distintos riesgos que se podían tomar en función de trabajar de una forma u otra, y los avances a los que se podía llegar por cada camino. Una vez hecho esto, se comenzaban a desarrollar los algoritmos para conseguir estos objetivos en función de la forma elegida. Despues se probaban en el robot real. Por último se proponía otra reunión para determinar si los avances eran los deseados o no, y en función de esto proponer los nuevos objetivos.

¹<https://github.com/RoboticsURJC-students/2016-tfg-Diego-Jimenez>

²<http://jderobot.org/Jimenez-tfg>



Figura 2.1: Método de desarrollo en espiral

2.4 Plan de trabajo

La planificación seguida en el desarrollo ha incluido las siguientes fases:

- Formación: Comprender las distintas herramientas de JdeRobot y trabajar con ellas, creando programas simples y viendo que funcionaban, así como trabajar con distintos robots reales para tener una toma de contacto con ellos.
- EStudio del protocolo MavLink: Familiarizarse con sus librerías, interfaces y código que se encuentra en el repositorio oficial de ArduPilot³.
- Desarrollo del driver e integración con JdeRobot: Aprovechar al máximo las funciones que puede desempeñar un dron a través del protocolo MavLink y construir un interfaz ICE en JdeRobot que sea capaz de manejarlo.
- Desarrollo de una aplicación de control en tierra: Para visualizar los datos sensoriales a bordo y para enviar al dron en tiempo real órdenes directas de movimiento a través de un interfaz visual que sea lo más intuitiva posible.
- Validación experimental: Tras marcar un objetivo, se procede al desarrollo y su posterior prueba y ajuste experimental. Una vez concluido y validado, se comienza un nuevo subobjetivo.

³<https://github.com/ArduPilot/MAVProxy>

Capítulo 3

Infraestructura

Una vez fijados los objetivos a abordar, en este capítulo se va a describir la infraestructura concreta en la que nos hemos apoyado para desarrollar las soluciones, tanto hardware como software.

3.1 3DR Solo Drone

Para este Trabajo Fin de Grado se ha elegido el dron 3DR Solo distribuido por la empresa norteamericana 3DRobotics [11]¹. Este dron se encuentra en una gama alta debido a sus capacidades ², tales como batería, distancia de comunicacion y potencia, cualidades que lo hacen un dron muy versatil. Durante el desarrollo se ha usado la versión oficial del firmware de 3DR 2.4.2. Se eligió esa debido a que era una versión estable. A bordo de este dron se encuentra una placa estabilizadora Pixhawk 2.

Este drone incluye una tarjeta ArduIMU³, es una Unidad de Medida Inercial (IMU) que integra un procesador compatible con Arduino y es capaz de ejecutar Attitude Heading Reference System (AHRS), basado en el algoritmo DCM (Direct Cosine Matrix) de Bill Premerlani. La tarjeta IMU consta de un acelerómetro de 3 ejes y tres sensores giroscópicos, un regulador de tensión dual (3.3V y 5V), un puerto de GPS, un Atmega328 @ 16MHz (como el Arduino Duemilanova), 3 LEDs de estado, cuenta con una MPU-6000 que integra un giroscopio que se comunica mediante el bus SPI, un magnetómetro HMC-5883L conectado mediante I2C y un Arduino Atmega328 de 16Mhz. La ArduIMU de la figura 3.1 no es ninguna placa de navegación o piloto automático, sólo una placa de orientación y se puede utilizar en cualquier dispositivo del cual deseemos conocer su orientación con respecto al suelo – barcos, coches, aviones.

¹<https://3dr.com/>

²<https://3dr.com/solo-drone/specs/>

³https://3dr.com/support/articles/arduimu_v3_kit/

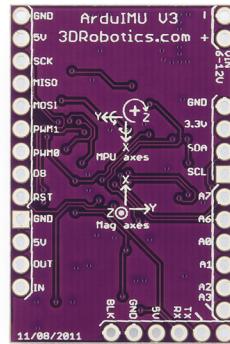


Figura 3.1: ArduIMU

El mando del Solo drone proporciona los mecanismos de control y muestra los datos del vuelo en una pantalla a todo color. Mediante el uso de antenas dobles de largo alcance, el mando actúa como el eje central para todas la comunicaciones de la red Link 3DR via radio, recibe todas las comunicaciones de Solo y la aplicación de tierra, reenvía las salidas telemétricas a la aplicación y administra la transmisión de todas las entradas de control a Solo.



Figura 3.2: 3DR Solo Drone

El único modo de conectar con el 3DR Solo será a través del mando, debido a que únicamente él es capaz de levantar la red wifi a la que conectarse. En posteriores evoluciones tanto del dron como del mando, desde los foros oficiales de 3DR ⁴. Comentan que ya se aborda la solución de que sea el propio dron quien levante la red wifi a la cual conectar y no tener que depender del enlace del mando.

Este dron incluye una placa estabilizadora Pixhawk. Esta placa es un desarrollo específico creado por la “Pixhawk[12] open hardware community” en colaboración con 3D Robotics y que vio como

⁴<https://3drpilots.com/threads/connecting-directly-to-the-pixhawk-2-on-a-solo.7926/>

primer destinatario el 3DR Solo. Ofrece un interfaz que se apoya en el protocolo MAVLink. A través de estos comandos se le puede también enviar órdenes al piloto automático.

3.2 Protocolo MAVLink

MAVLink⁵[9] (Micro Air Vehicle Link) es un protocolo desarrollado para comunicar las placas estabilizadoras dotadas de piloto automático a los GCS (*Ground Control Station*) o estación de tierra con las aplicaciones desde las que se envía misiones y se monitoriza el cumplimiento de las mismas desde tierra. MAVLink se publicó ⁵ en 2009 por Lorenz Meier, con licencia LGPL. Aspira a convertirse en el protocolo standard en robótica aérea y se ha probado su funcionamiento en PX4, PIXHAWK, APM⁶ y Parrot AR.Drone.

La lista completa de los comandos de este protocolo se encuentra en la pagina oficial de Mavlink⁷. La versión actual que se está utilizando en este TFG es la 2.0.

Cada comando tiene un identificador único el cual permite al dron reconocer la acción que se debe realizar. En función de este identificador los parámetros que se introducen a continuación dan la información necesaria al dron para actuar. Un ejemplo de la estructura del mensaje que se usa para GPS es el siguiente:

```
type GpsStatus struct {
    SatellitesVisible     uint8
    SatellitePrn          [20] uint8
    SatelliteUsed         [20] uint8
    SatelliteElevation   [20] uint8
    SatelliteAzimuth     [20] uint8
    SatelliteSnr          [20] uint8
}
```

Este mensaje trae la información del enlace actual con el GPS y se envía periódicamente en ciclos cuya frecuencia se configura en los parámetros de conexión con el dispositivo.

Un ejemplo de los mensajes más importantes del protocolo en los que se centra este TFG es el comando de velocidad. Este comando hace uso de la estructura del mensaje “SET_POSITION_TARGET_LOCAL_NED”, en el cual se indican las velocidades lineales que debe seguir en cada eje (en m/s). Este comando, junto con el de ordenar la rotación, cuya estructura es la del mensaje “COMMAND_LONG”, permite tener control total sobre las velocidades del dron.

En la figura 3.3 se puede comprobar la estructura de la comunicación entre la estación de tierra

⁵<https://github.com/mavlink/mavlink/commit/a087528b8146ddad17e9f39c1dd0c1353e5991d5>

⁶Ardupilot Mega

⁷<http://mavlink.org/messages/common>

y cada componente se establece. Continuamente se intercambian mensajes con información, ya sea para comunicar una acción, reclamar el estado de algún componente interno, como podría ser la batería, o un simple ACK para mantener la conexión activa.

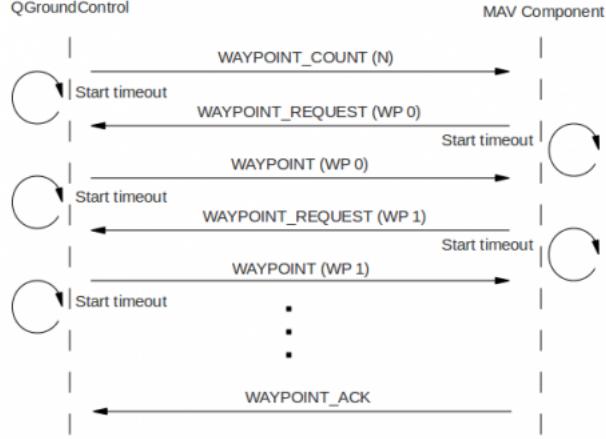


Figura 3.3: Comunicación MAVLink

3.2.1 Software MavProxy

MavProxy[14] es un módulo controlador multihilo que simplifica el uso del protocolo MAVLink desde aplicaciones en Python. Es un GCS totalmente funcional (sistema de comunicación grupal) para UAV, la versión 1.4.38 es utilizado en este trabajo fin de grado. Es un GCS minimalista, portátil y extensible para cualquier UAV que soporte el protocolo MAVLink. Tiene una serie de características clave, que incluyen la capacidad de reenviar mensajes de UAV a través de UDP (User Datagram Protocol) a otros programas de estación terrestre en otros dispositivos. Una de las opciones más interesantes de MavProxy es que permite reenviar mensajes provenientes de nuestro UAV hacia diferentes GCS ubicadas en diversas plataformas como ordenadores, tabletas o teléfonos móviles. De la misma manera, podemos enviar comandos en tiempo real al UAV desde el GCS[7].

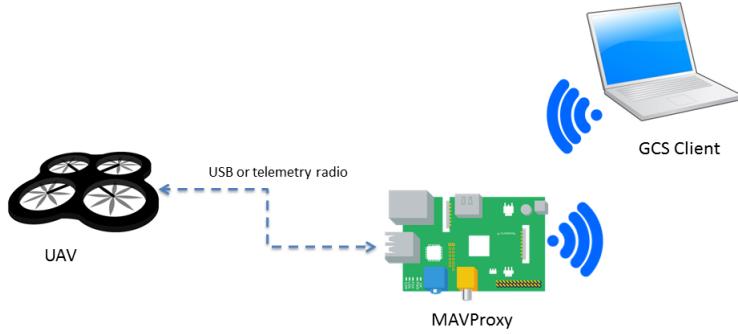


Figura 3.4: MavProxy

MAVProxy permite ordenar el diálogo desde un programa en Python con un UAV a través de varios canales simultáneos. Simplifica tanto la lectura de mensajes del UAV, típico de la información sensorial, como el envío de mensajes al UAV, típico de las órdenes de actuación (tanto de comandos de velocidad como puntos de paso para las misiones). Además también simplifica el intercambio de mensajes de configuración.

Tiene como principales características:

- Es una aplicación de línea de comandos y consola. Hay complementos incluidos en MAVProxy para proporcionar una GUI básica.
- Está escrito en Python.
- Es de código abierto.
- Es portátil; debería ejecutarse en cualquier sistema operativo POSIX con Python, pyserial y función llamadas, lo que significa Linux, OS X, Windows y otros.
- Admite módulos cargables y tiene módulos para admitir consolas, mapas en movimiento, joysticks, seguidores de antena, etc.

3.3 JdeRobot

JdeRobot^[8] es un entorno desarrollado por el laboratorio de robótica de la Universidad Rey Juan Carlos para el desarrollo de aplicaciones de robótica. Su última versión la 5.6 ⁸ se liberó el 9 de Octubre de 2017 y es la usada en este TFG. JdeRobot se compone de interfaces, drivers, utilidades y aplicaciones para el desarrollo de proyectos de robótica. Las aplicaciones y los drivers se comunican intercambiando mensajes entre sí utilizando interfaces ICE o interfaces ROS.

Algunos de los drivers más importantes que contiene:

1. Camer(server. Para enviar imágenes y video a través del interfaz ICE camera.
2. Gazebo(server. Conjunto de plugins en el simulador Gazebo que ejercen de drivers entre las aplicaciones inteligentes y los robots simulados.
3. Ardrone_server. Driver que ofrece acceso a los sensores y actuadores del Parrot Ar-Drone a través de interfaces ICE ⁹. Está escrito en C++, transforma el conjunto de comandos AT del drone en interfaces y viceversa, implementa los interfaces camera, cmdvel, navdata, extra y pose3D y permite acceder a la actitud del drone así como a sus 2 cámaras. Sirve también datos como el nivel de la batería y permite grabar vídeo o tomar fotos.

⁸<https://github.com/JdeRobot/JdeRobot>

⁹<http://jderobot.org/Amartinflorido-tfg>

Algunas de las herramientas disponibles en JdeRobot más usadas en este TFG son:

1. Cameraview. Una aplicación desarrollada en C++ capaz de recibir vídeo a través del interfaz camera.
2. UAV viewer. Aplicación desarrollada como control en tierra de robots aéreos. Permite teleoperar cualquier tipo de robot aéreo utilizando interfaces ICE. Ofrece de forma visualmente atractiva datos como la actitud, velocidades lineales y angulares, ofrece también la posibilidad de visualizar videos servidos por el interfaz camera como se ve en la figura 3.5 ¹⁰. Existen varias versiones de esta aplicación, escritas en lenguajes como C++, Javascript y la desarrollada en este Trabajo Fin de Grado en Python. Ofrece información del drone en tiempo real de los parámetros de vuelo como pueden ser la posición o velocidad. También permite teleoperar el drone mediante comandos de velocidad y órdenes de aterrizaje o despegue.

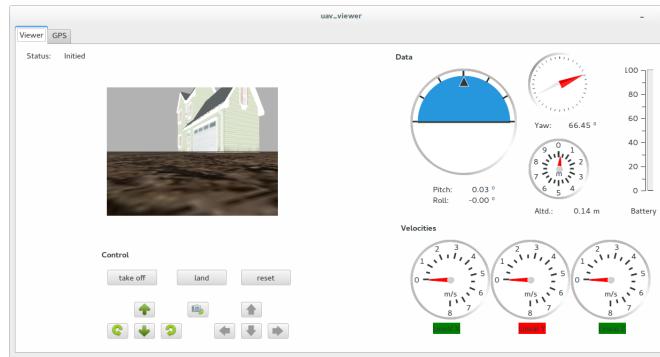


Figura 3.5: UAV Viewer

3.3.1 Interfaces ICE relativos a los drones

JdeRobot dispone de más de 30 interfaces pero los que se han utilizado durante este TFG que son los relacionados con los drones, son:

- Pose3D. Recoge los datos de actitud y la posición tridimensional de la aeronave. La posición se expresa como coordenadas cartesianas absolutas respecto de cierto sistema de referencia arbitrario y el ángulo se expresa en forma de cuaternión.

```
Pose3DData
{
    float x; /* x coord */
    float y; /* y coord */
    float z; /* z coord */
    float h; /* */
}
```

¹⁰<http://jderobot.org/Amartinflorido-tfg>

```

    float q0; /* qw */
    float q1; /* qx */
    float q2; /* qy */
    float q3; /* qz */
};


```

- CMDVel. Utilizado para enviar comandos de velocidad. Las variables “linear” afectan a la velocidad de traslacion relativa del drone, es decir, al avance o retroceso (linearX), al desplazamiento lateral (linearY) o al ascenso y descenso (linearZ). Por otro lado las variables “angular” se corresponden a las velocidades angulares, guiñada (angularX), alabeo (angularY) o cabeceo (angularZ).

```

class CMDVelData
{
    float linearX;
    float linearY;
    float linearZ;
    float angularX;
    float angularY;
    float angularZ;
};


```

- Extra. Utilizado principalmente para las órdenes de despegue y aterrizaje.

```

void land() - land drone.
void takeoff() - takeoff drone.
void reset()
void recordOnUsb(bool record)
void ledAnimation(int type, float duration, float req)
void flightAnimation(int type, float duration)
void flatTrim()
void toggleCam() - switch camera.


```

3.3.2 COMM

Biblioteca de mensajes que se sitúa entre las aplicaciones y los interfaces de comunicación, ICE o ROS. Esta biblioteca se ha desarrollado recientemente en JdeRobot como mediador entre las aplicaciones para comunicar indistintamente todos los drivers con la finalidad de abstraer del tipo de comunicación que se realice.

3.4 Biblioteca de comunicaciones ICE

ICE¹¹[13] (Internet Communications Engine) es un *middleware* orientado a objetos que proporciona llamadas a procedimientos remotos, *grid computing* y funcionalidad cliente / servidor desarrollada por ZeroC con una licencia GNU GPL y también una licencia privativa. ICE trabaja con objetos distribuidos, pueden estar en diferentes máquinas y comunicarse a través de la red a enviándose mensajes entre ellos.

ICE permite desarrollar aplicaciones distribuidas con un esfuerzo mínimo, abstraer al programador para que interactúe con una red de manera simple, usando interaces entre objetos distribuidos. El desarrollo de aplicaciones se enfoca así sólo en la lógica y no en las peculiaridades de la red. Es un *middleware* multilenguaje, podemos implementar clientes y servidores en diferentes lenguajes de programación y en diferentes plataformas. Está disponible para C++, Java, .Net languages, Objective-C, Python, PHP y Ruby, en la mayoría de los sistemas operativos. También hay una versión para teléfonos móviles llamada Ice-e.

JdeRobot puede utilizar ICE para la comunicación entre sus nodos, por lo tanto, la tarea de leer los valores de un sensor u órdenes de comando a un robot son tan simples como ejecutar un método de un objeto en la aplicación. Una ventaja significativa es la posibilidad de desarrollar aplicaciones independientes del contexto. Un programador puede desarrollar un driver en C++ para un robot particular que está incrustado en el robot, por otro lado, otro desarrollador puede programar una aplicación para el procesamiento de imágenes en Python que se ejecuta en un PC. Mediante ICE se pueden usar estas dos piezas, que originalmente eran independientes, como una sola aplicación sin preocuparse por las comunicaciones de bajo nivel.

La conexión entre el driver desarrollado en este TFG y las aplicaciones de control, así como la propia herramienta de teleoperación, se han realizado usando este *middleware*.

¹¹<https://zeroc.com/products/ice>

3.5 Python

Python[10] es un lenguaje de programación interpretado y multiplataforma que nació en los años 80 con idea de hacer más legible el código. Inicialmente se utilizaba para scripting, ha sabido crecer con los años y con la publicación de Python3 en 2009 ha recibido el impulso que necesitaba para ser hoy en día el 5º lenguaje más utilizado, por encima de PHP, .NET y Javascript, que baja hasta el 8º puesto según TIOBE en un estudio de Abril de 2017.

Tanto el driver como la herramienta desarrollados en este TFG se han creado empleando Python 2.7. El empleo de esta versión se debe a que JdeRobot es compatible con ROS y este *framework* únicamente es compatible en dicha versión de Python. Los motivos de utilizar Python son varios: mantiene el carácter multiplataforma de JdeRobot, su código es simple y legible y trabaja bien con dependencias muy utilizadas en robótica como OpenCV.

Capítulo 4

Driver MavLinkServer

Este capítulo describe el componente de software desarrollado para que las aplicaciones interactúen de forma autónoma con el dron. Es responsable de acceder a los sensores y actuadores del vehículo utilizando los mensajes de comunicación del protocolo MAVLink, y de traducirlos a las interfaces ICE de JdeRobot. Las aplicaciones de drones tendrán acceso a los sensores y actuadores del vehículo a través de esas interfaces.

La comunicación de nuestro driver con el dron es bidireccional. Dependiendo del módulo que se esté usando, esta comunicación puede ser bien dron-servidor, como por ejemplo recoger coordenadas GPS, bien servidor-dron, por ejemplo para ordenar comandos de velocidad, o bien bidireccional, para realizar la conexión con el servidor y envío de ACKs.



Figura 4.1: Diagrama comunicación

4.1 Diseño

MAVLinkServer es un driver basado en el software MAVProxy y desarrollado para actuar como un *middleware* de traducción. Ha sido diseñado como un controlador JdeRobot en lenguaje Python. Es responsable de mantener los canales de comunicación abiertos y actualizados, tanto en sentido ascendente como descendente.

En la imagen 4.2 se indica cómo es la relación entre el hardware y el software de cada componente, así como la comunicación interna o la comunicación hacia otros elementos.

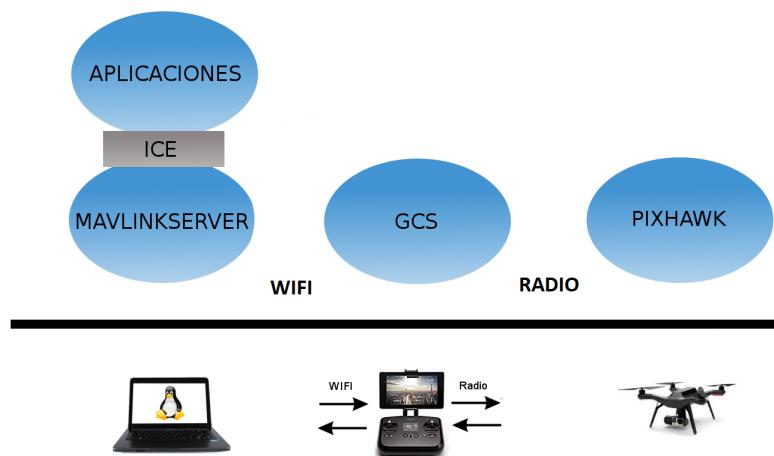


Figura 4.2: Diagrama de Hardware y Software

El driver se ha diseñado combinando tres bloques como se indica en la figura 4.3: el de configuración; el de diálogo con el drone usando MAVProxy; y el de diálogo con la aplicación usando interfaces ICE.

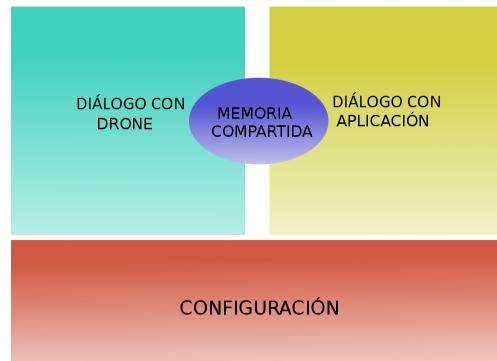


Figura 4.3: Diagrama de bloques en MAVLinkServer

El primer bloque es el que realiza la configuración necesaria para preparar todas las conexiones de los dos bloques siguientes. Toda la configuración se realiza a través de un único fichero para las interfaces ICE, ya que los módulos que se quieran añadir adicionalmente, se deben introducir mediante argumentos en la ejecución del script de arranque.

En el segundo bloque, **MAVLINKServer** establece la conexión con el piloto automático Pixhawk

a bordo del dron, mantiene el canal de comunicación operativo, adquiere, interpreta, crea y envía mensajes nuevos con la información solicitada u ordenada por la aplicación.

El código desarrollado en tercer bloque, está principalmente a cargo de la gestión de las interfaces ICE de JdeRobot. Es capaz de manejar la información proporcionada por el lado de MAVProxy. Regula la creación y modificación de las clases donde la información se almacena temporalmente y abre canales de comunicación ICE para hacer que el driver sea utilizable para aplicaciones JdeRobot.

Estos tres bloques en conjunto proporcionan un driver fiable y multi-compatible; **MAVLinkServer** puede conectarse con aplicaciones escritas en otros lenguajes, como C++, Python o Java, a través de las interfaces ICE de JdeRobot. La figura 4.4 representa un esquema de entradas y salidas dentro de **MAVLinkServer** y sus conexiones a otras aplicaciones.

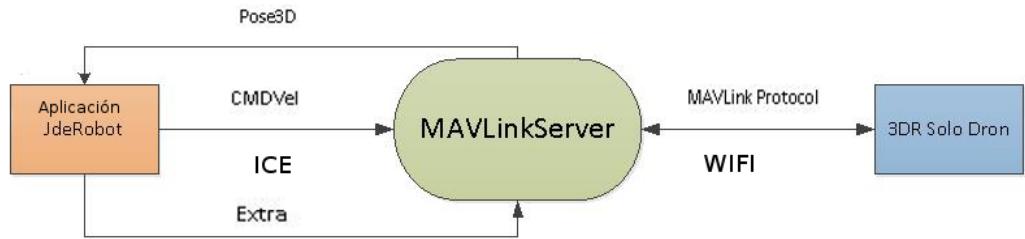


Figura 4.4: Diagrama de entradas y salidas bloques MAVLinkServer

La información que se maneja dentro del driver se gestiona a través de memoria compartida. Es necesario que la información sea bidireccional y que la memoria se esté leyendo y escribiendo de la manera más ágil posible. Todas estas acciones se realizan mediante semáforos de exclusión mutua que permiten un acceso seguro y sin condiciones de carrera. Por ejemplo, obtener la información de los sensores del dron, informar a la aplicación de su situación y que concurrentemente la aplicación envíe las acciones que debe realizar el dron sin que ninguna de dichas acciones se vea afectada por la otra.

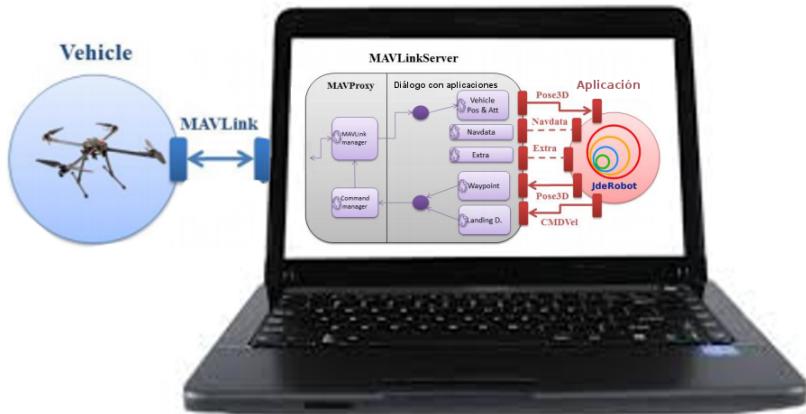


Figura 4.5: Diagrama bloques MAVLink-JdeRobot con detalle

Las interfaces JdeRobot utilizadas en este proyecto se muestran en la figura 4.6 son **Pose3D** (proporciona la posición y medidas del vehículo), **CMDVel** (para enviar los comandos de velocidad) y **Extra** (para despegue y aterrizaje).

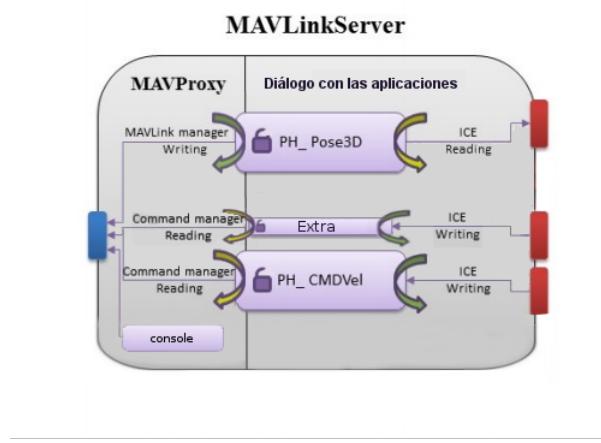


Figura 4.6: Diagrama de bloques de MavLinkServer

4.2 Diálogo con drone vía MavProxy

Antes de que el driver pueda ejecutar correctamente hay que configurar el software **MAVProxy** del sistema. En este apartado vamos a explicar todos los pasos a seguir.

4.2.1 Configuración

Para configurar **MAVProxy** debemos tener en cuenta de una serie de ficheros, llamados módulos, cada uno de los cuales facilita una utilidad diferente. Pueden ir desde conocer la batería, la altura del dron o configurar una web-cam.

Estos módulos se pueden descargar desde el repositorio oficial de **MAVProxy**¹ o crear una librería propia e implementarla. Los módulos que se han usado en este TFG para la comunicación vía **MAVProxy** son:

- **mavproxy_link:** Realiza la interconexión entre dron y MAVLinkServer.
- **mavproxy_arm:** Necesario para el despegue.
- **mavproxy_cmdlong:** Envía los comandos de velocidad al dron y órdenes de despegue y aterrizaje.
- **mavproxy_console:** Intérprete de comandos desde la consola de texto.

¹<https://github.com/ArduPilot/MAVProxy>

- **mavproxy_sensors**: Recoge información de los sensores.

4.2.2 Inicialización

El script de arranque se encarga de ejecutar todos los comandos preparatorios para MAVProxy y de lanzar el servidor. El script se encuentra en MAVProxy/MAVProxyWinLAN.sh. Antes de ejecutarlo debemos averiguar la IP que levanta el dron (en nuestro caso, con el 3DR Solo, dicha IP la levanta el mando) y modificar el script con la IP del dron. Una vez retocado hay que ejecutarlo sin parámetros adicionales.

Se necesita tener preinstalado tanto `pyserial` como una versión de `pyvmaulink` superior a la 1.1.50. Se realiza una descarga de todos los módulos, construye un directorio llamado MavProxy.egg en `/home/USER/.local/python2.7/site-packages` con el fin de tener almacenados todos los paquetes necesarios y con permisos suficientes.

La configuración que se puede modificar en el script de arranque está relacionada con el tipo de conexión que se quiere mantener con el dron. Esta conexión tiene valores por defecto pero se puede modificar si así se desea. Los comandos que se muestran en la tabla 4.1 son un ejemplo de la versatilidad que tiene el servidor. A través de los módulos descargados es posible acceder a los comandos que proporciona MAVLink. A continuación se proporciona un listado de los posibles comandos opcionales que se pueden añadir al script de arranque y una breve explicación de cada uno de ellos en la tabla 4.1:

Comando	Acción	Valor por defecto
master	Puerto maestro MAVLink y baudrate opcional	
udp	Arranca el servidor udp	TCP
tcp	Arranca el servidor tcp	TCP
out	Puerto de salida MAVLink	
baudrate		57600
sitl	Puerto de salida, esta opción únicamente es necesaria en caso de no tener disponible un dron.	
streamratedest	MAVLink stream rate	4
source-system	Código fuente MAVLink	255
source-component	Componente origen MAVLink	0
target-system	Sistema destino MAVLink	0
target-component	Componente destino MAVLink	0
logfile	Fichero de logs	mav.tlog
append-log	Añadir al fichero de log ya existente	False
continue	Continua el log	False
quadcopter	Usar acciones de control para cuadricopteros	False
setup	Arrancar en modo setup	False
nodtr	Deshabilitar DTR(Data Terminal Ready)	False
show-errors	Mostrar errores MAVLink	False
speech	Usar texto para hablar	False
aircraft	Establecer nombre para el dron (Visual en mensajes)	None
cmd	Comandos a ejecutar tras el arranque	None
console	Usar consola GUI para introducir comandos	
map	Carga un mapa de la zona	
load-module	Carga un modulo especifico, puede ser utilizado tantas veces como sea necesario separando con „,”)	
mav09	Usa protocolo MAVLink 0.9	False
auto-protocol	Auto detecta versión de protocolo MAVLink	False

Tabla 4.1: Tabla de comandos de arranque

Comando	Acción	Valor por defecto
nowait	No realiza comunicación continua con el dron (HearthBeat)	False
dialect	Dialecto MAVLink	ardupilotmega
rtscts	Habilita control de comunicación vía RT-S/CTS	False
mission	Nombre de la misión	None
daemon	Arranca en modo daemon, no muestra shell interactiva	False
profile	Arranca el Yappi python profiler	False
state-basedir	Directorio base para logs	None
version	Muestra información sobre la versión	False
default-modules	Módulos por defecto al iniciar.	log, wp, rally, fence, param, relay, tuneopt, arm, mode, calibration, rc, auxopt, misc, cmdlong, battery, terrain, output

Todos los módulos se cargan mediante una importación de paquetes. La lista que se carga mediante este procedimiento se divide en los módulos que se cargan por defecto, como pueden ser el que `link` o `sensors`, y otros opcionales que se pueden seleccionar desde la configuración, como por ejemplo `console`. A continuación se muestra la función que permite esta carga:

```
def load_module(modname, quiet=False):
    '''load a module'''
    modpaths = ['MAVProxy.modules.mavproxy_%s' % modname, modname]
    for (m,pm) in mpstate.modules:
        if m.name == modname:
            if not quiet:
                print("module %s already loaded" % modname)
            return False
    for modpath in modpaths:
        try:
            m = import_package(modpath)
            imp.reload(m)
```

```

        module = m.init(mpstate)
        if isinstance(module, mp_module.MPModule):
            mpstate.modules.append((module, m))
            if not quiet:
                print("Loaded module %s" % (modname,))
            return True
        else:
            ex = "%s.init did not return a
                  MPModule instance" % modname
            break
    except ImportError as msg:
        ex = msg
        if mpstate.settings.moddebug > 1:
            import traceback
            print(traceback.format_exc())
        print("Failed to load module: %s. Use 'set moddebug 3'
              in the MAVProxy console to enable traceback" % ex)
    return False

```

4.2.3 Funcionamiento continuo

Para llevar a cabo esta preparación el primer módulo que se carga es "Link", este paquete se encarga de la conexión entre el servidor y el dron. Es necesario conocer la dirección IP que levanta este dron, que ya habremos configurado en el paso 4.2.1.

Tras lograr la conexión se establece un periodo de chequeo necesario para no perder la conexión con el dron en caso de llegar a una distancia límite o un fallo de conexión, en cuyo caso se procede a detener el dron. Periódicamente se debe realizar una comunicación entre el dron y el servidor, aunque no se llegue a mandar ningún comando, ya sea de velocidad o algún tipo de acción. El servidor por su parte comunicará que aún está conectado. Después de un tiempo sin conexión, aproximadamente unos 10-15 segundos, el dron procederá a realizar un aterrizaje.

```

def periodic_tasks():
    '''run periodic checks'''
    if mpstate.status.setup_mode:
        return
    if (mpstate.settings.compdebug & 2) != 0:
        return

```

```

if mpstate.settings.heartbeat != 0:
    heartbeat_period.frequency = mpstate.settings.heartbeat

if heartbeat_period.trigger() and mpstate.settings.heartbeat != 0:
    mpstate.status.counters['MasterOut'] += 1
    for master in mpstate.mav_master:
        send_heartbeat(master)
if heartbeat_check_period.trigger():
    check_link_status()

set_stream_rates()

for (m,pm) in mpstate.modules:
    if hasattr(m, 'idle_task'):
        try:
            m.idle_task()
        except Exception as msg:
            if mpstate.settings.moddebug == 1:
                print(msg)
            elif mpstate.settings.moddebug > 1:
                exc_type, exc_value, exc_traceback = sys.exc_info()
                traceback.print_exception(exc_type, exc_value,
                                          exc_traceback, limit=2, file=sys.stdout)

# also see if the module should be unloaded:
if m.needs_unloading:
    unload_module(m.name)

```

En el lado de comunicación con el cuadricóptero, el driver sólo tiene una conexión con el dron a través de una IP y un puerto que se especifican en el arranque. Este bloque de **MAVLinkServer** está programado con tres hilos separados, uno para cada información relevante a intercambiar con el dron: posición, comandos de velocidad y comandos extra. La gestión de estos 3 hilos, respecto al único puerto abierto con el dron, se basa en prioridades. El hilo de comandos Extra predomina sobre los otros 2 debido a la importancia de las órdenes que envía. El segundo hilo prioritario es el que envía los comandos de velocidad. El hilo menos prioritario es el de recepción de actualizaciones

de posición.

```
mpstate.status.thread = threading.Thread(target=main_loop,
                                         name='main_loop')

mpstate.status.thread.daemon = True
mpstate.status.thread.start()

# Open an MAVLink TX communication and
# leave it open in a parallel thread

PoseTheading = threading.Thread(target=sendCMDVel2Vehicle,
                                 args=(PH_CMDVel, PH_Pose3D, ),
                                 name='TxCMDVel_Theading')
PoseTheading.daemon = True
PoseTheading.start()

# Open an MAVLink TX communication and
# leave it open in a parallel thread

PoseTheading = threading.Thread(target=landDecision,
                                 args=(PH_Extra, ),
                                 name='LandDecision2Vehicle_Theading')
PoseTheading.daemon = True
PoseTheading.start()
```

A continuación se detalla la relación entre cada interfaz ICE y el módulo `MAVProxy` que se ve afectado:

- CMDVel y Extra: Como se ha comentado en la sección 4.2.1, el módulo afectado es `mavproxy_cmdlong`. Este módulo nos dará lo necesario para poder mover el dron en los ejes x,y,z y sobre el yaw. La velocidad máxima que se le puede dar al dron a través del interfaz ICE en cada dirección viene dado en una escala de 0 a 1. Aparte, nos dará la facilidad de despegar y de aterrizar. Debido al protocolo MAVLink, el sistema de despegue se divide en 3 estados, que agruparemos bajo la misma orden ICE de despegue. Durante la fase de despegue el dron no admite ningún comando a excepción del comando "land" para aterrizar, o en caso del despegue, para detener el despegue. Este despegue dura aproximadamente unos 10 segundos hasta que se estabiliza en el aire por motivos de seguridad.

1. El arranque de las hélices.

2. Despegue del dron.
3. Habilitar comandos de velocidad.

```

def sendCMDVel2Vehicle(CMDVel, Pose3D):
    absolute = 0
    relative = 1

    while True:

        CMDVel2send = CMDVel.getCMDVelData()
        Pose3D2send = Pose3D.getPose3DData()
        #print(Pose3D2send)
        NEDvel = body2NED(CMDVel2send, Pose3D2send) # [x,y,z]
        linearXstring = str(NEDvel[0])
        linearYstring = str(NEDvel[1])
        linearZstring = str(NEDvel[2])

        #CMDVel.angularZ -1 y 1

        angular = CMDVel.angularZ

        if angular >= 0:
            direction = str(1)
        else:
            angular = -angular
            direction = str(-1)

        angularZstring = str(angular*30)

        movement = str(relative)

        velocitystring = 'velocity '+ linearXstring + ' ' +
                          linearYstring + ' ' +
                          linearZstring
        angularString = 'setyaw ' + angularZstring + ' ' +
                        direction + ' ' + movement

```

```

process_stdin(velocityString)
process_stdin/angularString)

```

- Pose3D: El módulo encargado es `mavproxy_sensors`. Este módulo nos indicará la posición del dron en todo momento, así como su orientación y altitud. Esta conexión nos va a servir tanto a la hora de aterrizar y despegar usando el otro módulo, para saber si puede aterrizar en un determinado momento o debe disminuir su altura antes de parar los motores.

```

def cmd_sensors(self, args):
    '''show key sensors'''
    if self.master.WIREPROTOCOLVERSION == '1.0':
        gps_heading = self.status.msgs['GPS_RAW_INT'].cog * 0.01
    else:
        gps_heading = self.status.msgs['GPS_RAW'].hdg

    self.console.writeln("heading: %u/%u      alt: %u/%u
                         r/p: %u/%u speed: %u/%u   thr: %u" %)
    self.status.msgs['VFR_HUD'].heading,
    gps_heading,
    self.status.altitude,
    self.gps_alt,
    math.degrees(self.status.msgs['ATTITUDE'].roll),
    math.degrees(self.status.msgs['ATTITUDE'].pitch),
    self.status.msgs['VFR_HUD'].airspeed,
    self.status.msgs['VFR_HUD'].groundspeed,
    self.status.msgs['VFR_HUD'].throttle))

```

4.2.4 Comunicación vía consola

Si se ha configurado la comunicación con el dron usando el comando de arranque "console" entonces se puede interactuar en cualquier momento con él a través de una consola de texto en la cual se pueden especificar los comandos de la tabla 4.2. Esta consola se utiliza como mecanismo alternativo de comunicación con el dron, y es muy útil para depurar o recuperar un estado razonable si la comunicación desde la aplicación y el driver se ha descontrolado. Estos comandos tienen una labor de modificar la experiencia de vuelo del dron o de mostrar los valores que nos pueden proporcionar sus sensores:

Comando	Acción
reboot	Reinicia el dron
arm	Habilita los medidores propios del dron. Ejemplo de ejecución: check (all —baro—compass—gps—ins—params —rc—voltage—battery), uncheck (all—baro—compass —gps—ins—params—rc— voltage—battery), list, throttle, safetyon, safetyoff (arm throttle arranca hélices del dron)
disarm	Detiene los motores del dron
takeoff	Despegue del dron
land	Aterrizaje del dron
mode	Cambia el modo de vuelo
velocity	Establece una velocidad en los ejes x,y,z
parachute	Habilita un aterrizaje del dron si pierde conexión o la batería es baja. Ejemplo de ejecución: parachute [enable—disable—release]
bat	Muestra batería del dron
alt	Muestra altitud del dron

Tabla 4.2: Tabla de comandos de la consola

4.3 Diálogo con las aplicaciones ICE

MAVLINKServer lanza varios hilos para canales de comunicación ICE, uno para cada tipo de información. A pesar de que sólo Pose3D, CMDVel y Extra se utilizan en este TFG, el driver ofrece las interfaces restantes para compatibilidad y usos futuros. Dichos hilos de comunicación realizan operaciones de lectura y escritura en memoria compartida, las necesarias para la comunicación entre los distintos interfaces ICE y el puerto de comunicaciones MAVProxy con el dron.

4.3.1 Configuración

El fichero de configuración que acepta el servidor únicamente se limita a establecer los puertos mediante los cuales se va a realizar la comunicación ICE con las aplicaciones. Por ejemplo, en la muestra siguiente los comandos de posición en el puerto 9998, de velocidad en el 9997 y las órdenes de despegue o aterrizaje en el 9996.

Pose3D :

```
Server: 1 # 0 -> Deactivate , 1 -> Ice , 2 -> ROS
Proxy: "default -h 0.0.0.0 -p 9998"
Topic: "/MavLink/Pose3D"
Name: MavLinkPose3d
```

```

CMDVel:
  Server: 1 # 0 -> Deactivate , 1 -> Ice , 2 -> ROS
  Proxy: "default -h 0.0.0.0 -p 9997"
  Topic: "/MavLink/CMDVel"
  Name: MavLinkCMDVel

```

```

Extra:
  Server: 1 # 0 -> Deactivate , 1 -> Ice , 2 -> ROS
  Proxy: "default -h 0.0.0.0 -p 9996"
  Topic: "/MavLink/Extra"
  Name: MavLinkExtra

```

4.3.2 Inicialización

El primer paso es definir las interfaces ICE correspondientes. **MAVLinkServer** suministra todas las interfaces JdeRobot para el acceso de drones desde cualquier componente externo. Estas interfaces permiten acceder a todos los datos sensoriales o actuadores de los drones. Cada interfaz ICE comprueba por separado en su hebra, la configuración y se lanza la conexión con los parámetros introducidos. A continuación se muestra el código de inicialización de la aplicación en el cual se inicia la conexión:

```

def openPose3DChannel(Pose3D , cfg ):
    status = 0
    ic = None
    Pose2Tx = Pose3D #Pose3D .getPose3DData()
    try :
        id = Ice .InitializationData()
        ic = Ice .initialize(None , id)

        endpoint = cfg .getProperty("Pose3D .Proxy")
        name = cfg .getProperty("Pose3D .Name")

        adapter = ic .createObjectAdapterWithEndpoints("Pose3D" , endpoint)

        object = Pose2Tx
        adapter.add(object , ic .stringToIdentity("Pose3D"))

```

```

        adapter.activate()
        ic.waitForShutdown()

    except:
        traceback.print_exc()
        status = 1

    if ic:
        # Clean up
        try:
            ic.destroy()
        except:
            traceback.print_exc()
            status = 1

    sys.exit(status)

def openCMDVelChannel(CMDVel, cfg):
    status = 0
    ic = None
    CMDVel2Rx = CMDVel #CMDVel.getCMDVelData()
    try:
        id = Ice.InitializationData()
        ic = Ice.initialize(None, id)

        endpoint = cfg.getProperty("CMDVel.Proxy")
        name = cfg.getProperty("CMDVel.Name")

        adapter = ic.createObjectAdapterWithEndpoints("CMDVel", endpoint)

        object = CMDVel2Rx
        adapter.add(object, ic.stringToIdentity("CMDVel"))
        adapter.activate()
        ic.waitForShutdown()

    except:
        traceback.print_exc()
        status = 1

```

```

if ic:
    # Clean up
    try:
        ic.destroy()
    except:
        traceback.print_exc()
        status = 1

    sys.exit(status)

def openExtraChannel(Extra, cfg):
    status = 0
    ic = None
    Extra2Tx = Extra
    try:
        id = Ice.InitializationData()
        ic = Ice.initialize(None, id)

        endpoint = cfg.getProperty("Extra.Proxy")
        name = cfg.getProperty("Extra.Name")

        adapter = ic.createObjectAdapterWithEndpoints("Extra", endpoint)

        object = Extra2Tx
        adapter.add(object, ic.stringToIdentity("Extra"))
        adapter.activate()
        ic.waitForShutdown()

    except:
        traceback.print_exc()
        status = 1

    if ic:
        # Clean up
        try:

```

```

        ic . destroy ()
    except :
        traceback . print_exc ()
        status = 1

    sys . exit (status)

```

4.3.3 Funcionamiento continuo

Todas las comunicaciones que se realizan mediante ICE se administran desde tres hebras, una por cada interfaz utilizada. Cada hebra hace uso de su propia función donde se realiza la configuración de ICE respectiva. Allí se realiza la publicación del interfaz y es importante garantizar qué datos se envían, para que otras aplicaciones obtengan la información correctamente y garanticen la compatibilidad. El mismo procedimiento se sigue para todas las interfaces ICE con las aplicaciones.

La conversión entre órdenes ICE y comandos MAVLink se realiza por separado en cada hebra y accede a módulos MAVProxy diferentes.

A continuación se muestra el ejemplo de la interfaz Pose3D:

```

import jderobot , time , threading

lock = threading.Lock()

class Pose3DI(jderobot.Pose3D):

    def __init__(self , _x , _y , _z , _h , _q0 , _q1 , _q2 , _q3 ):

        self.x = _x
        self.y = _y
        self.z = _z
        self.h = _h
        self.q0 = _q0
        self.q1 = _q1
        self.q2 = _q2
        self.q3 = _q3

    print ("Pose3D start")

```

```

def setPose3DDData(self, data, current=None):

    lock.acquire()

    self.x = data.x
    self.y = data.y
    self.z = data.z
    self.h = data.h
    self.q0 = data.q0
    self.q1 = data.q1
    self.q2 = data.q2
    self.q3 = data.q3

    lock.release()

    return 0

def getPose3DDData(self, current=None):

    time.sleep(0.05) # 20Hz (50ms) rate to tx Pose3D
    lock.acquire()
    data = jderobot.Pose3DDData()
    data.x = self.x
    data.y = self.y
    data.z = self.z
    data.h = self.h
    data.q0 = self.q0
    data.q1 = self.q1
    data.q2 = self.q2
    data.q3 = self.q3

    lock.release()

    return data

```

Esta clase hereda "jderobot.Pose3D" y se definen las funciones correspondientes. Ha sido necesario el uso de "bloqueos" (**locks**) para proteger los datos almacenados en la clase y garantizar un acceso en exclusión mutua. Esto se debe a que **MAVLinkServer** actualiza constantemente la información provista por los sensores y también la publica constantemente a través de ICE. Esto podría causar condiciones de carrera. Con el uso de los bloqueos en las clases la información no se puede leer mientras otra tarea está escribiendo en ella y viceversa, asegurando la gestión correcta de la información en exclusión mutua. Estos datos Pose3D se cogen del drone vía **MAVProxy** (`self`) y se enganchan con la función que responde a las peticiones ICE (`data`).

Las interfaces **CMDVel** y **Extra** que se muestran a continuación tienen la misma estructura que **Pose3D**, también con el uso de cierres.

```
import jderobot, time, threading

lock = threading.Lock()

class CMDVelII(jderobot.CMDVel):

    def __init__(self, lx, ly, lz, ax, ay, az):

        self.linearX = lx
        self.linearY = ly
        self.linearZ = lz
        self.angularX = ax
        self.angularY = ay
        self.angularZ = az

    #print ("cmdvel start")

    #def __del__():

    #print ("cmdvel end")

    def setCMDVelData(self, data, current=None):

        lock.acquire()
```

```

        self.linearX = data.linearX
        self.linearY = data.linearY
        self.linearZ = data.linearZ
        self.angularX = data.angularX
        self.angularY = data.angularY
        self.angularZ = data.angularZ

        lock.release()

    return 0

def getCMDVelData(self, current=None):

    time.sleep(0.05) # 20Hz (50ms) rate to rx CMDVel

    lock.acquire()

    data = jderobot.CMDVelData()
    data.linearX = self.linearX
    data.linearY = self.linearY
    data.linearZ = self.linearZ
    data.angularX = self.angularX
    data.angularY = self.angularY
    data.angularZ = self.angularZ
    lock.release()

    return data

```

```

import jderobot, time, threading

lockLand = threading.Lock()
lockTakeOff = threading.Lock()

class ExtraI(jderobot.ArDroneExtra):

    def __init__(self):

```

```

        print ("Extra start")
        self.landDecision = False
        self.takeOffDecision = False

    def land(self,xxx):
        self.setLand(True)
        lockLand.acquire()
        landDecision = self.landDecision
        lockLand.release()

        return landDecision

    def takeoff(self,xxx):
        self.setTakeOff(True)
        lockTakeOff.acquire()
        takeOffDecision = self.takeOffDecision
        lockTakeOff.release()

        return takeOffDecision

    def setLand(self,decision):
        lockLand.acquire()
        self.landDecision = decision
        lockLand.release()

    def setTakeOff(self, decision):
        lockTakeOff.acquire()
        self.takeOffDecision = decision
        lockTakeOff.release()

    def setExtraData(self, data, current=None):

        lockLand.acquire()
        self.landDecision = data.landDecision
        lockLand.release()

```

```
lockTakeOff.acquire()
self.takeOffDecision = data.takeOffDecision
lockTakeOff.release()

return 0
```

Capítulo 5

Visor UAV Viewer.py

Una estación en tierra o GCS (*Ground Control Station*) es una estación base donde se recibe la información generada por un dron y con la que se puede monitorizar y controlar toda la actividad de este vehículo. Estas estaciones pueden ser fijas o móviles y permiten al operario que maneje el dron conocer su estado de una manera simple.

En el entorno de JdeRobot existía previamente una estación en tierra para drones desarrollada en C++ llamada **UAV-Viewer**. Debido a la necesidad de crear un nuevo interfaz que se asemejara más a un mando de un dron real, se decidió implementar una nueva versión. Además ha servido de validación experimental del driver **MavLinkServer** puesto que se comunica directamente con él para enviarle comandos de movimiento así como recibir posición del GPS.

5.1 Diseño

El desarrollo de la nueva aplicación **UAV-Viewer.py** se ha realizado en Python 2.7 en vez de C++ por sencillez para que el entorno JdeRobot tuviera más variedad en sus aplicaciones, en vez de hacer un cambio en la interfaz gráfica de la versión de C++. Las principales diferencias entre ambos visores son el lenguaje y el interfaz gráfico.

El diagrama de entradas y salidas del nodo **UAV-Viewer.py** se muestra en la Figura 5.1, en este caso conectado a dos drivers: MAVLink para conectarse con el dron y **CameraServer** para conectarse con la cámara a bordo.

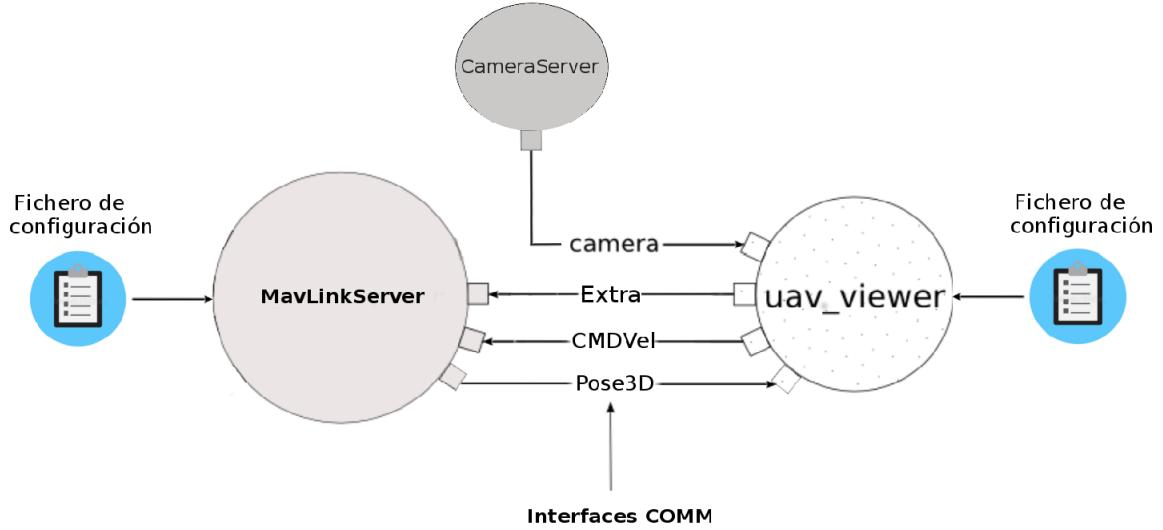


Figura 5.1: Diagrama de entradas y salidas de UAV-Viewer.py

La herramienta se ha diseñado combinando tres bloques como se indica en la figura 5.2: el de configuración; el de diálogo con el drone, usando interfaces ICE; y el de diálogo con el interfaz de usuario.

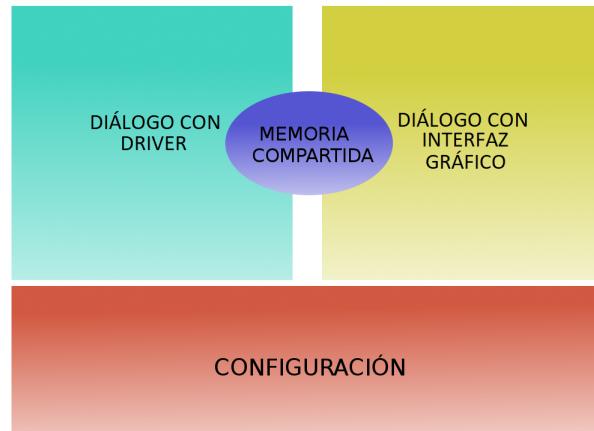


Figura 5.2: Diagrama de bloques de UAV-Viewer.py

Existen dos hebras dentro de la aplicación: una que controla la interfaz de usuario que se encarga de enviar la información de los joysticks y los botones; y otra hebra que recibe dicha información, se adecúa dependiendo el estado en el que se encuentre y envía la acción requerida al driver del dron.

5.2 Comunicación con los drivers

La biblioteca COMM permite comunicación tanto vía ICE como ROS. Todas las comunicaciones que se realizan mediante COMM, se administran desde dos hebras, una para realizar la comunicación con el dron y la otra administra el interfaz de usuario. Las interfaces ICE suministran todas las utilidades para el acceso a drones desde cualquier componente externo. Estas interfaces permiten acceder a todos los datos sensoriales o actuadores de los drones.

Para teleoperar el dron la interfaz gráfica captura los eventos. Estos eventos son creados por los joysticks que envía una señal constante con la velocidad indicada para cada uno de los ejes. La interfaz proporcionada además de ofrecer el envío de comandos al dron para su control, es capaz de proporcionar los datos de los sensores. A estos datos se los conoce como navdata o datos de navegación. El hilo que comunica `UAV-Viewer.py` con el dron hace uso de ésta interfaz para recibir dichos datos. A continuación se muestra el código de inicialización de la aplicación en el cual se inicia la conexión con la biblioteca de mensajes COMM:

```
#Lectura de la configuracion
cfg = config.load(sys.argv[1])

#Inicializacion de las comunicaciones
jdrc= comm.init(cfg , 'UAVViewer')

camera = jdrc.getCameraClient("UAVViewer.Camera")
navdata = jdrc.getNavdataClient("UAVViewer.Navdata")
pose = jdrc.getPose3dClient("UAVViewer.Pose3D")
cmdvel = jdrc.getCMDVelClient("UAVViewer.CMDVel")
extra = jdrc.getArDroneExtraClient("UAVViewer.Extra")
```

La adquisición de imágenes la realiza la hebra encargada de la comunicación con el dron dentro de un bucle de control independiente a la hebra encargada de la interfaz gráfica. De este modo la hebra encargada de la interfaz gráfica consultará a la hebra que controla el dron periódicamente en busca de nuevas imágenes. Cuando las obtenga, refrescará la imagen que muestra en un determinado periodo de tiempo que puede ser configurado.

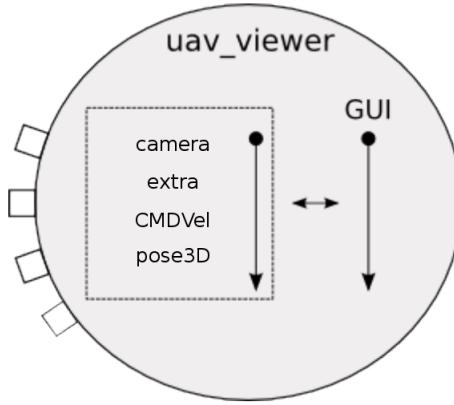


Figura 5.3: Arquitectura software de dos Hilos en Uav-Viewer.py

Las acciones que envía la interfaz gráfica están marcadas por ciclos de reloj fijos, cada ciclo actualiza la información necesaria en el interfaz correspondiente. Ha sido necesario el uso de "bloqueos" (*locks*) para proteger los datos almacenados en la clase y garantizar un acceso en exclusión mutua, de igual manera que se utiliza en MAVLinkServer. Toda la memoria compartida se administra desde las clases correspondientes y cada interfaz se ocupa de recibir los datos de la interfaz de usuario.

5.3 Interfaz Gráfica

La aplicación ha sido desarrollada en Python utilizando para el apartado gráfico la librería para interfaces de usuario Qt. Se ha dividido en tres ventanas: la figura 5.4 corresponde a la ventana principal, donde se teleopera el dron. La segunda ventana representada en la figura 5.5, se corresponde con la imagen en tiempo real de la cámara a bordo del dron. Por último, la tercera ventana muestra la información recogida por los sensores del dron, mostrado en la figura 5.6.

A continuación se muestra el código de inicialización de la interfaz de usuario junto a las conexiones COMM levantadas anteriormente:

```
#Inicializacion del interfaz grafico y su hebra
app = QApplication(sys.argv)
frame = MainWindow()
frame.setCamera(camera)
frame.setNavData(navdata)
frame.setPose3D(pose)
frame.setCMDVel(cmdvel)
frame.setExtra(extra)
frame.show()
```

```

t2 = ThreadGUI(frame)
t2.daemon=True
t2.start()

```

La ventana principal se ha realizado de tal manera que se asemeje lo más posible a los mandos de los drones físicos. Las crucetas simulan los joystick de control del dron, de tal forma que el joystick izquierdo controla la altura y el giro (o Yaw), y el joystick derecho controla los ejes X e Y.

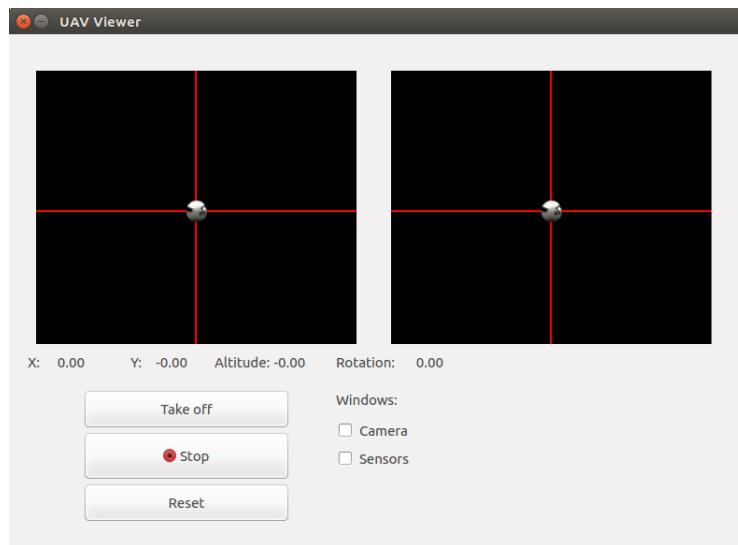


Figura 5.4: Interfaz gráfica de uav-viewer.py para teleoperación

A su vez, esta ventana dispone de tres botones:

- **Take off-Land:** Este botón permite despegar o aterrizar el dron en función del estado de vuelo en el que se encuentre.
- **Stop:** Cuadra los joysticks en la posición (0,0) para detener el dron por completo.
- **Reset:** Reinicia los valores por defecto: si hubiera alguna ventana auxiliar abierta, cámara o sensores, se cerrarían; además, realiza la misma acción del botón **Stop**; por último, respetaría el estado de vuelo del dron (la orden **Take off-Land** no se vería alterada por la acción de este botón).

```

def __init__(self, parent=None):
    super(MainWindow, self).__init__(parent)
    self.setupUi(self)
    self.teleop=TeleopWidget(self,0)

```

```

self.teleop1=TeleopWidget(self,1)
self.tlLayout.addWidget(self.teleop)
self.tlLayout_1.addWidget(self.teleop1)
self.teleop.setVisible(True)

self.record = False

self.updGUI.connect(self.updateGUI)

self.cameraCheck.stateChanged.connect(self.showCameraWidget)
self.sensorsCheck.stateChanged.connect(self.showSensorsWidget)

self.cameraWidget = CameraWidget(self)
self.sensorsWidget = SensorsWidget(self)

self.cameraCommunicator = Communicator()
self.trackingCommunicator = Communicator()

self.stopButton.clicked.connect(self.stopClicked)
self.resetButton.clicked.connect(self.resetClicked)
self.takeoffButton.clicked.connect(self.takeOffClicked)
self.takeoff = False
self.reset = False

```

Además de esta ventana principal, la aplicación desarrollada también implementa la visualización de cámara y de los sensores a bordo. Obtiene las imágenes a través de la interfaz camera. Con la imagen y los metadatos de ésta recuperados desde el driver que suministra las imágenes, **UAV-Viewer.py** la transforma en una imagen compatible con Qt. En función de la cámara activa el tamaño de la ventana donde se muestran las imágenes cambiará para ajustarse al ancho y alto de la imagen obtenida.

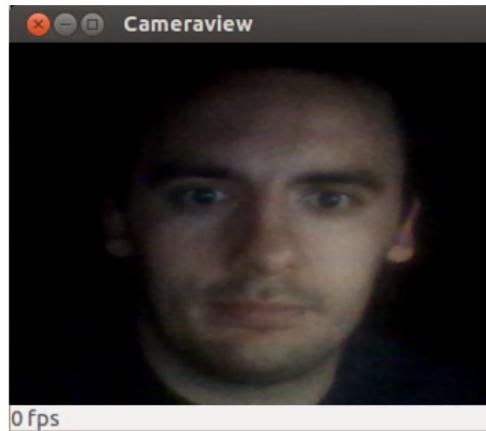


Figura 5.5: Interfaz gráfico de UAV-Viewer.py para mostrar la cámara

En la figura 5.6 se puede apreciar cómo el hilo que gestiona la interfaz de usuario rellena varios objetos gráficos con los datos sensoriales obtenidos a través del driver del dron. Podemos ver el porcentaje de batería restante, la altitud del dron, con el indicador de actitud se visualiza fácilmente el alabeo y el cabeceo. Además cuenta con tres velocímetros para indicar la velocidad medida en cada eje. Si esta velocidad es positiva la etiqueta de la velocidad será verde, mientras que si la velocidad es negativa, la etiqueta será roja.



Figura 5.6: Interfaz gráfico de UAV-Viewer.py para mostrar los sensores

5.4 Fichero de configuración

La aplicación `UAV-Viewer.py` utiliza ficheros de configuración YAML. A continuación se muestra el fichero de configuración en el cuál se determinan en qué puertos específicos se comunica con los drivers, `MAVLinkServer` y `CameraServer` para intercambiar datos sensoriales y órdenes de control.

Camera :

```
Server: 1 # 0 -> Deactivate , 1 -> Ice , 2 -> ROS
Proxy: "default -h 0.0.0.0 -p 9999"
Format: RGB8
Topic: "/MavLink/image_raw"
Name: MavLinkCamera
```

Pose3D :

```
Server: 1 # 0 -> Deactivate , 1 -> Ice , 2 -> ROS
Proxy: "default -h 0.0.0.0 -p 9998"
Topic: "/MavLink/Pose3D"
Name: MavLinkPose3d
```

CMDVel:

```
Server: 1 # 0 -> Deactivate , 1 -> Ice , 2 -> ROS
Proxy: "default -h 0.0.0.0 -p 9997"
Topic: "/MavLink/CMDVel"
Name: MavLinkCMDVel
```

Navdata :

```
Server: 1 # 0 -> Deactivate , 1 -> Ice , 2 -> ROS
Proxy: "default -h 0.0.0.0 -p 9996"
Topic: "/MavLink/Navdata"
Name: MavLinkNavdata
```

Extra :

```
Server: 1 # 0 -> Deactivate , 1 -> Ice , 2 -> ROS
Proxy: "default -h 0.0.0.0 -p 9995"
Topic: "/MavLink/Extra"
Name: MavLinkExtra
```

5.5 Experimentos

Tanto el driver `MAVLinkServer` como el visor `UAV-Viewer.py` que se han desarrollado se han validado con el 3DR Solo Drone real.

Las primeras pruebas que se realizaron para validar el funcionamiento de la herramienta y el driver fueron la realización de movimientos de traslación relativos, como el avance o retroceso, ascenso o descenso, desplazamiento lateral en cualquier dirección y velocidades angulares (guiñada, alabeo, cabeceo). Se muestran en la figura 5.7 y también en el video <https://www.youtube.com/watch?v=j5P7QywcfHk>. Se comprobó que los datos sensoriales IMU abordo (que van vía `Pose3D`) son recogidos correctamente por el driver y mostrados adecuadamente en la interfaz gráfica del teleoperador.

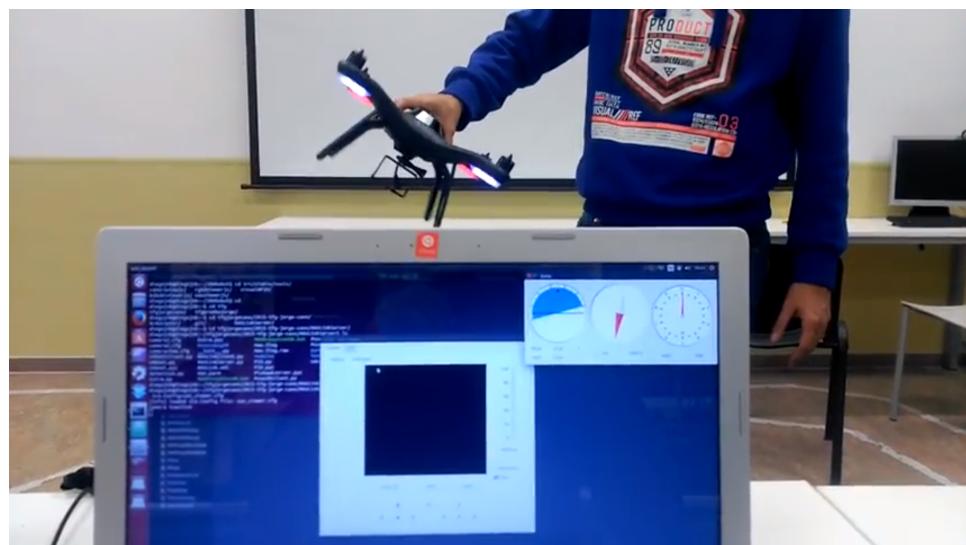


Figura 5.7: Captación de movimientos del drone

El desarrollo completo de todos los interfaces ICE de la herramienta se verificó experimentalmente en el simulador `Gazebo` con un drone simulado como se muestra en la figura 5.8.

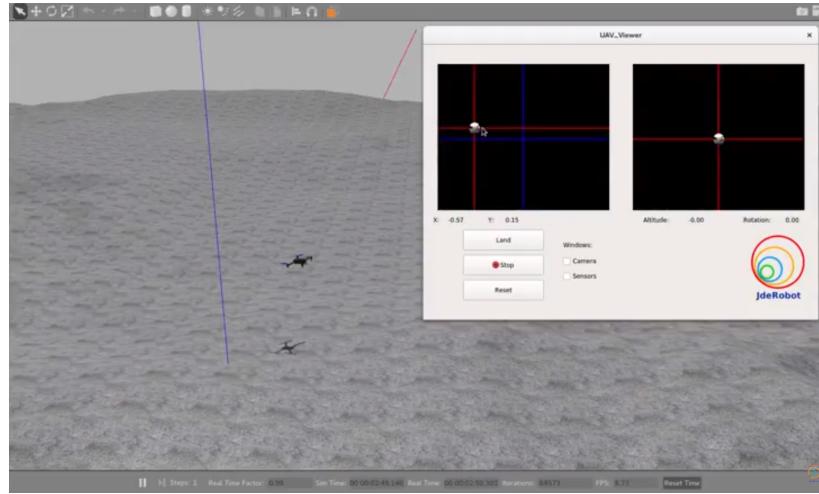


Figura 5.8: Experimento de vuelo simulado

A continuación se validaron las órdenes de aterrizaje y despegue. Para ello, se probó tanto en sitios abiertos como cerrados, pudiendo estudiar el efecto de las condiciones climáticas y las turbulencias generadas sobre la actuación del dron. Por último, se realizó una validación de los comandos de velocidad en dos pasos: el primero de ellos era comprobar que las órdenes que se enviaban y se recibían correctamente en el dron real; el segundo, como se ilustra en la figura 5.9 y en el video <https://www.youtube.com/watch?v=UtUWMH6s2Yk>, consistía en probar el comportamiento de estos comandos en pleno vuelo para tomar medidas y a continuación realizar ajustes.



Figura 5.9: Experimento de vuelo real

En este experimento se verificaron las órdenes de aterrizaje y despegue, y sobre todo el funcionamiento correcto de las órdenes relativas de movimiento: desplazamientos laterales, desplazamientos frontales, desplazamientos verticales y giro alrededor de la vertical (yaw).

Capítulo 6

Conclusiones

A lo largo del documento se han descrito los objetivos de este Trabajo Fin de Grado, la infraestructura hardware y software utilizada, el software desarrollado que incluye el driver `MAVLinkServer` en JdeRobot, y una versión nueva de la herramienta `UAV-Viewer.py` para la teleoperación. Se ha detallado el diseño, la implementación del driver y de la herramienta de la teleoperación. En este capítulo se exponen las conclusiones obtenidas durante todo el proceso y los trabajos futuros que pueden continuar o mejorar al actual.

6.1 Conclusiones

El principal objetivo de este trabajo era proponer una solución para el desarrollo de nuevas aplicaciones en drones cuya placa controladora se comunique mediante el protocolo MAVLink. Teniendo en cuenta los resultados obtenidos durante las pruebas, detallados en el capítulo de 5, se puede afirmar que se ha logrado el objetivo propuesto.

Se han cubierto todas las fases de un proyecto software, desde el análisis de requisitos y su especificación, hasta la realización de pruebas unitarias y de integración, pasando por el diseño e implementación.

- El primer subobjetivo del proyecto era el desarrollo dentro de la plataforma JdeRobot del driver que permitiera el acceso a los sensores y actuadores del cuadricóptero 3DR Solo Drone. Se comenzó con un estudio de las plataformas disponibles para el manejo y adquisición de los datos de los sensores de 3DR Solo Dron, y del SDK oficial de MAVLink. La base de este proyecto se basa en una ampliación del Trabajo Fin de Grado de Jorge Cano [5], cuya versión sirvió como base pero se decidió reprogramarla desde cero. Los principales motivos fueron que no estaban bien diferenciadas las partes de servidor y cliente, lo cual era fuente de problemas a la hora de codificar futuras mejoras. Se decidió desarrollar un envoltorio que permitiera la interacción de aplicaciones JdeRobot con el SDK oficial de MAVLink. El resultado es el driver `MAVLinkServer` en Python 2.7 (también se hizo una versión en Python 3.5, pero fue necesario

reprogramarla en 2.7 para integrar en la versión actual de JdeRobot, por compatibilidad con ROS Kinetic).

Una de las mayores ventajas de **MAVLinkServer** es el uso de interfaces de mensajes ICE, válido para interfaces de imágenes o de control del dron desde otros nodos de una manera sencilla y eficiente. La arquitectura distribuida de ICE y la facilidad para implementar componentes en casi cualquier lenguaje de programación, hacen que interactuar con el 3DR Solo Dron a través del componente **MAVLinkServer** sea una tarea sencilla, reduciendo la dificultad que supone trabajar con los dispositivos hardware a bajo nivel.

- El segundo subobjetivo del proyecto era el diseño y desarrollo de una nueva herramienta de teleoperación que gobernase el movimiento del cuadricóptero. Se encontró el problema de que la interfaz ya existente se diferenciaba mucho de un control natural. Esta nueva versión acerca a gente interesada en drones a la plataforma JdeRobot de una manera más simple. La nueva herramienta se llama **UAV-Viewer.py** y ha sido probada y utilizada en el Trabajo Fin de Grado de Jorge Vela[6].

UAV-Viewer.py utiliza las interfaces ICE que **MAVLinkServer** ofrece, permitiendo la teleoperación del dron y el visionado de sus sensores. Funciona con diferentes modelos de drones, siempre y cuando sus drivers ofrezcan las interfaces soportadas. La interfaz gráfica de este componente está descrita en un fichero XML, lo que permite una rápida modificación de la estética e incluso la funcionalidad. Los distintos hilos que componen la herramienta permiten atender debidamente la interfaz de usuario sin provocar bloqueos en la aplicación. Su diseño concurrente le permite desacoplar la interfaz gráfica de usuario de los hilos comunicación con **MAVLinkServer**.

- El tercer y último subobjetivo fue la validación experimental de ambos componentes desarrollados. En esta fase se realizaron muchas de pruebas, desde experimentos para justificar el correcto funcionamiento del driver y el componente **UAV-Viewer.py** hasta pruebas de configuración o pruebas para estudio del comportamiento del sistema a bordo del cuadricóptero que muestran el correcto funcionamiento de la solución alcanzada.

En este proyecto, además de estos tres objetivos, también se definieron una serie de requisitos que la solución tenía que cumplir.

- La infraestructura a desarrollar para MAVLink versión 1.0 tenía que estar integrada en JdeRobot. El github oficial de MAVLink¹, ha servido como apoyo y sobre todo como guía para las primeras etapas del desarrollo. Todas las aplicaciones desarrolladas en el proyecto son compo-

¹<https://github.com/mavlink/mavlink>

nentes JdeRobot, compatibles con su versión 5.6. Están integradas en el repositorio oficial y han sido probadas por terceros.

- El software creado debía ser eficiente computacionalmente. Todos los componentes desarrollados en el proyecto se basan en un diseño multihilo que se aprovechan de las arquitecturas modernas multicore. Además, se ha estudiado el comportamiento, la carga y la duración de los ciclos de cada hilo, lo cual ha permitido ajustar la duración de cada ciclo controlando así el tiempo que un hilo permanece ocioso.

Este es un proyecto heterogéneo y en él se han utilizado tecnologías variadas, resumidas en el capítulo 3, como por ejemplo JdeRobot, MAVLink, Python o ICE, librerías gráficas como PyQt5, etc.

Este Trabajo Fin de Grado se ha incluido en una ponencia que ha sido aceptada en el congreso CivilDron 2018 [16], titulada "Programación de aplicaciones para drones con el entorno software JdeRobot".

6.2 Lecciones prácticas

A lo largo de todas estas pruebas se aprendieron bastantes lecciones prácticas. Sin entrar en las especificaciones técnicas, el desarrollo de aplicaciones con el 3DR Solo Dron y en general con cualquier robot u otro dispositivo hardware conlleva una serie de inconvenientes. Al tratarse de un VANT el primer requisito indispensable para realizar aplicaciones sobre él es disponer de un lugar lo suficientemente amplio como para que el dron pueda volar sin que colisione con ningún objeto. Sobre todo en el comienzo del desarrollo, es altamente probable cometer errores en el código que provoquen que el cuadricóptero se estrelle, lo que trae consecuencias negativas como la rotura de algún componente del cuadricóptero, choques con objetos que se encuentren en el mismo espacio o incluso daños físicos para las personas que se encuentren cerca. En las primeras etapas del proyecto se tuvieron que reponer distintos componentes, como por ejemplo las hélices, sin que esto supusiera un gran problema. Pese a las piezas reemplazadas, el 3DR Solo Drone es una plataforma muy robusta que soportó numerosos accidentes a lo largo de este trabajo sin sufrir daño significativo.

Es importante tener en cuenta la normativa de drones vigente en cada momento, recientemente ha sido actualizada (Diciembre 2017), y en ella se especifican, con mayor detalle respecto a las anteriores, el entorno y las situaciones climáticas en las que está permitido el uso de los drones.

El 3DR Solo Drone tiene una batería de litio de 5200mAh a 14.8V con una autonomía de vuelo de aproximadamente 30 minutos. La carga de la batería con el cargador oficial oscila entre 50 y 60 minutos. Si todo va bien, esto significa que disponemos de 30 minutos para realizar las pruebas necesarias con nuestro código. Si algo falla y no se tienen más baterías, supone que como mínimo

habrá que esperar 50 minutos para reanudar las pruebas. Por esta razón es muy recomendable tener varios juegos de baterías y registrar todo el proceso que se realiza en las pruebas, desde la grabación en vídeo del vuelo del dron hasta el registro de los datos de la aplicación que nos ayuden a identificar el problema con exactitud. A veces puede resultar frustrante la corta vida de las baterías: aparte de disponer de un periodo de tiempo corto para realizar las pruebas, con el uso continuado de la batería la calidad de ésta se deteriora provocando que la autonomía disminuya o que el manejo del cuadricóptero se vea afectado por la falta de potencia necesaria: con baja batería no se comporta igual que con batería completamente cargada.

6.3 Trabajos Futuros

El desarrollo de este trabajo ha abierto las puertas a distintas ideas que pueden estudiarse y realizarse en próximos proyectos:

Una primera línea para continuar este desarrollo sería extender el soporte a otros sensores, como el soporte para GPS, ya que aumentaría en gran medida el número de aplicaciones que se pueden realizar con **MAVLinkServer**.

En segundo lugar, dar soporte a cámara a bordo del 3DR Solo Drone, ya sea a través de la API de MAVLink usando una GoPro, o usando el interfaz **cameraserver** que existe actualmente en JdeRobot.

En tercer lugar, relacionado con el punto anterior, sería deseable incorporar todo el sistema en una Intel Computer Stick a bordo del dron y enlazar directamente este pequeño computador con la placa controladora usando el puerto serie para prescindir de las comunicaciones via radio y wifi con el mando físico.

Finalmente, la integración de este Trabajo de Fin de Grado junto a otros trabajos ya realizados con el ArDrone de Parrot, podría dar lugar a nuevas aplicaciones y funcionalidades con el 3DR Solo Drone: com el recorrido de rutas en 3D, autolocalización o el aterrizaje visual. La integración de un sistema de autolocalización en interiores (SLAM) es delicada, ya que la potencia del 3DR Solo Drone hace que sea peligroso volar en interiores con espacios reducidos. Para el caso de la implementación de aterrizaje visual se puede utilizar como referencia el Trabajo de Fin de Grado de Jorge Vela[6].

Bibliografía

- [1] Alberto Martín Florido. Navegación visual en un cuadricóptero para el seguimiento de objetos. *Proyecto Fin de Carrera, URJC*, 2014.
- [2] Arturo Vélez. Seguimiento de un objeto con textura desde un drone con cámara. *Trabajo de Fin de Grado, URJC*, 2017.
- [3] Manuel Zafra Villar. Seguimiento de rutas 3D por un drone con autolocalización visual con balizas. <https://grvc.us.es/robot2017/> *Proyecto Fin de Carrera, URJC*, 2016.
- [4] Daniel Yagüe. Cuadricóptero ar.drone en gazebo y jderobot. *Proyecto Fin de Carrera, URJC*, 2015.
- [5] Jorge Cano. Building of an uav: from the hardware to the driver and autonomous applications. *Trabajo de Fin de Grado, ETSIT-URJC*, 2016.
- [6] Jorge Vela. Búsqueda y aterrizaje sobre baliza con Ardrone. *Trabajo de Fin de Grado, ETSIT-URJC*, 2017.
- [7] Daniel Plaza Rey. Navegación Autónoma de drones y automatización de rutas aplicadas a la limpieza de edificios. *Trabajo de Fin de Grado, Universitat Politècnica de Catalunya*, 2017. <https://www.civildron.com/>
- [8] Página oficial de JdeRobot. http://jderobot.org/Main_Page
- [9] Página oficial de MAVLink. <https://mavlink.io/en/>
- [10] Página oficial de Python. <https://www.python.org/>
- [11] Página oficial de 3DR. <https://www.parrot.com/es/>
- [12] Página oficial de Pixhawk. <https://pixhawk.org/>
- [13] Página oficial de ICE. <https://zeroc.com/products/ice>
- [14] Página oficial de MAVProxy. <http://ardupilot.github.io/MAVProxy>

[15] Normativa Diciembre 2018 sobre drones en España.<https://solodronesbaratos.com/drones-espana/>

[16] Congreso sobre aplicaciones civiles con drones.<https://www.civildron.com/>