



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN

GRADO EN INGENIERÍA EN TELEMÁTICA

TRABAJO DE FIN DE GRADO

Driver y soporte para drones con protocolo MavLink

Autor: Diego Jiménez Bravo

Tutor: Jose Maria Cañas Plaza

Curso Académico 2016/2017

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 2017, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 20XX

Agradecimientos

Resumen

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Índice General

| | | |
|----------|--|-----------|
| 1 | Introducción | 1 |
| 1.1 | Historia | 1 |
| 1.2 | Tipos de drones | 2 |
| 1.3 | Aplicaciones | 4 |
| 1.4 | Normativa sobre Drones | 5 |
| 1.4.1 | Normas | 6 |
| 1.4.2 | Prohibiciones | 6 |
| 1.5 | Hardware de drones | 8 |
| 1.6 | Software de drones | 12 |
| 1.7 | Robótica aérea | 14 |
| 1.7.1 | Robótica aérea en RoboticsLab de la URJC | 15 |
| 2 | Objetivos | 20 |
| 2.1 | Problemas a abordar | 20 |
| 2.2 | Requisitos | 20 |
| 2.3 | Metodología y plan de trabajo | 21 |
| 3 | Infraestructura | 23 |
| 3.1 | 3DR Solo Dron | 23 |
| 3.1.1 | Placa estabilizadora Pixhawk | 24 |
| 3.2 | Protocolo MAVLink | 25 |
| 3.3 | JdeRobot | 26 |
| 3.4 | Interfaces relativos a los drones | 27 |
| 3.5 | ICE | 29 |
| 3.6 | Python | 29 |
| 3.7 | MavProxy | 30 |
| 4 | MavLinkServer | 33 |
| 4.1 | Diseño | 34 |

| | | |
|----------|------------------------------------|-----------|
| 4.2 | Implementación | 36 |
| 4.2.1 | Pre-Configuración | 36 |
| 4.2.2 | Script de arranque | 37 |
| 4.2.3 | MAVLinkServer | 37 |
| 4.3 | Operación | 50 |
| 5 | Visor UAV Viewer | 53 |
| 5.1 | Interfaz Gráfica | 54 |
| 5.2 | Hilos de comunicación | 56 |
| 5.3 | Fichero de configuración | 57 |
| 6 | Conclusiones | 59 |
| 6.1 | Conclusiones | 59 |
| 6.2 | Lecciones prácticas | 62 |
| 6.3 | Trabajos Futuros | 63 |
| | Bibliografía | 64 |

Indice de imágenes

| | | |
|------|--|----|
| 1.1 | Ryan Firebee | 1 |
| 1.2 | Gnat | 2 |
| 1.3 | Clasificación por alas | 3 |
| 1.4 | Clasificación por aplicación | 5 |
| 1.5 | Normas | 6 |
| 1.6 | Prohibiciones | 8 |
| 1.7 | Diferentes movimientos que puede realizar un dron. | 9 |
| 1.8 | Distintas partes de un dron. | 12 |
| 1.9 | TFG Alberto Martín. | 15 |
| 1.10 | TFG Arturo Vélez. | 16 |
| 1.11 | TFG Manuel Zafra. | 16 |
| 1.12 | TFG Daniel Yagüe. | 17 |
| 1.13 | TFG Jorge Cano | 18 |
| 1.14 | TFG Jorge Vela. | 18 |
| 3.1 | 3DR Solo Drone | 23 |
| 3.2 | ArduIMU | 24 |
| 3.3 | Diagrama comunicación | 25 |
| 3.4 | Comunicación MAVLink | 26 |
| 3.5 | Protocolo Mavlink | 32 |
| 4.1 | Diagrama comunicación | 33 |
| 4.2 | Diagrama bloques MAVLink-JdeRobot | 34 |
| 4.3 | Gestión de memoria | 40 |
| 4.4 | Diagrama bloques MAVLink-JdeRobot con detalle | 46 |
| 5.1 | Esquema UAV viewer | 54 |
| 5.2 | Interfaz | 54 |
| 5.3 | Mando 3dr Solo Dron | 55 |
| 5.4 | Hilos Uav Viewer | 56 |

| | | |
|-----|--------------------|----|
| 5.5 | Sensores | 57 |
|-----|--------------------|----|

Capítulo 1

Introducción

1.1 Historia

Un dron no es más que una aeronave no tripulada que se controla remotamente y que, por lo general, se puede decir que siempre ha sido el sueño de todo estratega militar, ya que permite alcanzar al enemigo a distancia así como evitar pérdidas humanas.

Este sueño comenzó en el 1916, cuando el profesor Archibald Low, un militar científico, diseñó un torpedo aéreo que se manejaba con un sencillo control usando señales de radio. La aeronave fue un fracaso, pero sentó las bases para futuros diseños.

En Vietnam también se usaron otros modelos de dron como el Ryan Firebee que introdujo una nueva idea, la cámara de fotos para esppiar al bando contrario.



Figura 1.1: Ryan Firebee

Pero si hay que hablar de una salto en el desarrollo de los drones entonces es necesario mencionar al Gnat que fue desarrollado por General Atomics, un contratista de defensa de San Diego, California, EE. UU. que introdujo las cámaras de video y con esto empezó una nueva era en el mundo de los dron.



Figura 1.2: Gnat

Para comenzar a hablar del uso de los dron por el público sin duda hay que mencionar a Nikola Tesla quien patentó por primera vez un vehículo no tripulado controlado remotamente al que llamó teleautomation y que hoy es uno de los principios que rige el diseño de un dron.

Sin duda ha sido un gran salto, por un lado muchos encuentran divertido el uso de los dron para fines personales y comerciales como la entrega de paquetes y otros proyectos de Amazon y Google mientras que otros critican fuertemente su uso en la guerra para bombardear a terroristas.

1.2 Tipos de drones

Como vehículo aéreo puede tener diferentes formas, bien tipo avión, tipo helicóptero o incluso formas muy diferentes. Pero los drones no son algo nuevo, 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.

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.

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. La clasificación 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: Suelen ser 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 necesario 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.



Figura 1.3: 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 si se necesita la figura de un técnico humano. La labor de esta persona es proporcionar información y controlar el feedback 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.
- Monitorizado: En este caso si se necesita la figura de un técnico humano. La labor de esta persona es proporcionar información y controlar el feedback 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 que acción llevará a cabo.
- 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 Aplicaciones

Se pueden aplicar en ambientes de alta toxicidad química y radiológicos en desastres tipo Chernóbil, en los que sea necesario tomar muestras con alto peligro de vidas humanas y realizar tareas de control de ambiente. Las aeronaves cumplen con las normas regulatorias establecidas en el Tratado de Cielos Abiertos de 1992 que permiten los vuelos de VANT sobre todo el espacio aéreo de sus signatarios. 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 1.4: Clasificación por aplicación

1.4 Normativa sobre Drones

Debido a la gran rapidez con la que se está desarrollando la industria de los drones en España, a día de hoy (Diciembre 2017) existe una legislación[13] de drones y unas normas, pero que todavía tienen muchas lagunas. La mayoría de las personas cuando compran un dron, realmente desconocen qué pueden hacer con él, dónde lo pueden volar, y en qué momento pueden llegar a violar la Ley de Drones España 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.

1.4.1 Normas

- 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.
- Solo 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

1.4.2 Prohibiciones

- 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, y 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 vuela 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 este punto, e incluso prevé multas que pueden alcanzar los 225.000€. Aunque esto intimida mucho, con usar el sentido común, y tener presente el que no haya personas cerca, es más que suficiente para disfrutar de nuestros drones sin tener ningún problema.

Lo que se debe tener claro es que 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, respectando los puntos señalados anteriormente, a menos que, por ejemplo, se realice una grabación y se esta 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

1.5 Hardware de drones

Hay que destacar las partes que tiene un dron y su forma, pues es gran parte lo que lo hace tan especial, permite que tenga una gran libertad de movimientos, ya que puede moverse sin problema desde cualquier punto hacia los ejes X, Y y Z. Lo que se gana con esto son funcionalidades como poder permitirse 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 pueden 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. Sin embargo, en los anteriores UAV solo tenemos el movimiento hacia adelante, teniendo que jugar con Roll, Yaw y Pitch para poder movernos en los distintos ejes.

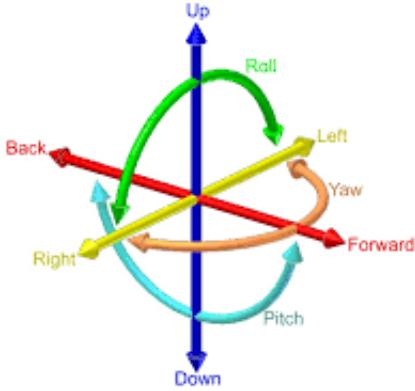


Figura 1.7: Diferentes movimientos que puede realizar un dron.

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 se trata de algún tipo de plástico, ya que es un material 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, también nombrar la fibra de vidrio. Este material también es muy utilizado por ser ligero, y tiene características como que no es conductor 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 de fuerzas, permitiendo así el movimiento del dron.
- Motores: Son los encargados de transformar la energía que llega en movimiento sobre el eje en el que se sitúan las hélices, para así permitirles a éstas hacer su trabajo. éste, a su vez, 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, ya que es la que 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. Puede variar 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 utilizado, 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 del dron, el que se encarga de recoger la información del dron y cuando le llega una orden ver que información tiene que mandar para que ésta 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 gente que tiene esto como un hobby, teniendo así un bajo costo y una alta disponibilidad. Es

un piloto automático en tiempo real y muy eficiente, proporcionando un entorno de estilo POSIX. éste es el autopiloto estándar de la industria, y por lo tanto, como veremos a continuación, 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.
- PixRacer: Se ha desarrollado para los drones de carreras, aunque también se utiliza en mini drones. Suele tener una mayor memoria flash.
- Navio2: Piloto automático diseñado de Raspberry Pi. Te permite convertir ésta en un controlador de dron.
- PXFmini: Es otro piloto automático de Raspberry Pi. éste tiene la electrónica para la mayoría de los componentes que puede utilizar un dron.
- FlytPOD: 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-Pilot: 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 un elemento importante como es la cámara, 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 parte 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 pueda realizar el 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 las imágenes sobre una pantalla LCD o utilizando unas gafas de realidad virtual. En esta línea 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 parezca incómodo tiene un joystick con el que también se pueden ordenar los movimientos realizar a la cámara.

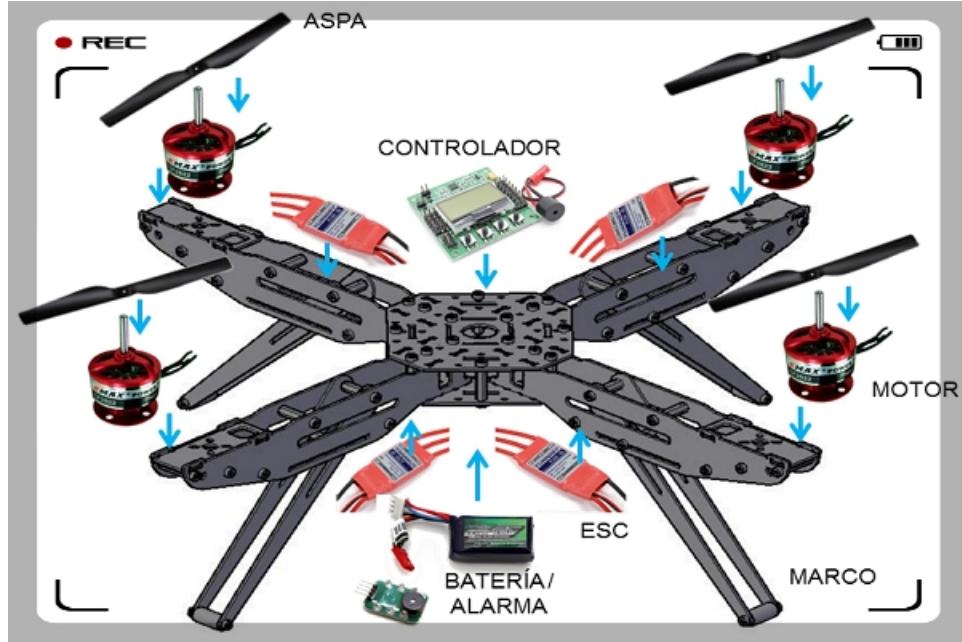


Figura 1.8: Distintas partes de un dron.

1.6 Software de drones

El software es el conjunto de programas que permiten realizar ciertas tareas, por ejemplo al dron una navegación autónoma. Dicho programa estará instalado en la placa controladora, y por tanto será el que 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 que se le comienzan a dar y no tanto al desarrollo militar. Existen varios entornos software que permiten el manejo y la programación de estos robots:

- ArduPilot3: Es un sistema OpenSource encargado de recibir la información que se le da y de enviar las señales correspondientes a los actuadores. 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 llegar a controlar, ya que trabaja con diversos dispositivos aéreos (aviones, helicópteros, drones, etc.) y con dispositivos marinos (como son los barcos y submarinos). Éste 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 dicho software es que sólo permite trabajar con plataformas de los mismos creadores. Otra característica es la facilidad con la que se le pueden añadir

diferentes sensores, como pueden ser módulos GPS o cámaras, algo que facilita la navegación autónoma.

- Megapirate-NG: Apareció como desarrollo del anterior. 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.
- MultiWii: 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 éste nos envía. Normalmente el fabricante del dron tiene ya un programa que realiza esta función. En este punto es importante el protocolo de comunicación que habrá para comunicar el vehículo con la estación terrena. Aquí hay un protocolo que destaca sobre los demás, el MAVLink (Micro Air Vehicle Communication Protocol). Este protocolo tiene la información contenida en ficheros .xml, lo que permite utilizarlo en diversos lenguajes de comunicación, y conlleva una mejora notable en su desarrollo. Al tener el fichero .xml los tipos de mensaje, permite con facilidad añadir nuevos tipos para asignar una tarea nueva. Otra ventaja es que hay muchos 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 como MAVProxy, Mission Planer o APM planner. Un problema en 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, este se utiliza en protocolos como TCP). MAVLink utiliza 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 ROS (Robot Operating System), sistema operativo de código abierto mantenido por la Open Source Robotics Foundation(OSRF). ROS tiene librerías y herramientas que permiten desarrollar software para robots. Es el software para robots más extendido en el mundo, lo que conlleva que sea el que más aplicaciones tiene para la robótica. También ofrece mucho software 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. Por otro lado, en protocolos de comunicación MAVLink lanzó también un software para la compatibilidad con ROS. A parte de todos éstos, existen también otras infraestructuras software como JdeRobot, que es la que hemos usado en este trabajo y que se describirá con más detalle en el capítulo 3.

1.7 Robótica aérea

La robótica aérea es la rama de la robótica que se encarga del estudio del comportamiento autónomo de los vehículos aéreos no tripulados, es la intersección entre las áreas descritas en las dos secciones anteriores. 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 VANT 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. En diciembre de 2013 Amazon sorprendió al mundo demostrando su interés en el envío de paquetes con drones. 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 realizase 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 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. En este capítulo viajaremos a lo largo del mundo recogiendo las aplicaciones de la robótica aérea. En él podremos encontrar desde aplicaciones comerciales, hasta los desarrollos punteros en algunos de los centros de investigación más prestigiosos del mundo. Daremos

una vuelta sobre las distintas competiciones de robótica aérea para acabar con los desarrollos que se están haciendo en nuestro país y en nuestra universidad. No pretende ser exhaustivo, sino visitar unos cuantos ejemplos representativos.

1.7.1 Robótica aérea en RoboticsLab de la URJC

En un contexto más cercano, este proyecto se inició tras otros realizados anteriormente en el laboratorio de robótica de la universidad. Los ejemplos más destacables son los siguientes:

- Alberto Martín^[1]¹ trabajó en el seguimiento de objetos con un dron utilizando su cámara. 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 desarrollo el driver para el ArDrone del Parrot real, que es el mismo que hemos utilizado en este proyecto.

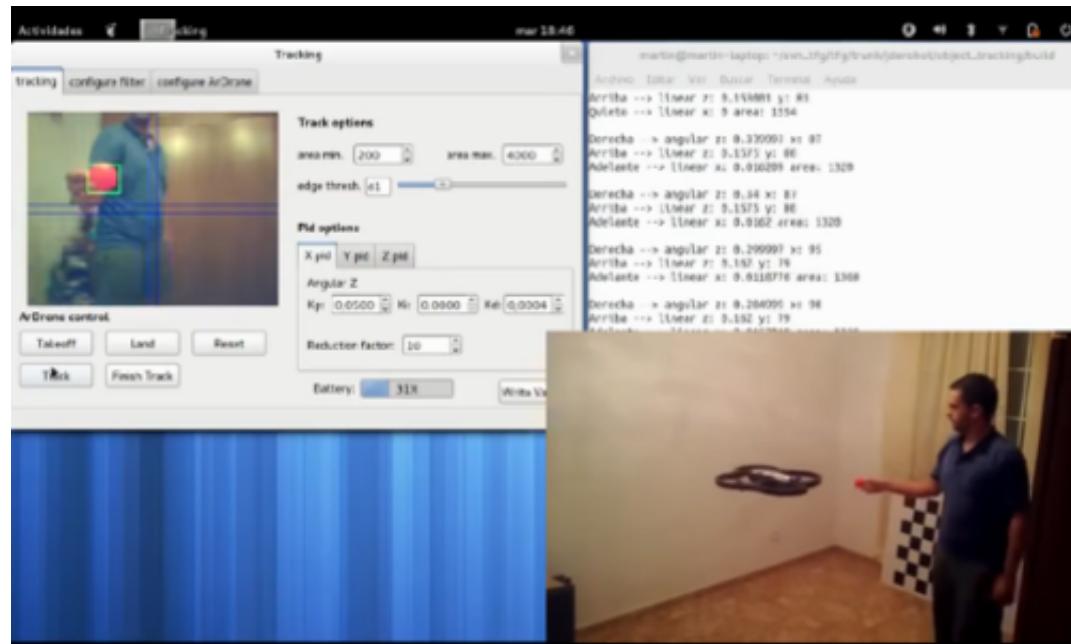


Figura 1.9: TFG Alberto Martín.

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

¹<http://jderobot.org/Amartinflorido-tfg>

²<http://jderobot.org/Avelez-tfg>

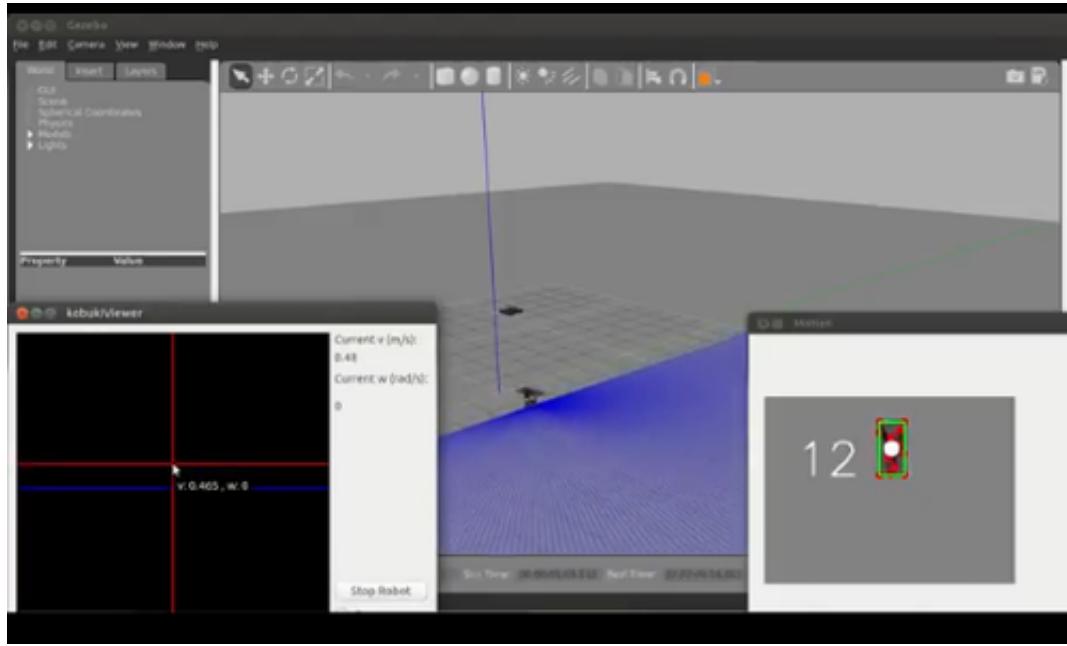


Figura 1.10: TFG Arturo Vélez.

- Manuel Zafra[3]³ trabajó en la localización del dron mediante April Tags. 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 April Tags que detectaba el dron podía detectar el punto en un mapa 3D en el que estaba situado.

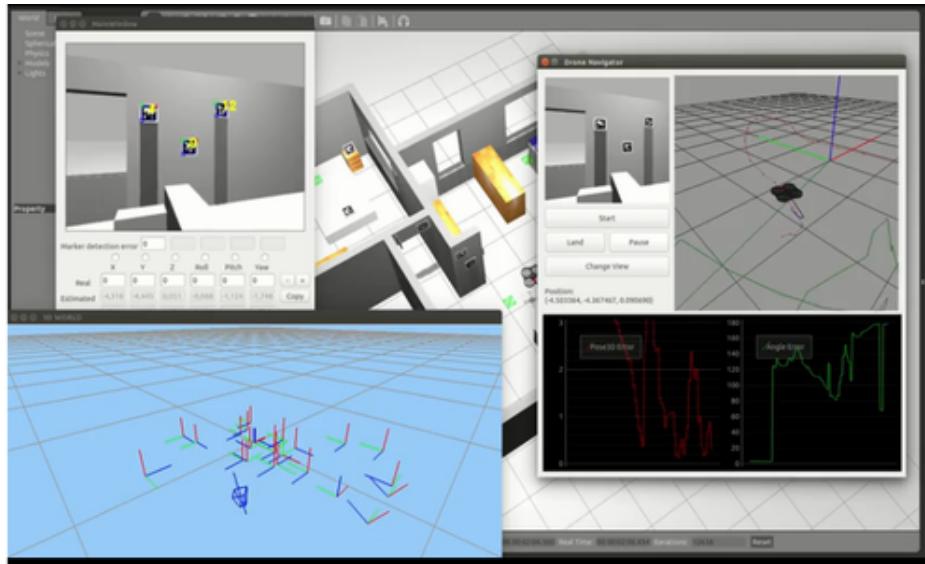


Figura 1.11: TFG Manuel Zafra.

³<http://jderobot.org/MazafraV-pfc>

- Daniel Yagüe[?] ⁴ trabajó con el ArDrone sobre Gazebo, probando distintas funcionalidades y escenarios. Por un lado trabajó en el driver para el cuadricóptero simulado, consiguiendo un plugin 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 vuelo. Por otro lado sobre simulador programó aplicaciones de control visual, haciendo que el dron siguiera una línea, una carretera o que un dron siguiera al otro.

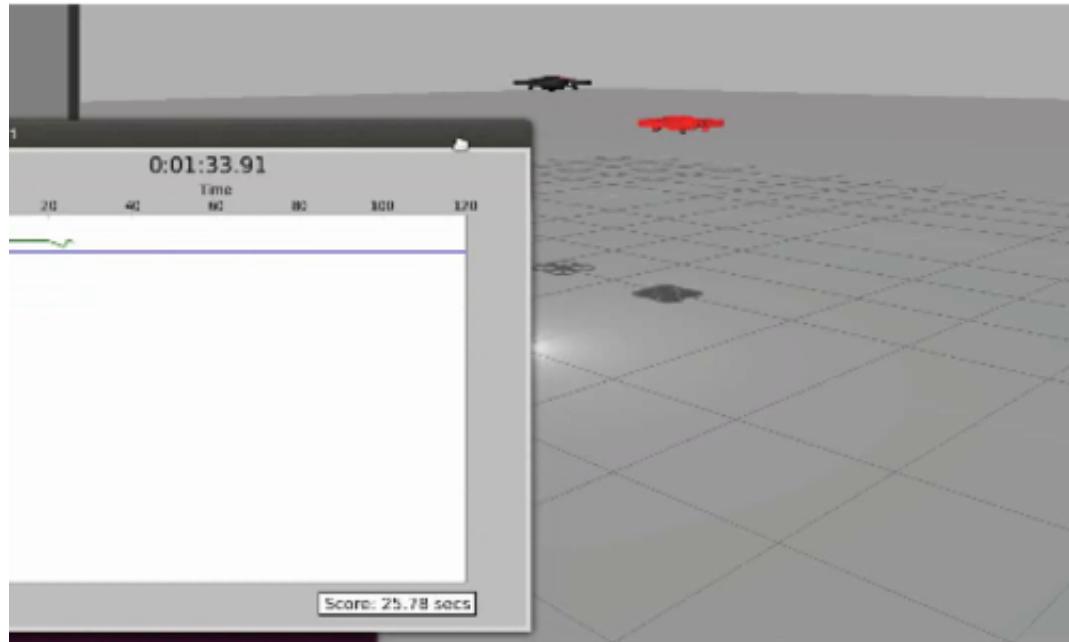


Figura 1.12: TFG Daniel Yagüe.

- Jorge Cano[5] ⁵ trabajó en el diseño, construcción y programación de un dron real. El driver que realizó para su control se basa en el protocolo MAVLink, en parte por su gran popularidad. Por una parte trabajó en el diseño y construcción de la plataforma hardware, como se muestra en las dos primeras imágenes de la figura 1.13, y por otra en el controlador software para la comunicación con el vehículo. Para hacer test de vuelo realizó pruebas perceptivas y de control, con filtros de colores y un controlador PID.

⁴<http://jderobot.org/Daniyague-pfc>

⁵<http://jderobot.org/J.canoma-tfg>



Figura 1.13: TFG Jorge Cano

- Jorge Vela^[6] ⁶ trabajó en el diseño de un algoritmo con la finalidad de que el dron navegue de forma autónoma, desde una baliza inicial hasta otra baliza final, de posición desconocida, y no necesite a ninguna persona diciéndole lo que tiene que hacer en cada momento o controlándole. Para conseguirlo se han tenido en cuenta los distintos estados en los que se puede encontrar el dron.

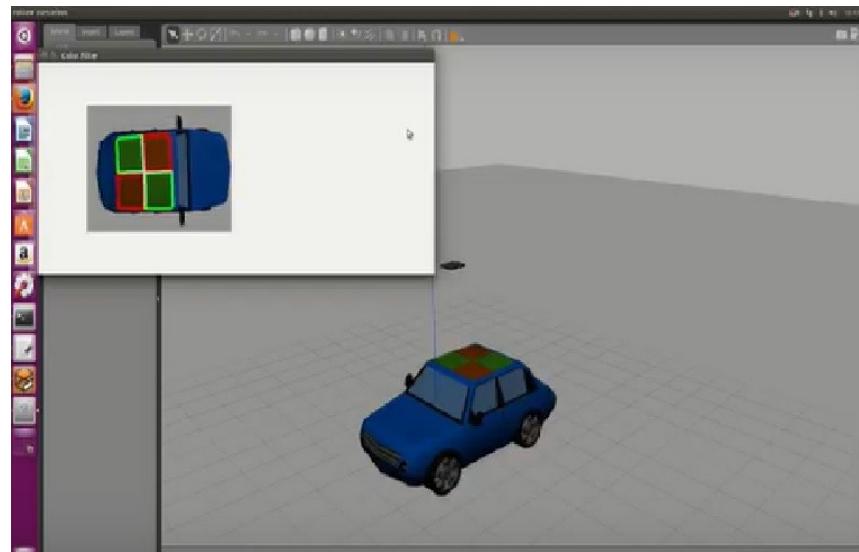


Figura 1.14: TFG Jorge Vela.

⁶<http://jderobot.org/Jvela-tfg>

Una vez terminada la breve introducción, se va a explicar lo realizado en este proyecto. 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 como 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 para dirigir 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

Los objetivos de este trabajo fin de grado es crear un driver en la plataforma software JdeRobot para drones que utilicen como interfaz de comunicación MAVLink. Para abordar el problema lo hemos dividido en:

1. Preparación del hardware necesario.
2. Desarrollar driver para pilotar drones en una interfaz de nivel medio. Se comunicara con el dron usando el protocolo MavLink y se usará el interfaz de nivel medio que facilita JdeRobot llamado CMDVel.
3. Desarrollar una herramienta que permita pilotar los drones con interfaz de nivel medio en python de manera más intuitiva. Esta herramienta es una evolución del interfaz ya existente del UavViewer que facilita JdeRobot.
4. Experimentos en dron real. Conectaremos nuestro 3DR Solo tanto al driver MavLinkServer, que nos facilitara la comunicacion y a la herramienta UavViewer que nos permitira pilotar.

2.2 Requisitos

Para abordar con éxito los puntos expuestos anteriormente debemos cubrir los siguientes requisitos:

1. Preparación del hardware necesario:
 - a Compra del hardware necesario:
 - 3DR Solo dron.
 - Webcam.
 - Intel Computer Stick.

- Batería externa.
- b Instalación de JdeRobot en Intel Computer Stick.
- c Pruebas de vuelo con todo el equipamiento a bordo del 3DR Solo y pilotaje externo.
2. MavLinkServer
- a Comunicación con el dron.
- b Administración de los modos de vuelo.
- c Fases de despegue y aterrizaje automáticas.
- d Envio de comandos de velocidad al dron.
3. UavViewerPy
- a Comunicación con MavLinkServer.
- b Ordenes de despegue y aterrizaje.
- c Envio de comandos de velocidad a MavLinkServer mediante 2 joysticks.
4. Experimentos en dron real.
- a Pruebas de interconexion entre todos los componentes.
- b Ejecución de plan de pruebas:
- Prueba de despegue y aterrizaje.
 - Prueba de vuelo controlado.

Cómo requisitos no funcionales debemos:

1. Ser multiplataforma.
2. Utilizar únicamente librerías de software libre.
3. Ser 100 % compatibles con los actuales interfaces JdeRobot.

2.3 Metodología y plan de trabajo

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 4 herramientas:

- GitHub como forja y control de versiones. En el repositorio <https://github.com/RoboticsURJC-students/2016-tfg-Diego-Jimenez> se almacenan todos los desarrollos que son objetivo de éste TFG así como ésta 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 JdeRobot dónde hemos actualizado periódicamente nuestros avances acompañados con explicaciones, vídeos e imágenes. <http://jderobot.org/Jimenez-tfg>
- Todos los vídeos del mediawiki han sido compartidos en Youtube.

Capítulo 3

Infraestructura

3.1 3DR Solo Dron

Para este desarrollo se ha elegido el dron 3DR[10] Solo distribuido por la empresa norteamericana <https://3dr.com/>. Este dron se encuentra en una gama alta debido a sus capacidades <https://3dr.com/solo-drone/specs/>, 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 software 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.



Figura 3.1: 3DR Solo Drone

La CPU que tiene este dron es la ArduIMU¹, es una tarjeta de Unidad de Medida Inercial que integra un procesador compatible con Arduino y que capaz de ejecutar Attitude Heading Reference System (AHRS), basado en el algoritmo DCM (Direct Cosine Matrix)de Bill Premerlani.

La tarjeta consta de un acelerómetro de 3 ejes y tres sensores giroscópicos, un regulador de tensión

¹https://3dr.com/support/articles/arduimu_v3_kit/

dual (3.3V y 5V), un puerto de GPS, un Atmega328 @ 16MHz (como el Arduino Duemilanova) y 3 LEDs de estado.

Es importante aclarar que la ArduIMU3.2 no es ninguna placa de navegación o piloto automático, sólo una placa de orientación y se puede utilizar en cualquier cosa en la que deseemos conocer su orientación con respecto al suelo – barcos, coches, aviones, ...

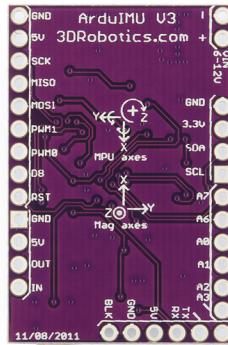


Figura 3.2: ArduIMU

Cuenta con una MPU-6000 que integra un giroscopio y acelerómetro de 3 ejes y se comunica mediante el bus SPI, un magnetómetro HMC-5883L conectado mediante I2C y un Arduino Atmega328 de 16Mhz.

3.1.1 Placa estabilizadora Pixhawk

Esta placa se trata de un desarrollo específico creado entre la "Pixhawk[11] open hardware community" en colaboración con 3D Robotics y que vio como primer destinatario el 3DR Solo. Ofrece un interfaz que se apoya en comandos llamado MAVLink. A través de estos comandos se le puede también enviar órdenes al piloto automático quien las ejecutará más adelante trataremos el protocolo MAVLink en profundidad. El único modo de conectar con el 3DR Solo será a través del mando, debido a que únicamente el mando es capaz de levantar la dirección IP a la que poder conectarse a ella. En posteriores evoluciones de la versión que controlan tanto el dron como el mando, desde los foros oficiales de 3DR², comentan que ya se aborda la solución de que sea el propio dron quien levante la dirección IP a la cual poder conectar y no tener que depender del enlace del mando.

La comunicación entre el driver y el dron quedaría de la siguiente forma:

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



Figura 3.3: Diagrama comunicación

3.2 Protocolo MAVLink

MAVLink[8] siglas de Micro Air Vehicle Link es un protocolo de comunicación desarrollado para comunicar las placas estabilizadoras con piloto automático a los GCS o estación de tierra, las aplicaciones desde las que se podía enviar misiones y seguir el cumplimiento de las mismas desde tierra. MAVLink se publicó³ en 2009 por Lorenz Meier, publicado bajo licencia LGPL aspira a convertirse en el protocolo standard en robótica aérea y se ha probado su funcionamiento en PX4, PIXHAWK, APM⁴ y Parrot AR.Drone.

La versión actual que se está utilizando en este TFG es la 2.0.

La lista completa de los comandos se encuentra a en la pagina oficial de Mavlink⁵. Un ejemplo de los mensajes más importantes del protocolo y en el que se centra principalmente este TFG es el comando actuador 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 (velocidad medida en m/s). Este comando, en conjunto al actuador que hace uso de la rotación, cuya estructura es la del mensaje "COMMAND_LONG", permite tener control total sobre el dron.

Cada comando tiene un identificador único el cual permite al dron reconocer la acción que se debe realizar. En función de este primer identificador los parámetros que se introducen a continuación permiten dar la información necesaria al dron para actuar. Esta comunicación y el contenido de los mensajes se comentará en mayor detalle en el apartado 4.1. Un ejemplo de la estructura del mensaje que se usa para GPS es el siguiente:

³<https://github.com/mavlink/mavlink/commit/a087528b8146ddad17e9f39c1dd0c1353e5991d5>

⁴Ardupilot Mega

⁵<http://mavlink.org/messages/common>

```

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 que decidimos en parámetros de conexión con el dispositivo.

En la figura 3.4 se puede comprobar que en la estructura de comunicación que se realiza entre la estación de tierra y cada componente se establece una conexión, en la cual 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, este último mensaje también es usado en este TFG para comprobar que la conexión con el dron sigue activa desde cualquier componente.

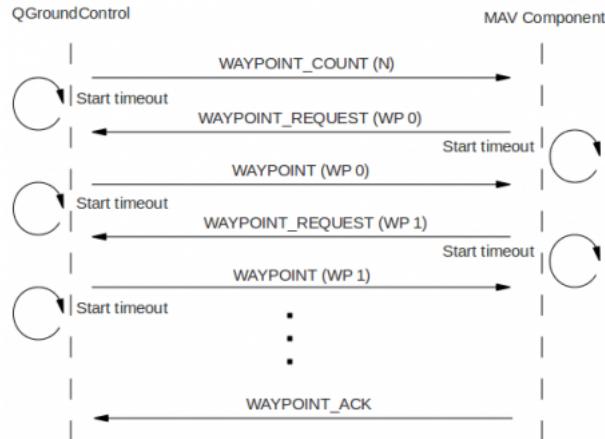


Figura 3.4: Comunicación MAVLink

3.3 JdeRobot

JdeRobot[7] es un entorno desarrollado por el laboratorio de robótica de la Universidad Rey Juan Carlos, para el desarrollo de aplicaciones de robótica. Su última realease la 5.6 se liberó el 9 de

Octubre de 2017 pudiendo ver los detalles de ésta en el github oficial⁶. JdeRobot se compone de interfaces, drivers, utilidades y aplicaciones para el desarrollo de cualquier proyecto de robótica, se apoya en estos interfaces, algunos de ellos los veremos en profundidad a continuación, para interconectar entre sí todos los aplicativos del mismo y en Zeroc ICE para la comunicación entre ellos. Algunos de los drivers más importantes que contiene:

1. Camer(server. Se trata de un driver para enviar imágenes y video a través del interfaz camera
2. Gazebo(server. Driver desarrollado para conectar los robots en el simulador Gazebo con aplicaciones y así poder simular los desarrollos ⁷.
3. Ardrone_server. Driver que conecta el Parrot Ar-Drone a JdeRobot⁸. Este driver 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.

Algunas de las herramientas web desarrolladas más importantes serían:

1. Cameraview. Se trata de una aplicación desarrollada en c++ capaz de recibir vídeo a través del interfaz camera.
2. UAV viewer. Aplicación desarrollada como ground control de robots aéreos. Esta aplicación permite teleoperar cualquier tipo de robot aéreo y 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. ⁹.

En este TFG se ha abordado la creacion de un nuevo driver y la versión mejorada de una herramienta de visualización y manejo de drones.

3.4 Interfaces relativos a los drones

JdeRobot dispone de más de 30 interfaces pero en este capítulo se explica lo que se han utilizado durante el desarrollo:

- Pose3D. Utilizado para recoger los datos de actitud y la posición de la aeronave.

```
Pose3DData
```

```
{
```

⁶<https://github.com/JdeRobot/JdeRobot/wiki/JdeRobot-5.6.0>

⁷<http://jderobot.org/Daniyague-pfc>

⁸<http://jderobot.org/Amartinflorido-tfg>

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

```

    float x; /* x coord */
    float y; /* y coord */
    float z; /* z coord */
    float h; /* */
    float q0; /* qw */
    float q1; /* qx */
    float q2; /* qy */
    float q3; /* qz */
};


```

- CMDVel. Utilizado para enviar comandos de velocidad.

```

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.


```

3.5 ICE

ICE[12] (Internet Communications Engine) es un *middleware* orientado a objetos que proporciona llamadas a procedimientos remotos, *grid computing* y funcionalidad cliente / servidor desarrollada por ZeroC bajo GNU GPL y una licencia privativa. 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. ICE permite desarrollar aplicaciones distribuidas con un esfuerzo mínimo, abstraer al programador para que interactúe con una red baja interfaces de trabajo. El proceso de desarrollo de aplicaciones debe enfocarse solo en la lógica y no en las peculiaridades de la red. Es un *middleware* multilenguaje y así, podemos implementar clientes y servidores en diferentes lenguajes de programación y en diferentes plataformas. ICE trabaja con objetos distribuidos, de modo que dos objetos en nuestra aplicación no necesitan ejecutarse en la misma máquina. Los objetos pueden estar en diferentes máquinas y comunicarse a través de la red a través del envío de mensajes entre ellos.

JdeRobot utiliza 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 controlador en C++ para un robot particular que está incrustado en el robot, por otro lado, otro desarrollador puede desarrollar una aplicación para el procesamiento de imágenes en Python que se ejecuta en una PC. Mediante ICE podemos usar estas dos aplicaciones, que originalmente eran independientes, como un solo aplicación sin tener que preocuparse por las comunicaciones de bajo nivel. Con esto podemos desarrollar aplicaciones modulares de gran complejidad sin esfuerzo adicional.

La conexión entre el driver y las aplicaciones de control, como la propia herramienta de teleoperación, se realiza usando este *middleware*.

3.6 Python

Python[9] es un lenguaje de programación interpretado y multiplataforma que nació en los años 80 en los países bajos con idea de hacer más legible el código. El lenguaje de programación que inicialmente se utilizaba principalmente 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.

El porqué de utilizar Python, muy sencillo mantiene el carácter multiplataforma de JdeRobot, su código es simple y legible y trabaja muy bien con dependencias muy utilizadas en robótica como

OpenCV.

Tanto el driver como la herramienta desarrolladas en este TFG se han creado empleando Python 2.7. Esta decisión se debe a que JdeRobot es compatible con ROS y éste *framework* únicamente es compatible en dicha versión de Python. Actualmente se encuentra preparada una versión en el repositorio migrado a la versión de Python 3.5.

3.7 MavProxy

MavProxy es un módulo multithread (maneja varios hilos de ejecución simultáneamente). El flujo de la información y la operación del controlador seguir la siguiente ruta de la tarea:

- El controlador comienza a ejecutar todos los hilos diferentes, abriendo su comunicación canales y definiendo la información que estaría en cada uno. Hace uso de dos canales de comunicación basados en Pose3D, uno para publicar la posición del vehículo y actitud y otra para recibir órdenes de puntos de referencia; y un CMDVel canal para los comandos de aterrizaje.
- Los mensajes de MAVLink entran en el controlador y se interpretan para obtener la información de posición y actitud del vehículo. Adquirida esta información se trata para transformarla en estándares JdeRobot (Pose3D). Latitud y la longitud se transforman en coordenadas xyz globales, utilizando WGS84 como la Tierra modelo, y la actitud de ángulos de Euler se transforman en cuaterniones.
- Pose3D está escrito en las clases locales correspondientes de forma controlada, haciendo uso de lock (librería que nos permite excluir ciertas variables para su uso).
- Las clases se publican en los canales de ICE correspondientes a medida que se ejecutan los hilos, para permitir que otras aplicaciones de JdeRobot puedan acceder a ellos.
- Los comandos de aterrizaje se reciben a través de Extra y comandos de velocidad a través de CmdVel. La información se extrae de las clases correspondientes.
- Los comandos se traducen a mensajes MAVLink y se envían a la placa Pixhawk.

Se puede ver que cada subproceso tiene una tarea pero no tienen la misma carga de trabajo. Por esta razón, el tiempo del ciclo de control de cada hilo es diferente. A pesar de la los hilos tienen diferentes ritmos, el módulo funciona con éxito en todas sus diferentes tareas.

MAVLink admite tipos de datos enteros de tamaño fijo, números de punto flotante de precisión simple IEEE 754, matrices de estos tipos de datos (por ejemplo, char [10]) y el campo especial de conversión de MAVLink, que se agrega automáticamente mediante el protocolo. Estos tipos están disponibles:

- char
- uint8
- int8
- uint16
- int16
- uint32
- int32
- uint64
- int64

Estos paquetes que se envían tienen la estructura de la imagen 3.5, y en la tabla 3.7 se hace referencia a cada uno de los campos con una breve explicación de cada uno de ellos:

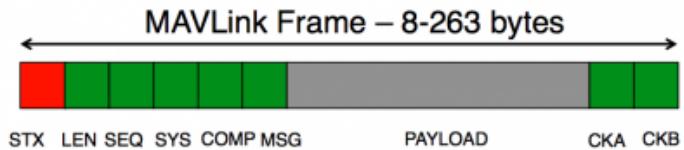


Figura 3.5: Protocolo Mavlink

| Índice del byte | Contenido | Valor | Explicación |
|-----------------|-----------------------|-----------------|--|
| 0 | Señal inicio paquete | 0xFE | Indica el inicio de un nuevo paquete. |
| 1 | Longitud de la carga | 0 - 255 | Indica la longitud de los datos que lleva. |
| 2 | Secuencia del paquete | 0 - 255 | Cada componente cuenta con su propia secuencia. Permite detectar paquetes perdidos. |
| 3 | ID del sistema | 1 - 255 | ID del sistema emisor. Permite diferenciar MAVs en la misma red. |
| 4 | ID del componente | 0 - 255 | ID del componente emisor. Permite diferenciar componentes en el mismo sistema, por ejemplo la IMU y la cámara. |
| 5 | ID del mensaje | 0 - 255 | Permite identificar los datos del paquete para su correcta decodificación. |
| 6 to (n+6) | Datos | (0 - 255) bytes | Datos del mensaje, depende del ID del mensaje. |
| (n+7) to (n+8) | Checksum | | El checksum incluye MAVLINK-CRC-EXTRA (protege el paquete de una decodificación errónea). |

Capítulo 4

MavLinkServer

Este capítulo describe el componente de software desarrollado para interactuar de forma autónoma con el dron de las aplicaciones. 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 que están hablando con este controlador.

De acuerdo con la figura 4.1, la comunicación de nuestro servidor que se comunica vía MAVLink con el dron es bidireccional. Dependiendo del modulo que se este usando, esta comunicación puede ser dron-servidor, como por ejemplo suministrar imágenes de una cámara, servidor-dron, para ordenar comandos de velocidad, o bidireccional, para realizar la conexión con el servidor y envío de ACKs.

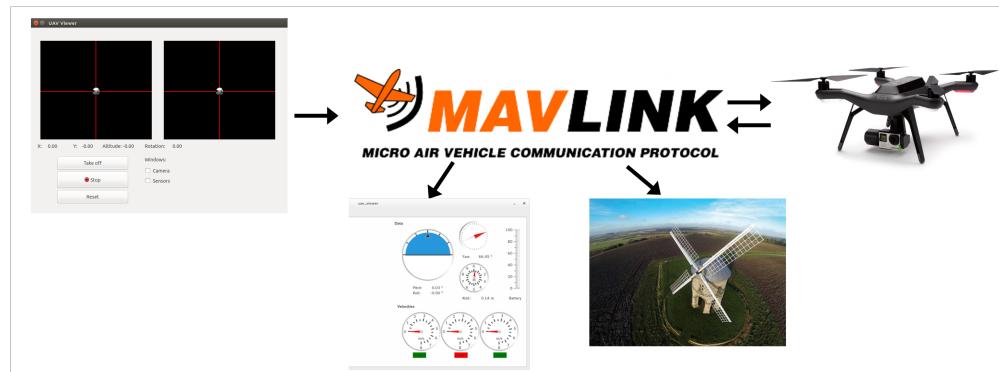


Figura 4.1: Diagrama comunicación

A continuación vamos a dividir en distintas fases el contenido de este driver y su ejecución:

- Diseño

- Implementación
- Operación

4.1 Diseño

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

MAVLinkServer se basa en el analizador MAVProxy, la parte del programa que está a cargo de la administración de los mensajes de MAVLink. Establece la conexión con el piloto automático Pixhawk, Mantiene el canal de comunicación operativo, adquiere, interpreta mensajes, crea y envía mensajes nuevos con la información solicitada u ordenada por la aplicación.

El código desarrollado está principalmente a cargo de la gestión de las interfaces 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 módulo sea utilizable para aplicaciones JdeRobot.

Estos dos lados del módulo proporcionan un controlador fiable y multi-compatible con las características de ICE; MAVLinkServer puede conectarse con aplicaciones escritas en otros diferentes idiomas, como C++, Python o Java, a través de las interfaces de JdeRobot. La figura 4.2 representa un esquema de los bloques dentro de MAVLinkServer y sus conexiones a otras aplicaciones.

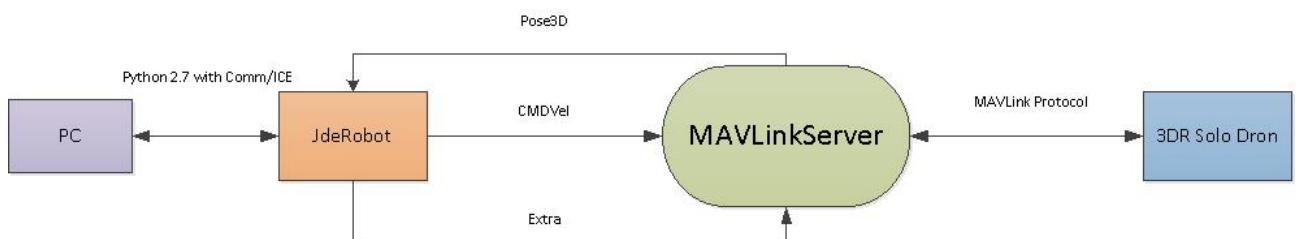


Figura 4.2: Diagrama bloques MAVLink-JdeRobot

MAVProxy es un sistema de comunicación grupal totalmente funcional para drones. Es un sistema 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 la red a través de UDP o TCP a otros programas de estación terrestre en otros dispositivos.

- 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.

Las interfaces JdeRobot utilizadas en este proyecto son Pose3D (proporciona la posición y medidas del vehículo), CMDVel (para enviar los comandos de velocidad) y Extra (para despegue y aterrizaje). Tenga en cuenta que Pose3D hace uso de cuaterniones en lugar de ángulos de Euler. Esto se hace para evitar singularidades angulares y superposiciones, y computacionalmente son más simples que otros formalismos de posicion como ángulos de Euler o una matriz de coseno de dirección.

Pose3D:

```
class Pose3DDData
{
    float x;
    float y;
    float z;
    float h;
    float q0;
    float q1;
    float q2;
    float q3;
};
```

CMDVel:

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

Extra:

```
interface ArDroneExtra{  
    void toggleCam();  
    void land();  
    void takeoff();  
    void reset();  
    void recordOnUsb(bool record);  
    void ledAnimation(int type, float duration, float req);  
    void flightAnimation(int type, float duration);  
    void flatTrim();  
};
```

4.2 Implementación

4.2.1 Pre-Configuración

En este apartado nos vamos a centrar en explicar todo lo relacionado con los pasos a seguir para realizar una configuración, que nos puede ofrecer dependiendo de este tipo de configuración y todas las librerías que tenemos a nuestra disposición.

Para llevar a cabo esta configuración del servidor debemos tener en cuenta la existencia de una serie de ficheros, llamados módulos (en lenguajes como java, suelen conocerse con el nombre de librerías), que cada uno de ellos nos va a facilitar una utilidad diferente, que pueden ir desde conocer la batería, altura del dron o configurar una web-cam.

Estos módulos se pueden descargar desde el repositorio oficial de MAVLink o crear una librería propia e implementarla. En el siguiente código que se muestra se puede ver como esta implementa obtener la información de la batería a través del voltaje que desprende:¹.

```
def vcell_to_battery_percent(self, vcell):  
    '''convert a cell voltage to an approximate  
    percentage battery level for a LiPO'''  
    if vcell > 4.1:  
        # above 4.1 is 100% battery  
        return 100.0  
    elif vcell > 3.81:  
        # 3.81 is 17% remaining, from flight logs  
        return 17.0 + 83.0 * (vcell - 3.81) / (4.1 - 3.81)
```

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

```

elif vcell > 3.2:
    # below 3.2 it degrades fast. It's dead at 3.2
    return 0.0 + 17.0 * (vcell - 3.20) / (3.81 - 3.20)
    # it's dead or disconnected
return 0.0

```

El fichero de configuración que usa el servidor únicamente se limita a establecer los puertos mediante los cuales se va a realizar la comunicación de cara a un usuario. Esta comunicación la realiza mediante las interfaces ICE que permite la comunicación con el servidor, pudiendo así recibir datos de sensores y motores, y enviar las instrucciones de velocidad necesarias en cada momento.

4.2.2 Script de arranque

El script de arranque se encargara de ejecutar todos los comandos previos y el servidor. El script se encuentra en MAVProxy/MAVProxyWinLAN.sh, deberemos averiguar la IP que levanta el dron, en nuestro caso, con el 3DR Solo, dicha IP la levanta el mando como hemos comentado en la introducción de este capítulo. Deberemos modificar el script con la IP del dron a la que nos hayamos conectado y ejecutarlo sin parámetros adicionales. El fichero README del repositorio contiene una descripción mas detallada de un ejemplo de ejecución.

Se necesita tener preinstalado tanto pyserial como una versión de pyvmavlink superior a la 1.1.50. Se realiza una descarga de todos los módulos, construye un directorio llamado MavProxy.egg en /home/USER/.local/python3.5/site-packages con el fin de tener almacenados todos los paquetes necesarios y con permisos suficientes.

A través de estos paquetes generados es posible acceder a los comandos que nos 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 se realizará en la sección 4.3 bajo la tabla 4.3.

4.2.3 MAVLinkServer

El primer paso es definir las interfaces correspondientes (explicadas en la sección 5.1). MAVLinkServer suministra todas las interfaces JdeRobot para el acceso de drones desde cualquier componente externo. Aquí está un 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 setPose3DData(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 getPose3DData(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. Eso se puede notar el uso de ”bloqueos” de programación para proteger los datos almacenados en la clase. 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 podía leer mientras otra tarea estaba escribiendo en ella y viceversa, asegurando la administración correcta de la información en exclusión mutua.

MAVLinkServer lanza varios hilos para canales de comunicación ICE, uno para cada tipo de información. A pesar de que solo Pose3D, CMDVel y Extra deben ser realmente utilizados, el driver ofrece las interfaces restantes para compatibilidad y usos futuros.

Cada subprocesso hace uso de su propia función donde se realiza la configuración de ICE. Allí se realiza la publicación de ICE y es importante garantizar qué datos se envían, para que otras aplicaciones obtengan la información correctamente y garanticen la compatibilidad. El mismo procedimiento se realiza para todas las interfaces.

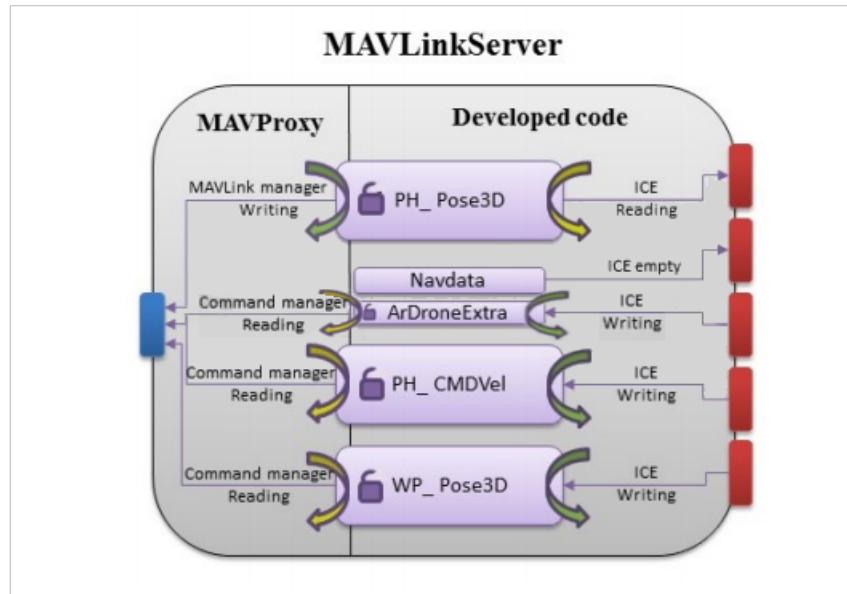


Figura 4.3: Gestión de memoria

La interfaz CMDVel y Extra tiene la misma estructura que Pose3D, también con el uso de bloqueos.

```

import jderobot, time, threading

lock = threading.Lock()

class CMDVel(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__(self):
#    #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

```

Por otro lado solo tiene una conexión con el puerto de comunicación con el dron (puerto serie). La gestión de estos 3 hilos de comunicación, respecto al único puerto serie que se mantiene por parte del dron, se basa en la gestión de prioridades. Extra es el más prioritario al ser el componente más restrictivo, este componente predomina sobre los otros 2 debido a la importancia de las órdenes que envía. Pose3D recibe la información del dron para actualizar la posición en un canal únicamente de lectura del puerto serie. Por otro lado conviven al mismo tiempo tanto CMDVel como Pose3D a la hora de realizar enviar comandos de velocidad y/o posición.

```

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

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

#Open an ICE TX communication and leave it open in a parallel threat

```

```

PoseTheading = threading.Thread(target=openPose3DChannel ,
                                args=(PH_Pose3D,) , name='Pose_Theading')
PoseTheading.daemon = True
PoseTheading.start()

# Open an ICE RX communication and leave it open in a parallel threat

CMDVelTheading = threading.Thread(target=openCMDVelChannel ,
                                    args=(PH_CMDVel,) , name='CMDVel_Theading')
CMDVelTheading.daemon = True
CMDVelTheading.start()

# Open an ICE TX communication and leave it open in a parallel threat

CMDVelTheading = threading.Thread(target=openExtraChannel ,
                                    args=(PH_Extra,) , name='Extra_Theading')
CMDVelTheading.daemon = True
CMDVelTheading.start()

# Open an ICE channel empty

CMDVelTheading = threading.Thread(target=openNavdataChannel ,
                                    args=() , name='Navdata_Theading')
CMDVelTheading.daemon = True
CMDVelTheading.start()

# Open an MAVLink TX communication and leave it open in a parallel
# threat
#
PoseTheading = threading.Thread(target=sendCMDVel2Vehicle ,
                                args=(PH_CMDVel, PH_Pose3D,) ,
                                name='TxCMDVel_Theading')
PoseTheading.daemon = True
PoseTheading.start()

```

```

# Open an ICE TX communication and leave it open in a parallel threat

PoseTheading = threading.Thread(target=openPose3DChannelWP ,
                                 args=(WP_Pose3D,) ,
                                 name='WayPoint_Theading')
PoseTheading.daemon = True
PoseTheading.start()

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

PoseTheading = threading.Thread(target=sendWayPoint2Vehicle ,
                                 args=(WP_Pose3D,) ,
                                 name='WayPoint2Vehicle_Theading')
PoseTheading.daemon = True
PoseTheading.start()

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

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

```

MAVproxy está constantemente manejando los mensajes MAVLink en un hilo paralelo en un bucle infinito. Este módulo lo aprovecha y refresca la información del sensor necesario. Como un controlador de alto nivel, este programa no interfiere con la fusión de datos realizado por Pixhawk y confía en su rendimiento, cuya fiabilidad ha sido ampliamente probado.

En el arranque del servidor se cargan los paquetes comentados en 4.3 y a partir de los cuales se podrán proporcionar funcionalidades, un ejemplo es el modo consola, el cual nos permite controlar al dron a partir de comandos definidos en la lista 4.3, que se comentará más en detalle en la sección 4.3.

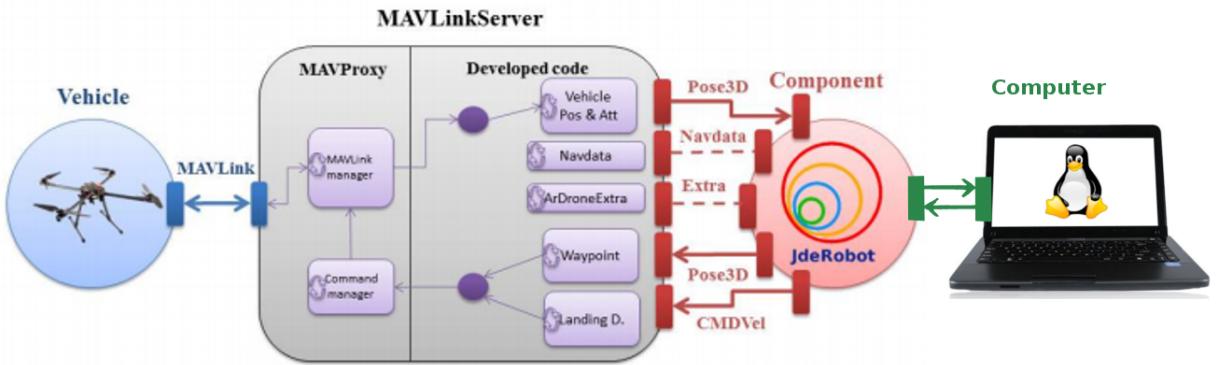


Figura 4.4: Diagrama bloques MAVLink-JdeRobot con detalle

Para llevar a cabo esta carga el primer paquete 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 4.2.2. Tras lograr la conexión se establece un periodo de "checkeo" 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 detenerse el dron. Esto se debe a 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 comunicara que aun esta conectado. Después de un tiempo sin conexión, aproximadamente unos 10-15 segundos, el dron procederá a realizar un despegue.

```
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)

```

Al acabar toda la carga de los módulos establecemos la conexión con los puertos necesarios con JdeRobot para poder pilotar el dron. Los módulos que usamos son Pose3D, CMDVel y Extra. Para cada uno de ellos vamos a necesitar crear 2 threads, uno para establecer la conexión con un sistema externo, en nuestro caso será el visor UavViewer, y otro para recibir los comandos que nos deseen enviar por el canal. Cada módulo lo necesitaremos por los siguientes motivos:

```

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")
return False

```

- CMDVel: 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 UavViewer en cada dirección viene dado en una escala de 0 a 1.

```

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)

```

- Extra: Este módulo nos dará la facilidad de despegar y de aterrizar. Debido al protocolo MAVLink, el sistema de despegue se compone en 3 fases, que con nuestro UavViewer se agrupan todas ellas. 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.
- Pose3d: Este módulo nos indicara la posición del dron en todo momento, así como su ori-

entación y altitud. Esta conexión nos va a servir tanto a la hora de aterrizar y despegar usando el otro módulo comentado Extra, para saber si puede aterrizar en un determinado momento o debe disminuir su altura antes de parar los motores.

4.3 Operación

En esta sección se va a explicar la posible configuración que permite el MAVLinkServer. Esta configuración se puede realizar en 2 niveles, a nivel del script de arranque o a partir de comandos una vez que el servidor este arrancado.

La configuración que se puede modificar a nivel de script esta 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 4.3 son un ejemplo de la versatilidad que tiene el servidor:

- master: Puerto maestro MAVLink y baudrate opcional. Por defecto=[].
- udp: Arranca el servidor udp. Por defecto se elige TCP.
- tcp: Arranca el servidor tcp. Por defecto se elige TCP.
- out: Puerto de salida MAVLink. Por defecto=[].
- baudrate: Por defecto=57600.
- sitl(Software in the loop): Puerto de salida, esta opción únicamente es necesaria en caso de no tener disponible un dron.
- streamratedest: MAVLink stream rate. Por defecto=4.
- source-system: Código fuente MAVLink. Por defecto=255.
- source-component: Componente origen MAVLink. Por defecto=0.
- target-system: Sistema destino MAVLink. Por defecto=0.
- target-component: Componente destino MAVLink. Por defecto=0.
- logfile: Fichero de logs. Por defect=mav.tlog
- append-log (También aceptado ”-a”): Añadir al fichero de log ya existente. Por defecto=False.
- continue (También aceptado con ”-c”): Continua el log. Por defecto=False.
- quadcopter: Usar acciones de control para cuadricopteros. Por defecto=False.
- setup: Arrancar en modo setup. Por defecto=False.
- nodtr: Deshabilitar DTR(Data Terminal Ready). Por defecto=False.
- show-errors: Mostrar errores MAVLink. Por defecto=False.

- speech: Usar texto para hablar. Por defecto=False.
- aircraft: Establecer nombre para el dron (Visual en mensajes). Por defecto=None.
- cmd: Comandos a ejecutar tras el arranque. Por defecto=None.
- console: Usar consola GUI para introducir comandos.
- map: Carga un mapa de la zona.
- load-module: Carga un modulo específico, puede ser utilizado tantas veces como sea necesario separando con ",". Por defecto=[].
- mav09: Usa protocolo MAVLink 0.9. Por defecto=False.
- auto-protocol: Auto detecta versión de protocolo MAVLink. Por defecto=False.
- nowait: No realiza comunicación continua con el dron (HearthBeat). Por defecto=False.
- dialect: Dialecto MAVLink. Por defecto=ardupilotmega.
- rtscts: Habilita control de comunicación vía RTS/CTS.
- moddebug type=int, help="module debug level default=0
- mission: Nombre de la misión. Por defecto=None.
- daemon: Arranca en modo daemon, no muestra shell interactiva.
- profile: Arranca el Yappi python profiler.
- state-basedir: Directorio base para logs. Por defecto=None.
- version: Muestra información sobre la versión.
- default-modules: Módulos por defecto al iniciar. Por defecto=log, wp, rally, fence, param, relay, tuneopt, arm, mode, calibration, rc, auxopt, misc, cmdlong, battery, terrain, output.

Los comandos que se muestran en la tabla 4.3, solo se pueden ejecutar si en el paso anterior se ha introducido el comando ”-console”, de cualquier otra manera, no se muestra la consola mediante la cual se pueden introducir comandos. Estos comandos tienen una labor de modificar la experiencia de vuelo del dron o muestra los valores que nos pueden proporcionar sus sensores:

- reboot: Reinicia el dron.
- arm: Habilita los medidores propios del dron. Ejemplo de ejecución: check (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. Ejemplo de ejecución: velocity 1 0 0
- 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.

Capítulo 5

Visor UAV Viewer

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 estos vehículos. Estas estaciones pueden ser fijas o móviles y permiten al operario que maneje el dron conocer su estado de una manera simple. Uav-viewer es una estación de tierra para drones. Para este trabajo se ha focalizado en la visión de los datos generados por Ar.Drone y además permite la teleoperación del cuadricóptero.

En el entorno de JdeRobot existía previamente una interfaz desarrollada en C++, 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. El desarrollo de la aplicación se ha realizado en Python 2.7 para que el entorno de JdeRobot tuviera más variedad en sus aplicaciones, en vez de hacer un cambio en la interfaz gráfica. Las principales diferencias entre ambos visores son el lenguaje y el interfaz gráfico.

Para cumplir con todos los requerimientos que pueden ser necesarios para la plataforma de JdeRobot, el visor que se ha desarrollado es multirobot y se ha probado tanto en simulador como en drones reales, como podría ser Gazebo, ArDrone-Parrot o 3DR Solo Drone.

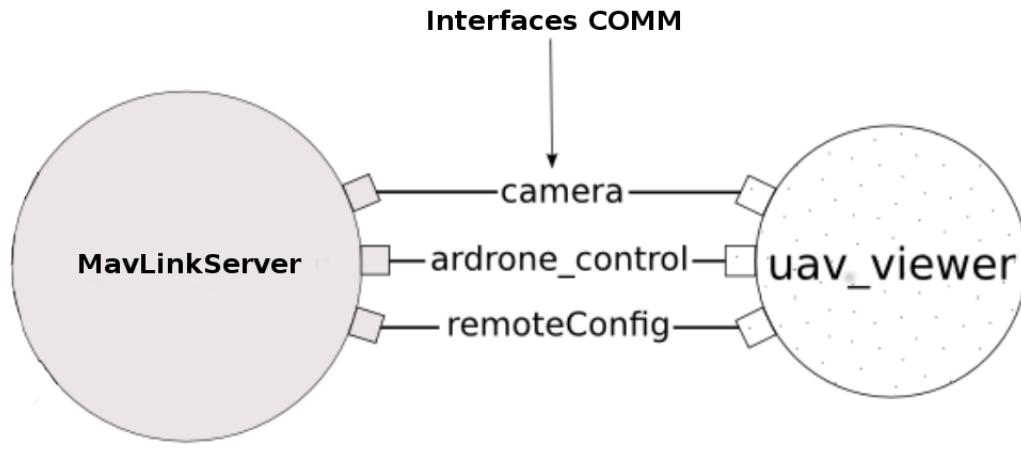


Figura 5.1: Esquema UAV viewer

5.1 Interfaz Gráfica

La aplicación ha sido desarrollada en python utilizando para el apartado gráfico la librería para interfaces de usuario Qt.

En la figura 5.2 se muestra la interfaz de este componente.

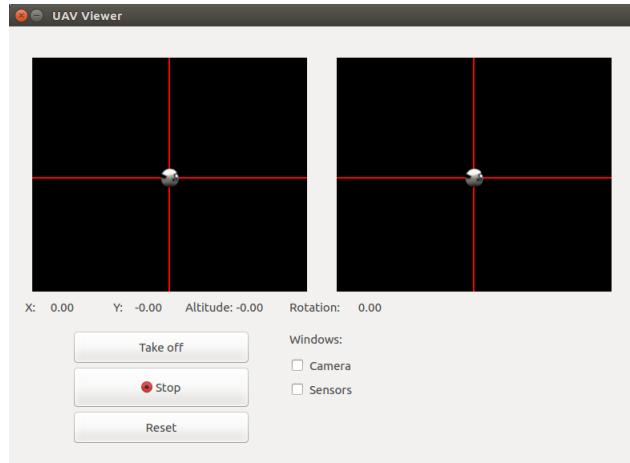


Figura 5.2: Interfaz

Este interfaz se ha realizado de tal manera que se asemeje lo más posible a los mandos de los drones físicos 5.3. De esta manera desde el joystick izquierdo se controla la altura y el giro (o Yaw) y en el joystick derecho se controlan los ejes X e Y.

Tambien hay con 3 botones:

- Take off-Land: sirven para despegar el dron y aterrizar el dron, ese mismo botón cambia el nombre dependiendo el estado del vuelo en el cual nos encontramos.
- Stop: Cuadra los joysticks en la posición (0,0) para detener el dron por completo.
- Reset: Reinicia los valores por defecto.



Figura 5.3: Mando 3dr Solo Dron

La aplicación también implementa de serie 2 utilidades que son la cámara y los sensores. Obtiene las imágenes a través de la interfaz camera, para ello se conecta a la interfaz que ofrece el dron. Con la imagen y los metadatos de ésta recuperados desde el dron, UAV viewer la transforma en una imagen compatible con Qt. En función de la cámara activa, la etiqueta donde se muestran las imágenes cambiará su tamaño para ajustar al ancho y alto de la imagen obtenida. 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.

5.2 Hilos de comunicación

Para teleoperar el dron la interfaz gráfica captura los eventos. Estos eventos son creados por los joysticks que enviaran una señal constante con la velocidad indicada para cada uno de los ejes. La interfaz proporcionada por 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 con el dron hace uso de ésta interfaz para recibir dichos datos. En la figura 5.5 se puede apreciar cómo el hilo que gestiona la interfaz de usuario rellena varios objetos gráficos con los datos sensoriales obtenidos del dron. Podemos ver el porcentaje de batería restante, la altitud del dron, con el indicador de actitud podemos visualizar 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. Además de para teleoperar el dron y poder visualizar los datos sensoriales de éste.

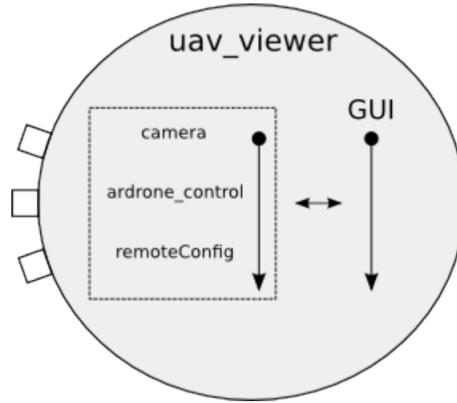


Figura 5.4: Hilos Uav Viewer



Figura 5.5: Sensores

5.3 Fichero de configuración

Se utilizan ficheros de configuración YAML, es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl, así como el formato para correos electrónicos especificado por el RFC 2822. YAML fue propuesto por Clark Evans en 2001, quien lo diseñó junto a Ingy döt Net y Oren Ben-Kiki.

```

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

```

A continuación se muestra el main de la aplicación en el cual iniciaremos la conexión con el interfaz COMM, este interfaz se situa entre nuestra aplicación y tanto los interfaces ICE como los interfaces de ROS. Este nuevo interfaz 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.

```

cfg = config.load(sys.argv[1])

#starting comm
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")

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()

```

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 para la teleoperación. Se ha detallado el diseño, la implementación del driver y de la herramienta de la teleoperación. Además, las pruebas realizadas de ambas herramientas conjuntamente en distintos escenarios, uno para que el robot aéreo comunicándose mediante MAVLinkServer junto a la herramienta UAV Viewer, otro para la integración del UAV Viewer con el robot aéreo Parrot y otro para que el UAV Viewer se integre con el simulador Gazebo y tener una cobertura completa para cualquier tipo de escenario. 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 para la teleoperación. Se ha detallado el diseño, la implementación del driver y de la herramienta de la teleoperación. Además, las pruebas realizadas de ambas herramientas conjuntamente en distintos escenarios, uno para que el robot aéreo comunicándose mediante MAVLinkServer junto a la herramienta UAV Viewer, otro para la integración del UAV Viewer con el robot aéreo Parrot y otro para que el UAV Viewer se integre con el simulador Gazebo y tener una cobertura completa para cualquier tipo de escenario. 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 base se comunique mediante el protocolo MAVLink. Para conseguir dicho objetivo, en primer lugar se llevó a cabo el desarrollo de un driver que permitiera el control del dron y la lectura de sus sensores desde cualquier componente de la plataforma JdeRobot. En segundo lugar, y haciendo uso de la plataforma desarrollada, se diseñó y se construyó una herramienta que nos permitiera pilotar cualquier tipo de dron soportando en el entorno una nueva versión de

UAV-Viewer JdeRobot.

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. Además, se ha documentado el progreso mediante vídeos, imágenes y texto en la bitácora del proyecto, que es accesible públicamente desde Internet y donde cualquier usuario puede acceder a toda la documentación y el código generado en este trabajo. Este Trabajo Fin de Grado, se ha incluido en una ponencia que ha sido aceptada en el congreso CivilDron 2018 [?], titulada "Programación de aplicaciones para drones con el entorno software JdeRobot".

- El primer subobjetivo del proyecto era el desarrollo dentro de la plataforma JdeRobot de la infraestructura software 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, desde el SDK oficial de MAVLink. La base de este proyecto se basa en una ampliación del Proyecto Fin de Grado de Jorge Cano ¹, cuya versión sirvió como base pero se decidió reprogramarla desde 0. Los principales motivos fueron que no estaban bien diferenciadas las partes de servidor y cliente, lo cual era una fuente de problemas a la hora de codificar futuras mejoras. Finalmente se decidió desarrollar un envoltorio que permitiera la interacción de componentes JdeRobot con el SDK oficial de MAVLink. El resultado es el componente 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 ICE, tanto las interfaces estándar de JdeRobot (que permiten que cualquier componente obtenga las imágenes del dron o enviar ficheros de configuración "en caliente"), como la interfaz específica que permite el control del cuadricóptero desde otros componentes de una manera sencilla y eficiente. Dada 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. La concurrencia del componente permite que éste pueda responder a las peticiones de otros componentes mientras realiza otras acciones como la obtención de las imágenes del cuadricóptero o el envío de órdenes de movimiento

- El segundo subobjetivo del proyecto era el diseño y desarrollo de una herramienta de teleoperación que gobernase el movimiento del cuadricóptero.

¹<http://jderobot.org/J.canoma-tfg>

Además, se encontró la problemática que la interfaz ya existente, se diferenciaba mucho de un control de dron habitual, este cambio acerca a gente interesada en drones a la plataforma JdeRobot de una manera más simple. Este componente ya ha sido probado y utilizado en el Trabajo Fin de Grado de Jorge Vela ²[?].

Utilizando las interfaces ICE que MAVLinkServer ofrece, UAV Viewer las explota permitiendo la teleoperación del dron y el visionado de sus sensores. Este visor ha sido desarrollado para poder operar con otros drones, siempre y cuando éstos ofrezcan las interfaces necesarias (ver sección 5.1). La interfaz gráfica de éste componente está descrita en un fichero XML, lo que permite una rápida modificación de la estética e incluso la funcionalidad de dicha interfaz. 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 de control que gestionan la comunicación con MAVLinkServer.

- El tercer y último subobjetivo fue la validación experimental de ambos componentes desarrollados. En esta fase se realizaron una gran cantidad de pruebas, desde experimentos para justificar el correcto funcionamiento del driver y el componente UAV Viewer 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 tener una serie de objetivos, también se definieron una serie de requisitos que la solución tenía que cumplir.

- El primer requisito era que la infraestructura a desarrollar para MAVLink versión 1.0 tenía que estar integrada en JdeRobot. El github oficial de MAVLink ³, ha servido como apoyo y sobre todo como guía para las primeras etapas del desarrollo. Todas las aplicaciones desarrolladas en el proyecto son componentes JdeRobot, compatibles con JdeRobot 5.6. Están integradas en el repositorio oficial y han sido probadas por terceros (ver secciones 4.1 y 5.1).
- El segundo requisito indica que la infraestructura debe ser eficiente computacionalmente. Todos los componentes desarrollados en el proyecto se basan en un diseño multihilo (ver secciones 4.1 y 5.1) 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 nos permite ajustar la duración de cada ciclo controlando así el tiempo que un hilo permanece ocioso.
- El SDK oficial de MAVLink para el desarrollo de aplicaciones, es un SDK complejo y a menudo complicado de entender. Para cumplir con el tercer requisito se implementaron las interfaces

²<http://jderobot.org/Jvela-tfg>

³<https://github.com/mavlink/mavlink>

ICE, que permiten que un desarrollador que quiera hacer aplicaciones con 3DR Solo Dron utilizando MAVLinkServer pueda utilizar el cuadricóptero como si fuera un objeto más de su clase obviando todas las dificultades que supone trabajar con dispositivos de bajo nivel. Es importante que las aplicaciones que se desarrollen funcionen con independencia del driver del dron: Solo 3DR, Parrot ArDrone, Gazebo.

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: El desarrollo de software a bajo nivel que, haciendo uso del SDK de MAVLink se utilizó para el desarrollo de MAVLinkServer; El desarrollo de interfaces gráficas de usuario con el uso de librerías gráficas que permitieron desarrollar las interfaces de UAV Viewer.

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, 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. Además, y pese a las piezas reemplazadas, es una plataforma muy robusta que soportó numerosos accidentes a lo largo de este trabajo sin sufrir daño alguno.

Es importante tener en cuenta las normativa de drones vigente en cada momento, recientemente ha sido actualizada (Diciembre 2017), en la cual se especifican con mayor detalle respecto a las anteriores normativas, el entorno y las situaciones climáticas en las que esta permitido el uso de los drones. El 3DR Solo Dron 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 tenemos más baterías, supone que como mínimo tendremos 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.

El cuadricóptero dispone de un soporte para una cámara GoPro con estabilizador gimbal, lo que permite hacer aplicaciones muy enriquecidas ya que se puede abordar cualquier tipo de problema cambiando la posición inicial de la cámara.

6.3 Trabajos Futuros

En primer lugar se debería implementar el soporte para GPS, ya que aumentaría en gran medida el número de aplicaciones que se pueden realizar con MAVLinkServer.

Otra línea a seguir sería incorporar un sistema de auto localización en interiores, este punto es delicado ya que la potencia del 3DR Solo Dron hace que sea peligroso volar en interiores con espacios reducidos.

Por último realizar un aterrizaje visual con el 3DR Solo Dron, este aterrizaje se ha probado en el Proyecto de Fin de Grado de Jorge Vela[?].

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. *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] Página oficial de JdeRobot. http://jderobot.org/Main_Page
- [8] Página oficial de MAVLink. <https://mavlink.io/en/>
- [9] Página oficial de Python. <https://www.python.org/>
- [10] Página oficial de 3DR. <https://www.parrot.com/es/>
- [11] Página oficial de Pixhawk. <https://pixhawk.org/>
- [12] Página oficial de ICE. <https://zeroc.com/products/ice>
- [13] Normativa Diciembre 2018 sobre drones en España. <https://solodronesbaratos.com/drones-espana/>