



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	Tipo de dron	6
1.4.2	Espacio aéreo	6
1.4.3	Seguridad	6
1.4.4	Carnet de piloto de Drones en España	6
2	Objetivos	7
2.1	Problemas a abordar	7
2.2	Requisitos	7
2.3	Metodología y plan de trabajo	8
3	Infraestructura	10
3.1	Hardware	10
3.1.1	Dron	10
3.1.2	Pixhawk 2	11
3.2	Software	11
3.2.1	JdeRobot	11
3.2.2	Interfaces	12
3.3	MAVLink	13
3.4	Python	14
3.5	ICE	14
4	MavLinkServer	16
4.1	Diseño	17
4.2	Pre-Configuración	21

4.3	Implementación	21
4.3.1	Script de arranque	21
4.3.2	MavLinkServer	23
5	Visor Uav Viewer	30
5.1	Diseño	30
5.2	Implementación	34
6	Conclusiones	36
7	Bibliografia	37

Indice de imágenes

1.1	Ryan Firebee	1
1.2	Gnat	2
1.3	Clasificacion por alas	3
1.4	Clasificacion por aplicación	5
3.1	3DR Solo Drone	10
3.2	Diagrama comunicacion	11
4.1	Diagrama comunicación	16
4.2	Diagrama bloques MavLink-JdeRobot	18
4.3	Protocolo Mavlink	20
4.4	MavProxy gestión	27
5.1	Esquema Uav viewer	30
5.2	Interfaz	31
5.3	Mando 3dr Solo Dron	32
5.4	Hilos Uav Viewer	33
5.5	Sensores	34

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 drone 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.
- Preprogramado: 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 drone 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 drone. El drone 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.
- 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

En el documento oficial del Estado quedan reflejadas las condiciones en las que se puede realizar trabajos técnicos y científicos, tales como grabación aérea, reportajes aéreos, fotografía aérea, estudios de fotogrametría, vigilancia y monitoreo y revisión de infraestructuras entre otros. Gran parte de este nuevo decreto de ley temporal, se basa en 4 puntos clave que toda empresa que desee operar con drones deberá contemplar y seguir:

- Tipo de Drone
- Espacio aéreo

- Seguridad
- Carnet de piloto de Drone

1.4.1 Tipo de dron

Se establecen dos categorías inciales: Drones con peso inferior a 2Kg. y drones con peso entre los 2Kg. y 25Kg. Para ambos es imprescindible disponer de un carnet de piloto de drones para poder operar en España. En caso de los drones de peso inferior a 2kg, no será necesario que estén inscritos en el registro de aeronaves ni disponer de un certificado de aeronavegabilidad. Para ambos tipos de drone, será necesario incluir obligatoriamente una placa identificativa con el nombre del fabricante del aparato así como los datos fiscales de la empresa que lleve a cabo dichas operaciones.

1.4.2 Espacio aéreo

El espacio aéreo pertenece a AESA, y como tal, para poder realizar cualquier tipo de actividad comercial o civil con un drone, se deberá obtener un permiso oficial, como mínimo 5 días antes de llevar a cabo cualquier operación en el aire. Esta nueva legislación sigue manteniendo la prohibición de sobrevolar núcleos urbanos o espacios con una alta masificación de gente sin el consentimiento especial por parte de la Agencia Española de Seguridad Aérea.

1.4.3 Seguridad

El pilar fundamental en el que se ha basado el Ministerio para la realización de la normativa de uso de drones civiles en España es la seguridad. Por ello cada empresa deberá disponer de un manual de operaciones cumplimentado siguiendo el estándar proporcionado por el Ministerio, así como un estudio de seguridad de cada una de las operaciones a realizar. Es decir, si alguien piensa en hacer volar un drone al margen de la ley, ya sea con un peso inferior a 2kg, o entre 2kg y 25kg, se expone a sanciones que van entre 3.000 a 60.000 euros.

1.4.4 Carnet de piloto de Drones en España

Para que las empresas puedan operar legalmente, como lo hace Dronair, los pilotos designados deberán disponer de un carnet oficial para el manejo de drones.. Si estos pilotos ya disponen de un título de piloto de avión, ultraligero u otro específico, no será necesario obtener dicha titulación. En caso contrario deberán cursar una serie de exámenes y pruebas oficiales para obtener el carnet oficial de piloto de drones. A día de hoy, no existen academias oficiales bajo la tutela del Gobierno que realicen estos cursos, por eso y mientras se empiezan a impartir estos cursos, será obligatorio demostrar que se dispone de los conocimientos teóricos y algún tipo de carnet oficial o documento que acredite a los pilotos en el manejo de drones para poder llevar a cabo cualquier operación.

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 Hardware

3.1.1 Dron

Para este desarrollo hemos elegido el dron 3DR 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 eligio esa debido a que era una versión estable y en versiones posteriores del software aun no se han corregido el principal fallo de conectividad debido a usar el mando como puerta de enlace. A bordo de este dron se encuentra una placa Pixhawk 2, con la que nos debemos comunicar a traves del mando con el que se pilota el dron. Este inconveniente 3DR trata de solventarlo para evitar tener que usar el mando como enlace obligatorio.



Figura 3.1: 3DR Solo Drone

3.1.2 Pixhawk 2

Esta placa se trata de un desarrollo específico creado entre la "Pixhawk 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 ordenes al piloto automático quien las ejecutará más adelante trataremos el protocolo MAVLink en profundidad. El único modo de conectarnos con el 3DR Solo sera a través del mando, debido a que únicamente el mando es capaz de levantar la dirección IP a la que poder conectarnos. En posteriores evoluciones de la versión que controlan tanto el dron como el mando, desde los foros oficiales de 3DR, comentan que ya se aborda la solución de que sea el propio dron quien levante la dirección Ip a la cual poder conectarnos y no tener que depender del enlace del mando.

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



Figura 3.2: Diagrama comunicacion

3.2 Software

3.2.1 JdeRobot

JdeRobot es un framework desarrollado por el laboratorio de robótica de la Universidad Rey Juan Carlos, para el desarrollo de aplicaciones de robótica. Su última realease la 5.5 se liberó el 15 de Marzo de 2017 pudiendo ver los detalles de ésta en el github oficial¹. 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

¹<https://github.com/JdeRobot/JdeRobot/wiki/JdeRobot-5.5.0>

interconectar entre sí todos los aplicativos del mismo y en Zeroc ICE para la comunicación entre ellos. Algunos de los driver mas importantes que contiene serían:

1. MAVLinkServer. Desarrollado para intercomunicar JdeRobot con placas que utilicen el protocolo de comunicación MAVLink. Desarrollo que se expone en este TFG.
2. Camerasher. Se trata de un driver para enviar imágenes y video a través del interfaz camera
3. Gazeboserver. Driver desarrollado para conectar la herramienta de simulación Gazebo con JdeRobot y así poder simular los desarrollos.
4. Ardrone_server. Driver que conecta el Parrot Ar-Drone a JdeRobot. Este driver escrito en c++ transforma el set 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 aplicaciones 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. Aplicación que se explicará con más detalle en este TFG más adelante.

3.2.2 Interfaces

JdeRobot expone más de 30 interfaces pero en este capítulo explicaremos los que durante nuestro desarrollo hemos implementado:

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

```
Pose3DData
{
    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.
```

3.3 MAVLink

MAVLink siglas de Micro Air Vehicle Link es un protocolo de comunicación desarrollado para comunicar las placas estabilizadoras con piloto automático a los GCS o Ground control station, las aplicaciones desde las que se podía enviar misiones y seguir el cumplimiento de las mismas desde tierra. MAVLink se publicó 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, APM² y Parrot AR.Drone.

Un ejemplo de comando MAVLink sería:

```
type GpsStatus struct {
    SatellitesVisible  uint8      Número de satélites visibles
    SatellitePrn       [20]uint8   Id Global de cada satélite
    SatelliteUsed      [20]uint8   Lista con el uso de cada satélite
    SatelliteElevation [20]uint8   Elevaci\'on, nos da el ángulo sobre el horizonte.
    SatelliteAzimuth   [20]uint8   Direcci\'on del satélite, 0: 0 grados, 255: 360 grados.
    SatelliteSnr        [20]uint8   Señal/ruido de cada uno de los satélites
}
```

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. Otro parámetro, esta vez vinculado a la actuación sería:

```
type MissionItem struct {
    Param1      float32     parámetro variable en funci\'on del comando.
```

²Ardupilot Mega

```

Param2      float32   parámetro variable en función del comando.
Param3      float32   parámetro variable en función del comando.
Param4      float32   parámetro variable en función del comando.
X          float32   latitud
Y          float32   longitud
Z          float32   altitud
Seq         uint16    Número del ítem en la misión
Command     uint16    Tipo de comando de navegación.
TargetSystem uint8    ID del sistema
TargetComponent uint8
Frame        uint8    Sistema de coordenadas que se utiliza.
Current      uint8    Misión actual no:0, si:1
Autocontinue uint8    Autocontinuar al siguiente objeto de misión.

}

```

3.4 Python

Python 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.

3.5 ICE

ICE (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 una plataforma 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.

Capítulo 4

MavLinkServer

El driver MavLinkServer es quien va a mediar entre las aplicaciones de JdeRobot y el dron con sus sensores y actuadores físicos. De ésta manera las aplicaciones pueden correr en máquinas distintas, no obligatoriamente abordo, y pueden estar implementadas en distintos lenguajes de programación. Estas ventajas vienen de utilizar la división habitual en JdeRobot entre componentes drivers y componentes aplicación. La comunicación será vía WIFI desde el Intel Computer Stick y el mando, ya que debido a un requerimiento que se esta realizando por parte del equipo de 3DR, es obligatorio realizar la conexión con el 3DR Solo dron ¹.

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.

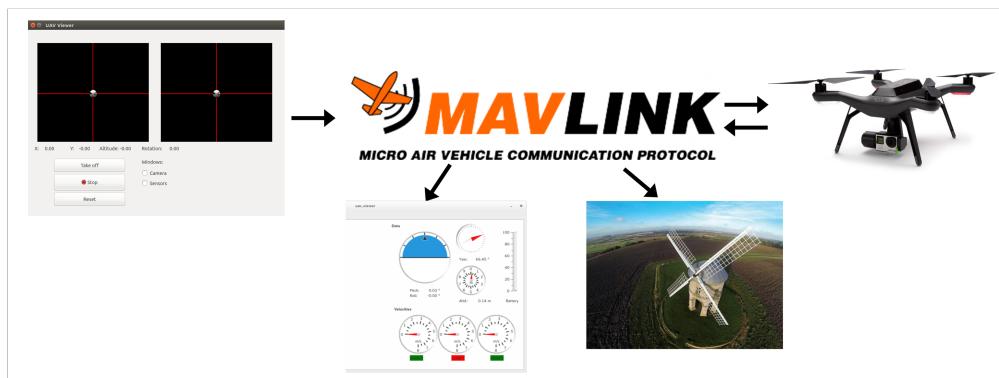


Figura 4.1: Diagrama comunicación

¹<https://discuss.dronekit.io/t/how-to-connect-to-3dr-solo-drone-using-only-pc-without-using-the-rc/>
412

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

- Diseño
- Pre-Configuración
- Implementación

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 componente 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 componente sea utilizable para aplicaciones JdeRobot. Estos dos lados del componente 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 otros componentes.

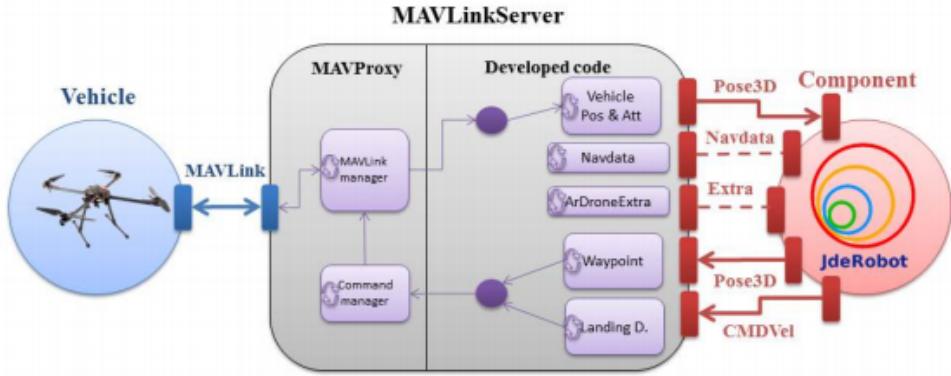


Figura 4.2: Diagrama bloques MavLink-JdeRobot

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.

MavProxy es un componente 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 otros componentes 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 subprocesso 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 componente 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 | • uint16 | • int32 |
| • uint8 | • int16 | • uint64 |
| • int8 | • uint32 | • int64 |

Estos paquetes que se envían tienen la estructura de la imagen 4.3, y en la tabla 4.1 se hace referencia a cada uno de los campos con una breve explicación de cada uno de ellos:

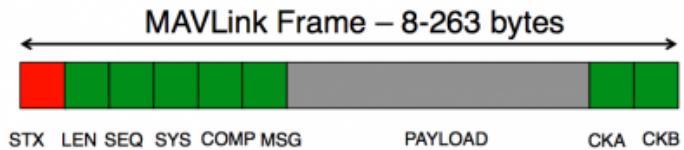


Figura 4.3: Protocolo Mavlink

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).

4.2 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:².

```
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)
    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.

4.3 Implementación

4.3.1 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

²<https://github.com/ArduPilot/MAVProxy>

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 (todos los comandos deben introducirse a continuación con “–” por delante):

- 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 defecto=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.

4.3.2 MavLinkServer

En el arranque del servidor se cargan los paquetes de antes comentados 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 siguiente lista:

- 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.

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 4.3.1. 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. Despues 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()

    # call optional module idle tasks. These are called at several hundred Hz
    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 componentes 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 componente 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 enable traceback" % ex)
    return False

```

- CMDVel: Este componente 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.
- Extra: Este componente nos dará la facilidad de despegar y de aterrizar. Debido al protocolo MayLink, 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 componente nos indicara la posición del dron en todo momento, así como su orientación y altitud. Esta conexión nos va a servir tanto a la hora de aterrizar y despegar usando el otro componente comentado Extra, para saber si puede aterrizar en un determinado momento, o debe disminuir su altura antes de parar los motores, como también a la hora de usar el otro componente CMDVel, ya que si necesitamos girar el dron sobre el yaw, MavLink tiene el inconveniente de necesitar ordenes de velocidad con grados absolutos de -180 a 180°, de modo que es necesario saber la posición en cada momento del dron para indicarle un movimiento relativo. La conversión se realiza de la siguiente forma:

```
angular = Pose3D2send.q3 + CMDVel.angularZ
if angular > 1:
    angular = angular - 2
elif angular < -1:
    angular = angular + 2
angularZstring = str(angular*180)
```

A través de la posición absoluta actual y con la orden que nos indican mediante el componente CMDVel, calcularemos el punto destino al cual debe rotar el dron.

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.

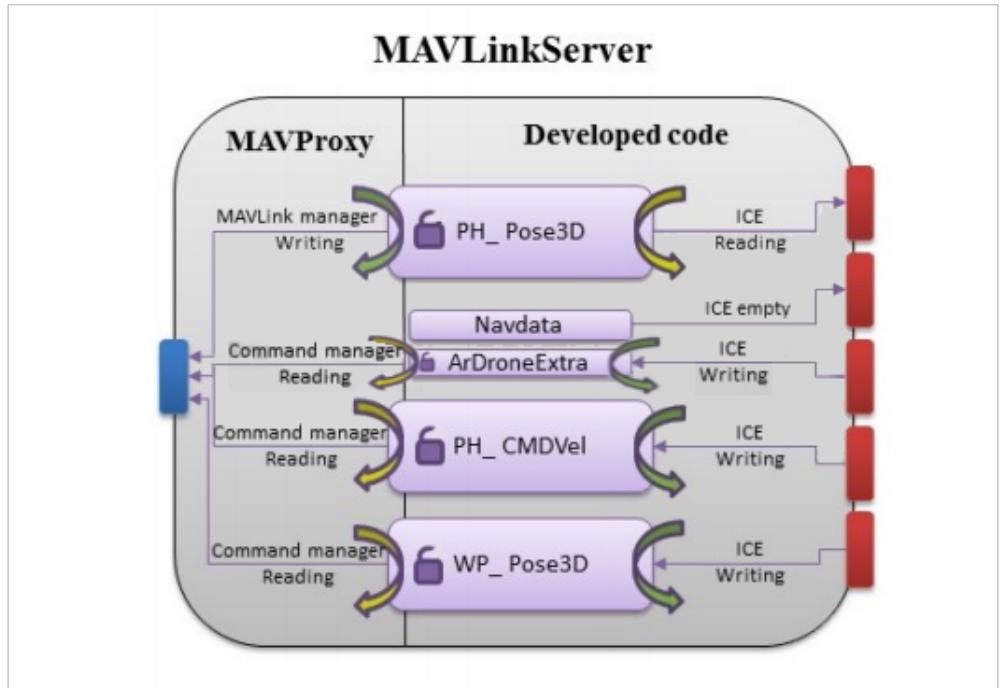


Figura 4.4: MavProxy gestión

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.

```
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 componente 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.

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.

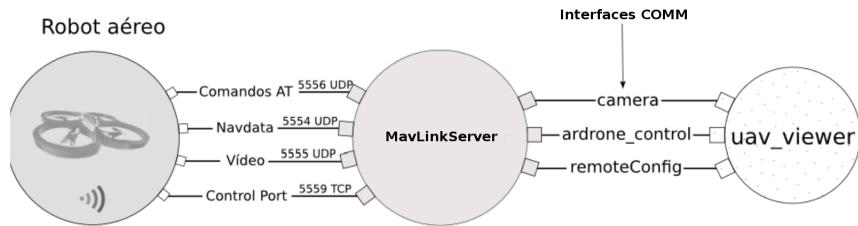


Figura 5.1: Esquema Uav viewer

5.1 Diseño

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.

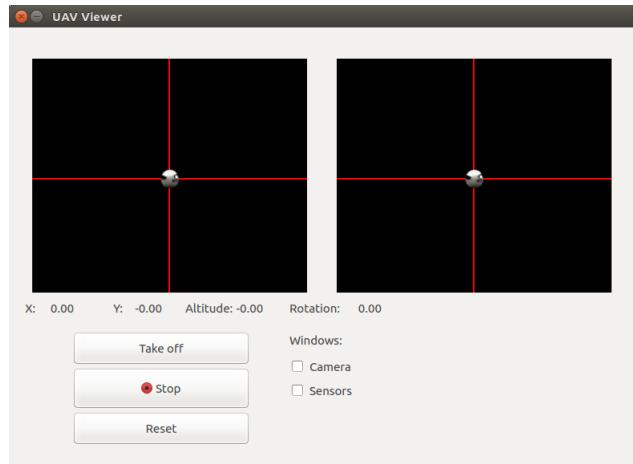


Figura 5.2: Interfaz

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 encontremos
- 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 tambien implementa de serie 2 utilidades que son la camara 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.

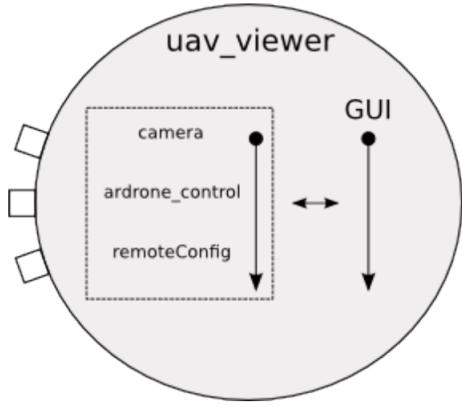


Figura 5.4: Hilos Uav Viewer

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 ?? 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 5.5: Sensores

5.2 Implementación

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.

```
UAVViewer:
  Camera: "default -h 0.0.0.0 -p 9995"
  Pose3D: "default -h 0.0.0.0 -p 9996"
  Navdata: "default -h 0.0.0.0 -p 9997"
  CMDVel: "default -h 0.0.0.0 -p 9998"
  Extra: "default -h 0.0.0.0 -p 9999"
```

A continuación se muestra el main de la aplicación en el cual iniciaremos la conexión con el interfaz COMM, este interfaz se sitúa 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()
```

Una vez iniciada la aplicación cada componente

Capítulo 6

Conclusiones

Capítulo 7

Bibliografia