UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño e implementación del módulo de comunicaciones entre un sistema de propulsión basado en pila de hidrógeno y el sistema autopiloto para vehículos aéreos no tripulados.

Trabajo Fin de Grado

Grado en Ingeniería Aeroespacial

AUTOR/A: Ceacero Escrig, Pablo

Tutor/a: García-Nieto Rodríguez, Sergio

CURSO ACADÉMICO: 2022/2023

El presente documento comprende la memoria, el manual de programación, el presupuesto y el pliego de condiciones, dispuestos en el orden mencionado.

Universidad Politécnica de Valencia
(UPV)

ETSID

Diseño e implementación del módulo de comunicaciones entre un sistema de propulsión basado en pila de hidrógeno y el sistema autopiloto para vehículos aéreos no tripulados.

Pablo Ceacero Escrig

Tutor: Sergio García-Nieto Rodríguez
Grado: Ingeniería en Aeroespacial
Curso: 2022/2023

4 de julio de 2023

Trabajo de Fin de Grado [Memoria]

Agradecimientos

En este momento culminante de mi carrera académica, deseo expresar mi más profundo agradecimiento a todas las personas que han contribuido de manera significativa a la realización de este Trabajo de Fin de Grado.

En primer lugar, me gustaría agradecer sinceramente a mi tutor, Sergio García-Nieto Rodríguez, por su inestimable orientación y explicaciones a lo largo de todo el proceso del TFG. Su conocimiento y apoyo fueron fundamentales para lograr el desarrollo de este trabajo final de grado. Estoy profundamente agradecido por su dedicación y disposición para escuchar mis ideas y brindarme valiosos consejos.

Asimismo, quiero expresar mi gratitud a la Universidad Politécnica de Valencia, así como al Instituto de Automática e Informática Industrial, por brindarme la oportunidad de llevar a cabo este Trabajo de Fin de Grado. La calidad de la educación y los recursos disponibles en esta institución han sido indispensables para el desarrollo exitoso de mi proyecto, y me siento honrado de haber formado parte de ella.

También quiero aprovechar esta oportunidad para agradecer a mis amigos y familiares por su constante apoyo, incluso en los momentos más complicados y estresantes. Su presencia y aliento incondicional han sido un pilar fundamental en mi trayectoria académica y personal.

Por último, pero no menos importante, quiero expresar mi profundo agradecimiento a mis padres, Jorge y Ester. Su dedicación y sacrificio en pos de mi educación han sido inigualables. Siempre han colocado mi formación como una prioridad, incluso por encima de sus propios intereses. Sus palabras de aliento, paciencia y comprensión han sido fundamentales para mantenerme motivado y superar los desafíos que este trabajo implicaba. Su amor y confianza en mí han sido el motor que me ha impulsado a dar lo mejor de mí en cada etapa de este proyecto, así como en mi carrera en general.

En resumen, mi trayectoria hasta este punto ha estado llena de desafíos y satisfacciones, pero no habría podido alcanzar este logro sin el apoyo y la ayuda de todas las

personas mencionadas anteriormente. Estoy sinceramente agradecido por su tiempo, dedicación y contribución a mi Trabajo de Fin de Grado.

Una vez más, quisiera expresar mi más profundo agradecimiento a todos aquellos que han sido parte de este viaje. Vuestra presencia ha dejado una huella imborrable en mi formación académica y personal. Espero que mis logros no solo sean míos, sino también un testimonio de la importancia del trabajo en equipo y la colaboración.

Con gratitud,

Pablo Ceacero Escrig

Resumen
Este trabajo se centra en el desarrollo de un módulo de comunicaciones para establecer una conexión efectiva entre una pila de hidrógeno y el Sistema de Control en Tierra (GCS), con el objetivo de supervisar el rendimiento y el estado de la pila en tiempo real. Se ha utilizado el software PX4 y el protocolo MAVLink para implementar este módulo, estableciendo una comunicación bidireccional sólida entre la pila de hidrógeno y la plataforma de control. Además, se ha personalizado el GCS con el software QGroundControl para mostrar de manera clara y concisa las variables relevantes en la pantalla principal. Esta personalización facilita la monitorización en tiempo real de los parámetros clave de la pila de hidrógeno, lo cual es esencial para tomar decisiones informadas y realizar un seguimiento preciso del rendimiento energético.

Abstract This work focuses on the development of a communication module to establish an effective connection between a hydrogen stack and the Ground Control System (GCS), with the aim of monitoring the performance and state of the battery in real time. The PX4 software and the MAVLink protocol have been used to implement this module, enabling a robust bidirectional communication between the hydrogen stack and the control platform. Additionally, the GCS has been customized using QGroundControl software to display the relevant variables clearly and concisely on the main screen. This customization allows for real-time monitoring of key parameters provided by the hydrogen battery, which is essential for making informed decisions and accurately tracking energy performance.

Resum
Aquest treball se centra en el desenvolupament d'un mòdul de comunicacions per a una connexió efectiva entre una pila d'hidrogen i el Sistema de Control a Terra (GCS), amb l'objectiu de supervisar el rendiment i l'estat de la bateria en temps real. S'ha utilitzat el programari PX4 i el protocol MAVLink per implementar aquest mòdul, establint una comunicació bidireccional robusta entre la pila d'hidrogen i la plataforma de control. També s'ha personalitzat el GCS amb el programari QGroundControl per mostrar de manera clara i concisa les variables rellevants a la pantalla principal. Aquesta personalització facilita la monitorització en temps real dels paràmetres clau de la bateria d'hidrogen, essencial per prendre decisions informades i fer un seguiment precís del rendiment energètic.

Índice general

1. Introducción **1**
1.1. Estado del arte 1
1.2. Descripción y objetivos 7

2. Objetivo y alcance del proyecto **9**

3. Descripción del proceso **11**
3.1. UAV Horus 11
3.2. Pila de combustible AV1000 12
3.3. Integración del sistema de propulsión 17

4. Soluciones alternativas **20**
4.1. Soluciones Alternativas Hardware: Pixhawk, veronte/embention, pixracer, APM, matek, CC3D, etc. 20
4.2. Soluciones Alternativas Software Autopiloto: PX4, ardupilot, betaflight, librepilot, etc. 25
4.3. Soluciones Alternativas Software GCS: qgroundcontrol, missioncontrol, Betaflight, Librepilot, etc. 28

5. Descripción solución adaptada **32**
5.1. Descripción general 32
5.2. Hardware Pixhawk 2 33
5.3. Software PX4 35
5.3.1. Programación de apps en el entorno de PX4 38
5.3.2. Declarar nuevos mensajes en el bus uORB 45
5.3.3. Mavlink: concepto y necesidad 47
5.4. Software Qgroundcontrol 51
5.5. Integración de todos los elementos 62

6. Test experimentales **64**

7. Conclusiones **67**

Capítulo 1

Introducción

Este trabajo se enmarca dentro del proyecto Hydrone, que tiene como objetivo principal diseñar, construir y controlar un sistema propulsivo híbrido para mejorar significativamente la autonomía de vuelo de los vehículos aéreos no tripulados (UAV) de ala fija. Con el fin de contextualizar adecuadamente el proyecto, se presentarán a continuación definiciones y clasificaciones pertinentes relacionadas con los UAV.

1.1. Estado del arte

Definición y clasificación

Un Vehículo Aéreo no Tripulado (UAV, por sus siglas en inglés) es una aeronave que puede operar sin la necesidad de un piloto humano a bordo. Los UAV son controlados de manera remota por operadores humanos o pueden ser programados para operar de forma autónoma utilizando sistemas de navegación y control automatizados. Estas aeronaves son utilizadas en una amplia gama de aplicaciones, desde operaciones militares y vigilancia hasta fotografía aérea, inspecciones industriales, entregas de paquetes y exploración científica.

Además de la denominación UAV, estas aeronaves también son conocidas como UAS (Sistema Aéreo No Tripulado), que se utiliza como sinónimo de UAV y puede referirse tanto al vehículo en sí como a todos los sistemas necesarios para su operación, como lanzaderas, sistemas de recuperación y comunicación. Otros términos utilizados son RPV (Vehículo Pilotado Remotamente) y RPAS (Sistema de Aeronave Pilotada Remotamente), que tienen diferencias sutiles pero similares a las existentes entre UAV y UAS. Es importante destacar que los RPV y RPAS no incluyen necesariamente vehículos completamente autónomos, a diferencia de los UAV. Por último, los términos "dron.º "drone"son utilizados popularmente para referirse a los UAV en general.

Los UAV se componen de varios componentes esenciales, que incluyen:

- **Estructura:** Los UAV tienen una estructura liviana y aerodinámica diseñada para optimizar el rendimiento y la eficiencia durante el vuelo. Pueden estar construidos con materiales como fibra de carbono, aleaciones de aluminio o materiales compuestos.

- **Sistema de propulsión:** Los UAV utilizan diferentes tipos de sistemas de propulsión, como motores de combustión interna, motores eléctricos o una combinación de ambos. La elección del sistema de propulsión depende de factores como la carga útil, el alcance y la duración del vuelo.

- **Sistemas electrónicos y aviónicos:** Los UAV están equipados con sistemas electrónicos y aviónicos que incluyen sensores, sistemas de comunicación, sistemas de navegación por satélite, sistemas de control de vuelo y sistemas de adquisición y procesamiento de datos.

- **Carga útil:** Los UAV pueden llevar diferentes tipos de carga útil, como cámaras de alta resolución, sensores de imagen y video, sistemas de telemetría, equipos de comunicación, sensores de infrarrojos y lidar, entre otros.

- **Estación de control en tierra (Ground Control Station, GCS):** La GCS es un componente esencial en el funcionamiento de los UAV. Se trata de un sistema ubicado en tierra desde el cual los operadores humanos controlan y supervisan el vuelo y las operaciones del UAV. La GCS proporciona una interfaz de usuario para monitorear y controlar el UAV, recibir datos de telemetría en tiempo real, establecer rutas de vuelo y enviar comandos de control. A través de la GCS, los operadores pueden interactuar con el UAV de manera remota y realizar ajustes necesarios durante el vuelo.

Los UAV (Vehículos Aéreos No Tripulados) se clasifican en diferentes categorías y configuraciones, lo que permite una mejor comprensión de sus características y aplicaciones. A continuación, se describen algunas de las clasificaciones más comunes de los UAV:

- **Clasificación por plataforma de vuelo:**

  - UAV de ala fija: Estos UAV tienen una configuración similar a la de un avión convencional, con alas fijas que generan la sustentación necesaria para el vuelo. Son conocidos por su eficiencia en el consumo de combustible y su capacidad para volar a velocidades más altas y durante períodos de tiempo prolongados. Son ampliamente utilizados en aplicaciones como la vigilancia, la fotografía aérea y la recolección de datos científicos.

- UAV de ala rotatoria: También conocidos como multirrotores o drones de rotor múltiple, estos UAV utilizan rotores giratorios para generar la sustentación y el control del vuelo. Son conocidos por su capacidad de despegue y aterrizaje vertical, su maniobrabilidad y su capacidad de vuelo estacionario. Son ampliamente utilizados en aplicaciones como la fotografía y videografía aérea, la inspección de infraestructuras y la entrega de paquetes.
- UAV híbridos: Estos UAV combinan características de los UAV de ala fija y los UAV de ala rotatoria. Pueden tener configuraciones que permiten el despegue y aterrizaje vertical, así como la capacidad de vuelo de largo alcance. Estos UAV híbridos ofrecen la versatilidad de ambos tipos de plataformas y se utilizan en una amplia gama de aplicaciones.

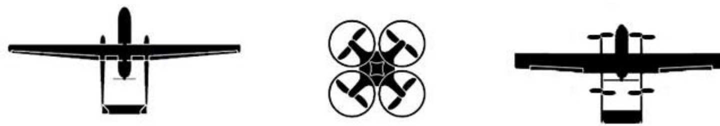

Figura 1.1: UAV de ala fija, de ala rotatoria e hibrido

- **Clasificación por tamaño:**
  - UAV Nano: son los más pequeños dentro de la clasificación de UAVs. Tienen dimensiones extremadamente reducidas, generalmente inferiores a 15 centímetros y un peso ligero. Estos UAVs son altamente portátiles y se pueden transportar fácilmente en la palma de la mano. Debido a su tamaño diminuto, su capacidad de carga útil y autonomía son limitadas. Se utilizan en aplicaciones que requieren acceso a espacios confinados, monitoreo cercano y operaciones en entornos urbanos.
  - UAV Micro: son un poco más grandes que los Nano UAVs y suelen tener envergaduras de alrededor de 15 centímetros a un metro. Estos UAVs son más capaces en términos de carga útil y autonomía en comparación con los Nano UAVs. Se utilizan en diversas aplicaciones, como reconocimiento y vigilancia en áreas de difícil acceso, inspecciones industriales y actividades recreativas.
  - UAV Mini: son UAVs de tamaño mediano, con envergaduras que oscilan entre uno y varios metros. Estos UAVs ofrecen una mayor capacidad de carga útil y autonomía en comparación con los UAVs más pequeños. Se utilizan en una amplia gama de aplicaciones, incluyendo vigilancia militar, monitoreo ambiental, cartografía y fotografía aérea.
  - UAV Pequeños, Medianos y Grandes: Estas categorías se refieren a los UAVs de tamaño más significativo. Los Pequeños UAVs generalmente tienen envergaduras de varios metros, mientras que los Medianos y Grandes UAVs pueden tener

envergaduras de decenas de metros. Estos UAVs son capaces de transportar cargas útiles más pesadas y tienen una mayor autonomía de vuelo. Se utilizan en aplicaciones como la vigilancia de grandes áreas, la entrega de carga, la exploración científica y las operaciones militares estratégicas.

- **Clasificación por capacidad de carga útil:**
  - UAV de vigilancia: Estos UAV se utilizan para recopilar información y realizar tareas de vigilancia a través de cámaras y sensores de alta resolución. Se utilizan en aplicaciones como la vigilancia fronteriza, la seguridad y la vigilancia del tráfico.
  - UAV de carga: Estos UAV tienen una capacidad de carga significativa y se utilizan para transportar mercancías, suministros médicos o equipos especializados. Se están explorando para aplicaciones de entrega de paquetes.
  - UAV especializados: Estos UAV están diseñados para realizar tareas específicas, como la inspección de infraestructuras, la detección y extinción de incendios, el monitoreo de la calidad del aire y la recolección de datos científicos.

Aplicaciones:

El uso de Vehículos Aéreos No Tripulados (UAVs) conlleva una serie de ventajas y desventajas significativas para sus diversas aplicaciones. Una de las ventajas más destacadas es la capacidad de realizar misiones de alto riesgo sin poner en peligro la vida de un piloto, especialmente en aplicaciones militares o de control de incendios. Esto permite abordar situaciones peligrosas de manera más eficiente y efectiva.

Otra ventaja importante de los UAVs es su menor impacto medioambiental en comparación con las aeronaves tripuladas. Debido a su tamaño y peso reducidos, los UAVs pueden utilizar fuentes de energía más limpias y generar menos emisiones contaminantes. Esto los convierte en una opción más sostenible en términos de impacto ambiental.

Además, el uso de UAVs ofrece beneficios económicos y de tiempo al eliminar la necesidad de contar con un piloto humano a bordo. Esto significa que no se requiere el costo adicional asociado con el entrenamiento y el mantenimiento de la tripulación, y las operaciones pueden ser más eficientes y rápidas.

Sin embargo, también existen desventajas asociadas con los UAVs. Una de ellas es la posibilidad de fallos de comunicación durante el vuelo, lo cual puede afectar la transmisión de datos y comandos, poniendo en riesgo el éxito de la misión. Además, las condiciones

climáticas adversas pueden limitar la capacidad de vuelo de los UAVs y afectar su rendimiento.

Otra desventaja importante es la complejidad de integrar los UAVs en el espacio aéreo existente. Esto implica la necesidad de desarrollar regulaciones y sistemas de gestión del tráfico aéreo específicos para garantizar una operación segura y coordinada con otras aeronaves en el espacio aéreo compartido.

En cuanto a las aplicaciones de los UAVs, se puede realizar una distinción fundamental entre las aplicaciones militares y las aplicaciones civiles. Sin embargo, también existe una categoría adicional que se relaciona con la utilización de UAVs en el ámbito ambiental.

A continuacion se enumerarn algunas de las aplicaciones más extendidas de los drones en estos campos:

- **Aplicaciones Militares:**
  - Vigilancia y reconocimiento: Los UAVs son utilizados para obtener información en tiempo real sobre el terreno, identificar objetivos y recopilar datos de inteligencia.
  - Apoyo de combate: Los UAVs pueden desempeñar roles ofensivos y defensivos, como ataques precisos aéreos, supresión de fuego enemigo y apoyo logístico.
  - Monitoreo de fronteras: Los UAVs ayudan en la vigilancia de fronteras, detección de actividades ilícitas y protección de infraestructuras críticas.
- **Aplicaciones Civiles:**
  - Fotografía y cinematografía aérea: Los UAVs se utilizan en la industria del entretenimiento y la producción audiovisual para capturar imágenes y videos desde perspectivas aéreas únicas.
  - Agricultura de precisión: Los UAVs pueden monitorear cultivos, analizar la salud de las plantas y ayudar en la gestión eficiente de recursos agrícolas.
  - Inspección de infraestructuras: Los UAVs facilitan la inspección de puentes, líneas eléctricas, tuberías y estructuras de difícil acceso, reduciendo costos y riesgos humanos.
- **Aplicaciones Ambientales:**
  - Monitoreo de la biodiversidad: Los UAVs se utilizan para estudiar la fauna y flora en áreas naturales, realizar conteos de especies y evaluar la salud de los ecosistemas.

- Control de incendios forestales: Los UAVs ayudan en la detección temprana, seguimiento y extinción de incendios forestales, proporcionando información en tiempo real a los equipos de respuesta.
- Monitoreo de calidad del aire y del agua: Los UAVs pueden recolectar muestras y datos para evaluar la calidad del aire y del agua en zonas de difícil acceso o peligrosas.

Historia

La historia de los UAVs (Vehículos Aéreos No Tripulados) se remonta a varias décadas atrás. Durante la Primera Guerra Mundial, se realizaron los primeros intentos de desarrollar aviones no tripulados para propósitos militares. Uno de los ejemplos notables es el avión de combate a control remoto Kettering Bug [1.2], creado en 1918. Durante la Segunda Guerra Mundial, los UAVs continuaron evolucionando y se utilizaron para labores de reconocimiento y vigilancia. El avión de reconocimiento aéreo a control remoto Radioplane OQ-2 [1.3] fue ampliamente utilizado por el ejército de los Estados Unidos en ese período.

Figura 1.2: Kettering Bug.

Figura 1.3: OQ-2.

Con el avance de la tecnología, se produjeron mejoras significativas en los sistemas de navegación, la telemetría y la miniaturización de los componentes electrónicos. Esto permitió el desarrollo de UAVs más sofisticados y autónomos. A medida que se hacían más eficientes en términos de autonomía y carga útil, los UAVs empezaron a encontrar aplicaciones en diversas industrias. Más allá del ámbito militar, se utilizan en agricultura, fotografía y videografía aérea, inspección de infraestructuras, entrega de paquetes, entre otros sectores.

La expansión del mercado de los UAVs ha llevado a un mayor interés y a una creciente investigación y desarrollo en el campo de los sistemas aéreos no tripulados. Actualmente, los UAVs desempeñan un papel importante tanto en aplicaciones militares como en aplicaciones civiles y comerciales. Su versatilidad, capacidad para operar en entornos peligrosos o de difícil acceso, menor impacto medioambiental y potencial para aumentar la eficiencia y reducir costos han impulsado su crecimiento y adopción en diferentes campos.

1.2. Descripción y objetivos

El proyecto HYDRONE es un esfuerzo de investigación y desarrollo llevado a cabo por un equipo de investigadores del Instituto de Automática e Informática Industrial (Instituto ai2) y del Instituto Universitario de Motores Térmicos (CMT) de la Universitat Politècnica de València.

El objetivo principal del proyecto es diseñar, construir y controlar un sistema propulsivo híbrido que permita ampliar de manera significativa la autonomía de vuelo de los vehículos aéreos no tripulados (UAV) de ala fija. Para lograr esto, el proyecto se enfoca en la hibridación de pilas de hidrógeno tipo PEM con las baterías Li–Po utilizadas comúnmente en los motores eléctricos de los UAV de tamaño micro y mini.

El proyecto HYDRONE tiene como meta extender la autonomía de vuelo de los UAV más allá de los 45 minutos, que es la duración típica en situaciones óptimas. Dependiendo del depósito de hidrógeno utilizado, se pretende alcanzar autonomías de más de dos horas. Para lograr esto, los investigadores están trabajando en el diseño de un sistema de control que tome en cuenta parámetros como el clima, la trayectoria y la altura de vuelo. El software de control se encargará de gestionar el conjunto de baterías Li-PO y pila de hidrógeno de manera óptima, maximizando el tiempo efectivo de vuelo.

Además de mejorar la autonomía de vuelo de los UAV, los resultados del proyecto HYDRONE tienen el potencial de beneficiar diversas áreas de aplicación, como el salvamento marítimo, el control de costas, la vigilancia en acuicultura y el transporte. La autonomía de vuelo es un elemento crítico en todas estas aplicaciones de drones civiles, por lo que el proyecto se ha enfocado en este aspecto durante más de una década de investigación.

El proyecto HYDRONE, que cuenta con una duración de tres años, se realiza en colaboración con el Instituto Universitario de Motores Térmicos (CMT) de la UPV. Uno de los objetivos secundarios del proyecto es diseñar un sistema de propulsión distribuida en los UAV de ala fija. Esto implica la incorporación de varios elementos propulsivos en el

fuselaje, como tres hélices pequeñas en cada ala, lo que permite una fuerza de propulsión equilibrada y eficiente en términos de consumo de energía.

Figura 1.4: Dron de ala fija del proyecto hydrone.

Capítulo 2

Objetivo y alcance del proyecto

El objetivo principal de este Trabajo de Fin de Grado (TFG) consiste en diseñar e implementar un módulo de comunicación entre una pila de hidrógeno y un sistema de piloto automático, utilizando en nuestro caso la placa Pixhawk. Para lograr este objetivo final, se han establecido una serie de objetivos a corto plazo que se describen a continuación de manera formal y detallada:

- Establecer una conexión funcional entre dos puertos serie virtuales mediante el uso de programas como «Virtual Serial Port Tools» y «Putty». Esto permitirá verificar la corrección de nuestro programa de lectura de puertos serie. La verificación se realizará mediante la simulación de la placa Pixhawk en el propio ordenador utilizando el Sistema de Simulación en Tiempo Real (SITL).

- Utilizar la placa Pixhawk y cargar el software PX4 junto con el PL2303, que se utilizará para establecer la conexión entre el puerto del ordenador y la placa. Posteriormente, se implementará la funcionalidad necesaria para que los mensajes recibidos y leídos por el puerto serie de la placa se almacenen en el bus de datos uORB, creando así un nuevo mensaje personalizado.

- Adaptar el programa de lectura de puertos serie para recibir el mensaje enviado por la pila de hidrógeno y extraer toda la información relevante para luego agregarla al mensaje uORB.

- Finalmente, se configurará el mensaje de uORB con la información de la pila y se enviará a través del protocolo MAVLink utilizando el mensaje predefinido «Battery_Status» del software PX4.

Además de los objetivos principales mencionados, se abordarán objetivos secundarios. Entre ellos se incluye la edición del código del QGroundControl, con el fin de visualizar el mensaje de la pila como un widget en la pantalla principal. También se proporcionará la documentación necesaria y se subirá todo el trabajo anteriormente mencionado, incluidos

los archivos originales del PX4, a la plataforma GitHub para que esté disponible y pueda ser utilizado por la comunidad.

Con este proyecto se persigue obtener información en tiempo real sobre el estado de la pila de hidrógeno durante todo el vuelo. El objetivo es monitorear de manera precisa y continua los parámetros relevantes de la pila, como el nivel de carga, la temperatura y cualquier otra información relevante para su correcto funcionamiento. Esto permitirá tomar decisiones informadas y realizar ajustes necesarios en el sistema de piloto automático para garantizar un rendimiento óptimo y seguro del vehículo aéreo no tripulado. El conocimiento detallado del estado de la pila de hidrógeno será fundamental para optimizar la planificación de vuelo y mantener un control adecuado de la energía disponible, asegurando así una operación eficiente y confiable.

Capítulo 3

Descripción del proceso

3.1. UAV Horus

HORUS UPV es un proyecto fundado en 2018 por estudiantes de la Universitat Politècnica de València (UPV) con el propósito de diseñar, fabricar y operar Sistemas Aéreos No Tripulados (UAS, por sus siglas en inglés) versátiles. El objetivo principal del equipo es participar en el UAS Challenge organizado por la Institution of Mechanical Engineers (IMechE), donde presentarán un UAS con un peso máximo al despegue (MTOW) que oscilará entre 7 y 10 kg. Además, se busca utilizar este proyecto como base de investigación para la incorporación de baterías de hidrógeno, lo que permitirá que la aeronave sea más respetuosa con el medio ambiente y tenga una mejora notable en su autonomía.

Actualmente, el equipo de HORUS UPV está conformado por 46 estudiantes de la UPV, especializados en diversas ramas de la ingeniería, como Aeroespacial, Mecánica, Electrónica Industrial y Automática, Diseño Industrial y Desarrollo de Productos, Bellas Artes y Administración y Dirección de Empresas.

HORUS UPV se concibió como una plataforma de investigación para la utilización de baterías de hidrógeno en UAS, con el objetivo de aumentar su alcance de vuelo. Desde su creación, el proyecto ha contado con la colaboración del Instituto de Automática e Informática Industrial de la UPV (ai2), lo que permitió iniciar el desarrollo de dicha aeronave.

Hasta la fecha, se han desarrollado varios prototipos, enfocados en dos líneas principales de trabajo: investigación y competición, con el desarrollo de las series H100 y H110, respectivamente.

Durante el año académico 2022, HORUS UPV participó en el evento de vuelo en julio del UAS Challenge, organizado por la Institution of Mechanical Engineers del Reino Unido (IMechE). El equipo obtuvo los primeros premios en las categorías de aptitud pa-

Figura 3.1: H-120 avión con el que Horus competirá en 2023

ra el vuelo y nuevo participante con mayor clasificación, y recibió menciones especiales en Propuesta de Negocio y Seguridad, logrando un destacado tercer lugar en la competición internacional. De esta manera, HORUS UPV se consagró como el único equipo universitario español que ha obtenido premios en esta competición desde su creación en 2014.

3.2. Pila de combustible AV1000

La pila de combustible es suministrada por la marca H^3*Dynamics*, concretamente el modelo *A1000-50 (LV)* [3.2]. A continuación se hará una breve explicación de como funciona, así como las especificaciones concretas de este modelo:

Funcionamiento

El objetivo principal de una pila de hidrógeno es convertir la energía química del hidrógeno y el oxigeno en electricidad y agua. Para ello es necesario un suministro de hidrógeno, en este caso la propia Pila tiene un tanque de material compuesto, donde se almacena el hidrógeno en forma gaseosa. Y en relación al oxigeno, este se obtiene del aire con unos ventiladores que a su vez hacen funciones de refrigeración.

El funcionamiento básico es el siguiente, en el interior de la Pila se pueden identificar tres componentes principales [3.3]:

- **Electrodo anódico (ánodo):** En el ánodo de la pila de hidrógeno, el hidrógeno

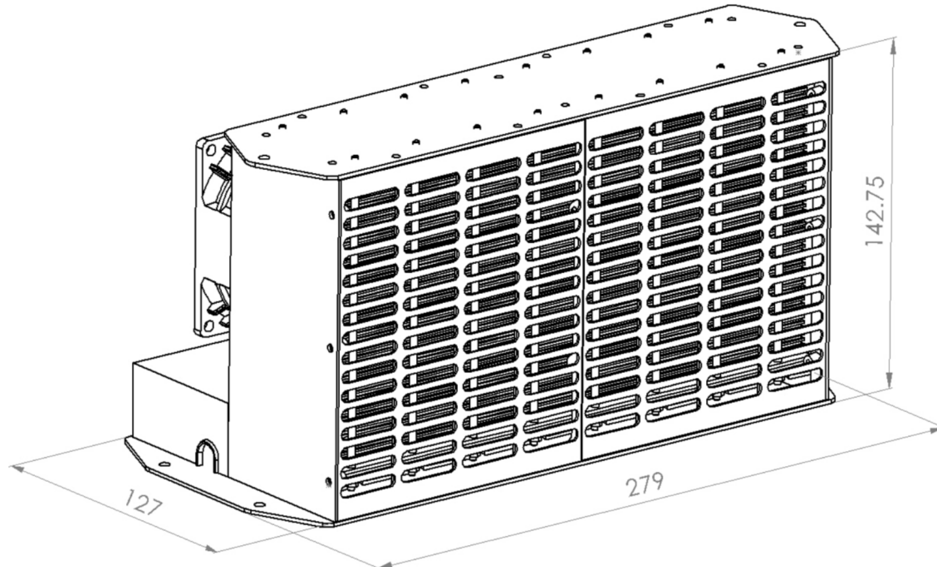

Figura 3.2: Modelo CAD de la Pila AEROSTAK A1000-50 (LV) con carcasa

es ionizado y se descompone en protones (H+) y electrones (e-). Los electrones se liberan y se conducen a través de un circuito externo, creando así una corriente eléctrica.

- **Electrolito:** Entre el ánodo y el cátodo, se encuentra el electrolito, que es una capa o membrana especial que permite el paso selectivo de los iones de hidrógeno (protones) mientras evita el paso de electrones. Hay varios tipos de electrolitos utilizados en diferentes tipos de pilas de hidrógeno, como membranas de intercambio de protones (PEM), óxido sólido (SOFC) o carbonato fundido (MCFC), en este caso es PEM.
- **Electrodo catódico (cátodo):** En el cátodo, los protones (H+) y los electrones (e-) se combinan con el oxígeno (O2) para formar agua (H2O). Este proceso es conocido como reducción del oxígeno.

Como se ha comentado anteriormente, en este proceso se obtiene electricidad y agua (acompañada de una pequeña cantidad de óxidos de nitrógeno.). La electricidad se obtiene durante la reacción entre el ánodo y cátodo, los electrones liberados son forzados a fluir a través de un circuito externo, lo que genera una corriente eléctrica. Por otra parte, esta el agua que es el único subproducto de esta reacción que se libera, en este caso, en forma líquida.

Componentes

Como especificaremos más adelante, este sistema se trata de una combinación híbrida de pila de hidrógeno y baterías de litio debido a las características particulares de las

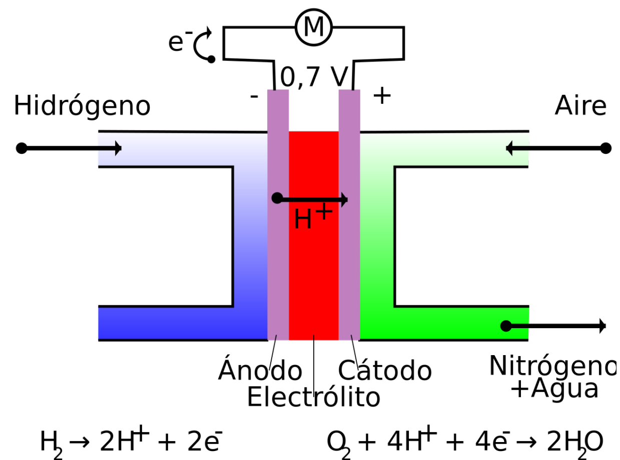

Figura 3.3: Esquema de funcionamiento de una pila de hidrógeno

pilas PEM. El modelo de pila utilizado en este sistema está equipado con los siguientes componentes:

- **Pila de combustible de 1000 W:** La pila de combustible de membrana de intercambio protónico (PEM) actúa como la fuente de energía principal para la carga. La pila de combustible consta de 50 celdas conectadas en serie (eléctricamente) para producir una potencia nominal de 1000 W. Incluye ventiladores que suministran aire/oxigeno al cátodo y flujo de aire para enfriar la pila. El aire entra a través de la rejilla y sale por los ventiladores. También cuenta con una placa de control principal que controla el funcionamiento de la pila y la gestión general del sistema.

- **Tarjeta híbrida:** Gestión del sistema de alimentación de la pila de combustible y de la batería híbrida LiPo. Recarga la batería híbrida LiPo durante el funcionamiento hasta 1.5A.

- **Conectores XT90 macho/hembra:** Conectan la tarjeta híbrida a la carga y a la batería LiPo 8S.

- **Tubo de suministro de H2 + Conexión rápida:** Suministra gas H_2 a la pila desde una fuente externa de H_2. El accesorio de conexión de gas compatible es fabricado por CPC, número de pieza PMCD2004.

- **Tubo de purga de H2:** Elimina el agua y los gases no deseados durante el proceso de purga. Utiliza un tubo de silicona suave de 1/8"de diámetro interno.

- **Cable de comunicación RS232:** Se conecta al sistema, proporcionando datos de rendimiento operativo para su monitoreo y recopilación en una computadora separada.

A: Ventiladores de enfriamiento

B: Purga de hidrógeno

C: Potencia de la pila de combustible

D: Puerto de comunicaciones RS232

E: Transductor de presión de almacenamiento de hidrógeno

F: Botón de inicio / parada

G: Inicio / parada remoto opcional

H: Entrada de hidrógeno

I: Conexión de carga XT90

J: Conexión de batería híbrida XT90

Especificaciones del sistema

Especificación	Unidades	Valor
Tipo de pila de combustible		PEM
Número de celdas		50
Dimensiones	mm	279 x 127 x 145
Peso	g	2150
Potencia nominal	W	1000
Potencia máxima	W	1200
Batería híbrida de LiPo recomendada		8 S
Voltaje de salida nominal	V	32 - 47
Reactantes		Hydrogen & Air
Pureza del gas de hidrógeno	%	99.999
Presión de entrada de hidrógeno	bar	0.6 – 0.85
Consumo de hidrógeno nominal	L/min	11.0
Temperatura de entrada de aire	ºC	0-35
Advertencia de alta temperatura del stack	ºC	>67
Apagado por alta temperatura del stack	ºC	>70
Advertencia de bajo voltaje del stack	V/celda	<0.6
Apagado por bajo voltaje del stack	V/celda	<0.5
Advertencia de bajo voltaje de la batería híbrida de LiPo	V/celda	<3.4
Apagado por bajo voltaje de la batería híbrida de LiPo	V/celda	<3.0
Humidificación		Auto-humidificado
Enfriamiento		Aire (Ventilador)
Tiempo de arranque	s	<20
Eficiencia eléctrica del stack	%	56.5 @ 1080W
Vida útil de diseño	h	500

Conexiones e información

Como se puede observar en las imágenes [3.4], hay un puerto para un cable de conexión RS232. Es este el que se va a emplear para conectarse a la Pila y así recibir las tramas de información que manda el sistema. El sistema trasmite datos ASCII, similares a estos:

```
FCS=50.4V,00.43A,025.1W,000.08Wh,BAT=35.5V,00.1A,LD=51.4V,...
30.3C,30.5C,31.3C,30.2C,TST=39.2C,PCB=025.3C, 0.75B,059.2mVPrs, 25.0%,6,
FCS=50.5V,00.42A,023.1W,000.08Wh,BAT=35.5V,00.1A,LD=50.7V,...
30.4C,30.6C,31.5C,30.1C,TST=39.2C,PCB=025.5C, 0.73B,058.9mVPrs, 25.0%,6,
FCS=50.9V,00.43A,022.1W,000.09Wh,BAT=35.5V,00.1A,LD=50.4V,...
```

```
30.5C,30.8C,31.8C,30.5C,TST=39.7C,PCB=025.9C, 0.72B,058.8mVPrs, 25.0%,6,
*****************************************************************************
```

El formato de la salida de datos RS232 es el siguiente:

- Voltaje pila (V)
- Corriente de carga (A)
- Potencia (W)
- Energía (Wh)
- Voltaje de la pila (V)
- Corriente de la pila (A)
- Voltaje de carga (V)
- Sensor de temperatura de la pila #1 (ºC)
- Sensor de temperatura de la pila #2 (ºC)
- Sensor de temperatura de la pila #3 (ºC)
- Sensor de temperatura de la pila #4 (ºC)
- Temperatura objetivo de la pila (ºC)
- Temperatura de la placa (ºC)
- Presión de suministro de almacenamiento de H_2 (mV)
- Velocidad del ventilador (%)
- Estado de operación

3.3. Integración del sistema de propulsión

Las baterías de hidrógeno, a pesar de ser una opción prometedora en términos de energía limpia y autonomía, presentan limitaciones para enfrentar a cambios bruscos en las demandas de energía, como puede ser el caso de un ascenso en una aeronave. Esto se debe a la naturaleza del proceso de generación de energía, que requiere una reacción química para producir electricidad. Esta reacción es más lenta y limitada en comparación con las baterías de iones de litio (LiPo), que pueden suministrar rápidamente altas corrientes de energía.

Conscientes de esta limitación, el enfoque adoptado en este proyecto es la creación de un avión híbrido que combina tanto una pila de hidrógeno como una batería de LiPo. La idea principal es aprovechar las fortalezas de cada tipo de batería y compensar las debilidades de la pila de hidrógeno en situaciones que requieran cambios bruscos de energía.

Durante maniobras que demanden cambios bruscos en la demanda de energía, como un ascenso o una aceleración rápida, la batería de LiPo asumirá el suministro energético necesario debido a su capacidad para proporcionar rápidamente altos niveles de corriente. Esto permite superar las limitaciones de la pila de hidrógeno en términos de capacidad de respuesta y capacidad de suministro instantáneo de energía.

Por otro lado, en momentos de demanda continua de energía, como durante un vuelo a crucero, la pila de hidrógeno entra en juego al proporcionar un flujo constante y sostenido de amperaje para alimentar el avión y, al mismo tiempo, recargar las baterías de LiPo. De esta manera, se logra un equilibrio entre el uso de la energía almacenada, permitiendo una eficiente gestión energética a lo largo del vuelo.

La combinación de una pila de hidrógeno y una batería de LiPo en este avión híbrido no solo busca superar las limitaciones de las pilas de hidrógeno en términos de cambios bruscos de demandas energéticas, sino que también aprovecha la ventaja de utilizar una fuente de energía limpia y renovable. Además, esta solución híbrida contribuye a maximizar la autonomía del avión al garantizar un suministro energético constante y eficiente en diferentes situaciones de vuelo. La combinación de estas tecnologías proporciona un sistema de propulsión más versátil y eficiente, permitiendo un rendimiento óptimo en términos de potencia y autonomía en el avión.

Además, es importante destacar que el modelo actual del avión cuenta con un único motor ubicado en la parte frontal del fuselaje. Sin embargo, como parte de las futuras mejoras y evoluciones del proyecto, se tiene previsto reemplazar este diseño por una configuración que incluya dos motores en cada ala.

Esta modificación tiene como objetivo principal aumentar la eficiencia del sistema de propulsión y reducir el consumo de energía durante el vuelo. Al distribuir los motores en las alas, se logra una mejor distribución del empuje, lo que resulta en una mayor eficiencia aerodinámica y una reducción en la resistencia inducida. Esto se traduce en un menor consumo de energía para mantener la sustentación y propulsión necesarias para volar.

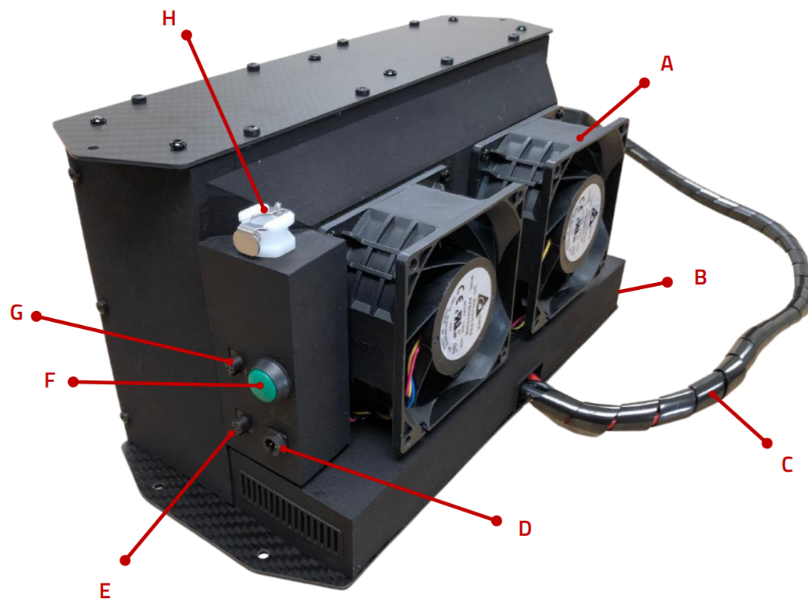

Figura 3.4: Vista trasera de la AEROSTAK A1000-50(LV)

Figura 3.5: Tarjeta híbrida

Capítulo 4

Soluciones alternativas

4.1. Soluciones Alternativas Hardware: Pixhawk, veronte/embention, pixracer, APM, matek, CC3D, etc.

Un hardware para autopiloto de una aeronave es un componente esencial de los sistemas de aviónica utilizados para controlar y gestionar el vuelo de una aeronave de manera automática. Consiste en una combinación de hardware y software diseñados específicamente para funcionar de forma autónoma o en conjunto con los pilotos humanos.

El hardware para autopiloto de una aeronave está compuesto por una serie de dispositivos electrónicos que interactúan entre sí para realizar diversas funciones relacionadas con el control de vuelo. Estos dispositivos incluyen:

- **Computadora de vuelo:** Se trata del núcleo central del sistema, donde se ejecuta el software del autopiloto. Esta computadora procesa los datos recopilados por los sensores y los somete a análisis y toma de decisiones basadas en algoritmos y parámetros preestablecidos. Es responsable de controlar y regular las diversas funciones del vuelo de la aeronave.
- **Sensores:** Estos dispositivos son responsables de recolectar información precisa sobre diferentes aspectos del estado de la aeronave, como su posición, velocidad, actitud (ángulos de inclinación y balanceo), altitud, aceleración y otros parámetros relevantes. Algunos ejemplos comunes de sensores utilizados incluyen giroscopios, acelerómetros, magnetómetros, altímetros, sistemas de posicionamiento global (GPS) y sensores de flujo de aire. La información capturada por estos sensores es esencial para el funcionamiento del autopiloto y se utiliza para mantener el control y la estabilidad del vuelo.

- **Actuadores:** Estos dispositivos electromecánicos son responsables de llevar a cabo las acciones físicas necesarias para controlar los movimientos de la aeronave. Los actuadores pueden variar dependiendo de la aeronave, pero comúnmente incluyen servomotores, actuadores hidráulicos o electrohidráulicos. Estos actuadores se encargan de mover los alerones, el timón, los alerones de profundidad y otros sistemas de control relevantes. Además, los actuadores también pueden gestionar funciones relacionadas con el motor y la potencia.

- **Interfaces de control:** Estos dispositivos permiten la interacción entre los pilotos o el sistema de gestión de vuelo y el hardware del autopiloto. Estas interfaces pueden adoptar diversas formas, como paneles de control, joysticks, teclados y pantallas de visualización. A través de estas interfaces, los pilotos pueden ingresar comandos y recibir información crítica sobre el estado y el funcionamiento del autopiloto.

El software del autopiloto es una parte integral del hardware y está diseñado para procesar los datos de los sensores, calcular las acciones necesarias y enviar las señales adecuadas a los actuadores para mantener la aeronave en una trayectoria de vuelo estable, realizar maniobras específicas o seguir una ruta de vuelo programada.

El hardware para autopiloto de una aeronave está diseñado con redundancia y sistemas de respaldo para garantizar la seguridad y la fiabilidad. Los sistemas de autopiloto modernos también están equipados con funciones avanzadas, como navegación por satélite, capacidad de aterrizaje automático y funciones de gestión de vuelo en situaciones de emergencia. A continuación se expondrán las caracteristicas principales de algunas de las placas de autopiloto del mercado:

- **Pixhawk:** Pixhawk es una plataforma de hardware de código abierto utilizada en sistemas de vehículos aéreos no tripulados (UAV) y aviones autopilotados. Proporciona una arquitectura modular y flexible que permite el control y la navegación de forma autónoma. Pixhawk es ampliamente utilizado en la industria de drones y es compatible con varios sistemas operativos y software de control de vuelo, como ArduPilot y PX4.

Figura 4.1: Placa Pixhawk

- **Veronte/Embention:** Veronte es una línea de productos desarrollada por la empresa Embention, especializada en sistemas de control de vehículos autónomos. Los sistemas Veronte están diseñados para su uso en una amplia gama de aplicaciones, incluyendo aviones no tripulados, vehículos terrestres y marinos. Proporcionan soluciones completas de control y navegación, que incluyen hardware y software personalizados.

Figura 4.2: Placa Veronte/Embention

- **Pixracer:** Pixracer es una placa de control de vuelo de tamaño compacto basada en el proyecto Pixhawk. Es una versión más pequeña y ligera de la placa original, diseñada para aplicaciones que requieren un factor de forma reducido. Pixracer ofrece capacidades similares a las de Pixhawk, pero en un paquete más pequeño y con una huella de energía reducida. Por otra parte tiene menor capacidad de expansión y opciones de conectividad en comparación con placas de mayor tamaño.

Figura 4.3: Placa Pixracer

- **APM (ArduPilot Mega):** APM es una plataforma de control de vuelo de código abierto que se utiliza en una variedad de vehículos aéreos, terrestres y acuáticos. Fue desarrollada inicialmente por la comunidad DIY (hazlo tú mismo) y ofrece funcionalidades avanzadas de piloto automático y navegación autónoma. ArduPilot Mega es una versión anterior de la plataforma que ha sido reemplazada por las versiones más recientes, como ArduPilot y PX4.

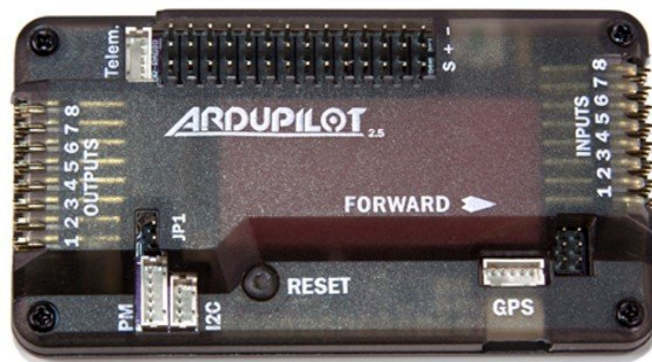

Figura 4.4: Placa ArduPilot Mega

- **Matek:** La placa de autopiloto de Matek es un componente electrónico utilizado en drones para controlar y estabilizar el vuelo de forma autónoma. Es modular, compatible con diversos sensores y sistemas de comunicación, y ofrece funciones avanzadas como control de actitud y navegación GPS. Es popular entre los entusiastas de los drones debido a su rendimiento confiable y su compatibilidad con software de control de vuelo como ArduPilot y Betaflight.

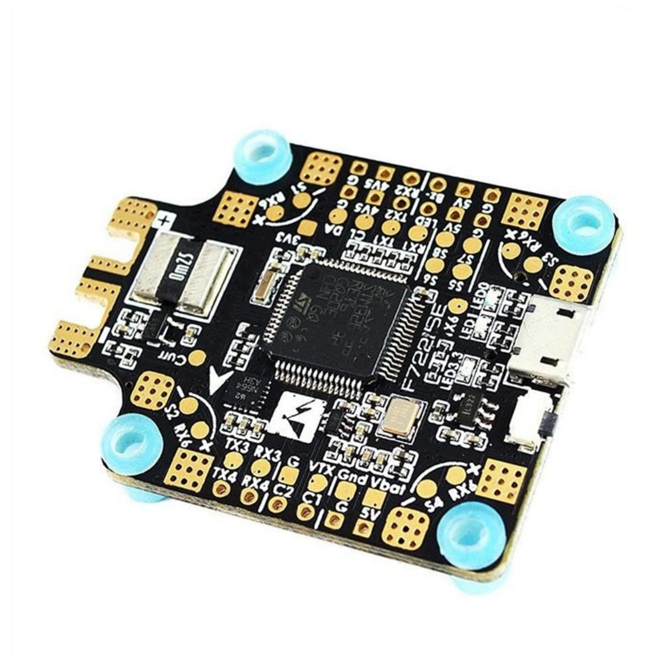

Figura 4.5: Placa Matek

◦ **CC3D (CopterControl 3D):** CC3D es una placa de control de vuelo diseñada específicamente para helicópteros y drones multirrotores. Es una plataforma de código abierto que proporciona estabilización y control de vuelo para este tipo de vehículos. CC3D ha sido ampliamente utilizado en la comunidad DIY y ofrece funcionalidades básicas de piloto automático y estabilización de vuelo.

Figura 4.6: Placa CopterControl 3D

Alternativa	**Procesador**	**IMU**	**Número de E/S**
Pixhawk	STM32F427	MPU6000	Amplio
Ve-ronte/Embention	ARM Cortex-M4	MPU9250	Amplio
Pixracer	STM32F427	MPU6000	Limitado
APM	ATMega2560	MPU6000	Amplio
Matek	STM32F405	MPU6000	Limitado
CC3D	STM32F103	MPU6000	Limitado
Alternativa	**Compatibilidad**	**Precio**	**Peso[g]**
Pixhawk	ArduPilot	Medio	38
Ve-ronte/Embention	Veronte Autopilot	Alto	198
Pixracer	ArduPilot	Bajo	5.5
APM	ArduPilot	Bajo	28
Matek	Betaflight/INAV	Bajo	4.5
CC3D	Cleanflight/OpenPilot	Bajo	7.3
Alternativa	**Nº de Puertos de Telemetría**	**Nº de IMUs**	**Dimensiones (mm)**
Pixhawk	4	3	81 x 44 x 21
Ve-ronte/Embention	2	1	76 x 40 x 64.5
Pixracer	4	1	36 x 36 x 10
APM	2	1	70 x 45 x 13
Matek	2	1	36 x 36 x 6
CC3D	2	1	36 x 36 x 12

4.2. Soluciones Alternativas Software Autopiloto: PX4, ardupilot, betaflight, librepilot, etc.

El software de autopiloto de una aeronave es una parte esencial de los sistemas de aviónica que permite el control y la automatización de las operaciones de vuelo de una aeronave. Es un conjunto de programas y algoritmos complejos diseñados para tomar decisiones basadas en datos de entrada y ejecutar acciones específicas para mantener la aeronave en una trayectoria de vuelo deseada.

El software de autopiloto interactúa con el hardware correspondiente y recopila información de una variedad de sensores, como giroscopios, acelerómetros, altímetros, sistemas de navegación global por satélite (GNSS), sistemas de referencia inercial (IRS) y otros sensores relevantes. Estos sensores proporcionan datos sobre la actitud de la aeronave (ángulos de inclinación y balanceo), la velocidad, la altitud, la posición y otros parámetros de vuelo importantes.

El software de autopiloto procesa continuamente estos datos de entrada y utiliza algoritmos y modelos matemáticos avanzados para analizar la situación y calcular las acciones necesarias para mantener el vuelo estable y seguro. Estos cálculos incluyen ajustes de control en los actuadores, como los servomotores que controlan los alerones, el timón y los alerones de profundidad, así como otras funciones del sistema, como el ajuste de la po-

tencia del motor y la gestión del combustible.

Además de mantener la aeronave en una trayectoria de vuelo deseada, el software de autopiloto también puede realizar otras funciones, como el control automático de la velocidad, la navegación precisa según rutas programadas, el seguimiento de puntos de referencia, la compensación de condiciones atmosféricas adversas y la ejecución de maniobras específicas, como despegues y aterrizajes automáticos.

El software de autopiloto está diseñado teniendo en cuenta principios de seguridad y redundancia. Por lo tanto, a menudo se implementan sistemas de respaldo y verificación cruzada para garantizar la fiabilidad del sistema. Además, se siguen estrictas normas y regulaciones de aviación para su desarrollo y certificación, con el objetivo de garantizar la seguridad de la aeronave y sus ocupantes.

En cuanto a las características de otros softwares de autopiloto, podemos mencionar:

- **PX4:** PX4 se caracteriza por su arquitectura modular y adaptable, control de vuelo robusto, soporte para diversas plataformas y controladores de vuelo, capacidades de navegación y planificación de misiones, integración de sensores variados, compatibilidad con protocolos de comunicación estándar, y su continua actualización y desarrollo como proyecto de código abierto.

Figura 4.7: Logo del software

- **ArduPilot:** ArduPilot es otro sistema de software de código abierto ampliamente utilizado en vehículos autónomos. Proporciona funcionalidades similares a PX4, incluyendo navegación autónoma, control de vuelo y seguimiento de rutas. ArduPilot tiene una comunidad activa de desarrolladores y ofrece soporte para una amplia gama de plataformas y sensores.

Figura 4.8: Logo del software

- **Betaflight:** Betaflight es un software de control de vuelo especializado en drones multirrotores de carreras. Se centra en el rendimiento y la agilidad, ofreciendo características específicas para la competición y acrobacias aéreas. Betaflight cuenta con una configuración y ajustes avanzados para optimizar el rendimiento de los drones de carreras.

Figura 4.9: Logo del software

- **LibrePilot:** LibrePilot es un proyecto de software de control de vuelo de código abierto que se centra en la estabilidad y la facilidad de uso. Ofrece una amplia gama de características, incluyendo control de vuelo, navegación y modos de vuelo autónomo. LibrePilot es compatible con diversos tipos de vehículos aéreos, terrestres y acuáticos.

Figura 4.10: Logo del software

Alternativa	Licencia	Plataformas	Características	Comunidad
PX4	BSD 3-Clause	Multiplataforma	Estabilidad, Flexibilidad	Activa
ArduPilot	GPLv3	Multiplataforma	Versatilidad, Autonomía	Muy Activa
Betaflight	GPLv3	FPV Racing Drones	Rendimiento, Sintonización	Activa
LibrePilot	GPLv3	Multiplataforma	Facilidad de Uso, Configuración	Activa

4.3. Soluciones Alternativas Software GCS: qgroundcontrol, missioncontrol, Betaflight, Librepilot, etc.

El software de GCS (Ground Control Station) de una aeronave es una herramienta de software que permite controlar y supervisar de manera remota las operaciones de vuelo de una aeronave no tripulada (UAV) desde una estación en tierra. El GCS actúa como una interfaz de usuario para el operador o piloto en tierra, brindándole información en tiempo real y la capacidad de interactuar con la aeronave de manera efectiva.

El software de GCS despliega una serie de funciones y características para facilitar el control y monitoreo de la aeronave. Algunas de las principales características son las siguientes:

- **Telemetría y visualización en tiempo real:** El GCS muestra datos en tiempo real sobre la posición, altitud, velocidad, rumbo, estado de la pila y otros parámetros relevantes de la aeronave. Esto proporciona al operador una visión actualizada del estado y el desempeño de la aeronave durante el vuelo.

- **Control de vuelo remoto:** El GCS permite al operador enviar comandos y órdenes a la aeronave de forma remota. Esto incluye acciones como el despegue, el aterrizaje, el cambio de altitud, el control de la velocidad y otras maniobras básicas de vuelo.

- **Planificación y gestión de misiones:** El software de GCS permite al operador planificar y programar misiones de vuelo de manera eficiente. Esto implica la definición de rutas de vuelo, la configuración de puntos de interés, la programación de acciones específicas y la asignación de tareas a la aeronave. Estas misiones pueden ser predefinidas o modificadas en tiempo real según sea necesario.

- **Interfaz de mapas y visualización de datos:** El GCS muestra información y mapas interactivos que permiten al operador tener una visión clara de la posición de la aeronave en relación con su entorno. También puede mostrar datos adicionales, como imágenes en tiempo real, videos, datos de sensores y análisis de vuelo.

- **Monitoreo de sensores y sistemas:** El GCS proporciona información detallada sobre el estado de los sensores y sistemas de la aeronave. Esto incluye la supervisión de la señal GPS, la calidad de la conexión de radio, el nivel de la pila, los datos del

sensor de imagen y otros parámetros relevantes. El GCS puede alertar al operador en caso de anomalías o fallas en los sistemas.

- **Registro y análisis de datos de vuelo:** El software de GCS registra y almacena datos de vuelo, incluyendo la trayectoria seguida, las acciones realizadas y otros eventos relevantes. Estos datos pueden ser utilizados posteriormente para análisis, seguimiento de rendimiento y mejora del sistema.

Exsiten varios softwares de GCS disponibles con características únicas. Algunos de ellos son:

- **QGroundControl:** QGroundControl es un software de GCS (Ground Control Station) de código abierto utilizado en sistemas de aeronaves no tripuladas (UAV). Proporciona una interfaz intuitiva para controlar y supervisar el vuelo, planificar misiones, visualizar datos en tiempo real, ajustar configuraciones de aeronaves y mostrar telemetría. Es compatible con múltiples plataformas y ofrece funciones avanzadas como seguimiento de rutas, gestión de pila y modos de vuelo autónomo.

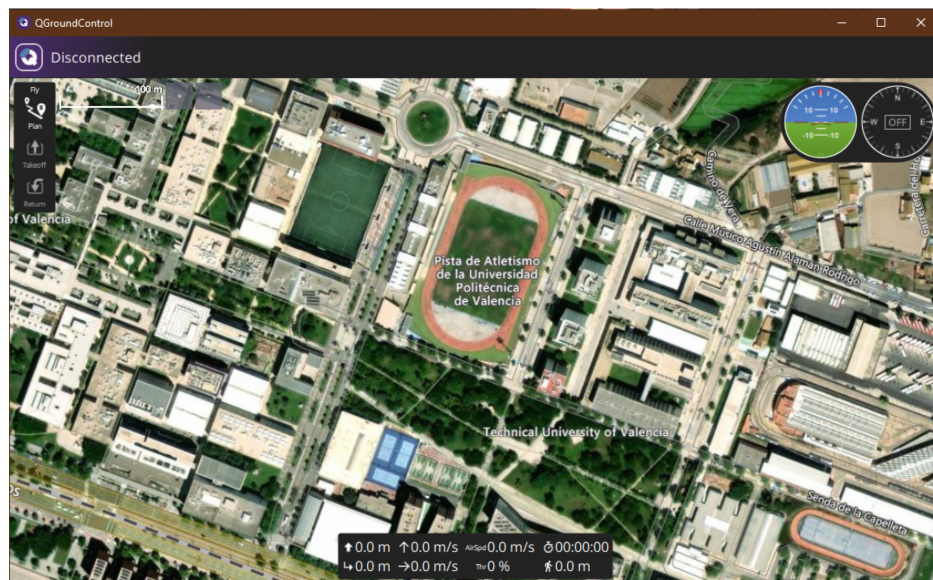

Figura 4.11: GUI del software

- **MissionControl:** MissionControl es un software de GCS desarrollado específicamente para la plataforma de control de vuelo ArduPilot. Ofrece funcionalidades similares a QGroundControl, incluyendo planificación de misiones, seguimiento en tiempo real y ajuste de parámetros.

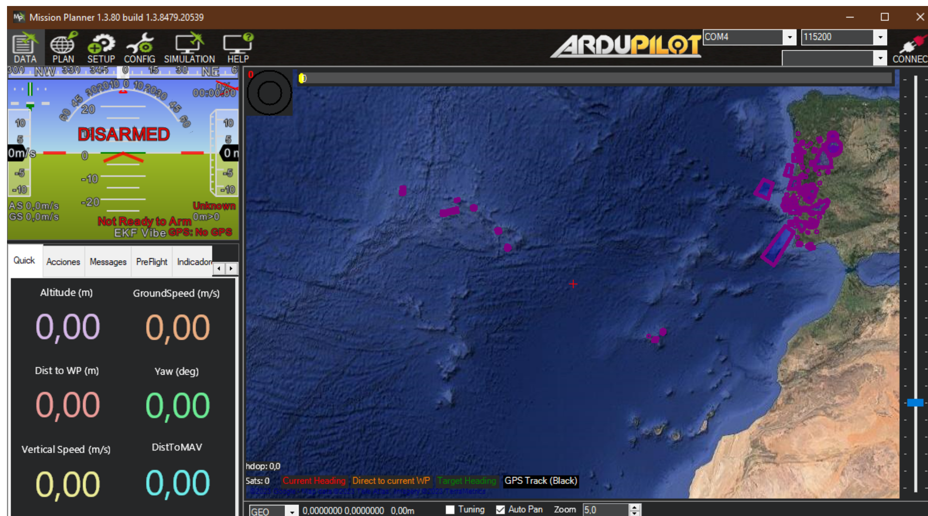

Figura 4.12: GUI del software

◦ **Betaflight:** Aunque Betaflight se mencionó anteriormente como un software de control de vuelo, también incluye una interfaz de GCS llamada Betaflight Configurator. Este software permite configurar y ajustar los parámetros de los drones multirrotores de carreras equipados con el firmware Betaflight.A diferencia de QGroundControl, que se centra en el control de UAV en general, Betaflight se enfoca específicamente en proporcionar una plataforma de configuración y ajuste de parámetros para drones de carreras, optimizando su rendimiento y maniobrabilidad.

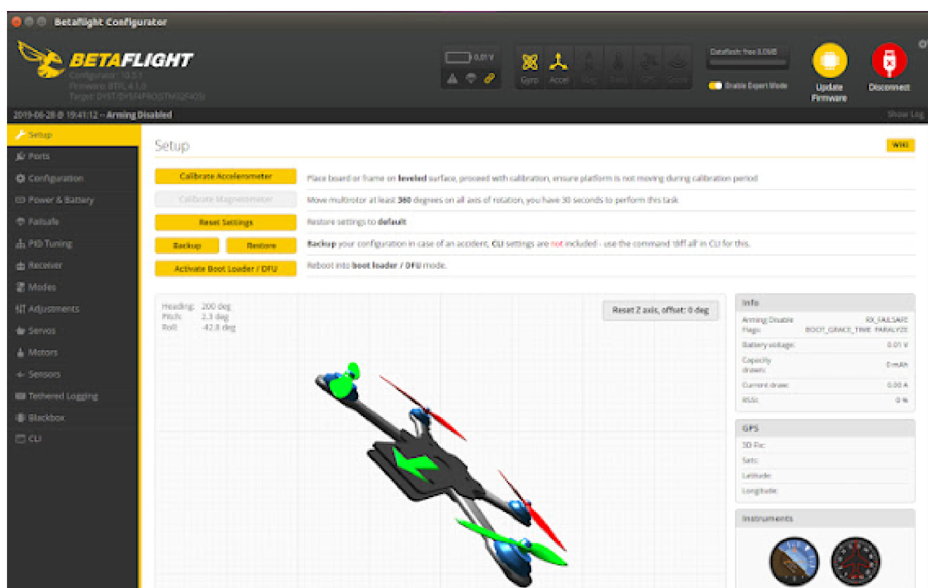

Figura 4.13: GUI del software

◦ **LibrePilot:** Aunque LibrePilot se mencionó anteriormente como un software de control de vuelo, también incluye un GCS completo. El GCS de LibrePilot proporciona funcionalidades de planificación de misiones, seguimiento en tiempo real y ajuste de parámetros para vehículos aéreos y terrestres.

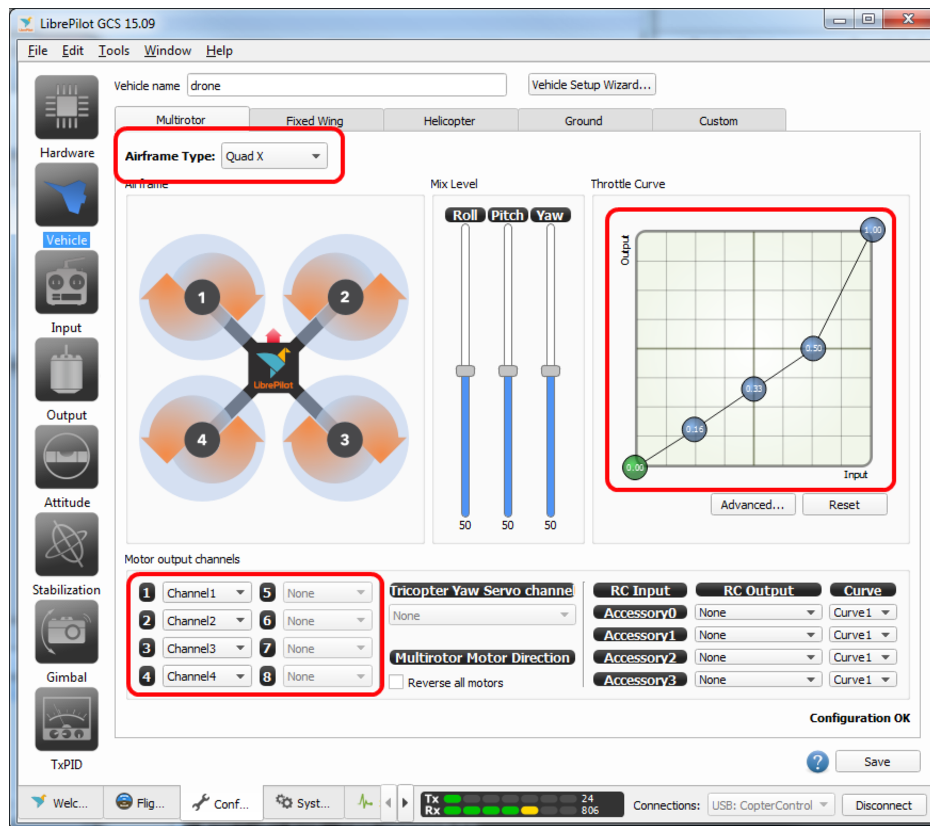

Figura 4.14: GUI del software

Alternativa	Plataformas	Interfaz Intuitiva	Características Avanzadas	Comunidad
QGroundControl	Multiplataforma	Sí	Sí	Activa
MissionControl	Multiplataforma	Sí	Sí	Activa
Betaflight	Windows, macOS	No	Limitadas	Activa
LibrePilot	Multiplataforma	Sí	Limitadas	Activa

Capítulo 5

Descripción solución adaptada

5.1. Descripción general

Para esta configuración específica, se ha tomado la decisión de utilizar la placa Pixhawk 2 como controlador de vuelo debido a sus características técnicas y capacidades avanzadas. La elección del software de autopiloto recae en el PX4, reconocido por su robustez y flexibilidad. Como Ground Control Station (GCS), se ha seleccionado QGroundControl debido a su interfaz intuitiva y su amplia gama de funcionalidades.

En términos de programación, se ha optado por utilizar un entorno de desarrollo en Linux mediante el Windows Subsystem for Linux (WSL), empleando Visual Studio Code como IDE para la programación del PX4. Por otro lado, la compilación del software QGroundControl se realizará en Windows utilizando Qt Creator.

Estas decisiones se han tomado tras evaluar los requisitos específicos del proyecto, considerando la compatibilidad, el rendimiento y la eficiencia de las herramientas seleccionadas, con el objetivo de lograr una implementación exitosa y un control de vuelo preciso y confiable.

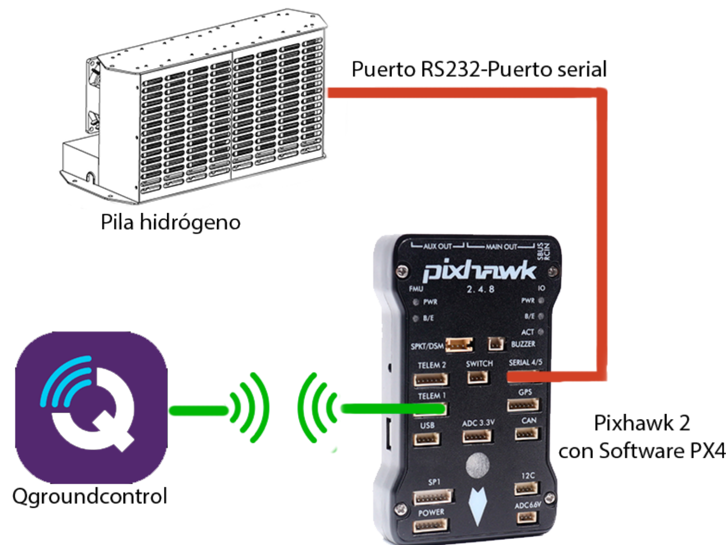

Figura 5.1: Esquema del funcionamiento

5.2. Hardware Pixhawk 2

La placa Pixhawk 2 es un hardware de control de vuelo diseñado específicamente para vehículos aéreos no tripulados (UAV) y sistemas robóticos autónomos. Es una de las opciones más populares y confiables en la industria de los UAV debido a sus características técnicas avanzadas y su robustez.

Características técnicas del hardware Pixhawk 2:

- **Arquitectura modular:** La placa Pixhawk 2 está diseñada con una arquitectura modular, lo que significa que se compone de varios módulos interconectados que se pueden adaptar y personalizar según las necesidades específicas de la aeronave o aplicación. Esto proporciona flexibilidad y escalabilidad al sistema, permitiendo la adición de módulos adicionales según sea necesario.

- **Procesador de alto rendimiento:** La Pixhawk 2 está equipada con un potente procesador de alto rendimiento que es capaz de manejar de manera eficiente las tareas de control de vuelo y procesamiento de datos. Esto permite un cálculo rápido de los algoritmos de control y una respuesta precisa a las entradas de los sensores y comandos del piloto.

- **Conectividad amplia:** La placa Pixhawk 2 ofrece una amplia gama de opciones de conectividad para interactuar con otros dispositivos y periféricos. Esto incluye

puertos UART, I2C, SPI y CAN, que permiten la conexión de sensores, sistemas de comunicación, módulos GPS y otros componentes externos necesarios para el funcionamiento de la aeronave.

- **Soporte para múltiples sensores:** La Pixhawk 2 es compatible con una amplia variedad de sensores, como acelerómetros, giroscopios, magnetómetros, barómetros y sistemas de posicionamiento global (GPS). Estos sensores proporcionan información vital sobre la actitud, posición, altitud y velocidad de la aeronave, permitiendo un control preciso y una navegación confiable.

- **Redundancia y seguridad:** La placa Pixhawk 2 está diseñada con características de redundancia para garantizar la seguridad y la integridad del vuelo. Esto incluye la incorporación de múltiples sensores y sistemas de control, permitiendo una conmutación automática en caso de fallo de un componente. Esta redundancia ayuda a prevenir errores críticos y aumenta la confiabilidad del sistema.

- **Amplia compatibilidad de software:** La Pixhawk 2 es compatible con una variedad de software de control de vuelo, incluyendo PX4 y ArduPilot, que son sistemas de control de vuelo populares y de código abierto. Esto brinda a los usuarios una amplia selección de opciones de software y la capacidad de personalizar y ajustar el comportamiento de la aeronave según sus necesidades específicas.

Cuadro 5.1: Pixhawk 2 - Características y Especificaciones

Características principales	**Especificaciones técnicas**
Procesador	32 bit ARM Cortex M4
Sistema Operativo	NuttX RTOS
Salidas PWM/Servo	14 (8 con tolerancia a fallos y anulación manual, 6 auxiliares)
Conectividad	UART, I2C, CAN
Alimentación	Entradas redundantes conmutación automática
Pulsador de seguridad externo	Sí
Indicador visual LED	Multicolor
Indicador de audio	Piezo de alta potencia y multitono
Tarjeta microSD	Sí
Aplicaciones	Navegación, investigación, control y sistemas estabilizados
Core	32 bit STM32F427 Cortex M4 con FPU
Frecuencia	168 MHz
RAM	256 KB
Memoria	2 MB Flash
Almacenamiento	Ranura para tarjeta microSD
Dimensiones (LxWxH)	81.5x50x15.5 mm
Peso	38 g
Interfaces	5x UART, 2x CAN, Spektrum DSM/DSM2/DSM-X compatible, Futaba S.BUS compatible, PPM sum signal, I2C, SPI, 3.3V y 6.6V entradas ADC, microUSB externo
Failsafe co-processor	32 bit STM32F103
Giroscopio	ST Micro L3GD20H 16 bit
Acelerómetro/Magnetómetro	ST Micro LSM303D 14 bit
Barómetro	MEAS MS5611

5.3. Software PX4

PX4 es un software de código abierto ampliamente utilizado para sistemas de control de vuelo y autopilotos en aeronaves no tripuladas (UAV) y vehículos aéreos de ala fija y multirrotores. Algunas de las principales características del software de autopiloto PX4 son las siguientes:

- **Arquitectura modular:** PX4 está diseñado con una arquitectura modular, lo que significa que se puede adaptar y personalizar para adaptarse a diferentes plataformas y requisitos específicos. Esto permite a los desarrolladores e integradores agregar y quitar módulos según sea necesario para satisfacer las necesidades de su aplicación particular.

- **Control de vuelo robusto:** El software de autopiloto PX4 proporciona algoritmos

de control de vuelo robustos y avanzados que permiten un vuelo estable y preciso. Estos algoritmos incluyen controladores PID (Proporcional, Integral y Derivativo), control de actitud y control de posición, entre otros.

- **Soporte para múltiples plataformas:** PX4 es compatible con una amplia gama de plataformas y controladores de vuelo, lo que lo hace altamente versátil. Puede ser utilizado en una variedad de vehículos aéreos, como drones de ala fija y multirrotores, así como en otros sistemas robóticos.
- **Navegación y planificación de misiones:** El software de autopiloto PX4 ofrece capacidades de navegación y planificación de misiones. Permite la definición de rutas de vuelo predefinidas, el seguimiento de puntos de referencia, la navegación GPS y la planificación de misiones autónomas.
- **Integración de sensores:** PX4 es compatible con una amplia gama de sensores, incluyendo giroscopios, acelerómetros, magnetómetros, sistemas de posicionamiento global (GPS), sensores de flujo óptico y muchos otros. Esto permite una integración flexible y precisa de los sensores necesarios para el control de vuelo y las aplicaciones específicas.
- **Compatibilidad con protocolos de comunicación estándar:** El software PX4 utiliza protocolos de comunicación estándar como MAVLink para la comunicación con estaciones terrestres y sistemas de control en tierra. Esto facilita la integración y la comunicación con otros componentes del sistema.
- **Actualizaciones y desarrollo activo:** PX4 es un proyecto de código abierto con una comunidad de desarrollo activa. Esto significa que se están realizando constantemente mejoras, actualizaciones y nuevas características en el software, lo que garantiza que esté en constante evolución y mejora.

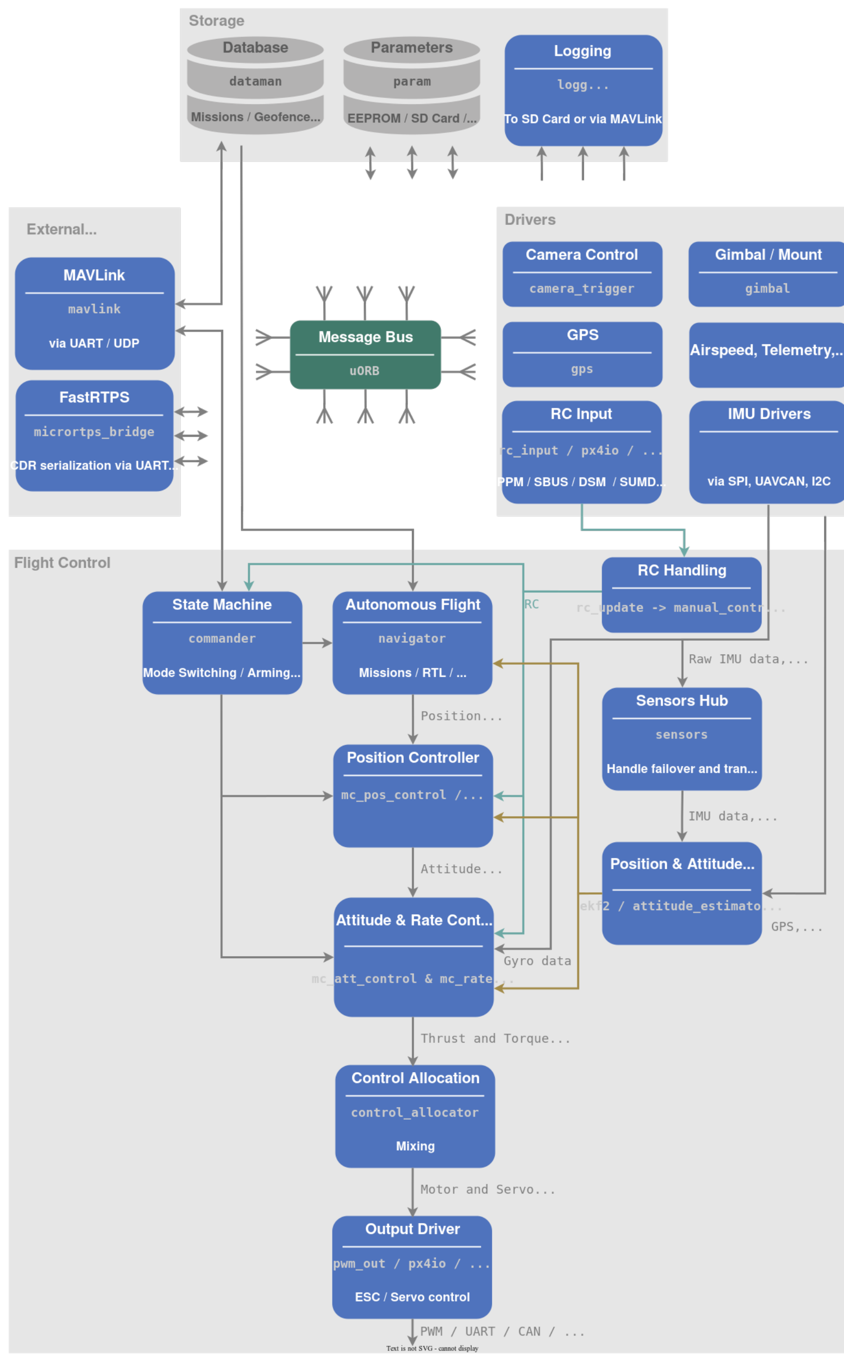

Figura 5.2: Arquitectura del software PX4

5.3.1. Programación de apps en el entorno de PX4

Conforme se ha mencionado previamente, para el desarrollo de aplicaciones en el entorno de programación PX4 se empleará el entorno de desarrollo integrado (IDE) Visual Studio Code en conjunto con Linux a través de WSL, (Windows Subsystem for Linux) es una funcionalidad de Windows que permite ejecutar distribuciones de Linux en entornos Windows sin necesidad de una máquina virtual. Proporciona una capa de compatibilidad para correr aplicaciones y herramientas de Linux en Windows, facilitando el desarrollo y la ejecución de aplicaciones Linux en entornos basados en Windows. Esta elección se basa en la disponibilidad de una extensión específica para dicho IDE que permite tal configuración.

El sistema operativo utilizado por PX4 es un sistema operativo en tiempo real basado en NuttX, que es un sistema operativo de código abierto diseñado para sistemas embebidos. PX4 se ejecuta en una amplia gama de hardware, desde controladores de vuelo de bajo costo hasta vehículos aéreos no tripulados (UAV) más complejos.

PX4 funciona utilizando un enfoque modular, donde diferentes componentes o módulos se comunican entre sí para realizar tareas específicas. Estos módulos incluyen el Controlador de Vuelo (Flight Controller), el Administrador de Tareas (Task Manager), el Sistema de Posicionamiento Global (GPS), el Sistema de Estabilización (Stabilization), entre otros. Cada módulo se encarga de una función específica y se comunica con otros módulos a través de interfaces bien definidas.

La integración de nuevas aplicaciones en PX4 se realiza a través del uso de módulos adicionales o mediante la creación de módulos personalizados. PX4 proporciona una arquitectura abierta y extensible que permite a los desarrolladores agregar nuevas funcionalidades y características al sistema. Las nuevas aplicaciones pueden ser desarrolladas en lenguajes como C++ y se integran con PX4 utilizando APIs y interfaces de comunicación definidas.

Es importante destacar que este software presenta características particulares, entre las cuales se destaca la posibilidad de escribir las aplicaciones tanto en C como en C++, y además cuenta con funcionalidades programadas en Python, entre otros. Para nuestro caso particular, nos centramos en programar en C. Asimismo, para que este software pueda compilar y ejecutar correctamente las aplicaciones, es necesario contar con dos archivos adicionales aparte de la propia aplicación: un archivo CMakeLists y un archivo Kconfig.

El archivo CMakeLists es un archivo de configuración esencial que se utiliza en el proceso de compilación para especificar los detalles del proyecto, como las fuentes de código,

las bibliotecas externas necesarias y las opciones de compilación. Este archivo permite al sistema de construcción, en este caso CMake, comprender cómo construir el proyecto y generar los archivos binarios resultantes.

Por otro lado, el archivo Kconfig se emplea para definir la configuración del kernel de PX4. Contiene las opciones de configuración disponibles y sus valores predeterminados, lo que permite personalizar la construcción del firmware según los requisitos específicos del proyecto.

Es importante destacar que esta aplicación se integrará en el sistema operativo de la placa, específicamente en el inicio de la ejecución (start up), lo que significa que los mensajes de depuración como «PX4_INFO» o «printf» no serán visualizados en la consola. Sin embargo, estos mensajes son útiles para facilitar la tarea de depuración durante el desarrollo y prueba del código.

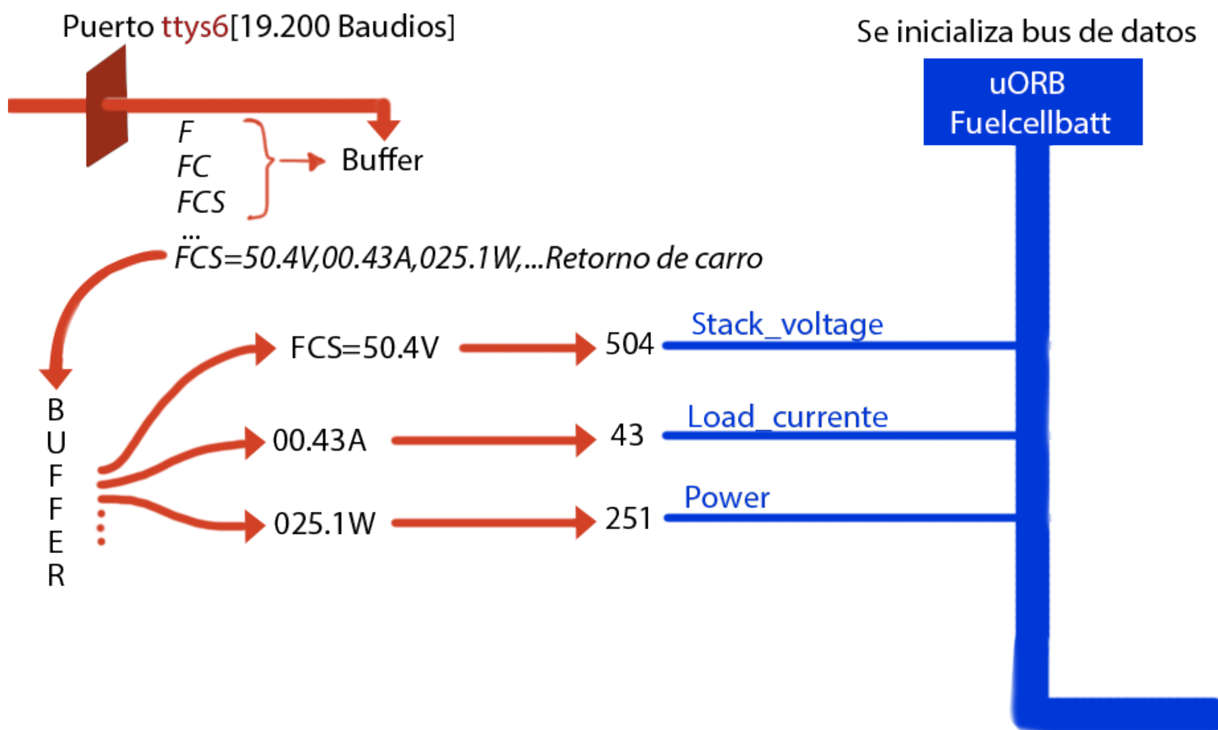

Figura 5.3: Esquema del Programa

El nuevo modulo diseñado tiene como objetivo principal abrir el puerto *ttys6* de la placa a una velocidad de transferencia de datos de 19200 baudios. Este puerto es utiliza-

do para recibir información proveniente de la pila de hidrógeno. Además, se inicializa un mensaje uORB, que es un sistema de comunicación entre procesos utilizado en el entorno PX4, con el fin de enviar la información recolectada posteriormente. A continuación, se entra en un bucle infinito (while true) donde se lee el puerto serial y se almacena cada lectura en una cadena de caracteres llamada *buffer*. Esta cadena se procesa y se reinicia cada vez que se recibe un carácter de retorno de carro.

En este punto es donde comienza la interpretación del mensaje recibido. Si la longitud de la cadena *buffer* es menor a 100 caracteres, se considera un mensaje de inicialización y no se realiza ningún procesamiento adicional. Por otro lado, si la longitud supera los 100 caracteres, se trata de un mensaje de estado de la pila y se procede a separar los valores que están intercalados por comas. Se extraen únicamente los números de cada mensaje, ya que son los datos relevantes para nuestro análisis. Estos valores numéricos se asocian a variables específicas según su posición en el mensaje. Por ejemplo, en el caso del mensaje «FCS=50.4V,00.43A,025.1W,...», se tomaría el valor «FCS=50.4V» y se asignaría a la variable *stack_ voltage*, luego se procesaría «00.43A» y se asociaría con la variable *load_ currente*. Este proceso se repite para cada variable y mensaje correspondiente.

A continuación se presentarán los códigos:

C:

```c
#define SERIAL_PORT "/dev/ttyS6"
#define BUFFER_SIZE 145

__EXPORT int prueba_main(int argc, char *argv[]);

int prueba_main(int argc, char *argv[])
{

    PX4_INFO("Iniciando puerto...");

    int serialPort;
    struct termios serialSettings;
    char buffer[BUFFER_SIZE];
    int buffIndex = 0;

    // Abrir el puerto serie en modo lectura
    serialPort = open(SERIAL_PORT, O_RDONLY);
    if (serialPort == -1) {
```

```
        perror("Error al abrir el puerto serie");
        return 1;
    }

    // Configurar la velocidad y otros parámetros del puerto serie
    tcgetattr(serialPort, &serialSettings);
    cfsetispeed(&serialSettings, B19200);  // Velocidad de baudios
    serialSettings.c_cflag &= ~PARENB;    // Sin paridad
    serialSettings.c_cflag &= ~CSTOPB;    // 1 bit de parada
    serialSettings.c_cflag &= ~CSIZE;     // Limpiar bits de tamaño de carácter
    serialSettings.c_cflag |= CS8;        // 8 bits de datos
    tcsetattr(serialPort, TCSANOW, &serialSettings);

    /* advertise attitude topic */
  struct fuelcellbatt_s att;
  memset(&att, 0, sizeof(att));
  orb_advert_t att_pub = orb_advertise(ORB_ID(fuelcellbatt), &att);

    // Leer del puerto serie y almacenar en el vector buff
    while (1) {

        char receivedChar;
        ssize_t bytesRead = read(serialPort, &receivedChar, sizeof(receivedChar));
        if (bytesRead > 0) {

            if (receivedChar == '\n') {
                buffer[buffIndex] = '\0';  // Agregar terminador de cadena
                printf("Lectura: %s\n", buffer);
                if (buffIndex >= 100) { //este if es lo nuevo borrar si deja de funcionar

                    //buffer[buffIndex] = '\0';  // Agregar terminador de cadena
```

```
50                    //printf("Lectura (tamaño máximo
                         alcanzado): %s\n", buffer);
51
52                    char datos[55];
53                    int contador = 0;
54                    int caso=1;
55                    for (int j = 0; j < (buffIndex+2); j++) {
56                        if (isdigit(buffer[j])) {
57                            datos[contador++] = buffer[j];
58                        }
59                        if(buffer[j] == ','){
60                            datos[contador] = '\0';
61                            switch (caso) {
62                                case 1:
63                                    printf("%s,",datos);
64                                    att.stack_voltage=atof(
                                         datos);
65                                    break;
66                                case 2:
67                                    printf("%s,",datos);
68                                    att.load_currente=atof(
                                         datos);
69                                    break;
70                                case 3:
71                                    printf("%s,",datos);
72                                    att.power=atof(datos);
73                                    break;
74                                case 4:
75                                    printf("%s,",datos);
76                                    att.energy=atof(datos);
77                                    break;
78                                case 5:
79                                    printf("%s,",datos);
80                                    att.battery_voltage=atof(
                                         datos);
81                                    break;
82                                case 6:
83                                    printf("%s,",datos);
84                                    att.battery_current=atof(
```

```
                                    datos);
85                                if(buffer[j-5]=='-'){
86                                    att.signo=1;
87                                }
88                                else{
89                                    att.signo=0;
90                                }
91                                break;
92                            case 7:
93                                printf("%s,",datos);
94                                att.load_voltage=atof(
                                    datos);
95                                break;
96                            case 8:
97                                printf("%s,",datos);
98                                att.temp1=atof(datos);
99                                int temp1 = atof(datos);
100                               break;
101                           case 9:
102                               printf("%s,",datos);
103                               att.temp2=atof(datos);
104                               int temp2 = atof(datos);
105                               break;
106                           case 10:
107                               printf("%s,",datos);
108                               att.temp3=atof(datos);
109                               int temp3 = atof(datos);
110                               break;
111                           case 11:
112                               printf("%s,",datos);
113                               att.temp4=atof(datos);
114                               int temp4 = atof(datos);
115                               int max = fmax(fmax(temp1
                                    , temp2), fmax(temp3,
                                    temp4));
116                               att.top_temp=max;
117                               break;
118                           case 12:
119                               printf("%s,",datos);
```

```
120                                         att.target_temp=atof(
                                               datos);
121                                         break;
122                                     case 13:
123                                         printf("%s,",datos);
124                                         att.board_temp=atof(datos
                                               );
125                                         break;
126                                     case 14:
127                                         printf("%s,",datos);
128                                         att.pila=atof(datos);
129                                         break;
130                                     case 15:
131                                         printf("%s,",datos);
132                                         att.h2_pressure=atof(
                                               datos);
133                                         break;
134                                     case 16:
135                                         printf("%s,",datos);
136                                         att.fan_speed=atof(datos)
                                               ;
137                                         break;
138                                     case 17:
139                                         printf("%s,\n",datos);
140                                         att.state=atof(datos);
141                                         break;
142                                     default:
143                                         printf("Opción␣inválida\n
                                               ");
144                                 }
145                                 caso++;
146                                 memset(datos, 0, 55);
147                                 contador=0;
148                             }
149                         }
150                         orb_publish(ORB_ID(fuelcellbatt), att_pub
                               , &att);
151                     }
152                     buffIndex = 0;  // Reiniciar el índice del
```

```
                        vector buff
153             }
154             else {
155                 buffer[buffIndex] = receivedChar;
156                 buffIndex++;
157             }
158         }
159     }
160 
161     PX4_INFO("Fuera␣del␣while");
162     // Cerrar el puerto serie
163     close(serialPort);
164     return 0;
165 }
```

Kconfig:

```
1 menuconfig EXAMPLES_PRUEBA
2   bool "prueba"
3   default n
4   ---help---
5     Enable support for prueba
```

CMakeLists:

```
1 px4_add_module(
2   MODULE examples__prueba
3   MAIN prueba
4   SRCS
5     prueba.c
6   DEPENDS
7   )
```

5.3.2. Declarar nuevos mensajes en el bus uORB

El uORB (micro Object Request Broker) es un sistema de mensajería y publicación-suscripción utilizado en el marco del proyecto PX4, que es una plataforma de código abierto para vehículos aéreos no tripulados (UAVs). El uORB proporciona un mecanismo eficiente para que los diferentes módulos y componentes del sistema se comuniquen entre sí a través de mensajes.

En uORB, los mensajes se definen mediante archivos de descripción (.msg) que especifican la estructura y el tipo de datos de cada mensaje. Estos mensajes pueden contener información sobre el estado del vehículo, datos de sensores, comandos de control, entre otros. Los módulos del sistema pueden publicar mensajes en un determinado tema (topic) y otros módulos pueden suscribirse a esos temas para recibir las actualizaciones de los mensajes.

El uORB gestiona la comunicación de mensajes de forma eficiente y desacoplada, lo que permite una comunicación fluida entre los diferentes componentes del sistema PX4. Además, proporciona una interfaz sencilla para el desarrollo de software, ya que los mensajes se generan automáticamente y se accede a ellos mediante una API simple.

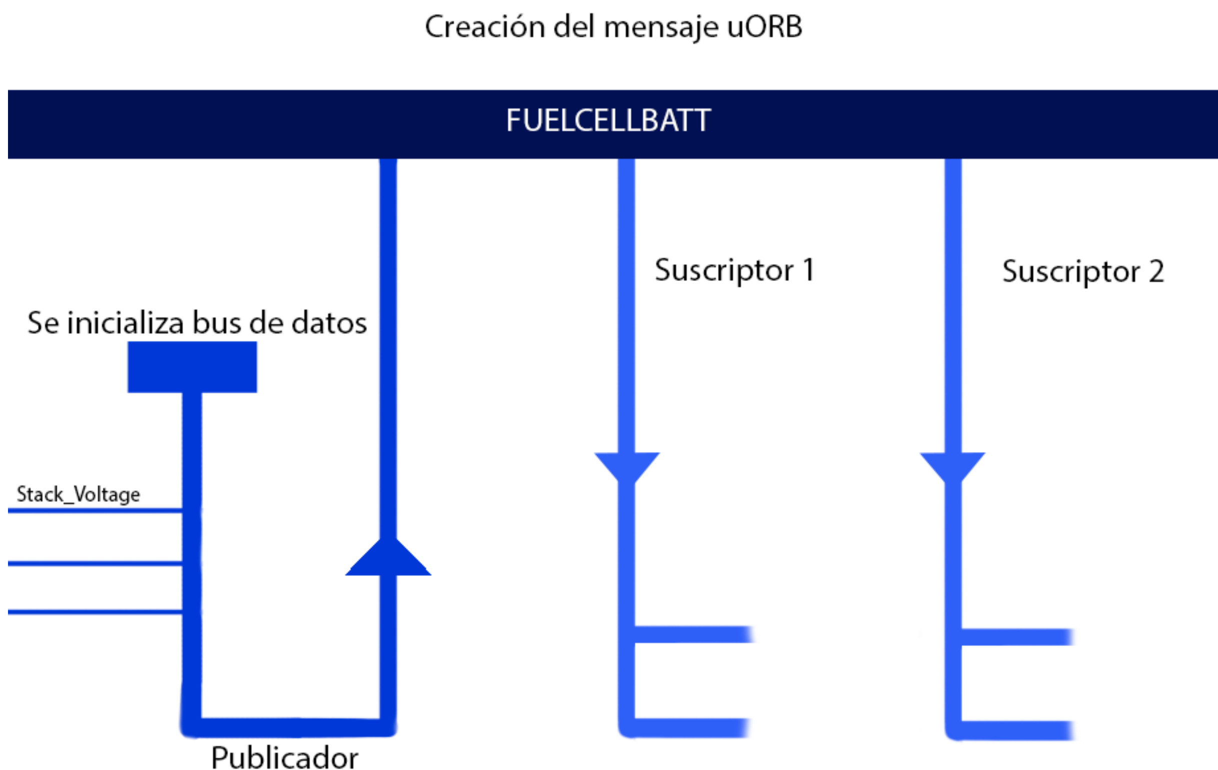

Figura 5.4: Esquema del funcionamiento del uORB

Para crear un nuevo tema (topic) para los mensajes uORB en este caso, se debe crear un nuevo archivo del tipo *.msg* en la carpeta correspondiente y añadir el nombre del mensaje creado al CMakeLists. En nuestro caso, el mensaje se llama «fuelcellbatt» y tiene la siguiente estructura:

```
uint64 timestamp            #tiempo desde el inicio del sistema (microsegundos)
uint64 timestamp_sample     #el tiempo del dato sin procesar (microsegundos)
uint16 pila                 #info--
uint16 stack_voltage        #info1--
uint16 load_currente        #info2--
uint16 power                #info3--
uint16 energy               #info4--
uint16 battery_voltage      #info5--
uint16 battery_current      #info6--
uint16 load_voltage         #info7--
uint16 temp1                #info8--
uint16 temp2                #info9--
uint16 temp3                #info10--
uint16 temp4                #info11--
uint16 target_temp          #info12
uint16 board_temp           #info13--
uint16 h2_pressure          #info14--
uint16 fan_speed            #info15--
uint16 state                #info16
uint16 signo

# TOPICS fuelcellbatt
```

La inclusión de la última línea en el programa es fundamental, ya que establece el mecanismo mediante el cual los demás archivos pueden acceder a este mensaje. En cada programa que haga uso del mensaje, se debe incorporar la directiva `#include <uORB/topics/fuelcellbatt.h>`. Cabe destacar que este archivo es de tipo .h, lo que indica que se trata de un encabezado (header) generado automáticamente durante la compilación a partir del mensaje .msg correspondiente. A partir de este punto, el mensaje puede ser utilizado en cualquier programa para suscribirse a este tema y así visualizar las actualizaciones de las variables o bien, leer este flujo de datos.

5.3.3. Mavlink: concepto y necesidad

MAVLink (Micro Air Vehicle Link) es un protocolo de comunicación ligero y de código abierto diseñado específicamente para la comunicación entre vehículos aéreos no tripulados (UAVs) y sistemas de control en tierra, como estaciones de control de misión o estaciones de control de vuelo. Fue desarrollado por la comunidad de código abierto de drones, lide-

rada por la organización sin fines de lucro Dronecode Foundation.

El propósito principal de MAVLink es facilitar la transmisión de datos y comandos entre los componentes de un sistema de vehículo aéreo no tripulado. Proporciona una estructura flexible y modular que permite la interoperabilidad entre diferentes plataformas de UAV y sistemas de control en tierra. A través de este protocolo, los datos se transmiten de manera eficiente y confiable, lo que resulta fundamental para el control de vuelo, la planificación de misiones y la recopilación de datos en tiempo real.

MAVLink se basa en un modelo de mensajes, donde los datos se envían en forma de mensajes entre los distintos componentes del sistema. Estos mensajes pueden contener información sobre la telemetría del vehículo, como la posición, la velocidad, el estado de la pila y los sensores a bordo. También se pueden enviar comandos de control de vuelo, como cambios de modo de vuelo, ajustes de parámetros y solicitudes de acción.

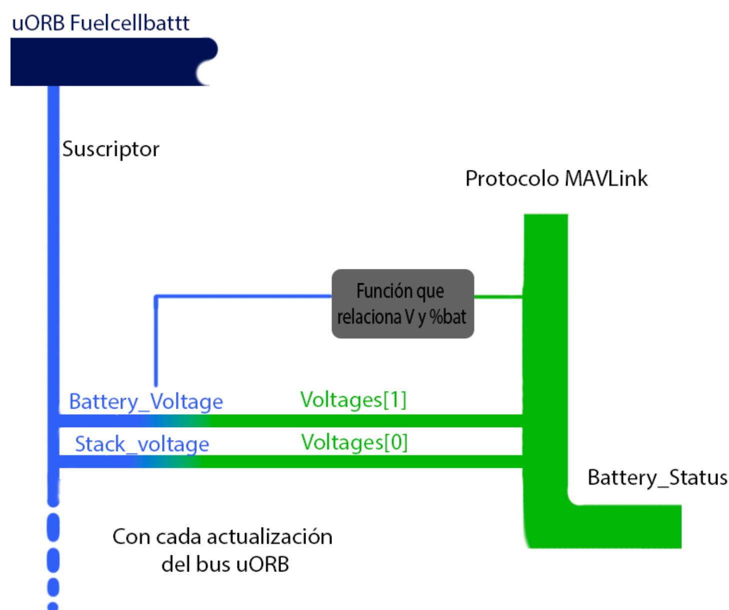

Figura 5.5: Esquema de MAVLink

La ventaja de utilizar MAVLink es su capacidad para adaptarse a diferentes tipos de sistemas de vehículos aéreos no tripulados y sistemas de control en tierra. Puede ser implementado en sistemas de código abierto o propietarios, lo que lo convierte en un estándar ampliamente adoptado en la industria de drones.

En este contexto, se empleará el mensaje Mavlink *Battery_ status*, el cual está predefinido en el propio sistema. Se utilizarán las variables incluidas en este mensaje para representar los datos del mensaje de la pila de hidrógeno. Aunque los nombres de las

variables no reflejen directamente su significado, se implementará un widget más adelante que mostrará la correspondencia entre cada número y su respectiva variable, así como las unidades asociadas. Además, se calcularán nuevas características basadas en los datos obtenidos.

A continuación, se presenta la edición realizada en el mensaje para enviarlo mediante el protocolo Mavlink:

```
#ifndef BATTERY_STATUS_HPP
#define BATTERY_STATUS_HPP


#include <uORB/topics/fuelcellbatt.h>


class MavlinkStreamBatteryStatus : public MavlinkStream
{
public:
  static MavlinkStream *new_instance(Mavlink *mavlink) {
     return new MavlinkStreamBatteryStatus(mavlink); }

  static constexpr const char *get_name_static() { return "
     BATTERY_STATUS"; }
  static constexpr uint16_t get_id_static() { return
     MAVLINK_MSG_ID_BATTERY_STATUS; }

  const char *get_name() const override { return
     get_name_static(); }
  uint16_t get_id() override { return get_id_static(); }

  unsigned get_size() override
  {
    return MAVLINK_MSG_ID_BATTERY_STATUS_LEN +
       MAVLINK_NUM_NON_PAYLOAD_BYTES;
  }

private:
  explicit MavlinkStreamBatteryStatus(Mavlink *mavlink) :
     MavlinkStream(mavlink) {}

```

```
27  uORB::Subscription _fuelcellbatt_sub{ORB_ID(fuelcellbatt)};
28
29  bool send() override
30  {
31    mavlink_battery_status_t bat_msg{};
32    bool updated = false;
33    if(_fuelcellbatt_sub.advertised()){
34
35      fuelcellbatt_s fuelcellbatt{};
36      if(_fuelcellbatt_sub.copy(&fuelcellbatt)){
37
38        bat_msg.voltages_ext[0]=fuelcellbatt.pila;
39        bat_msg.voltages_ext[1]=fuelcellbatt.h2_pressure;
40        bat_msg.voltages_ext[2]=fuelcellbatt.fan_speed;
41        bat_msg.mode=fuelcellbatt.state;
42        bat_msg.temperature=fuelcellbatt.top_temp;
43        bat_msg.voltages[0]=fuelcellbatt.stack_voltage;
44        bat_msg.voltages[1]=fuelcellbatt.battery_voltage;
45        bat_msg.voltages[2]=fuelcellbatt.load_voltage;
46        bat_msg.voltages[3]=fuelcellbatt.temp1;
47        bat_msg.voltages[4]=fuelcellbatt.temp2;
48        bat_msg.voltages[5]=fuelcellbatt.temp3;
49        bat_msg.voltages[6]=fuelcellbatt.temp4;
50        bat_msg.voltages[7]=fuelcellbatt.board_temp;
51        bat_msg.voltages[8]=fuelcellbatt.target_temp;
52        bat_msg.current_battery=fuelcellbatt.load_currente;
53        bat_msg.time_remaining=fuelcellbatt.battery_current;
54        bat_msg.current_consumed =fuelcellbatt.power;
55                    bat_msg.energy_consumed = fuelcellbatt.
              energy;
56        double mid=100*(-503.622+584.100*fuelcellbatt.
           battery_voltage/80.0-          252.554*pow(
           fuelcellbatt.battery_voltage/80.0,2)+48.204*pow(
           fuelcellbatt.battery_voltage/80.0,3)-3.422*pow(
           fuelcellbatt.battery_voltage/80.0,4));
57
58        if (mid<100){
59          if(mid<0){
60            bat_msg.battery_remaining=0;
```

```
          }
          bat_msg.battery_remaining=mid;
        }
        else{
          bat_msg.battery_remaining=100;
        }
        bat_msg.type = fuelcellbatt.signo;
                    updated = true;
      }
    }
    if (updated) {
      mavlink_msg_battery_status_send_struct(_mavlink->
         get_channel(), &bat_msg);
      return true;
    }
    return false;
  }
};
#endif // BATTERY_STATUS_HPP
```

5.4. Software Qgroundcontrol

Para el software del GCS (Ground Control Station) de este proyecto, se ha elegido QGroundControl debido a sus características avanzadas y su amplia aceptación en la comunidad de vehículos autónomos. QGroundControl es un sistema de software de código abierto diseñado para controlar y monitorear vehículos aéreos no tripulados (UAV) y sistemas robóticos terrestres.

Una de las ventajas principales de QGroundControl es su interfaz gráfica de usuario intuitiva y fácil de usar. Proporciona una representación visual en tiempo real de los datos del vehículo, como la posición, la altitud, la velocidad y la telemetría. Esto permite a los operadores supervisar y controlar de manera efectiva las operaciones del UAV.

QGroundControl ofrece una amplia gama de características y funcionalidades para la planificación y ejecución de misiones. Permite la creación y edición de rutas de vuelo, establecimiento de puntos de referencia y tareas específicas, como la captura de imágenes o la recopilación de datos. También ofrece la posibilidad de configurar modos de vuelo autónomos y personalizar el comportamiento del UAV en diferentes situaciones.

QGroundControl proporciona una visión completa de la telemetría de vuelo, lo que permite a los operadores supervisar y analizar datos cruciales durante la operación del UAV en tiempo real. Esto incluye información sobre la altitud, velocidad, posición GPS, estado de la pila y otros parámetros importantes. La capacidad de monitoreo en tiempo real facilita la toma de decisiones y el control preciso del UAV.

En cuanto a la configuración, QGroundControl permite una configuración avanzada y detallada de los UAV. Los operadores pueden calibrar los sensores, configurar modos de vuelo y asignar canales de radio de manera personalizada. Esto brinda la flexibilidad necesaria para adaptar el sistema a requisitos específicos y asegurar un rendimiento óptimo del UAV.

El software admite la visualización de imágenes y vídeos en tiempo real desde la cámara a bordo del UAV, lo que permite a los operadores tener una perspectiva visual completa durante las misiones. Además, QGroundControl proporciona herramientas de análisis de datos y registro de vuelo para revisar y evaluar el rendimiento de la misión.

Otras características notables de QGroundControl incluyen:

- Conexión y comunicación con el UAV a través de diferentes protocolos de comunicación, como MAVLink, UDP, TCP/IP y más.
- Capacidades de seguimiento y geolocalización utilizando sistemas de posicionamiento global (GPS).
- Soporte para la configuración y ajuste de parámetros del UAV, como los límites de vuelo, la sensibilidad de los controles y los modos de operación.
- Compatibilidad con una amplia gama de UAV y autopilotos, incluyendo PX4, ArduPilot y otros sistemas populares.

Tambien cabe destacar que QGroundControl es un proyecto de código abierto, puedes acceder a su código fuente y editarlo según tus necesidades. Esto te brinda la capacidad de personalizar y adaptar la estación de control de vuelo de acuerdo con tus requisitos específicos. A continuación, mencionaré algunas formas en las que puedes personalizar QGroundControl:

- **Interfaz de usuario (UI):**Puedes personalizar la interfaz de usuario para que se ajuste a tus preferencias o para adaptarla a requisitos específicos de uso. Esto incluye cambios en el diseño, la organización de los paneles, los iconos y la disposición de los elementos.

◦ **Traducciones:**Si deseas utilizar QGroundControl en un idioma distinto al inglés, puedes contribuir con traducciones para que esté disponible en otros idiomas. Esto permite que más personas utilicen y se beneficien del software.

◦ **Añadir funcionalidades:**Si hay características específicas que necesitas y no están disponibles en la versión estándar de QGroundControl, puedes desarrollar y añadir tus propias funcionalidades al código fuente. Esto te permite adaptar la estación de control a tus necesidades particulares o incluso contribuir con nuevas características para beneficio de la comunidad.

◦ **Integración de hardware personalizado:**QGroundControl es compatible con una amplia gama de hardware, pero si deseas integrar un sistema o dispositivo personalizado, puedes modificar el código fuente para que sea compatible con tu hardware específico. Esto permite una mayor flexibilidad y adaptabilidad a diferentes configuraciones de vehículos.

Operating States			
Name	Value	Occurs	Description
NO_STATE	0	Startup	Processor starting up
INPUTS_READY	1	Startup	Sensor Check
LOOP_READY	2	Startup	Sensor Check
START_HGS	3	Startup	Check Supply Pressure
CHECKING_STARTUP	4	Startup	Checking Battery Voltage
CLEAN_STACK	5	Startup	Open Supply Solenoid, and check
OPERATION	6	Operation	Fuel Cell Output On, and operating normally
CONDITIONING	7	Operation	Displayed during conditioning
SHUTDOWN	8	Shutdown	Shutting down the fuel cell
OFF	9	Shutdown	Fuel Cell output off
ERROR	10		Error Condition

Figura 5.6: Estados de la pila según el dato enviado por la pila

Una de las destacadas ventajas del software QGroundControl es su notable capacidad de personalización. En este contexto, se llevará a cabo la programación de un widget en la pantalla principal de la aplicación. Este widget tendrá la función de mostrar, entre otros elementos, el estado de la pila [5.6] y el porcentaje restante de carga, utilizando como referencia los valores de energía en vatios-hora (Wh) y el voltaje de la pila. Estos dos porcentajes proporcionarán una aproximación precisa del nivel de carga de la pila. Estos porcentajes aparte de estar en el widget lateral, se añadirán la barra de herramientas superior.

Para lograr esto, se introducirán nuevas variables en el archivo JSON correspondiente. Estas variables se obtendrán del protocolo MAVLink mediante la implementación de los

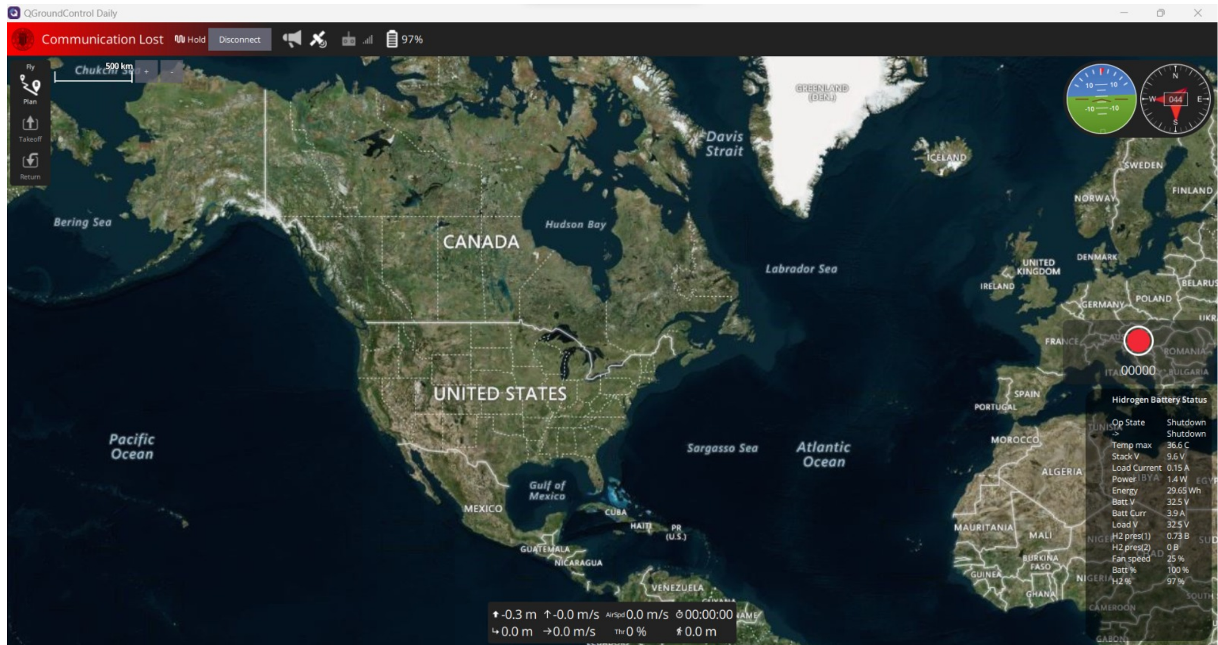

Figura 5.7: GUI del QgroundControl con las nuevas funcionalidades

archivos «VehicleBatteryFactGroup.h» y «VehicleBatteryFactGroup.cc». Posteriormente, se mostrarán los datos obtenidos en la pantalla principal de la interfaz de usuario, específicamente en el archivo «FlyViewCustomLayer.qml».

De esta manera, se habilitará la visualización y monitoreo del estado de la pila de manera clara y accesible para los usuarios, permitiéndoles tener una referencia precisa del nivel de carga restante y tomar decisiones informadas sobre la gestión de la energía en el sistema.

FlyViewCustomLayer.qml:

```
import QtQuick                  2.12
import QtQuick.Controls         2.4
import QtQuick.Dialogs          1.3
import QtQuick.Layouts          1.12

import QtLocation               5.3
import QtPositioning            5.3
import QtQuick.Window           2.2
import QtQml.Models             2.1

import QGroundControl               1.0
import QGroundControl.Airspace      1.0
import QGroundControl.Airmap        1.0
```

```
14 import QGroundControl.Controllers   1.0
15 import QGroundControl.Controls      1.0
16 import QGroundControl.FactSystem    1.0
17 import QGroundControl.FlightDisplay 1.0
18 import QGroundControl.FlightMap     1.0
19 import QGroundControl.Palette       1.0
20 import QGroundControl.ScreenTools   1.0
21 import QGroundControl.Vehicle       1.0
22
23 import MAVLink                      1.0
24
25
26 Item {
27     id: _root
28
29     property var parentToolInsets               // These
       insets tell you what screen real estate is available
       for positioning the controls in your overlay
30     property var totalToolInsets:   _toolInsets // These are
       the insets for your custom overlay additions
31     property var mapControl
32
33     QGCToolInsets {
34         id:                         _toolInsets
35         leftEdgeCenterInset:    0
36         leftEdgeTopInset:           0
37         leftEdgeBottomInset:        0
38         rightEdgeCenterInset:   0
39         rightEdgeTopInset:          0
40         rightEdgeBottomInset:       0
41         topEdgeCenterInset:       0
42         topEdgeLeftInset:           0
43         topEdgeRightInset:          0
44         bottomEdgeCenterInset:    0
45         bottomEdgeLeftInset:        0
46         bottomEdgeRightInset:       0
47     }
48     property int batteryMode: batteryStatusMessage.mode
49     Rectangle {
```

```
50          id: batt
51          width: 200
52          height: 400
53          color: Qt.rgba(0, 0, 0, 0.5)
54          radius: 20
55          anchors.bottom: _root.bottom
56          anchors.right: _root.right
57          anchors.bottomMargin: 10
58          anchors.rightMargin: 10
59
60          ColumnLayout {
61              id:                   mainLayout
62              anchors.margins:      ScreenTools.
                   defaultFontPixelWidth
63              anchors.top:          parent.top
64              anchors.right:        parent.right
65              spacing:              ScreenTools.
                   defaultFontPixelHeight
66
67              QGCLabel {
68                  Layout.alignment:   Qt.AlignCenter
69                  text:               qsTr("Hidrogen␣Battery␣
                       Status")
70                  font.family:        ScreenTools.
                       demiboldFontFamily
71              }
72
73              RowLayout {
74                  spacing: ScreenTools.defaultFontPixelWidth
75
76                  ColumnLayout {
77                      Repeater {
78                          model: _activeVehicle ?
                               _activeVehicle.batteries : 0
79
80                          ColumnLayout {
81                              spacing: 0
82
83                              property var
```

```
                                   batteryValuesAvailable:
                                   nameAvailableLoader.item
84
85                              Loader {
86                                  id:
                                      nameAvailableLoader
87                                  sourceComponent:
                                      batteryValuesAvailableComponent

88
89                                  property var battery: object
90                              }
91
92
93                              QGCLabel { text: qsTr("Op State")
                                    }
94                              QGCLabel { text: qsTr("        ")
                                    }
95
96                              QGCLabel { text: qsTr("Temp max")
                                   }
97                              QGCLabel { text: qsTr("Stack V")}
98                              QGCLabel { text: qsTr("Load
                                   Current")}
99                              QGCLabel { text: qsTr("Power")}
100                             QGCLabel { text: qsTr("Energy")}
101                             QGCLabel { text: qsTr("Batt V")}
102                             QGCLabel { text: qsTr("Batt Curr"
                                   )}
103                             QGCLabel { text: qsTr("Load V")}
104                             QGCLabel { text: qsTr("H2 pres(1)
                                   ")}
105                             QGCLabel { text: qsTr("H2 pres(2)
                                   ")}
106                             QGCLabel { text: qsTr("Fan speed"
                                   )}
107                             QGCLabel { text: qsTr("Batt %")}
108                             QGCLabel { text: qsTr("H2 %")}
109
```

```
110                     }
111                 }
112             }
113 
114             ColumnLayout {
115                 Repeater {
116                     model: _activeVehicle ?
                        _activeVehicle.batteries : 0
117 
118                     ColumnLayout {
119                         spacing: 0
120 
121                         property var
                            batteryValuesAvailable:
                            valueAvailableLoader.item
122 
123                         Loader {
124                             id:
                                valueAvailableLoader
125                             sourceComponent:
                                batteryValuesAvailableComponent

126 
127                             property var battery: object
128                         }
129 
130                         //QGCLabel { text: "" }
131 
132                         //QGCLabel { text: object.mode.
                            valueString  }
133                         QGCLabel {
134                             text: {
135                                 if (object.mode.
                                    valueString === "0") {
136                                     return "Startup";
137                                 } else if (object.mode.
                                    valueString === "1") {
138                                     return "Startup";
139                                 } else if (object.mode.
```

```
                                    valueString === "2") {
140                                   return "Startup";
141                               } else if (object.mode.
                                    valueString === "3") {
142                                   return "Startup";
143                               } else if (object.mode.
                                    valueString === "4") {
144                                   return "Startup";
145                               } else if (object.mode.
                                    valueString === "5") {
146                                   return "Startup";
147                               } else if (object.mode.
                                    valueString === "6") {
148                                   return "Operation";
149                               } else if (object.mode.
                                    valueString === "7") {
150                                   return "Operation";
151                               } else if (object.mode.
                                    valueString === "8") {
152                                   return "Shutdown";
153                               } else if (object.mode.
                                    valueString === "9") {
154                                   return "Shutdown";
155                               } else if (object.mode.
                                    valueString === "10")
                                    {
156                                   return "ERROR";
157                               } else {
158                                   return object.mode.
                                        valueString;
159                               }
160                           }
161                       }
162                       QGCLabel {
163                           text: {
164                               if (object.mode.
                                    valueString === "0") {
165                                   return "No␣state";
166                               } else if (object.mode.
```

```
                                        valueString === "1") {
167                                       return "Inputs␣ready"
                                            ;
168                                     } else if (object.mode.
                                        valueString === "2") {
169                                       return "Loop␣ready";
170                                     } else if (object.mode.
                                        valueString === "3") {
171                                       return "Start␣HGS";
172                                     } else if (object.mode.
                                        valueString === "4") {
173                                       return "Checking␣
                                            start␣up";
174                                     } else if (object.mode.
                                        valueString === "5") {
175                                       return "Clean␣Stack";
176                                     } else if (object.mode.
                                        valueString === "6") {
177                                       return "Operation";
178                                     } else if (object.mode.
                                        valueString === "7") {
179                                       return "Conditioning"
                                            ;
180                                     } else if (object.mode.
                                        valueString === "8") {
181                                       return "Shutdown";
182                                     } else if (object.mode.
                                        valueString === "9") {
183                                       return "OFF";
184                                     } else if (object.mode.
                                        valueString === "10")
                                        {
185                                       return "ERROR";
186                                     } else {
187                                       return object.mode.
                                            valueString;
188                                     }
189                                 }
190                             }
```

```
191                                 QGCLabel { text: object.temperature.valueString + "␣" + object.temperature.units }
192                                 QGCLabel { text: object.stackv.valueString/10 + "␣V" }
193                                 QGCLabel { text: object.currentbattery.valueString/100 + "␣A"  }
194                                 QGCLabel { text: object.mahConsumed.valueString/10 + "␣W" }
195                                 QGCLabel { text: object.energy.valueString/100 + "␣Wh" }
196                                 QGCLabel { text: object.battvolt.valueString/10 + "␣V" }
197                                 QGCLabel { text: object.type.rawValue == 1 ? "-"+object.timeRemaining.valueString/10 + "␣A" : object.timeRemaining.valueString/10 + "␣A" }
198                                 QGCLabel { text: object.loadvolt.valueString/10 + "␣V" }
199                                 QGCLabel { text: object.h2press1.valueString/100 + "␣B" }
200                                 QGCLabel { text: object.h2press2.valueString/10 + "␣B" }
201                                 QGCLabel { text: object.fanspeed.valueString/10 + "␣%" }
202                                 QGCLabel { text: object.percentRemaining.valueString + "␣%" }
203                                 QGCLabel { text: Math.round(100-object.energy.valueString/1060) + "␣%" }
204                             }
205                         }
206                     }
207                 }
208             }
```

```
209         }
210 }
```

5.5. Integración de todos los elementos

Con base en la información expuesta hasta ahora, hemos logrado definir nuestro sistema de manera exhaustiva, dividiéndolo en diversas partes. Ahora es necesario comprender la imagen global de dicho sistema. La premisa fundamental es que la pila de hidrógeno está instalada en la aeronave, y un cable conecta su conector RS232 a la placa Pixhawk. Es en esta placa donde se carga nuestro programa, el cual incluye la aplicación de lectura de puertos que se ejecutará desde el encendido de la placa. Se ha verificado que esta aplicación no interfiere con ningún otro proceso en curso.

La aplicación se encarga de leer el mensaje proveniente del puerto de comunicación, y a continuación, se divide dicho mensaje en los distintos valores que resultan relevantes para conocer el estado de la pila. Estos valores se transmiten al bus de datos uORB a través de nuestro mensaje personalizado. En este punto, el mensaje MAVLink de la batería adquiere los valores directamente del bus uORB y los envía. Es en ese momento cuando los datos llegan a QGroundControl y se presentan en la pantalla principal de la interfaz.

Esta secuencia de eventos asegura una comunicación fluida y confiable entre la pila de hidrógeno y el sistema de control en tierra, permitiendo una supervisión precisa y en tiempo real del estado de la pila a través de la interfaz de usuario proporcionada por QGroundControl.

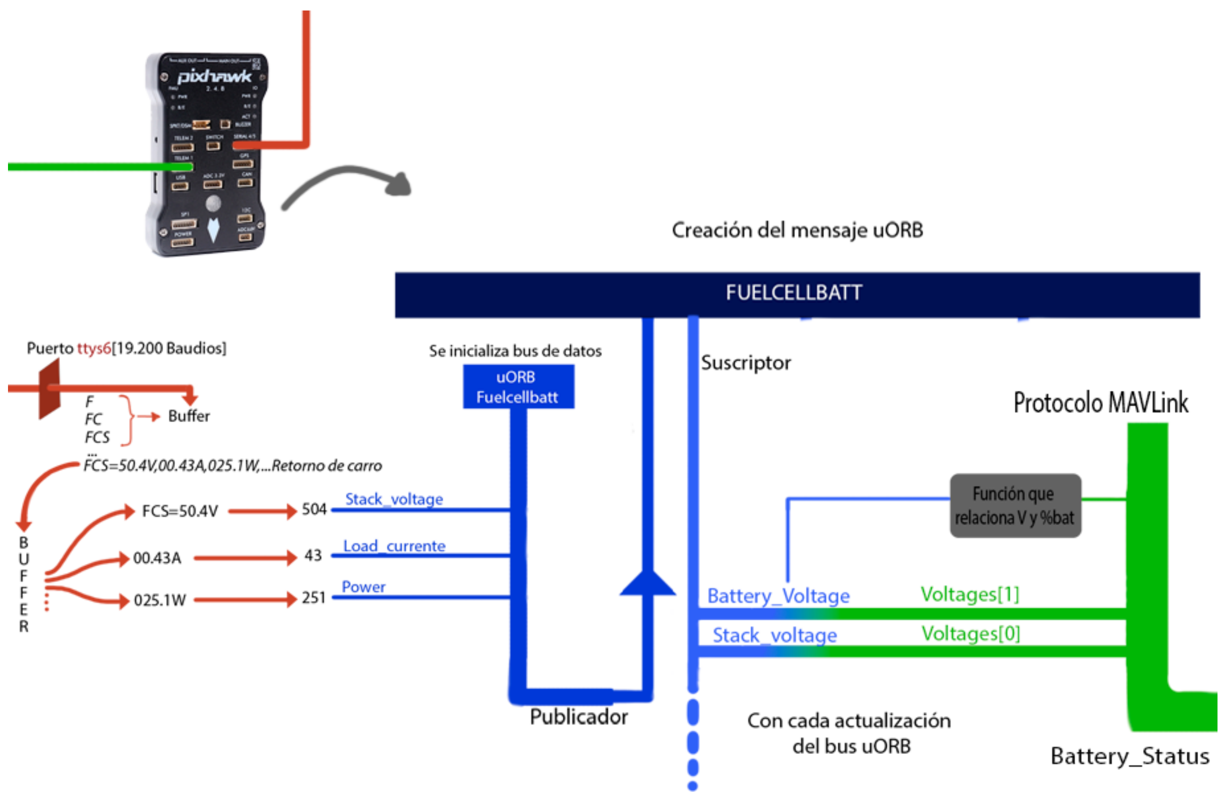

Figura 5.8: Todos los sistemas

Capítulo 6

Test experimentales

En relación a los experimentos realizados, se procedió inicialmente a la grabación de la secuencia de datos enviada por la pila de hidrógeno utilizando el software de captura de tramas denominado Realterm. Este enfoque permitió simular la recepción de datos en la placa Pixhawk sin la necesidad de contar con la pila de hidrógeno activa en el entorno de laboratorio. Además, estas tramas se guardaron en un archivo de texto (.txt), lo que posibilitó su edición y el análisis de escenarios extremos o situaciones que serían difíciles de replicar con la pila real, como valores altos de energía (Wh) o corrientes eléctricas negativas o nulas.

Una vez verificado el correcto funcionamiento de Realterm y la captura de datos, se procedió a establecer una conexión directa entre la pila y la placa Pixhawk, con el fin de comprobar la integridad y el correcto funcionamiento de la comunicación en tiempo real. Esta etapa de validación permitió corroborar la compatibilidad y la adecuada transferencia de datos entre ambos dispositivos, asegurando así el correcto enlace entre la pila de hidrógeno y la placa Pixhawk.

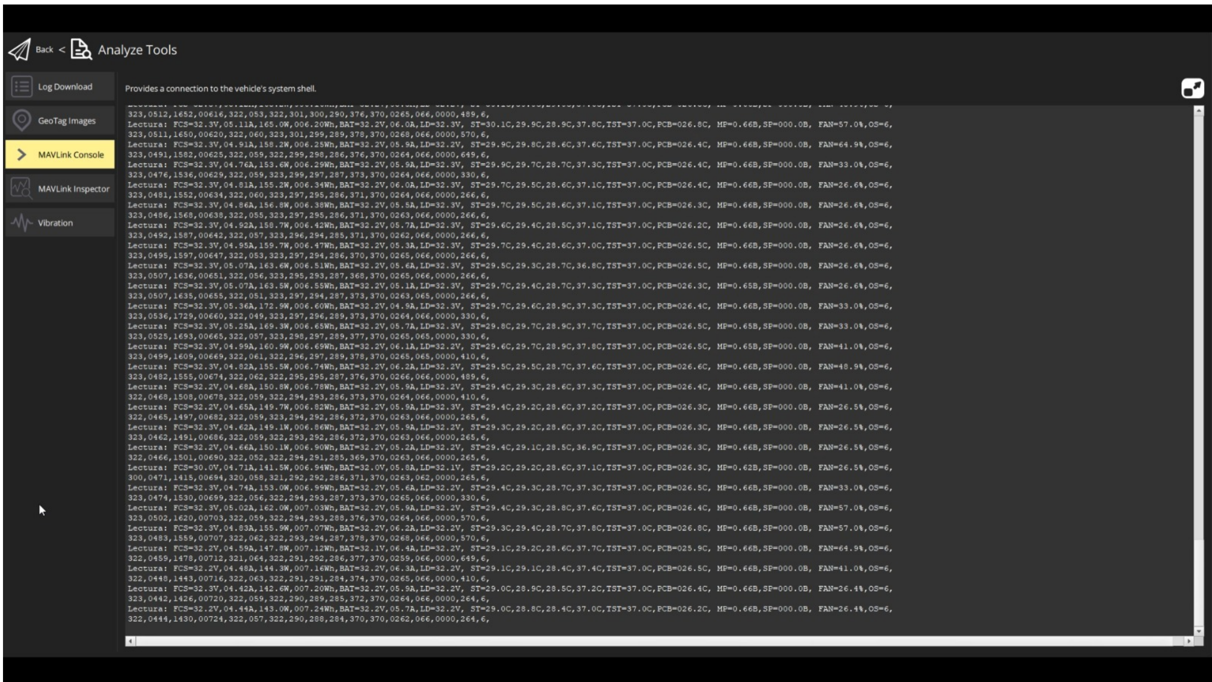

Figura 6.1: Recepción del mensaje

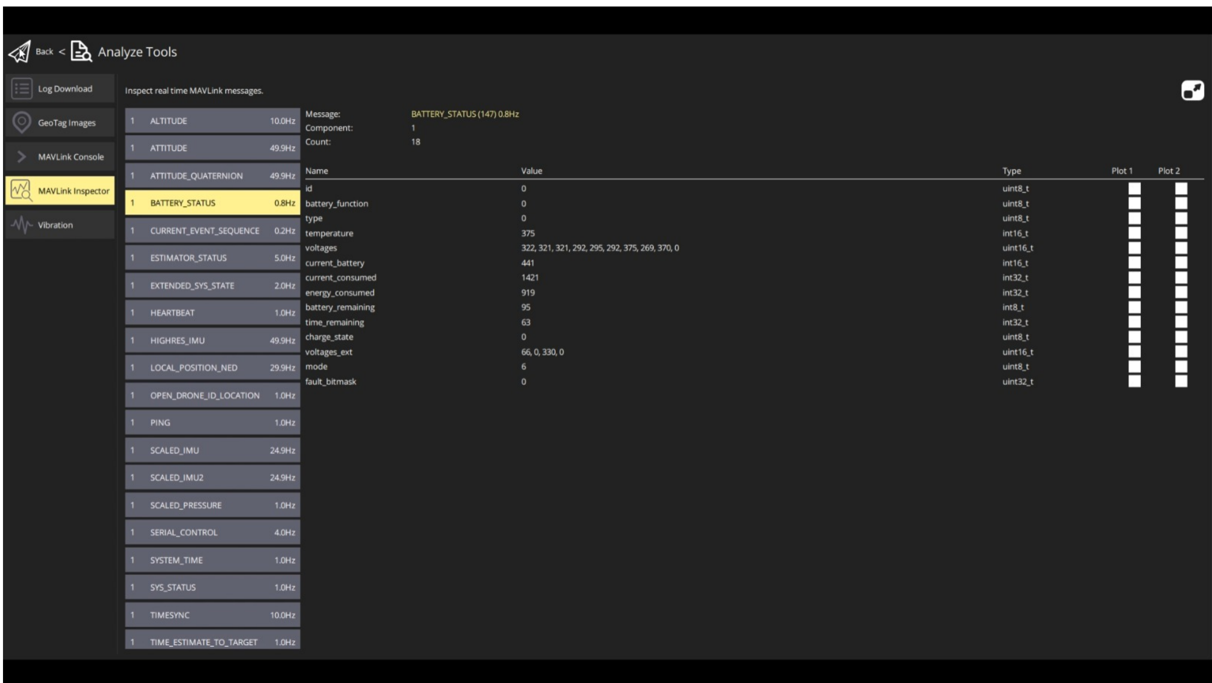

Figura 6.2: Inspección del mensaje MAVLink

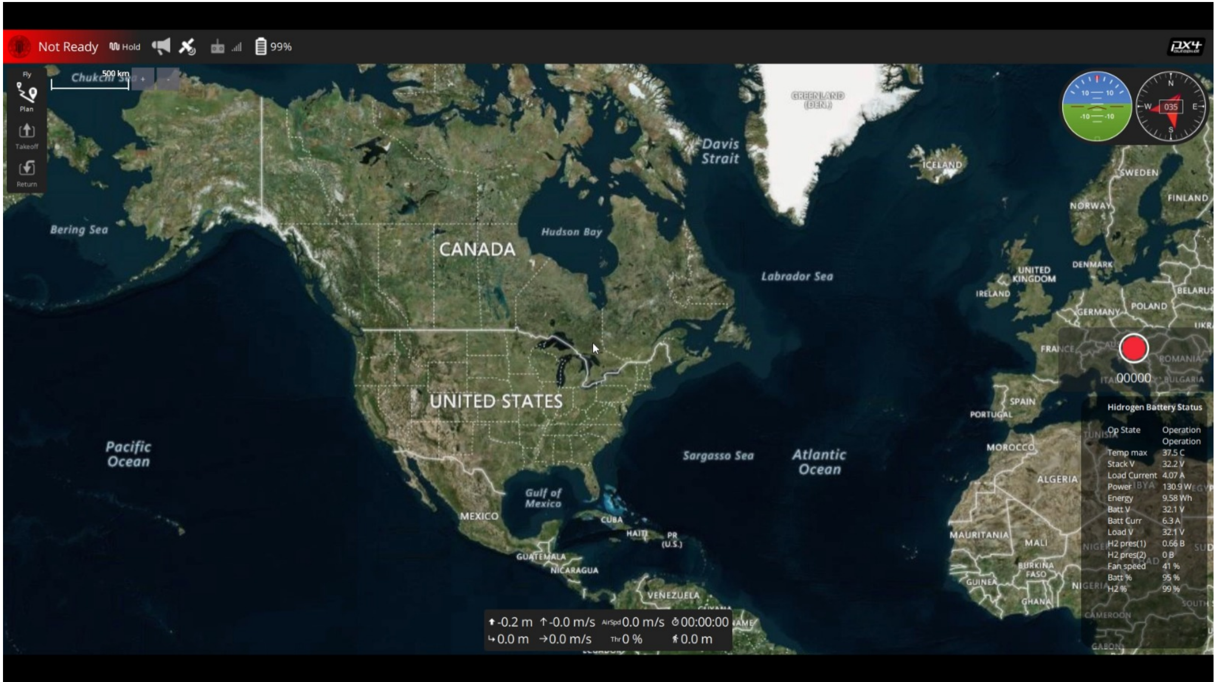

Figura 6.3: Qgroundcontrol personalizado

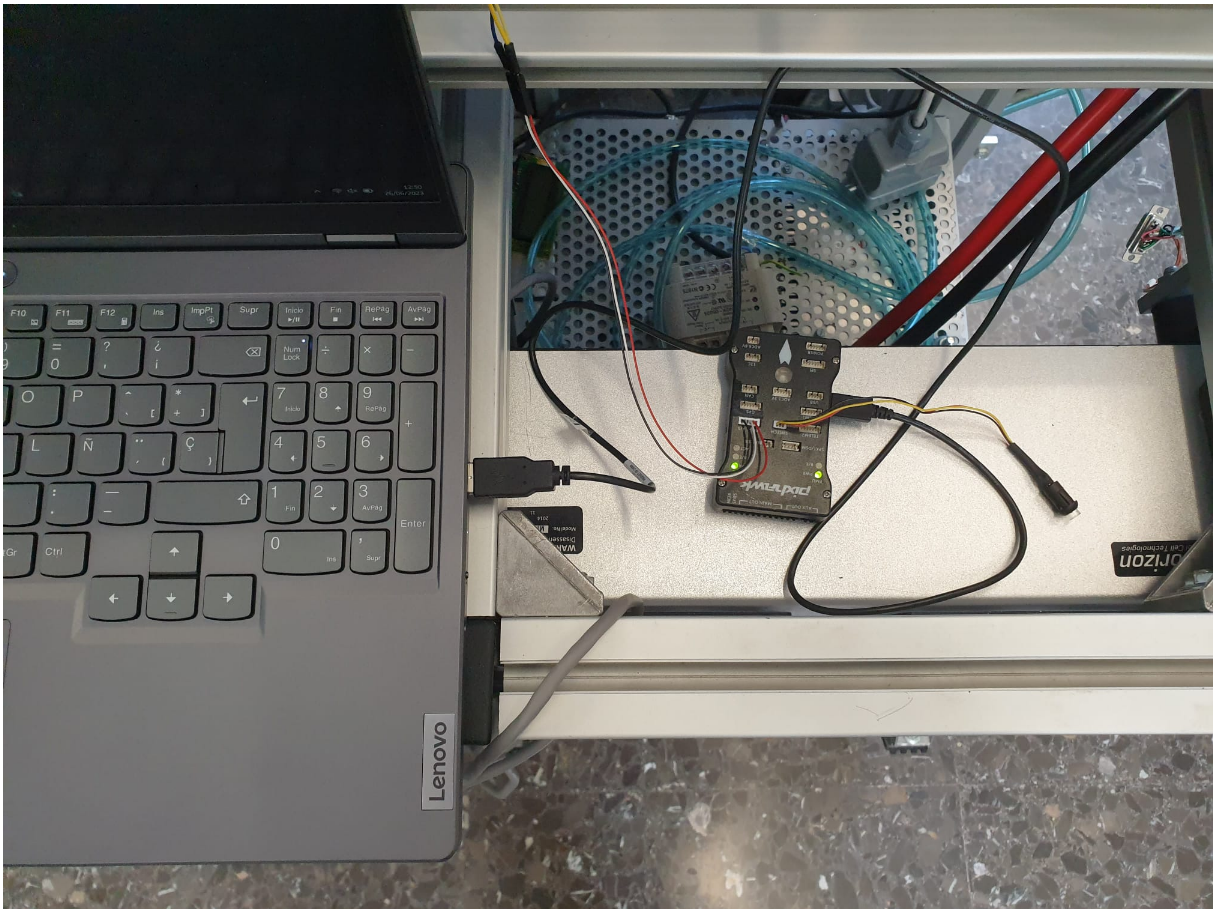

Figura 6.4: Foto del setup montado

Capítulo 7

Conclusiones

En resumen, se ha logrado establecer exitosamente una comunicación fluida y rápida entre la pila de hidrógeno y el Sistema de Control en Tierra (GCS), utilizando la placa de autopiloto Pixhawk 2 como interfaz. Mediante pruebas con el sistema real, se ha verificado el correcto funcionamiento de esta comunicación.

Se considera que esta funcionalidad de monitorización y control representa un avance significativo en el proyecto HYDRONE y en el desarrollo de drones basados en baterías de hidrógeno. Al permitir un monitoreo más detallado de la pila y la posibilidad de gestionar su rendimiento desde el GCS, se espera mejorar la eficiencia energética y optimizar el uso de los recursos, contribuyendo así a la evolución de los drones impulsados por baterías de hidrógeno.

Viendo que los objetivos marcados para este proyecto eran: diseñar e implementar un módulo de comunicación entre una pila de hidrógeno y un sistema de piloto automático, utilizando en nuestro caso la placa Pixhawk editar el código del QGroundControl, con el fin de visualizar el mensaje de la pila como un widget en la pantalla principal.Se puede concluir que los objetivos han sido cumplidos, tal como se evidencia en los puntos anteriores.

Referencias de imágenes

Nº Figura	Fuente
1.1	https://www.hispadrones.com/principiantes/aprendizaje-consejos/tipos-de-drones/
1.2	https://commons.wikimedia.org/wiki/File:Kettering_Bug.jpg#/media/Archivo:Kettering_Bug.jpg
1.3	https://commons.wikimedia.org/wiki/File:OQ-2A-Radioplane.jpg#/media/Archivo:OQ-2A-Radioplane.jpg
1.4	https://www.ai2.upv.es/proyecto/hydrone/
3.1	Realización propia
3.2	A1000LV_2022 User Manual
3.3	https://commons.wikimedia.org/wiki/File:Fuel_cell_ES.svg#/media/Archivo:Fuel_cell_ES.svg
3.4	A1000LV_2022 User Manual
3.5	A1000LV_2022 User Manual
4.1	https://www.roscomponents.com/es/imus/146-rc-pixhawk.html
4.2	https://www.embention.com/es/news/autopiloto-veronte-la-eleccion-perfecta/
4.3	https://es.aliexpress.com/i/1005005027961701.html
4.4	https://www.researchgate.net/profile/Emad-Samuel-Malki-Ebeid/publication/325134452/figure/fig5/AS:850151706996747@1579703370331/ArduPilot-Mega-APM-28-autopilot.jpg
4.5	https://m.media-amazon.com/images/I/615upzJ8vxL.jpg
4.6	https://www.ercmarket.com/image.php?id=4814&type=T
4.7	https://docs.px4.io/main/logo_pro_small.png
4.8	https://ardupilot.org/
4.9	https://github.com/betaflight/betaflight
4.10	https://commons.wikimedia.org/wiki/File:LibrePilot_Logo.svg#/media/File:LibrePilot_Logo.svg
4.11	Realización propia
4.12	Realización propia

Nº Figura	Fuente
4.13	https://blogger.googleusercontent.com/img/a/AVvXsEhprzR2WwA50wOgyblVW9S5vd4Z2hgmIpXYqBIBFHak7ajDcVWkEP-BjVj2CuTksYtThP79O pLXE7sIVNvBsICJkEYxyLdkPcK1ewNxfocTGymdKLbGi_zJ1DBPoNk4Y4c7StF8ML99ffOQ4YAG9B49BXHJ1kefCqbX_0s0l2k9e2jiyMAnFYtrTg=w571-h357
4.14	https://pbs.twimg.com/media/CtEBt_vWEAAKyed.png
5.1	Realización propia
5.2	https://docs.px4.io/main/en/concept/architecture.html
5.3	Realización propia
5.4	Realización propia
5.5	Realización propia
5.7	Realización propia
5.8	Realización propia
6.1	Realización propia
6.2	Realización propia
6.3	Realización propia
6.4	Realización propia

Bibliografía

[1] Dalamagkidis, K., Valavanis, K. P., & Piegl, L. A. (2010). *Unmanned aircraft systems: UAVS design, development and deployment.* Springer Science & Business Media.

[2] Quigley, M., Gerkey, B., & Smart, W. D. (2009). *Programming robots with ROS: a practical introduction to the Robot Operating System.* O'Reilly Media.

[3] Roca, R. V., Candelas, F. A., Torres, F., & Cunado, M. A. (2012). *Unmanned aerial vehicle control system.* In *Advances in unmanned aerial vehicles* (pp. 9-28). Springer.

[4] Piotrowska, A., & Niewiadomska-Szynkiewicz, E. (2019). *Classification and identification of UAVs - a review. Applied Sciences*, 9(14), 2859.

[5] Awwal, A. A., & Ansari, A. (2021). *A comprehensive review of multirotor drones: Design, control, and applications. IEEE Access*, 9, 130282-130307.

[6] Kamal, M. A., & Alazab, M. (2020). *Unmanned aerial vehicles in surveillance: A review on classification, applications, challenges, and recommendations. IEEE Access*, 8, 105848-105869.

[7] Arrieta, R. S., Díaz, J. M., Garrido, J. L., & Santiso, E. (2016). *UAVs for Specific Applications.* In *Unmanned Aircraft Systems*, Vol. 1, (pp. 1-32). Springer.

[8] Wightman, P. L. (2015). *Micro Air Vehicles.* In *Unmanned Aircraft Systems*, (pp. 29-41). Elsevier.

[9] Singh, J. R., & Kumar, S. (2019). *Unmanned Aircraft Systems: An Overview.* In *Unmanned Aircraft Systems*, Vol. 1, (pp. 1-24). Springer.

[10] Valavanis, P. (2015). *Unmanned Aerial Vehicles (UAVs) Classification and Applications.* In *Handbook of Unmanned Aerial Vehicles*, (pp. 3-30). Springer.

[11] Klimek, P., & Mulczyk, K. (2021). *Unmanned Aerial Vehicles (UAV) in Military Applications.* In *Proceedings of the 8th International Conference on Information Management and Industrial Engineering (ICII 2021)*, (pp. 417-425). Springer.

[12] Achtelik, M., & Mellinger, D. (2010). *Applications of small unmanned aerial vehicles for civil use. Journal of Field Robotics*, 27(1), 127-158.

[13] Torres-Sospedra, J., Knoben, N., Soto, J., & Huerta, J. (2018). *Unmanned Aerial Systems for Civil Applications: A Review. Drones*, 2(2), 13.

[14] Khatibi, A., et al. (2021). *Unmanned Aerial Vehicles (UAVs) for Environmental Monitoring and Management: A Review. Remote Sensing*, 13(3), 419.

[15] González-Jiménez, J., et al. (2018). *UAV-based remote sensing for environmental monitoring in Antarctica. Remote Sensing*, 10(7), 1151.

[16] Shachtman, N. (2012). *Unmanned: Drones, Data, and the Illusion of Perfect Warfare.* Simon & Schuster.

[17] Whittington, M. (2015). *Military Robots and Drones: A Reference Handbook.* ABC-CLIO.

[18] Rajaee, N., & Shiva, S. (2018). *Recent advances in unmanned aerial vehicle applications: A review. International Journal of Unmanned Systems Engineering*, 6(1), 1-17.

[19] Merino Calleja, J., et al. *Plataforma de simulación multinivel y análisis de prestaciones para sistemas robóticos basados en ROS.* 2021.

[20] *Proyecto HyDrone.* Recuperado de *https://www.ai2.upv.es/proyecto/hydrone/*

[21] *Avances HyDrone.* Recuperado de *https://www.ai2.upv.es/el-ai2-trabaja-en-el-diseno-del-sistema-de-propulsion-y-control-capaz-de-doblar-la-autonomia-de-vuelo-en-drones-de-ala-fija/*

[22] *HORUS - UAV Team UPV.* Recuperado de *https://www.linkedin.com/company/horusupv/about/*

[23] *University of Surrey crowned the winners of the 2022 UAS Challenge.* Recuperado de *https://www.imeche.org/news/news-article/university-of-surrey-crowned-the-winners-of-the-2022-uas-challenge*

[24] *Pixhawk Autopilot.* Recuperado de *https://pixhawk.org/products/*

[25] *Autopiloto UAV-UAM.* Recuperado de *https://www.embention.com/es/productos/componentes-drones-y-uam/autopiloto-uav-uam/*

[26] *PX4 Autopilot.* Recuperado de *https://docs.px4.io/main/en/*

[27] *ArduPilot - Open Source Autopilot.* Recuperado de *https://ardupilot.org/*

[28] *Mateksys Flight Controller.* Recuperado de *http://www.mateksys.com/?page_id=3834*

[29] *CC3D Flight Controller.* Recuperado de *https://opwiki.readthedocs.io/en/latest/user_manual/cc3d/cc3d.html*

[30] *PX4 Autopilot.* Recuperado de *https://px4.io/*

[31] *ArduPilot.* Recuperado de *https://ardupilot.org/*

[32] *Betaflight - FPV Flight Controller Software.* Recuperado de *https://betaflight.com/*

[33] *LibrePilot Open Source.* Recuperado de *https://librepilot.org/*

[34] *Ground Control Stations.* Recuperado de *http://www.diariodelaeromodelista.es/p/gcs.html*

[35] *QGroundControl.* Recuperado de *https://docs.qgroundcontrol.com/*

[36] *Betaflight.* Recuperado de *https://betaflight.com/*

[37] *LibrePilot.* Recuperado de *https://librepilot.org/*

[38] *Introducción al protocolo MAVLink.* Recuperado de *https://www.codio.es/2022/09/mavlink-protocolo-introduccion.html?showComment=1663487739513*

[39] *MAVLink.* Recuperado de *https://mavlink.io/en/*

UNIVERSIDAD POLITÉCNICA DE VALENCIA
(UPV)

ETSID

Diseño e implementación del módulo de comunicaciones entre un sistema de propulsión basado en pila de hidrógeno y el sistema autopiloto para vehículos aéreos no tripulados.

Pablo Ceacero Escrig

Tutor: Sergio García-Nieto Rodríguez
Grado: Ingeniería en Aeroespacial
Curso: 2022/2023

3 de julio de 2023

Trabajo de Fin de Grado [Manual de programación]

Índice general

1. Introducción **2**

2. PX4 **4**
2.1. Instalación y configuración del entorno 4
2.2. Github Hydrone PX4 10
2.3. Mensaje uORB 10
2.4. Aplicación 11
2.5. Edición del mensaje MAVLink 18
2.6. Compilar y subir 21

3. Qgrouncontrol **23**
3.1. Github Hydrone Qgrouncontrol 23
3.2. Modificación de los archivos 23

4. Conclusión **50**

Capítulo 1

Introducción

En este documento se presenta una descripción detallada sobre el diseño e implementación de una aplicación para la comunicación entre una pila de combustible A1000LV de H3Dynamics y una placa de autopiloto Pixhawk 2. El enfoque utilizado se basa en el software PX4, el cual se utiliza tanto en el repositorio original de PX4 (https://github.com/PX4/PX4-Autopilot) como en una versión personalizada preconfigurada. Para utilizar la aplicación, bastará con clonar el repositorio correspondiente y compilarlo.

Adicionalmente, se aborda una segunda parte del documento que se centra en la personalización del propio software QGroundControl, proporcionando información detallada sobre los procesos y técnicas necesarios para adaptar y ajustar la aplicación a requisitos específicos.

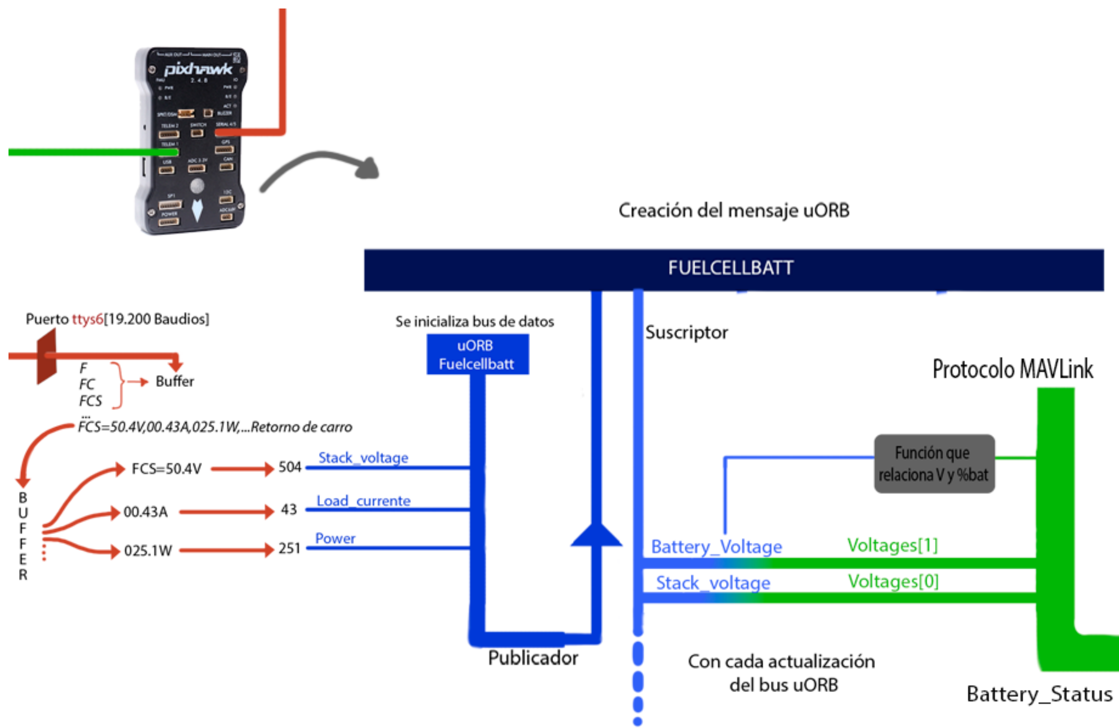

Figura 1.1: Esquema general de los programas en PX4

Capítulo 2

PX4

A continuación, se detalla el proceso de configuración del entorno de programación. En este caso, se enfoca en la programación en el sistema operativo Linux, el cual se elige debido a su eficiencia y rapidez en la compilación, además de ser el sistema donde PX4 ha sido ampliamente probado. Sin embargo, se trabajará en una computadora con Windows, por lo que será necesario instalar el Subsistema Windows para Linux (WSL, por sus siglas en inglés) a través de la siguiente fuente: https://learn.microsoft.com/en-us/windows/wsl/about. Esta herramienta permite la instalación y ejecución del entorno de desarrollo de Ubuntu dentro de Windows.

Adicionalmente, se puede trabajar desde el entorno de desarrollo integrado (IDE, por sus siglas en inglés) Visual Studio Code, el cual ofrece una extensión específica para WSL. Además, existen otras extensiones que facilitan la compilación y programación adecuada en este entorno.

En este proyecto, se llevará a cabo la creación de un nuevo mensaje uORB para agregar las variables provenientes del mensaje de la pila de hidrogeno, leídos y preclasificados por la aplicación, al bus de datos. Además, se explicará el proceso de desarrollo de una aplicación para llevar a cabo esta tarea. También se realizarán modificaciones en el mensaje MAVLink «battery_ status» para incluir la información relevante, aunque los nombres de las variables en este mensaje no se asociarán directamente con los campos del mensaje uORB. Sin embargo, en la interfaz del Sistema de Control en Tierra (GCS), se proporcionarán instrucciones claras para asignar cada número a la variable correspondiente.

2.1. Instalación y configuración del entorno

Es necesario realizar la instalación de WSL2, siendo preferible la versión 2 debido a su soporte para entornos gráficos, los cuales serán necesarios para las

pruebas iniciales, específicamente para el entorno de simulación SITL (Software-in-the-Loop).

En primer lugar, es importante asegurarse de que las máquinas virtuales están habilitadas en el ordenador. Para ello, se debe iniciar el ordenador en modo seguro y buscar una opción que generalmente se denomina «Virtualization Technology», «Intel VT-x» o «AMD-V», y activarla.

Una vez que se ha activado esta opción, se debe acceder a la consola de comandos (CMD) del ordenador como administrador y escribir el comando:

```
C:\> wsl --install
```

Luego, reinicie el ordenador una vez que aparezca el mensaje de finalización de la instalación. Al reiniciar, se debe abrir nuevamente la consola CMD (no es necesario que sea como administrador) y escribir el comando

```
C:\> WSL
```

para iniciar la máquina virtual. En este punto, se solicitará un nombre y una contraseña, los cuales se deben recordar, ya que serán necesarios más adelante.

En caso de no estar seguro de la versión de WSL instalada, se puede ejecutar el siguiente comando para verificarla.

```
C:\> wsl -l -v
```

Una vez que se tiene la consola de Ubuntu abierta, es importante recordar que al escribir

```
UbuntuMachine:~& exit
```

se saldrá de la consola de Ubuntu y se volverá a la de Windows, mientras que al escribir el siguiente comando en el cmd, se apagará la máquina virtual.

```
C:\> wsl --shutdown
```

A continuación, se procede a instalar la herramienta PX4 toolchain. Desde la consola de Ubuntu, se navega hasta la carpeta «home» (puede tener otro nombre, pero se recomienda utilizar esta). Luego, se clona el repositorio mediante el comando

```
UbuntuMachine:~& git clone https://github.com/PX4/PX4-Autopilot.git
    --recursive
```

y se ejecuta el instalador «ubuntu.sh» ubicado en la carpeta

```
UbuntuMachine:~& bash ./PX4-Autopilot/Tools/setup/ubuntu.sh
```

«PX4-Autopilot/Tools/setup» mediante el siguiente comando. Después de esto, se reinicia la máquina virtual Ubuntu.

```
UbuntuMachine:~& Exit
UbuntuMachine:~& wsl --shutdown
UbuntuMachine:~& wsl
```

Posteriormente, se accede al repositorio de PX4 mediante el comando `cd  /PX4-Autopilot` y se procede a compilar el objetivo PX4 SITL y probar el entorno.

Para el desarrollo del proyecto, se utilizará el entorno de desarrollo integrado (IDE) Visual Studio Code en Windows. A continuación, se detalla el proceso de instalación y configuración:

1. Descargue e instale Visual Studio Code desde el siguiente enlace: [https://code.visualstudio.com/].
2. Una vez instalado, abra Visual Studio Code y proceda a instalar la extensión llamada «Remote - WSL» [https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-wsl].

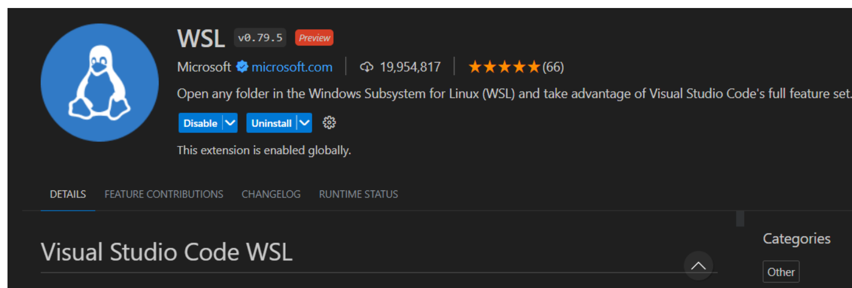

Figura 2.1: Extension de VS Code, WSL.

3. Abra un terminal de Windows Subsystem for Linux (WSL) dentro de Visual Studio Code. Para ello, presione Ctrl+E y escriba «wsl».Despues de la primera vez, podrá acceder al entorno haciendo clic en «File» y luego en «Open Recent».

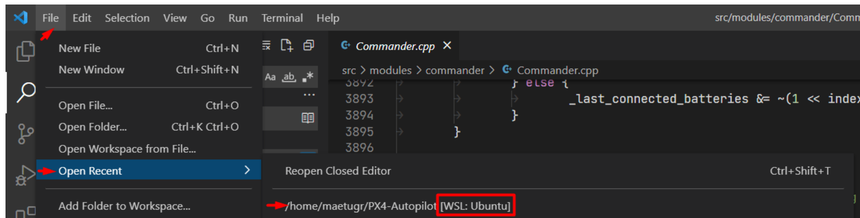

Figura 2.2: Como abrir de nuevo el entorno WSL

4. En el shell de WSL, dentro de VS, navegue hasta la carpeta «PX4» y ejecute el comando

```
UbuntuMachine:~/PX4-Autopilot$ code .
```

para iniciar Visual Studio Code integrado con el shell de WSL.

5. Ahora clique en abrir carpeta [2.3] y seleccione «PX4-Autopilot». Se le mostrará una notificación en la parte inferior derecha, indicando que este espacio de trabajo tiene extensiones recomendadas [2.4]. Haga clic en «Install All» para instalar estas extensiones recomendadas.

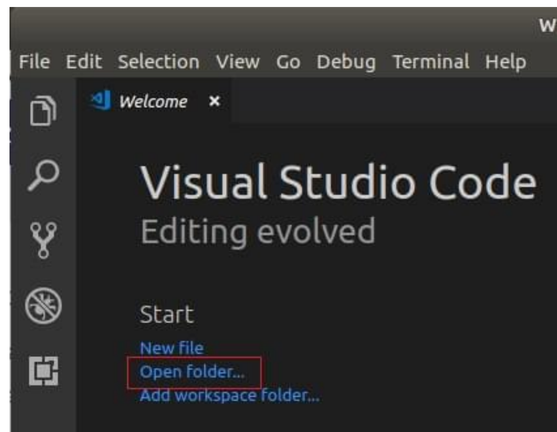

Figura 2.3: Como abrir carpeta de trabajo.

6. Si se le notifica sobre una nueva versión de CMake, elija la opción «No» ya que la versión especificada es la correcta para el proyecto.

Al seguir estos pasos, habrá configurado Visual Studio Code para trabajar de forma integrada con el shell de WSL, lo que facilitará el desarrollo en el entorno de PX4-Autopilot.

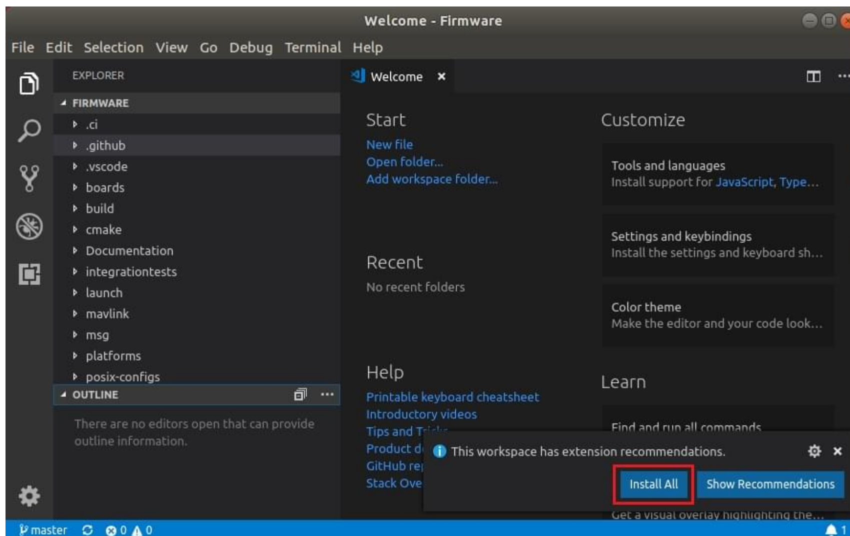

Figura 2.4: Extensiones recomendadas

```
UbuntuMachine:~/PX4-Autopilot$ code .
```

Ahora Solo falta completar la configuración, resta realizar la construcción de PX4 y verificar que todo se haya instalado correctamente. Siga los siguientes pasos:

1. Seleccione su constructor de código en Visual Studio Code. Si el constructor de código deseado ya se encuentra en la barra azul de herramientas, puede omitir este paso. De lo contrario, haga clic en el constructor de código y seleccione el que prefiera. Para esta aplicación, el constructor de código predeterminado es suficiente.

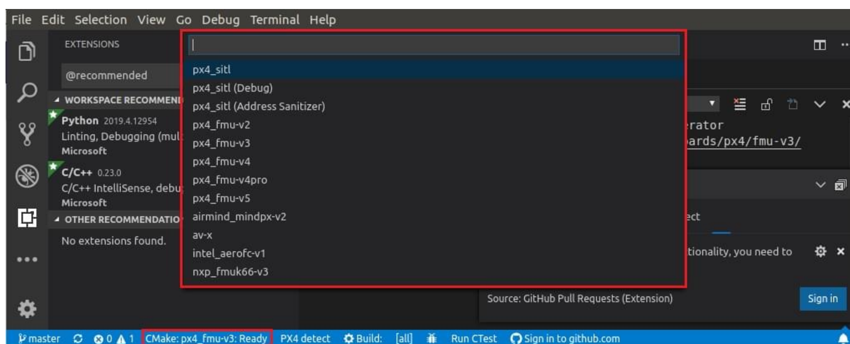

Figura 2.5: Selección de constructor.

2. El proyecto se configurará según el sistema de construcción CMake. Puede supervisar el progreso de la configuración en la pestaña «Output».

3. Una vez completada la configuración, escriba el comando

```
UbuntuMachine:~/PX4-Autopilot& make px4_sitl jmavsim
```

en la terminal integrada de Visual Studio Code. Este comando compilará el proyecto y ejecutará el simulador JMavSim. Además, se abrirá una consola NSH en la propia interfaz de Visual Studio Code, permitiéndole llamar comandos relacionados con la placa (en este caso, en el entorno SITL).

Al seguir estos pasos, se construirá PX4 y se lanzará el simulador JMavSim, brindándole la posibilidad de interactuar con la placa a través de la consola NSH dentro de Visual Studio Code. Esto le permitirá verificar el correcto funcionamiento de la configuración realizada.

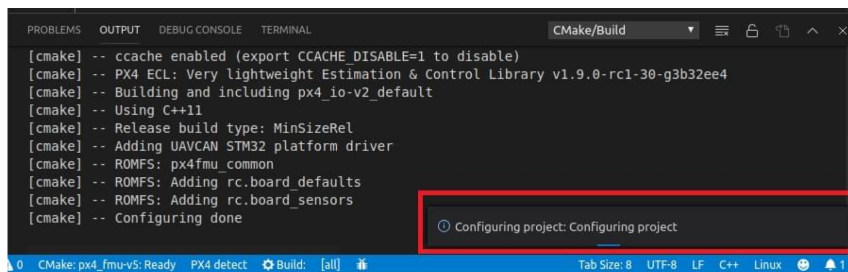

Figura 2.6: Progreso del configurado del constructor.

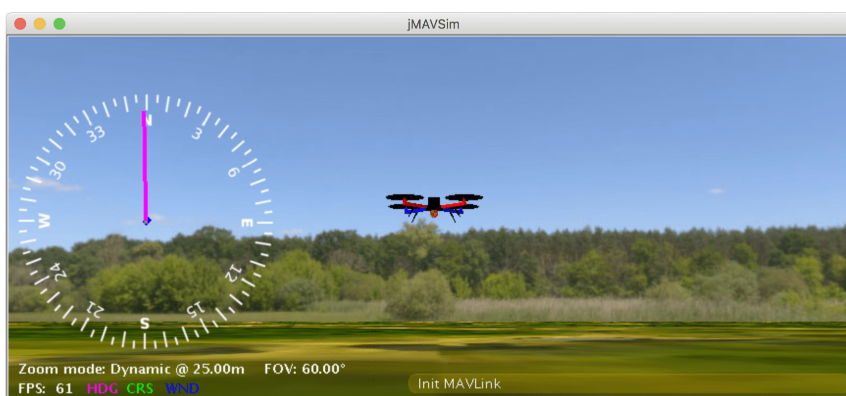

Figura 2.7: Simulador Jmavsim.

2.2. Github Hydrone PX4

Para descargarse el proyecto que se comentará más adelante ya hecho se debe clonar el siguiente código de github de la siguiente manera:

```
UbuntuMachine:~$ git clone https://github.com/sergarro/
    PX4_HyDrone.git --recursive
```

Este paso remplaza la línea de comando del final de la página 4, pero se sigue necesitando hacer toda la configuración anteriormente comentada. Si ha hecho esto, puede saltar hasta el punto 2.6. Si no continué con el manual.

2.3. Mensaje uORB

El uORB (micro Object Request Broker) es un sistema de mensajería, publicación y suscripción utilizado en el marco del proyecto PX4, que es una plataforma de código abierto para vehículos aéreos no tripulados (UAVs). El uORB proporciona un mecanismo eficiente para que los diferentes módulos y componentes del sistema se comuniquen entre sí a través de mensajes.

En uORB, los mensajes se definen mediante archivos de descripción (.msg) que especifican la estructura y el tipo de datos de cada mensaje. Estos mensajes pueden contener información sobre el estado del vehículo, datos de sensores, comandos de control, entre otros. Los módulos del sistema pueden publicar mensajes en un determinado tema (topic) y otros módulos pueden suscribirse a esos temas para recibir las actualizaciones de los mensajes.

El uORB gestiona la comunicación de mensajes de forma eficiente y desacoplada, lo que permite una comunicación fluida entre los diferentes componentes del sistema PX4. Además, proporciona una interfaz sencilla para el desarrollo de software, ya que los mensajes se generan automáticamente y se accede a ellos mediante una API simple.

Para crear un nuevo tema (topic) para los mensajes uORB en este caso, se debe crear un nuevo archivo del tipo *.msg* en la carpeta correspondiente «/msg» y añadir el nombre del mensaje creado al CMakeLists, este archivo se encuentra en la misma carpeta que los archivos .msg. En nuestro caso, el mensaje se llama «fuelcellbatt» y tiene la siguiente estructura:

```
1 uint64 timestamp                #tiempo desde el inicio del
    sistema (microsegundos)
```

```
uint64 timestamp_sample      #el tiempo del dato sin procesar (microsegundos)
uint16 pila                  #info--
uint16 stack_voltage         #info1--
uint16 load_currente         #info2--
uint16 power                 #info3--
uint16 energy                #info4--
uint16 battery_voltage       #info5--
uint16 battery_current       #info6--
uint16 load_voltage          #info7--
uint16 temp1                 #info8--
uint16 temp2                 #info9--
uint16 temp3                 #info10--
uint16 temp4                 #info11--
uint16 target_temp           #info12
uint16 board_temp            #info13--
uint16 h2_pressure           #info14--
uint16 fan_speed             #info15--
uint16 state                 #info16
uint16 signo

# TOPICS fuelcellbatt
```

La inclusión de la última línea en el programa es fundamental, ya que establece el mecanismo mediante el cual los demás archivos pueden acceder a este mensaje. En cada programa que haga uso del mensaje, se debe incorporar la directiva `#include <uORB/topics/fuelcellbatt.h>`, que en este caso será en la aplicación [archivo de C] y en el de MAVLink. Cabe destacar que este archivo es de tipo .h, lo que indica que se trata de un encabezado (header) generado automáticamente durante la compilación a partir del mensaje .msg correspondiente. A partir de este punto, el mensaje puede ser utilizado en cualquier programa para suscribirse a este tema y así visualizar las actualizaciones de las variables o bien, leer este flujo de datos.

2.4. Aplicación

En esta sección se va a explicar el proceso por el cual se crea la aplicación y los archivos necesarios para su correcta compilación.

Cabe destacar que la aplicación se puede programar tanto en la carpeta «src/e-

xamples/» o en la «src/modules/», en este caso se ha elegido la carpeta examples. En esta carpeta hay que crear una carpeta con el nombre que tendrá la aplición y dentro de esta habrán 3 archivos[para este caso] que es el archivo C, el Kconfig y el CMakeList.

El archivo CMakeLists es un archivo de configuración esencial que se utiliza en el proceso de compilación para especificar los detalles del proyecto, como las fuentes de código, las bibliotecas externas necesarias y las opciones de compilación. Este archivo permite al sistema de construcción, en este caso CMake, comprender cómo construir el proyecto y generar los archivos binarios resultantes.

Por otro lado, el archivo Kconfig se emplea para definir la configuración del kernel de PX4. Contiene las opciones de configuración disponibles y sus valores predeterminados, lo que permite personalizar la construcción del firmware según los requisitos específicos del proyecto.

Y por último, esta aplicación se explica más adelante su función concreta. Pero a grandes rasgos lee el puerto «ttys6» de la placa Pixhawk 2 que **corresponde al puerto fisico UART4**, si se desea cambiar esto hay que ver que puerto UART se esta usando y ver que TTYS le corresponde y editar en consecuencia la aplicación.

A continuación se presentarán los códigos:

prueba.c:

```
#define SERIAL_PORT "/dev/ttyS6"
#define BUFFER_SIZE 145

__EXPORT int prueba_main(int argc, char *argv[]);

int prueba_main(int argc, char *argv[])
{

    PX4_INFO("Iniciando puerto...");

    int serialPort;
    struct termios serialSettings;
    char buffer[BUFFER_SIZE];
    int buffIndex = 0;

    // Abrir el puerto serie en modo lectura
    serialPort = open(SERIAL_PORT, O_RDONLY);
    if (serialPort == -1) {
```

```
        perror("Error al abrir el puerto serie");
        return 1;
    }

    // Configurar la velocidad y otros parámetros del puerto serie
    tcgetattr(serialPort, &serialSettings);
    cfsetispeed(&serialSettings, B19200);  // Velocidad de baudios
    serialSettings.c_cflag &= ~PARENB;    // Sin paridad
    serialSettings.c_cflag &= ~CSTOPB;    // 1 bit de parada
    serialSettings.c_cflag &= ~CSIZE;     // Limpiar bits de tamaño de carácter
    serialSettings.c_cflag |= CS8;        // 8 bits de datos
    tcsetattr(serialPort, TCSANOW, &serialSettings);

    /* advertise attitude topic */
  struct fuelcellbatt_s att;
  memset(&att, 0, sizeof(att));
  orb_advert_t att_pub = orb_advertise(ORB_ID(fuelcellbatt), &att);

    // Leer del puerto serie y almacenar en el vector buff
    while (1) {

        char receivedChar;
        ssize_t bytesRead = read(serialPort, &receivedChar, sizeof(receivedChar));
        if (bytesRead > 0) {

            if (receivedChar == '\n') {
                buffer[buffIndex] = '\0';  // Agregar terminador de cadena
                printf("Lectura: %s\n", buffer);
                if (buffIndex >= 100) { //este if es lo nuevo borrar si deja de funcionar

```

```
49              //buffer[buffIndex] = '\0';  //
                   Agregar terminador de cadena
50              //printf("Lectura (tamaño máximo
                   alcanzado): %s\n", buffer);
51
52              char datos[55];
53              int contador = 0;
54              int caso=1;
55              for (int j = 0; j < (buffIndex+2); j
                   ++) {
56                  if (isdigit(buffer[j])) {
57                      datos[contador++] = buffer[j
                           ];
58                  }
59                  if(buffer[j] == ','){
60                      datos[contador] = '\0';
61                      switch (caso) {
62                          case 1:
63                              printf("%s,",datos);
64                              att.stack_voltage=
                                   atof(datos);
65                              break;
66                          case 2:
67                              printf("%s,",datos);
68                              att.load_currente=
                                   atof(datos);
69                              break;
70                          case 3:
71                              printf("%s,",datos);
72                              att.power=atof(datos
                                   );
73                              break;
74                          case 4:
75                              printf("%s,",datos);
76                              att.energy=atof(
                                   datos);
77                              break;
78                          case 5:
79                              printf("%s,",datos);
80                              att.battery_voltage=
```

```
                                atof(datos);
81                          break;
82                      case 6:
83                          printf("%s,",datos);
84                          att.battery_current=
                                atof(datos);
85                          if(buffer[j-5]=='-')
                                {
86                              att.signo=1;
87                          }
88                          else{
89                              att.signo=0;
90                          }
91                          break;
92                      case 7:
93                          printf("%s,",datos);
94                          att.load_voltage=
                                atof(datos);
95                          break;
96                      case 8:
97                          printf("%s,",datos);
98                          att.temp1=atof(datos
                                );
99                          int temp1 = atof(
                                datos);
100                         break;
101                     case 9:
102                         printf("%s,",datos);
103                         att.temp2=atof(datos
                                );
104                         int temp2 = atof(
                                datos);
105                         break;
106                     case 10:
107                         printf("%s,",datos);
108                         att.temp3=atof(datos
                                );
109                         int temp3 = atof(
                                datos);
110                         break;
```

```
111                     case 11:
112                         printf("%s,",datos);
113                         att.temp4=atof(datos
                               );
114                         int temp4 = atof(
                               datos);
115                         int max = fmax(fmax(
                               temp1, temp2),
                               fmax(temp3, temp4
                               ));
116                         att.top_temp=max;
117                         break;
118                     case 12:
119                         printf("%s,",datos);
120                         att.target_temp=atof
                               (datos);
121                         break;
122                     case 13:
123                         printf("%s,",datos);
124                         att.board_temp=atof(
                               datos);
125                         break;
126                     case 14:
127                         printf("%s,",datos);
128                         att.pila=atof(datos)
                               ;
129                         break;
130                     case 15:
131                         printf("%s,",datos);
132                         att.h2_pressure=atof
                               (datos);
133                         break;
134                     case 16:
135                         printf("%s,",datos);
136                         att.fan_speed=atof(
                               datos);
137                         break;
138                     case 17:
139                         printf("%s,\n",datos
                               );
```

```
                                    att.state=atof(datos);
                                    break;
                                default:
                                    printf("Opción inválida\n");
                            }
                            caso++;
                            memset(datos, 0, 55);
                            contador=0;
                        }
                    }
                    orb_publish(ORB_ID(fuelcellbatt), att_pub, &att);
                }
                buffIndex = 0;  // Reiniciar el índice del vector buff
            }
            else {
                buffer[buffIndex] = receivedChar;
                buffIndex++;
            }
        }
    }

    PX4_INFO("Fuera del while");
    // Cerrar el puerto serie
    close(serialPort);
    return 0;
}
```

Kconfig:

```
menuconfig EXAMPLES_PRUEBA
  bool "prueba"
  default n
  ---help---
    Enable support for prueba
```

CMakeLists:

```
px4_add_module(
  MODULE examples__prueba
```

```
3  MAIN prueba
4  SRCS
5    prueba.c
6  DEPENDS
7  )
```

El código en cuestión tiene como objetivo principal abrir el puerto *ttys6* de la placa a una velocidad de transferencia de datos de 19200 baudios. Este puerto es utilizado para recibir información proveniente de la pila de combustible A1000LV de H3Dynamics. Además, se inicializa un mensaje uORB, que es un sistema de comunicación entre procesos utilizado en el entorno PX4, con el fin de enviar la información recolectada posteriormente. A continuación, se entra en un bucle infinito (while true) donde se lee el puerto serial y se almacena cada lectura en una cadena de caracteres llamada *buffer*. Esta cadena se procesa y se reinicia cada vez que se recibe un carácter de retorno de carro.

En este punto es donde comienza la interpretación del mensaje recibido. Si la longitud de la cadena *buffer* es menor a 100 caracteres, se considera un mensaje de inicialización y no se realiza ningún procesamiento adicional. Por otro lado, si la longitud supera los 100 caracteres, se trata de un mensaje de estado de la pila y se procede a separar los valores que están intercalados por comas. Se extraen únicamente los números de cada mensaje, ya que son los datos relevantes para nuestro análisis. Estos valores numéricos se asocian a variables específicas según su posición en el mensaje. Por ejemplo, en el caso del mensaje «FCS=50.4V,00.43A,025.1W,...», se tomaría el valor «FCS=50.4V» y se asignaría a la variable *stack_ voltage*, luego se procesaría «00.43A» y se asociaría con la variable *load_ currente*. Este proceso se repite para cada variable y mensaje correspondiente.

Importante ahora hace falta añadir esta aplicación a la compilación de la placa, ya que no todos lo archivos de este proyecto se compilan para subir en la placa. Es por eso que escribimos `make px4_fmu-v2 boardconfig` en la consola y se abrira un menú donde se debe navegar por el hasta la carpeta de «examples» y activar la aplicación para la compilación.

2.5. Edición del mensaje MAVLink

En este contexto, se empleará el mensaje Mavlink *Battery_ status*, el cual está predefinido en el propio sistema. Se utilizarán las variables incluidas en este mensaje para representar los datos del mensaje de la pila de hidrógeno. Aunque los

```
(Top)
                                        PX4 Firmware Configuration
    *** Vendor: px4 ***
    *** Model: fmu-v2 ***
    *** Label: default ***
    Toolchain  --->
[ ] Testing
[ ] Ethernet
[ ] Crypto support
[ ] Memory protection
    Serial ports  --->
    drivers  --->
    modules  --->
    systemcmds  --->
    examples  --->

[Space/Enter] Toggle/enter  [ESC] Leave menu           [S] Save
[O] Load                    [?] Symbol info            [/] Jump to symbol
[F] Toggle show-help mode   [C] Toggle show-name mode  [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```

Figura 2.8: Menú de la configuración de la placa

nombres de las variables no reflejen directamente su significado, se implementará un widget más adelante que mostrará la correspondencia entre cada número y su respectiva variable, así como las unidades asociadas. Además, se calcularán nuevas características basadas en los datos obtenidos.

A continuación, se presenta la edición realizada en el mensaje para enviarlo mediante el protocolo Mavlink:

```
#ifndef BATTERY_STATUS_HPP
#define BATTERY_STATUS_HPP

#include <uORB/topics/fuelcellbatt.h>

class MavlinkStreamBatteryStatus : public MavlinkStream
{
public:
  static MavlinkStream *new_instance(Mavlink *mavlink) {
      return new MavlinkStreamBatteryStatus(mavlink); }

  static constexpr const char *get_name_static() {
     return "BATTERY_STATUS"; }
  static constexpr uint16_t get_id_static() { return
     MAVLINK_MSG_ID_BATTERY_STATUS; }

  const char *get_name() const override { return
     get_name_static(); }
  uint16_t get_id() override { return get_id_static(); }

  unsigned get_size() override
  {
    return MAVLINK_MSG_ID_BATTERY_STATUS_LEN +
```

```
        MAVLINK_NUM_NON_PAYLOAD_BYTES;
20    }
21
22  private:
23    explicit MavlinkStreamBatteryStatus(Mavlink *mavlink)
        : MavlinkStream(mavlink) {}
24
25    uORB::Subscription _fuelcellbatt_sub{ORB_ID(
        fuelcellbatt)};
26
27    bool send() override
28    {
29      mavlink_battery_status_t bat_msg{};
30      bool updated = false;
31      if(_fuelcellbatt_sub.advertised()){
32
33        fuelcellbatt_s fuelcellbatt{};
34        if(_fuelcellbatt_sub.copy(&fuelcellbatt)){
35
36          bat_msg.voltages_ext[0]=fuelcellbatt.pila;
37          bat_msg.voltages_ext[1]=fuelcellbatt.h2_pressure
              ;
38          bat_msg.voltages_ext[2]=fuelcellbatt.fan_speed;
39          bat_msg.mode=fuelcellbatt.state;
40          bat_msg.temperature=fuelcellbatt.top_temp;
41          bat_msg.voltages[0]=fuelcellbatt.stack_voltage;
42          bat_msg.voltages[1]=fuelcellbatt.battery_voltage
              ;
43          bat_msg.voltages[2]=fuelcellbatt.load_voltage;
44          bat_msg.voltages[3]=fuelcellbatt.temp1;
45          bat_msg.voltages[4]=fuelcellbatt.temp2;
46          bat_msg.voltages[5]=fuelcellbatt.temp3;
47          bat_msg.voltages[6]=fuelcellbatt.temp4;
48          bat_msg.voltages[7]=fuelcellbatt.board_temp;
49          bat_msg.voltages[8]=fuelcellbatt.target_temp;
50          bat_msg.current_battery=fuelcellbatt.
              load_currente;
51          bat_msg.time_remaining=fuelcellbatt.
              battery_current;
52          bat_msg.current_consumed =fuelcellbatt.power;
```

```
                              bat_msg.energy_consumed =
                                  fuelcellbatt.energy;
            double mid=100*(-503.622+584.100*fuelcellbatt.
                battery_voltage/80.0-             252.554*pow(
                fuelcellbatt.battery_voltage/80.0,2)+48.204*
                pow(fuelcellbatt.battery_voltage/80.0,3)
                -3.422*pow(fuelcellbatt.battery_voltage
                /80.0,4));

            if (mid<100){
               if(mid<0){
                  bat_msg.battery_remaining=0;
               }
               bat_msg.battery_remaining=mid;
            }
            else{
               bat_msg.battery_remaining=100;
            }
            bat_msg.type = fuelcellbatt.signo;
                              updated = true;
         }
      }
      if (updated) {
         mavlink_msg_battery_status_send_struct(_mavlink->
             get_channel(), &bat_msg);
         return true;
      }
      return false;
   }
};
#endif // BATTERY_STATUS_HPP
```

2.6. Compilar y subir

Ahora solo falta compilarlo, que se puede hacer simplemente escribiendo el comando

```
UbuntuMachine:~/PX4-Autopilot$ make px4\_fmu-v2
```

y cuando termine de compilar, pondrá cuanto tamaño ocupa y donde se ha guardado, que suele ser en «/PX4-Autopilot/build/px4_fmu-v2_default». Y después

usando un GCS, en este caso QGrounControl, se conecta la placa Pixhawk 2 al ordenador y se sube el filmware. Después se debe ir a la consola en el GCS y escribir

```
pablo@DESKTOP-EDMKP6Q:~/PX4-Autopilot$ make px4_fmu-v2
[13/15] Linking CXX executable px4_fmu-v2_default.elf
Memory region         Used Size  Region Size  %age Used
           flash:     1031641 B      1008 KB     99.95%
            sram:       22932 B       192 KB     11.66%
          ccsram:           0 GB        64 KB      0.00%
[15/15] Creating /home/pablo/PX4-Autopilot/build/px4_fmu-v2_default/px4_fmu-v2_default.px4
```

Figura 2.9: Compilación y output

«prueba» para inicializar la aplicación, si se desea que este activa desde el startup, solo se debe ir al archivo «/PX4-Autopilot/ROMFS/px4fmu_common/init.d/rcS» y en la linea 535 [para evitar conflictos con otras funcionalides], escribir el nombre de la aplicación, como si de la consola de comandos se tratara.

Capítulo 3

Qgrouncontrol

En esta sección se va a enseñar como editar para leer todo el mensaje MAVLink y mostrar estos datos en un widget en la pantalla principal de Qgroundcontrol.

Lo primero ser capaz de compilar la aplicación. Para ello es necesario instalar QT creator, importante que sea la versión **5.15** y abrir desde aquí el repositorio de Qgroundontrol clonado a windows desde «https://github.com/mavlink/qgroundcontrol».

```
C:Desktop> git clone https://github.com/mavlink/qgroundcontrol
    --recursive
```

Una vez hecho esto, clique en el «Play» o pulse «Ctrl+r», para comprobar que todo esta correctamente instalado

3.1. Github Hydrone Qgrouncontrol

Al igual que en el capítulo anterior, hay una versión en github con todo lo que se comenta a continuación ya realizado y solo hace falta su compilación.Si quiere hacerse con el comando desde el cmd habría que descargarse la aplicación git «https://git-scm.com/download/win», si no con descargarse todos los archivos en desde github funciona igual.

```
C:Desktop> git clone https://github.com/sergarro/
    qgroundcontrol_HyDrone.git --recursive
```

3.2. Modificación de los archivos

Para leer todos los mensaje del MAVLink necesarios, primero es editar el archivo «BatteryFact.json», añadiendo las variables necesarias para trabajar con estos datos, en este caso se han añadido estas:

```
{
      "name":                "mode",
      "shortDesc":           "mode",
      "type":                "double",
      "decimalPlaces":       0
},
{
      "name":                "stackv",
      "shortDesc":           "stackv",
      "type":                "double",
      "decimalPlaces":       0
},
{
      "name":                "loadcurrent",
      "shortDesc":           "loadcurrent",
      "type":                "double",
      "decimalPlaces":       0
},
{
      "name":                "currentbattery",
      "shortDesc":           "currentbattery",
      "type":                "double",
      "decimalPlaces":       0
},
{
      "name":                "battvolt",
      "shortDesc":           "battvolt",
      "type":                "double",
      "decimalPlaces":       0
},
{
      "name":                "loadvolt",
      "shortDesc":           "loadvolt",
      "type":                "double",
      "decimalPlaces":       0
},
{
      "name":                "h2press1",
      "shortDesc":           "h2press1",
      "type":                "double",
```

```
        "decimalPlaces":     0
},
{
        "name":              "h2press2",
        "shortDesc":         "h2press2",
        "type":              "double",
        "decimalPlaces":     0
},
{
        "name":              "fanspeed",
        "shortDesc":         "fanspeed",
        "type":              "double",
        "decimalPlaces":     0
},
{
        "name":              "energy",
        "shortDesc":         "energy",
        "type":              "double",
        "decimalPlaces":     0
}
```

Después de esto, se editan los archivos «vehiclebatteryfactgroup.cc» y «vehiclebatteryfactgroup.h» para tener estas nuevas variables introducidas en el mensaje *Fact*. Cabe destacar que se han añadido nuestras variables a la sintaxis ya creada, y se han añadido lineas en el «vehiclebatteryfactgroup.cc», concretamente las lineas 161-166, para sacar de esos vectores de variables los que interesan.

vehiclebatteryfactgroup.cc:

```
#include "VehicleBatteryFactGroup.h"
#include "QmlObjectListModel.h"
#include "Vehicle.h"

const char* VehicleBatteryFactGroup::
    _batteryFactGroupNamePrefix     = "battery";

const char* VehicleBatteryFactGroup::_batteryIdFactName
                    = "id";
const char* VehicleBatteryFactGroup::
    _batteryFunctionFactName        = "batteryFunction";
const char* VehicleBatteryFactGroup::
    _batteryTypeFactName            = "batteryType";
```

```
10 const char* VehicleBatteryFactGroup::_voltageFactName
                    = "voltage";
11 const char* VehicleBatteryFactGroup::
     _percentRemainingFactName       = "percentRemaining";
12 const char* VehicleBatteryFactGroup::
     _mahConsumedFactName            = "mahConsumed";
13 const char* VehicleBatteryFactGroup::_currentFactName
                    = "current";
14 const char* VehicleBatteryFactGroup::
     _temperatureFactName            = "temperature";
15 const char* VehicleBatteryFactGroup::
     _instantPowerFactName           = "instantPower";
16 const char* VehicleBatteryFactGroup::
     _timeRemainingFactName          = "timeRemaining";
17 const char* VehicleBatteryFactGroup::
     _timeRemainingStrFactName       = "timeRemainingStr";
18 const char* VehicleBatteryFactGroup::
     _chargeStateFactName            = "chargeState";
19 const char* VehicleBatteryFactGroup::_modeFactName
                       = "mode";
20 const char* VehicleBatteryFactGroup::_stackvFactName
                     = "stackv";
21 const char* VehicleBatteryFactGroup::
     _currentbatteryFactName         = "currentbattery";
22 const char* VehicleBatteryFactGroup::_energyFactName
                     = "energy";
23 const char* VehicleBatteryFactGroup::_battvoltFactName
                     = "battvolt";
24 const char* VehicleBatteryFactGroup::_loadvoltFactName
                     = "loadvolt";
25 const char* VehicleBatteryFactGroup::_h2press1FactName
                     = "h2press1";
26 const char* VehicleBatteryFactGroup::_h2press2FactName
                     = "h2press2";
27 const char* VehicleBatteryFactGroup::_fanspeedFactName
                     = "fanspeed";
28
29 const char* VehicleBatteryFactGroup::_settingsGroup =
                          "Vehicle.battery";
30
```

```
31 VehicleBatteryFactGroup::VehicleBatteryFactGroup(uint8_t
       batteryId, QObject* parent)
32     : FactGroup                   (1000, ":/json/Vehicle/
       BatteryFact.json", parent)
33     , _batteryIdFact          (0, _batteryIdFactName,
                               FactMetaData::valueTypeUint8)
34     , _batteryFunctionFact    (0, _batteryFunctionFactName
       ,               FactMetaData::valueTypeUint8)
35     , _batteryTypeFact        (0, _batteryTypeFactName,
                           FactMetaData::valueTypeUint8)
36     , _voltageFact            (0, _voltageFactName,
                                   FactMetaData::valueTypeDouble)
37     , _currentFact            (0, _currentFactName,
                                   FactMetaData::valueTypeDouble)
38     , _mahConsumedFact        (0, _mahConsumedFactName,
                           FactMetaData::valueTypeDouble)
39     , _temperatureFact        (0, _temperatureFactName,
                           FactMetaData::valueTypeDouble)
40     , _percentRemainingFact (0,
       _percentRemainingFactName,           FactMetaData
       ::valueTypeDouble)
41     , _timeRemainingFact      (0, _timeRemainingFactName,
                       FactMetaData::valueTypeDouble)
42     , _timeRemainingStrFact (0,
       _timeRemainingStrFactName,           FactMetaData
       ::valueTypeString)
43     , _chargeStateFact        (0, _chargeStateFactName,
                           FactMetaData::valueTypeUint8)
44     , _instantPowerFact       (0, _instantPowerFactName,
                         FactMetaData::valueTypeDouble)
45     , _modeFact               (0, _modeFactName,
                                      FactMetaData::
       valueTypeDouble)
46     , _stackvFact             (0, _stackvFactName,
                                    FactMetaData::valueTypeDouble)
47     , _currentbatteryFact     (0, _currentbatteryFactName,
                       FactMetaData::valueTypeDouble)
48     , _energyFact             (0, _energyFactName,
                                    FactMetaData::valueTypeDouble)
49     , _battvoltFact             (0, _battvoltFactName,
```

```
                               FactMetaData::valueTypeDouble)
50     , _loadvoltFact              (0, _loadvoltFactName,
                               FactMetaData::valueTypeDouble)
51     , _h2press1Fact              (0, _h2press1FactName,
                               FactMetaData::valueTypeDouble)
52     , _h2press2Fact              (0, _h2press2FactName,
                               FactMetaData::valueTypeDouble)
53     , _fanspeedFact              (0, _fanspeedFactName,
                               FactMetaData::valueTypeDouble)
54 {
55     _addFact(&_batteryIdFact,
          _batteryIdFactName);
56     _addFact(&_batteryFunctionFact,
          _batteryFunctionFactName);
57     _addFact(&_batteryTypeFact,
          _batteryTypeFactName);
58     _addFact(&_voltageFact,
          _voltageFactName);
59     _addFact(&_currentFact,
          _currentFactName);
60     _addFact(&_mahConsumedFact,
          _mahConsumedFactName);
61     _addFact(&_temperatureFact,
          _temperatureFactName);
62     _addFact(&_percentRemainingFact,
          _percentRemainingFactName);
63     _addFact(&_timeRemainingFact,
          _timeRemainingFactName);
64     _addFact(&_timeRemainingStrFact,
          _timeRemainingStrFactName);
65     _addFact(&_chargeStateFact,
          _chargeStateFactName);
66     _addFact(&_instantPowerFact,
          _instantPowerFactName);
67     _addFact(&_modeFact,
          _modeFactName);
68     _addFact(&_stackvFact,
          _stackvFactName);
69     _addFact(&_currentbatteryFact,
          _currentbatteryFactName);
```

```
70      _addFact(&_energyFact,
           _energyFactName);
71      _addFact(&_battvoltFact,
           _battvoltFactName);
72      _addFact(&_loadvoltFact,
           _loadvoltFactName);
73      _addFact(&_h2press1Fact,
           _h2press1FactName);
74      _addFact(&_h2press2Fact,
           _h2press2FactName);
75      _addFact(&_fanspeedFact,
           _fanspeedFactName);
76
77      _batteryIdFact.setRawValue           (batteryId);
78      _batteryFunctionFact.setRawValue     (
           MAV_BATTERY_FUNCTION_UNKNOWN);
79      _batteryTypeFact.setRawValue         (
           MAV_BATTERY_TYPE_UNKNOWN);
80      _voltageFact.setRawValue             (qQNaN());
81      _currentFact.setRawValue             (qQNaN());
82      _mahConsumedFact.setRawValue         (qQNaN());
83      _temperatureFact.setRawValue         (qQNaN());
84      _percentRemainingFact.setRawValue    (qQNaN());
85      _timeRemainingFact.setRawValue       (qQNaN());
86      _chargeStateFact.setRawValue         (
           MAV_BATTERY_CHARGE_STATE_UNDEFINED);
87      _instantPowerFact.setRawValue        (qQNaN());
88      _modeFact.setRawValue                (qQNaN());
89      _stackvFact.setRawValue              (qQNaN());
90      _currentbatteryFact.setRawValue      (qQNaN());
91      _energyFact.setRawValue              (qQNaN());
92      _battvoltFact.setRawValue            (qQNaN());
93      _loadvoltFact.setRawValue            (qQNaN());
94      _h2press1Fact.setRawValue            (qQNaN());
95      _h2press2Fact.setRawValue            (qQNaN());
96      _fanspeedFact.setRawValue            (qQNaN());
97
98      connect(&_timeRemainingFact, &Fact::rawValueChanged,
            this, &VehicleBatteryFactGroup::
           _timeRemainingChanged);
```

```
99  }
100 
101 void VehicleBatteryFactGroup::
        handleMessageForFactGroupCreation(Vehicle* vehicle,
        mavlink_message_t& message)
102 {
103     switch (message.msgid) {
104     case MAVLINK_MSG_ID_HIGH_LATENCY:
105     case MAVLINK_MSG_ID_HIGH_LATENCY2:
106         _findOrAddBatteryGroupById(vehicle, 0);
107         break;
108     case MAVLINK_MSG_ID_BATTERY_STATUS:
109     {
110         mavlink_battery_status_t batteryStatus;
111         mavlink_msg_battery_status_decode(&message, &
               batteryStatus);
112         _findOrAddBatteryGroupById(vehicle,
               batteryStatus.id);
113     }
114         break;
115     }
116 }
117 void VehicleBatteryFactGroup::handleMessage(Vehicle*
        vehicle, mavlink_message_t& message)
118 {
119     switch (message.msgid) {
120     case MAVLINK_MSG_ID_HIGH_LATENCY:
121         _handleHighLatency(vehicle, message);
122         break;
123     case MAVLINK_MSG_ID_HIGH_LATENCY2:
124         _handleHighLatency2(vehicle, message);
125         break;
126     case MAVLINK_MSG_ID_BATTERY_STATUS:
127         _handleBatteryStatus(vehicle, message);
128         break;
129     }
130 }
131 void VehicleBatteryFactGroup::_handleHighLatency(Vehicle
        * vehicle, mavlink_message_t& message)
132 {
```

```
133     mavlink_high_latency_t highLatency;
134     mavlink_msg_high_latency_decode(&message, &
           highLatency);
135 
136     VehicleBatteryFactGroup* group =
           _findOrAddBatteryGroupById(vehicle, 0);
137     group->percentRemaining()->setRawValue(highLatency.
           battery_remaining == UINT8_MAX ? qQNaN() :
           highLatency.battery_remaining);
138     group->_setTelemetryAvailable(true);
139 }
140 void VehicleBatteryFactGroup::_handleHighLatency2(
       Vehicle* vehicle, mavlink_message_t& message)
141 {
142     mavlink_high_latency2_t highLatency2;
143     mavlink_msg_high_latency2_decode(&message, &
           highLatency2);
144 
145     VehicleBatteryFactGroup* group =
           _findOrAddBatteryGroupById(vehicle, 0);
146     group->percentRemaining()->setRawValue(highLatency2.
           battery == -1 ? qQNaN() : highLatency2.battery);
147     group->_setTelemetryAvailable(true);
148 }
149 void VehicleBatteryFactGroup::_handleBatteryStatus(
       Vehicle* vehicle, mavlink_message_t& message)
150 {
151     mavlink_battery_status_t batteryStatus;
152     mavlink_msg_battery_status_decode(&message, &
           batteryStatus);
153 
154     VehicleBatteryFactGroup* group =
           _findOrAddBatteryGroupById(vehicle, batteryStatus
           .id);
155 
156     double totalVoltage = qQNaN();
157     double stackv = batteryStatus.voltages[0];
158     double battvolt = batteryStatus.voltages[1];
159     double loadvolt = batteryStatus.voltages[2];
160     double h2press1 = batteryStatus.voltages_ext[0];
```

```
161     double h2press2 = batteryStatus.voltages_ext[1];
162     double fanspeed = batteryStatus.voltages_ext[2];
163     for (int i=0; i<10; i++) {
164         double cellVoltage = batteryStatus.voltages[i]
              == UINT16_MAX ? qQNaN() : static_cast<double
              >(batteryStatus.voltages[i]) / 1000.0;
165         if (qIsNaN(cellVoltage)) {
166             break;
167         }
168         if (i == 0) {
169             totalVoltage = cellVoltage;
170         } else {
171             totalVoltage += cellVoltage;
172         }
173     }
174     for (int i=0; i<4; i++) {
175         double cellVoltage = batteryStatus.voltages_ext[
              i] == UINT16_MAX ? qQNaN() : static_cast<
              double>(batteryStatus.voltages_ext[i]) /
              1000.0;
176         if (qIsNaN(cellVoltage)) {
177             break;
178         }
179         totalVoltage += cellVoltage;
180     }
181     group->function()->setRawValue          (
           batteryStatus.battery_function);
182     group->type()->setRawValue              (
           batteryStatus.type);
183     group->temperature()->setRawValue       (
           batteryStatus.temperature == INT16_MAX ?    qQNaN
           () : static_cast<double>(batteryStatus.
           temperature) / 10.0);
184     group->voltage()->setRawValue           (
           totalVoltage);
185     group->current()->setRawValue           (
           batteryStatus.current_battery == -1 ?       qQNaN
           () : static_cast<double>(batteryStatus.
           current_battery) / 100.0);
186     group->mahConsumed()->setRawValue       (
```

```
        batteryStatus.current_consumed == -1  ?    qQNaN
        () : batteryStatus.current_consumed);
187     group->percentRemaining()->setRawValue  (
        batteryStatus.battery_remaining == -1 ?    qQNaN
        () : batteryStatus.battery_remaining);
188     group->timeRemaining()->setRawValue     (
        batteryStatus.time_remaining);
189     group->chargeState()->setRawValue       (
        batteryStatus.charge_state);
190     group->instantPower()->setRawValue      (
        totalVoltage * group->current()->rawValue().
        toDouble());
191     group->mode()->setRawValue              (
        batteryStatus.mode);
192     group->stackv()->setRawValue            (stackv);
193     group->currentbattery()->setRawValue    (
        batteryStatus.current_battery);
194     group->energy()->setRawValue            (
        batteryStatus.energy_consumed);
195     group->battvolt()->setRawValue            (battvolt)
        ;
196     group->loadvolt()->setRawValue            (loadvolt)
        ;
197     group->h2press1()->setRawValue            (h2press1)
        ;
198     group->h2press2()->setRawValue            (h2press2)
        ;
199     group->fanspeed()->setRawValue            (fanspeed)
        ;
200     group->_setTelemetryAvailable(true);
201 }
202
203 VehicleBatteryFactGroup* VehicleBatteryFactGroup::
    _findOrAddBatteryGroupById(Vehicle* vehicle, uint8_t
    batteryId)
204 {
205     QmlObjectListModel* batteries = vehicle->batteries()
        ;
206
207     // We maintain the list in order sorted by battery
```

```
        id so the ui shows them sorted.
208     for (int i=0; i<batteries->count(); i++) {
209         VehicleBatteryFactGroup* group = batteries->
            value<VehicleBatteryFactGroup*>(i);
210         int listBatteryId = group->id()->rawValue().
            toInt();
211         if (listBatteryId >  batteryId) {
212             VehicleBatteryFactGroup* newBatteryGroup =
                new VehicleBatteryFactGroup(batteryId,
                batteries);
213             batteries->insert(i, newBatteryGroup);
214             vehicle->_addFactGroup(newBatteryGroup,
                QStringLiteral("%1%2").arg(
                _batteryFactGroupNamePrefix).arg(
                batteryId));
215             return newBatteryGroup;
216         } else if (listBatteryId == batteryId) {
217             return group;
218         }
219     }
220
221     VehicleBatteryFactGroup* newBatteryGroup = new
        VehicleBatteryFactGroup(batteryId, batteries);
222     batteries->append(newBatteryGroup);
223     vehicle->_addFactGroup(newBatteryGroup,
        QStringLiteral("%1%2").arg(
        _batteryFactGroupNamePrefix).arg(batteryId));
224
225     return newBatteryGroup;
226 }
227
228 void VehicleBatteryFactGroup::_timeRemainingChanged(
    QVariant value)
229 {
230     if (qIsNaN(value.toDouble())) {
231         _timeRemainingStrFact.setRawValue("--:--:--");
232     } else {
233         int totalSeconds     = value.toInt();
234         int hours            = totalSeconds / 3600;
235         int minutes          = (totalSeconds % 3600) /
```

```
            60;
236         int seconds          = totalSeconds % 60;
237
238         _timeRemainingStrFact.setRawValue(QString::
            asprintf("%02dH:%02dM:%02dS", hours, minutes,
             seconds));
239     }
240 }
```

vehiclebatteryfactgroup.h:

```
1       #pragma once
2
3   #include "FactGroup.h"
4   #include "QGCMAVLink.h"
5
6   class Vehicle;
7
8   class VehicleBatteryFactGroup : public FactGroup
9   {
10      Q_OBJECT
11
12  public:
13      VehicleBatteryFactGroup(uint8_t batteryId, QObject*
           parent = nullptr);
14
15      Q_PROPERTY(Fact* id                  READ id
                      CONSTANT)
16      Q_PROPERTY(Fact* function            READ function
                 CONSTANT)
17      Q_PROPERTY(Fact* type                READ type
                     CONSTANT)
18      Q_PROPERTY(Fact* temperature         READ temperature
               CONSTANT)
19      Q_PROPERTY(Fact* voltage             READ voltage
                  CONSTANT)
20      Q_PROPERTY(Fact* current             READ current
                  CONSTANT)
21      Q_PROPERTY(Fact* mahConsumed         READ mahConsumed
               CONSTANT)
22      Q_PROPERTY(Fact* percentRemaining    READ
           percentRemaining   CONSTANT)
```

```
23    Q_PROPERTY(Fact* timeRemaining      READ
         timeRemaining       CONSTANT)
24    Q_PROPERTY(Fact* timeRemainingStr   READ
         timeRemainingStr    CONSTANT)
25    Q_PROPERTY(Fact* chargeState        READ chargeState
                 CONSTANT)
26    Q_PROPERTY(Fact* instantPower       READ
         instantPower        CONSTANT)
27    Q_PROPERTY(Fact* mode               READ mode
                       CONSTANT)
28    Q_PROPERTY(Fact* stackv             READ stackv
                     CONSTANT)
29    Q_PROPERTY(Fact* currentbattery     READ
         currentbattery      CONSTANT)
30    Q_PROPERTY(Fact* energy             READ energy
                     CONSTANT)
31    Q_PROPERTY(Fact* battvolt             READ battvolt
                     CONSTANT)
32    Q_PROPERTY(Fact* loadvolt             READ loadvolt
                     CONSTANT)
33    Q_PROPERTY(Fact* h2press1             READ h2press1
                     CONSTANT)
34    Q_PROPERTY(Fact* h2press2             READ h2press2
                     CONSTANT)
35    Q_PROPERTY(Fact* fanspeed             READ fanspeed
                     CONSTANT)
36
37    Fact* id                          () { return &
         _batteryIdFact; }
38    Fact* function                    () { return &
         _batteryFunctionFact; }
39    Fact* type                        () { return &
         _batteryTypeFact; }
40    Fact* voltage                     () { return &
         _voltageFact; }
41    Fact* percentRemaining            () { return &
         _percentRemainingFact; }
42    Fact* mahConsumed                 () { return &
         _mahConsumedFact; }
43    Fact* current                     () { return &
```

```
        _currentFact; }
44      Fact* temperature                    () { return &
          _temperatureFact; }
45      Fact* instantPower                   () { return &
          _instantPowerFact; }
46      Fact* timeRemaining                  () { return &
          _timeRemainingFact; }
47      Fact* timeRemainingStr               () { return &
          _timeRemainingStrFact; }
48      Fact* chargeState                    () { return &
          _chargeStateFact; }
49      Fact* mode                           () { return &
          _modeFact; }
50      Fact* stackv                         () { return &
          _stackvFact; }
51      Fact* currentbattery                 () { return &
          _currentbatteryFact; }
52      Fact* energy                         () { return &
          _energyFact; }
53      Fact* battvolt                         () { return &
          _battvoltFact; }
54      Fact* loadvolt                         () { return &
          _loadvoltFact; }
55      Fact* h2press1                         () { return &
          _h2press1Fact; }
56      Fact* h2press2                         () { return &
          _h2press2Fact; }
57      Fact* fanspeed                         () { return &
          _fanspeedFact; }
58
59      static const char* _batteryIdFactName;
60      static const char* _batteryFunctionFactName;
61      static const char* _batteryTypeFactName;
62      static const char* _temperatureFactName;
63      static const char* _voltageFactName;
64      static const char* _currentFactName;
65      static const char* _mahConsumedFactName;
66      static const char* _percentRemainingFactName;
67      static const char* _timeRemainingFactName;
68      static const char* _timeRemainingStrFactName;
```

```
69      static const char* _chargeStateFactName;
70      static const char* _instantPowerFactName;
71      static const char* _modeFactName;
72      static const char* _stackvFactName;
73      static const char* _currentbatteryFactName;
74      static const char* _energyFactName;
75      static const char* _battvoltFactName;
76      static const char* _loadvoltFactName;
77      static const char* _fanspeedFactName;
78      static const char* _h2press1FactName;
79      static const char* _h2press2FactName;
80
81      static const char* _settingsGroup;
82
83      /// Creates a new fact group for the battery id as
           needed and updates the Vehicle with it
84      static void handleMessageForFactGroupCreation(
           Vehicle* vehicle, mavlink_message_t& message);
85
86      // Overrides from FactGroup
87      void handleMessage(Vehicle* vehicle,
           mavlink_message_t& message) override;
88
89  private slots:
90      void _timeRemainingChanged(QVariant value);
91
92  private:
93      static void                      _handleHighLatency
                    (Vehicle* vehicle, mavlink_message_t&
           message);
94      static void                      _handleHighLatency2
                   (Vehicle* vehicle, mavlink_message_t&
           message);
95      static void                      _handleBatteryStatus
                   (Vehicle* vehicle, mavlink_message_t&
           message);
96      static VehicleBatteryFactGroup*
           _findOrAddBatteryGroupById (Vehicle* vehicle,
           uint8_t batteryId);
97
```

```
98      Fact                _batteryIdFact;
99      Fact                _batteryFunctionFact;
100     Fact                _batteryTypeFact;
101     Fact                _voltageFact;
102     Fact                _currentFact;
103     Fact                _mahConsumedFact;
104     Fact                _temperatureFact;
105     Fact                _percentRemainingFact;
106     Fact                _timeRemainingFact;
107     Fact                _timeRemainingStrFact;
108     Fact                _chargeStateFact;
109     Fact                _instantPowerFact;
110     Fact                _modeFact;
111     Fact                _stackvFact;
112     Fact                _currentbatteryFact;
113     Fact                _energyFact;
114     Fact                _battvoltFact;
115     Fact                _loadvoltFact;
116     Fact                _h2press1Fact;
117     Fact                _h2press2Fact;
118     Fact                _fanspeedFact;
119
120
121     static const char* _batteryFactGroupNamePrefix;
122 };
```

Y por último, solo queda introducir todos estos datos en un widget en la pantalla principal. Por suerte, este software esta preparado para este tipo de personalizaciones y tiene un archivo dedicado unica y exclusivamente a esto, que se llama «Flyviewcustomlayer.qml»:

```
1  import QtQuick                  2.12
2  import QtQuick.Controls         2.4
3  import QtQuick.Dialogs          1.3
4  import QtQuick.Layouts          1.12
5
6  import QtLocation               5.3
7  import QtPositioning            5.3
8  import QtQuick.Window           2.2
9  import QtQml.Models             2.1
10
11 import QGroundControl               1.0
```

```
12 import QGroundControl.Airspace      1.0
13 import QGroundControl.Airmap        1.0
14 import QGroundControl.Controllers   1.0
15 import QGroundControl.Controls      1.0
16 import QGroundControl.FactSystem    1.0
17 import QGroundControl.FlightDisplay 1.0
18 import QGroundControl.FlightMap     1.0
19 import QGroundControl.Palette       1.0
20 import QGroundControl.ScreenTools   1.0
21 import QGroundControl.Vehicle       1.0
22
23 import MAVLink                      1.0
24
25
26 Item {
27     id: _root
28
29     property var parentToolInsets               // These
           insets tell you what screen real estate is
          available for positioning the controls in your
          overlay
30     property var totalToolInsets:   _toolInsets // These
           are the insets for your custom overlay additions
31     property var mapControl
32
33     QGCToolInsets {
34         id:                     _toolInsets
35         leftEdgeCenterInset:    0
36         leftEdgeTopInset:           0
37         leftEdgeBottomInset:        0
38         rightEdgeCenterInset:   0
39         rightEdgeTopInset:          0
40         rightEdgeBottomInset:       0
41         topEdgeCenterInset:       0
42         topEdgeLeftInset:           0
43         topEdgeRightInset:          0
44         bottomEdgeCenterInset:    0
45         bottomEdgeLeftInset:        0
46         bottomEdgeRightInset:       0
47     }
```

```
48      property int batteryMode: batteryStatusMessage.mode
49      Rectangle {
50          id: batt
51          width: 200
52          height: 400
53          color: Qt.rgba(0, 0, 0, 0.5)
54          radius: 20
55          anchors.bottom: _root.bottom
56          anchors.right: _root.right
57          anchors.bottomMargin: 10
58          anchors.rightMargin: 10
59
60          ColumnLayout {
61              id:                    mainLayout
62              anchors.margins:       ScreenTools.
                    defaultFontPixelWidth
63              anchors.top:           parent.top
64              anchors.right:         parent.right
65              spacing:               ScreenTools.
                    defaultFontPixelHeight
66
67              QGCLabel {
68                  Layout.alignment:   Qt.AlignCenter
69                  text:               qsTr("Hidrogen␣
                        Battery␣Status")
70                  font.family:        ScreenTools.
                        demiboldFontFamily
71              }
72
73              RowLayout {
74                  spacing: ScreenTools.
                        defaultFontPixelWidth
75
76                  ColumnLayout {
77                      Repeater {
78                          model: _activeVehicle ?
                                _activeVehicle.batteries : 0
79
80                          ColumnLayout {
81                              spacing: 0
```

```
82
83                      property var batteryValuesAvailable: nameAvailableLoader.item
84
85                      Loader {
86                          id: nameAvailableLoader
87                          sourceComponent: batteryValuesAvailableComponent
88
89                          property var battery: object
90                      }
91
92
93                      QGCLabel { text: qsTr("Op␣State") }
94                      QGCLabel { text: qsTr("␣␣␣␣␣␣␣␣") }
95
96                      QGCLabel { text: qsTr("Temp␣max")}
97                      QGCLabel { text: qsTr("Stack␣V")}
98                      QGCLabel { text: qsTr("Load␣Current")}
99                      QGCLabel { text: qsTr("Power")}
100                     QGCLabel { text: qsTr("Energy")}
101                     QGCLabel { text: qsTr("Batt␣V")}
102                     QGCLabel { text: qsTr("Batt␣Curr")}
103                     QGCLabel { text: qsTr("Load␣V")}
104                     QGCLabel { text: qsTr("H2␣pres(1)")}
```

```
105                                 QGCLabel { text: qsTr("H2␣pres(2)")}
106                                 QGCLabel { text: qsTr("Fan␣speed")}
107                                 QGCLabel { text: qsTr("Batt␣%")}
108                                 QGCLabel { text: qsTr("H2␣%")}
109
110                             }
111                         }
112                     }
113
114                     ColumnLayout {
115                         Repeater {
116                             model: _activeVehicle ? _activeVehicle.batteries : 0
117
118                             ColumnLayout {
119                                 spacing: 0
120
121                                 property var batteryValuesAvailable: valueAvailableLoader.item
122
123                                 Loader {
124                                     id: valueAvailableLoader
125                                     sourceComponent: batteryValuesAvailableComponent
126
127                                     property var battery: object
128                                 }
129
130                                 //QGCLabel { text: "" }
131
132                                 //QGCLabel { text: object.mode.valueString  }
```

```
133                                 QGCLabel {
134                                     text: {
135                                         if (object.mode.
                                            valueString === "
                                            0") {
136                                             return "Startup"
                                                ;
137                                         } else if (object.
                                            mode.valueString
                                            === "1") {
138                                             return "Startup"
                                                ;
139                                         } else if (object.
                                            mode.valueString
                                            === "2") {
140                                             return "Startup"
                                                ;
141                                         } else if (object.
                                            mode.valueString
                                            === "3") {
142                                             return "Startup"
                                                ;
143                                         } else if (object.
                                            mode.valueString
                                            === "4") {
144                                             return "Startup"
                                                ;
145                                         } else if (object.
                                            mode.valueString
                                            === "5") {
146                                             return "Startup"
                                                ;
147                                         } else if (object.
                                            mode.valueString
                                            === "6") {
148                                             return "
                                                Operation";
149                                         } else if (object.
                                            mode.valueString
                                            === "7") {
```

```
150                                     return "
                                           Operation";
151                                 } else if (object.
                                       mode.valueString
                                       === "8") {
152                                     return "Shutdown
                                           ";
153                                 } else if (object.
                                       mode.valueString
                                       === "9") {
154                                     return "Shutdown
                                           ";
155                                 } else if (object.
                                       mode.valueString
                                       === "10") {
156                                     return "ERROR";
157                                 } else {
158                                     return object.
                                           mode.
                                           valueString;
159                                 }
160                             }
161                         }
162                         QGCLabel {
163                             text: {
164                                 if (object.mode.
                                       valueString === "
                                       0") {
165                                     return "No␣state
                                           ";
166                                 } else if (object.
                                       mode.valueString
                                       === "1") {
167                                     return "Inputs␣
                                           ready";
168                                 } else if (object.
                                       mode.valueString
                                       === "2") {
169                                     return "Loop␣
                                           ready";
```

```
170                     } else if (object.
                           mode.valueString
                           === "3") {
171                         return "Start␣
                               HGS";
172                     } else if (object.
                           mode.valueString
                           === "4") {
173                         return "Checking
                               ␣start␣up";
174                     } else if (object.
                           mode.valueString
                           === "5") {
175                         return "Clean␣
                               Stack";
176                     } else if (object.
                           mode.valueString
                           === "6") {
177                         return "
                               Operation";
178                     } else if (object.
                           mode.valueString
                           === "7") {
179                         return "
                               Conditioning"
                               ;
180                     } else if (object.
                           mode.valueString
                           === "8") {
181                         return "Shutdown
                               ";
182                     } else if (object.
                           mode.valueString
                           === "9") {
183                         return "OFF";
184                     } else if (object.
                           mode.valueString
                           === "10") {
185                         return "ERROR";
186                     } else {
```

```
187                                     return object.
                                           mode.
                                           valueString;
188                                 }
189                             }
190                         }
191                         QGCLabel { text: object.
                               temperature.valueString +
                                "␣" + object.temperature
                               .units }
192                         QGCLabel { text: object.
                               stackv.valueString/10 + "
                               ␣V" }
193                         QGCLabel { text: object.
                               currentbattery.
                               valueString/100 + "␣A"  }
194                         QGCLabel { text: object.
                               mahConsumed.valueString
                               /10 + "␣W" }
195                         QGCLabel { text: object.
                               energy.valueString/100 +
                               "␣Wh" }
196                         QGCLabel { text: object.
                               battvolt.valueString/10 +
                                "␣V" }
197                         QGCLabel { text: object.type
                               .rawValue == 1 ? "-"+
                               object.timeRemaining.
                               valueString/10 + "␣A" :
                               object.timeRemaining.
                               valueString/10 + "␣A" }
198                         QGCLabel { text: object.
                               loadvolt.valueString/10 +
                                "␣V" }
199                         QGCLabel { text: object.
                               h2press1.valueString/100
                               + "␣B" }
200                         QGCLabel { text: object.
                               h2press2.valueString/10 +
                                "␣B" }
```

```
201                                                  QGCLabel { text: object.
                                                         fanspeed.valueString/10 +
                                                          "␣%" }
202                                                  QGCLabel { text: object.
                                                         percentRemaining.
                                                         valueString + "␣%" }
203                                                  QGCLabel { text: Math.round
                                                         (100-object.energy.
                                                         valueString/1060) + "␣%"
                                                         }
204                                              }
205                                          }
206                                      }
207                                  }
208                              }
209                          }
210  }
```

Cabe recalcar que no solo se muestran las variables provenientes del mensaje MAVLink, si no que ademas se calculan dos porcentajes de bateria restante, basandose uno en Wh y otro en el voltaje de la pila y ademas se saca en que estado esta la pila [3.1].

Operating States			
Name	Value	Occurs	Description
NO_STATE	0	Startup	Processor starting up
INPUTS_READY	1	Startup	Sensor Check
LOOP_READY	2	Startup	Sensor Check
START_HGS	3	Startup	Check Supply Pressure
CHECKING_STARTUP	4	Startup	Checking Battery Voltage
CLEAN_STACK	5	Startup	Open Supply Solenoid, and check
OPERATION	6	Operation	Fuel Cell Output On, and operating normally
CONDITIONING	7	Operation	Displayed during conditioning
SHUTDOWN	8	Shutdown	Shutting down the fuel cell
OFF	9	Shutdown	Fuel Cell output off
ERROR	10		Error Condition

Figura 3.1: Estados de la pila según el dato enviado por la pila

Ya para terminar, se han editado el archivo «maintoolbat.qml», concretamente la linea 76 para tener en vez del logo del software, se ha colocado el de la UPV, para esto es necesario copiar el archivo .SVG en la carpeta de recursos y en esta linea escribir la ruta a la imagen. Por otra parte en la barra de herramientas superior sale un porcentaje en grande, pero en un principio es el porcentaje calculado mediante el voltaje dado por la pila, pero al interesar más el calculado por los Wh, se ha

editado el archivo «Batteryindicator.qml», las lineas 81-85:

```
if (battery.energy.rawValue < 1) {
    return qsTr("100%")
} else {
    return Math.round(100-battery.energy.rawValue/1060)...
    + battery.percentRemaining.units
}
```

```
19:39:06: Running steps for project qgroundcontrol...
19:39:06: Configuration unchanged, skipping qmake step.
19:39:06: Starting: "C:\Qt\Tools\QtCreator\bin\jom\jom.exe"
19:39:10: The process "C:\Qt\Tools\QtCreator\bin\jom\jom.exe" exited normally.
19:39:10: Elapsed time: 00:04.
```

Figura 3.2: Mensaje al terminar la Compilación

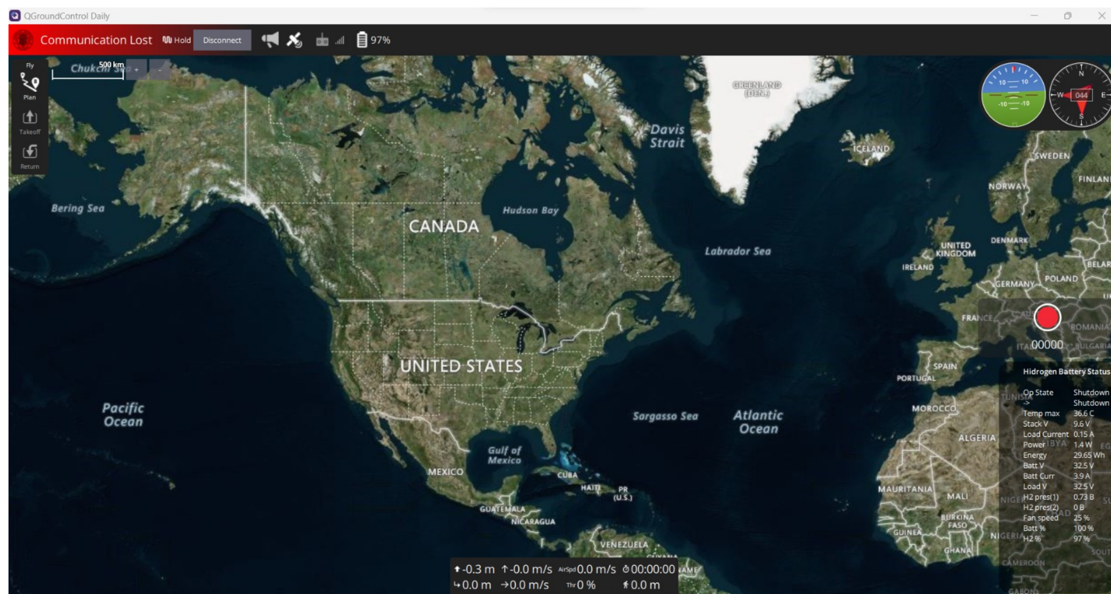

Figura 3.3: GUI del QgroundControl con las nuevas funcionalidades

Como se puede observar esta es una versión editada del software de código abierto *Qgroundcontrol*, se puede ver arriba a la izquierda el logo de la Universidad Politécnica de Valencia, Además abajo a la derecha tenemos un widget con todos los datos importantes para concocer el estado de la batería. Y por ultimo se puede ver que el porcentaje de la barra de herramientas es el calculado con el hidrógeno y no el proveniente de la variable *percentRemaining*.

Capítulo 4

Conclusión

En este documento se ha enseñado, como desde los repositorios originales de px4 y qgroundcontrol, leer un mensaje mandado por la pila de hidrogeno recibido por la placa Pixhawk 2 por el puerto Uart 4, separarlo en las variables correspondientes, mandalras por MAVLink y despues de recibirlas en el QGroundControl y mostrarlas en un widget en la pantalla principal.

UNIVERSIDAD POLITÉCNICA DE VALENCIA (UPV)

ETSID

Diseño e implementación del módulo de comunicaciones entre un sistema de propulsión basado en pila de hidrógeno y el sistema autopiloto para vehículos aéreos no tripulados.

Pablo Ceacero Escrig

Tutor: Sergio García-Nieto Rodríguez
Grado: Ingeniería en Aeroespacial
Curso: 2022/2023

3 de julio de 2023

Trabajo de Fin de Grado [Presupuesto]

Índice general

1. Descripción del proyecto **2**

2. Presupuesto software **3**

3. Presupuesto hardware **4**

4. Presupuesto personal **5**

5. Presupuesto total **6**

Capítulo 1

Descripción del proyecto

Este proyecto de final de carrera versa sobre el desarrollo de un módulo de comunicaciones para establecer una conexión efectiva entre una pila de hidrógeno y el Sistema de Control en Tierra (GCS), con el objetivo de supervisar el rendimiento y el estado de la batería en tiempo real. Se ha utilizado el software PX4 y el protocolo MAVLink para implementar este módulo, estableciendo una comunicación bidireccional sólida entre la pila de hidrógeno y la plataforma de control. Además, se ha personalizado el GCS con el software QGroundControl para mostrar de manera clara y concisa las variables relevantes en la pantalla principal.

En este documento se va a presupuestar el proyecto, desglosando las horas de software, el hardware empleado y las horas de trabajo invertidas.

Capítulo 2

Presupuesto software

En esta sección se va a relatar el presupuesto para el software empleado. En este caso, casi todo el software empleado es de código abierto, pero para el caso del *QT enterprise* se ha dividido su coste anual por un uso aproximado de 1800 h y se ha multiplicado por las horas empleadas en este proyecto [75h] .

Programa	**Tipo de licencia**	**Coste(€)**
Visual Studio Code	permanente	0.00
Qgroundcontrol	permanente	0.00
QT enterprise	Anual(3746 €)	156.1
Overleaf	Permanente	0.00
Total		156.1

Capítulo 3

Presupuesto hardware

En la sección de hardware no se va a tener en cuenta la pila de combustible A1000LV de H3Dynamics, ya que se puede realizar todo el proyecto sin necesidad de ella. Por otra parte se tendrá en cuenta el precio completo de la placa de autopiloto Pixhawk 2 al igual que cables y conectores empleados, y sobre el ordenador empleado [Lenovo Legion 5 15IAH7H, i7, 16GB, 512GB SSD, GeForce RTX 3070 8GB] se considerara un periodo de amortización del ordenador a 5 años, trabajando unas 1800 horas al año y teniendo en cuenta que a este proyecto se ha trabajado en el unas 320 horas.

Hardware	**Coste de adquisición(€)**	**Coste por hora(€/h)**	**Coste total(€)**
Lenovo Legion 5	2000	1.11	355.2
Pixhawk 2	154	1.03	154
Cables y conectores	30	0.2	30
Total			539.2

Capítulo 4

Presupuesto personal

En esta sección se calculara el presupuesto en base a las horas empleadas por un ingeniero aeroespacial junior, teniendo en cuenta que el sueldo de esta posición ronda las 19€/h. se ha calculado la implicación en horas y su consiguiente precio:

Tarea	**Horas empleadas**	**Coste(€)**
Preparación del entorno	15	285
Programación apertura y lectura de puertos SITL	50	950
Declaración de mensaje uORB	25	475
Programación protocolo MAVLink	115	2185
Pruebas y depuración del código	21	399
Edición del Qgroundcontrol	75	1425
Redacción de documentos	24	456
Total	320	6080

Capítulo 5

Presupuesto total

En la siguiente tabla se resumen los costes totales brutos de este proyecto al que también se incluye en esta cantidad el costo después de aplicar el beneficio industrial (BI) del 6 % y añadir los gastos generales (GG) del 13 %.

Concepto	**Coste total(€)**
Software	156.1
Hardware	539.2
Personal	6080
BI+GG	1287.31
Total	8062.61

Y por último se le va a añadir un 21 % de IVA:

Concepto	**Coste total(€)**
Precio en bruto	8062.61
IVA(21 %)	1693.14
Total	9755.75

El presupuesto para este trabajo final de carrera es de **nueve mil setecientos cincuenta y cinco con setenta y cinco** euros.

UNIVERSIDAD POLITÉCNICA DE VALENCIA
(UPV)

ETSID

Diseño e implementación del módulo de comunicaciones entre un sistema de propulsión basado en pila de hidrógeno y el sistema autopiloto para vehículos aéreos no tripulados.

Pablo Ceacero Escrig

Tutor: Sergio García-Nieto Rodríguez
Grado: Ingeniería en Aeroespacial
Curso: 2022/2023

3 de julio de 2023

Trabajo de Fin de Grado [Pliego de condiciones]

Índice general

0.1. Introducción 2
0.2. Descripción del Proyecto 2
0.3. Condiciones Administrativas 2
0.4. Propiedad intelectual 3

0.1. Introducción

Este documento tiene como finalidad presentar el pliego de condiciones correspondiente al trabajo de fin de grado titulado «Diseño e implementación del módulo de comunicaciones entre un sistema de propulsión basado en pila de hidrógeno y el sistema autopiloto para vehículos aéreos no tripulados». El objetivo de este pliego de condiciones es establecer de manera exhaustiva y precisa los requisitos, especificaciones y criterios que deben cumplirse en el desarrollo y ejecución de dicho proyecto.

0.2. Descripción del Proyecto

Este trabajo se centra en el desarrollo de un módulo de comunicaciones para establecer una conexión efectiva entre una pila de hidrógeno y el Sistema de Control en Tierra (GCS), con el objetivo de supervisar el rendimiento y el estado de la batería en tiempo real. Se ha utilizado el software PX4 y el protocolo MAVLink para implementar este módulo, estableciendo una comunicación bidireccional sólida entre la pila de hidrógeno y la plataforma de control. Además, se ha personalizado el GCS con el software QGroundControl para mostrar de manera clara y concisa las variables relevantes en la pantalla principal. Esta personalización facilita la monitorización en tiempo real de los parámetros clave de la batería de hidrógeno, lo cual es esencial para tomar decisiones informadas y realizar un seguimiento preciso del rendimiento energético.

0.3. Condiciones Administrativas

La fecha límite para la entrega de todos los documentos escritos del trabajo es el 4 de junio, con el objetivo de realizar la presentación antes del 16 de dicho mes. Los documentos a entregar incluyen una memoria, un manual de programación para garantizar la replicabilidad del trabajo realizado en el TFG, un presupuesto y un pliego de condiciones. Todos estos documentos deberán ser entregados de forma virtual a través de la plataforma de trabajos de fin de grado de la Universidad Politécnica de Valencia.

Es fundamental cumplir con esta fecha establecida para garantizar el cumplimiento de los plazos académicos y permitir una adecuada revisión y evaluación del trabajo realizado. La memoria debe contener una descripción detallada del proyecto, incluyendo los objetivos, la metodología utilizada, los resultados obtenidos y las conclusiones alcanzadas. El manual de programación debe proporcionar instrucciones claras y precisas sobre cómo replicar y utilizar el sistema desarrollado

en el TFG, asegurando así la comprensión y la reproducibilidad del trabajo por parte de terceros.

Además de la memoria y el manual de programación, también se debe entregar un presupuesto que detalle los recursos financieros utilizados en el desarrollo del proyecto. Este presupuesto debe incluir los costos asociados a los materiales, equipos, software y cualquier otro gasto relevante en el marco del TFG.

Finalmente, se requiere la entrega de un pliego de condiciones, el cual establece los requisitos, especificaciones y criterios que deben cumplirse en el proyecto. Este documento es esencial para proporcionar una guía clara sobre las condiciones técnicas, administrativas y legales que deben considerarse en el desarrollo del trabajo.

La entrega de todos estos documentos de manera virtual a través de la plataforma de trabajos de fin de grado de la Universidad Politécnica de Valencia asegura una gestión eficiente y centralizada de los archivos, facilitando el acceso y la revisión por parte del equipo evaluador y demás personas involucradas en el proceso de evaluación del TFG.

0.4. Propiedad intelectual

El proyecto PX4 es un proyecto de código abierto que se basa en una licencia de software libre. El código fuente de PX4 se distribuye bajo la Licencia Pública General de GNU (GPL) versión 3. Esta licencia garantiza ciertos derechos y libertades a los usuarios y desarrolladores, y establece las condiciones para el uso, distribución y modificación del software.

Bajo la Licencia GPL versión 3, los usuarios tienen el derecho de usar, copiar, modificar y distribuir el código fuente de PX4, siempre y cuando se respeten los términos de la licencia. Al distribuir o modificar el software PX4, es necesario incluir los avisos de copyright y la licencia correspondiente.

Es importante tener en cuenta que aunque el código fuente de PX4 es de libre acceso y puede ser modificado y distribuido, cualquier modificación o extensión que se realice sobre el código fuente original de PX4 también debe ser distribuida bajo los términos de la Licencia GPL versión 3.

QGroundControl, al igual que PX4, es un proyecto de código abierto y su código fuente también se distribuye bajo la Licencia Pública General de GNU (GPL)

versión 3. Por lo tanto, la propiedad intelectual de QGroundControl sigue los principios de la licencia GPL v3.

Bajo la Licencia GPL v3, los usuarios tienen el derecho de usar, copiar, modificar y distribuir el código fuente de QGroundControl, siempre y cuando se cumplan los términos de la licencia. Esto significa que los usuarios pueden acceder, modificar y distribuir el software de QGroundControl de manera libre.

Al igual que con PX4, si se realizan modificaciones o extensiones en el código fuente original de QGroundControl, es necesario distribuir esas modificaciones bajo los términos de la Licencia GPL v3.

La plataforma GitHub es responsable de generar la documentación para Qgroundcontrol y PX4. se puede encontrar los términos de uso generales de los repositorios de GitHub y la documentación en su página web (https://help.github.com/en/articles/github-terms-of-service). Sin embargo, es importante tener en cuenta que cada documento puede tener sus propias condiciones de uso específicas.

Finalmente, los documentos creados durante la ejecución de este proyecto se rigen por las condiciones de uso y distribución establecidas en la plataforma Riunet de la Universitat Politècnica de València.