Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Diseño de un dron marítimo
TITULACIÓN: Grado en Ingeniería de Sistemas de Telecomunicación
AUTOR: David Campoy Sánchez
DIRECTOR: Dolors Royo Vallés
FECHA: 2 de febrero del 2018

Resumen

En este proyecto se ha diseñado un dron marítimo, partiendo de un barco de carreras de radiocontrol, cuyo objetivo es el control de profundidades mediante un sensor de profundidad con datos geolocalizados. Este proyecto es parte de un proyecto mayor llamado PilotFish, una aplicación que permite la comunicación en tiempo real o post procesada, del estado de las embarcaciones entre barcos o entre estaciones terrestres. Por ello, parte del software, cliente y servidor de esta aplicación fue diseñada por otro proyectista. Este proyecto constara de dos grandes bloques:

Primero veremos la arquitectura del barco, es decir, todos los componentes que lo forman y se estudiará que tipo de sensor de profundidad es el más adecuado dentro del abanico de posibilidades que tenemos en el mercado. Luego se diseñará la estructura física del barco, que tendrá como objetivo aportar mayor estabilidad al oleaje y aumentar el espacio físico para incluirle sensores al dron. Para cumplir estos requisitos se pensará en un diseño parecido al de un catamarán. También se discutirá sobre los materiales utilizados para la implementación de este.

Por otro lado, tendremos la parte software. El objetivo de esta parte es comunicar los diferentes sensores que se utilizarán con el servidor para poder procesarlos y enviarlos al cliente. También veremos los diferentes estándares y protocolos que se utilizarán para cumplir los requisitos del proyecto PilotFish. A parte, se introducirá un programa que utilizaremos para establecer las rutas autónomas que hará el barco, y un simulador que nos ayudará a la hora de programar la comunicación con uno de los sensores.

Por último, veremos los resultados finales de diferentes misiones que se han llevado a cabo, se razonará si se han cumplido los objetivos del trabajo y se añadirán propuestas para poder mejorar aún más el proyecto.

Title: Design of a martime drone

Author: David Campoy Sánchez

Director: Dolors Royo Vallés

Date: February 2 nd 2018

Overview

In this project, a maritime drone has been designed, starting from a radio control racing boat, whose objective is the depth control by means of a depth sensor with geolocalized data. This project is part of a larger project called PilotFish, an application that allows real-time or post-processed communication of the state of ships between ships or between land stations. Therefore, part of the software, client and server of this application was designed by another designer. This project will consist of two large blocks:

First, we will see the architecture of the ship, explaining all the components that form it and we will study which type of depth sensor is the most suitable within the range of possibilities that we have in the market. Then the physical structure of the ship will be designed, which will aim to provide greater stability to the waves and increase the physical space to include sensors to the drone. To fulfill these requirements, we will think of a design similar to that of a catamaran. The materials used for the implementation of this will also be discussed.

On the other hand, we will have the software part. The objective of this part is to communicate the different sensors that will be used with the server in order to process them and send them to the client. We will also see the different standards and protocols that will be used to meet the requirements of the PilotFish project. Besides, we will introduce a program that we will use to establish the autonomous routes that the boat will make, and a simulator that will help us when programming communication with one of the sensors.

Finally, we will see the final results of different missions that have been carried out, it will be reasoned if the objectives of the work have been met and proposals will be added to be able to further improve the project.

ÍNDICE

CAPITULO 1. INTRODUCCIÓN

Este proyecto es parte de un proyecto mayor llamado PilotFish. PilotFish es una aplicación que permite la entrada y la salida de los datos de navegación de un barco en tiempo real o guardados en el momento de la navegación y posteriormente presentados al servidor. Una de las características de PilotFish es que los datos son recogidos en el estándar NMEA0183 y traducidos a mensajes Signal k para poder procesarlos con la aplicación del cliente.

El objetivo del proyecto es diseñar un dron de agua que tenga un sensor de profundidad y otro de GPS y que se puedan acceder a los datos de los sensores en tiempo real desde un cliente que se ejecute en un ordenador. También se guardarán los datos en un fichero para que puedan ser postprocesadas.

Para esto, se montará un sistema cliente-servidor, dónde el servidor se instalará en una Raspberry Pi, un ordenador de placa reducida a bordo del dron, y el cliente se ejecutará en un ordenador en tierra. Por lo tanto, será el servidor el encargado de procesar las lecturas tanto del sensor de profundidad como de las coordenadas GPS, y enviarlas al cliente. A bordo, también encontraremos una placa que se encargará de controlar la hélice y el motor del barco. Esta placa será el Pixhawk, una controladora de vuelo, que tendrá un firmware capaz de controlar toda la electrónica a bordo de nuestro dron. Este firmware lo descargaremos e instalaremos desde un software gratuito llamado Mission Planner, el cual también se utilizará como estación de tierra, dónde podremos consultar el estado del dron en cualquier momento y programar misiones para que el dron se mueva de forma autónoma.

La Raspberry recibirá los datos de dos fuentes diferentes. Por una parte, tendremos la conexión directa por USB del transductor (sensor de profundidad), el cual envía datos periódicamente cada segundo. Por otra parte, puesto que el módulo GPS está conectado a nuestra controladora de vuelo Pixhawk, se utilizará un protocolo llamado MAVLink, para la comunicación de esta con la Raspberry.

Adicionalmente, debido a la falta de espacio en el dron, se construirá una estructura que acoplará unas cajas dónde irá montado el transductor de profundidad.

Dado que se tratan diferentes temas en este proyecto, se hará una separación lógica de capítulos para el mejor entendimiento de cada uno de estos.

En el segundo capítulo, se describe la arquitectura del dron, todos los componentes que lo forman y como se ha decidido su diseño. El tercero explicará toda la parte software, los programas utilizados, la comunicación entre los diferentes componentes y los protocolos y estándares que utilizaremos. Para demostrar el buen funcionamiento del sistema completo, el

cuarto capítulo incluye dos pruebas que se han hecho. Finalmente, en las conclusiones en el último capítulo, se harán propuestas sobre cómo mejorar más en un futuro todo el sistema del dron.

CAPITULO 2. ARQUITECTURA DEL BARCO

2.1. Estado inicial

En la **figura 2.1.1** se muestra el estado inicial del barco. Los componentes que lo formaban eran los siguientes (**Fig. 2.1.2**):

- Pixhawk (**Fig 2.1.4**), un sistema de piloto automático que controla la electrónica del dron.
- ESC o Electronic Speed Controller, es un elemento capaz de controlar la velocidad de giro del motor mediante pulsos.
- BEC o Battery Elimination Circuit, es un elemento que permite alimentar la electrónica del dron, regulando la tensión de las baterías a 5V.
- Power Module (**Fig. 2.1.3**), es un elemento que se encarga de alimentar el Pixhawk, a través de un cable de 6 pines.

Fig. 2.1.1 Barco, estado inicial

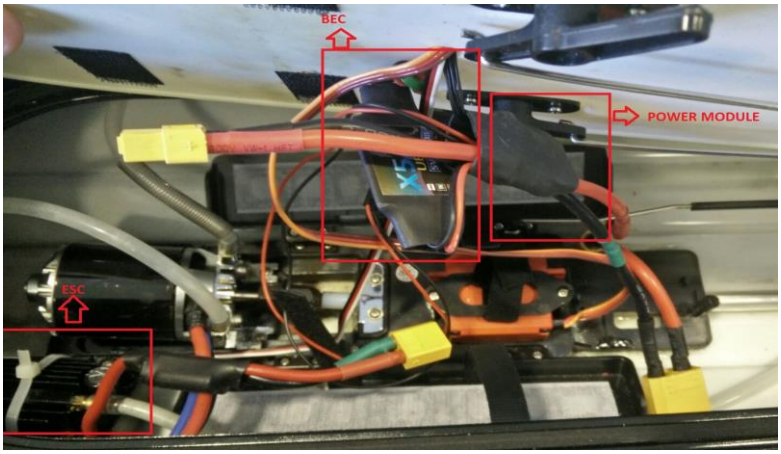


Fig. 2.1.2 Electrónica barco

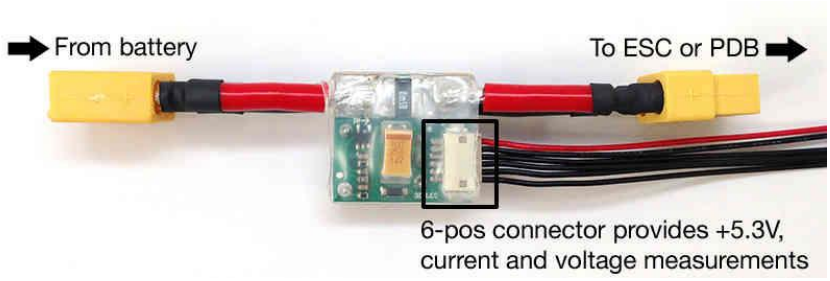


Fig. 2.1.3 Power module

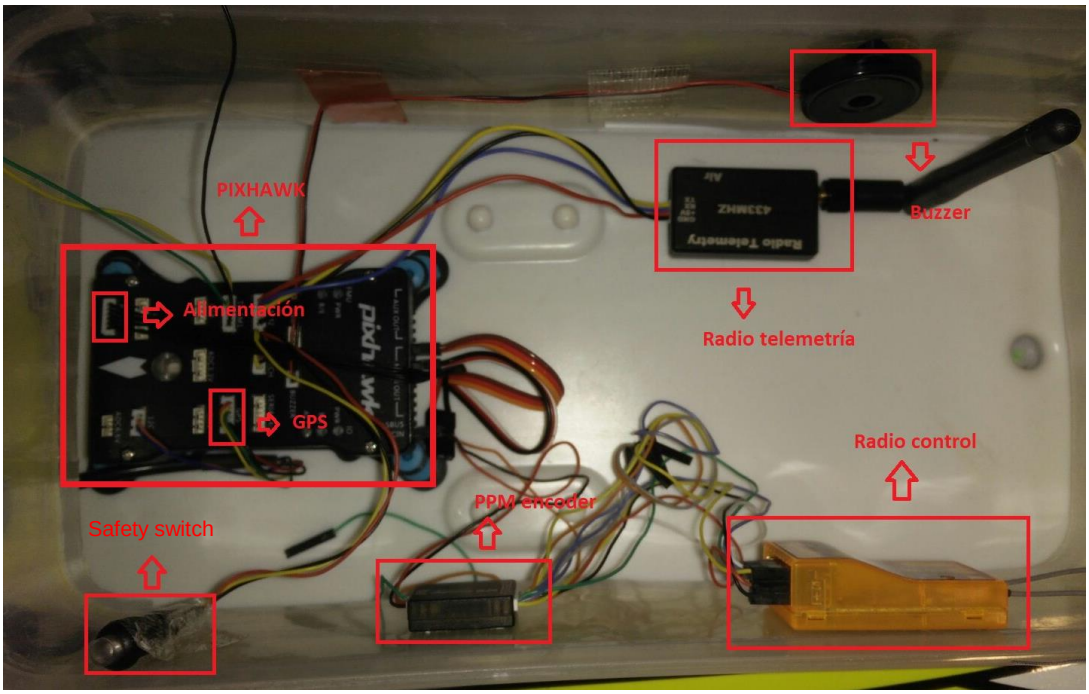


Fig. 2.1.4 Elementos Pixhawk

2.2. Pixhawk

Pixhawk (**Fig. 2.2.1**) es un sistema de piloto automático el cual nos permitirá controlar la electrónica de nuestro dron.
Este sistema está formado por diferentes componentes:

- Buzzer (**Fig. 2.2.2**): Proporciona sonido a nuestro sistema, mediante el cual podemos saber en qué estado se encuentra:
  - Encendido OK/Fail
  - No tarjeta SD
  - Armado/Desarmado
  - Fallo al armar
  - Batería baja
  - Fallo en GPS

- Secure switch (**Fig. 2.2.3**): Es un botoncito rojo para la seguridad el cual hay que activar para que podamos armar los motores del dron.

- GPS (**Fig. 2.2.4**): Sensor que nos ofrece las coordenadas de latitud y longitud.

- Radio telemetría (**Fig. 2.2.5**): Antena de radio para podernos conectar mediante el software Mission Planner.

- Radiocontrol (**Fig. 2.2.6**): Antena de radio receptora para poder tomar el control con un mando radiocontrol.

- PPM encoder (**Fig. 2.2.7**): Utilizado para multiplexar los diferentes canales radiocontrol en uno solo.

Fig. 2.2.1 Placa Pixhawk

Fig. 2.2.2 Buzzer

Fig. 2.2.3 Secure switch

Fig. 2.2.4 GPS

Fig. 2.2.5 Radio telemetría

Fig. 2.2.6 Radiocontro

Fig. 2.2.7 PPM encoder

Los siguientes subapartados explican los diferentes elementos que se han añadido en este proyecto al barco.

2.3. Raspberry

Raspberry (**Fig. 2.3.1**) es un ordenador de placa reducida de bajo costo que utilizaremos como servidor en nuestro sistema. Utilizaremos el modelo Raspberry Pi 3 B, que es el último modelo, el cual incluye conexión Wi-Fi sin necesidad de adaptadores.

La Raspberry necesita una memoria externa, por lo que usaremos una tarjeta SD de 32GB. En esta instalaremos Raspbian, una distribución del sistema operativo Linux, que es la recomendada por la misma fundación y permite configurar fácilmente el sistema operativo mediante una interfaz llamada *raspi-config.*

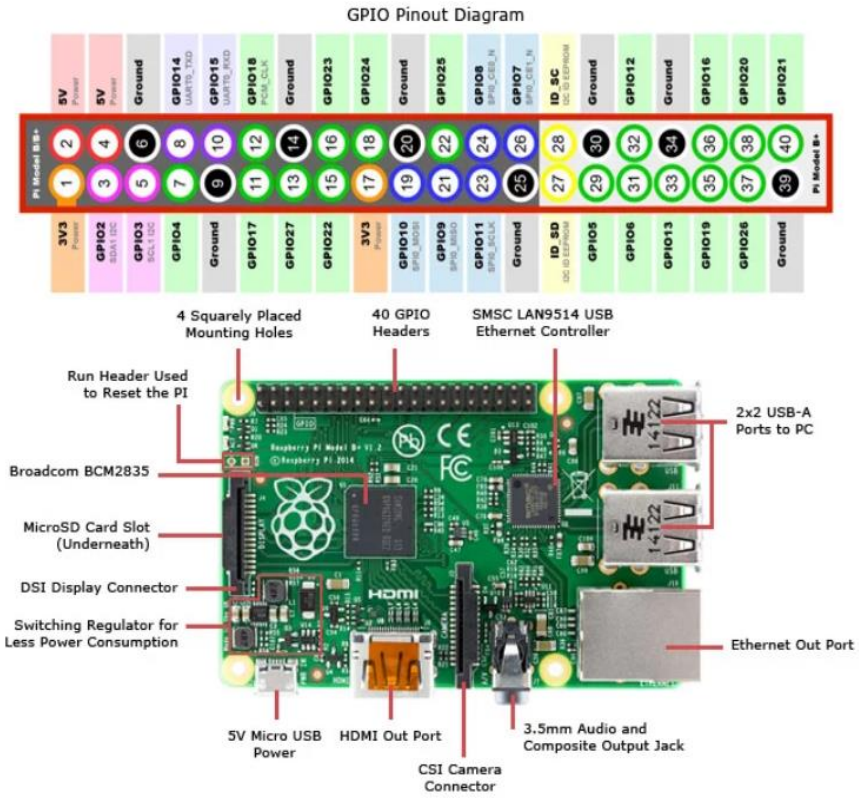


Fig. 2.3.1 Raspberry Pi 3 Modelo B

2.3.1. Conexiones

Conectado a las Raspberry podremos encontrar:

1. USB conectado al puerto de telemetría de Pixhawk para poder comunicarnos vía MAVLink y recibir datos GPS.
2. USB conectado al transductor, por el cuál recibiremos datos de profundidad y temperatura.
3. USB conectado a un adaptador de red, el cuál utilizaremos para obtener un rango mayor de conectividad Wi-Fi.
4. Alimentación de 5V y tierra, conectados a los pines 2 y 6 (**Fig 2.3.1**), a través del BEC.

2.3.2. Configuración

Primero de todo debemos habilitar la comunicación SSH, porqué será la forma que tendremos para comunicarnos remotamente con la Raspberry mediante el cliente PuTTY.

Para ello abriremos una pantalla de comandos y escribimos “sudo raspi-config” el cuál abre una interfaz de configuración del SO. Desde aquí podemos activar y/o desactivar diferentes opciones, entre ellas SSH.

Para poner a punto la Raspberry necesitamos instalar dos programas:

1. Servidor: programado en C#, el cuál compilaremos y quedará un ejecutable.

2. MAVProxy: *Micro Air Vehicle* Proxy es un GCS minimalista basado en líneas de comando y compatible con el protocolo MAVLink. Para la instalación de este necesitaremos conexión a internet puesto que hay que descargar los siguientes paquetes:

    - sudo apt-get update
    - sudo apt-get install screen python-wxgtk2.8 python-matplotlib python-opencv python-pip python-numpy python-dev libxml2-dev libxslt-dev
    - sudo pip install future
    - sudo pip install pymavlink
    - sudo pip install mavproxy

2.4. Transductor de profundidad

¿Qué es un transductor de profundidad? Es un dispositivo capaz de emitir señales sonoras que dirigen hacia el fondo en forma de cono, estas rebotan de forma inmediata con un objeto, en este caso el fondo marino, y vuelven al transductor, el cual interpreta el tiempo transcurrido entre la emisión y la recepción para poder calcular la profundidad.

Hay varios tipos de transductores:

1. **De popa**: El transductor de popa (**Fig. 2.4.1**) es quizás el más habitual en las embarcaciones de pequeña eslora. Su instalación se realiza en el espejo de popa, cerca de la quilla.
    - Ventaja: Menor coste.
    - Contra: Debido a su fácil instalación no se compromete la estanqueidad del barco dado que se atornilla al casco

2. **Pasacascos**: Este equipo (**Fig. 2.4.2**) tiene la peculiaridad de que se instala en un agujero practicado en el casco.
    - Contra: Mayor tamaño, mayor coste, hace falta modificar la estructura de la embarcación.

3. **De interior**: Una versión intermedia son los que se instalan en el interior del barco (**Fig. 2.4.3**), solo útiles para cascos de fibra de vidrio, puesto que la señal de emisión y recepción debe atravesar el material, y la emisión a través de otro tipo de material queda distorsionada o es, simplemente, imposible.
   - Ventaja: No hace falta modificar la estructura de la embarcación.
   - Contra: No es capaz de leer la temperatura puesto que no está sumergido, al contrario que los demás dispositivos.

Fig. 2.4.1 Transductor de popa

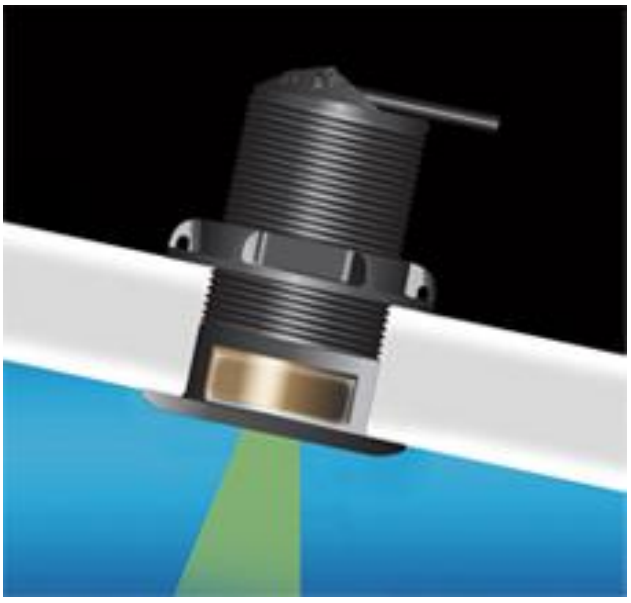


Fig. 2.4.2 Transductor pasacascos

Fig. 2.4.3 Transductor de interior

Cada transductor, además, necesita una fuente de alimentación para poder funcionar. Para saber que transductor se utilizará se deben de valorar

diferentes aspectos. El espacio necesario, tanto para el transductor como para su fuente de alimentación, y su instalación.

El transductor de popa, como primera opción al ser la más económica, se instala en la parte exterior del casco. Aquí aparecen varios inconvenientes.
Debido al diseño, no hay espacio suficiente en el espejo de popa. Otra opción sería colocarlo en un lateral del casco, pero la superficie ni es plana, ni es fácilmente accesible desde el interior del barco, lo que dificultaría mucho su instalación y el sellado de este. Por otro lado, colocarlo en un lateral afectaría al equilibrio del barco, pudiendo hacer que este no funcione como debería.

El pasa cascos y el interior, debido al poco espacio y a la estructura no plana del barco son imposibles de instalar.

Puesto que el principal problema es la falta de espacio, se plantea añadir una estructura adicional al barco, para poder colocar el transductor. Esta estructura necesita ser hermética, antes y después de la instalación del transductor, con lo que el transductor de popa queda descartado, ya que su instalación es la menos hermética de todas.
En este escenario, y puesto que el de interior reduce las posibilidades de estructura a solo fibra de vidrio, se decide utilizar el pasa cascos. Adicionalmente, respecto al de interior, también ganaremos información sobre la temperatura del agua, como comentado en la descripción de los diferentes tipos.

Adicionalmente, nuestro sensor deberá utilizar un protocolo específico para comunicarse con el resto de electrónica a bordo. Este protocolo será NMEA 0183.

Con todos estos datos, se procede a elegir el transductor que mejor se adapte al proyecto. Filtrando por las especificaciones deseadas encontramos 3 diferentes productos. (**Fig. 2.4.4**)

Fig. 2.4.4 Variedad de transductores en el mercado

Entre ellos podemos diferenciar algunos aspectos decisivos a la hora de escoger uno.

- **Funciones**: Como ya se ha explicado, la función principal de este transductor será poder medir la profundidad. Adicionalmente, este tipo de transductor nos ofrece la posibilidad de extraer también los datos de temperatura y, entre ellos, uno nos ofrece la función extra de tomar la

velocidad de navegación. Una funcionalidad extra no necesaria encarecerá el precio de nuestro proyecto.

- **Material**: El material del transductor será el factor más decisivo, puesto que el tipo de material influencia directamente en su peso y necesitamos que este sea ligero. Los fabricantes nos ofrecen el transductor en tres materiales diferentes: bronce, acero inoxidable y plástico. La versión B122 del transductor, solo fabricado en bronce, tendrá un peso de 3.9kg, mientras que DT800 y DST800 fabricados en los tres diferentes materiales, serán más ligeros con pesos de 0.9kg a 1.6kg.

Por lo tanto, se elegirá el modelo DT800 (**Fig. 2.4.1.1**) de plástico, puesto que es la opción más ligera de todas, y la más económica ya que no dispone de funcionalidades extras.

2.4.1 DT800

Fig. 2.4.1.1 DT800 y kit de montaje

Specifications

- Weight:
  - **0.9 kg (2 lb)—Plastic**
  - 1.5 kg (3.4 lb)—Bronze
  - 1.6 kg (3.6 lb)—Stainless Steel
- Acoustic Window: Urethane
- Data Update Rate: 1 per second
- Minimum Depth Range: 0.5 m (1.6’)
- Maximum Depth Range:
  - Up to 100 m (330’)—Non-Broadband
  - Up to 180 m (590’)—Broadband
- Pressure Rating: 3 m (10’)
- Supply Voltage:
  - 10 VDC to 25 VDC—NMEA 0183

- Supply Current:
  - <40 mA—NMEA 0183
- Standard Cable Length:
  - 10 m (33’)—NMEA 0183
- Temperature Accuracy: ±0.5°C (±1.8°F)
- Temperature Sensor Range: -10°C to 40°C (14°F to 104°F)

Fig. 2.4.1.2 Información cableado NMEA

Para la alimentación del transductor utilizaremos una batería (**Fig.2.4.1.3**) que cumpla los requisitos de las especificaciones:

- Supply Voltage:10 VDC to 25 VDC—NMEA 0183
- Supply Current: <40 mA—NMEA 0183

Fig. 2.4.1.3 Batería

Para el conexionado (**Fig. 2.4.1.2**) del transductor con los equipos necesitaremos añadir una conexión USB (**Fig. 2.4.1.4**) al cableado para la raspberry y un conector macho para la batería. (**Fig. 2.4.1.3**)

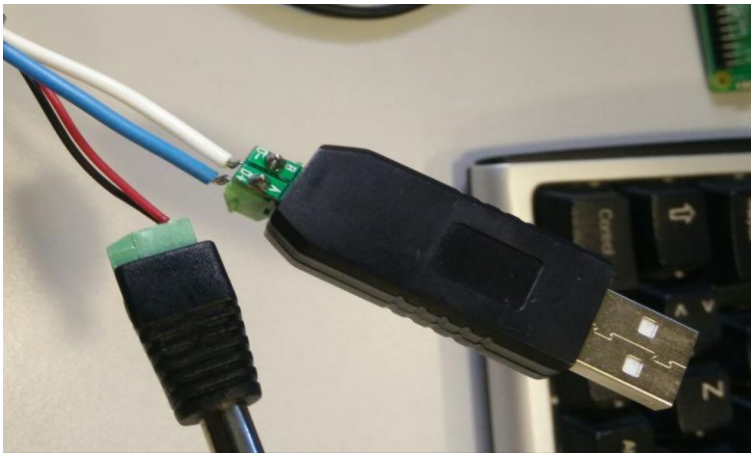


Fig. 2.4.1.4 Conexiones transductor

Para asegurar el buen funcionamiento del transductor, el conexionado y la batería, se realizó una primera prueba experimental (**Fig. 2.4.1.5**) con un software en Windows que leía e interpretaba las frases NMEA del puerto USB. Puesto que la profundidad mínima para leer es de 0.5 metros y no se disponía de un contenedor de ese tamaño, se utilizaron vasos de agua con diferentes temperaturas.

Fig. 2.4.1.5 Prueba experimental con vasos

Los resultados muestran el buen funcionamiento del sistema, pudiendo leer las sentencias y la variación de temperatura (**Fig. 2.4.1.7**), por otra parte, como el valor de profundidad está por debajo del mínimo este mostraba un valor de 0.6 metros (**Fig. 2.4.1.6**).

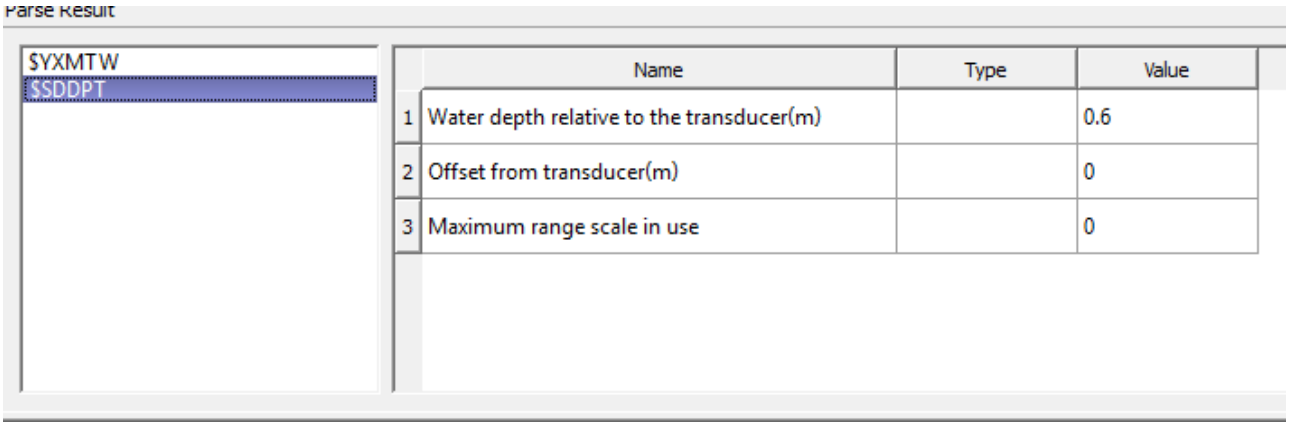


Fig. 2.4.1.6 Datos profundidad

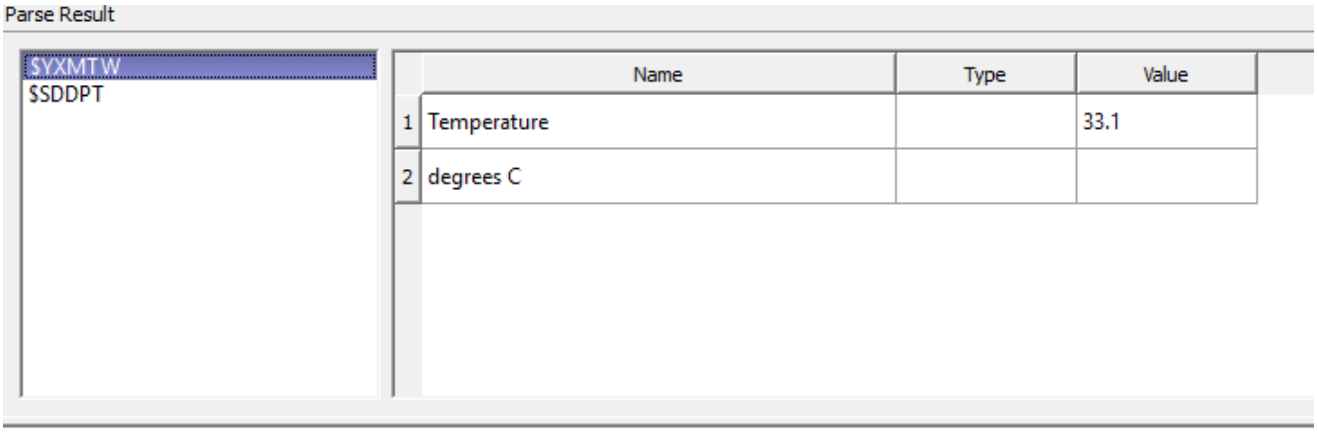


Fig. 2.4.1.7 Datos temperatura

2.5. Diseño y material

Debido a la falta de espacio en el interior del barco para instalar el transductor y la batería, se decide añadir una estructura adicional. El objetivo, integrar al dron con cajas herméticas, dos para mantener la simetría y no quitar flotación.
Antes de empezar con el montaje se hicieron dos esbozos (**Fig. 2.5.1**) y se compararon los pros y los contras.

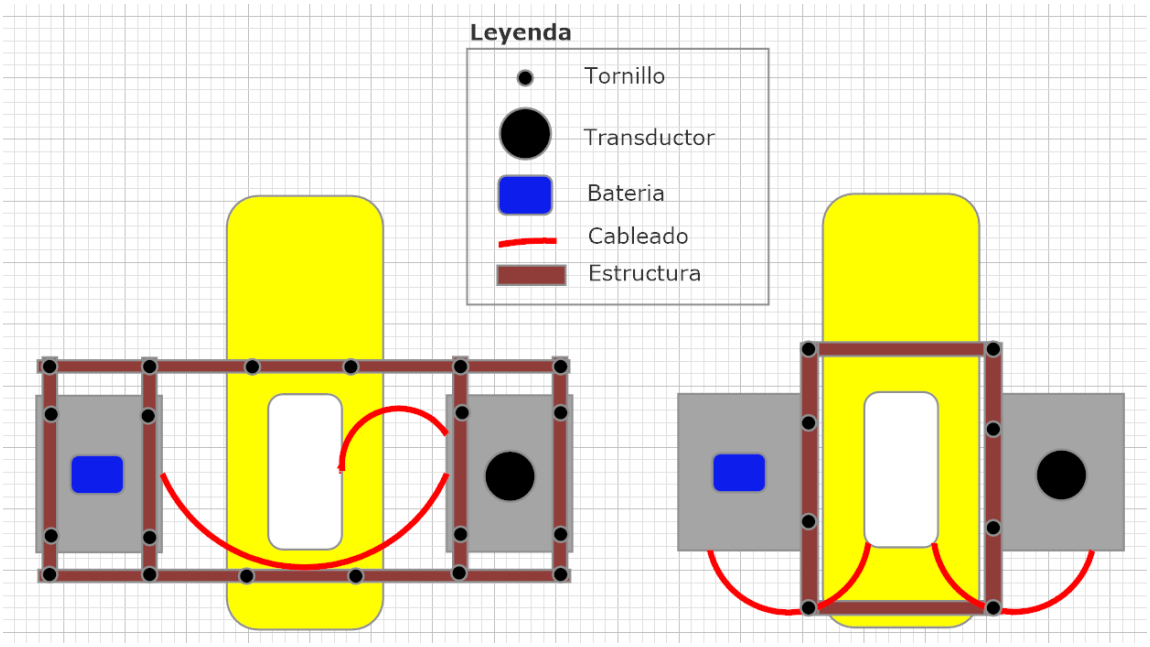


Fig. 2.5.1 Diseño 1 y Diseño 2

El primer diseño consta de unas pletinas fijas, atornilladas directamente al casco del barco, tanto en la parte delantera como trasera, unidas entre si con otras pletinas que soportan las cajas.
En el segundo diseño, la estructura forma una abrazadera, con el contorno del barco para mayor fijación, por la parte delantera y trasera. Esta luego irá unida a unas pletinas laterales que soportan las cajas.

Mientras que el primero ofrece mayor rigidez, porque las piezas no se necesitarían moldear, el segundo diseño ofrece mejores garantías:

- No es necesario agujerear el casco, lo cual sería un problema ya que no habría margen de error y debería asegurar el hermetismo, y debido al poco espacio en el interior para manipularlo sería complicado.

- Al ser una estructura extraíble podemos recuperar el barco inicial sin ningún problema.

Inicialmente se pensó en utilizar fibra de carbono como material de construcción de la estructura, puesto que este es muy resistente, ya que son piezas únicas creadas a partir de un molde. Además, es un material muy ligero, y no es corrosivo en contacto con el agua. Pero debido al coste y la complicación de la producción se optó por utilizar aluminio, un material más barato, pero más fácil de moldear con herramientas caseras.

2.5.1. Material

- 2 cajas estancas (**Fig. 2.5.1.1**) de plástico libre de halógeno, con grado de protección IP67, lo que implica que tiene protección contra el polvo **(6)** y que puede sumergirse 1 metro en el agua durante 30 minutos **(7)**. El tamaño de las cajas, 151x175x155 milímetros, viene dado por la altura del sensor, 12.5 centímetros.

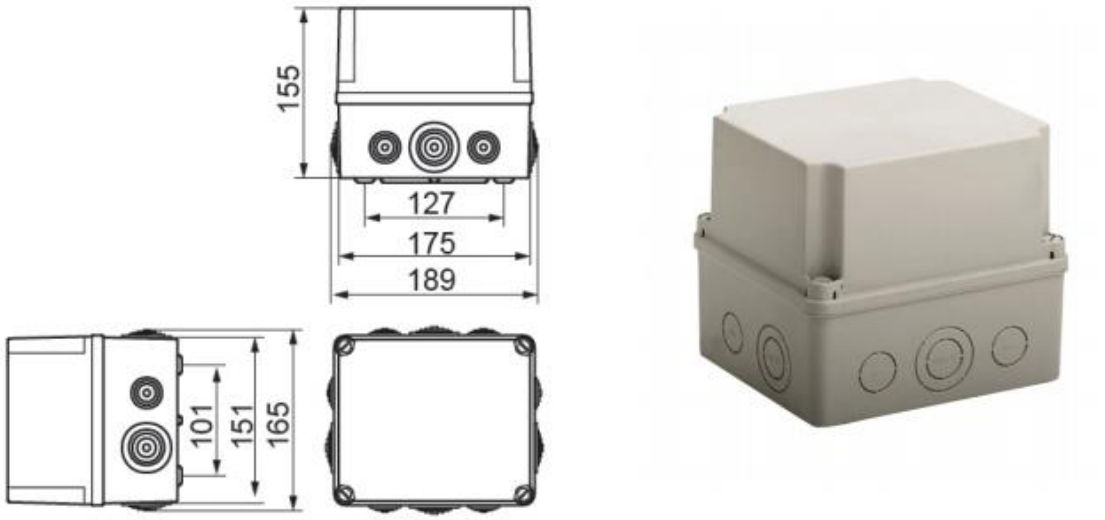


Fig. 2.5.1.1 Caja estanca

- Varillas de cobre (**Fig. 2.5.1.2**), utilizadas para hacer el molde de la grapa delantera y trasera de la cubierta del barco.

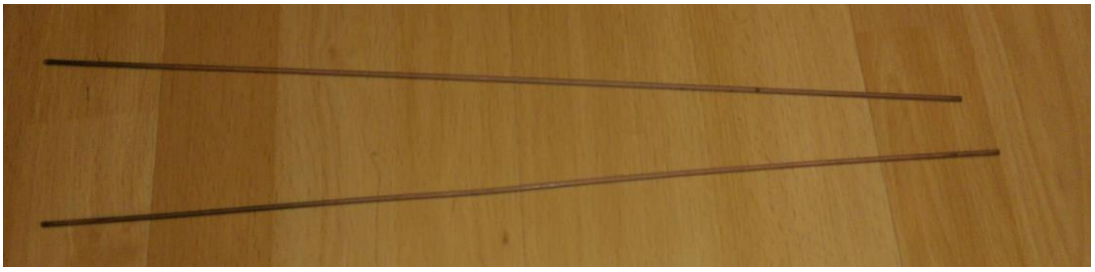


Fig. 2.5.1.2 Varillas de cobre

- Pletinas de aluminio (**Fig. 2.5.1.3**) de 1.5 y 3 milímetros de grosor, para la estructura.

Fig. 2.5.1.3 Pletina de aluminio

Fig. 2.5.1.4 Prensaestopas

Fig. 2.5.1.5 Escuadra inox

Fig. 2.5.1.6 Sellador de roscas

Fig. 2.5.1.7 Tornillos y tuercas

Fig. 2.5.1.8 Tuerca autoblocante

Fig. 2.5.1.9 Termorretráctil

2.6. Montaje

Antes de empezar con el montaje de la estructura, se instala el transductor en su caja. La decisión de dónde practicar el agujero para el transductor viene dada por la forma del interior de la caja, puesto que uno de los fondos es totalmente liso a diferencia del otro (**Fig. 2.6.1**), lo que facilita mucho el trabajo. El agujero se hará en la parte central de la caja.

Fig. 2.6.1 Interior caja

Fig. 2.6.2 Transductor instalado

Una vez hecho esto, se creará la estructura para incorporar las cajas al barco. La primera parte de la estructura que se necesita montar son las grapas (**Fig. 2.6.4**), que rodearán la parte delantera y trasera del barco y ofrecerán el agarre necesario para la estructura. Se harán dos grapas, una para adelante y otra para atrás, y cada grapa tendrá dos piezas, una para la parte de arriba del barco y otra para abajo.

Por comodidad se forman unos moldes con las varillas de cobre (**Fig. 2.6.3**), que son sencillas de dominar encima de la misma estructura del barco.

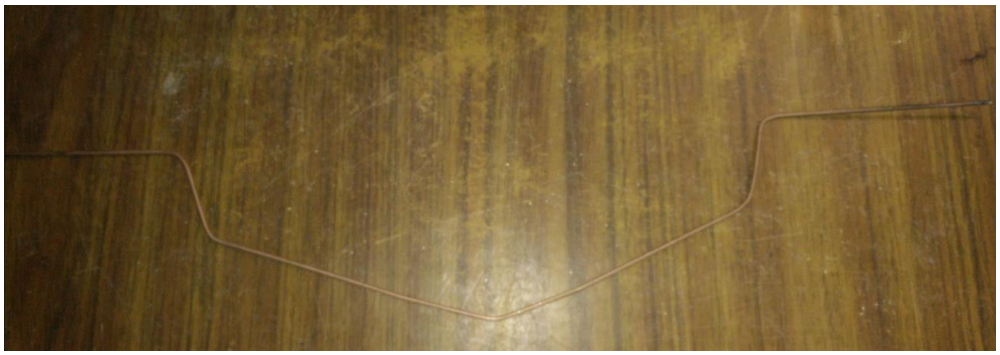


Fig. 2.6.3 Molde de cobre

A continuación, se procede a dar forma al aluminio de 1.5 milímetros de grosor. Se decide utilizar este grosor puesto que otro más ancho sería muy difícil de moldear con las herramientas disponibles.

Fig. 2.6.4 Grapa de aluminio

Para unir las grapas entre sí, se utilizarán escuadras de acero inoxidable (**Fig. 2.5.1.5**), que aportarán resistencia a las grapas.

Con tal de mejorar el agarre al barco y para protegerlo de la corrosión del agua, se cubren estas piezas con termorretráctil. (**Fig. 2.5.1.9**)

Fig. 2.6.5 Colocación de las grapas

Una vez colocadas las grapas, se unen con una pletina de 3 milímetros de grosor, mucho más resistente, dónde más tarde colocaremos las cajas.

Se agujerearán las cajas por la mitad para unirlas a la estructura. Para sellar estos agujeros se pondrá sellador de roscas. (**Fig. 2.5.1.6**)

Fig. 2.6.6 Estructura terminada

Todas las uniones de piezas se harán con tornillería de acero inoxidable (**Fig. 2.5.1.7**), y se añadirán tuercas autoblocantes (**Fig. 2.5.1.8**) en todos los tornillos para asegurar que no pueda llegar a desenroscarse por vibraciones.

Fig. 2.6.7 Estructura con las cajas

Para acabar, puesto que se necesita pasar cable de las cajas al barco se colocarán prensaestopas (**Fig. 2.1.5.4**), para sellar herméticamente los puntos de conexión.

2.7. Estado final

Finalmente colocamos la Raspberry a bordo del dron (**Fig. 2.7.1**). Por motivos de espacio se posiciona verticalmente en una de las paredes, por lo que se adquirió la caja de la Raspberry para poder adherirla con cinta de doble cara más fácilmente.

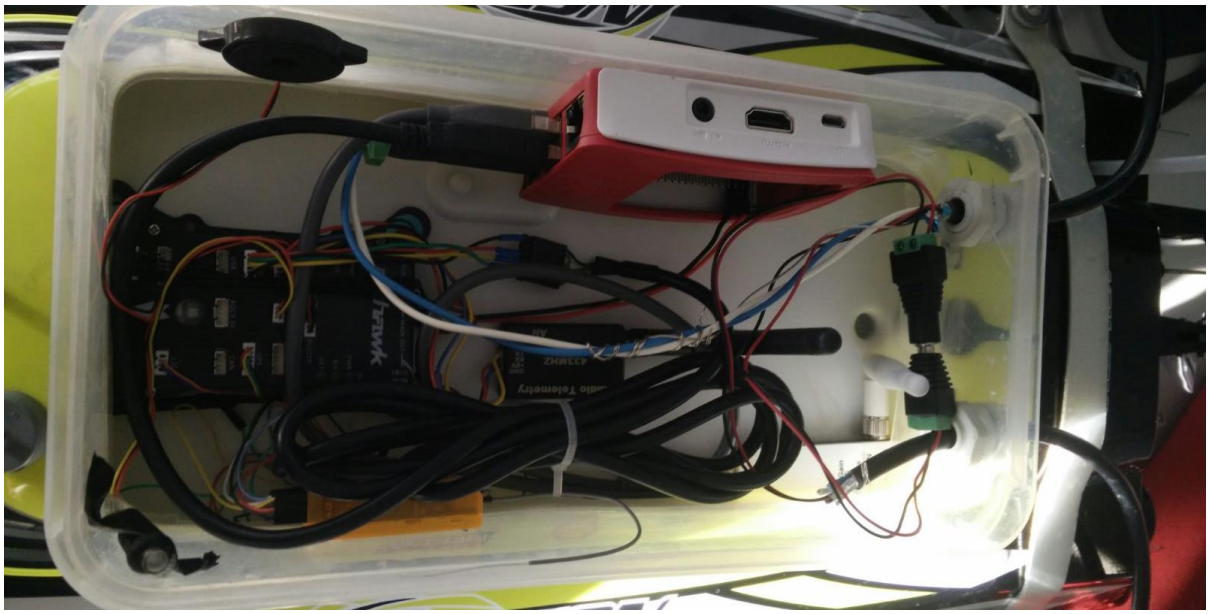


Fig. 2.7.1 Electrónica final

Para comunicar la Raspberry con Pixhawk utilizaremos un cable FTDI 3.3V de 6 pines- USB. Para saber cómo conectar los pines, se comprueban los pinouts de ambos dispositivos (**Fig. 2.7.2**) para la buena conexión de estos.

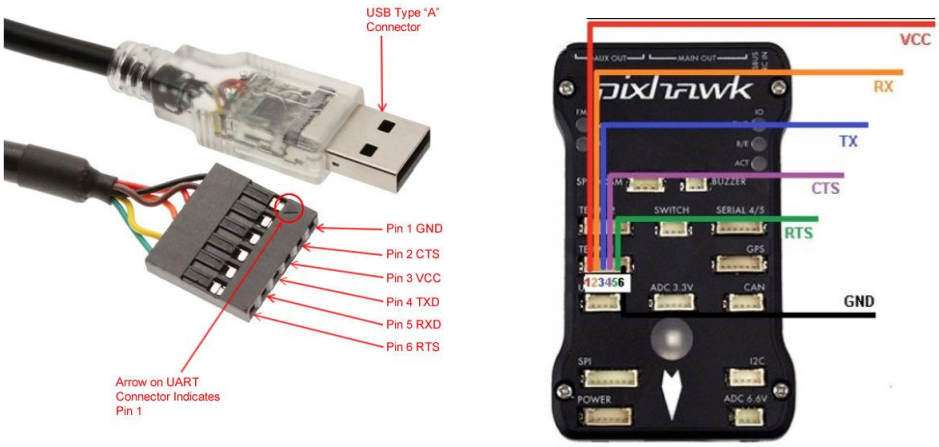


Fig. 2.7.2 Pinouts

Para ampliar la conectividad Wi-Fi, se incorpora una antena TPLink (**Fig. 2.7.3**). De esta forma, podremos seguir recibiendo mensajes a tiempo real en un rango más amplio desde nuestro cliente en tierra.

Por la falta de espacio para la distribución de la electrónica a bordo, se utiliza un alargo de USB para poder situar la antena dentro de la caja.

Fig. 2.7.3 Antena TPLink

CAPITULO 3. SOFTWARE

En este capítulo se describe el software que se ha desarrollado y/o adaptado para el funcionamiento del dron. El objetivo es programar un software que permita acceder a los datos del sensor en tiempo real pero que también se guarden en local para poder post procesarlos en el caso que no sea posible la conectividad en tiempo real.

3.1. Mission Planner

Mission Planner (**Fig. 3.1**) es una estación terrestre (GCS), una aplicación de software que se ejecuta en un ordenador en tierra y que se comunica con el dron mediante telemetría inalámbrica. Para ello necesitaremos una antena pareja, es decir con la misma configuración, como la que irá a bordo conectada al Pixhawk.

Este programa muestra datos en tiempo real sobre el rendimiento y la posición del dron y puede servir como una "cabina virtual", que muestra muchos de los mismos instrumentos que tendrías si estuvieras volando en un avión real. Un GCS también se puede usar para controlar un UAV en vuelo, cargando nuevos comandos de misión y parámetros de configuración.

En este caso, se utilizará Mission Planner para poder establecer una ruta mediante waypoints.

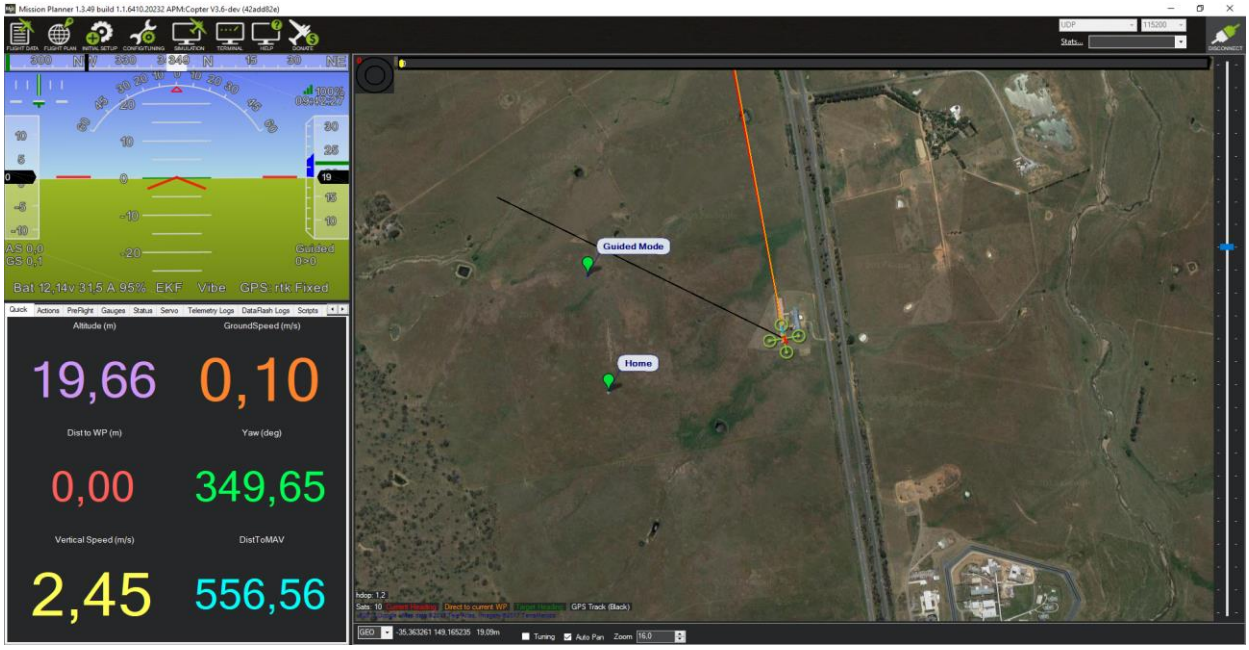


Fig. 3.1 Pantalla principal MP

3.1.1. Configuraciones iniciales

En MP hay ciertas configuraciones importantes que deben hacerse antes de empezar a hacer pruebas con el dron.

3.1.1.1. Calibración Radio

La calibración radio trata de ajustar los niveles de potencia máximos y mínimos de los diferentes canales de nuestro mando radiocontrol (**Fig. 3.1.1.1.1**). Cada una de las palancas y de los joysticks es un canal diferente.

- Los joysticks los utilizaremos para dirigir el movimiento y la velocidad del motor.
- Las palancas las utilizaremos para cambiar el modo de vuelo.

Fig. 3.1.1.1.1 Mando radiocontrol

Fig. 3.1.1.1.2 Interfaz MP para calibrar el radiocontrol

Para la configuración, solo tendremos que mover los diferentes joysticks y palancas de su posición máxima a su posición mínima. (**Fig. 3.1.1.1.2**)

3.1.1.2. Modos de vuelo

Los modos de vuelo (**Fig. 3.1.1.2.1**) son configuraciones que controlaremos a través el radio control con diferentes funcionalidades. Existen varios modos, pero debido a que nuestro dron no es volador tan solo necesitamos los siguientes:

- RTL: En el modo Return to Launch o Vuelta al lugar de lanzamiento, el vehículo navega desde su actual posición, para volver a *Home*.

- Auto: En el modo automático, el vehículo seguirá una misión programada, almacenada en el piloto automático.

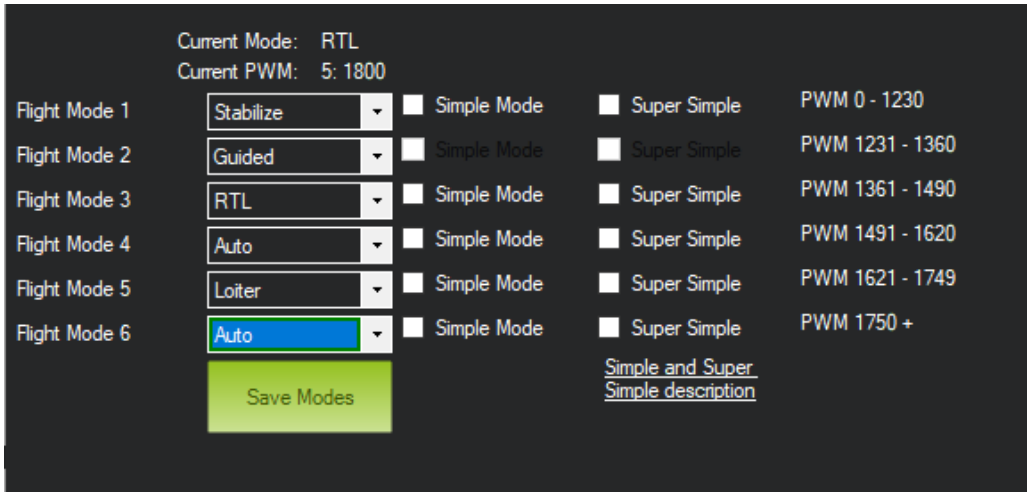


Fig. 3.1.1.2.1 Interfaz MP para configurar modos de vuelo

3.1.2. Misión

Para empezar, debemos definir una ruta en la pestaña Flight Plan, es tan sencillo como hacer click en un lugar del mapa, cada punto que marquemos será un waypoint. Como se puede ver en (**Fig. 3.1.2.1**) dónde se han marcado 3 waypoints, y 2 comandos. El primero de ellos takeoff hará que el dron pueda despegar, los metros indicados en su casilla, en este caso 100. Cuando este a 100 metros de altura, se dirigirá al primer waypoint, luego al segundo y finalmente al tercero, en este punto el siguiente comando es RTL, por lo que volverá al punto de Home y se desarmará.

Fig. 3.1.2.1 Configuración de misión

Antes de poder armar hay que pasar unas comprobaciones del piloto automático:

- Safety switch: Hay que dejarlo pulsado dos segundos hasta que deje de parpadear.
- GPS: Debe de tener señal.

Hay dos formas de armar:

1. Apretando el botón de armar en MP.
2. Aguantar la palanca del radio control hacia la derecha durante dos segundos y dejar en punto muerto el acelerador

Una vez se arma el dron, establece ese sitio como *Home* y ya estará listo para empezar la misión.

3.2. Simulador

Uno de los problemas a la hora de recibir las coordenadas GPS en el momento de la programación es que en interior de un edificio no recibe la señal satélite. Como solución, se instalará Software in the Loop (SITL).

SITL nos permitirá simular nuestro ArduPilot sin ningún hardware. También instalaremos MAVProxy como GCS y para poder recibir vía MAVLink (**Fig. 3.2.1**). MAVProxy se conectará al simulador por TCP.

```
STABILIZE> 1 links
0: tcp:127.0.0.1:5760
```

Fig. 3.2.1 Enlace SITL y MAVProxy

Para ello, utilizaremos Cygwin, una colección de herramientas que proporciona un comportamiento similar a Linux. Después de la instalación de todos los paquetes necesarios, ya podremos ejecutar el simulador.

Cuando lo iniciamos, nos aparecerán tres pantallas (**Fig. 3.2.2**):

1. MAVProxy: desde aquí podremos ver el estado del dron o enviarle ordenes, tales como armar motores, despegar…
2. Mapa: Mapa en tiempo real del simulador, desde el cuál se puede controlar el movimiento del vehículo.
3. Consola de ArduPilot: dónde veremos todo lo que sucede en el dron, su estado actual y mensajes.

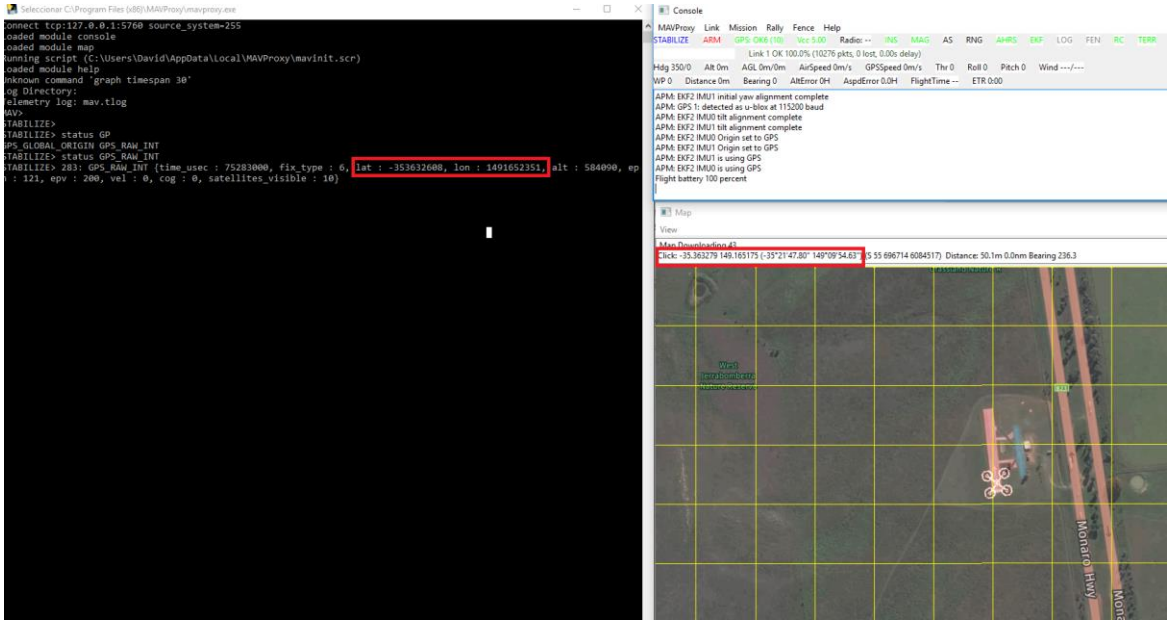


Fig. 3.2.2 Simulador**,** estado del GPS

Ahora ya está listo el sistema para poder establecer una comunicación vía MAVLink desde el servidor hasta MAVProxy. Primero nos aseguramos que MAVProxy tenga abierto el puerto a la dirección IP deseada, en este caso será todo en localhost.

```
STABILIZE> output list
STABILIZE> 2 outputs
0: 127.0.0.1:14550
1: 127.0.0.1:14551
```

Fig. 3.2.3 MAVProxy**,** lista de salidas

```
void MAVudp()
{
    IPAddress TargetIpAddress = new IPAddress(new byte[] { 127, 0, 0, 1 });
    mEndPoint = new IPEndPoint(TargetIpAddress, 14550);
```

Fig. 3.2.4 Servidor parámetros UDP para la comunicación

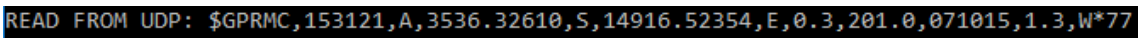


Fig. 3.2.5 Resultado, frase NMEA localización GPS

3.3. Comunicaciones

El objetivo de esta apartado es montar un sistema de comunicación cliente-servidor que pueda mostrarnos en tiempo real las coordenadas de longitud y latitud, la profundidad y la temperatura por pantalla en la estación terrestre.

Este capítulo se dividirá en tres bloques, primero se introducirán los diferentes protocolos que tendremos en el sistema. En otra parte, se explicará el funcionamiento del servidor, la lectura de los sensores, y la forma de enviar estos datos al cliente. Y finalmente, se hablará sobre el cliente y la manera de pedirle los datos al servidor.

3.3.1 Protocolos y estándares

3.3.1.1 NMEA 0183

La *National Marine Electronics Association (NMEA)* es una organización de comercio electrónico estadounidense que establece estándares de comunicación entre electrónica marina. Para este proyecto se utilizará el estándar NMEA 0183.

Los dispositivos NMEA 0183 están designados a transmitir o recibir con los siguientes parámetros:

- Baud rate: 4800
- Number of data bits: 8 (bit 7 is 0)
- Stop bits: 1 (or more)
- Parity: none
- Handshake: none

NMEA 0183 permite solo un transmisor y varios receptores en el mismo circuito. Desde el estándar se recomienda que la salida del transmisor sea con EIA RS422, un sistema diferencial con dos señales, A y B.

La forma de transmitir los datos de este protocolo es mediante frases. Todos los caracteres utilizados son ASCII imprimibles, más un retorno de carro (CR) y avance de línea (LF). Cada frase empieza con el símbolo “$” y termina con CR y LF. El formato de las frases es el siguiente:

$ttsss,d1,d2,....<CR><LF>

Las primeras dos letras que siguen a "$" son el identificador del transmisor. Los siguientes tres caracteres (sss) son el identificador de frase, seguido de un

número de campos de datos separados por comas. Para acabar encontramos el checksum, retorno de carro y avance de línea. El checksum consta de un "*" y dos dígitos hexadecimales que representan el OR exclusivo de todos los caracteres entre, sin incluir "$" y "*".
Un ejemplo sería:

$YXMTW,31.7,C*17<CR><LF>

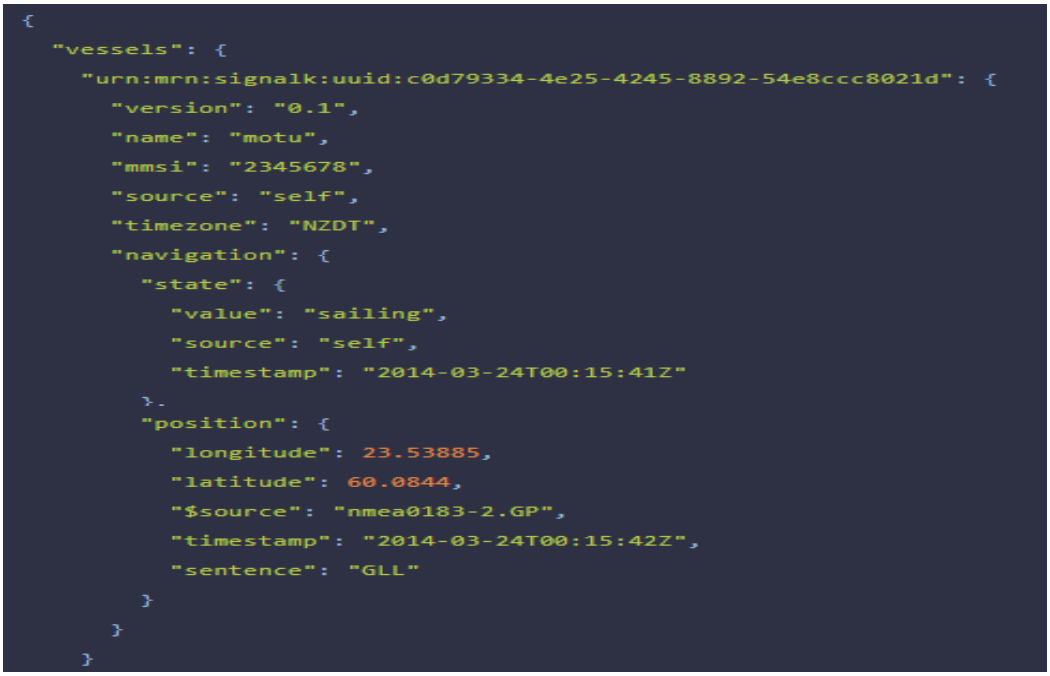
Analizando las partes de esta frase:

- YX: Identificador del dispositivo transmisor, transductor
- MTW: Identificador de frase, Mean Temperature of the Water
- 31.7: Temperatura
- C: Unidades, Celsius
- 17: Checksum

3.3.1.2 Signal K

SignalK es un protocolo pensado como solución a la comunicación de datos marinos tanto entre sensores e instrumentos, así como entre otras embarcaciones y estaciones de tierra. Está diseñado para utilizarse con aplicaciones web y móviles y también para ser conectado con el Internet de las cosas, IoT.

Los datos de Signal K se representan como un esquema JSON llamado "*full format*" (**Fig. 3.3.1.2.1**), lo que facilita mucho la lectura, al contrario que los mensajes NMEA 0183. El atributo clave siempre será "*vessels*", haciendo referencia a la embarcación en cuestión.



```
{
  "vessels": {
    "urn:mrn:signalk:uuid:c0d79334-4e25-4245-8892-54e8ccc8021d": {
      "version": "0.1",
      "name": "motu",
      "mmsi": "2345678",
      "source": "self",
      "timezone": "NZDT",
      "navigation": {
        "state": {
          "value": "sailing",
          "source": "self",
          "timestamp": "2014-03-24T00:15:41Z"
        },
        "position": {
          "longitude": 23.53885,
          "latitude": 60.0844,
          "$source": "nmea0183-2.GP",
          "timestamp": "2014-03-24T00:15:42Z",
          "sentence": "GLL"
        }
      }
    }
```


Fig. 3.3.1.2.1 Full format

Los valores están normalizados, por lo que no hace falta especificar las unidades. Por ejemplo, si hablamos de profundidad, siempre serán metros, y nunca centímetros, o decámetros.

Este formato es útil para realizar copias de seguridad, o para actualizar todos los datos con algún mensaje de estado actual, pero enviar todos los datos cada vez supondría un desperdicio de ancho de banda, y de CPU, porque la mayoría de datos no cambian con tanta frecuencia, por lo que se necesita poder enviar solo partes del modelo. Para ello se utiliza el "*Delta format*" (**Fig. 3.3.1.2.2**).

```
{
    "context": "vessels.urn:mrn:imo:mmsi:234567890",
    "updates": [{
        "source": {
            "type": "NMEA2000",
            "src": "017",
            "pgn": 127488,
            "label": "N2000-01.017"
        },
        "timestamp": "2010-01-07T07:18:44Z",
        "values": [{
            "path": "propulsion.0.revolutions",
            "value": 16.341667
        }, {
            "path": "propulsion.0.boostPressure",
            "value": 45500.0
        }]
    }]
}
```


Fig. 3.3.1.2.2 Delta format

Este formato se reconoce fácilmente por la propiedad de cabecera "*updates*", la otra cabecera "*context*", puede faltar, lo que haría referencia a sí mismo. Los elementos están formados por dos variables "*path*" y "*value*". "*Path*" debe ser una rama del árbol del "full *format*", dónde se encuentra el valor de Signal K asociado. Por ejemplo, "***navigation.position"*** sería un objeto como **{"latitude": -41.2936935424, "longitude": 173.2470855712}.** También podríamos consultar únicamente uno de estos valores así "**navigation.position.latitude"**.

3.3.1.3 MavLink

Micro Air Vehicle Link es un protocolo para comunicarse con vehículos pequeños no tripulados, diseñado como una librería de recopilación de mensajes de encabezado. Empaqueta estructuras y las envía a la estación de tierra.

La estructura de los paquetes es la siguiente (**Tab. 3.3.1.3.1**):

Start of frame	0	Indica el inicio de la trama, 0xFE.
Payload-length	1	Longitud de la carga útil (n).
Packet sequence	2	Cada componente cuenta su secuencia de envío. Permite la detección de pérdida de paquetes.
System ID	3	Identificación del sistema de envío. Permite diferenciar diferentes sistemas en la misma red.
Component ID	4	Identificación del componente enviado. Permite diferenciar diferentes componentes del mismo sistema.
Message ID	5	Identificación del mensaje: la identificación define qué significa la carga útil y cómo debe decodificarse correctamente.
Payload	6 to (n+6)	Los datos en el mensaje dependen de la identificación del mensaje.
CRC	(n+7) to (n+8)	Checksum del paquete completo, excluyendo el signo de inicio de paquete.

Tab. 3.3.1.3.1 Tabla de estructura de paquetes MAVLink

3.3.2 Servidor

El servidor se ejecutará en la Raspberry que estará a bordo del dron, programado en C#. Este recibirá los datos de dos fuentes, del transductor, en formato NMEA 0183, y del Pixhawk por MavLink. Estos datos los añadirá a un log.txt para poder postprocesarlos y en un buffer para convertirlos a datos Signal K y entenderlos mejor.
En los siguientes apartados veremos cómo se reciben los datos de todas las fuentes y como se procesan, para entenderlo mejor se se incluirán partes del código.

3.3.2.1 Transductor

El sensor de profundidad, conectado directamente al puerto USB de la Raspberry, envía periódicamente tres frases cada segundo con los datos de profundidad y temperatura que mide en ese momento.

Las frases que recibiremos tendrán esta forma:

$YXMTW,31.7,C*17
$SDDBT,1.1,f,0.3,M,0.1,F*04
$SDDPT,0.3,* 0.1*7A

MTW – Temperatura media del agua
1. Grados
2. Unidad de medida, Celsius
3. Checksum

DBT - Profundidad debajo del transductor
1. Profundidad, pies
2. f = pies
3. Profundidad, metros
4. M = metros
5. Profundidad, brazas
6. F = brazas
7. Checksum

DPT – Profundidad del agua
1. Profunidad, metros
2. Desplazamiento del transductor, positivo significa distancia del transductor a la línea de agua. Negativa significa distancia del transductor a la quilla
3. Checksum

En este caso, puesto que dos de las frases, DBT y DPT, ofrecen los mismos datos, solo procesaremos una, la DBT, la otra la ignoraremos.

Primero necesitamos que el programa lea estas frases para luego poder procesarlas. Se utilizará la función ScanComPortsLinux(), que buscará todos

los puertos USB utilizados (directorio /dev/tty en Raspbian/Linux) y comprobará si los datos leídos son frases NMEA y, si lo son, añadirá estas a un buffer.
Para la correcta lectura de los datos, es necesario hacerlo a un *baudrate* específico de 4800 como comentado anteriormente. (**Fig. 3.3.2.1.1**)

```
private void ScanComPortsLinux()//lee serial de linux
{
    while (scanActive)
    {
        List<string> ports = new List<string>();

        string[] ttys = Directory.GetFiles("/dev/", "tty*");

        foreach (string dev in ttys)
        {
            if (dev.StartsWith("/dev/ttyS") || dev.StartsWith("/dev/ttyUSB") || dev.StartsWith("/dev/ttyACM"))
            {
                ports.Add(dev);
            }

        }

        foreach (string p in ports)
        {
            Console.WriteLine("COM port found: " + p);
        }
        if (ports.Count > 0)
        {
            int i = 0;
            foreach (string port in ports)
            {
                try
                {
                    if (stream.Find(x => x.PortName == port) == null)
                    {
                        //SerialPort sp = new SerialPort(port, 9600);
                        SerialPort sp = new SerialPort(port, 4800);
                        sp.ReadTimeout = 10000;
                        sp.DataReceived += new SerialDataReceivedEventHandler(DataReceivedHandler);
                        sp.Open();
                        stream.Add(sp);
                        int aux = i;
                        Thread t = new Thread(() => CheckComPort(sp));
                        t.Start();
                        i++;
```

Fig. 3.3.2.1.1 Código para leer puertos USB

Para poder comprobar si es NMEA se mira el primer carácter de la frase. Si este es “$” tendremos una frase NMEA (**Fig. 3.3.2.1.2**), por lo tanto, la añadiremos al buffer dónde se hará el procesado de datos.

```
private void CheckComPort(SerialPort sp)
{
    int cont = 0;

    try
    {
        while (control && cont < 3)
        {
            string sentence;
            lock (locker)
            {
                sentence = sp.ReadLine();
            }

            if (sentence[0] == '$')
            {
                NmeaBuffer.GetInstance().AddNmeaSentence(sentence);
                Log.WriteLine("Read from COM: " + sentence);
            }

            else
            {
                cont++;
                if (cont == 3)
                {
                    sp.Close();
                    sp.Dispose();
                    stream.Remove(sp);
                }
            }
        }
```

Fig. 3.3.2.1.2 Código para comprobar y añadir frases NMEA

Finalmente, para convertir los datos de NMEA a Signal K, cada ID de frase tendrá un *Merge (***Fig. 3.3.2.1.3***)*. Simplemente separamos la frase por el delimitador "," y guardamos en variables.

```
public static Signalk Merge(string sentence, Signalk sk)
{
    Signalk res = null;

    MtwMessage message = new MtwMessage();

    sentence = sentence.Substring(0, sentence.IndexOf("*"));

    string[] info = sentence.Split(',');

    message.Degrees = info[1];
    message.Celcius = info[2];              //Unit of Measurement, Celcius

    if (message.Degrees != "")
    {
        res = MergeMtwMessage(sk, message.Degrees);
    }
    else
    {
        Log.WriteLine("Status: Invalid data.");
    }
    return res;
}
```

Fig. 3.3.2.1.3 Código para pasar de NMEA a Signal K de MTW

3.3.2.2 GPS

El GPS está conectado al Pixhawk, y este a su vez a la Raspberry por USB. Para poder comunicarnos a través de nuestro servidor y así recibir datos de GPS, tenemos que tener instalado MAVProxy en la Raspberry.

Gracias a la conexión USB de Pixhawk y Raspberry podemos conectar fácilmente MAVProxy con Pixhawk. Al igual que con Mission Planner, podemos consultar el estado actual de nuestro vehículo, así como la localización GPS. (**Fig. 3.3.2.2.1**)

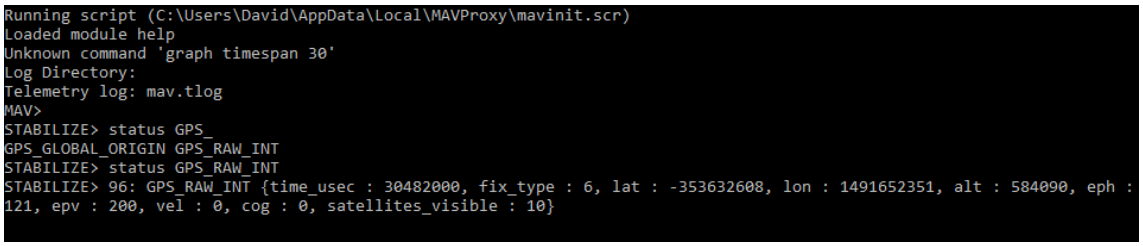


Fig. 3.3.2.2.1 MAVProxy consulta GPS

Ahora necesitamos poder enviar los datos GPS del MAVProxy a nuestro servidor, para ello se utilizará una librería de MAVLink en C#.

Primero abrimos una conexión UDP (**Fig. 3.3.2.2.2**) contra el MAVProxy, para luego poder empezar a recibir datos.

```
void MAVudp()
{
    IPAddress TargetIpAddress = new IPAddress(new byte[] { 127, 0, 0, 1 });
    mEndPoint = new IPEndPoint(TargetIpAddress, 14550);
    mUdpClient = new UdpClient(mEndPoint);
    while (mRunning)
    {
        IAsyncResult result = mUdpClient.BeginReceive(AsyncRecv, this);
        //Always time out. Never block.
        result.AsyncWaitHandle.WaitOne(1000);
        Thread.Sleep(33);

    }
    //sw.Close();
}
```

Fig. 3.3.2.2.2 Código de conexión UDP a MAVProxy

El protocolo MAVLink consta con una tabla de mensajes comunes (**Fig. 3.3.2.2.3**), dónde indica todos los parámetros que la componen, el tipo de valor, y descripción de la forma en que los encontraremos cuando analicemos el paquete. Consultando esta podemos buscar cuál de los mensajes nos aportará los parámetros que mejor se ajusten. En este caso, como necesitamos latitud y

longitud se decide utilizar el mensaje GLOBAL_POSITION_INT, cuyo MSG_ID es el 33.
Una vez empecemos a recibir paquetes vía MAVLink los separaremos por partes (**Fig. 3.3.2.2.4**) para separar los datos de la cabecera. Para empezar, miramos los primeros bytes, para saber dónde empieza la trama. y filtramos por MSG_ID (**Tab. 3.3.1.3.1**).

GLOBAL_POSITION_INT (#33)

The filtered global position (e.g. fused GPS and accie position is in GPS-franded, Z-up). It is designed as scaled integer message since the resolution of float is not suffic

Field Name	Type	Description
time_boot_ms	uint32_t	Timestamp (milliseconds since system boot)
lat	int32_t	Latitude, expressed as degrees * 1E7
lon	int32_t	Longitude, expressed as degrees * 1E7
alt	int32_t	Altitude in meters, expressed as * 1000 (millimeters), AMSL (not WGS84 - note that virtually
relative_alt	int32_t	Altitude above ground in meters, expressed as * 1000 (millimeters)
vx	int16_t	Ground X Speed (Latitude, positive north), expressed as m/s * 100
vy	int16_t	Ground Y Speed (Longitude, positive east), expressed as m/s * 100
vz	int16_t	Ground Z Speed (Altitude, positive down), expressed as m/s * 100
hdg	uint16_t	Vehicle heading (yaw angle) in degrees * 100, 0.0..359.99 degrees. If unknown, set to: UINT

Fig. 3.3.2.2.3 Tabla mensajes comunes MAVLink

```
void parseMessage(ref byte[] msg)
{
    //We need to be able to read at least the length of the payload
    for (int offset = 0; offset < msg.Length - 1;)
    {
        //look header
        if (msg[offset] == 0xfe)// 0
        {

            int msg_offset = MAVLINK_HEADER_SIZE;
            int len = msg[offset + 1]; //length of payload, not including header and checksum offset=1 --n
            //sanity check -- ditch this packet if it fails
            if (offset + len + MAVLINK_HEADER_SIZE + MAVLINK_CHECKSUM_SIZE > msg.Length)
                break;

            int sequence = msg[offset + 2];//offset=2 --n+1
            int msg_id = msg[offset + 5];//offset=5 --n+4
                                         //Look at ID

            switch (msg_id)
            {
```

Fig. 3.3.2.2.4 Código para analizar paquete MAVLink

De todos los campos que lo componen este mensaje sólo necesitaremos la latitud y la longitud.
Por lo tanto, ahora que sabemos que datos contiene, el orden de estos, su tipo, y sus unidades podemos guardarlos en variables.

Para hacerlo todo más estándar, se decidió generar a partir de las coordenadas procesadas una frase NMEA del tipo RMC y tratarla como tal.

$GPRMC,153121,A,4115.4540,N,00155.4470,E,0.3,201.0,071015,1.3,W*63

RMC – Información de navegación mínima recomendada

1. UTC Time
2. Estado, V= Advertencia | A=Valido
3. Latitud
4. N o S
5. Longitud
6. E o W
7. Velocidad sobre suelo, nudos
8. Seguimiento, grados
9. Fecha
10. Variación magnética, grados
11. E o W
12. Checksum

Puesto que los únicos datos que necesitamos que varíen de esta frase son latitud, longitud, sus respectivas posiciones, y el checksum, se generará a partir de una frase genérica. (**Fig. 3.3.2.2.5**) Una vez tenemos esta frase, la añadiremos al buffer dónde se procesarán a Signal K.

```
//gps id 33
case MAVLINK_MSG_ID_GLOBAL_POSITION_INT:
    {
        string lonOri, latOri;
        latOri = "N";
        lonOri = "E";
        UasGlobalPositionInt gps = new UasGlobalPositionInt();
        gps.TimeBootMs = BitConverter.ToUInt32(msg, offset + msg_offset); msg_offset += sizeof(int);
        gps.Lat = BitConverter.ToInt32(msg, offset + msg_offset); msg_offset += sizeof(int);
        gps.Lon = BitConverter.ToInt32(msg, offset + msg_offset); msg_offset += sizeof(int);
        if (gps.Lat < 0)//latitud positiva norte, negativa sud
        {
            gps.Lat = -gps.Lat;
            latOri = "S";
        }

        if (gps.Lon < 0)  //longitud positiva este, negativa oeste
        {
            gps.Lon = -gps.Lon;
            lonOri = "W";
        }
        string resLat = separarnumero(gps.Lat);
        string resLon = separarnumero(gps.Lon);
        sentenceGPS = "GPRMC,153121,A," + resLat + "," + latOri + "," + resLon + "," + lonOri + ",0.3,201.0,071015,1.3,W";
        string chk=Checksum(sentenceGPS);
        sentenceGPS = "$GPRMC,153121,A," + resLat +"," + latOri+ ","+resLon+ ","+lonOri+",0.3,201.0,071015,1.3,W*"+chk;
        NmeaBuffer.GetInstance().AddNmeaSentence(sentenceGPS);
        Log.WriteLine("READ FROM UDP: " +sentenceGPS);
```

Fig. 3.3.2.2.5 Código para procesar el paquete

Finalmente, convertimos los datos de NMEA a Signal K. (**Fig. 3.3.2.2.6**)

```
public static Signalk Merge(string sentence, Signalk sk)
{
    Signalk res = null;

    RmcMessage message = new RmcMessage();

    sentence = sentence.Substring(0, sentence.IndexOf("*"));

    string[] info = sentence.Split(',');

    message.PositionStatus = info[2];
    message.Latitude = info[3];
    message.LatOrientation = info[4];
    message.Longitude = info[5];
    message.LonOrientation = info[6];
    message.Date = info[9];
    message.UtcTime = info[1];

    if (message.PositionStatus == "A")
    {
            string latitude = Utilities.ParseNmeaCoord(message.Latitude, message.LatOrientation);
            string longitude = Utilities.ParseNmeaCoord(message.Longitude, message.LonOrientation);
            string timestamp = Utilities.ParseUtcTime(message.UtcTime, message.Date);
            res = MergeRmcMessage(sk, latitude, longitude, timestamp);
    }
    else
    {
        Log.WriteLine("Status: Invalid data (Void sentence)");
    }
    return res;
}
```

Fig. 3.3.2.2.6 Código para pasar de NMEA a Signal K de RMC

3.3.2.3 Buffer

Cuando añade una frase al buffer comprueba el identificador de frase, si este es MTW (**Fig. 3.3.2.3.1**), crea una lista con todas las frases del buffer, las valida y las transforma en datos Signal K (**Fig.3.3.2.3.2**).

```
private string[] locationMsgs = new string[] { "MTW" };
public void AddNmeaSentence(string text)
{
    lock (nmeaLocker)
    {
        nmeaBuffer.Add(text);
    }
    if (text[0] == '$')
    {
        bool found = false;
        int i = 0;
        while (i < locationMsgs.Length && !found)
        {
            if (text.Substring(3, 3) == locationMsgs[i])
            {
                found = true;
            }
            i++;
        }
```

Fig. 3.3.2.3.1 Código para añadir frases al buffer

```
private void ConvertNmeaData(object sender, NmeaLocationReachedEventArgs e)
{
    List<string> sentences = e.NmeaBuffer;
    if (sentences.Count > 0)
    {
        Signalk currentK = new Signalk();
        currentK.Self = Singleton.GetInstance().Self;
        for (int i = 0; i < sentences.Count; i++)
        {
            if (sentences[i][0] == '$')
            {
                if (ValidateCheckSum(sentences[i]))
                {
                    try
                    {
                        string aux = sentences[i].Substring(3, 3);
                        string type = aux.Substring(0, 1).ToUpper() + aux.Substring(1, 2).ToLower();
                        Type t = Type.GetType("SignalkServer.Parser." + type + "Parser");
                        if (t != null)
                        {
                            Signalk old = currentK;
                            currentK = (Signalk)t.GetMethod("Merge").Invoke(null, new object[] { sentences[i], currentK });
                            if (currentK == null)
                            {
                                currentK = old;
                            }
                        }
                        else
                        {
                            Log.WriteLine("Parser does not exist " + type);
```

Fig. 3.3.2.3.2 Código para convertir mensaje a Signal K

3.3.3 Cliente

El cliente se ejecutará en un ordenador en tierra, que estará en la misma red que la Raspberry y la comunicación será TCP (**Fig. 3.3.3.1**).

Las consultas para pedir los datos al servidor se realizan mediante dos parámetros:

1. Vessel: Hace referencia a la embarcación
2. Path: Hace referencia a una de las ramas del árbol de Signal K.

Por lo tanto, si nos subscribimos a dos paths, tendremos valores, de sus respectivas ramas.
En nuestro caso, la referencia vessel de la embarcación será “123456789”, y los valores que queremos consultar serán:

1. Environment: Nos devolverá valores de la temperatura y la profundidad.
2. Navigation.position: Nos devolverá las coordenadas GPS.

```
public void CallTcpSocket()
{
    TcpClient client = new TcpClient();
    //client.Connect("192.168.1.90", 55555);
    client.Connect("127.0.0.1", 55555);

    string message = @"{""context"":""vessels.123456789"",""subscribe"":[{""path"":""environment""},{""path"":""navigation.position""}]}";
    //[{""path"":""environment.depth.belowTransducer""},{""path"":""navigation.position""}]}"
    //string message = @"{""context"":""vessels.123456789"",""subscribe"":[{""path"":""environment.depth.belowTransducer""},{""path"":""nav
    //string message = @"{""context"":""vessels.123456789"",""subscribe"":[{""path"":""environment.depth.belowTransducer""},{""path"":""env
    //string message = "{\"context\":\"vessels.123456789\",\"subscribe\":[{\"path\":\"environment\"}]}";
    //string message4 = "{\"context\":\"vessels.366982320\",\"subscribe\":[{\"path\":\"navigation.position\"}]}";

    byte[] data = Encoding.UTF8.GetBytes(message);

    // Get a client stream for reading and writing.
    Stream stream = client.GetStream();

         NetworkStream ns = client.GetStream();

    // Send the message to the connected TcpServer.
    StreamWriter sw = new StreamWriter(ns);
    sw.WriteLine(message);
    sw.Flush();
    Console.WriteLine("Sent: {0}", message);
```

Fig. 3.3.3.1 Código para subscribirse al server

CAPITULO 4. PRUEBAS

El objetivo de las pruebas es ver el correcto funcionamiento del dron, tanto la parte de diseño de la estructura, para saber si el dron puede moverse con facilidad, si la estructura se soporta bien, comprobar el nivel de flotación del conjunto, como para la parte de diseño de software, que haya comunicación cliente-servidor y que los datos sean coherentes.

Lo interesante de estas pruebas es hacer que el barco se mueva a una velocidad lenta, puesto que recibiremos los datos de profundidad cada segundo, y así podemos tener muestras cada uno o dos metros.

4.1. Canal Olímpico

La primera prueba se hará en el Canal Olímpico de Cataluña.

Fig. 4.1.1 Nivel de flotación

Lo primero que se comprueba es que el barco puede navegar correctamente, que las cajas queden a un buen nivel de flote correcto para que pueda tomar datos el sensor de profundidad. Una vez visto el correcto funcionamiento de este, se programará una ruta en MP (**Fig. 4.1.2**) para ver que todo el sistema cliente-servidor funciona bien.

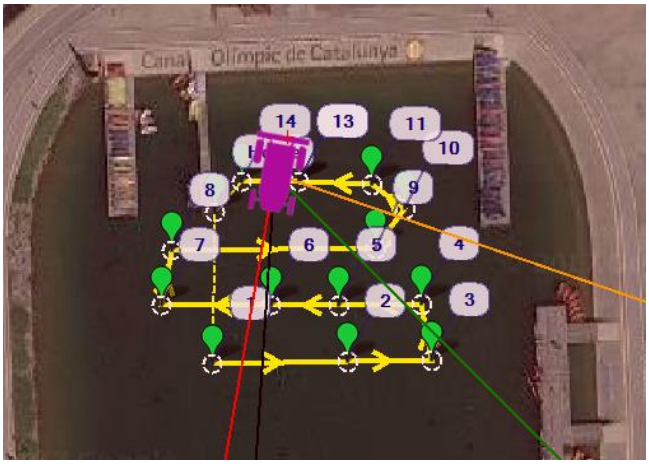


Fig. 4.1.2 Ruta, waypoints MP

Uno de los problemas que se presentaron antes de la prueba era el corto alcance entre el dron y el cliente en tierra, puesto que la red Wi-Fi en la que estaban conectados era la generada desde mi teléfono.

Por lo tanto, se configuró un router (**Fig. 4.1.3**) en el laboratorio con una red llamada “Tfgdroncanal”. Para que la raspberry pudiese conectarse a la red siempre con la misma IP y no tener que modificar ningún código, se puso una IP estática a la interfaz de la antena Wlan1 de la raspberry (**Fig.4.1.4**). Por lo tanto, la interfaz Wlan1 tendrá la IP 192.168.1.3 estática.

Fig. 4.1.3 Router

```
auto wlan1
iface wlan1 inet static
address 192.168.1.3
netmask 255.255.255.0
gateway 192.168.1.1
wpa-essid tfgdroncanal
wpa-psk displays
```

Fig. 4.1.4 Configuración IP

Para poder alimentar este router, se prepara una batería aparte que proporcione la alimentación necesaria 12 voltios y 0.5 amperios. (**Fig. 4.1.5**)

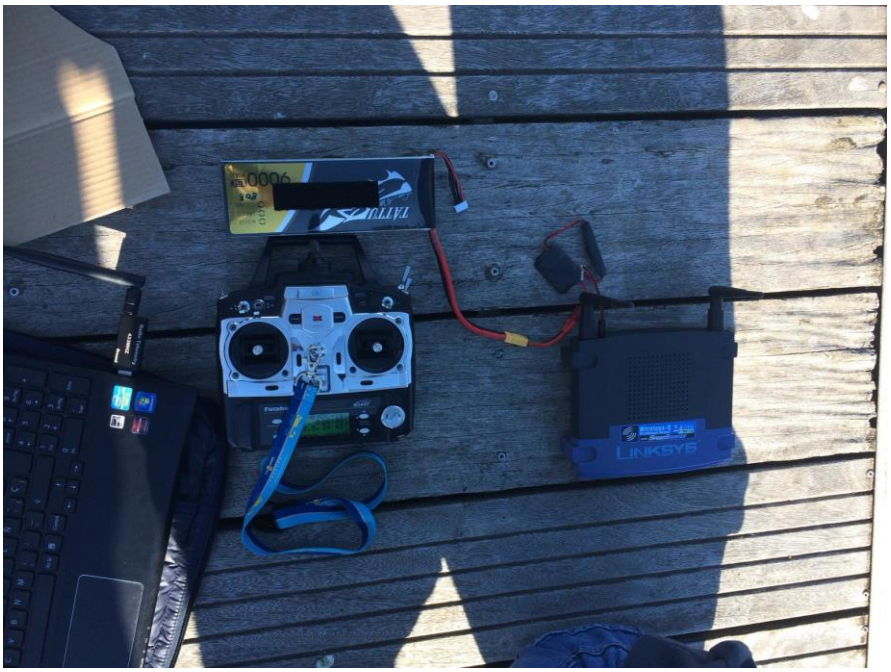


Fig. 4.1.5 Router con batería

```
14:09:02 Read from COM: $SDDPT,3.8,*72
14:09:02 Read from COM: $SDDBT,12.7,f,3.8,M,2.1,F*3A
14:09:02 Parser does not exist Dpt
14:09:02 Read from COM: $YXMTW,12.5,C*14
14:09:02 READ FROM UDP MAVLINK: $GPRMC,153121,A,4128.05514,N,199.19756,E,0.3,201.0,071015,1.3,W*66
14:09:03 Read from COM: $SDDPT,3.6,*7C
```

Fig. 4.1.6 Resultados canal servidor


```
Received: {"context":"vessels.123456789","updates":[{"source":{"device":"SignalkServer","timest
amp":"2018-01-29T19:02:58Z","src":"114","pgn":"12345"},"values":[{"path":"environment","value":
{"water":{"temperature":{"value":12.5}},"depth":{"belowTransducer":{"value":3.8}}}},{"path":"na
vigation.position","value":{"source":{"label":"PilotFish","type":"NMEA0183","src":"vessels.1234
56789.sources.nmea.0183.RMC","pgn":0.0},"timestamp":"2015-10-07T15:31:21Z","longitude":1.991978
,"latitude":41.2805519,"altitude":0.0}}]}]}
```


Fig. 4.1.7 Resultados canal cliente

Para contrastar los datos, se utiliza una aplicación de Google Maps (**Fig. 4.1.8**) para introducir las coordenadas que recibimos en el cliente (**Fig. 4.1.7**) desde el servidor (**Fig. 4.1.6**).

También, consultando la página oficial del canal olímpico podemos comprobar que la profundidad es de unos 4m, tal como indica la prueba.

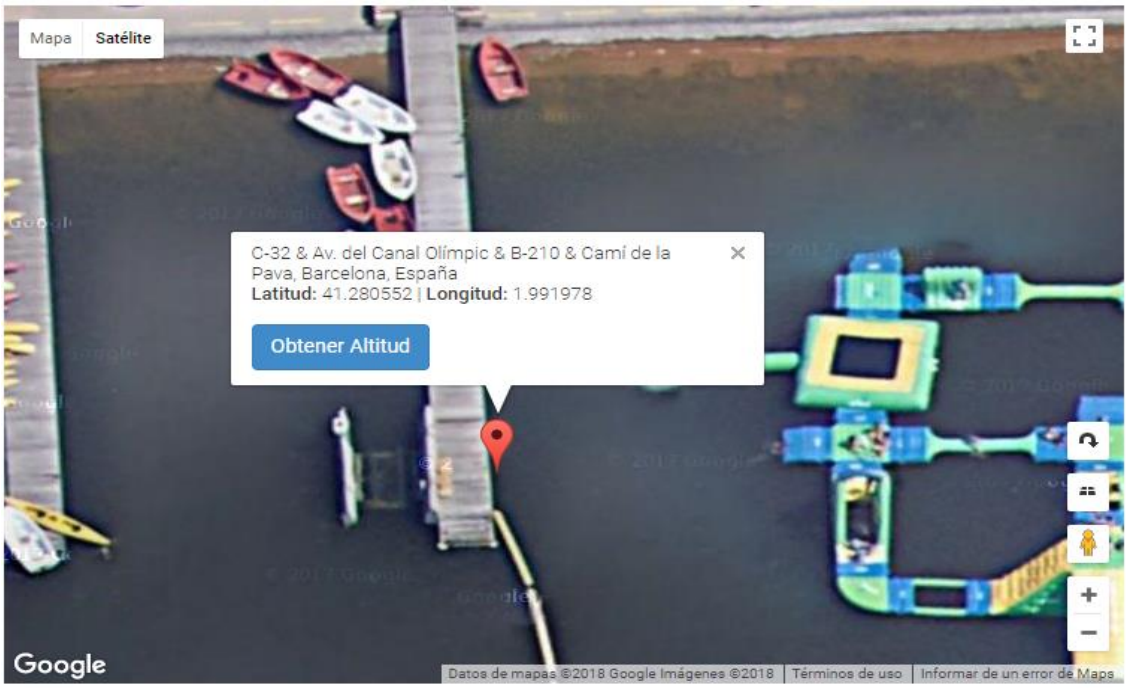


Fig. 4.1.8 Coordenadas Google Maps

4.2 Delta del Ebro

La segunda prueba se hará en el delta del Ebro, dónde dispondremos de más espacio para poder hacer pruebas de rango con la Wi-Fi, puesto que en el canal hay muchas boyas dónde el barco podría atascarse o chocar.

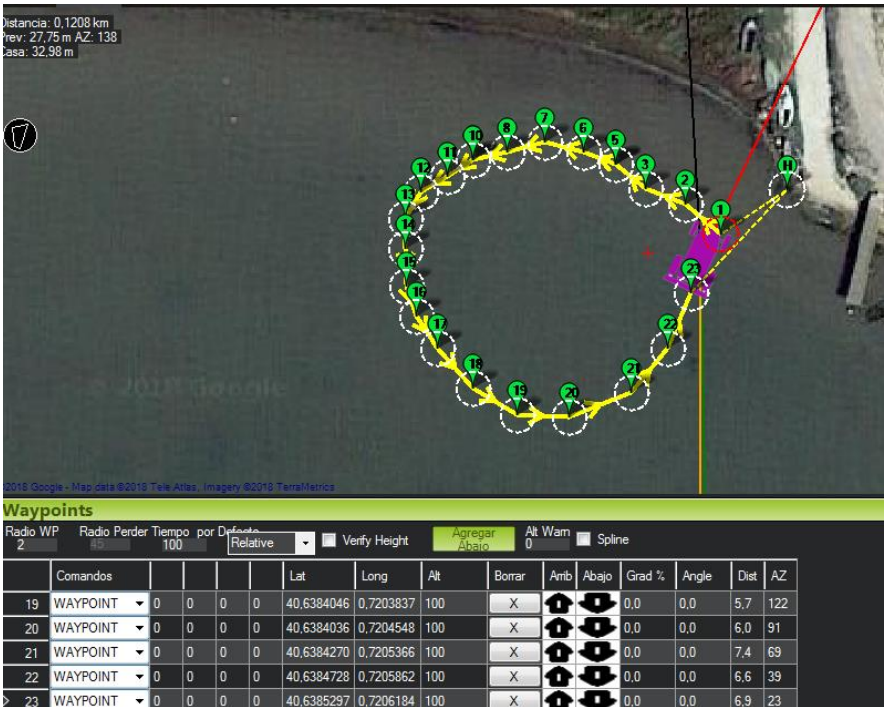


Fig. 4.2.1 Ruta, waypoints MP

Primero se hará una ruta circular con MP (**Fig. 4.2.1**) para poder ver como de abiertas tienen que ser las curvas para que el barco pueda girar.

Finalmente, alejamos el barco manualmente con el radiocontrol hasta perder la cobertura Wi-Fi, el alcance máximo fue 67 metros.

Esta vez solo utilizamos el servidor para tomar los datos, y luego los post procesamos leyendo del log.txt (**Fig. 4.2.3**).

```
Read from FILE: $YXMTW,14.8,C*1F
Read from FILE: $GPRMC,153121,A,4063.86170,N,72.07364,E,0.3,201.0,071015,1.3,W*5B
Read from FILE: $SDDPT,0.3,*7A
Read from FILE: $SDDBT,1.1,f,0.3,M,0.1,F*04
```

Fig. 4.2.2 Server leyendo de Log.txt


```
Received: {"context":"vessels.123456789","updates":[{"source":{"device":"SignalkServer","timestamp":"2018-01-
29T18:59:56Z","src":"114","pgn":"12345"},"values":[{"path":"environment","value":{"water":{"temperature":{"va
lue":14.8}},"depth":{"belowTransducer":{"value":0.3}}}},{"path":"navigation.position","value":{"source":{"lab
el":"PilotFish","type":"NMEA0183","src":"vessels.123456789.sources.nmea.0183.RMC","pgn":0.0},"timestamp":"201
5-10-07T15:31:21Z","longitude":0.7207368,"latitude":40.63862,"altitude":0.0}}]}]}
```


Fig. 4.2.3 Resultados en el cliente

Comprobando las coordenadas en Google Maps podemos ver la exactitud de estos datos (**Fig. 4.2.4**).

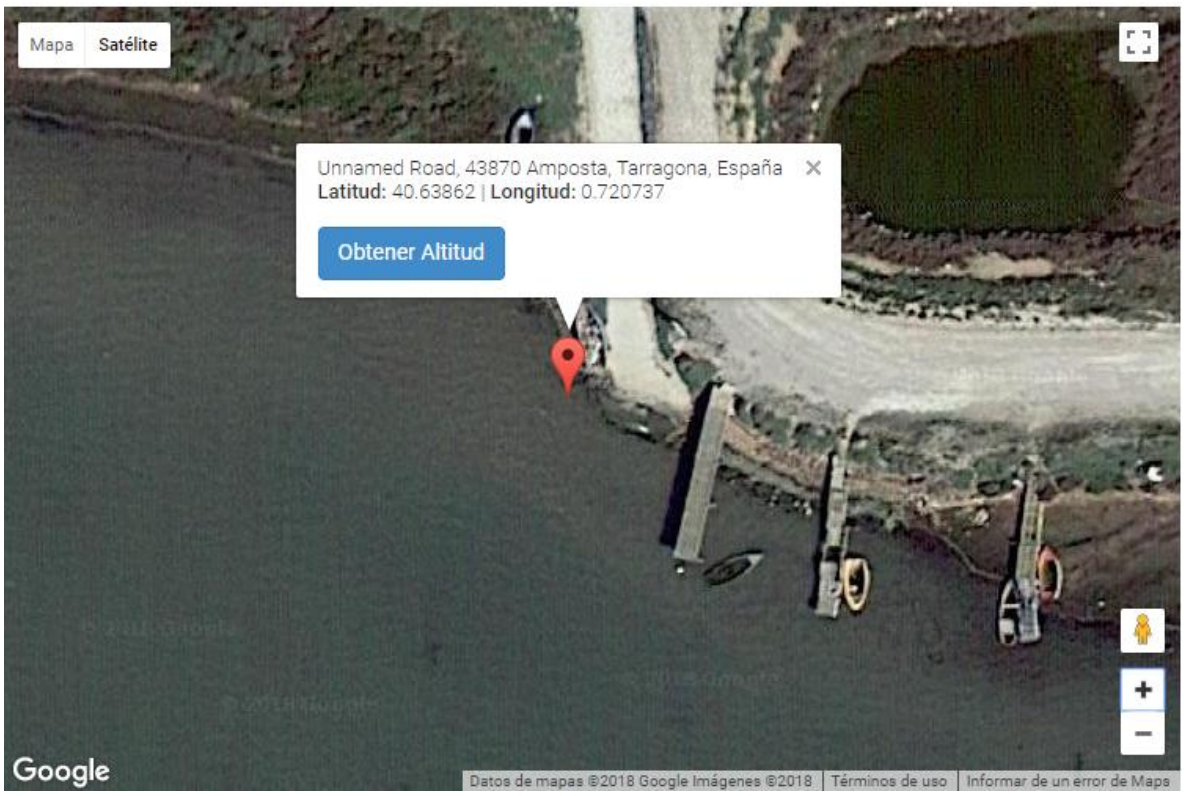


Fig. 4.2.4 Coordenadas Google Maps

Adicionalmente, se han post procesado todas las sentencias NMEA relacionadas con coordenadas para crear un mapa en Google Maps de la ruta que siguió el barco (**Fig. 4.2.5**).

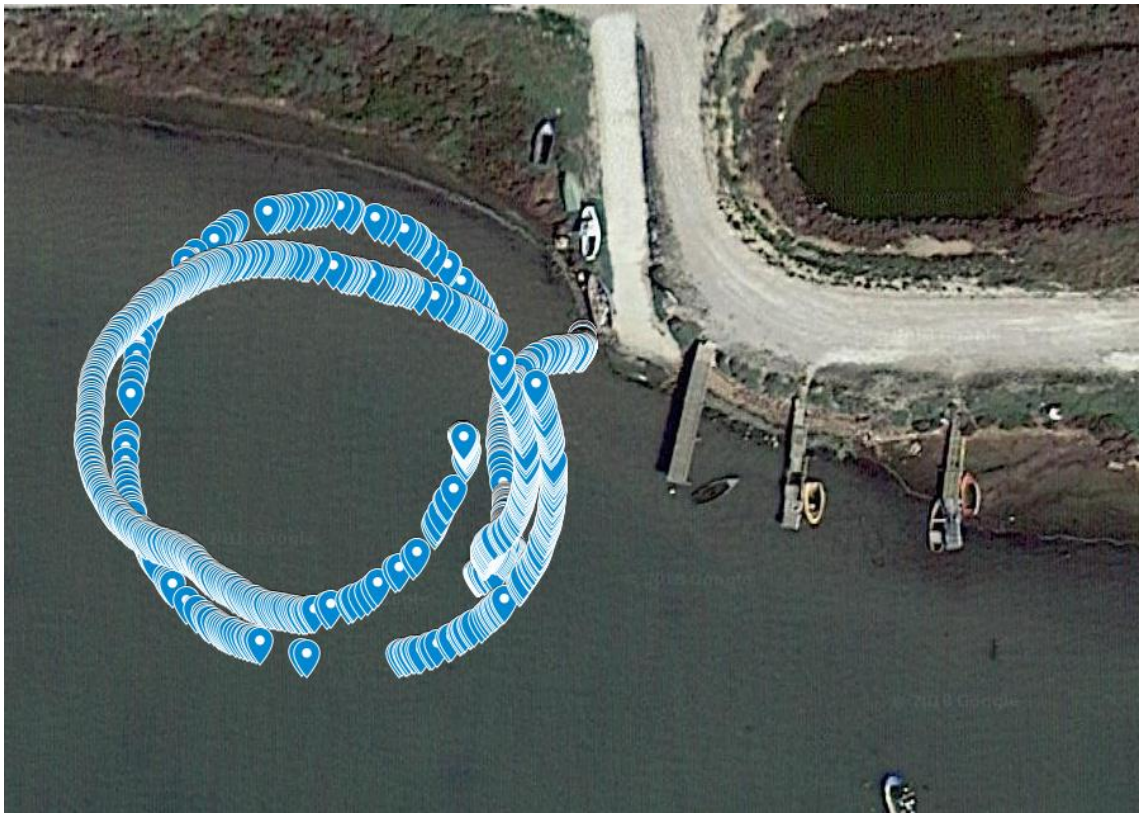


Fig. 4.2.5 Mapa GPS Google Maps

CAPITULO 5. CONCLUSIONES

En este proyecto se ha podido comprobar la funcionalidad de PilotFish para poder obtener los datos del estado actual de nuestra embarcación. Se ha podido incorporar un sensor de profundidad que transmite datos NMEA para poder procesarlos y post procesarlos, y se han podido generar datos NMEA a través de datos procesados de otra fuente.

Por último, y puesto que este es un proyecto que puede seguir creciendo en un futuro, propondré algunas mejoras al barco.

La estructura actualmente hecha de aluminio hay que cuidarla puesto que poco a poco las grapas pueden ir abriéndose debido al peso que ejercen las cajas laterales. Como se ha comentado en el apartado del diseño, el mejor material para la estructura hubiese sido fibra de carbono, puesto que es mucho más ligero, y más resistente.

Otro problema actual es el rango de transmisión cliente-servidor que ofrece la Wi-Fi. Para solventar esto, se podría añadir cobertura 4G, lo que daría muchísimo más alcance.

Como última mejora, se podrían añadir otro tipo de sensores para un mejor funcionamiento autónomo como, por ejemplo, un detector de obstáculos.

REFERENCIAS

http://ardupilot.org/ardupilot/index.html

http://www.gpsinformation.org/dale/nmea.htm

http://signalk.org/

http://qgroundcontrol.org/mavlink/start

http://www.airmartechnology.com/xref.html

ACRÓNIMOS

GPS	Global Positioning System
NMEA	National Marine Electronics Association
MAVLink	Micro Air Vehicle Link
MAVProxy	Micro Air Vehicle Proxy
MP	Mission Planner
GCS	Ground Control Station
USB	Universal Serial Bus
ESC	Electronic Speed Controller
BEC	Battery Elimination Circuit
SD	Secure Digital
SITL	Software in the Loop
JSON	JavaScript Object Notation
RTL	Return to Launch
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Tab. 1.1 Acrónimos utilizados en el proyecto