industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Avances en el desarrollo de un sistema de control para la navegación de un catamarán de forma autónoma

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Jose Carlos Urrea Celdrán**
Director: Miguel Almonacid Kroeger
Codirector: José Manuel Cano Izquierdo

Cartagena, 31 de octubre de 2020

Universidad Politécnica de Cartagena

Resumen

En la última década han surgido muchos desarrollos de electrónica para el consumo enfocados a aficionados y estudiantes, que hábilmente han sabido adaptar en un campo como el marítimo, que comercialmente tiene estructura de monopolio en cuanto a tecnologías y estándares de comunicaciones.

A la misma vez, los avances en IA (Inteligencia artificial) y visión artificial hacen ya posibles la navegación autónoma a modo experimental de distintas embarcaciones para objetivos de investigación y afianzamiento de estas tecnologías.

Este proyecto sigue las líneas de las tecnologías actuales para desarrollar un piloto automático para cualquier embarcación que se propulse con dos motores en los laterales, sean eléctricos o no, y que carezcan de timón. Pasando por un estudio de las tecnologías actuales y las herramientas disponibles con una perspectiva de futuro.

Abstract

In the last decade arrived many developments in consumer electronics focused on DIY amateurs and students, which they have skillfully adapted in a field such as the maritime, which is commercially a monopoly structure in terms of communication technologies and standards.

At the same time, advances in AI (Artificial intelligence) and computer vision make autonomous navigation possible in an experimental way of different vessels for research purposes and the consolidation of these technologies.

This project follows the lines of the current technologies to develop an automatic pilot for any boat that is propelled with two motors on the sides, whether is electric or not, and lacks a rudder. Going through a study of the current technologies and available tools with a future perspective.

Índice de contenido

Índice de Tablas y Figuras

1. Introducción

1.1. Antecedentes

Este proyecto se nutre de anteriores trabajos desarrollados en las distintas escuelas de ingeniería de la Universidad Politécnica de Cartagena (Navales, Telecomunicaciones e Industriales). Todos estos trabajos están conectados entre sí, ya que forman parte de un equipo que desarrolla un sistema de embarcación autónoma bajo el nombre de VNAS (Vehículo de Navegación Autónoma en Superficie).

En los trabajos realizados durante los cursos anteriores se desarrollaron diversos sistemas, como el diseño e implementación de un sistema de radiocomunicaciones para la transmisión de la telemetría entre tierra y el barco, o una aplicación web de telemetría donde recibir estos datos por parte de la escuela de Telecomunicaciones, así como un modelo en MATLAB para el comportamiento y la navegación del catamarán por parte de la escuela de Industriales.

El proyecto VNAS actual consta de un prototipo de catamarán realizado en la Escuela de Navales, este prototipo posee dos motores eléctricos como método de desplazamiento y navegación. Es en este prototipo donde se pretende validar el diseño final y toda la experiencia acumulada en estos trabajos. Se complementa con un compás magnético, antena GPS, anemómetro con veleta y un sensor bajo agua para la velocidad y temperatura. El anexo completo de componentes se encuentra en (1), aunque se hablará en este trabajo de los componentes esenciales más adelante.

El objetivo final de este grupo de trabajo inter-escuelas, el VNAS, es participar con la embarcación desarrollada en el Microtransat Challenge. El Microtransat Challenge (2) es una regata transatlántica para embarcaciones autónomas. La regata tiene como objetivo estimular el desarrollo de embarcaciones autónomas a través de una competición amistosa. Las embarcaciones están limitadas a una eslora total máxima de 2,4 metros de largo. Y existen dos divisiones, la autónoma que no permite el envío de información desde tierra, y la no tripulada que si lo permite.

Se recomienda de cara a analizar este proyecto leer previamente los siguientes trabajos del grupo VNAS indicados en la bibliografía (1) (3) (4).

1.2. Objetivos

El objetivo final de este trabajo es **desarrollar un sistema totalmente funcional y ajustable de piloto automático para la navegación autónoma de cualquier embarcación que se desplace mediante dos motores eléctricos**

Haremos uso de los materiales ya disponibles en el proyecto VNAS (cuyo logo se muestra al final de esta página), apoyándose en los trabajos previamente realizados. El sistema realizado tiene que ser lo suficientemente auto explicativo y de fácil acceso en su interfaz, y sobre todo en su código fuente, para que sea modificado y mejorado en el futuro por los estudiantes que recojan el testigo, añadiendo nuevas funcionalidades y mejoras.

Cuando nos referimos anteriormente a que en este trabajo el sistema de piloto automático tiene que ser completamente autónomo, nos referimos a que sea capaz de navegar sin ninguna intervención por parte de un operador desde el inicio de la navegación hasta la llegada al final de la ruta. Solamente habría intervención para iniciar la ruta y para cuando suceda algún error inesperado.

Para referenciamos al nivel de automatización adoptado en este piloto automático, nos basamos en la escala de automatización adoptada por la SAE (Sociedad de Ingenieros de Automoción) para clasificar el nivel de automatización de los distintos vehículos en 5 niveles disponibles (5), siendo 5 el más alto grado de automatización. Es el nivel 4 el que se pretende alcanzar en este trabajo. Según la definición se podría considerar un sistema en el que no es necesaria la supervisión de un operador para eventos de seguridad, y la navegación está restringida a áreas limitadas espacialmente o bajo circunstancias especiales. Fuera de estas áreas o circunstancias la embarcación debe ser capaz de abortar la navegación o mantenerse fija en unas coordenadas si no se mandan otras instrucciones.

Para que el sistema cumpla el objetivo final marcado, debe ser capaz de cumplir los siguientes subobjetivos: **adquirir los datos de la telemetría** y **una ruta de navegación proporcionada a través de una conexión online**, ejercer el **control adecuado sobre los motores**, gestionar y tomar **diferentes acciones ante errores o fallos**.

Figura 1 - Logo VNAS

1.3. Estado del arte

Durante los últimos años se han ido produciendo cambios técnicos significativos en varios aspectos en el campo de los transportes. Muchos de estos cambios son casi siempre propiciados por los nuevos avances tecnológicos que traen los incipientes desarrollos de universidades y empresas. Siempre se ha intentado desde el punto de vista de la ingeniería automatizar, agilizar y mejorar todos los tipos de vehículos, en nuestro caso embarcaciones, es por eso que en las últimas décadas se observa cómo se está produciendo una mayor automatización de los medios de transporte con la introducción de sistemas de visión artificial cada vez más avanzados, y la mejora en los algoritmos de detección de estos mismos sistemas.

Un caso muy representativo y mediático en los últimos años, son las ASDS (Plataformas autónomas de puerto aeroespacial) de SpaceX (el cual se muestra en la Figura 2), una compañía aeronáutica estadounidense, que para aterrizar sus cohetes o para recuperar partes del fuselaje con paracaídas, usan barcazas automatizadas. Estas barcazas se desplazan autónomamente desde sus zonas de atraque hasta zonas de altamar alejadas de la costa y son capaces de mantenerse estáticas en unas coordenadas fijadas con un error de posicionamiento de tan solo 3 metros, incluso en condiciones de tormenta, valiéndose para ello de la información GPS y cuatro motores a Diesel.

Figura 2 - ASDS Of Course I Still Love You

El uso de estas barcazas autónomas permite recuperar en alta mar estos cohetes sin necesidad de operarios a bordo, ya que si no fueran autónomas podría representar un peligro mortal para los operarios, ya que no hay lugar seguro en la barcaza si se da cualquier explosión o fallo durante la recuperación de los cohetes en el aterrizaje.

Desde otra empresa, el recién lanzado trimarán totalmente autónomo de IBM llamado Mayflower pretende cruzar el atlántico desde la costa de Plymouth en el Reino Unido, hasta su contraparte, Plymouth en Massachussets. Según la propia IBM ha sido diseñado para proporcionar una forma segura, flexible y rentable de recopilar datos sobre el océano, trabajando en conjunto con científicos y otras embarcaciones autónomas para ayudar a comprender problemas críticos como el calentamiento global, la contaminación de micro plásticos y la conservación de mamíferos marinos.

Aparte lleva una inteligencia artificial a bordo para tomar decisiones en el mar, valiéndose de la telemetría y 6 cámaras de visión artificial para detectar posibles peligros en el horizonte, hacer decisiones y cambios de curso basándose en datos en tiempo real, haciéndolo un vehículo de autonomía de nivel 5, la máxima posible (6). En la Figura 3 se puede observar el Mayflower durante sus pruebas en Reino Unido.

Figura 3 - Trimarán Mayflower

Mientras que estos ejemplos no cumplen las características para ser considerados en el Microtransat, presentan sin embargo los avances tecnológicos necesarios para hacerlo perfectamente viable en el marco actual. Por lo que la aparición de un vehículo autónomo capaz de realizar esta pequeña proeza no dista mucho, y con este proyecto se quiere ayudar a ello.

2. Desarrollo

2.1. Concepto y materiales

El concepto inicial del que parte el proyecto VNAS es crear un barco autónomo, y a partir del primer prototipo que se desarrolle, enfocar futuros desarrollos para la competición trasatlántica.

A continuación, vamos a enumerar los puntos básicos que se van a abordar en el desarrollo del piloto automático:

- Adquisición de telemetría (Posición y orientación) a través de una gran variedad de tipos de sensores y tipologías.
- Gestión de rutas de navegación propia y online.
- Control PID de motores a través del rumbo. Ajuste fino personalizable y fácilmente ampliable.
- Hardware asequible y comercial.
- Código versátil y multiplataforma.

En cuanto al control general, se puede observar el concepto básico en el que nos basamos. El barco sería autónomo, pero puede y debe ser intervenido por un operario en caso de errores no salvables (como desconexión física o fallo de componentes).

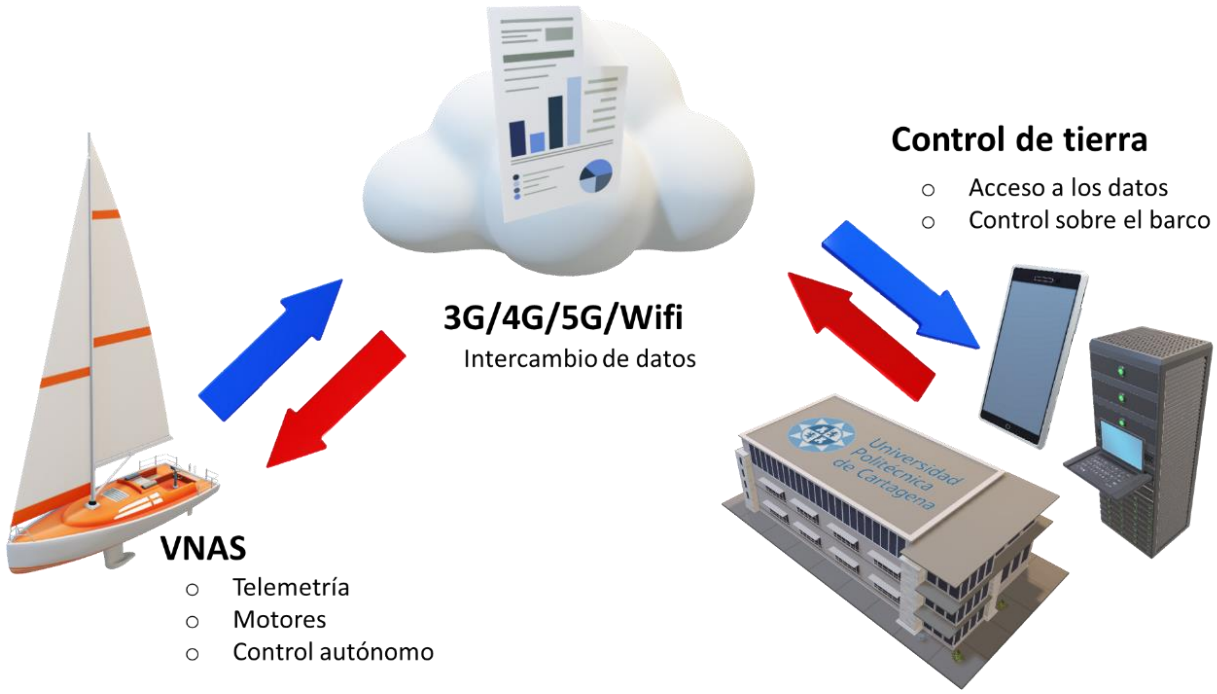


Figura 4 - Flujo de control

En cuanto a la elección de materiales, partimos de los materiales heredados de los anteriores TFG, de los cuales hemos usado los que están en la siguiente tabla.

Materiales
Antena GPS GARMIN 19X 10Hz NMEA 2000
Compás ciego AIRMAR H2183 NMEA 2000
Backbone N2K y cables de conexionado
RPi 3B con pantalla táctil y shield PICAN2 de conexión a bus CAN

Tabla 1 - Lista de materiales heredados

Recurriendo a lo investigado y planteado inicialmente en otros TFG, se había sugerido el uso de dos unidades de computación Raspberry Pi 3 Modelo B, una como sistema chartplotter (con la aplicación myPlotter del profesor Humberto Martínez Barberá recogiendo la telemetría), y otra Raspberrry Pi haciendo las funciones de piloto automático y control. Pero viendo las posibilidades de computación del hardware Raspberry Pi 3 durante el desarrollo de este proyecto, se decidió usar un único sistema Raspberry Pi, siendo este una Raspberry Pi 3 Modelo B+, con algunas mejoras (De ahora en adelante usaremos “RPi” para hacer alusiones a la Raspberry Pi 3 Modelo B+). La cual se puede ver en la Figura 5.

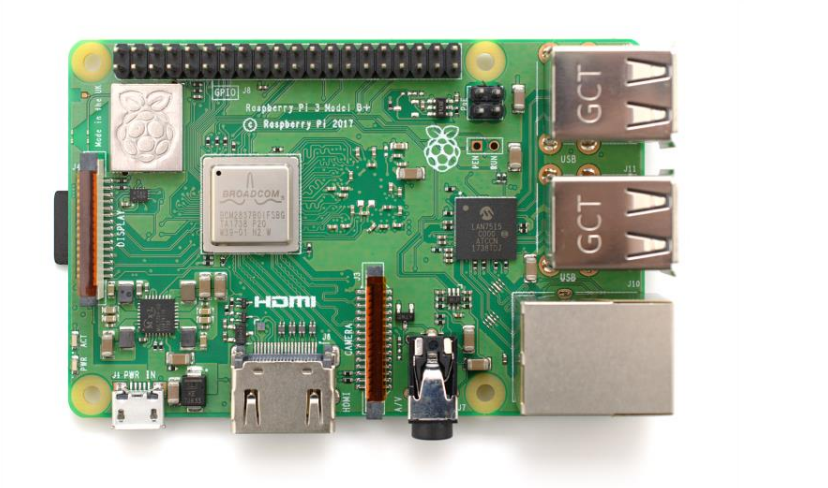


Figura 5 - Raspberry Pi 3B+ (7)

Anteriormente el sistema que se había proporcionado por el profesor Humberto corría sobre un SO que se ejecutaba desde el sistema de almacenamiento principal de la RPi, una tarjeta microSD. Y esto técnicamente supone ciertas desventajas que se desgranan a continuación:

- Ejecutar un SO sobre una tarjeta SD es contraproducente; Los sistemas de almacenamiento SD en general son sistemas que están especialmente diseñados para leer y escribir datos de forma secuencial, es decir, cámaras de fotos y video, sistemas que leen o escriben datos uno detrás de otro y que no requieren de acceso rápido y frecuente a datos aleatorios, es decir, que no están escritos seguidamente. Al ejecutar el SO se produce un cuello de botella en las órdenes

del SO al escribir datos o leer, ya que no suelen estar convenientemente escritos los datos de forma secuencial, si no que están almacenados de forma aleatoria, además que el SO no está adaptado para funcionar en un soporte secuencial.

- Y no solamente el problema reside en el cuello de botella que limita la velocidad de ejecución de nuestro SO, si no también que se llegan a producir con cierta frecuencia corrupción de datos y ficheros, ya que ni el gestor de I/O del SO, como el sistema de almacenamiento están optimizados para funcionar de forma aleatoria, y esto puede ser catastrófico para nuestro VNAS, ya que requiere de cierta fiabilidad y robustez, y no queremos ni podemos permitir que se cuelgue en alta mar por un error de este tipo.

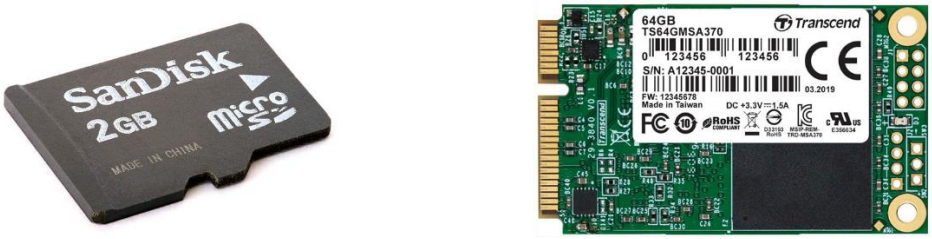
- Un acceso continuado (sobre todo de escritura) a una tarjeta microSD genera una notable cantidad de calor en una superficie muy pequeña en comparación una memoria SSD, lo que puede llevar a rotura física de la propia memoria microSD en un entorno cerrado y sellado como sería la caja que contiene al sistema RPi, sin posibilidad de evacuación de ese calor generado. Si bien es cierto que existen tarjetas microSD de grado industrial, como la que se muestra en la Figura 7, incluso con un rango de temperaturas de funcionamiento de -40ºC a 85ºC, el problema de desempeño se sigue manteniendo.

Figura 6 - microSD grado Industrial (8)

Para solucionar estos problemas se escogió la revisión B+ de la RPi 3, la cual ya tiene activada de fábrica por defecto el bit de arranque externo por USB. Si utilizáramos cualquier modelo anterior tendríamos que realizar un inicio desde una tarjeta SD especial para poder modificar el bit de arranque en la memoria de solo lectura (OTP) de la RPi.

Optar por la RPi3 B+ nos permite usar cualquier otro tipo de almacenamiento físico no secuencial que disponga de conectividad o adaptador USB como, por ejemplo, discos duros de estado sólido (SSD) o memorias USB.

Estos problemas se han resumido y sintetizado en la siguiente Figura 7.



microSD (SDHC/SDXC)

- Optimizada para multimedia
- Lectura secuencial rápida
- Lectura aleatoria lenta
- Velocidad de transferencia global de datos baja
- Bajo consumo

SSD (Solid State Drive)

- Optimizada para SO
- Lectura secuencial y aleatoria rápida
- Velocidad de transferencia global de datos alta
- Bajo consumo

Figura 7 - Comparación microSD vs SSD (9)

Como es de esperar, se ha tomado en cuenta para esta elección que el medio de almacenamiento microSD es un sistema de bajo consumo ya de por sí, con un consumo medio de 300mA a 3.3V (1 W), por lo que hemos acudido una gama SSD con un consumo bajo, de los que menos consume actualmente, el SSD mSATA que funciona al mismo voltaje 3.3V. La unidad que hemos adquirido para este proyecto tiene un consumo máximo de 800mA (2,64W), un precio realmente pequeño a pagar por los beneficios en robustez, rapidez y fiabilidad que obtenemos a cambio de este incremento en el consumo. Hay que tener muy en cuenta que estos consumos que se mencionan son los medidos por el fabricante a máxima actividad en el soporte de almacenamiento, por lo que el consumo medio real será más bajo.

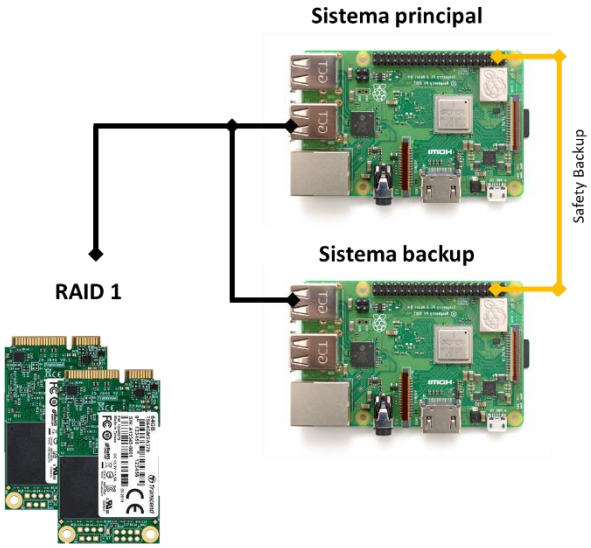


Figura 8 - Posible sistema redundante (7)

Anteriormente se ha mencionado el uso de dos sistemas RPi, uno como chartplotter, y otro como piloto y central de control. RPi tiene capacidad de procesamiento suficiente para correr estos dos sistemas con holgura manteniendo una carga de CPU aceptable por debajo del 50%. Es justo decir que el uso de dos sistemas RPi se vería más que justificado en el caso de crear un sistema de redundancia/backup para que, en el caso de fallo de una RPi, entre el otro sistema en funcionamiento.

Para obtener los datos de telemetría hay que conectar la RPi al backbone N2K para poder leer las sentencias de los distintos dispositivos. Se pueden observar en la Figura 10.

El primer obstáculo es conectar físicamente el Bus CAN a la RPi, ya que de fábrica viene con conectores USB-A y las conectividades I2C, SPI y UART. Pero ninguno de ellos sirve para conectarse directamente a un bus de estas características.

Figura 9 - Conectores N2K

En los TFG anteriormente mencionados se había usado un desarrollo del tipo shield para RPi, que permitía enviar y recibir datos de un Bus CAN. Pero después de una evaluación del sistema, se observó que disponía de un soporte pobre y documentación escasa como para poder validarlo para este nuevo desarrollo, es por ello que se evaluaron otras opciones.

Al adoptar una puerta de enlace (Gateway) que transmita las sentencias de N2K por USB, nos da una mayor libertad en cuanto al soporte/hardware elegible, ya que funciona con cualquier dispositivo con conexión USB al contrario que el Shield de RPi, pudiendo salir de RPi en un futuro desarrollo manteniendo mismo código y periféricos, lo que es una gran ventaja en el desarrollo del prototipo.

La alternativa que más peso y beneficios traía es el Actisense NGT-1, una interfaz o puerta de enlace entre el Bus CAN NMEA2000 y un conector USB-A. Nos permite conectar el bus a casi cualquier tipo de dispositivo, ya que la conectividad USB es la más extendida en el mundo actualmente, al contrario que el shield anteriormente mencionado.

Con esto logramos la capacidad de leer y escribir mensajes en formato N2K en el bus, y a todos los sensores conectados a él.

Figura 10 - Actisense NGT-1

En la Figura 10 se muestra su aspecto, aparte es compatible con un gran número de dispositivos comerciales marítimos. Viene con aislamiento galvánico para evitar cualquier lazo de tierra en nuestro dispositivo, ya que el lado N2K funciona de 9 a 35V, mientras el lado USB funciona a 5V. Con una carcasa de ABS retardante de llama.

También debemos pensar en un encapsulado o caja de protección para nuestra RPi. Las condiciones en el mar nos obligan a adoptar el uso de una caja estanca, con conectores de alimentación y USB empotrados y protegidos contra el agua. No había ninguna caja previamente para este cometido, por lo que se ha adquirido una caja de reducidas dimensiones (212x123x60mm) de sixfab con una IP65. Además, si queremos usar la RPi in situ debemos incluir una pantalla lo suficientemente grande para poder visualizar el piloto automático, así como un teclado y ratón.

Hemos recurrido a una pantalla táctil TFT de 800x480 por HDMI de Adafruit, para alimentarla se necesita conectarla por USB (10). Y hemos adquirido el teclado inalámbrico con pad táctil más pequeño que se ha encontrado.

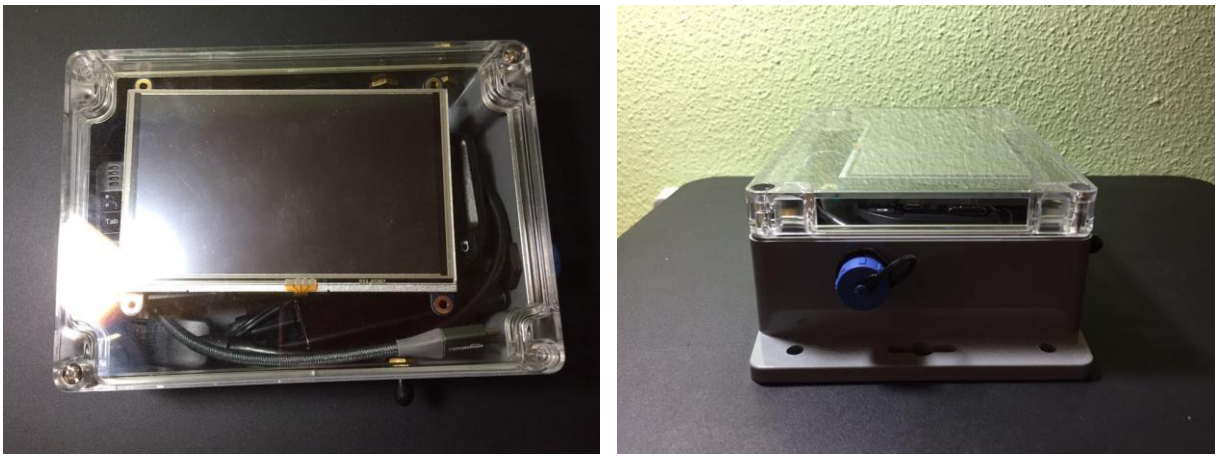


Figura 11 - Caja piloto automático VNAS

En la Figura 11 de arriba se aprecia el aspecto de la caja de sixfab con todo el material en su interior. Tanto la pantalla como el teclado deben de estar siempre desconectados en caso de que no se usen, como todos los dispositivos electrónicos, generan calor, y puede ser catastrófico en la caja una vez cerrada. Se ha instalado una carcasa de aluminio disipadora de calor a la RPi con dos ventiladores para refrigerar tanto el procesador como la unidad de procesamiento gráfica en la parte posterior de la placa. A continuación, se aprecia en la Figura 12 lo que contiene la caja de sixfab.

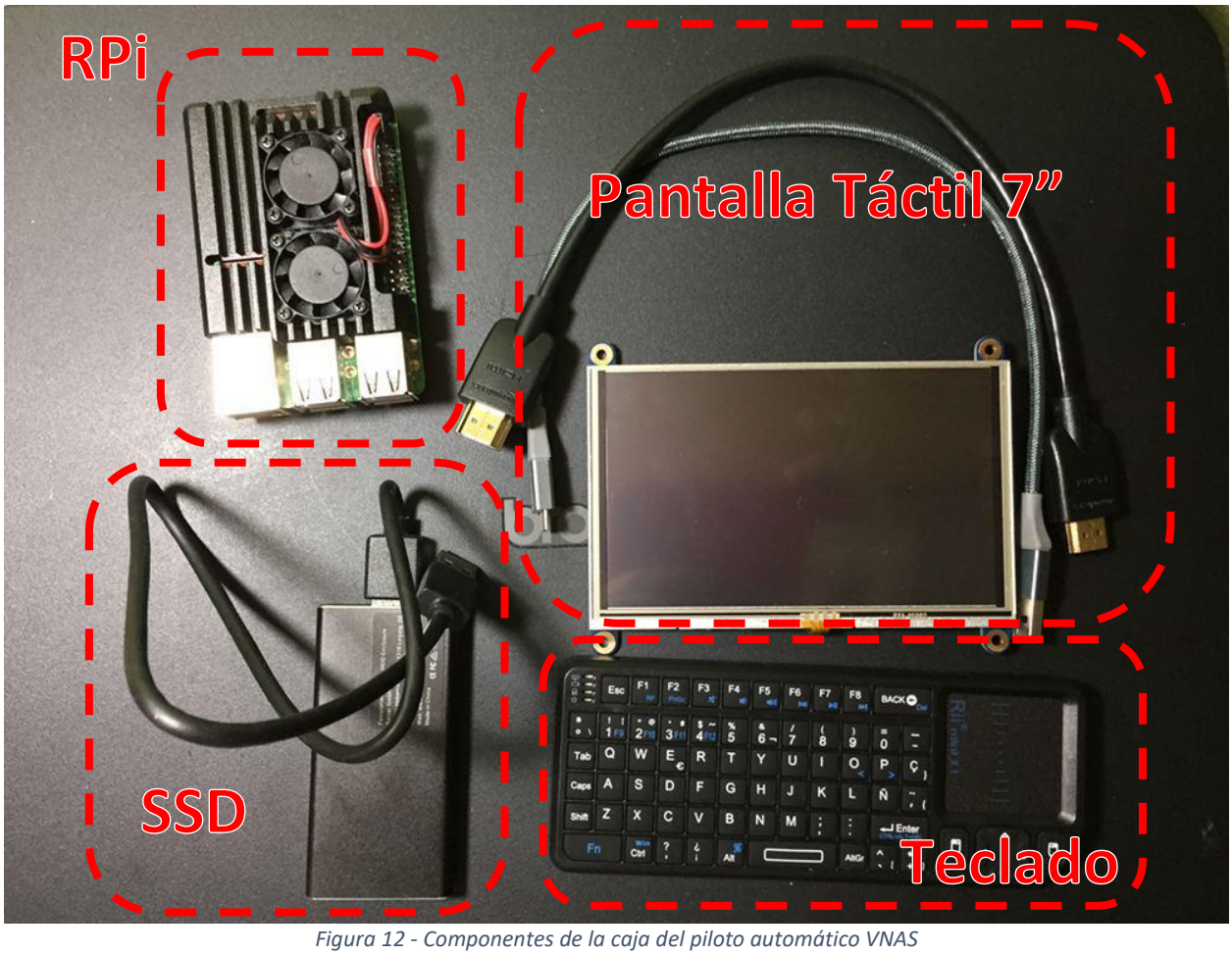


Figura 12 - Componentes de la caja del piloto automático VNAS

2.1. Chartplotter

Para poder crear y organizar rutas a seguir por nuestra embarcación, existen a la venta equipos comerciales para instalación en embarcaciones de “chart plotters”, ordenadores con pantalla que muestran un mapa con la ubicación de nuestros alrededores, así como la posibilidad de planificar y crear rutas a seguir por nuestra embarcación. Por sí solos no funcionan, ya que necesitan estar conectados a la red de señales del barco, y opcionalmente a un piloto automático, normalmente este piloto es otro sistema aparte al que se conecta el plotter.

Hemos decidido recurrir al mayor chartplotter de código libre que hay actualmente. OpenCPN (11) en su versión 5, el cual está escrito en C y C++. Su aspecto con cartas náuticas instaladas es tal como se ve en la siguiente Figura 13.

Figura 13 - OpenCPN mostrando el puerto de Cartagena

OpenCPN funciona sobre el estándar NMEA0183, y es capaz de emitir y generar las sentencias NMEA0183 necesarias para transmitir toda la información de las rutas que se crean en el plotter a través de UDP (User Datagram Protocol) o TCP (Transmission Control Protocol). Lo que nos sirve como planificador de rutas para nuestro VNAS.

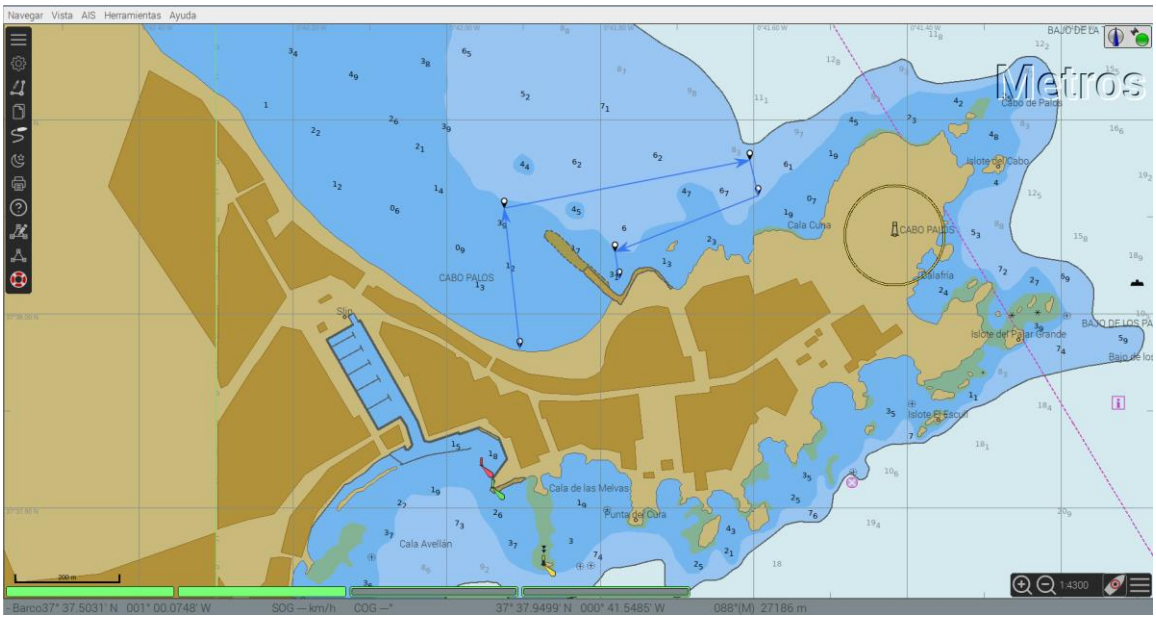


Figura 14 - Una ruta creada en OpenCPN

En la Figura 14 se puede observar una ruta creada en el mapa, con tan solo activarla, se manda a nuestro piloto automático periódicamente la información actualizada para que el VNAS la siga. OpenCPN se configura para que emita toda la información por UDP y que la recoja nuestro VNAS. Por lo que deberá ejecutarse en conjunto con el piloto automático para lograr el funcionamiento deseado.

OpenCPN lleva un mapa global vectorizado básico que no permite realizar planificaciones de ruta en detalle cerca de las costas, es por eso que es necesario adquirir mapas marítimos. Existen muchos mapas gratuitos, pero hemos decidido comprar las cartas para España y poder realizar rutas en cualquiera de sus costas.

OpenCPN Encrypted System Electronical Nautical Charts (oeSENC) son las cartas vectorizadas más económicas de las que dispone OpenCPN, y están manejadas por ellos mismos. Las cartas están vinculadas como máximo a dos dispositivos, y pasado un año no se pueden vincular a ningún otro dispositivo, es más, si se cambia el SO o la plataforma, dejarán de ser válidas, por lo que es importante la imagen que se del SSD de la RPi. Hay que tener esto en cuenta antes de realizar modificaciones drásticas. Abajo se muestra la ruta de los mapas, y el aviso que se da al abrir OpenCPN.

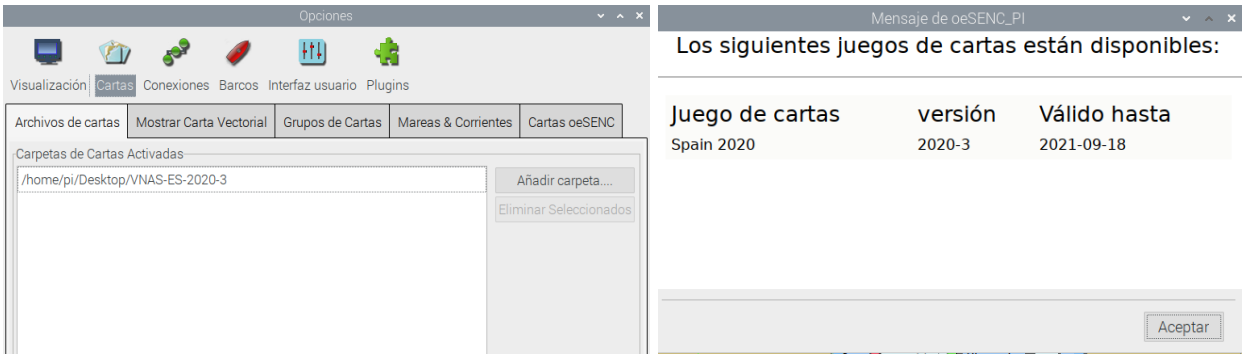


Figura 15 - Configuración y aviso de las cartas oeSENC

En las cartas vectoriales se puede alternar entre las distintas secciones y zonas:

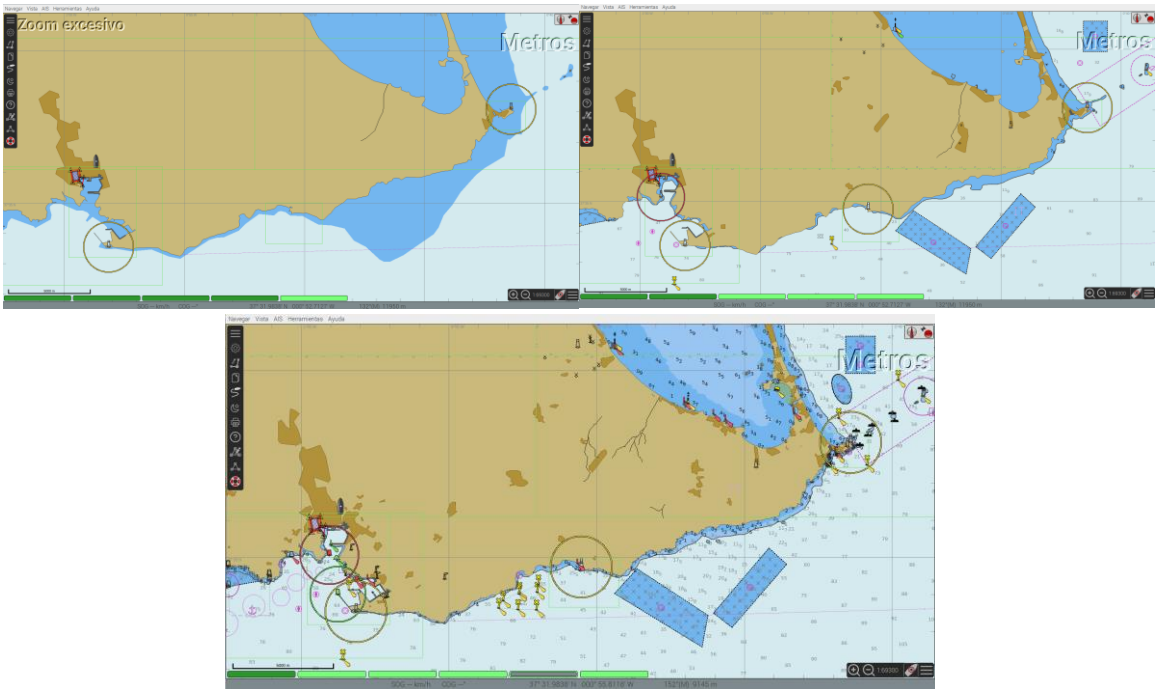


Figura 16 - Modos de carga y visualización de las cartas náuticas

Arriba se observan las distintas capas de los mapas vectoriales. Aparte es necesario tener instalado el plugin oeSENC para cargar las cartas correctamente.

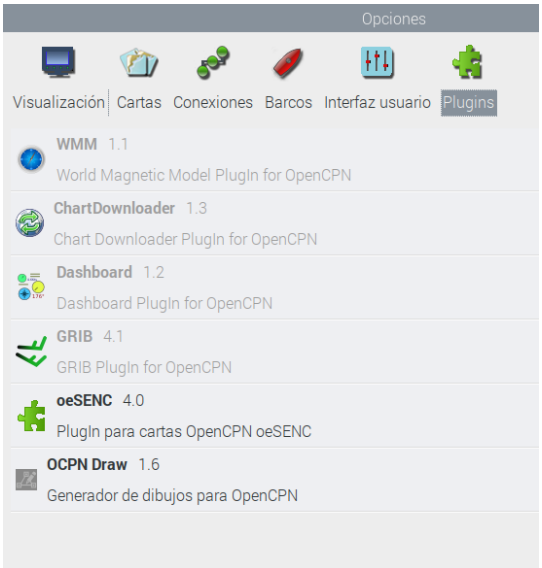


Figura 17 – Configuración de Plugin OCPN

Como se ve en la anterior imagen, usamos también el plugin OCPN Draw. Lo que nos permite crear zonas de inclusión, en la cual nuestro barco se mantendría dentro.

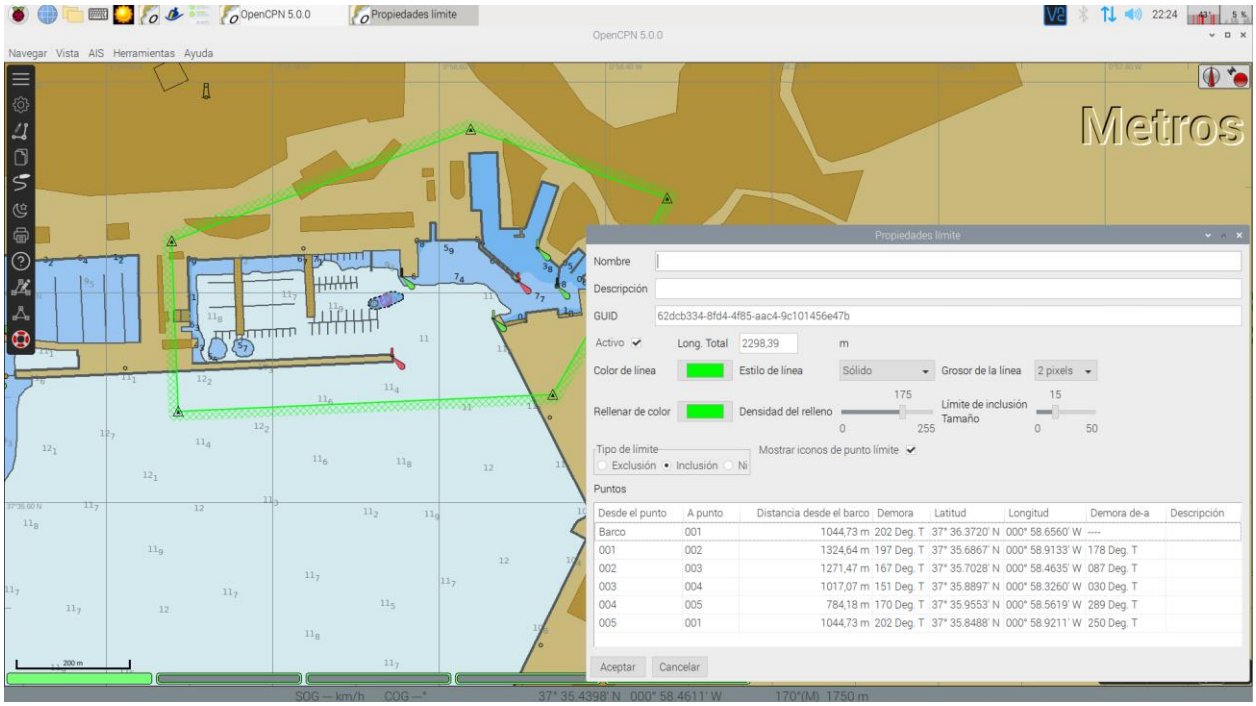


Figura 18 - Zona de Inclusión en OpenCPN

Y zonas de exclusión, para evitar el acceso, ya sea en forma de áreas o líneas como en la siguiente imagen.

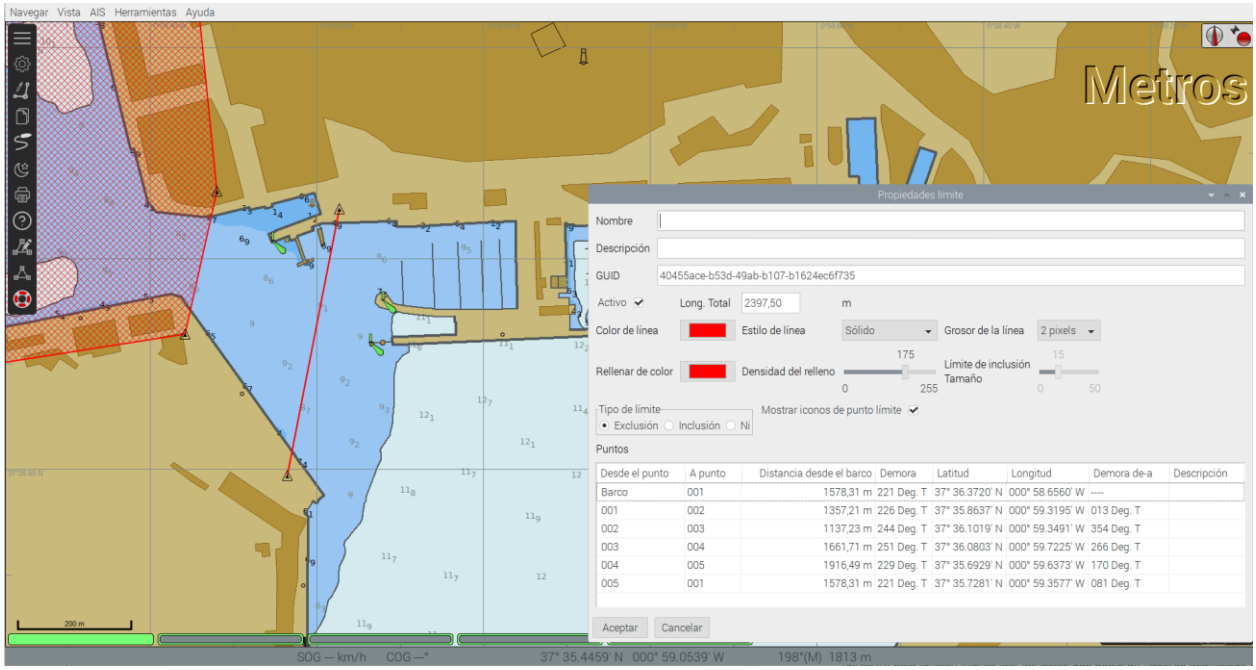


Figura 19 - Zona de Exclusión en OpenCPN

Las zonas de inclusión se pueden usar en conjunto con las de exclusión, teniendo prioridad las de inclusión por encima de las de exclusión.

Puede resultar muy útil en futuros desarrollos para la creación automática de Waypoints entre origen y destino.

La configuración de las conexiones es como sigue a continuación, se añaden las conexiones en la pestaña correspondiente.

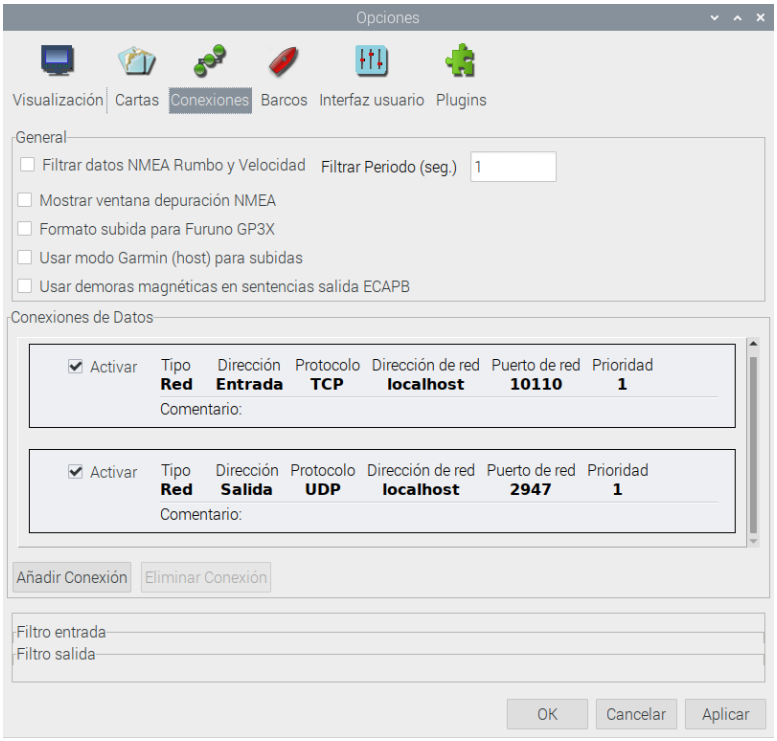


Figura 20 - Pestaña de conexiones

La conexión de entrada de telemetría se hace desde la misma RPi, no se obtiene desde una dirección externa, por lo que usamos localhost y el puerto por el que esperamos recibir la información, marcando la casilla de recibir entradas en ese puerto.

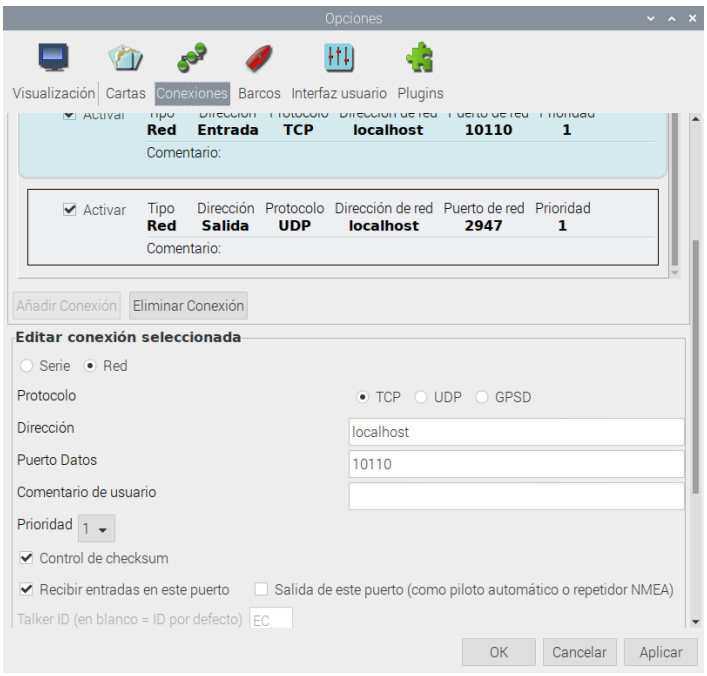


Figura 21 - Parámetros de conexión de entrada

Para la conexión de salida de señales, vamos a configurarla como UDP para enviar la telemetría al piloto automático. Como el envío es a la propia RPi, la dirección vuelve a ser localhost. Y el puerto de escucha del VNAS es el 2947. Además de marcar la casilla de salida. Solo necesitaríamos transmitir las sentencias RMB y HDT desde el propio OpenCPN, tal y como se muestra aquí.

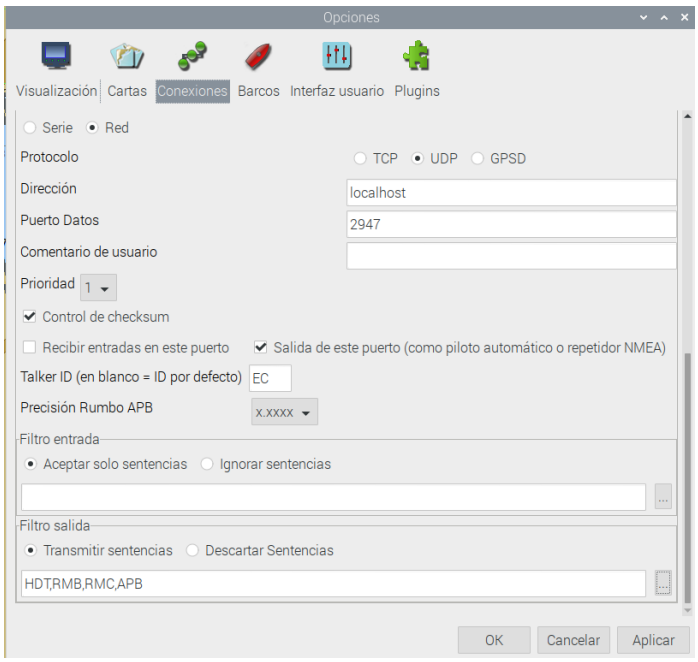


Figura 22 - Parámetros de conexión de salida

Si se deseara controlar desde tierra a VNAS, se podría instalar OpenCPN en Windows, Linux, Mac u otra RPi y configurar las conexiones con las IP o dominios dinámicos establecidos para tal uso, recibiendo datos desde la RPi del barco la telemetría, y enviando datos de vuelta de ruta.

Se han hecho pruebas en laboratorio exitosamente, pero el control en tierra forma parte de otro TFG que se completará próximamente. Actualmente se controla desde la misma RPi, sin establecimiento de ningún control en tierra solo se puede acceder inalámbricamente al escritorio remoto de RPi.

Se pueden dibujar rutas y asignarles nombres, así como nombrar a cada waypoint de la ruta. La ventana de propiedades te indica la distancia total de la ruta, así como las propiedades visuales editables de la misma.

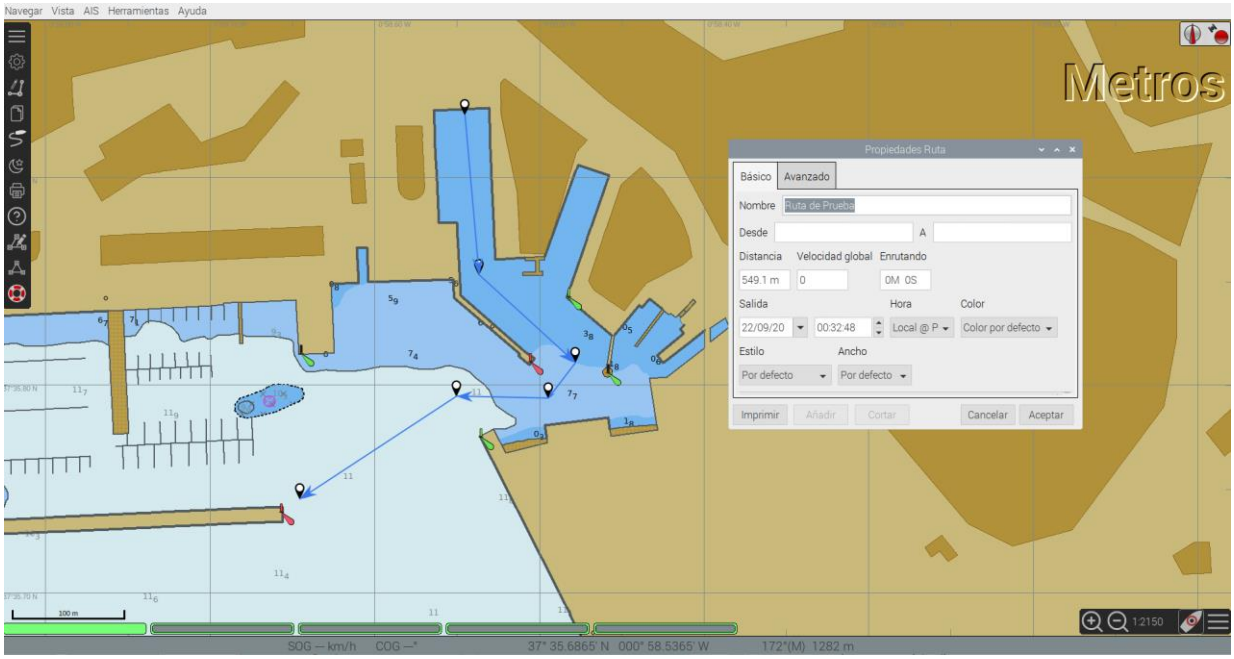


Figura 23 - Propiedades de Ruta

Desde el gestor de rutas te permite ver, cargar, activar e invertir rutas, algo muy útil para realizar viajes de retorno sin necesidad de crear una segunda ruta.

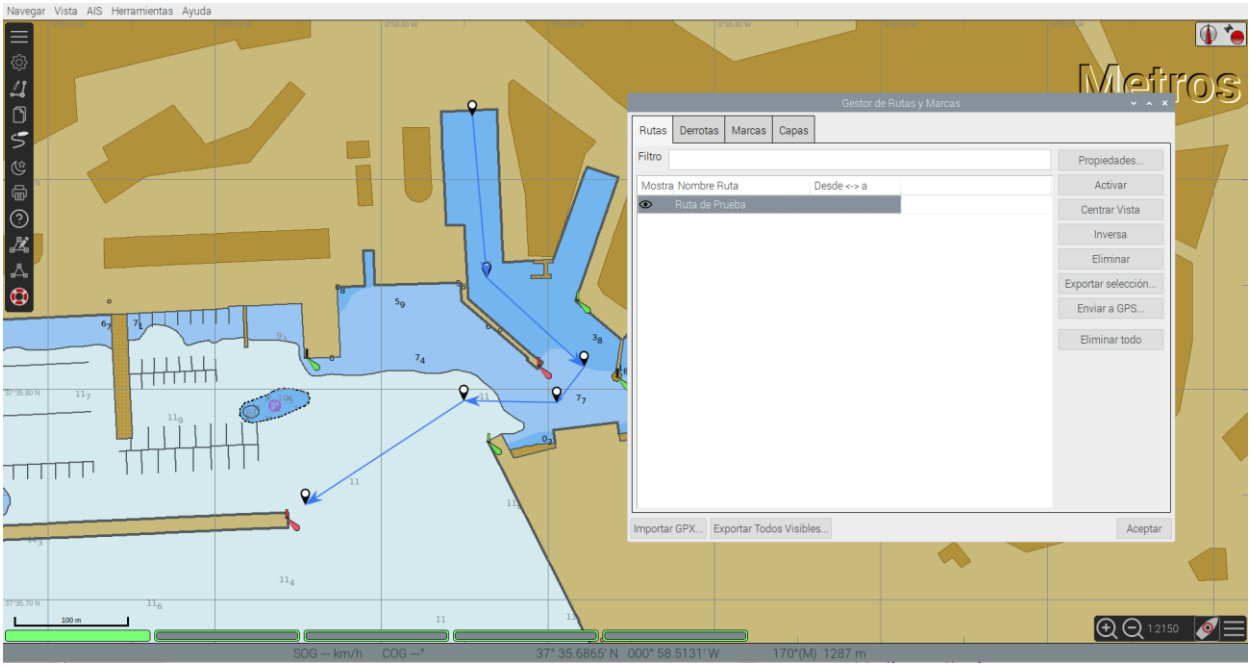


Figura 24 - Gestor de Rutas

Las rutas se pueden activar desde el gestor de rutas, haciendo doble clic sobre la misma, o botón derecho desde el menú conceptual que se abre. Una vez que la ruta está activa aparecerá una ayuda a la derecha que indica las órdenes y parámetros más relevantes en NMEA0183. En la siguiente Figura 25 se muestra el aspecto de una ruta activada.

Figura 25 - Ruta Activa

2.2. Protocolos y comunicación

En anteriores proyectos solo se ha trabajado se forma marginal con los protocolos y estándares en los que se transmite los datos de telemetría, es por eso que creo oportuno explicar en detalle los distintos formatos y protocolos que están involucrados en este proyecto. Hablemos primero de NMEA.

NMEA es el acrónimo para *National Marine Electronics Association* (Asociación Nacional de Electrónica Marina). Esta organización que crea estándares para dispositivos eléctricos, uno de sus más famosos es el NMEA0183.

NMEA0183 es un estándar cerrado y propietario grandemente extendido durante las últimas décadas, siendo el predominante en todos los equipos comerciales, este estándar se basa en la transmisión de sentencias de un orador a múltiples receptores a la vez basado en la norma técnica RS-422, aunque también se ha adaptado para emitirse por datagramas UDP para su transmisión por sistemas inalámbricos.

Al ser NMEA0183 un protocolo cerrado y propietario, todo lo que se conoce de él es gracias a la labor de ingeniería inversa llevada a cabo durante años.

Un ejemplo de una sentencia NMEA0183 sería esta, obtenida en nuestro sistema VNAS:

*$ECRMB,A,0.000,L,,001,3637.929,N,00057.554,W,3748.937,87.078,0.000,V,A*59\r*

Cada uno de los valores separados por comas representa un campo específico:
(1) ECRMB: EC es un añadido de OpenCPN para diferenciar las señales provenientes de los sensores de las que emite OpenCPN. Se puede modificar sin problemas ya que VNAS solo busca el nombre de la sentencia en la cadena de texto.

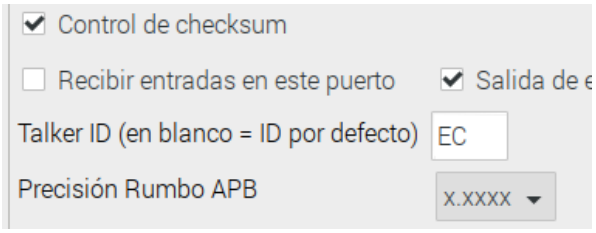


Figura 26 - Detalle de la conexión de salida

RMB es "Recommended minimum navigation information", y enviar información relacionada con el Waypoint. En el caso de una señal RMB el resto del formato de la sentencia es el siguiente:

(2) Estado de los datos A = OK, V = Void (Aviso)
(3) Error Cross-track (En Millas Náuticas, 9.99 máximo),
(4) Virar a la Izquierda (L) o a la Derecha (R) para corregir
(5) ID de Origen de Waypoint
(6) ID de Destino de Waypoint
(7) Latitud del Waypoint de Destino
(8) Longitud del Waypoint de Destino
(9) Distancia hasta el Destino, en Millas náuticas (999.9 máximo)
(10) Demora Absoluta al Destino
(11) Velocidad hacia el Destino, en Nudos
(12) Alarma de Llegada A = Llegada, V = Pendiente de llegada
(13) *XX checksum

De esta sentencia por ejemplo extraemos la latitud y longitud del waypoint actual, la demora, la distancia y la velocidad hacia el destino. También se puede realizar el checksum de la sentencia, pero debido a que la lectura de las sentencias es cada milisegundo y la ejecución de control es cada segundo y medio, es innecesario realizar cálculos adicionales.

Hablemos ahora de **NMEA2000** o **N2K**, este estándar es mucho más robusto y extensible en cuanto al número de dispositivos que pueden conectarse a la misma red. Ya que se ha actualizado estableciendo un bus CAN como medio de transmisión entre todos los dispositivos, migrando del antiguo RS232. En la actualidad todos los fabricantes producen dispositivos basados en N2K, si bien pueden usar conectores especiales distintos a los establecidos en el estándar, o usar protocolos rebautizados pero que son en realidad el mismo sistema N2K, como SeaTal, Raymarine y otros, es decir todos están unificados realmente bajo este estándar, aunque a primeras pueda parecer lo contrario.

Se emplea un backbone al que se conectan todos los dispositivos N2K (hasta un máximo de 50 dispositivos) y por el que los dispositivos envían y reciben las sentencias propias de este estándar. Como se ha explicado antes, es propietario y cerrado, por lo que lo que se conoce públicamente es debido a ingeniería inversa.

Figura 27 - Compás y GSS para N2K

Este compás magnético y GPS son los dos dispositivos usados en las pruebas de laboratorio conectados al backbone N2K. El Compas AIRMAR y el GPS Garmin. Ambos envían las sentencias en N2K. Hay proyectos de código abierto como CANBOAT han implementado herramientas de decodificación y codificación de sentencias N2K a través de ingeniería inversa. (12)

Y por último hablemos de **Signal K**, con la introducción en esta década de la electrónica de consumo y placas DIY (Do It Yourself), como Arduino y Raspberry Pi, las posibilidades para los entusiastas marinos se han ido ampliando para crear su propio estándar abierto. Y en esto se ha traducido el empeño de varios años por varios grupos de personas, una plataforma de código abierto que pretende ser el nuevo estándar para el intercambio de datos en el entorno marítimo. No solo dentro de un mismo barco, si no entre embarcaciones, puertos, etc, un estándar adaptado al IoT y al escenario tecnológico actual.

Sabiendo ya esto, vamos a empezar explicando desde el origen, los datos que se usan para el control de la embarcación.

Los sensores que están instalados por el barco están todos conectados como ya se sabe a un backbone Bus CAN N2K. Estos emiten mensajes en formato N2K en el bus CAN, y a través de la puerta de enlace Actisense NGT-1, se trasladan esos mensajes por conexión USB a nuestro servidor Signal K instalado en la RPi, donde transforma los mensajes recibidos en formato N2K a formato Signal K.

Al ser código abierto y colaborativo, existe un plugin que nos permite transformar la información del servidor Signal K, a las sentencias que deseemos de NMEA0183, y poder

emitirlas a través de protocolo TCP a OpenCPN, el cual recibe todas las sentencias necesarias, y añadiendo las suyas propias en cuanto al waypoint se trata, y emite todo esto haciendo de pasarela NMEA0183 por el protocolo que prefiramos, UDP o TCP, nosotros queremos UDP para nuestro auto piloto VNAS, el cual finalmente controlará por Serie los actuadores PWM Arduino de los motores.

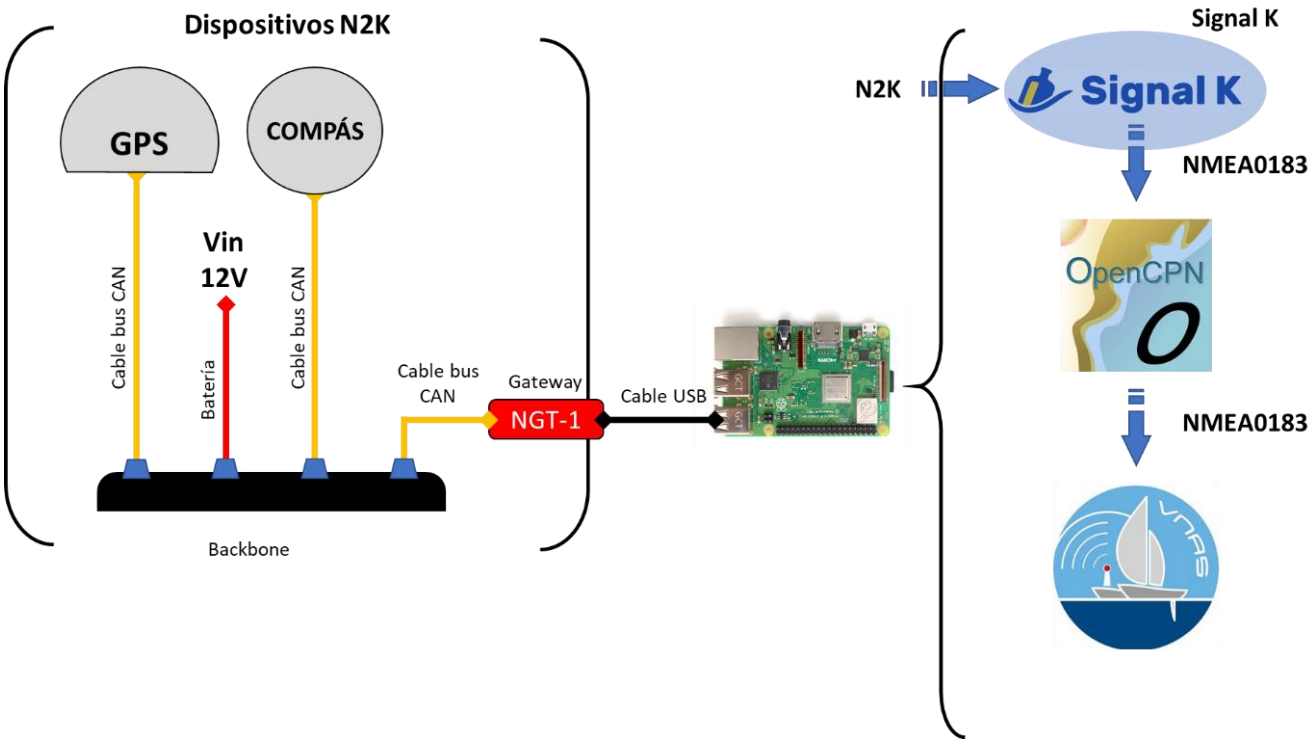


Figura 28 - Diagrama de flujo de datos y conexiones

Siendo el flujo de datos como se indica en la Figura 28. ¿Parece complicado? En realidad, no lo es tanto, es el resultado evidente al usar distintas tecnologías y estándares propietarios, mientras se intenta acercar todo ello a un escenario de código abierto y no depender del código propietario.

En desarrollos futuros es absolutamente necesario llegar a eliminar esta dependencia de NMEA y pasar a usar únicamente Signal K, tanto a nivel de software, como a nivel de hardware (Telemetría, sensores, etc.).

Para que el lector se haga una idea de lo rápido que va el desarrollo de estos sistemas, durante la redacción final de esta memoria me he visto obligado a realizar cambios importantes en la estructura después de actualizar desde la versión 5.0 de OpenCPN a la 5.2 en la que OpenCPN ya soporta entradas de Signal K.

Es un lujo poder contar con un chartplotter que con soporte para Signal K finalmente, esto libera totalmente del uso de NMEA0183 en todos los aspectos del piloto automático. Como entenderá el lector el concepto de este proyecto sigue siendo el mismo, y se encomendará a futuros desarrollos la eliminación de NMEA tanto de la telemetría, como del piloto automático.

En cuanto a comunicaciones, Python solo nos permite una conexión de cada tipo (TCP/UDP) para cada uso (Entrada/Salida), a no ser claro, que se ejecuten distintos programas separados, o se implemente la utilización programación multihilo con la librería threading. Habiendo mencionado ya TCP y UDP, es importante remarcar la diferencia entre ellas. TCP es un protocolo que requiere interacción entre el oyente y el emisor, tal y como se muestra en la imagen.

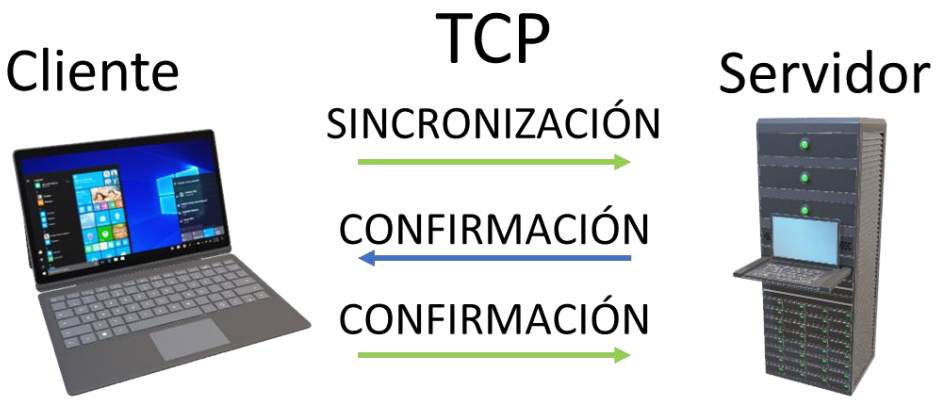


Figura 29 - Tipología TCP

Al contrario, con UDP, el cual simplemente manda los mensajes, sin importar si alguien escucha o los recibe. UDP es perfecto para nuestro auto piloto, ya que se realiza en el mismo soporte, no viaja fuera del entorno y la perdida de datos es nula, como se puede observar en la Figura 29.

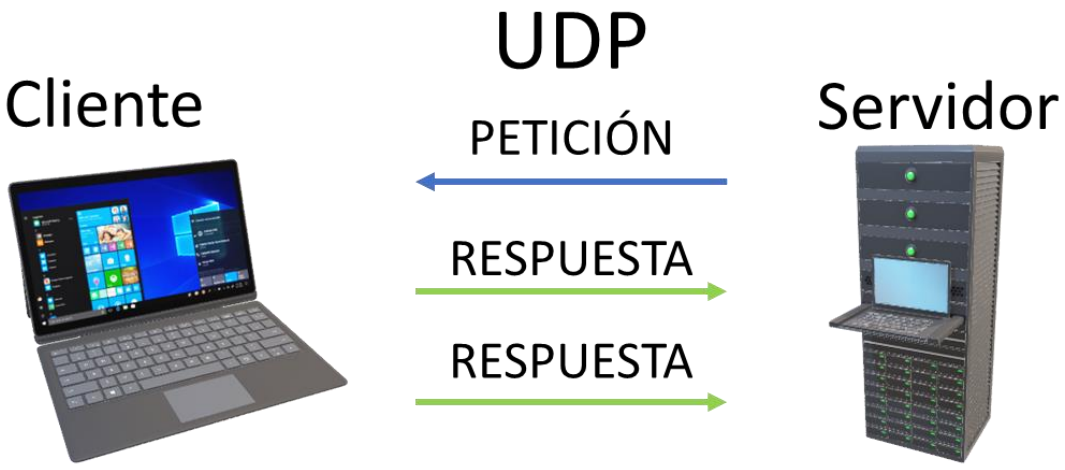


Figura 30 - Tipología UDP

Una vez que el auto piloto recibe las sentencias NMEA0183 por UDP, se identifican individualmente y almacena únicamente la última sentencia recibida de cada tipo. Se actualizan los datos en la GUI y el lazo de control interactúa con ellas para determinar la casuística. Ya determinadas las órdenes a mandar a los motores, se crea una palabra de control que se envía por Serie a los Arduinos Micro que controlan y generan el PWM de los motores. Y a su vez se recibe feedback del estado de los motores y las baterías, y si se ha activado el modo manual.

2.1. Servidor de señales

Con la conectividad del Bus CAN solucionado y el hardware básico establecido, nos queda obtener la telemetría a través del mismo.

Recurriendo al código libre encontramos a CANBoat, un conjunto de herramientas que incluye la codificación/decodificación de NMEA2000. Está basado en C y sirve para una gran variedad de interfaces, entre ellas nuestro NGT-1. Con esto, solucionamos totalmente la conectividad con los sensores, y somos capaces de recibir la telemetría.

Pero, siempre tiene que haber algún pero, NMEA2000 es propietario de acceso restringido a sus miembros previo pago, y por ello, todo desarrollo de código abierto que existe se ha desarrollado a través de ingeniería inversa. Todo nuestro desarrollo se vería empañado y obstaculizado por la opacidad de este protocolo, y queremos poder trabajar libremente con los datos con un estándar transparente y abierto. Es por eso que se ha recurrido a Signal K (13).

Signal K es un ambicioso formato de código abierto para uso marino. No se queda solamente en proporcionar un formato para la codificación de los datos (Signal K), sino que también proporciona el código y las herramientas para distribuir ese código en nuestros desarrollos a través servidores y clientes para diseñar nuestras propias redes de transmisión. Un esquema general se muestra en la Figura 31.

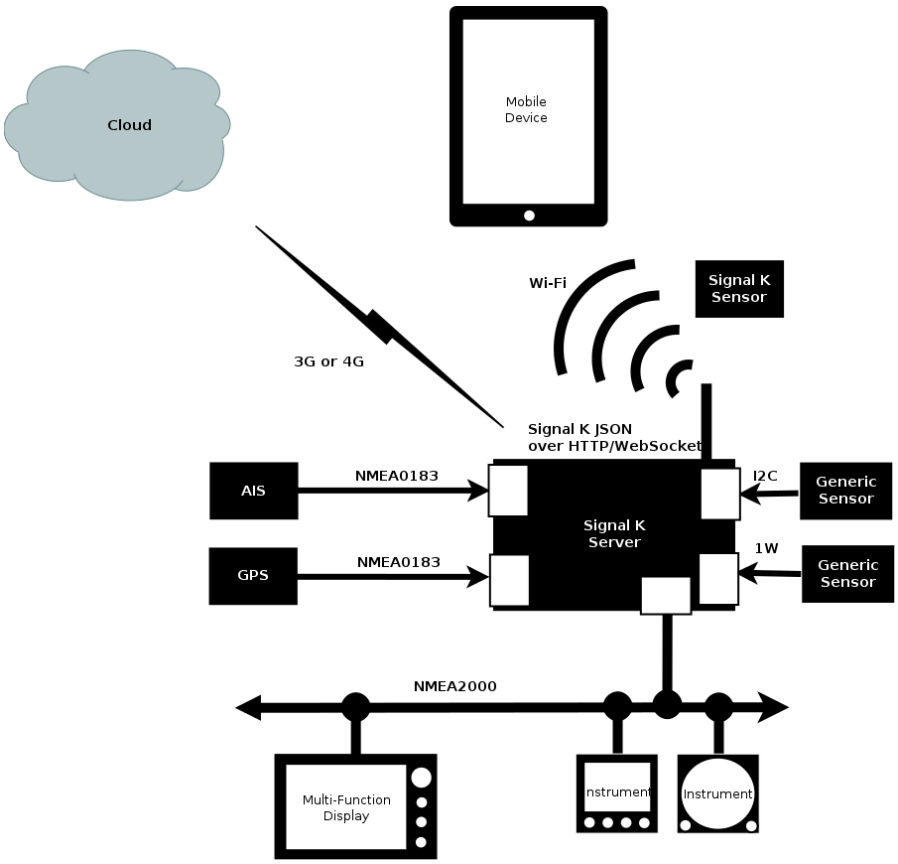


Figura 31 - Conectividad y funcionamiento de un servidor Signal K

Como se puede ver en la Figura 31, Signal K permite la interconexión de una multitud de señales distintas en formatos y protocolos distintos, y su retransmisión a través de la nube para su procesado o seguimiento.

Es más, CANBoat tiene un desarrollo en JavaScript para usar en conjunción con Signal K, habilitando al servidor Signal K de la adquisición de datos, y la capacidad de distribuir la telemetría a nuestro gusto (14). Con esto el servidor Signal K convierte todos los formatos que reciba (NMEA2000, NMEA0183, etc.) en su propio formato Signal K, y viceversa nos permite reconvertirlos a cualquier otro formato, por ejemplo, al NMEA0183 para poder enviarlo a nuestro chartplotter u otro dispositivo físico o digital que solo funcione con ese estándar.

Vamos a optar por el servidor Signal K para controlar y distribuir nuestra información en el VNAS. Para instalarlo referirse al apartado SO de este proyecto o su página oficial

Una vez en funcionamiento, si accedemos al servidor a través del navegador para configurarlo, podemos observar que detecta todas las entradas activas por donde se reciben señales, y la densidad de información de las mismas.

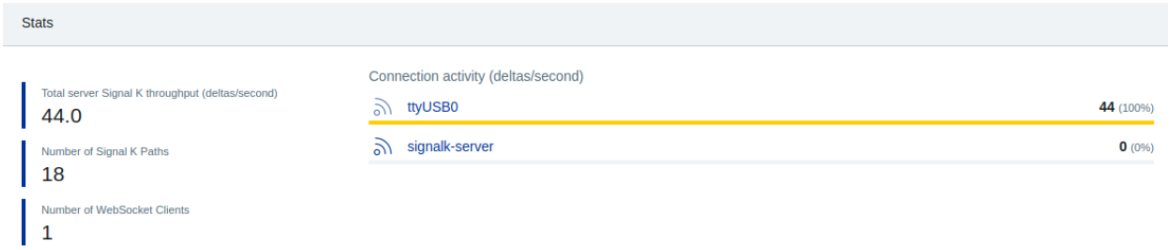


Figura 32 - Entrada de señales en Signal K

Si hacemos clic sobre la interfaz USB (ttyUSB0) que se muestra en el servidor, se accederá una pantalla de configuración donde aparecen las características de nuestra interfaz NGT-1.

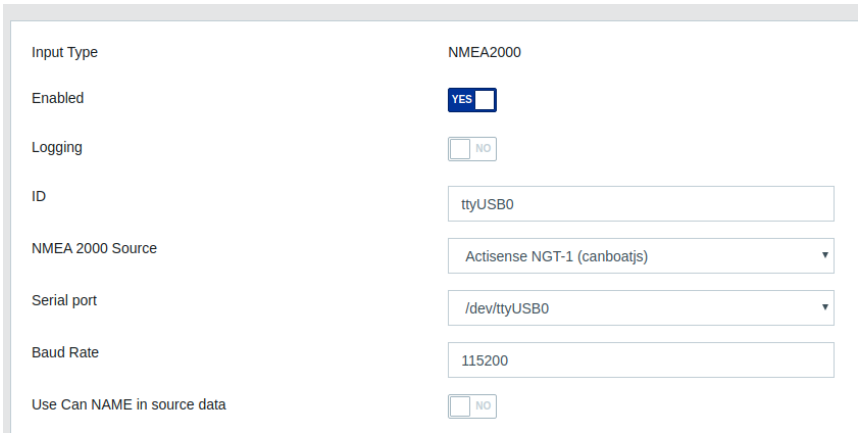


Figura 33 - Configuración de entrada de señales

Nuestro interés pasa ahora a convertir la información del servidor en formato NMEA0183 para que pueda usarla el chartplotter. Para ello recurrimos al plugin “Signal K to NMEA0183”, que emite las señales que deseemos y las emite por TCP en el puerto 10110 por defecto. Aunque se puede configurar para que lo haga por puerto serie. (15)

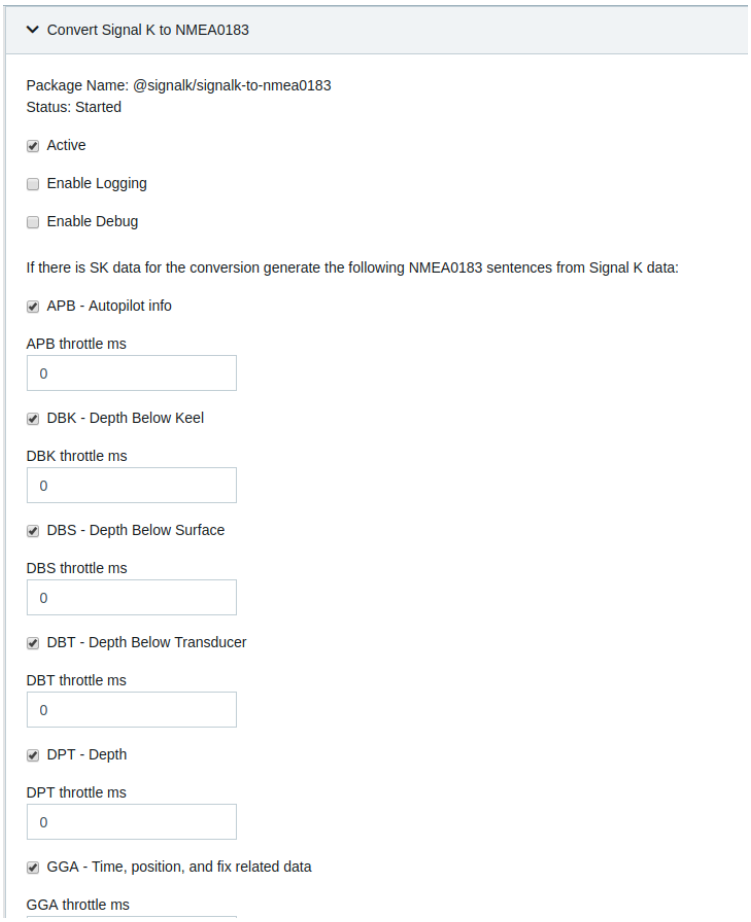


Figura 34 - Configuración del plugin "Signal K to NMEA0183"

2.2. SO

Y todo esto nos lleva al siguiente punto, el SO.

Raspberry Pi Foundation ofrece una distribución GNU/Linux basado en Debian adaptada a la arquitectura ARM, llamada Raspbian (aunque recientemente ha pasado a denominarse “Raspbian Pi OS”). Raspbian dispone de distintas versiones que se han ido liberando a lo largo de los años y de la venta de nuevas placas RPi, nuestra RPi 3B+ soporta a partir de la versión 9 “Stretch”, con Kernel 4.14 hasta la actual versión más reciente 10 “Buster”.

La elección parece sencilla, y lo es, a lo largo de los años RPi ha acumulado una gran comunidad de aficionados y desarrolladores a sus espaldas, creando una ingente cantidad de repositorios y aplicaciones de gran utilidad. El soporte que recibe es perfecto

para iniciar un desarrollo o proyecto desde cero. Es por todo esto que se elige la última versión de Raspbian como SO donde ejecutar nuestro auto piloto, o sistema avanzado de control.

Pero ya que exaltamos en este proyecto el valor del código abierto y la colaboración, vamos a usar una versión modificada de Raspbian; Openplotter (16). Hemos creado un logo en conjunción que es el que se muestra en nuestro escritorio.

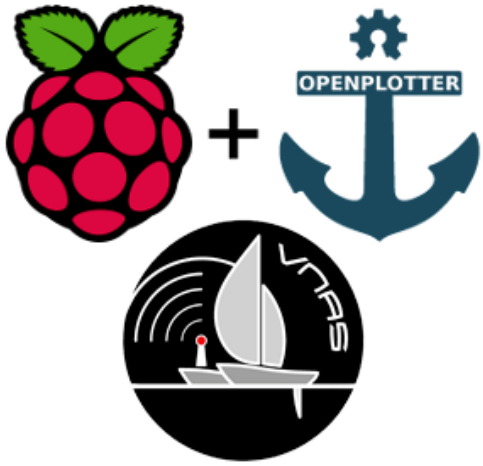


Figura 35 - Unión de Raspbian, Openplotter y VNAS

Openplotter es la aglutinación de todo el software libre anteriormente mencionado, basado en Raspbian y distribuido por Sailoong para desarrollos marinos. Incluye una cantidad ingente de desarrollos de terceros, preinstalados y configurados para funcionar "on the go" sin ningún problema de compatibilidad en sistemas RPi y ordenadores que ejecuten una variante de Debian. Dispone de hardware específico hecho por aficionados, autopiloto para embarcaciones de vela, visualización de mapas meteorológicos, etc.

Se puede optar por realizar la instalación por separado de las distintas instancias del software, pero en este proyecto usaremos Openplotter como SO base.

Para instalar Openplotter se proporciona una imagen del sistema entre los archivos de este proyecto. Pero se puede recurrir a una instalación limpia sin ningún problema. Para ello es recomendable descargarse la última versión de Openplotter desde su sitio oficial (16). Y usar Raspberry Pi Imager para escribir la imagen en el soporte físico que vayamos usar en la RPi. En nuestro caso, el disco SSD.

Con Openplotter instalado y configurado tenemos que realizar ahora el piloto automático. Aunque Openplotter incluye un piloto automático, este está diseñado para embarcaciones con timón y vela, lo que dista mucho de nuestro desarrollo eléctrico a dos motores. Y precisamente es uno de los puntos de desarrollo de este proyecto.

3. VNAS Autopiloto – Análisis de Código Fuente

3.1. Python y GUI

Para la programación y creación del auto piloto lo primero fue analizar la plataforma de desarrollo y las herramientas disponibles.

Vamos a realizar desde cero nuestro propio piloto automático para embarcaciones con dos motores eléctricos (o VNAS para simplificarlo). Y antes tenemos que elegir un lenguaje de programación, este estudiante tiene conocimientos de C y Java. Pero vamos a realizar un poco de investigación previa:

El ecosistema de lenguajes de programación ha permanecido invariante durante muchos años, pero en la última década, la irrupción de nuevos paradigmas y lenguajes de programación cada vez de nivel más alto, ha cambiado la balanza. Según datos estadísticos de Stackoverflow (17) y el Índice Pypl (18), Python es el lenguaje más popular y usado actualmente. Ciertamente es el más usado en Raspberry Pi, ya que incluye por defecto los entornos para Python2 y 3, y se propicia a su uso. Python en la actualidad ha superado ya en popularidad a Java, tal como se demuestra en la siguiente Figura 36.

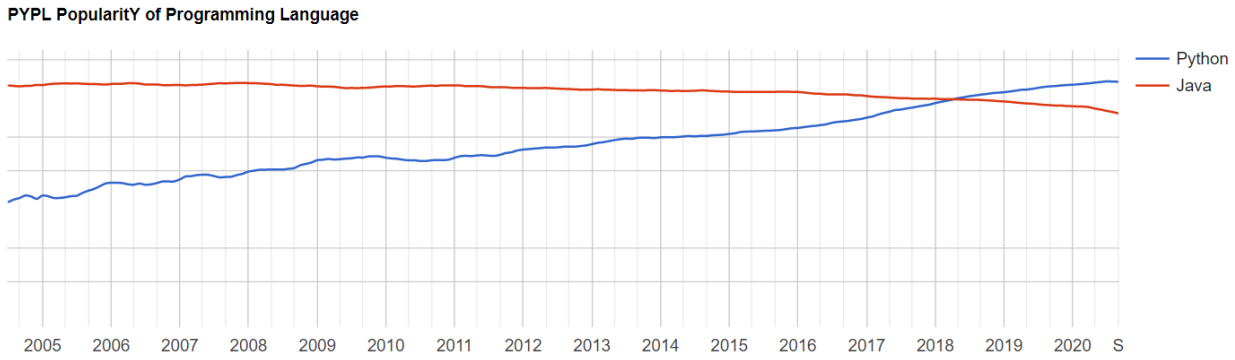


Figura 36 - Evolución de la popularidad de Python vs java (Pypl) (18)

Dentro de Python coexistían dos versiones, la 2 y la posteriormente introducida versión 3. Hay ciertas librerías y repositorios que puede que no hayan sido adaptados todavía a Python3, pero es la norma general y de futuro la migración hacia esta versión, ya que se ha descontinuado oficialmente este año Python 2, por lo que se desarrolla naturalmente nuestro auto piloto bajo Python3.

Python es un lenguaje interpretado, al contrario que Java o C, no se codifican las líneas de código a código máquina, si no que el interpretador realiza esta acción para poder ejecutar el mismo código en una gran variedad de sistemas, haciéndolo un lenguaje multiplataforma. Una de sus grandes bazas es la gestión de excepciones y la peculiaridad de una forma de escritura con indentaciones obligatorias.

Viendo que RPi parece el entorno ideal para programarlo en Python, y la versatilidad que ofrece, se va proceder a programar el piloto automático VNAS con Python.

Figura 37 - Logo de Python

Nacido en los 80, de la mano de Guido van Rossum, Python (logo mostrado en la Figura 37) es un lenguaje fácil de aprender, y bastante potente ya que posee estructuras de alto nivel y posibilidad de programación orientada a objetos. Hereda del lenguaje ABC muchas ideas y conceptos. Los tipos de datos son fijos, el diseño de instrucciones es sencillo y potente a la vez, por ejemplo, el ciclo “for” con el operador “in”. Y hereda o copia instrucciones de C con el mismo significado, pero a diferencia de C, no usa los paréntesis para agrupar código, si no que se usan las indentaciones.

Es necesario destacar a lo que muchos usuarios de Python se refieren como su filosofía o principios que fueron enunciados por Tim Peters en El Zen de Python en 2004. (19)

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Lo práctico gana a lo puro.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una —y preferiblemente solo una— manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ya mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres (namespaces) son una gran idea ¡Hagamos más de esas cosas!

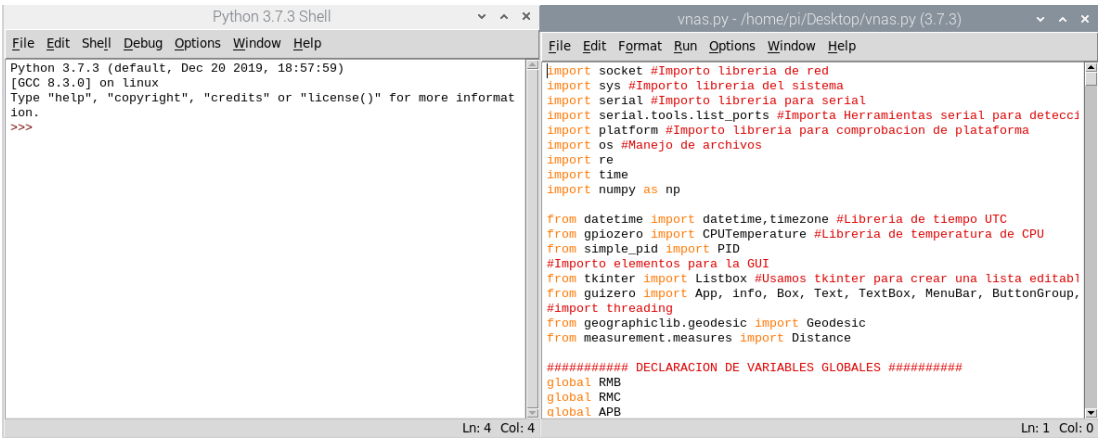


Figura 38 - IDLE Entorno de programación

Para programar vamos a recurrir al entorno de programación IDLE (Figura 38) que se incluye en RPi, y todo se va a realizar en la propia RPi, sin necesidad de usar otro equipo para programar.

El aspecto general que se desea crear es una interfaz de usuario para interactuar y lanzar el lazo de control de los motores teniendo como entrada de datos la telemetría del barco y la ruta a seguir que le marquemos en OpenCPN. Entrando en el apartado de librerías de interfaces para Python, existe principalmente Tkinter, la más extendida. A su vez existe una versión más sencilla y ligera para creación rápida de interfaces llamada GUIzero (20), basada en Tkinter, y que permite usar Tkinter en sus propias funciones. Lo que permite desarrollar con facilidad GUIs, pudiendo ampliarse más allá de las funciones estándar que incluye GUIzero si se da la necesidad.
En este proyecto vamos a usar principalmente GUIzero, mientras que se ha usado Tkinter para interactuar con el color de los elementos de una lista desplegable, así como para mantener siempre encima la ventana de ajustes, opciones que no están incluidas en las funciones estándar de GUIzero, pero que se pueden modificar sin problema gracias a la posibilidad de usarse en conjunción con Tkinter.

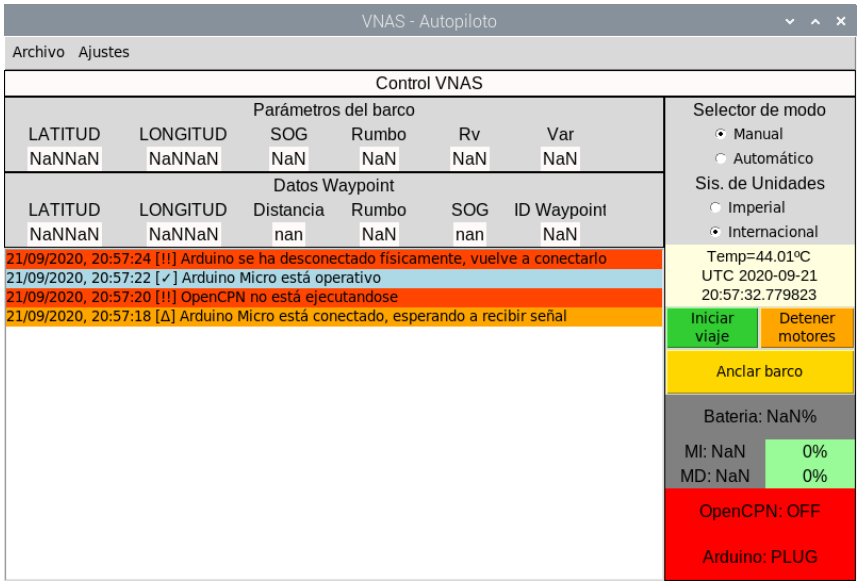


Figura 39 - Interfaz de VNAS

Al desarrollar con GUIzero existe una limitación, y es que la interfaz funciona como un bucle principal, sin opción de ejecución de código en paralelo, a excepción de las líneas previas a la creación de la interfaz, y de las llamadas a funciones temporizadas e invocadas por el usuario al interactuar con la interfaz.

Por ello, todo el código del auto piloto se ha separado en funciones, y de ellas 4 funciones principales se ejecutan llaman indefinidamente en un bucle de tiempo:

- Adquisición de valores UDP
- Actualización de valores de la interfaz
- Comunicación serial
- Lazo de control de los motores

El resto de funciones como el Logbook, ajuste de parámetros y demás, se llaman a través de botones en la interfaz.

La interfaz se ha diseñado para funcionar a pantalla completa a una resolución de 800x480, para adaptarse a la pantalla táctil y funcionar in situ, plug & play, con dicha resolución, y además reducir la carga de transferencia si se utiliza escritorio remoto.

En la interfaz se muestran los datos necesarios y de interés, como la posición del barco, SOG, orientación, posición del waypoint y su orientación relativa, así como la carga de las baterías, el consumo de los motores y el estado de las conexiones con OpenCPN y Arduino.

Además, permite el uso de pop-ups y múltiples ventanas, como se puede observar en esta Figura 40.

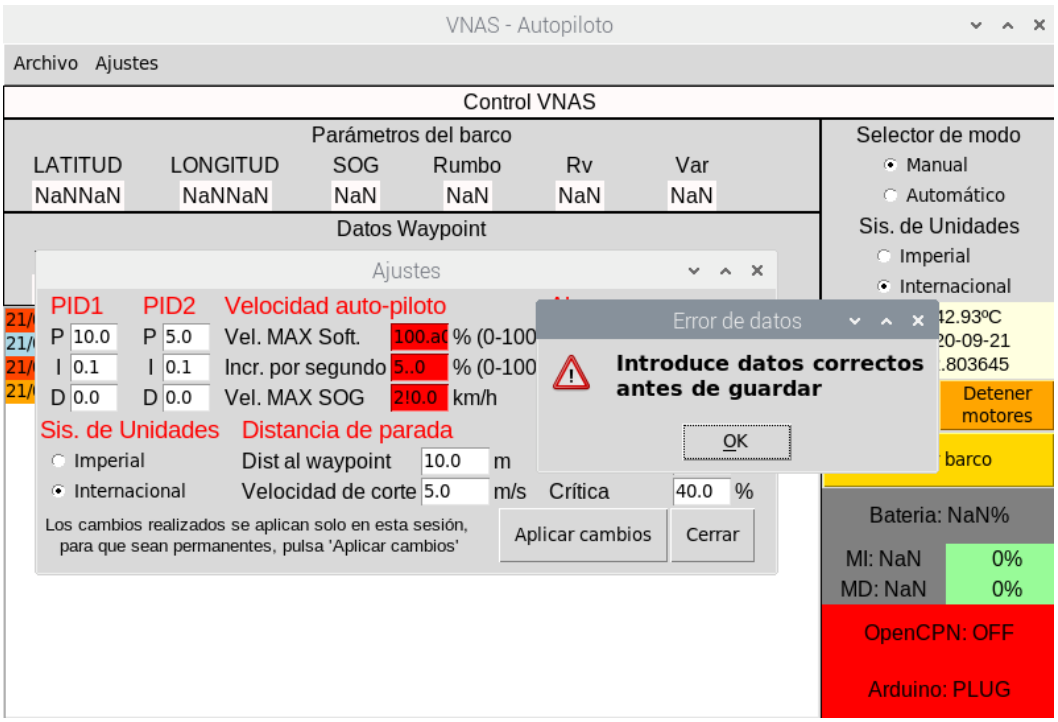


Figura 40 - VNAS con pop-ups y ventanas secundarias

En Python, como en la mayoría de lenguajes, se procede a importar primero las librerías que contienen las funciones específicas que se van a usar:

```
from tkinter import Listbox #Usamos tkinter para crear una lista editable
from guizero import App, info, Box, Text, TextBox, MenuBar, ButtonGroup,
Picture, Slider, Combo, CheckBox, PushButton, ListBox, Window
```

Nosotros importaremos de tkinter solamente el componente Listbox, que se explicará en el Logbook. Y de GUIzero seleccionamos los componentes que vamos a usar en nuestra interfaz. Ya se ha mencionado la peculiaridad de que GUIzero se ejecuta como un bucle infinito para mostrar la interfaz, y que todo código que se coloque después de la inicialización de la interfaz no se ejecutará, ya que ha entrado en el bucle de la interfaz. Cualquier llamada a un código se debe realizar a través de las llamadas temporizadas o las llamadas vinculadas a componentes de la interfaz con las que interacciona el usuario. Al final del código se empieza a configurar la interfaz.

```
################################################################################
########################### SETUP DE LA GUI  ###################################
################################################################################

#Creo las dimensiones de la GUI
app = App(width=800,height=480,title="VNAS - Autopiloto")
app.full_screen=True
```

La app se declara con las dimensiones que deseamos, y se le asigna un título a la ventana, también se indica que se desea que se ejecute en pantalla completa.

A continuación, se procede a “scriptar” la interfaz. Cuando se añade cualquier elemento hay que referenciarlo a una instancia, en este caso la app u otra ventana o elemento que exista, pudiendo anidarse elementos uno dentro de otro. Por ejemplo, una caja con texto en el interior, o una caja con varios botones y texto en el interior del botón. Es recomendable hacerse un diseño en papel para poder llevar con mayor facilidad la tarea de diseñarlo todo.

```
# Ventana de ajustes
ajustes = Window(app, height=215, title="Ajustes", width=560, visible=False)
a_grid_box = Box(ajustes, layout="grid", width="fill", height="fill",
align="top", border=False)
####PID1
PID1_box = Box(a_grid_box, grid=[0,0],layout="grid", width="fill",
height="fill", align="top", border=False)
Text(PID1_box, grid=[0,0,2,1], text="PID1",size=14, align="left",
color="red", font="Arial")
Text(PID1_box, grid=[0,1], text="P")
input_P = TextBox(PID1_box,grid=[1,1], width=3)
Text(PID1_box, grid=[0,2], text="I")
input_I = TextBox(PID1_box,grid=[1,2], width=3)
Text(PID1_box, grid=[0,3], text="D")
input_D = TextBox(PID1_box,grid=[1,3], width=3)
####PID2
PID2_box = Box(a_grid_box, grid=[1,0],layout="grid", width="fill",
height="fill", align="top", border=False)
```

```
Text(PID2_box, grid=[0,0,2,1], text="PID2",size=14, align="left",
color="red", font="Arial")
Text(PID2_box, grid=[0,1], text="P")
input_P2 = TextBox(PID2_box,grid=[1,1], width=3)
Text(PID2_box, grid=[0,2], text="I")
input_I2 = TextBox(PID2_box,grid=[1,2], width=3)
Text(PID2_box, grid=[0,3], text="D")
input_D2 = TextBox(PID2_box,grid=[1,3], width=3)
####Velocidad
speed_box = Box(a_grid_box, grid=[2,0],layout="grid", width="fill",
height="fill", align="top", border=False)
Text(speed_box, grid=[0,0], text="Velocidad piloto-auto",size=14,
align="left", color="red", font="Arial")
speed_values= Box(speed_box, grid=[0,1],layout="grid", width="fill",
align="left", border=False)
Text(speed_values, grid=[0,1], text="Vel. MAX Software", align="left")
speed_max = TextBox(speed_values,grid=[1,1], width=3)
Text(speed_values, grid=[2,1], text="% (0-100)", align="left")
Text(speed_values, grid=[0,2], text="Incr. por segundo", align="left")
speed_incr = TextBox(speed_values,grid=[1,2], width=3)
Text(speed_values, grid=[2,2], text="% (0-100)", align="left")
Text(speed_values, grid=[0,3], text="Vel. MAX SOG", align="left")
speed_sog= TextBox(speed_values,grid=[1,3], width=3)
Text(speed_values, grid=[2,3], text="km/h", align="left")
####Alarmas
alarm_box = Box(a_grid_box, grid=[3,0,1,2],layout="grid", width="fill",
height="fill", align="top", border=False)
Text(alarm_box, grid=[0,0], text="Alarmas",size=14, align="left",
color="red", font="Arial")
alarm_values= Box(alarm_box, grid=[0,1],layout="grid", width="fill",
align="left", border=False)
Text(alarm_values, grid=[0,1], text="MotorI MAX", align="left")
motorI_max = TextBox(alarm_values,grid=[1,1], width=3)
Text(alarm_values, grid=[2,1], text="A", align="left")
Text(alarm_values, grid=[0,2], text="MotorD MAX", align="left")
motorD_max = TextBox(alarm_values,grid=[1,2], width=3)
Text(alarm_values, grid=[2,2], text="A", align="left")
Text(alarm_values, grid=[0,3], text="Alarma Bateria", align="left",size=14,
color="red", font="Arial")
Text(alarm_values, grid=[0,4], text="Llena", align="left")
llena = TextBox(alarm_values,grid=[1,4], width=3)
Text(alarm_values, grid=[2,4], text="%", align="left")
Text(alarm_values, grid=[0,5], text="Baja", align="left")
baja = TextBox(alarm_values,grid=[1,5], width=3)
Text(alarm_values, grid=[2,5], text="%", align="left")
Text(alarm_values, grid=[0,6], text="Crítica", align="left")
critica = TextBox(alarm_values,grid=[1,6], width=3)
Text(alarm_values, grid=[2,6], text="%", align="left")
```

También incluye el parámetro grid, el cual nos permite cambiar el interior de un elemento en una rejilla en la que ir colocando elementos e ir forzándolos en esa posición. El primer elemento es la columna, y el segundo, la fila. De la misma forma, se pueden anidar rejillas dentro de rejillas, como se ha hecho en la ventana de ajustes, pero puede resultar un pequeño desafío si no se planifica correctamente.

```
#Sistema de unidades
units_box = Box(a_grid_box, grid=[0,1,2,1],layout="grid", width="fill",
height="fill", align="left", border=False)
Text(units_box, grid=[0,0], text="Sis. de Unidades",size=14,align="left",
color="red", font="Arial")
units_values= Box(units_box, grid=[0,1],layout="grid", width="fill",
align="left", border=False)
units_settings = ButtonGroup(units_values,grid=[0,0],options=[
                              ["Imperial", "1"],
                              ["Internacional", "2"]
                            ])

#Distancia de llegada
arrival_box = Box(a_grid_box, grid=[2,1,1,1],layout="grid", width="fill",
height="fill", align="top", border=False)
Text(arrival_box, grid=[0,0], text="Distancia de
parada",size=14,align="left", color="red", font="Arial")
arrival_values= Box(arrival_box, grid=[0,1],layout="grid", width="fill",
align="left", border=False)
Text(arrival_values, grid=[0,1], text="Dist al waypoint", align="left")
arrival_dist = TextBox(arrival_values,grid=[1,1], width=6)
Text(arrival_values, grid=[2,1], text="m", align="left")
Text(arrival_values, grid=[0,2], text="Velocidad de parada", align="left")
stop_speed = TextBox(arrival_values,grid=[1,2], width=6)
Text(arrival_values, grid=[2,2], text="m/s", align="left")

#Aceptar o Cancelar ajustes
alarm_box2 = Box(a_grid_box, grid=[0,2,4,1],layout="grid", width="fill",
height="fill", align="right", border=False)
button_box2 = Box(alarm_box2, grid=[0,0],height="fill", width="fill",
border=False)
Text(button_box2, text="Los cambios realizados se aplican solo en esta
sesión,     \n para que sean permanentes, pulsa 'Aplicar cambios'     ",
align="left",size=10)
ok_settings=PushButton(button_box2, text="Aplicar cambios", align="left",
command=apply_settings)
cancel_setting=PushButton(button_box2, text="Cerrar", align="left",
command=close_settings)

#Creo menu
menubar = MenuBar(app,
                  toplevel=["Archivo", "Ajustes"],
                  options=[
                      [ ["Forzar conexión", file_function], ["Cerrar",
exit_function] ],
                      [ ["Editar PID", settings_function], ["Alternar
pantalla completa", fullscreen], ["Información", informa] ]
                  ])
```

La declaración para la barra de menú `MenuBar` tiene dos opciones, “top level” que sen las pestañas del menú, y luego “options”, que es la agrupación de todas las opciones de cada una de las pestañas.

Pasa igual con `ButtonGroup` del sistema de unidades, se incluye la opción en texto, y el valor asignado a esa opción, ambos totalmente personalizables, y que suponen una ventaja a la hora de guardar datos en forma de números y no en texto.

```
#Titulo
title_box = Box(app, width="fill", align="top", border=True)
Text(title_box, text="Control VNAS", bg="snow", width="fill")
###########Caja derecha#########
options_box = Box(app, height="fill",width=130, align="right", border=True)
#Selector modo automatico y manual
Text(options_box, text="Selector de modo")
auto_manu = ButtonGroup(options_box, options=["Manual", "Automático"],
selected="Manual",command=modo_selec)
#Sistema de unidades
Text(options_box, text="Sis. de Unidades")
is_imperial = ButtonGroup(options_box, options=[
                                ["Imperial", "1"],
                                ["Internacional", "2"]
                            ], selected="1")
#Caja con temperatura y fecha del sistema
cpu_temp_fondo = Box(options_box, width="fill", height=58,border=False)
cpu_temp_fondo.bg="light yellow"
cpu_temp= Text(cpu_temp_fondo, text="", width=14,height=58,size=11)

#Parada total
button_box = Box(options_box, height=40, width=130, border=False)
paradas = PushButton(button_box, text="Detener\nmotores",
width=5,height="fill",align="right", command=detener_viaje)
paradas.bg="orange"
#Inicio
iniciar = PushButton(button_box, text="Iniciar\nviaje",
width=6,height="fill",align="right", command=iniciar_viaje)
iniciar.bg="lime green"

anclar = PushButton(options_box, text="Anclar barco", width="fill",height=1,
command=iniciar_viaje)
anclar.bg="gold"
#Bateria
bateria_estado = Box(options_box, width="fill", height="fill")
bateria_estado.bg="pale green"
bat_estado= Text(bateria_estado, text="Bateria", width=14,height="fill",
align="right")
#Motores
motor_state= Box(options_box, layout="grid", width="fill", border=False)

motorI_estado = Box(motor_state,grid=[0,0], width="fill", height="fill")
motorI_estado.bg="pale green"
miestado= Text(motorI_estado, text="MI: 0A", width=7,height="fill",
align="left")
motorD_estado = Box(motor_state, grid=[0,1], width="fill", height="fill")
motorD_estado.bg="pale green"
mdestado= Text(motorD_estado, text="MD: 0A", width=7,height="fill",
align="left")

motorI_pw = Box(motor_state,grid=[1,0], width="fill", height="fill")
motorI_pw.bg="pale green"
mipw= Text(motorI_pw, text="0%", width=7,height="fill", align="right")
motorD_pw = Box(motor_state, grid=[1,1], width="fill", height="fill")
motorD_pw.bg="pale green"
mdpw= Text(motorD_pw, text="0%", width=7,height="fill", align="right")

#OpenCPN
CPN_estado = Box(options_box, width="fill", height="fill")
CPN_estado.bg="pale green"
```

```
OCPNestado= Text(CPN_estado, text="OpenCPN",
width=14,height="fill",align="right")


#Arduino
arduino_estado = Box(options_box, width="fill", height="fill")
arduino_estado.bg="pale green"
ardu_estado= Text(arduino_estado, text="Arduino PLUG",
width=14,height="fill",align="right")

fondo_estado = Box(options_box, width="fill", height="fill",border=True)
fondo_estado.bg="pale green"
estado= Text(fondo_estado, text=" Esperando \nactualmente",
width=14,height="fill")


#Parametros del barco
content_box = Box(app, align="top", width="fill", border=True)
Text(content_box, text="Parámetros del barco")
#Grid de parametros del barco
form_box = Box(content_box, layout="grid", width="fill", align="left",
border=False)
Text(form_box, grid=[0,0], text="LATITUD", width=12)
Text(form_box, grid=[1,0], text="LONGITUD", width=12)
Text(form_box, grid=[2,0], text="SOG", width=9)
Text(form_box, grid=[3,0], text="Rumbo", width=9)
Text(form_box, grid=[4,0], text="Rv", width=9)
Text(form_box, grid=[5,0], text="Var", width=9)
boat_lat=Text(form_box, grid=[0,1], text="xxxxxx",bg="snow", width="fill")
boat_long=Text(form_box, grid=[1,1], text="xxxxxx",bg="snow", width="fill")
boat_sog=Text(form_box, grid=[2,1], text="xxxxxx",bg="snow", width="fill")
boat_rm=Text(form_box, grid=[3,1], text="xxxxxx",bg="snow", width="fill")
boat_rv=Text(form_box, grid=[4,1], text="xxxxxx",bg="snow", width="fill")
boat_UTC=Text(form_box, grid=[5,1], text="xxxxxx",bg="snow", width="fill")

#Parametros del waypoint
content_box2 = Box(app, width="fill", border=True)
Text(content_box2, text="Datos Waypoint")
#Grid de parametros del waypoint
form_box2 = Box(content_box2, layout="grid", width="fill", align="left",
border=False)
Text(form_box2, grid=[0,0], text="LATITUD", width=12)
Text(form_box2, grid=[1,0], text="LONGITUD", width=12)
Text(form_box2, grid=[2,0], text="Distancia", width=9)
Text(form_box2, grid=[3,0], text="Rumbo", width=9)
Text(form_box2, grid=[4,0], text="SOG", width=9)
Text(form_box2, grid=[5,0], text="ID Waypoint", width=9)
way_lat = Text(form_box2, grid=[0,1], text="xxxxxx",bg="snow", width="fill")
way_long=Text(form_box2, grid=[1,1], text="xxxxxx",bg="snow", width="fill")
way_distance=Text(form_box2, grid=[2,1], text="xxxxxx",bg="snow",
width="fill")
way_rv=Text(form_box2, grid=[3,1], text="xxxxxx",bg="snow", width="fill")
way_sog=Text(form_box2, grid=[4,1], text="xxxxxx",bg="snow", width="fill")
way_destino=Text(form_box2, grid=[5,1], text="xxxxxx",bg="snow",
width="fill")

#Log de estado
log_estado_list = Listbox()
app.add_tk_widget(log_estado_list)
log_estado_list.pack(side="left", fill="both", expand=1)
```

Y finalmente se declaran las funciones para cuando se cierra la aplicación (Se pide confirmación), junto con las 4 funciones temporizadas; refresh, values, control_codigo y comunicacion_serial. Terminando con app.display ()

```
############################## FIN SETUP GUI ##################################
###############################################################################
###############################################################################
app.when_closed = exit_function
ajustes.when_closed = close_settings
#Llamada de actualizacion de valores mediante UDP
app.repeat(1, refresh) #Comunicacion UDP
app.repeat(1000, values) #Actualización valores GUI
app.repeat(1000, control_codigo) #Actualización autopiloto
app.repeat(1500, comunicacion_serial) #Actualizacion Serial
#Eventos de los botones
#iniciar.when_clicked = iniciar_viaje
#paradas.when_clicked = detener_viaje

#thread = threading.Thread(target=comunicacion_serial)
#thread.start()
app.display()
```

Es aquí, cuando comienza el bucle sin fin de la GUI. Cualquier línea colocada por debajo de esta llamada, no se ejecutará normalmente.

Vamos a repasar ahora las funciones que intervienen y son llamadas desde la GUI.

```
############################## FUNCIONES DE GUI ###############################
#          Funciones relacionadas con la GUI y sus menus y ventanas           #
###############################################################################

############### Funciones de la Barra de menus ###############
def exit_function(): #Funcion de salida del programa
    if app.yesno("Cerrar", "¿Desea cerrar el piloto?"):
        if app.yesno("Cerrar", "Permiteme que insista, ¿seguro que quieres
cerrar?"):
            app.destroy()
            sys.exit(42) #Salida normal, código 42
```

La exit_function() es llamada siempre que la app se cierra por cualquier medio, ya sea por la x conceptual del SO, acceso de teclado para cerrar la aplicación, o si se llama a “salir” desde la barra de menú. Esta función realiza una doble confirmación para salir del VNAS.

```
def informa(): #Funcion "Mas sobre el autor y programa"
    info("Info", "Versión 1.0\nDesarrollado por JCUC para el proyecto
VNAS\nFunciona en conjunto con OpenCPN y SignalK\n")
```

Esta pequeña función es simplemente indica la versión del código y una pequeña pieza de información sobre el mismo.

```
############### Funciones de los elementos de la ventana principal
###############
def iniciar_viaje(): #Inicio de ruta
    global run
    global auto
    global Primera_vez
    if (boat_lat.value=="NaN") or (boat_long.value=="NaN") or
(boat_sog.value=="NaN") or (boat_rm.value=="NaN") or (way_lat.value=="NaN")
or (way_long.value=="NaN"):
        info("Info", "OpenCPN no proporciona todos los datos, comprueba que
recibe señal o que hay un waypoint activo")
    elif run == True:
        info("Info", "Ya se está ejecutando")
    elif auto==False:
        info("Info", "Barco en modo manual")
    else:
        iniciar_viaje_b = app.yesno("Inicio", "¿Desea inicia la ruta?")
        if iniciar_viaje_b == True:
            if sanitize()==False:
                ajustes.tk.attributes('-topmost', True) #Me rompí la cabeza
con esto, gracias gracias, no hacen falta rosas
                ajustes.warn("Error de datos", "Introduce datos correctos
antes de iniciar el viaje")

            else:
                ajustes.hide()
                run=True
                Primera_vez=True
                control_codigo()
        else:
            run=False
###################### DETENER VIAJE ###################
def detener_viaje():
    global run
    global arduino
    global control

    if auto==False: #Si está en modo manual, no hay nada que hacer
        info("Info", "Barco en modo manual, no se puede interactuar con los
motores")
    else: #Si por el contrario está en modo automático
        if run==True: #Si está activada la navegación
            run=False #Desactivamos el flag de navegación para que la funcion
de control deje de funcionar

            try:
                control="0511.0511\n"
                control_codigo()
                log_estado("[✓] Parada confirmada", "light green")
            except:
                log_estado("[!!] No se ha podido confirmar la parada",
"orange red")


        else: #Por seguridad
            if detect_arduino() !=False:
                control="0511.0511\n"
                control_codigo()
                log_estado("[✓] Parada reconfirmada", "light green")
            else:
                app.error("Oh!", "Arduino Micro no disponible")
```

Estas dos funciones son las que están asignadas a los botones de inicio del viaje, y detención del viaje. Para iniciar el viaje, hacemos una doble comprobación por si hay desfase en los datos, se comprueba que no hay datos faltantes, que no está ya en ejecución, que la ventana de ajustes no tiene valores erróneos, y que los estados son correctos. Solo entonces establece la variable de ejecución del viaje (run) como verdadera.

Para detener los motores es más sencillo, se desactiva la ejecución del viaje (run) y ajustamos la palabra de control.

```
def modo_selec():
    global auto
    if auto_manu.value == "Automático":
        auto=True
    else:
        auto=False

def settings_function(): #Muestra la ventana de Ajustes
    global run
    if run == True:
        info("Info", "Solo se pueden modificar los ajustes en parada")
    else:
        ajustes.show()
        ajustes.tk.attributes('-topmost', True) #Me rompí la cabeza con esto,
gracias gracias, no hacen falta rosas

def apply_settings(): #Guarda los ajustes y cierra la ventana
    if sanitize()==False:
        ajustes.warn("Error de datos", "Introduce datos correctos antes de
guardar")
    else:
        settings_save()
        ajustes.hide()


def close_settings(): #Cierra la ventana sin guardar los ajustes
    if sanitize()==False:
        ajustes.warn("Error de datos", "Introduce datos correctos antes de
cerrar la ventana")
    else:
        ajustes.hide()

def fullscreen():
    if app.full_screen==False:
        app.set_full_screen()
    else:
        app.exit_full_screen()
```

La función fullscreen alterna el modo pantalla completa, ya que la tecla F11 no está asignada por defecto a esta función. Se observa que al contrario de ser un parámetro True o False, es una función específica que modifica el modo.

Lo que sigue a continuación es la función que se invoca cada segundo, esta función obtiene de las variables globales de estado y de UDP los valores necesarios, y los actualiza en los respectivos cambios de la interfaz.

```
########### ACTUALIZACION DE VALORES DE LA INTERFAZ #################
def values():
    global first_load
    #NMEA da la Latitud en DDMM.mmm, la longitud en DDDMM.mmm
    #DMS(Grados minutos y segundos) a DM.m(Grados minutos decimales) -> d=d
M.m= M+s/60
    #DM.m a D.d (grados decimales) -> d=M.m/60  D.d= D+d

    if first_load==True:
        settings_load()
        is_imperial.value=units_settings.value

    #Bateria
    try:
        if serial_battery!= "NaN":
            bat_estado.value = "Bateria: " + serial_battery +"%"
            if float(serial_battery) >= float(llena.value):
                bat_estado.bg = "pale green"
            elif (float(serial_battery) < float(llena.value)) and
(float(serial_battery) >= float(baja.value)):
                bat_estado.bg = "yellow"
            elif (float(serial_battery) < float(baja.value)) and
(float(serial_battery) >= float(critica.value)):
                bat_estado.bg = "orange"
            elif float(serial_battery) < float(critica.value):
                bat_estado.bg = "red"
        else:
            bat_estado.value = "Bateria: NaN%"
            bat_estado.bg = "grey"
    except:
        bat_estado.value = "Bateria: NaN%"
        bat_estado.bg = "grey"

    #Intensidades de los motores
    try:
        if serial_current_I!= "NaN":
            miestado.value = "MI: " + str(int(serial_current_I)) +"A"
            if int(serial_current_I) < int(motorI_max.value):
                motorI_estado.bg = "pale green"
            else:
                motorI_estado.bg = "red"
        else:
            miestado.value = "MI: NaN"
            motorI_estado.bg = "grey"
    except:
        miestado.value = "MI: NaN"
        motorI_estado.bg = "grey"

    try:
        if serial_current_D!= "NaN":
            mdestado.value = "MD: " + str(int(serial_current_D)) +"A"
            if int(serial_current_D) < int(motorD_max.value):
                motorD_estado.bg = "pale green"
            else:
                motorD_estado.bg = "red"
```

```
        else:
            mdestado.value = "MD: NaN"
            motorD_estado.bg = "grey"
    except:
        mdestado.value = "MD: NaN"
        motorD_estado.bg = "grey"
    #Potencia de los motores
    mipw.value=str(int((MotorI-511)/512*100))+"%"
    mdpw.value=str(int((MotorD-511)/512*100))+"%"

    #Arduino
    ardu_estado.value  = "Arduino: " + arduino_status
    if arduino_status == "ON":
        ardu_estado.bg = "pale green"
    elif arduino_status == "WAIT":
        ardu_estado.bg = "orange"
    elif (arduino_status == "ERR") or (arduino_status == "PLUG") :
        ardu_estado.bg = "red"

    #OpenCPN
    OCPNestado.value = "OpenCPN: " + opencpn_status
    if opencpn_status == "ON":
        OCPNestado.bg = "pale green"
    elif opencpn_status == "OFF":
        OCPNestado.bg = "red"
    elif opencpn_status == "WAIT":
        OCPNestado.bg = "orange"
    elif opencpn_status == "ERR":
        OCPNestado.bg = "red"

##    #Estado
##
    if opencpn_status == "OFF" or arduino_status == "OFF" or arduino_status
== "ERR" or opencpn_status == "ERR":
        estado.value = "**Sistema\nIndisponible**"
        fondo_estado.bg = "orange red"
    elif opencpn_status == "WAIT" or arduino_status == "WAIT":
        fondo_estado.bg = "pale green"
        estado.value = "**Sistema\na la espera**"
    elif opencpn_status == "ON" and arduino_status == "ON":
        fondo_estado.bg = "pale green"
        estado.value = "**Sistema\nDisponible**"

    # Actualización de datos sistema imperial
    if is_imperial.value=="1":
        boat_lat.value = RMC[3] + RMC[4] #Latitud
        boat_long.value = RMC[5] + RMC[6] #Longitud
        boat_sog.value = RMC[7] #Speed over Ground (knots)
        boat_rm.value = HDT[1] #Magnetic variation degrees (Easterly var.
subtracts from true course)
        boat_rv.value = RMC[8] #True bearing - Track angle in degrees (True)
        boat_UTC.value = RMC[10]

        way_lat.value = RMB[6] + RMB[7] #Latitud del siguiente waypoint
        way_long.value = RMB[8] + RMB[9] #Longitud del siguiente waypoint
        way_distance.value = RMB[10] #Distancia al waypoint (en millas
nauticas)
        way_rv.value = RMB[11] #Bearing to destination, degrees True

        way_sog.value = RMB[12] #Destination closing velocity in knots
        way_destino.value = RMB[5]
```

Para obtener la temperatura del sistema RPi, se invoca a la función `CPUTemperature().temperature` la cual proviene de la librería gpiozero.

```
        cpu_temp.value =
"Temp="+str(round(CPUTemperature().temperature+273,2))+"°K\nUTC
"+str(datetime.now(timezone.utc).date())+"\n"+str(datetime.now(timezone.utc).
time())

    else: # Actualización de datos SI
        try:
            boat_lat.value = str(round(float(RMC[3][0:2]),7)+
round(float(RMC[3][2:])/60,7)) #Latitud
            if RMC[4] =="S": boat_lat.value=str(-float(boat_lat.value))
        except: boat_lat.value = "NaNNaN"

        try:
            boat_long.value = str(round(float(RMC[5][0:3]),7)+
round(float(RMC[5][3:])/60,7)) #Longitud
            if RMC[6] =="W": boat_long.value=str(-float(boat_long.value))
        except: boat_long.value = "NaNNaN"
        boat_sog.value = RMC[7] #Speed over Ground (knots)
        boat_rm.value = HDT[1] #Magnetic variation degrees (Easterly var.
subtracts from true course)
        boat_rv.value = RMC[8] #True bearing - Track angle in degrees (True)
        boat_UTC.value = RMC[10]

        try:
            way_lat.value = str(round(float(RMB[6][0:2]),7)+
round(float(RMB[6][2:])/60,7)) #Latitud del siguiente waypoint
            if RMB[7] =="S": way_lat.value=str(-float(way_lat.value)) #Check
signo
        except: way_lat.value = "NaNNaN"
        try:
            way_long.value = str(round(float(RMB[8][0:3]),7)+
round(float(RMB[8][3:])/60,7)) #Longitud del siguiente waypoint
            if RMB[9] =="W": way_long.value=str(-float(way_long.value))
#Check signo
        except: way_long.value = "NaNNaN"
        try: way_distance.value = str(round(float(RMB[10])*1.852,6))
#Distancia al waypoint (en millas nauticas)
        except: way_distance.value = "NaN"
        way_rv.value = RMB[11] #Bearing to destination, degrees True
        way_sog.value = str(round(float(RMB[12])*1.852,3)) #Destination
closing velocity in knots
        way_destino.value = RMB[5]
        cpu_temp.value =
"Temp="+str(round(CPUTemperature().temperature,2))+"°C\nUTC
"+str(datetime.now(timezone.utc).date())+"\n"+str(datetime.now(timezone.utc).
time())
```

Esta print es muy importante:

```
    #!!!! Este inofensivo print y flush, es el que confirma al lanzador que
el piloto sigue funcionando correctamente
    print(1)
    sys.stdout.flush()
    #!!!!
```

Literalmente imprime un uno en su línea de comando y la vacía. Esto lo que hace es enviar un valor a otro programa que se encarga de monitorear que el VNAS no está colgado, si se ha quedado colgado procede a matar a la aplicación y reiniciarla. Para evitar que se acumulen los datos en memoria, hacemos un flush del Shell de VNAS.

3.2. Starter

El starter es el programa que se inicia al encender la RPi, y se encarga de lanzar OpenCPN y VNAS. Y como se dice más arriba, controla la ejecución del VNAS con la impresión de un 1 en su Shell.

Este es relativamente más sencillo, pero involucra 2 librerías que no se usan en el código principal del VNAS. Subprocess (21) y select (22). Subprocess nos servirá para realizar las llamadas a las aplicaciones y tomar posesión de ellas, select nos capacitará para hacer poll a las aplicaciones y comprobar si se están ejecutando y cuál es su ouput.

Además, crea un log con cada inicio, intentos y lanzamientos fallidos.

```
import time, datetime, subprocess, sys
from select import select

def log_function(cadena, estado):
    #Abrimos log del sistema
    archivoLog = open("starter_log.txt", "a")
    #Siempre UTC
    timeStamp = datetime.datetime.utcnow()
    archivoLog.write(cadena % (timeStamp, estado))
    archivoLog.flush()
    archivoLog.close()

global inicio
inicio="[E] %s Iniciando...%s\n"
global cpn_start_error
cpn_start_error = "[!] %s No se ha podido lanzar OpenCPN%s\n"
global vnas_start_error
vnas_start_error = "[!] %s No se ha podido lanzar VNAS%s\n"
global cpn_start
cpn_start = "[✓] %s OpenCPN Lanzado%s\n"
global vnas_start
vnas_start = "[✓] %s VNAS Lanzado%s\n"
global retry_error
retry_error= "[✓] %s Código de salida %d Cierre por orden de usuario\n"
global salida
salida = "[✓] %s Código de salida %d Cierre por orden de usuario\n"
global retry
retry = "[Δ] %s Código de salida %d Relanzado\n"

log_function(inicio,"")
# Al ejecutarse le damos un tiempo prudencial para que carguen todas las
librerias
time.sleep(20)
# Ejecutamos OpenCPN primero
try:
    opencpn = subprocess.Popen(["opencpn"])
except:
```

```
    log_function(cpn_start_error,"")

if opencpn != None:
    log_function(cpn_start,"")
# Damos un tiempo para que cargue
time.sleep(10)
# Iniciamos el programa principal como un proceso al que tenemos acceso a su
salida de datos
try:
    vnas = subprocess.Popen(["python3",
"/home/pi/Desktop/vnas.py"],stdout=subprocess.PIPE)
except:
    log_function(vnas_start_error,"")

if vnas != None:
    log_function(vnas_start,"")

attempts=0

while True:
    # Compruebo que el proceso sigue funcionando
    estado = vnas.poll()
    estado_2 = opencpn.poll()
    # Vuelco salida del proceso y compruebo que emite el mensaje "estoy vivo"
    WatchDog = time.time() # Guardo valor time antes de hacer el poll
    poll_result = select([vnas.stdout], [], [], 10)[0] # Hago el poll del
proceso con un limite de tiempo de 10 seg
    print(poll_result)
    if abs(WatchDog - time.time())>5: # Si ha pasado mas de 5 segundos, se ha
quedado pillado
        vnas.terminate() # Cierro proceso
        attempts+=1 # Incremento intentos
        estado=0
        #print(attempts)
        if attempts > 5: # Si hemos superado 5 intentos seguidos, dejamos de
intentar (Incluimos funcion de alerta aqui)
            log_function(retry_error,"")
            estado=66
            sys.exit()
    else:
        attempts=0 # Si funciona normalmente, intentos a cero
        poll_result=""

# Ejecución de estados
    if estado == 42: # Programa terminado de manera natural, así es la vida
        log_function(salida,estado)
        sys.exit()
    elif estado != None: # Programa terminado de manera anormal, relanzando
        log_function(retry,estado)
        try:
            vnas = subprocess.Popen(["python3",
"/home/pi/Desktop/vnas.py"],stdout=subprocess.PIPE)
        except:
            log_function(vnas_start_error,"")
        if vnas != None:
            log_function(vnas_start,"")
    else: # Continua ejecutandose de manera natural
        time.sleep(1)
```

3.3. Comunicación Serial - Arduino

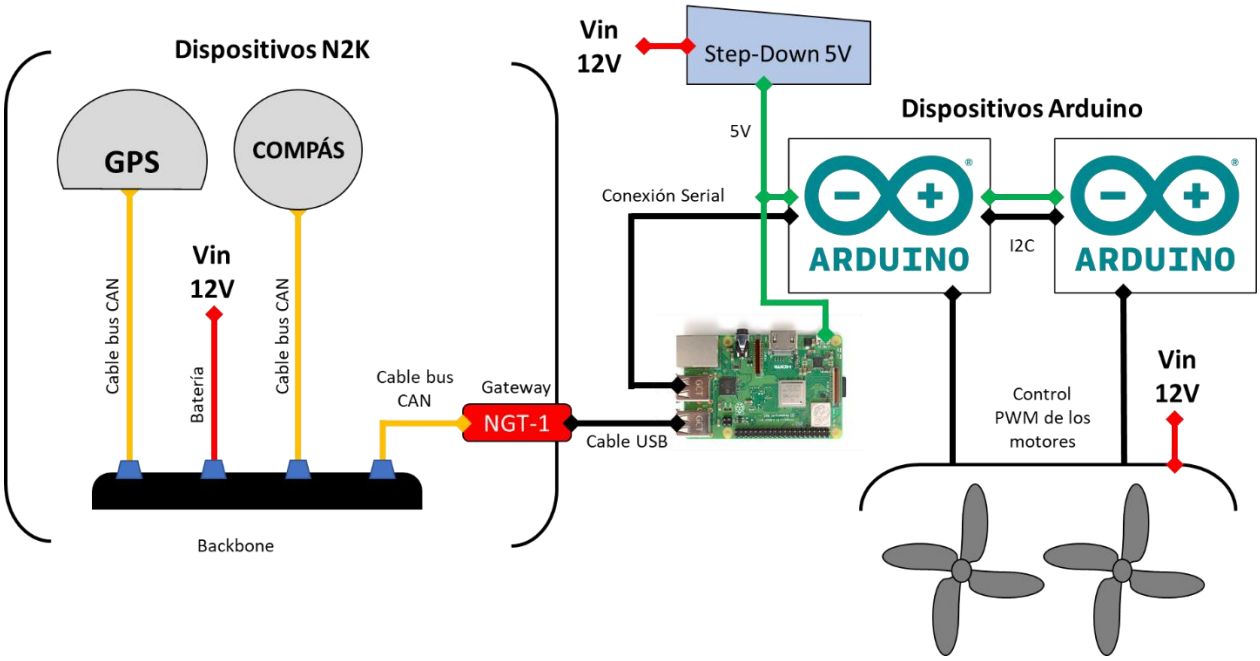


Figura 41 - Alimentación y datos Arduino-RPi (7) (23)

En la Figura 41 se puede ver que los motores están controlados por sendos Arduinos, uno para cada motor. RPi solo tiene conexión directa por serie con uno de los dos Arduinos, da igual cual sea, siempre y cuando sea el Arduino maestro, ya que este recibe la sentencia de control de RPi, y envía la parte necesaria al otro Arduino por I2C para el correcto funcionamiento. Además, se recibe un feedback siguiendo el camino inverso, con el consumo de los motores, el nivel de carga de la batería y para que VNAS detecte el estado y el puerto de conexión de Arduino, se ha creado la siguiente función:

```
def detect_arduino(): #Sirve para detectar dinámicamente el puerto de
ubicación actual de Arduino
    ports = serial.tools.list_ports.comports(include_links=False) #Obtenemos
la lista de todos los serial
    for port in ports : #Por cada puerto en la lista de puertos
        if "VID:PID=2341:0043" in port.hwid: #Buscamos la identificación del
arduino en los puertos, VID:PID=2341:0043 ARDUINO UNO ### VID:PID=2341:8037
ARDUINO MICRO
            return port.device #Devolvemos la interfaz (String)
    return False #Si no detecta al dispositivo, devuelve un False para
comprobaciones lógicas
```

La función se sirve de la librería Serial (24) en Python para listar todas las conexiones USB a las que tiene acceso el SO. Una vez que obtenemos el listado, usamos la sintaxis “for in” de Python para buscar un dato/valor dentro de una cadena más compleja. En este caso lo que hacemos es un for para buscar dentro de cada campo de la lista (puertos), y otro for para buscar dentro del puerto, en el campo hwid (Hardware Identification) la identificación única de nuestro Arduino.

Para las pruebas en laboratorio se usó un Arduino Uno, por lo que ya se proporciona la identificación en los comentarios del código, pero el identificador usado realmente es el de Arduino Micro, el que está incorporado en los motores.
Si se utiliza un Arduino Micro clónico o se cambia por otra versión habrá que averiguar el identificador de este nuevo Arduino con la función detect_arduino(), simplemente incluye la librería serial y el código aquí indicado pero con un print(port.hwid) en vez del segundo for.

En el código principal, al lanzar la función, si se localiza el identificador, la función devuelve la localización (Device), si no lo encuentra devuelve False.

Hay que darse cuenta que la función devolverá solo el primer Arduino que se detecte. Ya que en este desarrollo solo se usa un Arduino haciendo de maestro. Si se desea que se devuelvan todas las conexiones disponibles con esa ID, hay que declarar una variable intermedia que acumule los (Devices) y sea la que se devuelva, puedes tratar la salida en tu código.

Una vez que tenemos definida la función para obtener la interfaz usada por Arduino, creamos una conexión Serial para comunicarnos.

```
def crear_arduino(): #Crea y abre la conexión con arduino
    global arduino #Invoco la global a usar para la conexión serial
    try:arduino = serial.Serial(detect_arduino(), 9600, timeout = 0) #Intento
crear la conexion
    except: return False
    return True
```

Hemos denominado “arduino” a una variable global a la que asignamos el puerto serial. Lo que nos permitirá acceder a la conexión serial desde todas las funciones. En los argumentos de establecimiento de la conexión hay que facilitar:

- La identificación del puerto físico con el que se quiere realizar la conexión. En nuestro caso gracias a la función detect_arduino() nos devuelve la identificación si existe.
- La tasa de baudios (número de unidades de señal por segundo). Arduino funciona a 9600 baudios.
- El parámetro opcional timeout, al establecerlo en cero conseguimos que la conexión no sea bloqueante, es decir, que no se quede esperando la recepción de datos bloqueando la ejecución del código, si no que, si falla al recibir datos o buffer vacío, continua su ejecución.

Esta asignación serial está controlada por la función try de Arduino. Si no resulta posible establecer la conexión devuelve un False.

Este es el código principal de comunicación entre la RPi y Arduino, se ejecuta cada segundo y medio por defecto, ya que no es necesaria una alta cadencia de actualización de los valores.

En el mismo código realizamos la gestión de errores y estado de Arduino.

```
def comunicacion_serial(): #Función de transmisión y recepción de datos
    global control #Cadena de control enviada a Arduino
    global run #Flag de ejecución de navegacion
    global auto #Flag de modo automatico/manual
    global arduino #Puerto de transmisión
    global arduino_status #Estado de Arduino - GUI
    global arduino_try #Flag de intentos de conexión Arduino - GUI
    global serial_battery #Valor de carga de la bateria
    global serial_current_I #Intensidad consumida motor Izq.
    global serial_current_D #Intensidad consumida motor Der.
    global serial_mode #Flag de modo automático/manual, sobreescribe al flag
auto
    x=""
    x_utf8=None
    x_struct=""

    if arduino==None: #Si Arduino no está configurado todavía,
        crear_arduino() # creo la configuración
        if arduino==None: #Si despues de crearlo sigue siendo None, ha
fallado la creación
            if (arduino_status != "PLUG") and detect_arduino()==False: #Si no
está ya en modo PLUG, activarlo
                arduino_status = "PLUG"
                arduino_try=0
                log_estado("[!!] Conecta físicamente Arduino Micro", "orange
red")
                return
                #He intentado crearlo y no he podido, no se ha conectado
todavía arduino, hay que conectarlo
        else: #Por el contrario, se ha configurado satisfactoriamente
            arduino_status = "WAIT" #Paso a modo de espera
            arduino_try=0
            log_estado("[Δ] Arduino Micro está conectado, esperando a recibir
señal", "orange")
            #He creado satisfactoriamente la conexión con arduino, la
conexión está creada, falta ver si está libre

    if arduino!=None: #Si Arduino ya está configurado
        try: #Envío la cadena de control
            arduino.write(control.encode()) #Envio la cadena de control a
arduino
            x=arduino.readline()            #Recibo el feedback de Arduino
            arduino.flushInput()            #Limpiamos buffer
            arduino.timeout = 0             #Non-Blocking
            x_utf8=x.decode('utf8')         #Limpiamos cadena
            if x_utf8!= "":
                x_struct = str(x).split("'")
                x_struct = str(x_struct[1]).split(",")
                serial_battery = x_struct[0]
                serial_current_I = x_struct[1]
                serial_current_D = x_struct[2]
                serial_mode = x_struct[3]
        except: # Oye, que da la casualidad de que no puedes enviarla, pueden
ser 3 cosas
            #Procedemos a hacer una preciosa gestión de errores

    ###### CASO 1 Se ha desconectado físicamente
            if detect_arduino()==False: #No aparece conectado en los USB
                arduino_status ="PLUG" #Lo pasamos a estado PLUG
```

```
                arduino=None #Borramos anterior conexión
                arduino_try=0
                serial_battery = "NaN"
                serial_current_I = "NaN"
                serial_current_D = "NaN"
                serial_mode = "NaN"
                log_estado("[!!] Arduino se ha desconectado físicamente,
vuelve a conectarlo", "orange red")
                return

    ###### CASO 2 Se acaba de conectar y hay que esperar
        if (arduino_try<=2) and (x_utf8==""): #Las primeras lineas despues de
iniciar suelen ser en blanco, damos 3 intentos antes de consideralo error
            arduino_try+=1  #Incrementamos el conteo (1 intento aprox 1.5
segundos)
            return
        elif (arduino_try==3) and (x_utf8==""): #Si se alcanzan los 3
intentos de comunicación en blanco, o está configurando RPi los puertos, o
Arduino se ha colgado
            arduino_status ="WAIT"
            arduino_try+=1
            serial_battery = "NaN"
            serial_current_I = "NaN"
            serial_current_D = "NaN"
            serial_mode = "NaN"
            log_estado("[Δ] Serial posiblemente ocupado (espera 25
segundos)", "orange")
            return
        elif (arduino_try>3) and (x_utf8=="") and (arduino_try<20): #Si
pasamos mas allá de los 20 intentos (30 segundos), no es la RPi, (siempre
tarda 25 segundos)
            arduino_try+=1
            return
    ###### CASO 3 Arduino se ha quedado colgado
        elif (arduino_try==20) and (x_utf8==""):
            if arduino_status !="ERR": #Pasamos al modo "error", seguimos
intentando leer, pero dejamos claro que arduino no va
                arduino_status ="ERR"
                serial_battery = "NaN"
                serial_current_I = "NaN"
                serial_current_D = "NaN"
                serial_mode = "NaN"
                log_estado("[!!] Arduino no responde, prueba a reiniciarlo",
"orange red")
            return
        if ((arduino_status == "PLUG") or (arduino_status == "WAIT") or
(arduino_status == "ERR")) and (x_utf8!=""): #Si tenemos la suerte de recibir
dos lineas seguidas, estamos de vuelta
            if arduino_try==20:
                arduino_try=0
            else:
                arduino_status = "ON"
                log_estado("[✓] Arduino Micro está operativo", "light blue")
                arduino_try=0
            return
```

En ella se envía la sentencia de control de los motores codificada en ASCII. Y el feedback recibido en forma de sentencia, se separa mediante las comas de separación y se almacena en las variables globales correspondientes.

3.4. Comunicación UDP - OpenCPN

En VNAS para recibir la información de telemetría y waypoint declaramos una conexión UDP que ligamos al puerto 2947, que es por el cual recibiremos las sentencias NMEA0183. Al igual que con Arduino, la conexión se establece para que sea no bloqueante. La librería usada es socket (25).

```
########################## SETUP DE UDP / OPENCPN ##########################
#                       Configuración UDP, IP y puertos                    #
###########################################################################

#Declaracion y configuracion del puerto UDP por el que OpenCPN envia los
datos
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setblocking(0)
server_address = ('localhost', 2947)
sock.bind(server_address)
```

Para leer los datagramas que recibimos por parte de OpenCPN, ejecutamos cada milisegundo la función `refresh()`.Esta función intenta leer los datos que pudieran haberse acumulado en el puerto UDP, si no recibe ningún tipo de dato durante más de 2 segundos, damos por hecho que hay un error de conexión con OpenCPN. Si recibimos datagramas, pero ninguno de los que se ha recibido es información del GPS en al menos 2 segundos, hay un problema con la telemetría. Si, por el contrario, lo que no se recibe es la información de waypoint, no hay un waypoint activo.

```
###########################################################################
#################        FUNCION DE ADQUISICION UDP         #################
##       Adquisición de las sentencias NMEA0183 por parte de OpenCPN       ##
###########################################################################
###########################################################################

def refresh(): #Se ejecuta cada ciclo

    #Invocación de variables
    global data
    global data_try
    global open_crash
    global opencpn_status
    global waypoint_data
    global boat_data
    global run

    #Tenemos que hacer try, si no está disponible da error
    try:
        data, address = sock.recvfrom(200)
    except:
        data=""

    if data != "": # Habemus lectura UDP
        data_try=0 # Siempre que hay una lectura reseteamos data_tray
        datos = str(data).split(',') # Separamos los distintos campos de la
instrucción NMEA
        if datos[0]=="b'$ECRMB": # Datos de Waypoint
            for i in range(len(datos)):
```

```
                RMB[i] = datos[i]
                waypoint_data=0
                if opencpn_status== "WAIT":
                    opencpn_status="ON"
                    log_estado("[✓] OpenCPN tiene una ruta activa", "light
blue")
        elif datos[0]=="b'$GPRMC": # Datos de Barco
            for i in range(len(datos)):
                RMC[i] = datos[i]
                boat_data=0
                if opencpn_status== "ERR":
                    opencpn_status="ON"
                    log_estado("[✓] OpenCPN vuelve a recibir datos GPS",
"light blue")
        elif datos[0]=="b'$IIHDT": # Datos del compas
            for i in range(len(datos)):
                HDT[i] = datos[i]
        data=None #Tras guardar los datos borramos data
```

Arriba se puede observar cómo nos valemos de nuevo de la sintaxis for in de Python para hacer un barrido rápido en todos los elementos de los datagramas, y almacenando en las variables globales la telemetría necesaria para el control.

A continuación, se observa la gestión de estados y timeouts para OpenCPN, como se puede observar es sencilla y exclusiva, usamos la librería time para medir exactamente segundos del sistema y no se usan variables intermedias para la contabilización del tiempo, ahorrando varios ciclos.

```
        if boat_data!=0 and (time.time()-boat_data)>2 and
(opencpn_status=="ON"):
            opencpn_status = "ERR"
            log_estado("[!!] OpenCPN no envia datos de GPS. Verifique
conexión física", "orange red")
        elif opencpn_status!= "ERR" and opencpn_status!= "OFF" and
boat_data==0:
            boat_data=time.time()
        if waypoint_data!=0 and (time.time()-waypoint_data)>2 and
opencpn_status=="ON":
            opencpn_status = "WAIT"
            log_estado("[Δ] OpenCPN no tiene una ruta activa", "orange")
        elif opencpn_status!= "OFF" and opencpn_status!= "WAIT" and
waypoint_data==0:
            waypoint_data=time.time()
        if opencpn_status== "NaN" or opencpn_status== "OFF":
            opencpn_status="ON"
            log_estado("[✓] OpenCPN está operativo", "light blue")

    elif data == "" and data_try==0: #Si OpenCPN deja de enviar señal, dejo
de navegar(Watchdog)
        data_try=time.time()
        #print(data_try)
    elif data == "" and (time.time()-data_try)>2 and opencpn_status!= "OFF":
        opencpn_status = "OFF"
        boat_data=0
        waypoint_data=0
        log_estado("[!!] OpenCPN no está ejecutandose", "orange red")
        return
```

3.5. Parámetros de Ajuste

Hay que dotar a VNAS la capacidad de modificar sus parámetros, y que estas modificaciones sean persistentes para cuando se vuelva a iniciar el programa.

Al principio se indicaba que se desea un sistema ajustable, es por ello que se ha creado una ventana a la que se accede desde el menú de Ajustes, y en la que se nos permite modificar la gran mayoría de parámetros que forman el lazo de control y distintas alarmas y preferencias, evitando así el Hard-Code de parámetros. En el apartado de GUI se incluía el diseño de esta ventana de ajustes, la cual se muestra abajo.

Figura 42 - Ventana de edición de Ajustes

Los valores que se ven son los que el programa crea por defecto si no existe o se ha dañado el archivo de ajustes. Para mantener estos valores los guardamos en un archivo de texto plano que se carga al iniciar el VNAS. Este archivo tiene el aspecto que se muestra en la Figura 43.

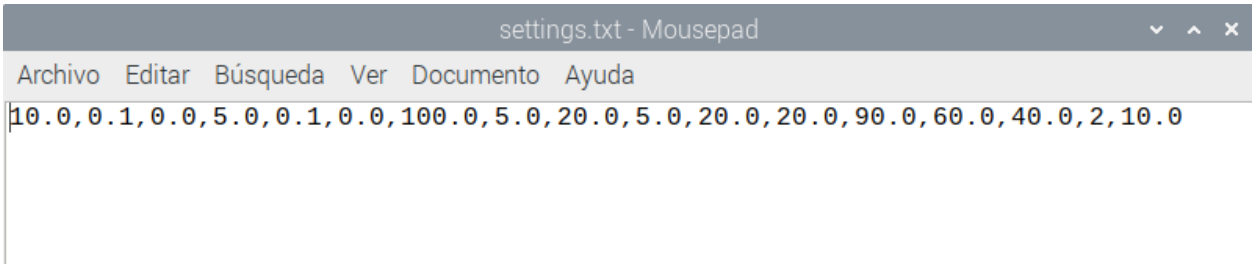


Figura 43 - Archivo de ajustes

Donde cada campo de la ventana de ajustes está separado por una coma. Primero vamos a explicar la función de manejo de archivos de Python. Básicamente funciona de manera muy parecida a cualquier otro lenguaje que hayas podido manejar.

Primero se tiene que abrir un archivo `open()` después escribimos `write()` o leemos `read()` y cuando hemos terminado, cerramos y liberamos el archivo `close()`. Cuando usamos la función `open()`, hay distintos modos en el que abrir un archivo, pero a todos ellos se le puede añadir un + para indicar que es de lectura y escritura:

Modo	Función
r	Modo por defecto, lectura.
w	Para escritura, si no existe el archivo, lo crea. Si existe, lo trunca.
x	Crea un nuevo archivo, si existe devuelve un error FileExistsError.
a	Para añadir al archivo, si no existe el archivo, lo crea.
t	Modo por defecto, texto.
b	Abre el archivo en modo binario.

Tabla 2 - Referencias de edición de archivos

Al lanzarse el VNAS, lo primero que hace es intentar cargar los parámetros de ajuste almacenados en el archivo “settings.txt” ubicado en el escritorio de la RPi. Usamos el modo “x+”, si no existe el archivo, se crea y se escriben los datos por defecto y cerramos. Seguidamente volvemos a abrir el archivo y procedemos a leer esos mismos datos que hemos escrito, y los almacenamos en los valores que deben mostrarse en la interfaz, nos ahorramos el usar variables intermedias. Ciertamente al cargar los datos en la interfaz puede entrañar riesgos, pero si se realiza un adecuado trato de los datos, no habrá problema alguno.

```
def settings_load(): #Carga al inicio los parámetros previamente guardados
del autopiloto
    global first_load
    first_load=False
    try:#Comprobamos que el archivo existe
        with open("settings.txt", "x+") as settings_file: #Si el archivo no
existe previamente, lo creamos y guardamos los ajustes por defecto

settings_file.write("10,0.1,0,5,0.1,0,100,5,20,20,20,90,60,40,2,10")
            settings_file.close()
    except FileExistsError: #Si el archivo existe no hace falta hacer nada
        settings_dump=""

    try: #Seguidamente procedemos a cargar los parámetros
        with open("settings.txt", "r") as settings_file:
            settings_dump=settings_file.read()
            datos = str(settings_dump).split(',')
            for i in range(len(datos)):
                try:
                    float(datos[i])
                except ValueError:
                    os.remove("settings.txt")
                    settings_load()
                    info("Error de carga", "Error al comprobar los datos del
archivo de configuración, el archivo está dañado o ha sido modificado con
valores no válidos")
                    return

            ####PID1
            input_P.value = datos[0]
            input_I.value = datos[1]
            input_D.value = datos[2]
            ####PID2
            input_P2.value = datos[3]
            input_I2.value = datos[4]
            input_D2.value = datos[5]
            ####Velocidad
            speed_max.value = datos[6]
```

```
            speed_incr.value = datos[7]
            speed_sog.value = datos[8]
            ####Alarmas
            motorI_max.value = datos[9]
            motorD_max.value = datos[10]
            llena.value = datos[11]
            baja.value = datos[12]
            critica.value = datos[13]
            #Sistema de unidades
            units_settings.value = datos[14]
            #Distancia de llegada
            arrival_dist.value = datos[15]

            settings_file.close()

    except:
        info("Error de carga", "Error al comprobar los datos del archivo de
configuración, el archivo está dañado o bloqueado por otro proceso")
```

Con los problemas nos referimos a la introducción por error o de forma maliciosa de valores fuera del rango o que no se ajusten al tipo deseado, como por ejemplo que se introduzcan letras o carácter especiales, convirtiéndose en valores que no se pueden tratar matemáticamente en nuestro código. Es por ello que solo se puede modificar los datos de la interfaz cuando el barco está con los motores detenidos. Si se accede a la interfaz mientras se está realizando una ruta, los campos de los parámetros aparecerán deshabilitados. Manteniéndolos a salvo como se muestra a continuación en la Figura 44.

Figura 44 - Ajustes inhabilitados

Si por el contrario se mantiene abierta la ventana de ajustes con valores no admisibles y se intenta iniciar una ruta, se marcan todos los que incumplen las condiciones y se muestra un cartel de aviso (Figura 45). Con esto, aunque esté abierta la ventana no se pueden modificar los valores, si se lanza el viaje y está abierta, se cierra automáticamente y se deshabilitan los campos. Si por el contrario se quieren poner valores no admisibles, al salir o al intentar aplicar los cambios, se ejecuta una función de "limpieza" para comprobar que cada campo cumple con las especificaciones.

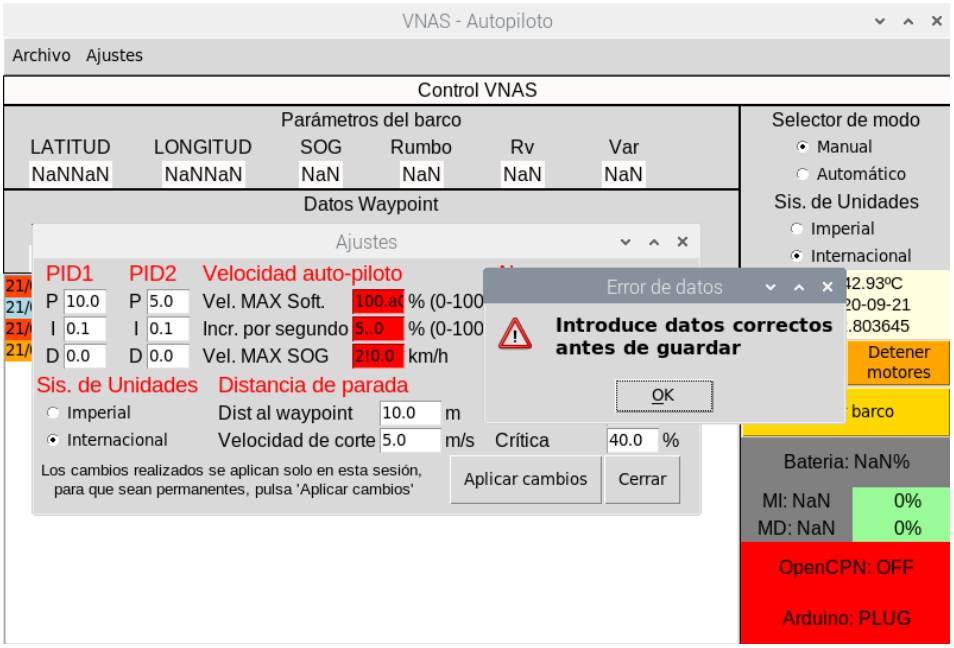


Figura 45 - Valores no admisibles en Ajustes

Esta función de limpieza usa la librería “re”, esta librería nos proporciona la capacidad de usar expresiones regulares para buscar en nuestros strings. Se construyen expresiones siguiendo las siguientes reglas:

Caracteres no especiales se igualan. Las excepciones son los caracteres especiales	
\	Salta un carácter especial al principio de una secuencia
.	Iguala cualquier carácter menos “nueva línea”
^	Iguala al principio del string
$	Iguala al final del string
[]	Compacta un set the caracteres
R|S	Iguala cualquiera de las expresiones regulares R o S.
Después de un '[', compacta un set y los únicos caracteres especiales son	
]	Al final del set, si no es el primero
-	Para un rango, ej. a-c iguala a, b o c
^	Nega el set solo si es el primer carácter
Cuantificadores (añadir ? para que no sean inclusivos):	
{m}	Exactamente m repeticiones
{m,n}	De m (por defecto 0) a n (por defecto infinito)
*	0 o mas. Igual que {,}
+	1 o mas. Igual que {1,}
?	0 o 1. Igual que {,1}

Secuencias especiales	
\A	Comienzo de un string
\d	Digito
\D	No digito
\s	Espacios en blanco UNICODE [\t\n\r\f\v]
\S	Espacios no en blanco
\w	Alfanumérico: [0-9a-zA-Z_]
\W	No alfanumérico
\Z	Final del string

Tabla 3 - Operadores para expresiones regulares (26)

Para verificar el contenido usamos las siguiente expresión `'[^\.^0-9]|[\.]{2,}'` dentro de la función `re.search` la cual nos indica si existe la expresión en el string que se indica:

`'[^\.^0-9]|[\.]{2,}'`

[Creamos un set de caracteres

^ Y Negamos

\. El carácter de punto que se indica con la barra invertida para que reconozca el interpretador que es un punto y no su expresión de carácter especial.
→ Cualquier carácter que no sea un punto

^ Negamos

0-9 Un rango de 0 al 9
→ Cualquier carácter que no sea un numero

] Y cerramos el set
→ Cualquier carácter que no sea un numero ni un punto

| O

[Creamos un set de caracteres

\. El carácter de punto que se indica con la barra invertida para que reconozca el interpretador que es un punto y no su expresión de carácter especial.

] Y cerramos el set

{2,} Que la expresión anterior se repita de 2 a infinito nº de veces
→ Cualquier combinación de puntos mayor o igual de 2

Esta expresión hace que solo números y un único punto como máximo nos confirme la validez del campo. Cada campo es individual para tener unas condiciones separadas para cada campo:

```
def sanitize():
    error=0
    if (re.search('[^\.^0-9]|[\.]{2,}', input_P.value)==None) and
(re.search('[0-9]|', input_P.value)!=None):
        input_P.bg="white"
    else:
        input_P.bg="red"
```

```
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', input_I.value)==None) and
(re.search('[0-9]', input_I.value)!=None):
        input_I.bg="white"
    else:
        input_I.bg="red"
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', input_D.value)==None) and
(re.search('[0-9]', input_D.value)!=None):
        input_D.bg="white"
    else:
        input_D.bg="red"
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', input_P2.value)==None) and
(re.search('[0-9]', input_P2.value)!=None):
        input_P2.bg="white"
    else:
        input_P2.bg="red"
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', input_I2.value)==None) and
(re.search('[0-9]', input_I2.value)!=None):
        input_I2.bg="white"
    else:
        input_I2.bg="red"
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', input_D2.value)==None) and
(re.search('[0-9]', input_D2.value)!=None):
        input_D2.bg="white"
    else:
        input_D2.bg="red"
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', speed_max.value)==None) and
(re.search('[0-9]', speed_max.value)!=None):
        speed_max.bg="white"
    else:
        speed_max.bg="red"
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', speed_incr.value)==None) and
(re.search('[0-9]', speed_incr.value)!=None):
        speed_incr.bg="white"
    else:
        speed_incr.bg="red"
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', speed_sog.value)==None) and
(re.search('[0-9]', speed_sog.value)!=None):
        speed_sog.bg="white"
    else:
        speed_sog.bg="red"
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', motorI_max.value)==None) and
(re.search('[0-9]', motorI_max.value)!=None):
        motorI_max.bg="white"
    else:
        motorI_max.bg="red"
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', motorD_max.value)==None) and
(re.search('[0-9]', motorD_max.value)!=None):
        motorD_max.bg="white"
    else:
        motorD_max.bg="red"
        error=1
```

```
    if (re.search('[^\.^0-9]|[\.]{2,}', llena.value)==None) and
(re.search('[0-9]', llena.value)!=None):
        llena.bg="white"
    else:
        llena.bg="red"
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', baja.value)==None) and
(re.search('[0-9]', baja.value)!=None):
        baja.bg="white"
    else:
        baja.bg="red"
        error=1
    if (re.search('[^\.^0-9]|[\.]{2,}', critica.value)==None) and
(re.search('[0-9]', critica.value)!=None):
        critica.bg="white"
    else:
        critica.bg="red"
        error=1

    if (re.search('[^\.^0-9]|[\.]{2,}', arrival_dist.value)==None) and
(re.search('[0-9]', arrival_dist.value)!=None):
        arrival_dist.bg="white"
    else:
        arrival_dist.bg="red"
        error=1
    if error==1:
        return False
    else:
        return True
```

Para guardar los datos de los ajustes, primero comprobamos si existe el archivo de ajustes. Podemos sobrescribirlo con write, o borrarlo y crear uno nuevo, hemos optado por la segunda opción:

```
def settings_save():
    #Comprobamos que el archivo existe

    try:
        with open("settings.txt", "x") as settings_file: #Si el archivo no
existe

settings_file.write(input_P.value+","+input_I.value+","+input_D.value+","+inp
ut_P2.value+","+input_I2.value+","+input_D2.value+","+speed_max.value+","+spe
ed_incr.value+","+speed_sog.value+","+motorI_max.value+","+motorD_max.value+"
,"+llena.value+","+baja.value+","+critica.value+","+units_settings.value+","+
arrival_dist.value)
            settings_file.close()



    #Si el archivo existe preguntamos si queremos sobreescribir los ajustes
    except FileExistsError:
        try:
            os.remove("settings.txt")
            with open("settings.txt", "w") as settings_file: #Si el archivo
existe

settings_file.write(input_P.value+","+input_I.value+","+input_D.value+","+inp
ut_P2.value+","+input_I2.value+","+input_D2.value+","+speed_max.value+","+spe
```

```
ed_incr.value+","+speed_sog.value+","+motorI_max.value+","+motorD_max.value+"
,"+llena.value+","+baja.value+","+critica.value+","+units_settings.value+","+
arrival_dist.value)
                settings_file.close()

        except:
            with open("settings.txt", "w") as settings_file: #Si el archivo
te lo borran a medio ejecutar el ciclo (O_O)

settings_file.write(input_P.value+","+input_I.value+","+input_D.value+","+inp
ut_P2.value+","+input_I2.value+","+input_D2.value+","+speed_max.value+","+spe
ed_incr.value+","+speed_sog.value+","+motorI_max.value+","+motorD_max.value+"
,"+llena.value+","+baja.value+","+critica.value+","+units_settings.value+","+
arrival_dist.value)
                settings_file.close()
```

3.6. Log de eventos, logbook y estados

Un registro o "log" de eventos cumple la importante tarea de almacenar y mostrar al usuario el estado actual e indicar cualquier error o evento que requiera de su atención a través de una lista. Esta lista se va actualizando automáticamente y asignando un color distinto a cada evento según el tipo que sea (avisos, errores, confirmaciones, etc.).

Para ello se ha diseñado la función `log_estado(message,colour)` donde se facilitan como argumentos de la función el mensaje que se quiere mostrar, y el color que debe tener, el color se pasa como un string siguiendo la convención de nombres de Tcl en la que se basa Tkinter. El mensaje igualmente, es un string, pero añadimos al principio del string [!!], [Δ], [✓], según si es un error, un aviso o una confirmación respectivamente.

Color		Estado relacionado
orange	(255,165,0)	Alerta / Batería
orange red	(255,69,0)	Error / Batería
yellow	(255,255,0)	Batería
pale green	(152,251,152)	Todo correcto / Funcionando correctamente / Batería
grey	(128,128,128)	Desactivado / Sin datos
light blue	(173,216,230)	Confirmación

Tabla 4 - Colores de estados y avisos en la interfaz

Símbolo	Estado relacionado
[!!]	Error / Fallo crítico
[Δ]	Alerta / Aviso
[✓]	Confirmación / Acción completada correctamente

Tabla 5 - Símbolos de estado en el log

Este mensaje que se le proporciona a la función, se le suma a la fecha y hora del sistema y se añade como el primer elemento de la lista de eventos con el fondo del color indicado con las siguientes funciones:

```
log_estado_list.insert(0, time_stamp+" "+message)
log_estado_list.itemconfig(0,background=colour)
```

Además, `log_estado` tiene doble funcionalidad, a la misma vez que muestra en el log de eventos, añade ese mismo evento a un logbook.

El logbook es un archivo de texto plano que se guarda en la carpeta log del escritorio. El código crea un archivo distinto para cada día, y en el va añadiendo al final del archivo los eventos (función append de escritura de archivo). Esto nos sirve para mantener un registro de todos los eventos que han trascendido. Cada vez que se genera un evento, también se realiza un volcado de los parámetros de telemetría de la interfaz.

```
#############################################################################
##########################         LOGBOOK        ##########################
## Funciones de registro,guardado y backup de los registros del autopiloto ##
#############################################################################
#############################################################################

def log_estado(message, colour):
    time_stamp=datetime.now(timezone.utc).strftime("%d/%m/%Y, %H:%M:%S")
#Estilo de fecha d/m/a para el log
    file_stamp=datetime.now(timezone.utc).strftime("%Y_%m_%d") #Estilo a/m/d
para mejor clasificación del archivo
    log_estado_list.insert(0, time_stamp+" "+message) #Insertamos el mensaje
en nuestro log
    log_estado_list.itemconfig(0,background=colour) #Asignamos el color al
item del log recién añadido
    #Procedemos a abrir o crear el log para el día actual (separamos los logs
por días)
    #En este log solo guardamos los eventos, el volcado de los datos de
eventos se hace aparte
    file = open("log/"+file_stamp+"_VNAS_eventos.txt","a")
    file.write("\n")
    file.write(time_stamp+" "+message)
    file.close()
    #Realizamos el volcado de datos relacionados con eventos (se buscan por
time_stamp idénticos)
    dump="Bateria:"+ serial_battery +"%\nArduino:" +
arduino_status+"\nOpenCPN:" + opencpn_status
+"\nBarco_Lat:"+boat_lat.value+"\nBarco_Long:"+boat_long.value
+"\nBarco_SOG:"+boat_sog.value +"\nBarco_rm:"+boat_rm.value
+"\nBarco_rv:"+boat_rv.value+"\nWay_Lat:"+way_lat.value
+"\nWay_Long:"+way_long.value +"\nWay_Dist:"+way_distance.value
+"\nWay_rv:"+way_rv.value +"\nWay_Dest:"+way_destino.value+"\n\n"
    file = open("log/"+file_stamp+"_VNAS_dump.txt","a")
    file.write("\n")
    file.write(time_stamp+"\n"+dump)
    file.close()
```

3.7. Errores y Alertas

Cada una de las distintas funciones tiene incorporada su propia gestión de errores. Usando la sintaxis try estratégicamente colocada en cada función, recogemos las excepciones y procedemos a indicar o solucionar los posibles errores y alertas que se generen.

Como uno de los ejemplos vamos a recurrir a la asignación del valor recibido por serial de la batería a su apartado en la interfaz. El try intenta realiza la asignación del valor en la interfaz, y modifica el color del fondo según los parámetros de los ajustes. La única parte del código que puede fallar es la conversión a coma flotante de los parámetros si hubiera caracteres especiales y no solo números o letras, si fallara, el except que recoge el error dejaría el valor de la batería vacío como “NaN” y el fondo gris, hasta el próximo valor que se reciba. En este caso puede ser el valor recibido o el valor del ajuste. Aunque este último también tiene otra doble comprobación para evitarlo a la hora de cargar o guardar los datos.

```
    try:
        if serial_battery!= "NaN":
            bat_estado.value = "Bateria: " + serial_battery +"%"
            if float(serial_battery) >= float(llena.value):
                bat_estado.bg = "pale green"
            elif (float(serial_battery) < float(llena.value)) and
(float(serial_battery) >= float(baja.value)):
                bat_estado.bg = "yellow"
            elif (float(serial_battery) < float(baja.value)) and
(float(serial_battery) >= float(critica.value)):
                bat_estado.bg = "orange"
            elif float(serial_battery) < float(critica.value):
                bat_estado.bg = "red"
        else:
            bat_estado.value = "Bateria: NaN%"
            bat_estado.bg = "grey"
    except:
        bat_estado.value = "Bateria: NaN%"
        bat_estado.bg = "grey"
```

En la función de comunicación UDP, desplazamos todo el peso de la excepción a que la variable de datos esté vacía o no. Nunca estará vacía si ha recibido datos, ya que la función de UDP no devuelve una cadena vacía si recibe un espacio en blanco (`b''`). Y a partir de aquí se hace la gestión de errores.

```
    try:
        data, address = sock.recvfrom(200)
    except:
        data=""
```

En cuanto a los estados, las funciones de Arduino y UDP también incluyen sus indicadores de estado, como `opencpn_status` (ligado a UDP), según los datos que recibe

o no recibe varía su estado en cada lectura. Y podemos realizar decisiones en el código de control según el valor de los distintos estados. Lo que nos da un control total sobre las distintas situaciones que puedan darse.

```
        if boat_data!=0 and (time.time()-boat_data)>2 and
(opencpn_status=="ON"):
            opencpn_status = "ERR"
            log_estado("[!!] OpenCPN no envia datos de GPS. Verifique
conexión física", "orange red")
        elif opencpn_status!= "ERR" and opencpn_status!= "OFF" and
boat_data==0:
            boat_data=time.time()
        if waypoint_data!=0 and (time.time()-waypoint_data)>2 and
opencpn_status=="ON":
            opencpn_status = "WAIT"
            log_estado("[Δ] OpenCPN no tiene una ruta activa", "orange")
        elif opencpn_status!= "OFF" and opencpn_status!= "WAIT" and
waypoint_data==0:
            waypoint_data=time.time()
        if opencpn_status== "NaN" or opencpn_status== "OFF":
            opencpn_status="ON"
            log_estado("[✓] OpenCPN está operativo", "light blue")
```

GUIzero proporciona avisos o pop-ups / ventanas emergentes que están usadas ampliamente en la interfaz para confirmar acciones y también para avisar de errores y alertas.

3.8. Lazo de Control

El lazo de control se ha diseñado para que sea una función a la que se llama cada segundo, en la cual, con la telemetría y los valores en ese momento, se recalculan los parámetros de control de los motores. Para ello nos valemos de las coordenadas de posición del barco y del waypoint, así como la marcación del waypoint, y la orientación del barco, ambas con el norte verdadero.

La marcación es el ángulo que existe desde la visual del barco hasta el punto en cuestión en referencia con el norte verdadero. El rumbo es el ángulo que existe desde el norte verdadero hasta la dirección de avance del barco, nosotros tomaremos el rumbo verdadero o de proa como acepción a rumbo. Mas allá de estos valores solo necesitaremos la velocidad sobre la superficie. Aunque este valor se puede deducir con el cambio de las coordenadas del barco por segundo.

OpenCPN nos proporciona la marcación del waypoint, pero deja de mandarlo una vez que se ha llegado al destino. Si queremos anclarnos en esa posición o cualquier otra en cualquier momento, debemos ser capaces de calcular la marcación de este punto de anclaje, y compararlo con el rumbo del barco.

Para ello recurrimos a la librería geograpichLib, en su apartado Geodesic, el cual nos proporciona las funciones necesarias para realizar cálculos geodésicos. La geodesia es la ciencia que abarca la representación de la forma y de la superficie de la Tierra. La tierra no tiene una forma de canica esférica perfecta, si no que está achatada y representa muchas diferencias de elevación a lo largo de su superficie, no siendo uniforme. Es por eso que se ha ido adoptando un sistema geodésico de coordenadas geográficas mundialmente según se han ido desarrollando nuevos avances en la medición y posicionamiento (GPS). El último en adoptarse fue en 1984, y se denominó WGS84 (World Geodetic System 1984), el cual aporta un error de cálculo menor a 2 cm. En la Figura 46 se muestran sus rasgos generales.

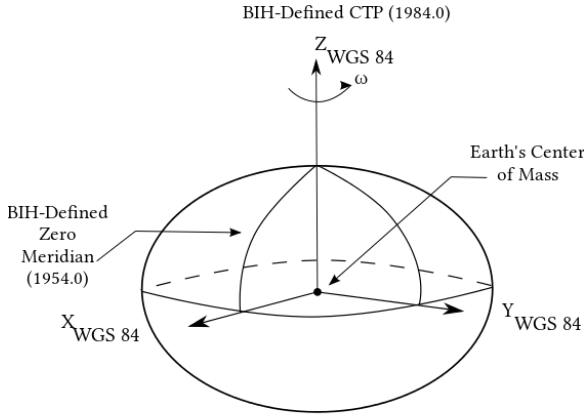


Figure 1.1 WGS 84 Reference Frame

Figura 46 - Modelo WGS84

La librería geograpichLib, nos permite realizar cálculos directos e inversos con coordenadas en grados decimales del SI, y obtener distancia entre puntos, marcación, rumbo, etc. Nosotros solo usaremos la función inversa para determinar la marcación y la distancia del waypoint.

```
marca_w = Geodesic.WGS84.Inverse(boat_latitud, boat_longitud, way_latitud, way_longitud)['azi1']

distancia_geo = Geodesic.WGS84.Inverse(boat_latitud, boat_longitud, way_latitud, way_longitud)['s12']
```

Estas funciones, junto con el rumbo obtenido de OpenCPN nos sirven para controlar nuestro barco. Se podría decir que son los datos mínimos y básicos para poder realizar un control adecuado.

En el código realizamos comprobaciones de estado y datos antes de proceder a la actualización de la palabra de control.

```
#########################################################################
#################       FUNCION DE PILOTO AUTOMÁTICO       ##############
##                      Código de control de navegación                ##
#########################################################################
#########################################################################

def control_codigo():
    #Saco las variables globales run y data, tengo que declararlas como tal en la funcion
    global run
    global data
    global data_try
    global auto
    global Primera_vez
    global arduino
    global control
    global inc_velocidad
    global velocidad
    global arrival
    global MotorD
    global MotorI
    global cambio

    last_arrival=False

    #Gestión de estados para modificar la cadena de control, solo deja avanzar si se cumplen las condiciones
    if run==True and auto==True:
        if Primera_vez==True:
            if arduino_status=="ON":
                if opencpn_status=="ON":
                    Primera_vez=False
                    log_estado("[✓] Navegación iniciada", "light blue")
                elif opencpn_status =="ERR":
                    run=False
                    app.error("Oh!", "OpenCPN sin datos de GPS. Imposible iniciar")
                    return
```

```
                else:
                    run=False
                    app.error("Oh!", "OpenCPN desconectado. Imposible
iniciar")
                    return
            else:
                run=False
                app.error("Oh!", "Arduino Micro no disponible, espera a que
esté disponible o comprueba su conexión")
                return

        else:
            if arduino_status=="ON":
                if opencpn_status=="ON":
                    arrival = False
                elif opencpn_status=="OFF":
                    run=False
                    control="0511.0511\n"
                    log_estado("[!!] Error de conexión con OpenCPN.
Deteniendo la navegación", "orange red")
                    app.error("Oh!", "Error de conexión con OpenCPN.
Deteniendo la navegación")
                    return
                elif opencpn_status=="ERR":
                    run=False
                    control="0511.0511\n"
                    log_estado("[!!] Error de GPS. Deteniendo la navegación",
"orange red")
                    app.error("Oh!", "Error de GPS. Deteniendo la
navegación")
                    return
                elif opencpn_status=="WAIT":
                    if arrival == False:
                        Primera_vez=False
                        arrival = True
                        log_estado("[Δ] Ruta finalizada, manteniendo
posición", "orange")
            else:
                run=False
                control="0511.0511\n"
                log_estado("[!!] Arduino desconectado durante la travesía.
Pasando a modo manual", "orange red")
                app.error("Oh!", "Arduino desconectado durante la travesía.
Pasando a modo manual")
                return

    elif run==True and auto==False:
        run=False
        control="0511.0511\n"
        log_estado("[Δ] Se ha pasado a modo manual. Detenida la navegación.",
"orange")
        app.error("Oh!", "Se ha pasado a modo manual. Detenida la
navegación")
        return
    else:
        run=False
        velocidad=0
        control="0511.0511\n"
        return
```

```
    if (boat_lat.value=="NaN") or (boat_long.value=="NaN") or
(boat_sog.value=="NaN") or (boat_rm.value=="NaN") or (way_lat.value=="NaN")
or (way_long.value=="NaN"):
        run=False
        velocidad=0
        control="0511.0511\n"
        info("Error", "OpenCPN no proporciona todos los datos, comprueba que
recibe señal o que hay un waypoint activo. Navegación detenida")
        app.error("Oh!", "[!!] OpenCPN no proporciona todos los datos,
comprueba que recibe señal o que hay un waypoint activo")
```

A partir de aquí, recuperamos las variables que contienen los datos de posición y los convertimos a grados decimales.

```
    #Guardo las variables en coma flotante (están como cadena de texto por
defecto)
    heading_waypoint=float(RMB[11]) #Rumbo waypoint OpenCPN
    heading_boat=float(HDT[1])      #Rumbo barco

    #Coordenadas Barco y Waypoint
    boat_latitud = round(float(RMC[3][0:2]),7)+ round(float(RMC[3][2:])/60,7)
#Latitud barco
    if RMC[4] =="S": boat_latitud=-boat_latitud
    boat_longitud = round(float(RMC[5][0:3]),7)+
round(float(RMC[5][3:])/60,7) #Longitud barco
    if RMC[6] =="W": boat_longitud=-boat_longitud
    way_latitud = round(float(RMB[6][0:2]),7)+ round(float(RMB[6][2:])/60,7)
#Latitud waypoint
    if RMB[7] =="S": way_latitud=-way_latitud #Check signo
    way_longitud = round(float(RMB[8][0:3]),7)+ round(float(RMB[8][3:])/60,7)
#Longitud waypoint
    if RMB[9] =="W": way_longitud=-way_longitud #Check signo
```

Realizamos las operaciones inversas para la marcación y la distancia. Y además declaramos el PID de control usando la librería simplepid, esta librería nos libera de tener que scriptar un PID, y directamente lo crea con los parámetros que le pasamos a la función, incluyendo el setpoint deseado. Vamos a establecer cero, como el setpoint deseado, para que cualquier desviación sea positiva para un sentido, y negativa para el otro, simplificando la ejecución del lazo de control.

```
    #Bearing del waypoint calculado a través de las coordenadas del barco
    heading_w = Geodesic.WGS84.Inverse(boat_latitud, boat_longitud,
way_latitud, way_longitud)['azi1']

    distancia_geo = Geodesic.WGS84.Inverse(boat_latitud, boat_longitud,
way_latitud, way_longitud)['s12']

    sog=float(RMC[7])               #Velocidad sobre el agua
    distan_nm=float(RMB[10])        #Distancia hasta el destino
    #arrival=RMB[13]                 #Flag de llegada al destino
    pid1 = PID(float(input_P.value), float(input_I.value),
float(input_D.value), setpoint=0)#Configuración del PID de control fino
    pid2 = PID(1, 0.1, 0, setpoint=0)#Configuración del PID de control fino
```

Los ángulos que obtenemos de rumbo y marcación siempre están referenciados al norte como 0º, hasta un valor máximo de 360º, donde el valor vuelve a ser 0º. Si queremos que nuestro PID siga un setpoint de 0º, es decir que sea 0º cuando el rumbo del barco, y la marcación del barco sean iguales. El problema viene de que este setpoint deseado es absoluto al norte y no relativo a la orientación/rumbo del barco, y no se puede obtener una formula directa con los valores de rumbo y marcación al estar limitados a 360º como una vuelta completa, parecido a lo que se muestra en la Figura 47. Para ello vamos a obtener la diferencia entre el rumbo del barco y la marcación del waypoint, y obtenemos el módulo de 360, esto lo que hace es referenciar relativamente al rumbo del barco cualquier diferencia entre el rumbo y la marcación. Al hacer matemáticamente controlable el rumbo, queremos que 0º sea el centro, no el origen, por lo que el origen y final de los grados sería de -180º a 180º (360º), restándole 360 º al resultado si supera los 180 º.

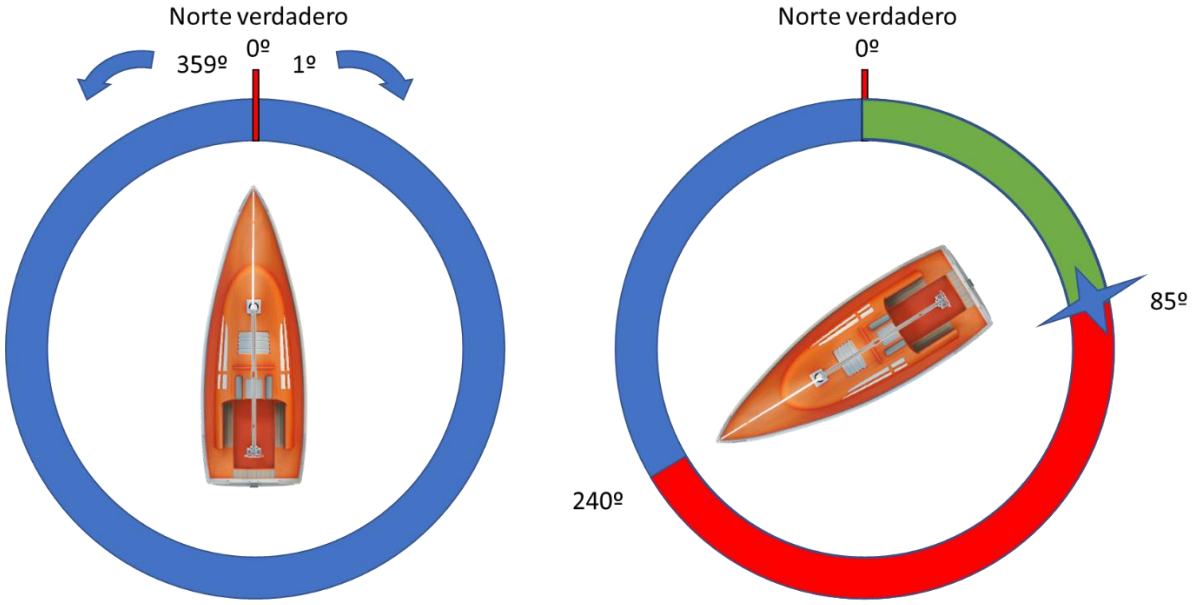


Figura 47 - Funcionamiento del rumbo y la marcación

Este método se denomina wrapping.

```
    #Uso el metodo denominado wrapping para poder hacer relativa la
diferencia de rumbos
    correction=(heading_boat-heading_w)%360 #Calculo el ángulo relativo entre
ambos y me quedo con el módulo de 360°
    if correction > 180:
        correction -=360
    #Y con esto último lo convierto en un valor matemático perfectamente
PDIzable de-180 a 180, siendo siempre 0 el rumbo a seguir.

    #Distancia de Llegada
    if distancia_geo <=float(arrival_dist.value):
        last_arrival=True
    else:
        last_arrival=False


    if last_arrival==True or arrival==True:
```

```
        #Frenamos
        if sog > 0.1:#Si la velocidad no es nula
            if sog > sog_d:#Si la velocidad actual es mayor que la velocidad
anterior, hay que invertir el sentido
                cambio*= -1
            MotorI=MotorI-cambio
            MotorD=MotorD-cambio

            if MotorI>1023:
                MotorI=1023
            if MotorI<0:
                MotorI=0
            if MotorD>1023:
                MotorD=1023
            if MotorD<0:
                MotorD=0


        else:#Si la velocidad es nula
            MotorD=511
            MotorI=511
        sog_d=sog

    #Lazo Normal
    else:
        #IMPLEMENTACIÓN PID (recomiendo un 80% de velocidad para empezar)
        #511 es parada
        #de 0 a 510 es marcha atrás (potencia inversa)
        # de 512 a 1023 es avance
        #MotorI=511
        #MotorD=511
        #Velocidad incremental
        vel_max=round(float(speed_max.value)/100,2)
        if velocidad < vel_max:
            velocidad+=round(float(speed_incr.value)/100,2)

            print(round(float(speed_incr.value)/100,2))
        if velocidad>vel_max:
            velocidad=vel_max
```

Con el método de wrapping obtenemos un valor que nos indica si hay una desviación hacia dirección u otra, y esa desviación al pasarla por el PID, podemos controlar la salida del motor. Hemos denominado inter (interferencia) a la salida del PID, según si es positivo o negativo el valor (corrección de rumbo a la derecha o a la izquierda respectivamente), se aplica en un motor u otro la corrección.

Arduino funciona en una escala de 1024 valores Analógico a Digital (0 a 1023) en la cual considera la mitad de la escala, 511, como parada del motor, 0 como reversa completa y 1023 como avante completa. Se ha decidido que la interferencia en la velocidad de los motores no puede ser mayor de 512, para que los giros cuando el PID sature, se hagan con un solo motor, mientras el otro está en parada, y así evitar que uno esté en avante y el otro en reversa. Es por ello que para limitar los valores de inter, se ha usado la función clip de la librería numpy. Al valor de parada de 511 se le suma o resta el valor de la corrección aplicada, multiplicada por el límite de software de la velocidad. Y el valor de

los motores se asigna a la palabra de control, que la función de comunicación serial se encarga de transmitir a los motores.

```
        inter=pid1(correction)
        if inter>0:
            inter=np.clip(inter,0,512)
            MotorI=511+512*velocidad
            MotorD=511+(float(512)-inter)*velocidad
        elif inter<0:
            inter=np.clip(inter,-512,0)
            MotorI=511+(float(512)+inter)*velocidad
            MotorD=511+512*velocidad
        else:
            MotorI=511+512*velocidad
            MotorD=511+512*velocidad


    control=str(int(MotorI))+"."+str(int(MotorD))
```

La velocidad también se ve limitada por software como límite de seguridad. Según el parámetro de los ajustes se puede limitar en un tanto por ciento respecto de su valor máximo. La velocidad siempre empieza en 0%, y se va incrementando según el incremento que esté designado en los ajustes hasta que se alcanza el límite.

```
        vel_max=round(float(speed_max.value)/100,2)
        if velocidad < vel_max:
            velocidad+=round(float(speed_incr.value)/100,2)

            print(round(float(speed_incr.value)/100,2))
        if velocidad>vel_max:
            velocidad=vel_max
```

4. Resultados y conclusiones

Este TFG se ha realizado a lo largo de 10 meses, aunque el tiemplo planificado era menor, pero debido a la irrupción del SARS-CoV-2 las pruebas de campo no se han podido realizar ya que otros TFGs relacionados no han podido llegar a completarse. Sin embargo, las pruebas en laboratorio han sido completamente satisfactorias, consiguiéndose todos los objetivos enmarcados en este TFG.

No se disponen de las baterías para las pruebas en laboratorio, por lo que se ha alimentado el bus CAN al que se conectan los sensores mediante una fuente regulable de corriente continua que proporcionó la universidad.

A pesar de esta y otras limitaciones, se ha podido completar exitosamente un piloto automático funcional para embarcaciones impulsadas por dos motores de tipo eléctrico. Las conclusiones se desgranan aquí:

Piloto automático:

- Se han realizado multitud de ensayos de rumbo y control de potencia de los motores. Y con ello se ha configurado un valor PID aproximado para el comportamiento del catamarán. Sin embargo, el código dispone de la capacidad para calcular las constantes de giro para reconfigurar el PID durante la primera prueba de campo real, y es apto para usarlo en cualquier embarcación sin importar dimensiones o morfología del casco.
- El código desarrollado contempla todas las excepciones en su ejecución y acciones supresivas para cada caso. Esto hace que el código sea más robusto y resistente frente a errores externos como valores erróneos de sensores, desconexión de los mismos o fallo de componentes como Arduino.
- La programación se puede ampliar e incluir mas configuraciones, ventanas, señales y complejidad según se vaya ampliando el diseño del proyecto VNAS, se han hecho pruebas para comprobar el correcto funcionamiento durante largos periodos de tiempo, y durante una semana con el piloto automático en ejecución se han ido forzando excepciones y errores en el hardware para comprobar el correcto funcionamiento y que la gestión de la memoria era correcta, obteniendo un resultado perfecto, con cero problemas de ejecución en todas las áreas.

Raspberry Pi y Python:

- Respecto al diseño original el soporte de hardware ha sido mejorado fehacientemente. En estos 10 meses no se ha producido ningún error de almacenamiento ni corrupción de datos, y el rendimiento se ha visto mejorado como interfaz de usuario, eliminando largas esperas y cuelgues temporales en comparación con el uso del soporte microSD.
- La integración de Openplotter con el resto de componentes es perfecta, incluso más ahora que OpenCPN pasa a tener compatibilidad con Signal K a partir de su versión 5.2. Sin embargo, se ha detectado un pequeño incremento en la carga del procesador si se alimentan los datos directamente del servidor Signal K, al estar

todavía en fase de pruebas, y es por eso que se recomienda usar la conversión integrada a NMEA0183 hasta que salga la siguiente revisión, que es la que tiene programada el piloto automático.

- El entorno de RPi es user-friendly y permite desarrollar fácilmente este tipo de aplicaciones, se poseen de las herramientas adecuadas y la colaboración de la comunidad detrás de todo el soporte online que existe. Ninguno de los problemas que se han encontrado durante el proyecto ha sido insalvable y se ha encontrado una solución a todos los planteamientos iniciales.

Sensores:

- El bus CAN ha funcionado correctamente, y la puerta de enlace ha sido una apuesta correcta en este desarrollo.
- El posicionador GPS requiere de espacios abiertos donde poder coger señal, es especialmente débil en cualquier tipo de entorno cerrado, cosa que no se da en el mar.

Los experimentos llevados a cabo son:

- Prueba de transmisión de datos: Implementado un código sencillo que comprueba que las cadenas enviadas por Serial y por UDP no se han visto afectadas durante la transmisión. Ejecución a lo largo de una semana, ninguno aviso se activó.

- Prueba de carga de memoria y robustez: Durante una semana se mantuvo el Piloto Automático en funcionamiento continuo, forzando puntualmente excepciones de Serial y telemetría. (La telemetría se falseó para no mantener la fuente de alimentación activa 7 días). No se vio afectada la carga en RAM, ni errores en las instrucciones.

- Pruebas manuales de corrección de rumbo: Se ha probado reiteradamente la asignación de los motores respecto a variaciones de marcación del waypoint y variación manual del rumbo del barco, funcionando exitosamente en todas y cada una de las pruebas.

5. Líneas futuras

El grueso de este proyecto se ha visto desarrollado en la figura del piloto automático, pero hay líneas de mejora que se puede llevar a cabo en el ámbito del proyecto VNAS.

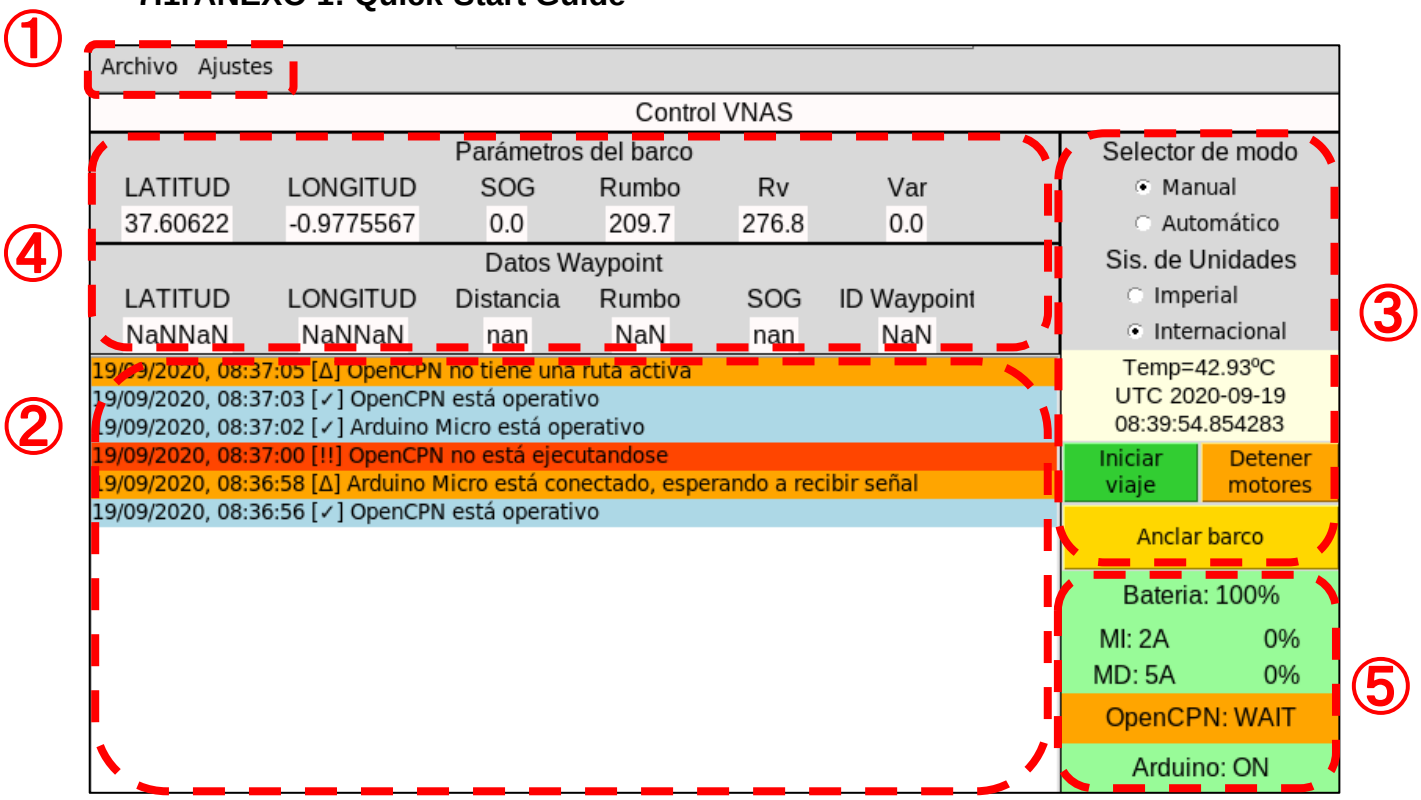
- **Adaptación del Piloto automático a Signal K** y eliminación de NMEA0183. A partir de la versión 5.2, OpenCPN ya es compatible con Signal K, pero las coordenadas que envía, si recibe directamente de Signal K sin el plugin de NMEA0183, son también en Signal K, por lo que solo hay que cambiar el formato de datos en la función UDP. Lo que nos dará mayor precisión a la hora de calcular las coordenadas (1 decimal más)
- **Desarrollo propio para crear sensores basados en Signal K**. Usando Arduino Nano o Raspberry Pi Zero se pueden crear módulos totalmente funcionales e independientes de GPS, Compás, Veletas, SOG, etc, para conectar a cualquier dispositivo por bus CAN, USB, Bluetooth y/o Wifi. y que podrían comercializarse.
- **Lazo de control en un modelo Simulink** con doble RPi, esta mejora se basa en instalar el modelo físico de Simulink desarrollado en uno de los TFG enmarcados en el proyecto VNAS, en una RPi, y otra RPi se encarga de la gestión de señales y transmisión de datos a tierra los motores y a la RPi con Simulink. Matlab por el momento no permite ejecución separada de su entorno de simulación para RPi.
- **Ampliación de la conectividad en tierra** del piloto automático. Con el TFG de Telecomunicaciones completado, se puede hacer que OpenCPN y el piloto automático funcionen en cualquier dispositivo y tomen control remoto del barco mediante un modo bypass programado en el piloto automático.
- **Sistema de redundancia y caja negra**. Aunque su utilidad puede ser cuestionada, de cara a la competición trasatlántica puede llegar a ser de un alto valor la capacidad de instalar una baliza de emergencia en el barco de diseño propio, para incrementar las posibilidades de recuperar la embarcación ante posibles eventos catastróficos. También la instalación de un sistema de discos redundantes para duplicar la SSD y obtener los últimos parámetros del barco.
- **Instalación y navegación mediante AIS (Automatic Identification System)**. Instalar el sistema de radiofrecuencia UHF para detectar barcos y a la vez emitir nuestra identificación y recalcular rutas es un beneficio añadido que podría hacer que otras embarcaciones no chocaran con nuestro barco avisando de nuestra localización. Añadido al sistema de SOS.
- **Visión artificial**. En el caso de que no esté un sistema AIS instalado por el posible incremento en el consumo energético, también se pueden instalar cámaras para realizar un análisis mediante OpenCV en RPi de las condiciones que rodean al barco y tomar acciones si se detectan peligros en el horizonte.
- **IA**. Aunque el piloto automático cumple con su tarea de realizar rutas, requiere de intervención humana cuando hay un fallo grave como la pérdida de conexión con los motores o un cuelgue de alguno de los componentes software, algo que una programación inteligente podría salvar, reanudando rutas cuando las condiciones son favorables o informando al control de tierra de la situación.

6. Bibliografía

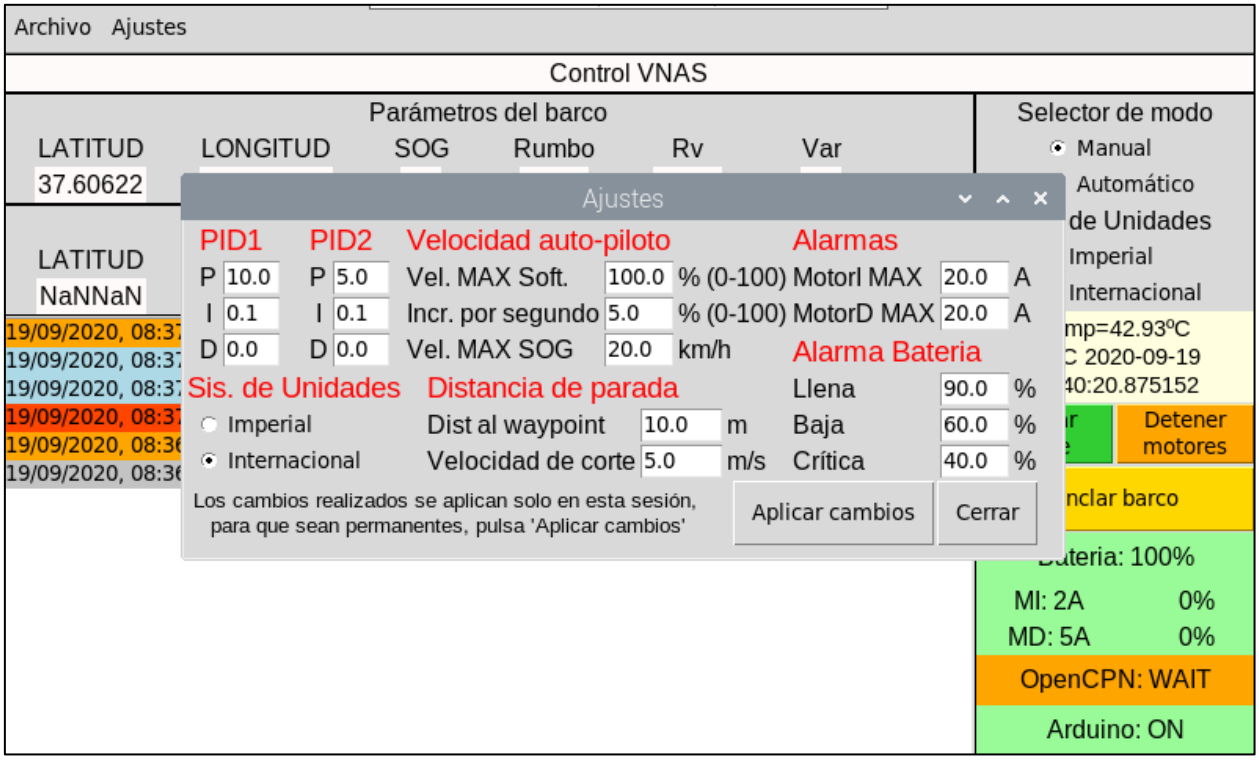
1. **Rhetassi, Hassan Bahari.** Desarrollo de una aplicación web de telemetría para el control de un barco autónomo. [En línea] https://repositorio.upct.es/bitstream/handle/10317/7966/tfg-bah-des.pdf?sequence=1&isAllowed=y.
2. **The Microtransat Challenge.** The Microtransat Challenge. [En línea] https://www.microtransat.org/index.php.
3. **Redel, Alejandro González.** Diseño e implementación de un sistema de radiocomunicaciones. [En línea] https://repositorio.upct.es/xmlui/bitstream/handle/10317/7568/tfg-gon-dis.pdf?sequence=1&isAllowed=y.
4. **Martínez, Daniel Martínez.** Desarrollo de un sistema de control para la navegación de un barco autónomo. [En línea] https://repositorio.upct.es/bitstream/handle/10317/7898/tfg-mar-des.pdf?sequence=1&isAllowed=y.
5. **SAE International.** Automated Driving - Levels of Driving Automation J3016. [En línea] 2014. https://web.archive.org/web/20180701034327/https://cdn.oemoffhighway.com/files/base/acbm/ooh/document/2016/03/automated_driving.pdf.
6. **IBM.** IBM Newsroom. [En línea] https://newsroom.ibm.com/2020-09-15-Mayflower-Autonomous-Ship-Launches#assets_all.
7. **Halfacree, Gareth.** Raspberry Pi 3 B+ (39906369025). [En línea] https://commons.wikimedia.org/wiki/File:Raspberry_Pi_3_B%2B_(39906369025).png.
8. **Kingston.** Industrial UHSI. [En línea] https://www.kingston.com/spain/es/memory-cards/industrial-temperature-microsd-uhs-i .
9. **Afrank99.** MicroSD card 2GB focus-stacked.jpg. [En línea] https://commons.wikimedia.org/wiki/File:MicroSD_card_2GB_focus-stacked.jpg .
10. **Adafruit.** Adafruit 5" and 7" 800x480 TFT HDMI Backpack. [En línea] https://learn.adafruit.com/adafruit-5-800x480-tft-hdmi-monitor-touchscreen-backpack/raspberry-pi-config.
11. **OpenCPN.** Guía de Usuario de OpenCPN. [En línea] https://opencpn.org/wiki/dokuwiki/doku.php?id=opencpn:opencpn_user_manual.
12. **Verruijt, Kees.** Github - CANBOAT. [En línea] https://github.com/canboat/canboat.
13. **Signal K.** Especificaciones de Signal K. [En línea] https://signalk.org/specification/1.4.0/doc/.
14. **Bender, Scott.** canboatjs. [En línea] https://github.com/canboat/canboatjs.
15. **Kurki, Teppo.** signalk-to-nmea0183. [En línea] https://github.com/SignalK/signalk-to-nmea0183.
16. **Openplotter.** Guía de usuario de OpenPlotter. [En línea] https://openplotter.readthedocs.io/en/latest/getting_started/installing.html.
17. **StackOverflow.** Developer Survey Results. [En línea] 2019. https://insights.stackoverflow.com/survey/2019.
18. **Pierre Carbonnelle.** PYPL PopularitY of Programming Language. [En línea] http://pypl.github.io/PYPL.html.
19. **Peters, Tim.** El Zen de Python. [En línea] https://es.wikipedia.org/wiki/Zen_de_Python.
20. **Sach, Laura.** GUIzero. [En línea] https://lawsie.github.io/guizero/start/.
21. **Python.** subprocess — Subprocess management. [En línea] https://docs.python.org/3/library/subprocess.html.
22. —. select — Waiting for I/O completion. [En línea] https://docs.python.org/2/library/select.html.
23. **Desconocido.** Arduino Logo.svg. [En línea] https://es.wikipedia.org/wiki/Archivo:Arduino_Logo.svg.
24. **Liechti, Chris.** pySerial. [En línea] https://pyserial.readthedocs.io/en/latest/pyserial_api.html.
25. **Python.** Socket. [En línea] https://docs.python.org/3/library/socket.html.
26. **Hartley, Jonathan.** Regex CheatSheet. [En línea] https://github.com/tartley/python-regex-cheatsheet/blob/master/cheatsheet.rst.

7. Anexos

7.1. ANEXO 1: Quick-Start Guide



① **Menú** – Acceso a la ventana de ajustes, modo pantalla completa, y salir.
② **Telemetría** – Se muestran los valores obtenidos de los sensores del barco y la ruta:
   a. Cuando no hay datos aparece “NaN”. En el caso de waypoint, se mantiene para usarlo como punto de anclaje.
③ **Eventos** – Listado de eventos en la sesión actual:
   a. Aquí se muestran todos los sucesos, la mayoría auto explicativos, cada uno de ellos se vuelca con todos los datos de estado en el ‘logbook’ que se encuentra en el escritorio del sistema.
④ **Controles** – Botones para el control de la navegación:
   a. Iniciar Viaje: Si se cumplen las condiciones, inicia la función de control de los motores para seguir la ruta marcada en OpenCPN.
   b. Detener Motores: Los motores se detienen, pero no realiza el frenado del barco.
   c. Anclar Barco: Frena el barco y mantiene la posición, activándose los motores cuando se distancia más de lo asignado al punto de anclaje cuando se activó el modo.
⑤ **Estado** – Datos de consumo y estado de los sistemas:
   a. Muestra los datos recibidos vía Serie de Arduino; nivel de batería y consumo de los motores. El porcentaje de potencia viene referenciado por el lazo de control.
   b. Estado de OpenCPN y Arduino
      i. ON: Funciona correctamente
      ii. OFF/PLUG: Desconectado físicamente u apagado
      iii. ERR: Se ha producido un error
      iv. WAIT: Esperando a recibir señal



En la ventana de ajustes. Los valores que se dan por defecto se muestran en la imagen.

① **PID1** – Valores para el controlador PID de seguimiento del rumbo.

② **Velocidad auto-piloto:**

a. Velocidad Máxima por Software: límite de velocidad máxima proporcional
b. Incremento proporcional por segundo: Valor del incremento lineal en la velocidad de los motores
c. Velocidad Máxima Real: Limita los motores al alcanzar una velocidad máxima en desplazamiento superficial.

③ **Sistema de Unidades** – Sistema por defecto en el que se muestran los valores al iniciar el VNAS.

④ **Distancia de parada:**

a. Distancia al waypoint: distancia al waypoint en la que empezará a decelerar, también se considera como el radio máximo al waypoint en el que volverá a activar los motores para recolocarse en el modo de anclaje.
b. Velocidad de corte: Velocidad máxima en la que se considera que el barco se ha detenido completamente. Ya que 0 m/s puede ser un valor difícil de obtener, excepto en tierra.

⑤ **Alarmas:**

a. Motores: Intensidades registradas a las que se alertará visualmente del consumo excesivo de los motores.
b. Batería: Porcentajes ajustables para considerar el estado de carga de las baterías.

Cualquier cambio en los campos se mantiene durante la sesión activa. Cuando se aplican los cambios, se hacen permanentes para el próximo inicio del VNAS.

Para iniciar un viaje

Pasos:

1- Antes de encender el sistema RPi hay que asegurase de:
   a. Actisense NGT-1 está conectado a un puerto USB
   b. Arduino está conectado a un puerto USB
   c. SSD está conectado a un puerto USB
2- Cuando RPi se inicia, el Starter lanza automáticamente a los 20 segundos OpenCPN, 15 segundos después se inicia VNAS. No los ejecutes manualmente para evitar duplicaciones y errores por bloqueo de puertos o sobrecarga.
3- Crea y activa una ruta en OpenCPN
4- VNAS recibirá la telemetría y se pondrán en verde todos los indicadores.
5- Pulsar el botón verde “Iniciar viaje” y confirmar la acción, el barco empezará a arrancar los motores y a posicionarse.

Para detener un viaje

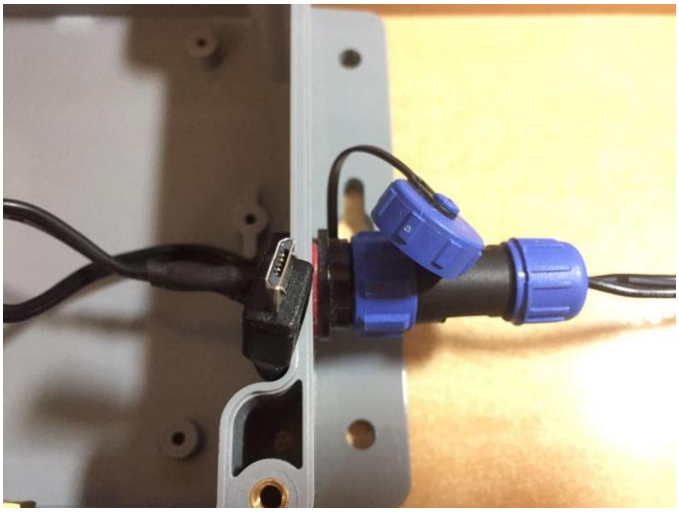
A modo de parada de emergencia, pulsar en el botón “Detener motores”. Lo motores se detendrán y el barco seguirá con su inercia o las corrientes.

Si se desea parar el barco debido a su velocidad en ese momento, pulsar “Anclar barco”, y cuando se haya reducido la velocidad al nivel deseado, pulsar “Detener motores”

7.2. ANEXO 2: Ampliación del Quick-Start (Guía de usuario)

El piloto automático VNAS se ejecuta al iniciarse la RPi junto con OpenCPN a través de un programa llamado Starter. Ambos archivos Python se encuentran en el escritorio para su acceso fácil y rápido para cualquier modificación necesaria. Si se elimina, modifican, mueven de localización o renombran, dejarán de funcionar y habrá que ejecutar entonces manualmente OpenCPN y el VNAS

RPi necesita una alimentación de 5v con un mínimo de 2A. En el cuadro eléctrico del barco hay un conector de 5 voltios al que se conecta por USB la caja del piloto automático y que alimenta la RPi.



Antes de iniciar la RPi hay que asegurarse bien de (check list):

- ☐ La interfaz Actisense NGT-1 está conectada a un puerto USB aleatorio de la RPi y al backbone N2K. No es un dispositivo plug & play, si no está conectada en el momento del inicio de la RPi, habrá que reiniciar el sistema RPi para que lo reconozca. Eso sí, una vez iniciado correctamente, puede conectarse y desconectarse o dejar de alimentar el backbone sin problema, que lo seguirá reconociendo.
- ☐ Arduino está conectado a un puerto USB aleatorio de la RPi. Raspbian tiene cierta animadversión al control Serial de Arduino y tarda aproximadamente unos 20 o 25 segundos en dejar sincronizado y disponible el puerto tras conectarlo físicamente. Mientras no se desconecte físicamente en ningún punto la conexión, se puede reiniciar Arduino sin ningún problema de esperas.
- ☐ El disco duro SSD está conectado a cualquier puerto USB de la RPi. El SO está instalado en este SSD y no dispone tarjeta microSD de arranque alternativa.
- ☐ Si no se va a utilizar físicamente la pantalla táctil, y solo se va a acceder de forma remota, desconectar pantalla y teclado. Ya que estos se calientan y consumen bastante energía.

Acceso In situ – Sin pantalla táctil

Si no se desea usar la pantalla táctil y portas un móvil o Tablet, puedes conectarte directamente con la red wifi propia que está configurada en la RPi a través de VNC con conexión encriptada.

Los datos de acceso son:

SSID: VNAS
Pass: politecnica
(Canal 6)

En tu cliente VNC la IP de acceso al sistema es siempre 10.10.10.1, las claves son:

User: pi
Pass: raspberry

Todas las claves y usuarios se pueden modificar. Si se modifican dejad constancia para que no se pierdan los datos de acceso. En tal caso acceder con físicamente a la RPi (pantalla y teclado) y reconfigurar los datos de autentificación.

Para acceder remotamente a través de 3G, hay que asegurarse de la IP pública o instalar un programa que use un dominio y la RPi actualice el dominio con su IP.

Acceso In situ – Con pantalla táctil

La pantalla táctil de Adafruit necesita alimentación de un puerto USB (también se comunica por USB para actuar como puntero) y conectarse por HDMI. Es plug & play y puede conectarse en cualquier momento, no es necesario tenerla conectada al iniciar el sistema RPi. Si se quiere usar el teclado inalámbrico hay que desconectar el USB de Arduino, a no ser que se use un concentrador USB y se conectan pantalla y teclado en el mismo.

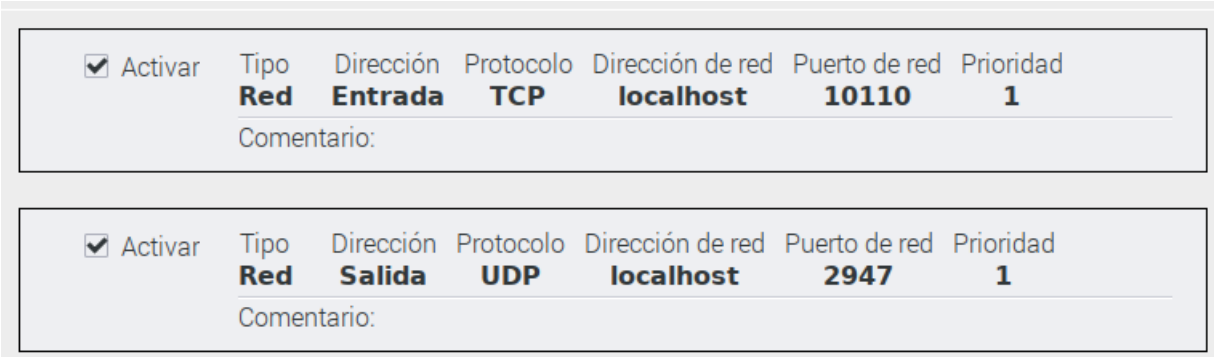
Reconfiguración Pantalla táctil

La pantalla táctil solo funciona a la resolución de 800x480 píxeles. No incluye ningún controlador que redimensione la imagen, por lo que si se modifica la resolución hay que reescribir el archivo /boot/config.txt para establecer esta configuración.

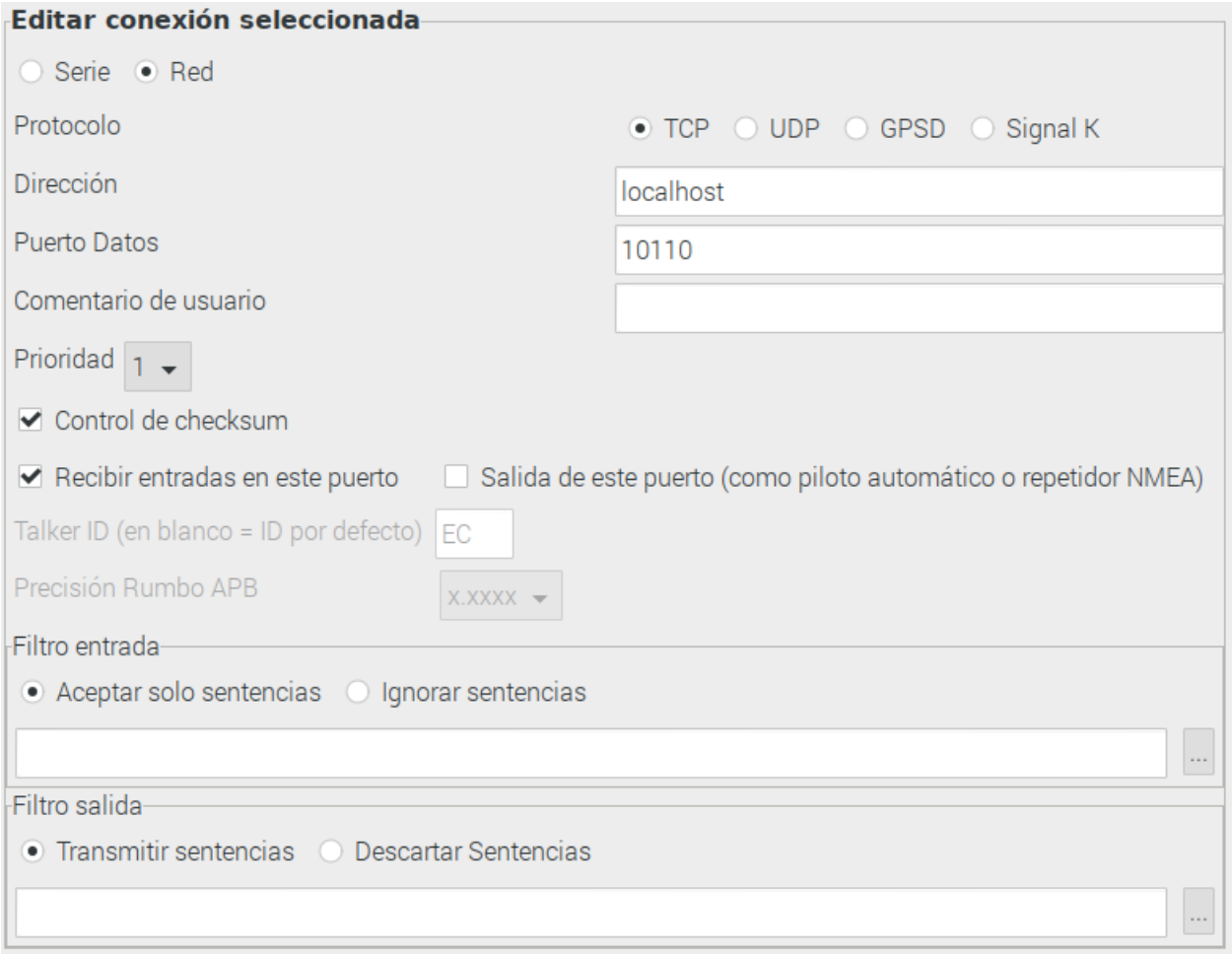
```
# uncomment if you get no picture on HDMI for a default "safe" mode
#hdmi_safe=1

# uncomment this if your display has a black border of unused pixels visible
# and your display can output without overscan
#disable_overscan=1

# uncomment the following to adjust overscan. Use positive numbers if console
# goes off screen, and negative if there is too much border
#overscan_left=16
#overscan_right=16
#overscan_top=16
#overscan_bottom=16

# uncomment to force a console size. By default it will be display's size
minus
# overscan.
#framebuffer_width=1280
#framebuffer_height=720

# uncomment if hdmi display is not detected and composite is being output
hdmi_force_hotplug=1

# uncomment to force a specific HDMI mode (here we are forcing 800x480!)
hdmi_group=2
hdmi_mode=87
hdmi_cvt=800 480 60 6 0 0 0
hdmi_drive=1

max_usb_current=1

# uncomment to force a HDMI mode rather than DVI. This can make audio work in
# DMT (computer monitor) modes
#hdmi_drive=2

# uncomment to increase signal to HDMI, if you have interference, blanking,
or
# no display
#config_hdmi_boost=4

# uncomment for composite PAL
#sdtv_mode=2

#uncomment to overclock the arm. 700 MHz is the default.
#arm_freq=800
```

OpenCPN

Se ha actualizado a la versión 5.2 en la última revisión del TFG. Los datos de conexiones tienen que figurar como se muestra en las imágenes.

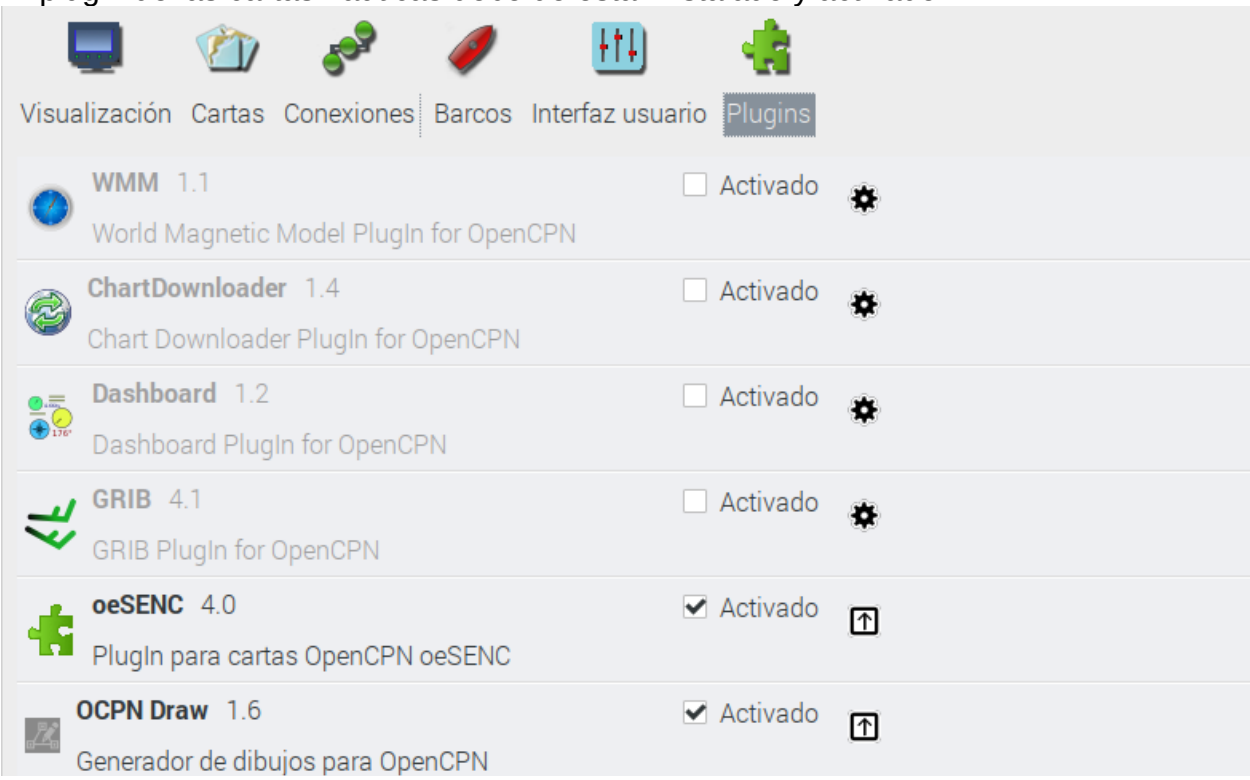


-TCP



-UDP

El plugin de las cartas náuticas debe de estar instalado y activado:



Si se cambia la RPi o el SO, habría que volver a comprar las cartas náuticas a través de o-charts. Sigue los siguientes pasos:
(https://manuals.o-charts.org/oesenc_es_ES.html)

Abre OpenCPN Opciones → Plugins → oeSENC y activalo. Crea su archivo de identificación del sistema desde Preferencias. El plugin te informará sobre la ruta de acceso al archivo. Para sistemas Windows y macOS se crea una copia directamente en el escritorio. Para los sistemas Linux se crea el archivo en la carpeta *~/.opencpn.

Copie el archivo de identificación en un dispositivo portátil y busque una computadora con conexión a Internet.

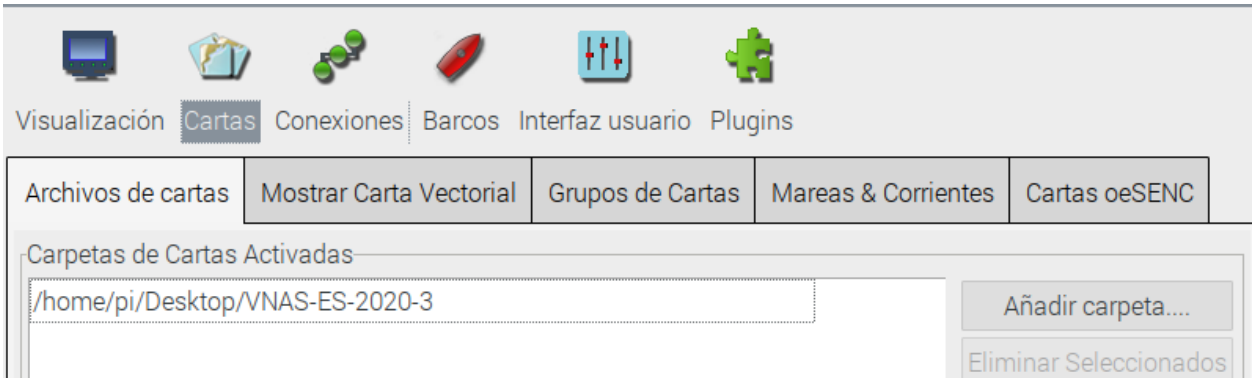
Vaya a o-charts shop y obtenga la licencia para los conjuntos de cartas que le interesen. Ignore este paso si ya ha comprado sus cartas.
Vaya a la página My oeSENC Charts y cree un Identificador de sistema, suba el archivo de identificación y elija un Nombre de Sistema.

Seleccione un Nombre de Sistema para cada conjunto de cartas licenciadas. Una vez asignado, no se puede cambiar. Si su computadora se daña, puede seleccionar un segundo nombre para cada conjunto de cartas como copia de seguridad. En ese caso, crea un nuevo Identificador de Sistema para el nuevo equipo.

Solicite las cartas presionando el botón que aparece en la columna Última solicitud y se procesará el conjunto de cartas. El tiempo de procesamiento depende del tamaño del conjunto de cartas, la cola en el servidor y la carga de red en ese momento, pero nunca será más de 2 horas. Más bien estamos hablando de minutos.

Recibirá un email con un enlace para descargar su conjunto de cartas. También puedes descargarla desde la página Mis Cartas oeSENC. Si no descarga su carta en una semana tendrás que hacer una nueva solicitud. Descarga su carta, cópiala en algún dispositivo de almacenamiento portátil y vuelve al barco.

Descomprime las cartas en una carpeta a tu elección, y localízalas en tu OpenCPN



7.3. ANEXO 3: Materiales y coste

Los materiales que se han adquirido, así como el coste se desglosa a continuación.

- Raspberry Pi IP65 Weatherproof IoT Project Enclosure**.** Caja con protección IP65 en su apertura de plástico, y un conector USB de IP68. **66,55€**
- Conectores USB IP68 PX0845/B. **12,32€**
- Conectores USB IP68 DCC-US3AT-180. **12,79€**
- Display táctil HDMI 5" 800x480. **74,95$**
- Cable HDMI flexible, de 0,3 m. **5,99€**
- Cable USB 2.0 A a micro USB B con trenzado doble de 0,3 m. **8,44€**
- Rii Mini X1 teclado inalámbrico con ratón táctil. **15,99€**
- GeeekPi Caja de aleación de Aluminio con Ventilador de refrigeración Doble. **14,99€**
- Actisense NGT-1 N2K a USB. **125€**

El monto total de los componentes adquiridos es alrededor de los **337€.**