UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Integración del sistema de control electrónico de motores eléctricos para el desarrollo de un barco con navegación autónoma

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN SISTEMAS DE TELECOMUNICACIÓN

Autor: ESCOBAR HERNÁNDEZ, MIGUEL ÁNGEL

Director: MATEO AROCA, ANTONIO

Codirector: MOLINA GARCÍA-PARDO, JOSÉ MARÍA

Cartagena, a 5 de Julio de 2021

Autor	Miguel Ángel Escobar Hernández
E-mail del autor	miguelangelescobarhernandez@gmail.com
Director	Antonio Mateo Aroca
E-mail del director	antonio.mateo @upct.es
Título del TFG	Integración del sistema de control electrónico de motores eléctricos para el desarrollo de un barco con navegación autónoma.

Resumen:

El proyecto consiste en el diseño y desarrollo de un barco autónomo propulsado a motor eléctrico y con vela rígida, el cual deberá superar unas pruebas que se realizarán en el puerto de Cartagena para posteriormente en un futuro poder competir en The microtransat challenge. Este proyecto se enmarca en un trabajo multidisciplinar que parte de la recopilación de los datos de la Raspberry en una placa de Arduino situada en el barco autónomo, que se transmiten vía radio y que posteriormente, se muestran a través de un software desarrollada específicamente para este propósito. Con objetivo de poder efectuar esa comunicación entre la Raspberry y la placa de Arduino y el posterior movimiento del barco este proyecto se encarga de la integración física y la interpretación de la información de la cadena de datos que recibe la placa y traducirla al funcionamiento de los motores del barco autónomo.

Titulación	Grado en Ingeniería de Sistemas de Telecomunicación
Departamento	Automática, Ingeniería Eléctrica y Tecnología Electrónica
Fecha de Presentación	Julio/2021

ÍNDICE

1. INTRODUCCIÓN

1.1 MOTIVACIÓN

La principal motivación para realizar este proyecto es la de cumplir el objetivo final, que se trata de entender y controlar la tecnología para, en un futuro poder competir en "The Microtransat Challenge". Esta competición consiste en una carrera transatlántica para embarcaciones autónomas, la cual tiene como objetivo promover el desarrollo de dichas embarcaciones dirigidas remotamente y de forma automatizada a través de esta competencia.

Como caso previo, se ha diseñado un catamarán autónomo propulsado por motores eléctricos alimentados por baterías, el cual se pondrá a prueba en el puerto de Cartagena empleando el mecanismo autónomo y el de control manual en la navegación.

Este proyecto es de carácter multidisciplinar, por lo que engloba una serie de tareas divididas en proyectos específicos que han sido asignados en función de los estudios realizados por cada alumno.

A este proyecto explícitamente le compite el desarrollo e implementación de la comunicación entre la Raspberry Pi y un controlador, que en este caso es un Arduino Micro, para poder controlar los dos motores que componen la embarcación.

1.2 OBJETIVOS

El objetivo de este TFG (Trabajo Final de Grado) es el desarrollo de un sistema de control de los motores de un barco autónomo por medio de un controlador con objeto de poder garantizar su movimiento a partir de la información recibida de manera remota y manual por un usuario.

El proyecto engloba el desarrollo de dos fases diferentes:

La primera hace referencia a la comunicación entre una Raspberry, el controlador y el motor al cual le ordenaremos varios parámetros.

La segunda fase engloba el diseño y la integración física del sistema en el barco para poder conseguir el objetivo final de poner en movimiento el barco en el mar. En esta segunda fase se comprenden el diseño de las tarjetas de circuito impreso y el cableado necesario para poder encaminar las señales y la alimentación desde las baterías.

1.3 FASES DEL PROYECTO

Encontramos varias fases dentro de la planificación y la realización de este proyecto:

- Búsqueda de información y estudio técnico del entorno de programación correspondiente al controlador que ha sido imprescindible realizar para poder conocer la interfaz IDE de programación de Arduino con sus respectivas librerías, protocolos de comunicación y componentes necesarios.
- Comunicación del controlador con el motor a través del puerto serie. Esta fase consiste en garantizar la comunicación con el motor desde la consola del ordenador a través de nuestro controlador y el driver del motor que describiremos más adelante con más precisión.
- Pruebas para conocer el comportamiento del driver en vacío y en el agua en la piscina de la Escuela de Ingeniería Naval. Una vez terminada la fase anterior de comunicación se procede a la realización de una serie de pruebas con el objetivo de analizar el comportamiento del motor y la resistencia del driver bajo funcionamiento.
- Modificar programa para interpretar información recibida por la Raspberry. Al observar que el motor y el controlador responden de manera satisfactoria procedemos a modificar el código con el objetivo de automatizar la comunicación de nuestra placa con la Raspberry Pi. En este apartado se comienza a colaborar juntamente con José Carlos Urrea para concretar el modo de comunicación, que será vía serial.
- Desarrollo del sistema manual. El VNAS debe constar de un sistema de control manual que está implementado mediante dos potenciómetros deslizantes y un interruptor, controlados por el controlador.
- Diseño de tarjetas de circuito impreso (PCB) con el conexionado de nuestro sistema. Para este proceso se utilizará el entorno Eagle y el conexionado entre las distintas placas se realiza a través de conectores de varios pines (hasta 4) para los cables de señal y cable de potencia para los de su mismo tipo.
- Integración y montaje de todo el sistema.
- Pruebas y diagnóstico de nuestro sistema.

1.4 ESTRUCTURA DE LA MEMORIA

El proyecto consta de una memoria dividida en nueve partes principales:

- En la primera fase se introduce el objetivo principal del proyecto, un pequeño resumen de este y las fases que comprenden su correcta elaboración.
- La siguiente fase se corresponde a la referencia histórica de este proyecto, con antecedentes, etc.
- En la tercera fase contemplamos una visión más cercana y minuciosa del proyecto en cuestión, con su correspondiente descripción detallada de los materiales y las tecnologías usadas para llevar a cabo el proyecto.
- En la cuarta fase se procede a indicar todas las herramientas de software usadas para conseguir el propósito del proyecto.
- La quinta fase corresponde a la descripción del diseño detallado de las tarjetas de circuito impreso (PCB) que se han implementado en la estructura del VNAS.
- La sexta fase engloba una descripción técnica y más profundizada de la parte software, que contiene la explicación de los protocolos de comunicación usados y las distintas formas de comunicación, ya sea serie o manual.
- En el séptimo apartado se llevan a cabo las comprobaciones de los apartados descritos anteriormente para poder garantizar su correcto funcionamiento.
- Seguidamente se introduce una conclusión relacionada con las simulaciones y sus respectivos resultados.
- Por último, se muestran las opciones a líneas futuras para la continuación del proyecto y se incluyen también los anexos incluidos a lo largo de la memoria y su bibliografía.

2. ESTADO DEL ARTE

A continuación, se realiza una introducción teórica al proyecto en la que se describirán algunos conceptos fundamentales como son la telemetría, señales y conexiones del controlador utilizado en el proyecto y unos antecedentes de proyectos similares.

2.1 LA TELEMETRÍA

Se conoce como telemetría al sistema que permite la monitorización, mediación y/o rastreo de magnitudes físicas o químicas a través de datos que son transferidos a una central de control.

El sistema de telemetría se realiza normalmente mediante comunicación inalámbrica pero también se puede realizar a través de otros medios como: teléfono, redes de ordenadores, enlace de fibra óptica, entre otros. La telemetría es usada en áreas muy diversas que va desde el automovilismo, aviación, astrología, pasando por la agricultura, industria de petróleo, medicina y hasta biología.

La telemetría tiene como objetivo permitir la mediación de magnitudes físicas o químicas, conocer los estados de los procesos y sistema, así como controlar de manera remota el funcionamiento, corregir los errores y enviar la información recabada hacia un sistema de información para su uso y provecho.

El sistema de telemetría funciona por medio de un transductor como dispositivo de entrada, un medio de transmisor en forma de líneas de cable u ondas de radio, procesamiento de señales, dispositivo de grabación o visualización de datos. El transductor tiene como principal función convertir la magnitud física o química como: la temperatura, presión, vibraciones, voltaje, en una señal eléctrica, que es transmitida a distancia a efecto de ser registrada y medida.

La telemetría permite supervisar los niveles de líquidos en ríos, contenedores, depósitos, entre otros, permite medir los parámetros de fluidos como temperatura, presión, caudales, y el monitoreo del medio ambiente como la propiedad del viento, agua, aire, y detectar gases peligrosos para el mismo. Del mismo modo, prevé cuando pueda ocurrir un desastre natural como tsunami, a través de telemetría por radio, que mide el comportamiento de ondas y tamaños.

Etimológicamente, la palabra telemetría es de origen griego “tele" que significa “distancia” y “metría” que expresa “medida”.[1]

Figura 1: Ejemplo de telemetría en transportes de carga. [31]

A continuación, se muestran los conceptos básicos de funcionamiento con los que cuenta un sistema de telemetría en vehículos.

- **Recopilación de datos**: En los puntos clave del vehículo se instalan sensores capaces de medir datos como velocidad, ubicación o estado de la carga, entre otros.
- **Uso de aparatos externos a los equipos**: Los datos se recogen con aparatos externos a los equipos, por lo que no interfieren en su funcionamiento ni es necesario desmontar las piezas originales para su instalación. Los sensores nunca tocan los equipos.
- **Transmisión a un emisor instalado en el vehículo**: Los datos recogidos pasan por transmisión inalámbrica a un dispositivo instalado en el vehículo, que puede ser desde un teléfono celular hasta un ordenador u otro transmisor conectado a una red por radio o comunicación satelital.
- **Levantamiento de datos**: Los datos recogidos por los sensores son convertidos en magnitudes comunes que pueden ser leídas rápidamente, tales como km/h, temperatura, niveles de baterías, velocidad del viento o ubicación en un mapa satelital (en función del tipo de sensores que tengamos).
- **Comunicación inalámbrica**: Ya sea por red celular o red satelital, los datos se transmiten desde el dispositivo a una central ubicada a distancia, en la que se centralizan los datos en tiempo real.
- **Retroalimentación inmediata**: La central está en condiciones de recibir y evaluar los datos en cualquier momento. Esto permite enviar órdenes de forma inmediata, ya sea por contacto con operador humano o por una orden automatizada.

En un sistema de telemetría es posible encontrar algunas de las siguientes características, que sirven para monitorear y evaluar variables relacionadas con la seguridad y la óptima gestión de los recursos y activos:

- **Acelerómetro**: Su función es la de medir el cambio de velocidad respecto al tiempo, como las aceleraciones y desaceleraciones. Para que ejerzan su función con total eficacia es necesario que estén bien fijados al vehículo teniendo en cuenta los movimientos e inclinaciones del rumbo fijado, que puede afectar levemente a su precisión.
- **GPS**: A través de información satelital, se encarga de fijar la ubicación del vehículo, permitiendo su localización y además, gestión en tiempo real de información de carga, temperaturas y seguridad del conductor (en caso de que el vehículo no sea autónomo).
- **Sensores**: Dentro del sistema existen diversos tipos de dispositivos que, en conjunto con el GPS y acelerómetro, permiten añadir información externa. Estos dispositivos son capaces de medir velocidades en ruedas, presión de frenado e información de la columna de dirección.[2]

2.2 CONEXIONES ANALÓGICAS, PWM Y SERIAL DE ARDUINO

Las entradas analógicas se utilizan para leer la información de la magnitud física que nos proporciona los sensores de temperatura, luz, distancia, etc. La tensión que leemos del sensor no la proporciona un circuito asociado a dicho sensor en un rango de valores de tensión continua entre 0V y 5V.

La placa de Arduino tiene 6 entradas analógicas marcados como “A0”, “A1”,..., “A5” que reciben los valores continuos en un rango de 0V a 5V, pero la placa Arduino trabaja sólo con valores digitales, por lo que es necesario una conversión del valor analógico leído a un valor digital. La conversión la realiza un circuito analógico/digital incorporado en la propia placa.

El conversor A/D de la placa tiene 6 canales con una resolución de 10 bits. Estos bits de resolución son los que marcan la precisión en la conversión de la señal analógica a digital, ya que cuantos más bits tenga más se aproxima al valor analógico leído. En el caso de la placa Arduino el rango de los valores analógicos es de 0 a 5 V y con los 10 bits de resolución se puede obtener de 0 a 1023 valores digitales y se corresponde cada valor binario a (5V/1024) 5 mV en el rango analógico.

En estas condiciones son suficientes para hacer muchos proyectos tecnológicos. En el caso de necesitar mayor resolución y como no podemos aumentar el número de bits de conversor A/D se puede variar el rango analógico utilizando el voltaje de referencia Vref.

Las entradas analógicas tienen también la posible utilización como pines de entrada-salida digitales, siendo su enumeración desde 14 al 19.[3]

Por otro lado, la señal PWM (Pulse Width Modulation, Modulación de Ancho de Pulso) es una señal que utiliza el microcontrolador para generar una señal continua sobre el proceso a controlar. Por ejemplo, la variación de la intensidad luminosa de un led, el control de velocidad de un motor de corriente continua y demás.

Para que un dispositivo digital, microcontrolador de la placa Arduino, genere una señal continua lo que hace es emitir una señal cuadrada con pulsos de frecuencia constante y tensión de 5V. A continuación, variando la duración activa del pulso (ciclo de trabajo) se obtiene a la salida una señal continua variable desde 0V a 5V.

Veamos gráficamente la señal PWM:

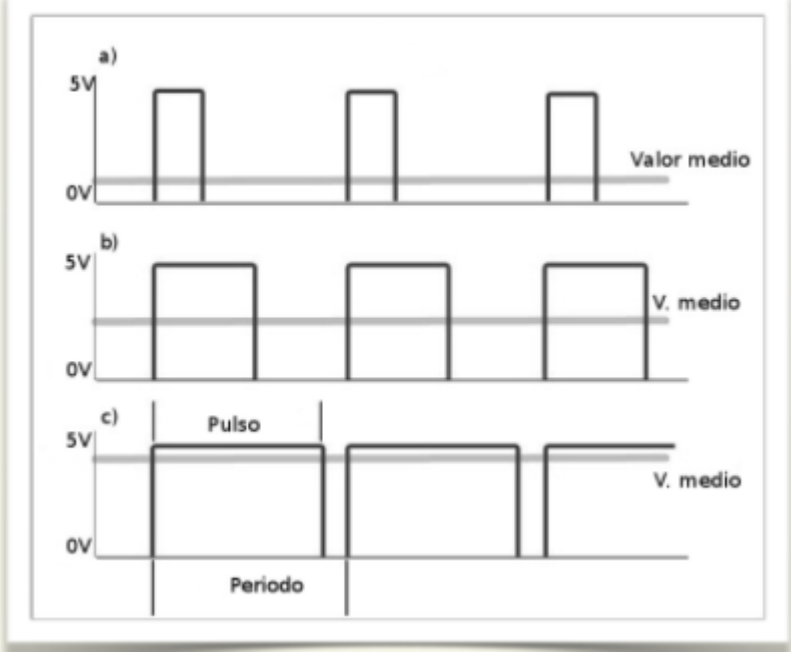

Figura 2: Ejemplo señal PWM [4]

Los pines digitales de la placa Arduino que se utilizan como salida de señal PWM generan una señal cuadrada de frecuencia constante (490Hz), sobre esta señal periódica por programación podemos variar la duración del pulso como vemos en estos 3 casos:

- La duración del pulso es pequeña y la salida va a tener un valor medio de tensión bajo, próximo a 0V.
- La duración del pulso es casi la mitad del período de la señal, por tanto, la salida va a tener un valor medio de tensión próximo a 2,5V.
- La duración del pulso se aproxima al tiempo del período y el valor medio de tensión de salida se aproxima a 5V. [4]

A continuación, en referencia al puerto serie, una buena definición sería la siguiente:

"Un puerto serie o puerto en serie es una interfaz de comunicaciones de datos digitales, frecuentemente utilizado por computadoras y periféricos, donde la información es transmitida bit a bit, enviando un solo bit a la vez; en contraste con el puerto paralelo que envía varios bits simultáneamente."

Es decir, el puerto serie nos va a servir para comunicar nuestra Raspberry Pi con la placa de Arduino, tanto para mandar datos -en forma de secuencias de bits-, desde la Raspberry Pi a Arduino, como al revés y a una velocidad determinada, en bits por segundo (baudios, o lo que es lo mismo, número de unidades de señal por segundo, teniendo en cuenta que un baudio puede contener varios bits, dependiendo del esquema de modulación). También podríamos usar el puerto serie para realizar la comunicación entre dos Arduinos u otros dispositivos.

Para el envío físico de las secuencias de bits el puerto serie necesita de al menos dos conectores, RX (recepción) y TX (transmisión), a través de ellos podrá realizar la comunicación de datos.[5]

2.3 APLICACIONES Y EJEMPLOS DE TELEMETRÍA EN EMBARCACIONES

- **Saildrone**: El primer velero no tripulado en cruzar el Atlántico en ambas direcciones. La unidad SD-1021 de la serie Saildrone -veleros sin patrón de 7 metros-, completó, el 22 de octubre de 2019, su regata transatlántica de Este a Oeste, de Lymington (Inglaterra) a Newport (Estados Unidos), en 68 días por la ruta norte. Sigue una travesía de Oeste a Este del Atlántico, entre Bermudas y el Solent, realizada el 7 de agosto en 75 días. Tras la gira por la Antártida, finalizada el 3 de agosto de 2019, los veleros autónomos de Richard Jenkins confirman su liderazgo en el dron de vela.[6]

Figura 3: Saildrone SD-1021 [6]

- **X Shore:** El barco eléctrico del futuro. El Eelex 8000 es un diseño que revoluciona la náutica de recreo. Tradicionalmente, para desarrollar el proyecto de un nuevo barco se recurre a un gabinete de diseño y de arquitectura naval. Pero no es el caso de X Shore, que ha contratado a especialistas de otros ámbitos (Rolls Royce, Chalmers University), aunque para la producción, ha optado por el astillero sueco Storebro. Para muchos, Rolls Royce, evoca automóviles de lujo, pero ésta no la única actividad de esta empresa. Durante más de 40 años, la firma británica, adquirida en 1998 por BMW, cuenta con su propio centro de investigación de hidrodinámica HRC (Hydrodynamic Research Centre) equipado con dos túneles de cavitación que permiten evaluar el rendimiento del sistema de propulsión de un barco. [7]

Figura 4: X Shore [7]

- **USV Vendaval**: La embarcación de 10,23 metros de eslora está pensada para actuar en labores de vigilancia y rescate, que puede desarrollar tanto de forma autónoma, como remota o con tripulación a bordo. En su construcción fue clave el trabajo del astillero moañés Aister Aluminium Shipyard, que se encargó de la fabricación en aluminio del barco, así como del diseño e instalación de todos los automatismos que interactúan con el software ideado e instalado por Navantia en su base de San Fernando (Cádiz). Según explican sus creadores, la embarcación dron, nombrada USV Vendaval, será la primera en entrar en servicio en España. [8]

Figura 5: USV Vendaval [8]

3. PROYECTO VNAS

En este apartado se definen los proyectos que conforman el prototipo VNAS, un esquema global del sistema y los componentes y materiales utilizados para la realización de este proyecto.

3.1 DESCRIPCIÓN DEL PROYECTO VNAS

La idea de la realización de este proyecto ha sido impulsada por un grupo de profesores pertenecientes a diferentes escuelas que componen la Universidad Politécnica de Cartagena, la cual está encabezada por el codirector de este trabajo José María Molina García-Pardo, profesor de la escuela de telecomunicaciones, integrante del departamento de tecnologías de la información.

El proyecto VNAS también cuenta con profesores de la escuela de ingeniería naval y de ingeniería industrial, los cuales se encargan de supervisar sus partes correspondientes del proyecto.

Como hemos descrito anteriormente este proyecto es de carácter multidisciplinar, lo que significa que lo componen varios trabajos de fin de grado de diferentes escuelas y para ello es necesario una coherencia y concordancia común.

Encontramos varias partes para llevar a cabo el proyecto:

- Diseño y construcción del catamarán.
- Desarrollo de un sistema de control para la navegación de un catamarán de forma autónoma.
- Integración del sistema de control electrónico de motores eléctricos para el desarrollo de un barco con navegación autónoma.
- Implementación de una herramienta software para el control y seguimiento remoto de un barco autónomo.

Figura 6: Logo VNAS

3.2 ESQUEMA GLOBAL DEL SISTEMA

En este apartado podemos observar en la figura 8 el esquema global del sistema compuesto por todos sus componentes, los cuales describimos a continuación a partir del siguiente apartado.

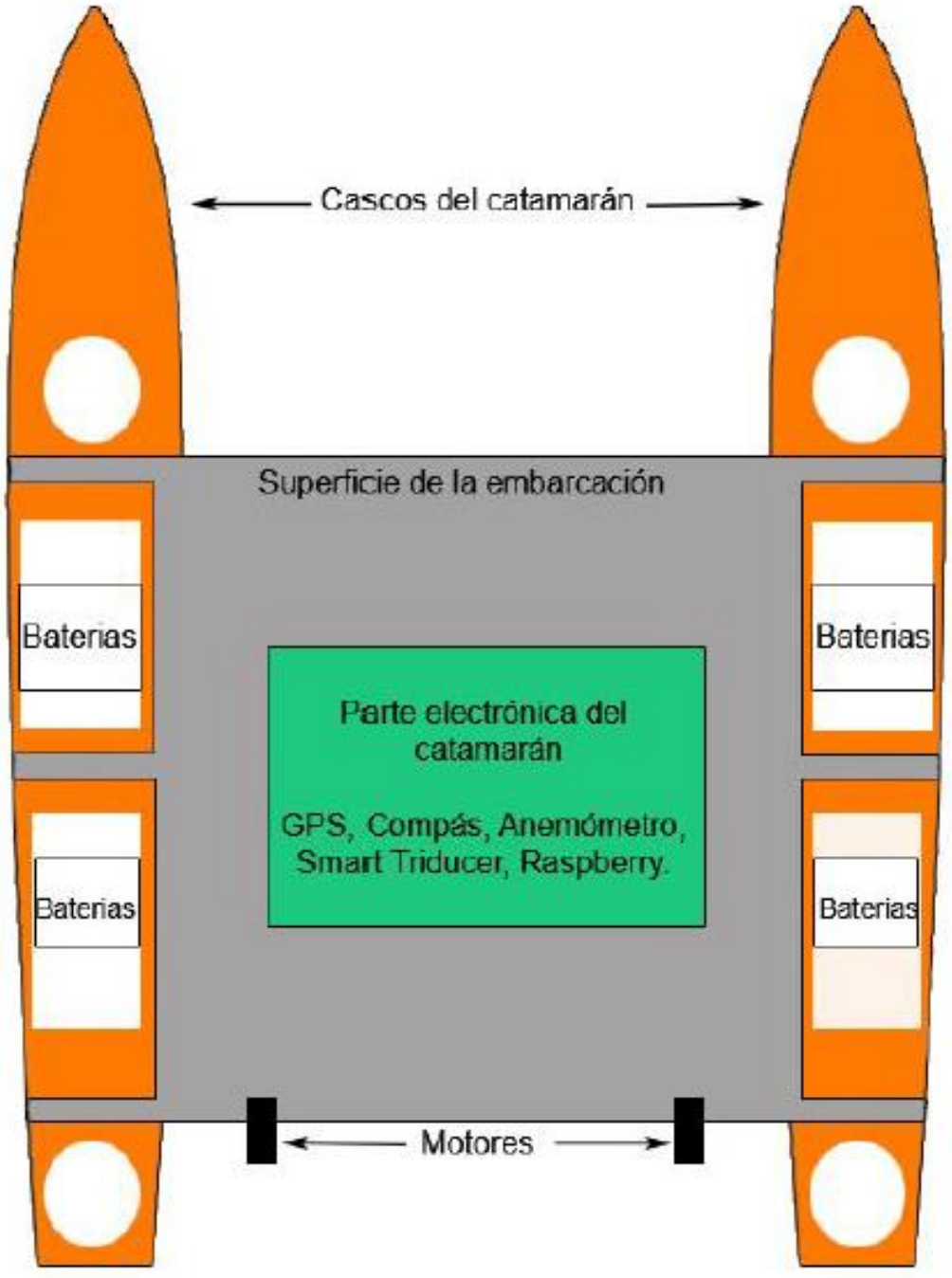

Figura 7:Esquema global del sistema. Vista alzada. [9]

La superficie de la embarcación es una plancha de acero sobre la que irá colocada la mayor parte hardware y el cableado correspondiente a este proyecto y al de “Avances en el desarrollo de un sistema de control para la navegación de un catamarán de forma autónoma” cuyo autor es José Carlos Urrea Celdrán. También podemos observar que el anclaje de los dos motores que describiremos más adelante se realiza sobre esta misma superficie.

Los cascos del catamarán tienen como funcionalidad permitir mantener el VNAS a flote, y dentro de ellos irán incrustadas las baterías que alimentan todo el sistema.

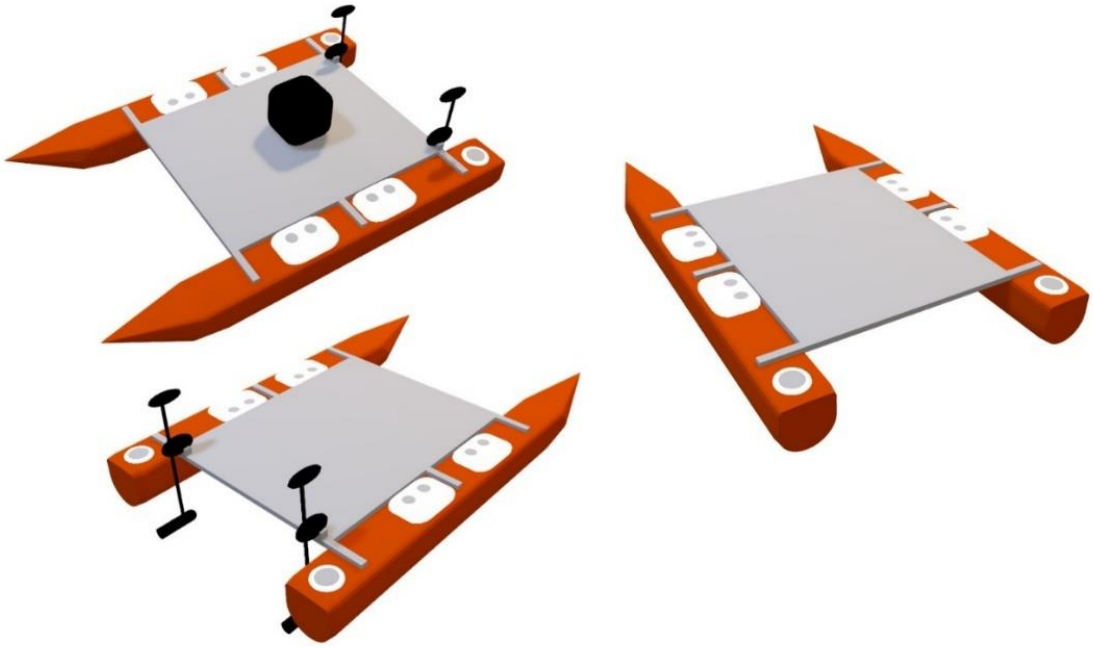

Figura 8: Esquema global del sistema. Vista 3D

3.3 COMPONENTES HARDWARE UTILIZADOS

En este apartado nombraremos y realizaremos una introducción de los distintos elementos y dispositivos utilizado para llevar a cabo el proyecto.

3.3.1 RASPBERRY PI

Este componente, la Raspberry PI 3 Modelo B+, será el "cerebro" de todo el proyecto global VNAS, e irá conectada a nuestro controlador por el puerto serie. Mediante esta conexión, en el controlador recibiremos los valores analógicos de ambos motores que describiremos a continuación y a su vez, desde el Arduino enviaremos varios parámetros que se describirán más adelante también.

Figura 9: Raspberry PI 3 Modelo B+ [10]

3.3.2 ARDUINO MICRO

En su esencia, Arduino es una pieza pequeña y asequible del llamado hardware abierto – una placa de circuito que se puede utilizar para proyectos electrónicos sencillos. También puede unirse a algunos de ellos para construir estructuras más complejas.

Hay muchos modelos diferentes de placas Arduino.

En una parte de nuestro proyecto nos centramos en la programación de la placa Arduino Micro, que vamos a describir detalladamente a continuación.

El Arduino Micro está basado en el microcontrolador Atmel ATmega32U4. Con un oscilador de cristal de 16 MHz, el microcontrolador cuenta con una resolución de 8 bits, 32 KB de flash y 2.5 KB de RAM. Su controlador USB integrado es una característica útil del ATmega32U4 que le permite ahorrar espacio, pues reduce la necesidad de microcontroladores secundarios y le permite conectarse a cualquier dispositivo con puerto serie de la misma manera que un teclado o un mouse.

La característica principal de la placa es su microconector USB, que le permite conectarse fácilmente al dispositivo, que en nuestro caso será una Raspberry. También contiene un botón para resetear, un cabezal de reprogramación ISP de 6 pines y 20 pines digitales I/O, de los cuales, 12 se pueden utilizar para entradas análogas y 7 se pueden reconfigurar para salidas de modulación por ancho de pulsos (PWM), que utilizaremos para la señal del motor.

El Arduino Micro cubrirá todas nuestras necesidades para integrar la tecnología de microcontrolador en nuestro proyecto.[11]

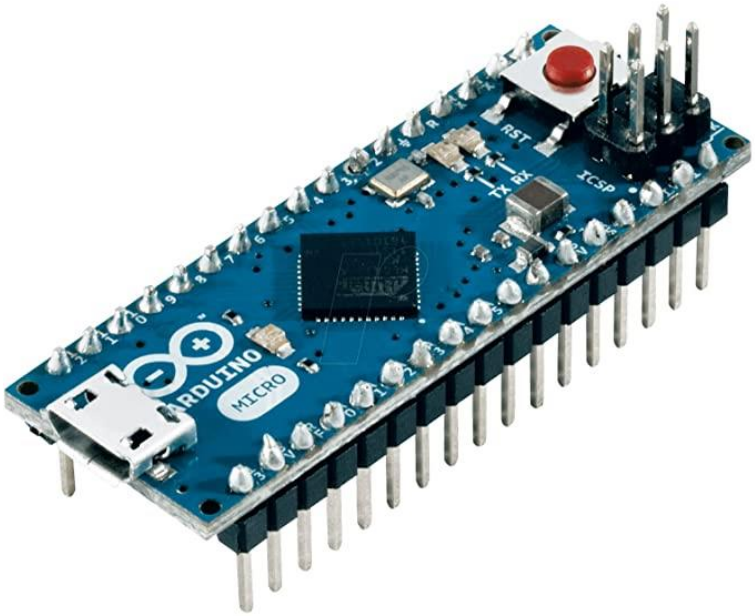

Figura 10:Arduino Micro

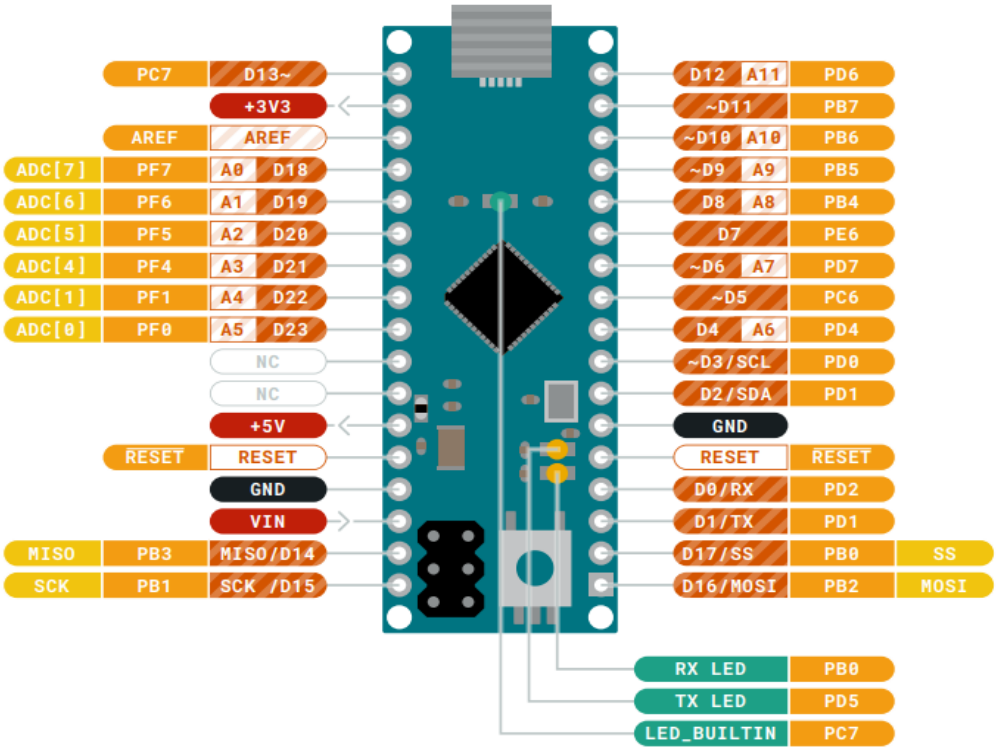

Figura 11: Pinout Arduino Micro [12]

3.3.3 DRIVER DEL MOTOR

El controlador escogido para controlar la salida PWM de la placa Arduino y controlar el motor es el Pololu G2 High-Power Motor Driver 18v25.

Sus características principales son:

- Voltaje de funcionamiento: 6.5V a 30V (máximo absoluto)
- Corriente de salida: 25 A continuos
- Entradas compatibles con lógico de 1.8V, 3.3V y 5V
- Funcionamiento PWM hasta 100KHz
- Salida de detección de corriente proporcional a la corriente del motor
- Limitación de corriente activa (corte) con umbral predeterminado de 60 A
- Protección de voltaje inverso
- Apagado por subtensión
- Protección contra cortocircuitos.

[13]

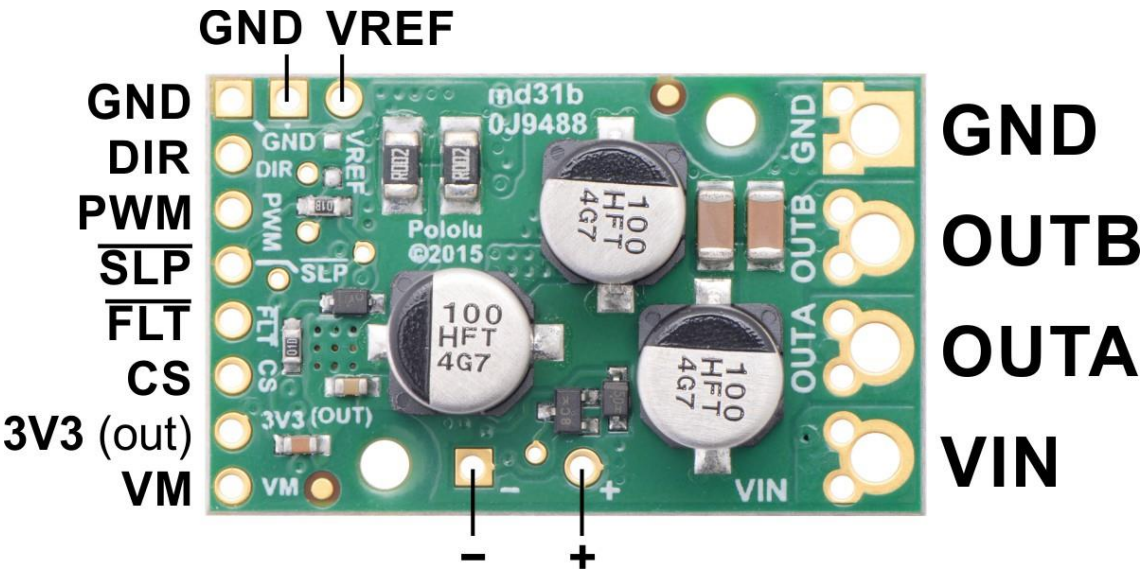

Figura 12: Controlador Pololu G2 18v25. Vista superior pinout etiquetado [14]

Como se puede apreciar en la imagen disponemos de varios pines de señal (situados en la parte izquierda de la imagen), y también disponemos de cuatro pines de potencia (situados en la parte derecha de la imagen).

En nuestro proyecto utilizaremos los siguientes pines conectados al arduino:

- GND: Conectado al pin de masa
- DIR: Conectado a pin digital
- PWM: Conectado a pin digital
- FLT: Conectado a pin digital
- CS: Conectado a pin analógico

Y dentro de los pines de potencia conectados a la batería y al motor.

- GND: Conectado a la masa de la batería
- OUTA: Conectado a un terminal del motor
- OUTB: Conectado a otro terminal del motor
- VIN: Conectado al terminal positivo de la batería

Para un buen rendimiento, es muy importante instalar un condensador grande a través del suministro del motor y la tierra cerca del controlador del motor. Por lo general, recomendamos utilizar un condensador de al menos unos pocos cientos de µF y con una clasificación muy superior a la tensión de alimentación máxima; la capacitancia requerida será mayor si la fuente de alimentación es deficiente o está lejos (más de un pie) del controlador, y también dependerá de otros factores como las características del motor y la frecuencia PWM aplicada. Se puede instalar un condensador de orificio pasante directamente en la placa en los orificios etiquetados '+' y '-' (conectados a VM y GND, respectivamente). El controlador incluye tres capacitores integrados de 150 µF, que pueden ser suficientes para pruebas breves y un funcionamiento limitado de baja potencia, pero se recomienda encarecidamente agregar un capacitor más grande para la mayoría de las aplicaciones.[13]

3.3.4 MOTORES

El proyecto también está compuesto por dos motores de corriente continua Jago Electric Outboard, modelo que podemos observar con más detenimiento en las figuras “tal”.

Dentro de sus principales características podemos destacar las siguientes.

- Voltaje de funcionamiento 12 V.
- La rotación de la unidad de control es desde 0° hasta 360°.
- Potencia de entrada máxima es 0.564 kW.
- Es ajustable y se puede variar la profundidad de inmersión de la hélice.
- Mango de extensión del timón extraíble.

[15]

Figura 13: Parte de unidad de control. [15]

Figura 14: Parte de calibración de inmersión de hélice. [15]

Figura 15: Motor Jago Electric Outboard [15]

3.3.5 BATERÍAS

Para la alimentación de todo el sistema es necesaria la utilización de baterías de plomo ácido que suministran una tensión de 12 V.

Figura 16: Batería Yuasa [16]

3.3.6 TRANSFORMADOR 12/5V

Se utilizará un convertidor de 12 a 5V de tensión, ya que para el control de los motores necesitamos 12V para alimentar los propios motores y a su vez los 5V para alimentar la electrónica del circuito (Raspberry, controladores, potenciómetros, botón de cambio de modo).

Figura 17: Transformador 12/5V [17]

3.3.7 POTENCIÓMETRO DESLIZANTE

Figura 18: Potenciómetro lineal [18]

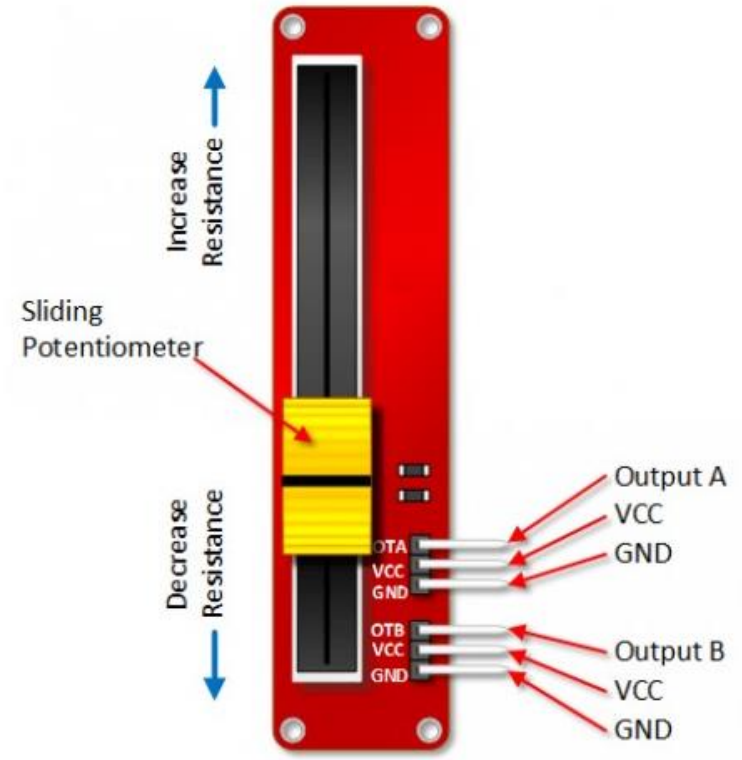

Figura 19: PinOut Potenciómetro lineal [18]

En nuestro caso utilizaremos únicamente una salida analógica del slider que estará conectada a una entrada analógica del controlador que corresponde a un valor comprendido entre 0 y 1023.

3.3.8 INTERRUPTOR

Se utilizarán dos interruptores, uno para seleccionar el modo de control deseado y otro para alimentar la CPU principal o dejarla apagada.

- El interruptor de cambio de modo de control nos permite cambiar el modo de lectura de los datos a gusto de la persona que maneja el VNAS, es decir, podemos permitir que el VNAS lea los datos y ajuste la velocidad y sentido de giro de los motores automáticamente mediante la comunicación con la Raspberry, o por el contrario podemos manejar manualmente dichos parámetros con los dos potenciómetros deslizantes que vamos a disponer tal y como los hemos descrito anteriormente.

- El interruptor encargado de gestionar la alimentación de la CPU central (Raspberry) se utilizará para poder ahorrar consumo de baterías si estamos utilizando el VNAS en modo manual y no necesitamos tener conectada la CPU central.

Figura 20: Interruptor luminoso estanco [19]

3.3.9 BOTÓN DE EMERGENCIA

Este botón es necesario para poder cortar la alimentación de todo el sistema en el caso de observar alguna avería o cualquier imprevisto que aparezca en las pruebas o en su función de trabajo. Nos sirve para poder prevenir un mayor grado de roturas o accidentes ya que debe ser empleado cuando sea necesario realmente.

Figura 21: Seta de parada de emergencia [20]

3.3.9 CONECTORES, CABLES Y PROTECCIÓN DE LA CIRCUITERÍA

Los conectores utilizados serán desde dos hasta cuatro pines, de los cuales se realiza su correspondiente crimpado y preparación para su uso.

En las siguientes figuras vemos las imágenes de estos con sus respectivas carcasas.

- Contacto de crimpado

Figura 22: Contacto de crimpado [21]

Los contactos del conector de PCB son una de las partes clave de los componentes de las placas de circuito impreso. Se usan para completar el circuito al conducir la corriente eléctrica desde el conector a la PCB. El conector suele estar dentro de una carcasa de conector para PCB, especialmente cuando se utilizan conectores múltiples.[22]

- Conector de dos pines

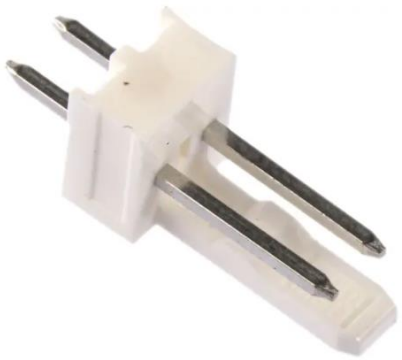

Figura 23: Conector macho dos pines [23]

Figura 24: Carcasa conector hembra dos pines [24]

- Conector de tres pines

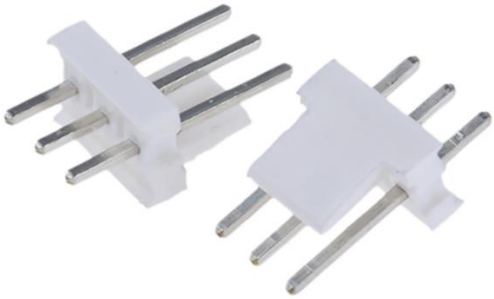

Figura 25: Conector macho tres pines [25]

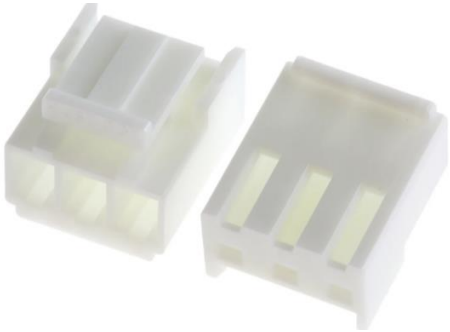

Figura 26: Carcasa conector hembra tres pines [26]

- Conector de cuatro pines

Figura 27:Conector macho cuatro pines [27]

Figura 28:Carcasa conector hembra cuatro pines [28]

- Conector tipo fastón

Este conector se utiliza para conectar los cables de alimentación que suministran un voltaje de 12 V.

Figura 29: Terminal de lengüeta [29]

Figura 30: Conector fastón [30]

Los cables usados serán formados por hilos internos independientes. Ese número de hilos corresponde al número de pines del conector que desemboca de ese cable. Es decir, se utilizarán cables que contienen de dos a cuatro hilos en su interior.

Figura 31: Ejemplo cable de cuatro hilos [31]

Para la protección de la electrónica y del motor se ha escogido un portafusibles con su correspondiente fusible de cartucho. Esto nos garantiza protección ante cualquier pico de corriente, ya que el fusible actúa como un cable que, si sobrepasa la intensidad que es capaz de soportar como máximo, actúa como un cable roto y no permitiría que se inyectara tanta cantidad de corriente a nuestro circuito.

En las siguientes figuras vemos el componente.

Figura 32: Portafusibles [32]

Figura 33: Fusible de cartucho [33]

Y, por último, para la protección del cableado y la electrónica y el aislamiento del mismo dentro de las cajas donde se integrará la electrónica se han utilizado prensaestopas para evitar la entrada de agua en el interior.

Figura 34: Prensaestopa [34]

4. LENGUAJES DE PROGRAMACIÓN Y HERRAMIENTAS USADAS

En este apartado vamos a realizar una pequeña descripción de las herramientas utilizadas para llevar a cabo el desarrollo del proyecto.

Han sido necesarios el estudio y profundización en el lenguaje de programación Arduino para la implementación software y el manejo del programa Eagle para el diseño de la implementación hardware.

4.1 LENGUAJES DE PROGRAMACIÓN

4.1.1 ARDUINO ENTORNO IDE

El entorno de desarrollo integrado (IDE, por sus siglas en inglés) de Arduino es un software de código abierto. Al igual que otros productos de Arduino, el Micro cuenta con una biblioteca asociada, desarrollada por la comunidad, que incluye código y otras funciones de ejecución para ayudar a los usuarios inexpertos a aprender y agilizar los proyectos de los fanáticos de Arduino que tienen más experiencia. El IDE también cuenta con otras características útiles, como sintaxis predictiva y resaltado de sintaxis, para hacer que su programación sea más fácil y eficaz.[11]

Debido a su simplicidad, los programas que se escriben usando el IDE de Arduino se llaman sketches o bocetos. En su esencia, son archivos de texto escritos en el lenguaje Arduino. Para guardarlos y subirlos a su placa Arduino, es necesario usar la extensión .ino.

Existen tres partes principales que componen el lenguaje de programación Arduino.

En primer lugar, tiene funciones que le permiten controlar tu placa. Usando funciones, puedes analizar caracteres, realizar operaciones matemáticas, y realizar otras tareas, por ejemplo, digitalRead() y digitalWrite() le permite leer o escribir un valor en un pin determinado.

Hay dos funciones que contiene cada boceto escrito en el lenguaje Arduino. Estas son setUp() y loop(). Un sketch siempre comienza con setUp(), que se ejecuta una vez después de encender o reiniciar tu placa. Después de crearlo, se utiliza loop() para hacer un bucle del programa repetidamente hasta que se apague o se resetee la placa.

A continuación, tenemos los valores de Arduino que representan constantes y variables. La mayoría de los tipos de datos (array, bool, char, float, etc.) son similares a los de C++. También puede realizar la conversión de tipos. La última parte del lenguaje de Arduino se llama estructura. Contiene pequeños elementos de código, como operadores.[35]

Figura 35:Logo Arduino IDE

4.2 HERRAMIENTAS DE DESARROLLO

4.2.1 EAGLE

EAGLE es una aplicación de automatización de diseño electrónico (EDA) programable con captura de esquemas, diseño de placa de circuito impreso (PCB) , enrutador automático y características de fabricación asistida por computadora (CAM). EAGLE significa Editor de diseño gráfico de fácil aplicación.

EAGLE contiene un editor de esquemas para diseñar diagramas de circuitos. Los esquemas se almacenan en archivos con extensión .SCH, las partes se definen en bibliotecas de dispositivos con extensión .LBR. Las piezas se pueden colocar en muchas hojas y conectar juntas a través de puertos.

El editor de diseño de PCB almacena archivos de placa con la extensión .BRD. Permite la anotación inversa al esquema y el enrutamiento automático para conectar trazas automáticamente en función de las conexiones definidas en el esquema.

EAGLE proporciona una interfaz gráfica de usuario de múltiples ventanas y un sistema de menús para editar, administrar proyectos y personalizar la interfaz y los parámetros de diseño. El sistema se puede controlar mediante el mouse, las teclas de acceso rápido del teclado o ingresando comandos específicos en una línea de comandos incorporada. Se pueden combinar varios comandos repetidos en archivos de script (con la extensión de archivo .SCR). También es posible explorar archivos de diseño utilizando un lenguaje de programación orientado a objetos específico de EAGLE (con extensión .ULP). [36]

Figura 36: Logo EAGLE

5. ESTRUCTURA ELECTRÓNICA Y DISEÑO DE PCB

Se realiza la descripción y justificación del conexionado empleado en este proyecto y seguidamente vemos la evolución del diseño y montaje de los PCB desarrollados.

5.1 DESCRIPCION DEL CONEXIONADO DEL VNAS

En primer lugar, contamos con dos controladores Arduino-Micro, los cuales, cada uno de ellos se encuentra dentro de cada motor y conectado a un driver de motor (componentes descritos anteriormente aquí). Estos dos controladores se comunicarán entre sí mediante el protocolo de comunicación I2C.

En la parte central del VNAS se ha instalado un armario Rack, que va a contener el sistema de navegación y la Raspberry Pi con todo el sistema de control manual y su cableado integrado en una tarjeta de circuito impreso que veremos a continuación.

A continuación, con el fin de simplificar el conexionado y ahorrar recursos, se ha declarado al Arduino del motor izquierdo como maestro, y al Arduino del motor derecho como esclavo.

Como ya sabemos, tenemos dos métodos de control integrados, mediante puerto serie y mediante potenciómetros deslizantes de forma manual. Vamos a describir los inconvenientes que podríamos encontrar según el método de control, para así justificar el uso del protocolo de comunicación I2C.

- Control serial: Si empleamos esta comunicación simultanea entre ambos Arduinos y la Raspberry podríamos tener problemas en el momento de distinguir información, y el código de programación de la Raspberry sería más complejo al tener dos dispositivos diferentes conectados de los cuales tendría que administrar y gestionar la información con más complejidad. Por esta razón, es más sencillo administrar con un único Arduino toda la información de los motores que nos transmite la Raspberry y por consecuencia se optimiza el cableado del sistema ahorrándonos un cable serial desde el motor derecho a nuestro rack central.

- Control manual: La mayor razón por la que nos conviene emplear la comunicación I2C en este método de control es por la optimización del cableado del sistema, así ahorramos dirigir al controlador esclavo un cable de uno de los potenciómetros y llevamos los dos cables de los potenciómetros al controlador principal, que es quien gestiona la información.

A continuación, se muestra un esquema unifilar del sistema con la descripción de las conexiones.

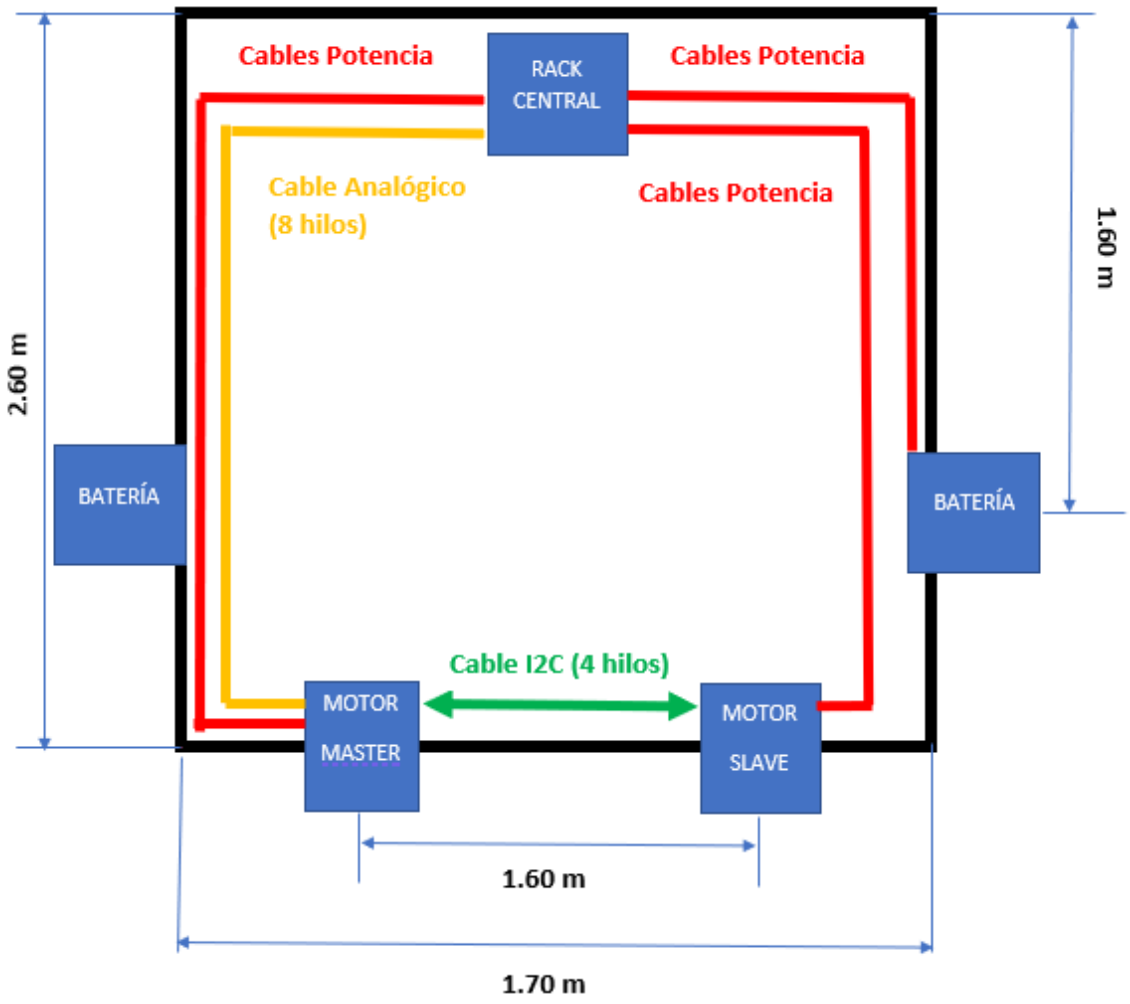

Figura 37: Esquema unifilar simple del conexionado del sistema

- Cables potencia: Primero debemos señalar que cada línea de cables de potencia está conformada por 2 cables, que son alimentación y masa. Teniendo en cuenta que nuestro Arduino estará instalado en la caja que tiene elevada el motor, necesitaremos (para cada conexión) 4.5 m de cable de 10 AWG→5.26 mm^2**.** Contando que van en "par" las conexiones con los cables de alimentación y de masa, para cada conexión hacia los Arduinos de nuestro esquema se necesitarán 10 metros de cable. Por tanto, al tener dos conexiones, una para cada Arduino, necesitaremos 20 metros. A esta cifra habría que añadirle la conexión desde la batería al Rack Central, que serían 6 metros al tener otro "par" de cables. Así llegamos a la conclusión de que necesitamos aproximadamente 26 metros de cable de 10AWG →5.26 mm^2 para poder realizar el conexionado físico de nuestro circuito.

- Cable analógico: Sabiendo las medidas anteriores, utilizaremos un cable de 8 hilos de 4.5 metros de longitud.

- Cable I2C: Observando la disposición de los motores, manteniendo la misma altura, necesitaremos un cable de 4 hilos de 2.5 metros de longitud.

5.2 DISEÑO Y MONTAJE DE PCB

Una tarjeta de circuito impreso (PCB) es un circuito cuyos componentes y conductores están contenidos dentro de una estructura mecánica. Las funciones conductoras incluyen trazas de cobre, terminales, disipadores de calor o conductores planos. La estructura mecánica se hace con material laminado aislante entre capas de material conductivo. La estructura general es chapada y está cubierta con una máscara de soldadura no conductora y una pantalla de impresión para la ubicación de leyenda de componentes electrónicos.[37]

En este apartado se muestra la versión digital de los esquemáticos y el montaje final con los componentes soldados de cada una de las tres PCB que forman el sistema.

Además, se describirá y explicará los componentes añadidos a los PCB y las funciones que le corresponden a cada uno.

A continuación, vemos un pequeño grafo resumen de los principales componentes y funciones que conforman cada PCB.

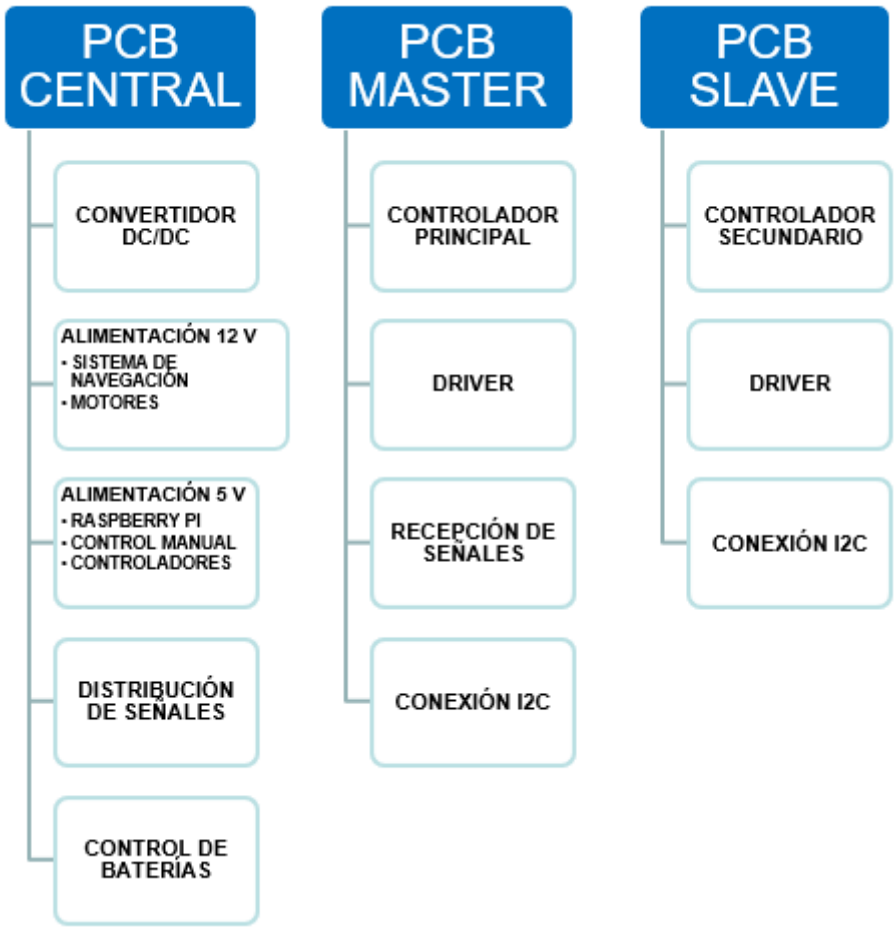

Figura 38: Esquema de componentes y funciones realizadas por cada PCB

5.2.1 PCB CENTRAL

En primer lugar, usando la plataforma Eagle, se elabora el esquemático del circuito usando las librerías de los componentes que vamos a utilizar. Es indispensable asegurar que las librerías corresponden exactamente con los componentes utilizados para que las medidas y el espacio distribuido para cada componente sea el mismo que su tamaño real, por esa misma razón, se han creado manualmente las librerías.

A continuación, vamos a describir los componentes que forman esta tarjeta de circuito impreso.

Los fastones J2 y J1 son conectores donde se conectan los cables de alimentación procedentes de la batería. Respecto a los fastones que los continúan, el positivo de la alimentación se conecta al mecanismo de parada de emergencia, que es un botón. Y el negativo de la alimentación irá conectado al fastón que tiene la etiqueta “GND”.

Los siguientes conectores de tipo fastón, provienen del mecanismo de emergencia en el caso del polo positivo, y el negativo del conector descrito anteriormente. Estos están conectados con otros 2 conectores por cada polo de alimentación, que corresponden a la alimentación de los motores, es decir, dos positivos y dos negativos.

Estos conectores fastón están conectados a un fusible para proteger la electrónica y a un divisor de tensión formado por dos resistencias de 8.2 y 4.7 kΩ a partir del cual extraemos una señal analógica que encaminaremos al controlador principal para poder extraer el porcentaje de carga de la batería.

De los conectores fastón se realiza una conexión al transformador DC/DC de 12 a 5 V. La salida de este transformador se utilizará para alimentar los componentes del control manual, la Raspberry y los controladores que funcionan a 5 V.

Los conectores A0, A1 y BOTÓN corresponden a los potenciómetros deslizantes y al interruptor de cambio de modo, con su correspondiente resistencia de 10 kΩ, que irán incrustados en la caja estanca central.

El conector USB Raspberry irá conectado al puerto micro-usb de la Raspberry Pi a través de un cable de cuatro hilos y será el encargado de alimentarla.

Los dos conectores de cuatro pines restantes, USB Maestro y ANALOGICAS se conectarán a través de un cable de ocho hilos con el PCB Maestro.

Seguidamente se adjunta la evolución del diseño desde cero, hasta el montaje final con los componentes descritos anteriormente soldados y dispuestos para su uso.

Para la descripción de los demás PCB seguiremos el mismo patrón empleado en este apartado.

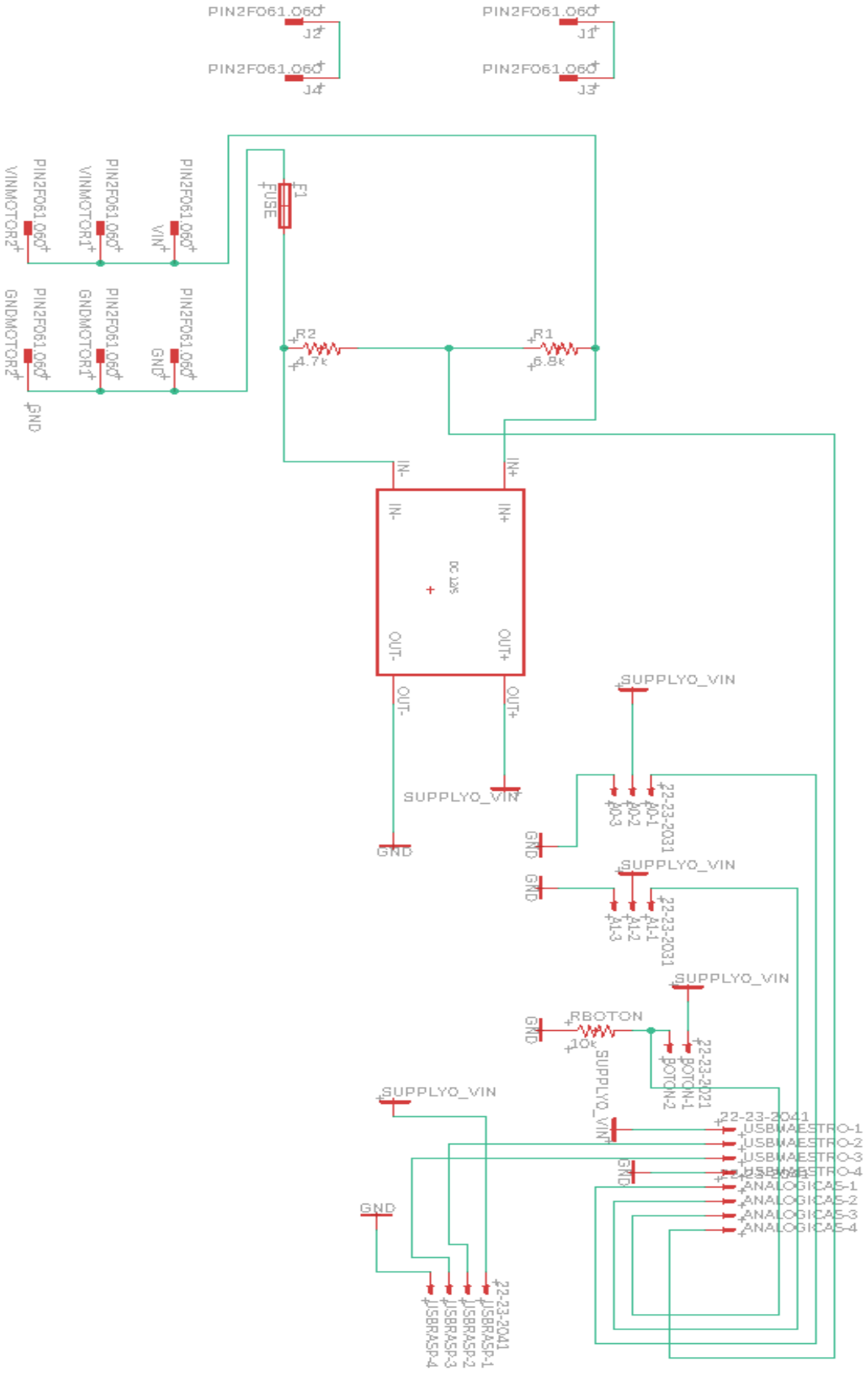

Figura 39: Esquemático PCB Central EAGLE

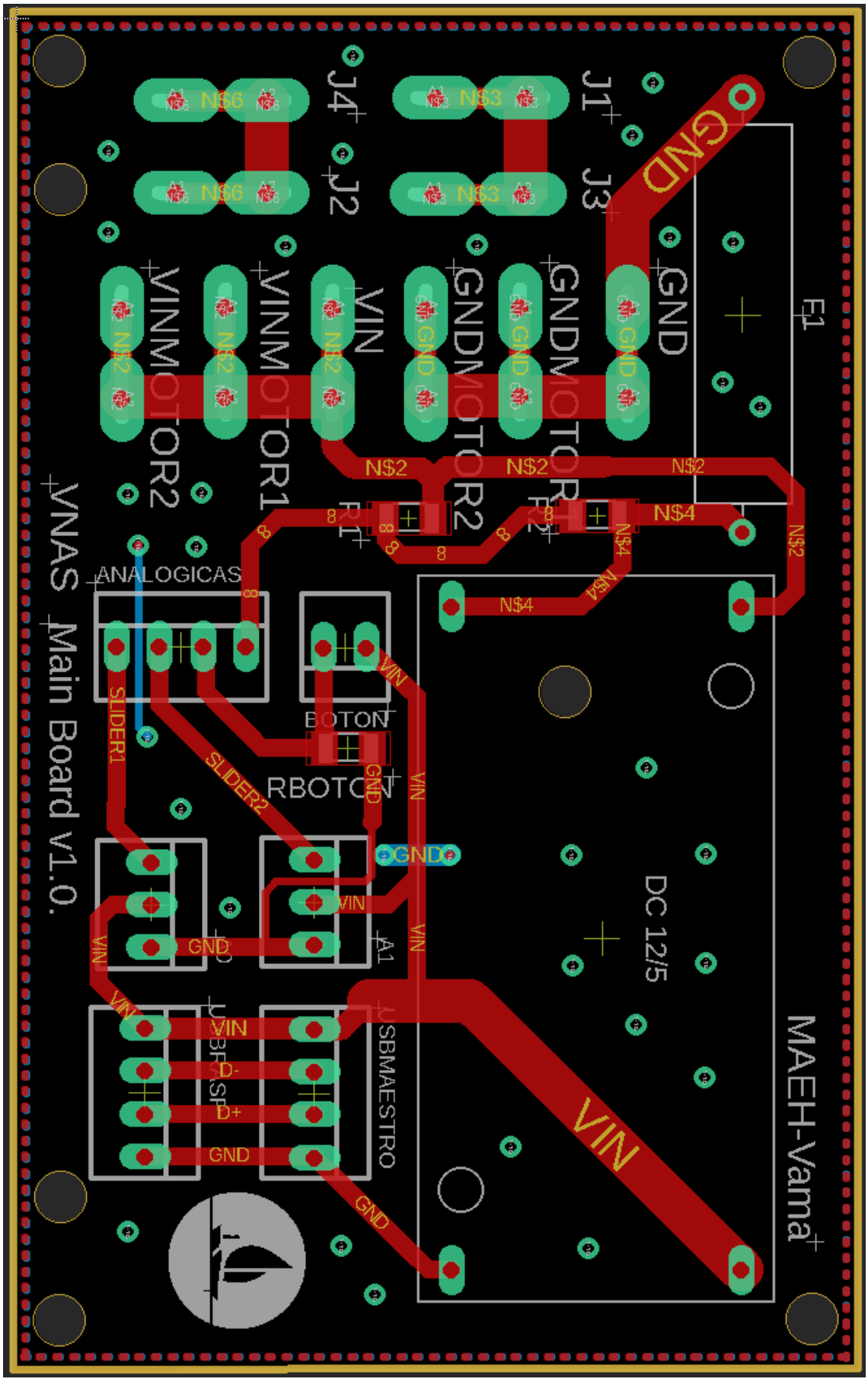

Figura 40: Board PCB Central EAGLE

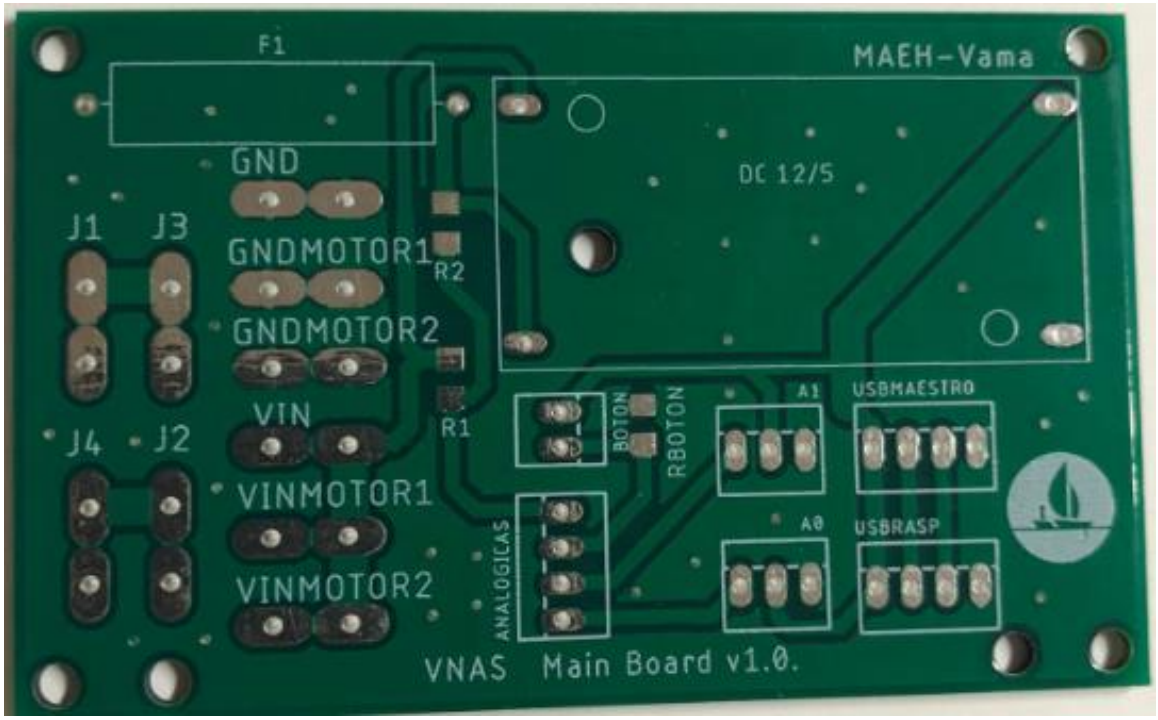

Figura 41: Prototipo Main Board v1.0

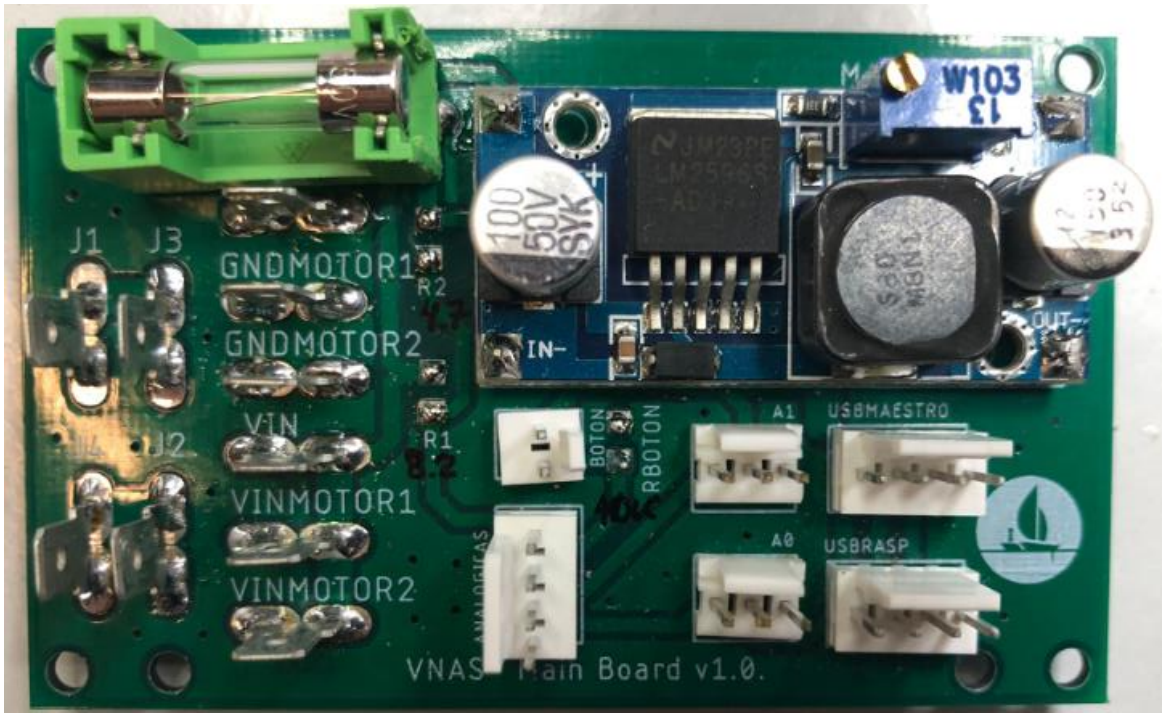

Figura 42: Main board v1.0 (Montaje completo)

5.2.2 PCB ARDUINO MAESTRO

El componente principal, es el llamado Arduino-Micro, al que conectamos las señales analógicas que debemos tratar en el desarrollo software, los pines de conexión I2C, y las señales del driver necesarias para el control PWM del motor.

El otro componente incorporado es el Driver, al cual conectaremos con la misma GND que el controlador, el pin de medición de consumo del motor, el pin de control PWM y el pin de falla.

Los conectores de cuatro pines ANALOG y DATOS USB provienen de la PCB Central, vienen encaminados por un cable de ocho hilos.

Los cuatro pines que desembocan en el conector ANALOG corresponden a:

- Señal Slider 1
- Señal Slider 2
- Señal de botón
- Señal de control de baterías

Y, por otra parte, los pines correspondientes a DATOS USB son:

- Alimentación 5 V
- GND
- Bus de Datos +
- Bus de Datos -

El conector ESCLAVO de cuatro pines se utilizará para llevar a cabo la conexión con el controlador secundario (Arduino Esclavo), y con esos pines alimentarlo y conectarlo a otros pines para llevar a cabo el protocolo de comunicación I2C. Lo conforman estas líneas de conexión:

- Alimentación 5 V
- GND
- SDA (I2C)
- SCL (I2C)

El driver irá alimentado por la señal proveniente del PCB Central que desemboca en los conectores de tipo fastón, que pasarán por un fusible para proteger el circuito.

Se realiza una pequeña **modificación**, conectando por medio de un cable al controlador la señal de alimentación de 5 V que recibimos de parte del PCB Central. Esta acción se realiza para que, en caso de que la CPU Central se desconecte, nuestro controlador principal siga alimentado para poder usar el modo de navegación manual-

Seguidamente se adjunta la evolución del diseño desde cero, hasta el montaje final con los componentes descritos anteriormente soldados y dispuestos para su uso.

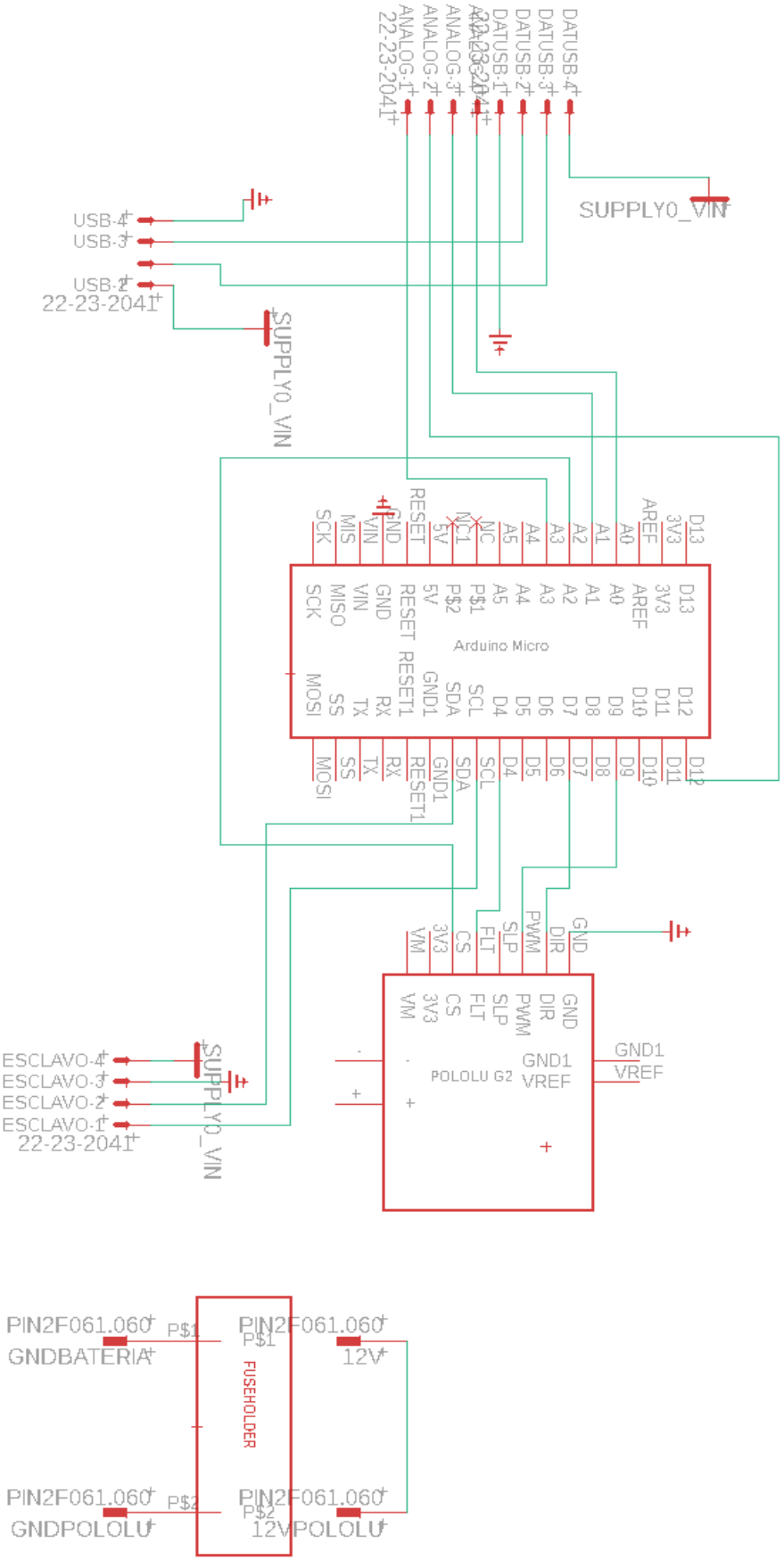

Figura 43: Esquemático PCB Arduino Maestro EAGLE

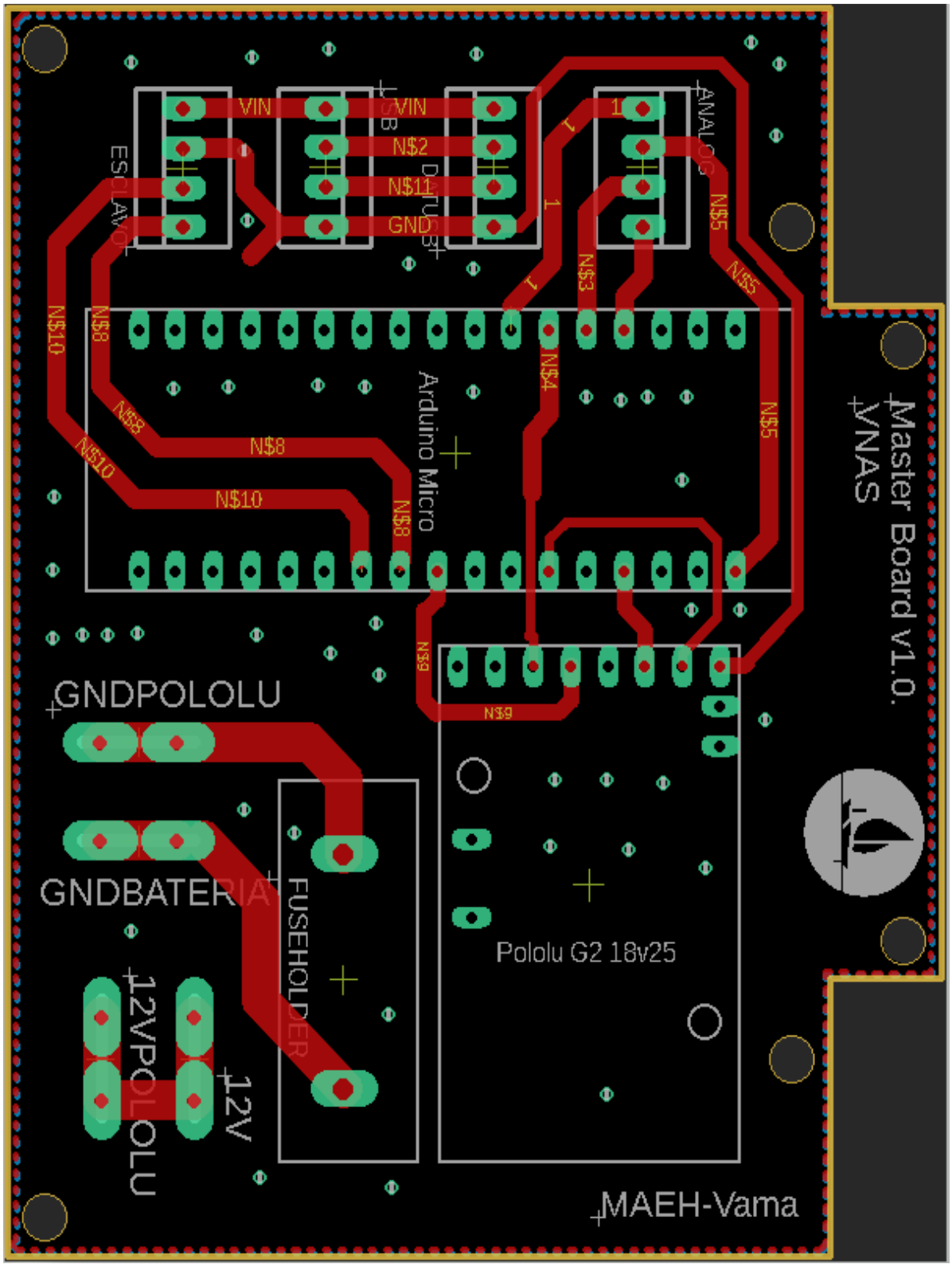

Figura 44: Board PCB Arduino Maestro EAGLE

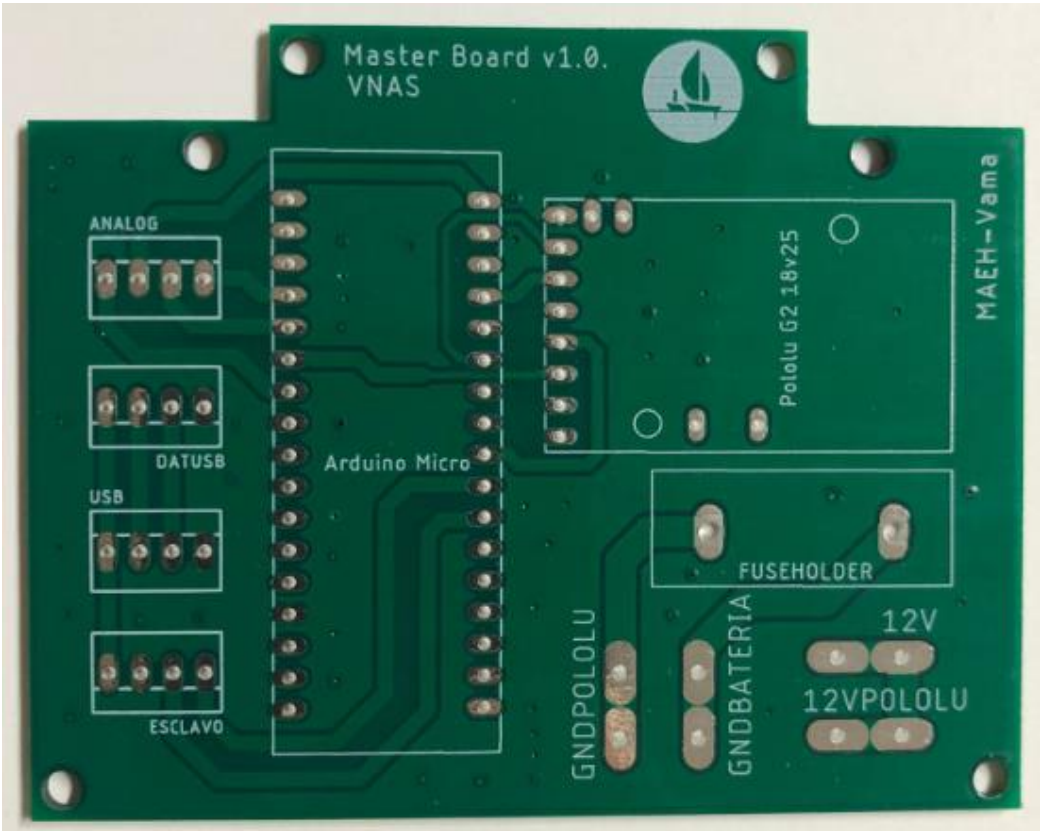

Figura 45: Prototipo Master Board v1.0

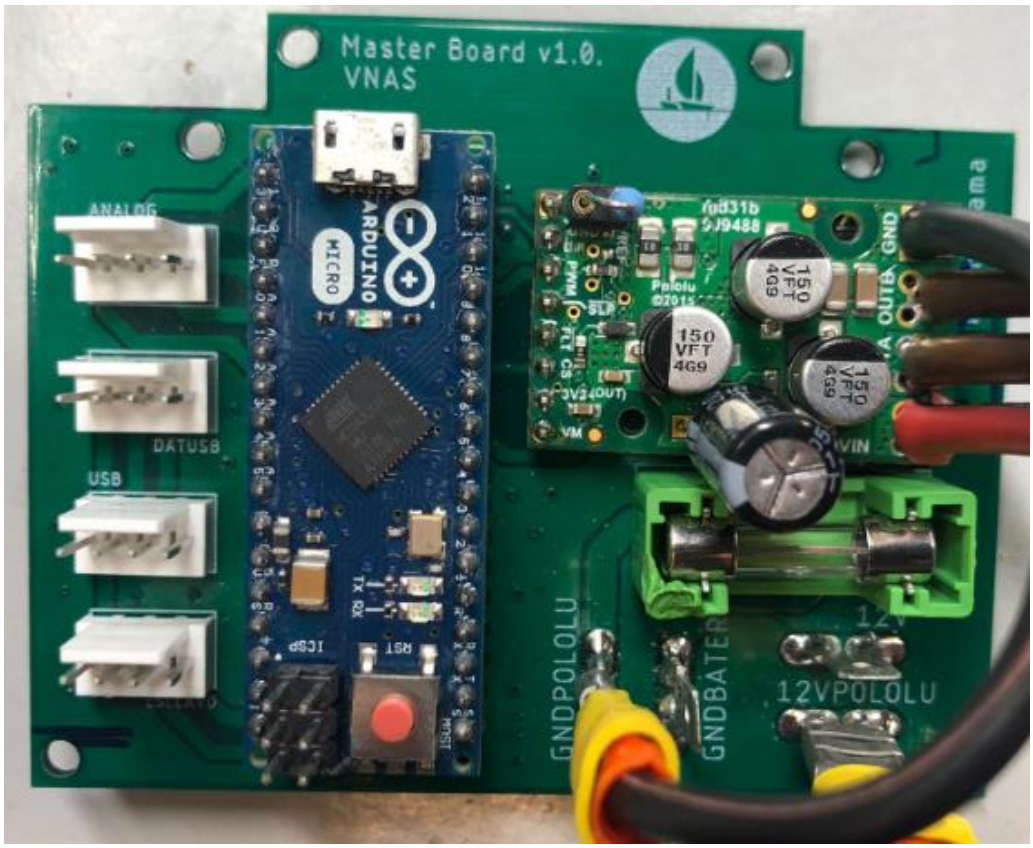

Figura 46: Master Board v1.0 (Montaje completo)

5.2.3 PCB ARDUINO ESCLAVO

El componente principal, es el llamado Arduino-Micro, al que conectamos los pines de conexión I2C, y las señales del driver necesarias para el control PWM del motor.

El otro componente incorporado es el Driver, al cual conectaremos con la misma GND que el controlador, el pin de medición de consumo del motor, el pin de control PWM y el pin de falla.

El único conector de cuatro pines es proveniente de la PCB Arduino Maestro, el cual lleva datos de la comunicación I2C y los pines de alimentación y masa necesarios para alimentar el controlador. Estas líneas de conexión son:

- Alimentación 5 V
- GND
- SDA (I2C)
- SCL (I2C)

El driver irá alimentado por la señal proveniente del PCB Central que desemboca en los conectores de tipo fastón, que pasarán por un fusible para proteger el circuito.

Seguidamente se adjunta la evolución del diseño desde cero, hasta el montaje final con los componentes descritos anteriormente soldados y dispuestos para su uso.

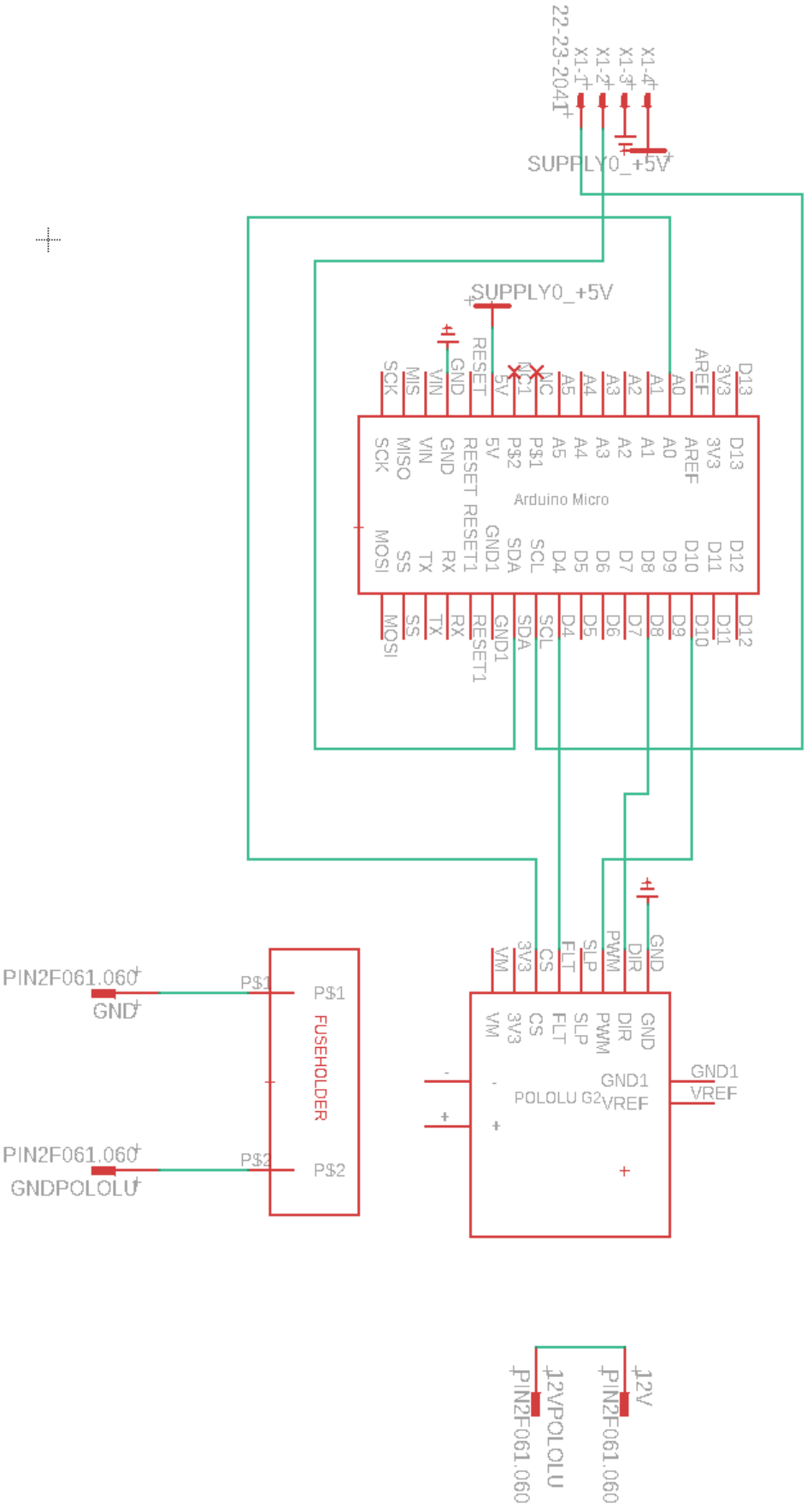

Figura 47: Esquemático PCB Arduino Esclavo EAGLE

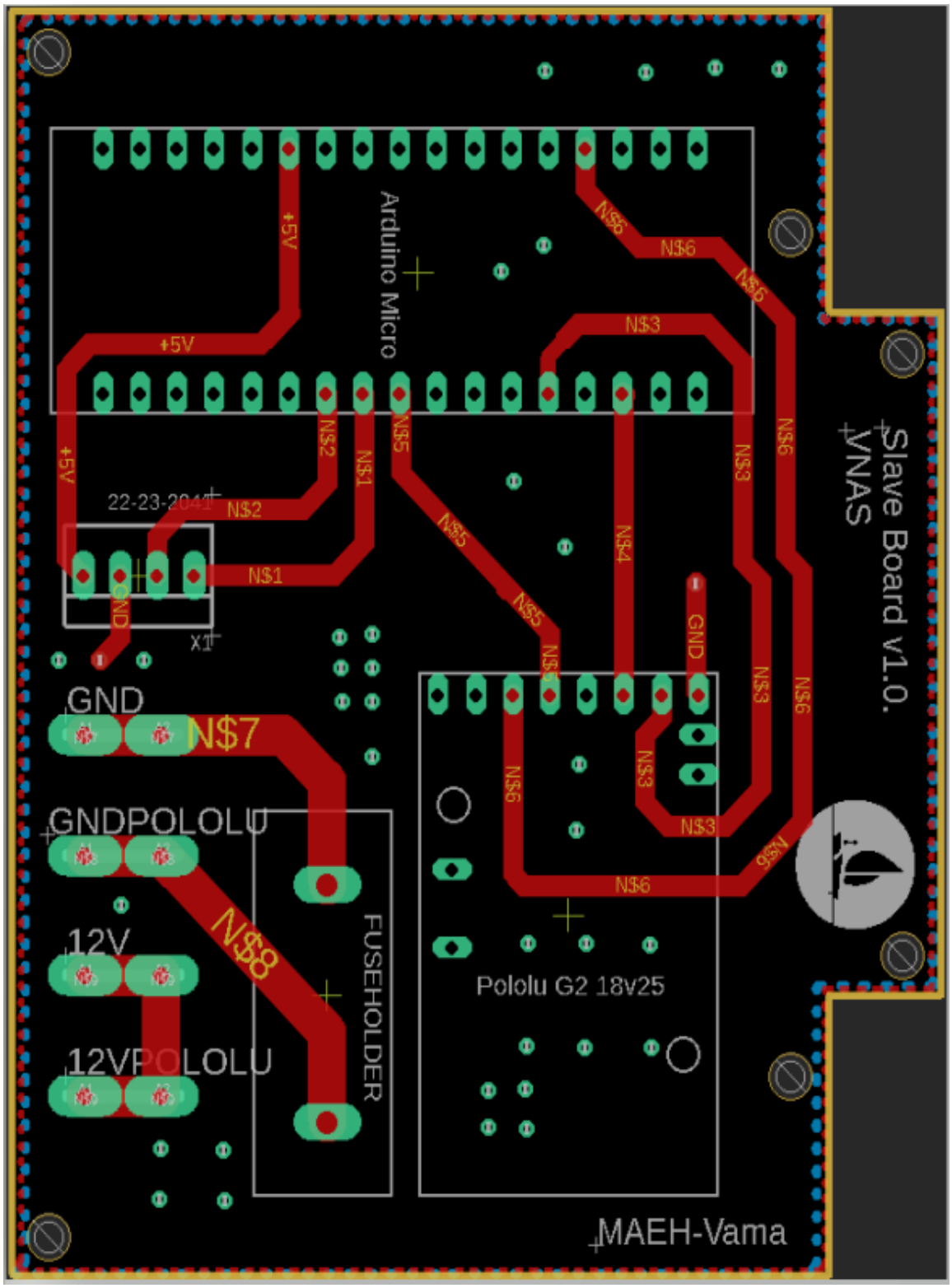

Figura 48: Board PCB Arduino Esclavo EAGLE

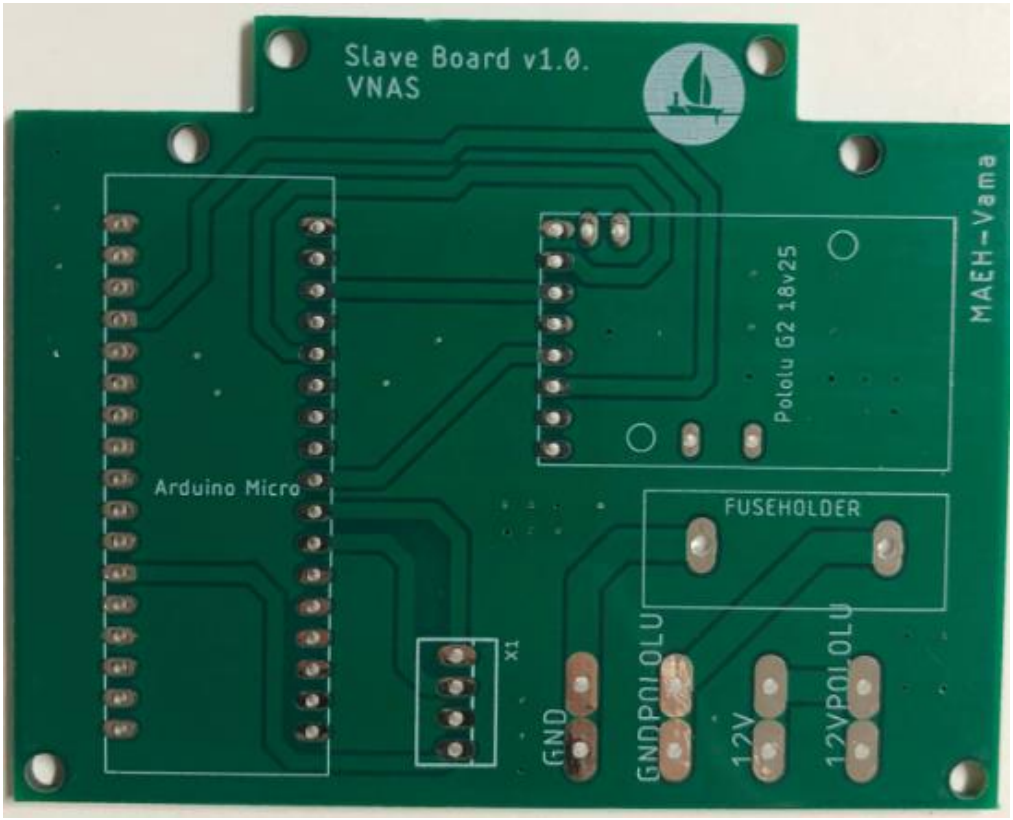

Figura 49: Prototipo Slave Board v1.0

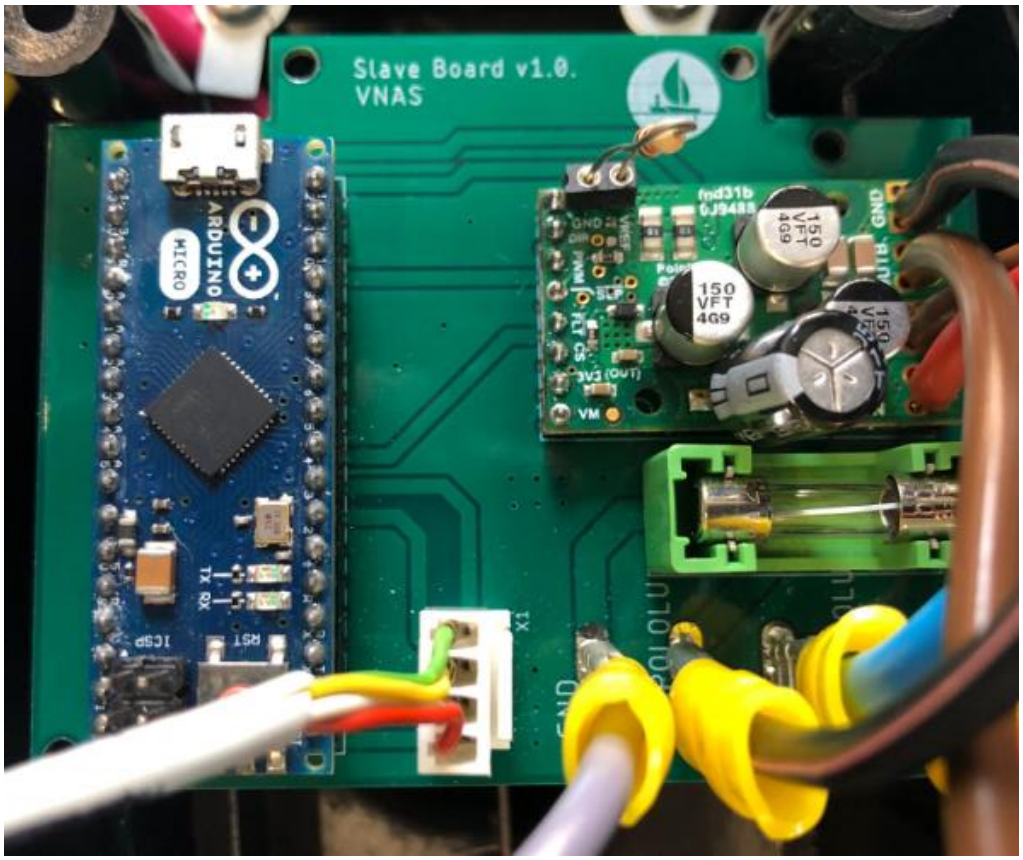

Figura 50: Slave Board v1.0 (Montaje completo)

5.3 MONTAJE CAJA DE CONTROL

En este apartado podemos observar el montaje final obtenido de la caja estanca central desde la cual controlamos el sistema.

Vemos en la siguiente figura que se ha incrustado dentro de la caja el PCB Central, la seta de emergencia y el interruptor que da alimentación a la CPU Central.

El cableado se ha aislado dentro de la caja a través del uso de prensaestopas para evitar que acceda agua al interior.

Podemos observar que esta caja es la encargada de gestionar todo el sistema de energía y alimentación del sistema.

Figura 51: Montaje interior de la caja de control

A continuación, analizaremos la tapa de la caja donde también llevará algunos componentes incrustados.

Podemos apreciar los sliders y el interruptor correspondiente al modo de control de navegación. Estos componentes se han anclado mediante unas ranuras creadas en la tapa y su sujeción se realiza a partir de unos soportes creados a medida (en el caso de los sliders).

Sus conexiones están realizadas con el PCB Central.

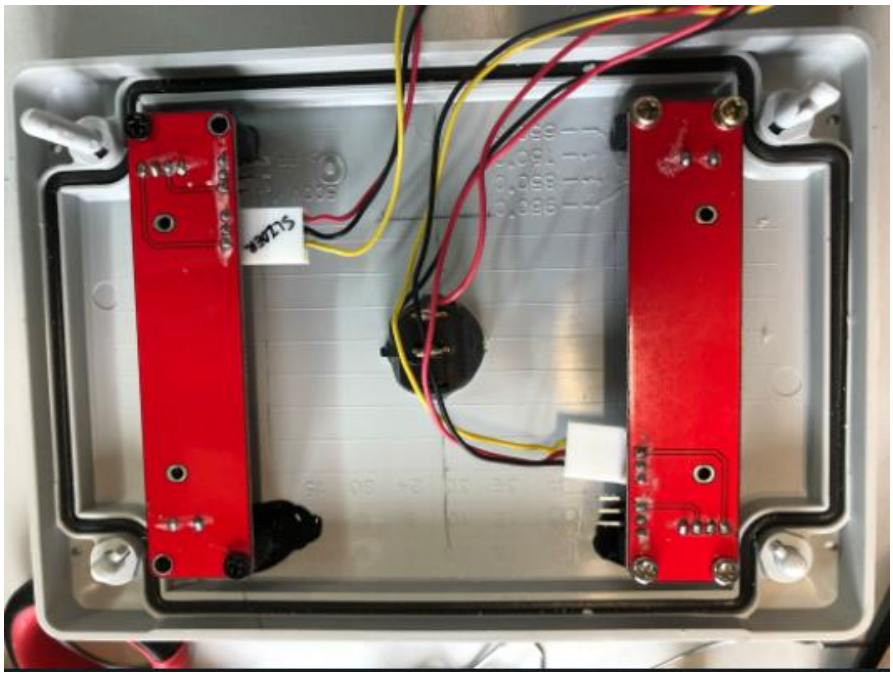

Figura 52: Montaje de sliders e interruptor en la tapa de control

Finalmente, observamos el resultado final de este montaje con la caja ya cerrada. Se han realizado las indicaciones correspondientes a cada ranura de los sliders, marcando como “0” el estado parado de los motores, “F” con el significado de forward y “R” con el significado de reverse para conocer en que sentido están girando los motores.

Las indicaciones correspondientes al interruptor, en el caso de la letra “A” indica que corresponde al modo automático de navegación, y para el caso de la letra “M” indica que corresponde al modo manual de navegación.

Por último, vemos a la derecha de la imagen la seta de emergencia.

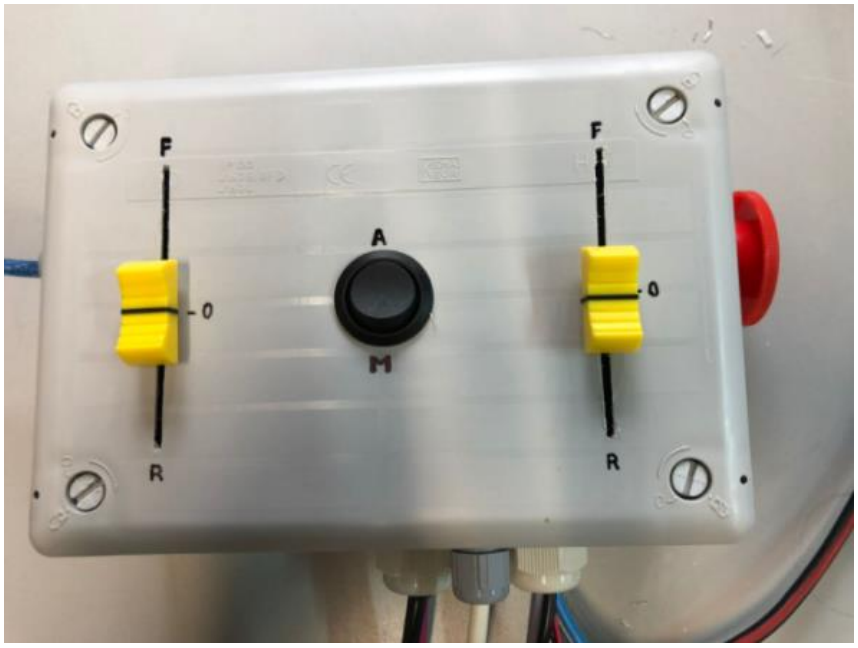

Figura 53: Imagen final de la caja de control

6. DESARROLLO DE LA APLICACIÓN DE CONTROL

Se procede a explicar todo el desarrollo software dividido en protocolos de comunicación, modos de control de navegación y la información proporcionada a la Raspberry Pi.

A continuación, se muestra un pequeño resumen de los protocolos de comunicación empleados para el control de los motores a partir de la caja de control y el sistema de navegación.

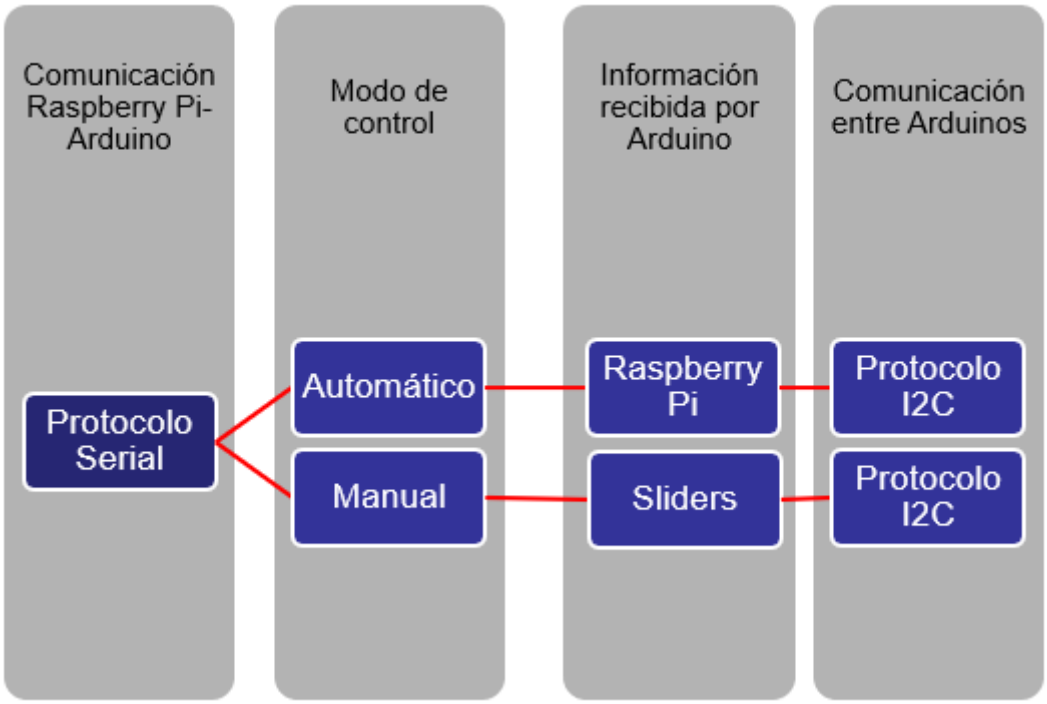

Figura 54: Resumen de protocolos de comunicación usados en función del modo de control

En este resumen podemos observar que, para poder usar el sistema de navegación junto con la Raspberry debemos estar conectados mediante protocolo serial. Seguidamente, el modo de control que se quiera usar depende únicamente del usuario, los dos modos son compatibles sin necesidad de desconectar el enlace serial entre nuestro controlador principal y la Raspberry.

De la misma forma, cuando el modo seleccionado es el automático, el controlador principal recibirá los datos por parte de la Raspberry, y al contrario, si el modo de control es manual, el controlador principal recibirá los datos de parte de los sliders.

Una vez que el controlador principal recibe los datos, independientemente del modo de control, éste realiza el tratamiento de los datos y le comunica el valor correspondiente al motor secundario mediante el protocolo de comunicación I2C.

6.1 COMUNICACIÓN POR I2C ENTRE AMBOS CONTROLADORES

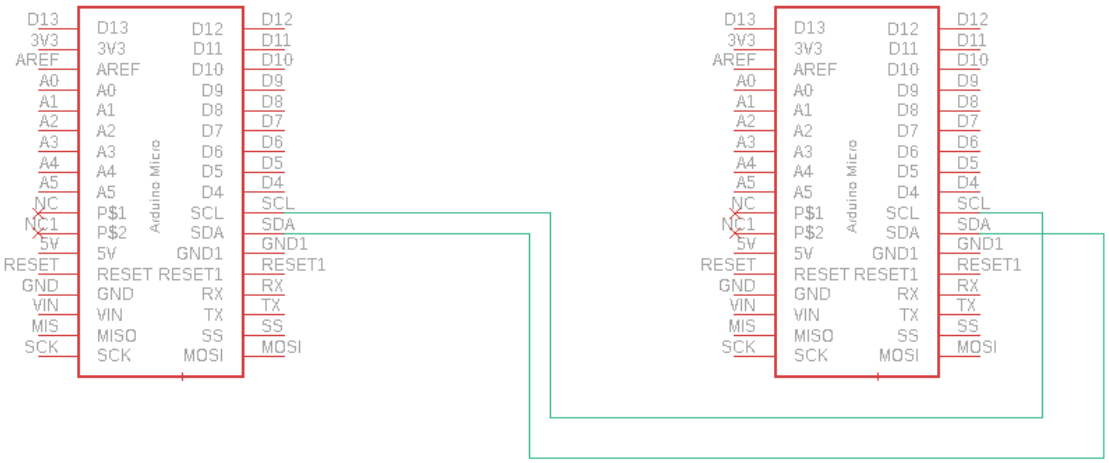

Figura 55: Conexionado I2C EAGLE

La comunicación mediante el protocolo de comunicación I2C se lleva a cabo mediante dos hilos. En estos dos hilos se pueden conectar diferentes periféricos, o también dos arduinos entre sí como se ha realizado en este proyecto.

Los dos hilos que forman el bus de comunicación son el SDA (señal de datos) y el SCL (señal de reloj).

Existen dos denominaciones, los maestros y los esclavos.

En este proyecto se utilizará un controlador Arduino Maestro y un controlador Arduino Esclavo.

Vamos a describir la información que se envía y recibe entre ambos.

Ahora vamos a describir las fases de la comunicación apoyándonos en el código utilizado de forma específica.

Lo primero que se debe realizar es, declarar la librería Wire, ya que sin esta no se podría efectuar la comunicación. Para ello se utiliza el comando “include <Wire.h>”.

A continuación, vemos el código utilizado para cada controlador.

- ARDUINO MAESTRO

El dato que enviamos desde nuestro maestro al esclavo es el valor numérico en formato analógico de la velocidad del motor controlado por el esclavo, es decir, un valor comprendido entre 0 y 1023. Para facilitar la comunicación, se envía el dato en forma de string, como una cadena de caracteres que beneficiará la transmisión.

Vemos el proceso a continuación.

```
//-------------ENVÍO SPEED SET A ESCLAVO---------
    Wire.beginTransmission(10);//inicializamos la transmisión i2c al
esclavo 1
  for(byte i=0;i<=Speed_i2c.length();i++){
    Wire.write(Speed_i2c[i]); //le enviamos carácter por carácter
  }
  Wire.endTransmission(10);
```

Código 1: Envío de datos maestro-esclavo I2C

Al igual que enviamos el dato descrito anteriormente, también necesitamos recibir otro dato de parte de nuestro esclavo. Esto se lleva a cabo a partir de la recepción mediante un bucle en el que vamos iterando y guardando en una cadena los datos que voy recibiendo, para luego mostrarlo por vía serie a la Raspberry. El dato que recibimos por parte de nuestro esclavo es el valor de la corriente (Amperios) consumida por el motor. El mecanismo de recepción consiste en recibir dato a dato (sabiendo el número de caracteres que conforma la cadena) e ir almacenándolo en otro string, por lo que, al terminar todas las iteraciones, obtenemos el dato en formato string y no hay que darle ningún otro tratamiento, estaría listo para enviarlo a la raspberry a través de la conexión serie.

```
//--------------RECEPCION DE DATOS I2C------------
    Wire.requestFrom(10,4);
  byte j=0;
  while(Wire.available()>0){
    current2[j] = Wire.read();
    j++;
  }
```

Código 2: Recepción de datos maestro-esclavo I2C

- ARDUINO ESCLAVO

Al igual que anteriormente hemos descrito el comportamiento de la comunicación en el código del Arduino maestro, para el esclavo es similar, pero en este caso, los datos que enviábamos desde el maestro, los recibimos por el esclavo, lo que quiere decir que los datos de emisión y recepción se intercambian.

Dicho esto, el dato que envía el esclavo es el valor de la corriente que circula por el motor, que se envía de la misma forma que lo realiza el maestro, es decir, guardando el valor en un string y lo vamos enviando dato a dato.

```
//----------------------ENVIAR DATOS I2C A MAESTRO----------------
void MandaCorriente()
{
  //Wire.beginTransmission(1);
  for(byte i=0;i<=current_mA.length();i++){
    Wire.write(current_mA[i]);//Envio caracter por caracter
}
  //Wire.endTransmission();
}
```

Código 3: Envío de datos esclavo-maestro I2C

En el caso de recepción el valor que recibimos es el valor numérico en formato analógico de la velocidad del motor, que extrae el maestro de la información recibida, ya sea por los potenciómetros o por la raspberry.

El proceso de recepción es similar al de nuestro maestro, salvo por una excepción. Este valor que recibimos como string hay que convertirlo a un entero para darle el posterior tratamiento e interpretarlo como hemos descrito aquí.

Lo podemos observar aquí.

```
//-------------------RECIBIR DATOS I2C DEL MAESTRO-------------
void datoRecibido()
{
while(!Wire.available()){
      Speed_Set2=512;
    }
  while(Wire.available()>0)    {
    for (byte i=0;i<=4;i++){
    mensaje1[i] = Wire.read(); // Recibe la palabra del i2c tx
    }
    Speed_Set = String (mensaje1);
    Speed_Set2= Speed_Set.toInt();
    delay(500);
  }
```

Código 4: Recepción de datos esclavo-maestro I2C

6.2 CONTROL MANUAL DE LA POTENCIA DE CADA MOTOR

Anteriormente hemos señalado que se van a implementar dos modos de control; el control manual y el control por vía serial o comúnmente conocido puerto usb.

En este apartado vamos a profundizar en el control manual, viendo todo el proceso realizado y la parte del código que le corresponde específicamente.

Para este modo de control, se utilizan dos potenciómetros deslizantes; y un interruptor para cambiar de modo de lectura. Ambos componentes descritos anteriormente.

La parte del código correspondiente al funcionamiento del interruptor la vemos en el siguiente cuadro de texto.

```
void loop() {
  serie_manual = digitalRead(PIN_SELECT_MODO);
   if (serie_manual == HIGH){
    Captura_Datos_Pot();
    Modo="1";
    }
   else {
      Captura_Datos_Serie();
      Modo="0";
      }
 delay(100);
```

Código 5: Funcionamiento del botón de modo de control de lectura

El interruptor de cambio de modo está configurado para que, en estado abierto, el modo de recepción de datos sea mediante el puerto serie.

De esta forma, si accionamos el interruptor y cerramos el circuito, el modo de lectura cambiará a modo manual, y viceversa si realizamos la misma acción.

Esto nos permite ir alternando de un modo a otro y nos ofrece flexibilidad.

Los potenciómetros son entradas analógicas que corresponden a cada motor, van a enviar su señal a nuestro Arduino Maestro, por lo que el Esclavo no tiene que realizar nada para interpretar la información que proviene de estos dos componentes. Desde nuestro Maestro gestionamos la información y se la hacemos saber al Esclavo mediante el protocolo I2C descrito en el apartado anterior.

```
void Captura_Datos_Pot(void){
    pot_read1=analogRead(A0);
    pot_read2=analogRead(A1);
    if (pot_read1 <540 && pot_read1>480) {
    Speed_Set1=511;
  }
   else{
    Speed_Set1 = pot_read1;
   }
   //Serial.println(Speed_Set1);
   if (pot_read2 <540 && pot_read2>480) {
    Speed_Set2=511;
  }
   else{
    Speed_Set2 = pot_read2;
   }
   Speed_i2c = String (Speed_Set2);
}
```

Código 6:Funcionamiento de modo de lectura de datos manual

Como vemos en el código anterior, cuando llego a esta rutina, el proceso que se realiza inicialmente es la lectura del valor de las entradas analógicas correspondientes a los dos potenciómetros, y además imponemos una condición. Esta condición se realiza porque el potenciómetro esta calibrado para que en la zona intermedia el motor este parado, en la zona superior el motor adquiera el máximo de velocidad en un sentido, y en la zona inferior el motor adquiera el máximo de velocidad en el otro sentido.

Por esa razón, partiendo de que 511 sería el valor medio para que el motor este parado, realizamos una extensión para un rango de valores de [480-540] para que podamos ser capaces con la mano humana de poder situar el marcador entre esos niveles para que el motor se pare. Esto se realiza ya que sería prácticamente imposible llegar al valor justo de 511 para que el motor se detenga. Una vez realizada esa condición, mientras el valor de lectura no se encuentre entre el rango de valores descrito previamente, lo guardaremos en una variable de tipo entero con su valor correspondiente sin alterarlo.

Seguidamente para finalizar la rutina, preparamos el dato convirtiéndolo en una cadena de caracteres para enviarlo por protocolo de comunicación I2C.

6.3 CONTROL DE LOS MOTORES CON RASPBERRY PI VIA SERIAL

En este apartado describiremos el código empleado para recibir e interpretar la información de los motores que recibimos desde la Raspberry y a su vez, la información que nosotros le ofrecemos desde nuestro Arduino Maestro.

```
void Captura_Datos_Serie(void){
  // Recepcion comando SERIAL

  if (Serial.available() > 0)
  {
    t=millis();
    String str = Serial.readStringUntil('\n');
    Motor1=str.substring(0,4);
    Speed_Set1=Motor1.toInt();
    Motor2=str.substring(5,9);
    Speed_Set2=Motor2.toInt();
    Speed_i2c= String (Speed_Set2);
  }
  else {
    if ((millis()-t)>=10000) {//SI NO RECIBO DATOS TEMPORIZO Y
PONGO SPEED A 0
      Speed_Set1=511;
      Speed_Set2=511;
      Speed_i2c= String (Speed_Set2);
    }
  }
}
```

Código 7: Funcionamiento de modo de lectura de datos automático-puerto serie

Al entrar en la rutina, lo primero que realizamos es comprobar si la comunicación está habilitada y si tenemos información. En el caso de que esto suceda, guardaremos en un string toda la cadena de datos recibida por parte de la Raspberry.

Esta cadena está conformada por el valor analógico de la potencia de cada motor, separados por una coma.

Por ejemplo:

0958,1023

Esta cadena de datos estaría formada por el valor de Speed_Set1 que seria 958 y el valor de Speed_Set2 que seria 1023.

En esta rutina lo que se realiza es el desglosamiento de la cadena de datos y a su vez los convierte en datos de tipo entero para su posterior tratamiento.

Como hemos visto en el apartado anterior, en el caso de la variable correspondiente al Arduino Esclavo la enviamos por I2C como un string.

También se puede observar la llamada de la función “millis()”.

Esta función es utilizada con el propósito de prevenir una mala interpretación de los datos. Nos permite establecer un temporizador, que es usado para que; en el caso de que la Raspberry falle o no pueda enviar información, si nuestro Arduino no recibe información por el puerto serie durante diez segundos, automáticamente establezca el valor analógico de ambos motores como parado.

Ahora vamos a ver qué información enviamos mediante el puerto serie a la Raspberry.

```
current2_A= String(current2);
current1_A=String(current1);
leer_voltios();
Carga_bateria= String(porcentaje);
String Serie= (Carga_bateria+","+current1_A+","+current2_A+","+Modo);
Serial.println(Serie);
```

Código 8: Información enviada por el puerto serie

Los datos que se envían son los referentes a la capacidad de carga que tienen las baterías, la corriente que circula por los dos motores y el modo de lectura de datos en el que estamos en ese momento, ya sea manual o serie.

Estos datos se envían mediante una cadena de strings, que forman el string llamado “Serie”. Cada dato está separado por una coma del siguiente.

Vamos a ver como obtenemos el valor de los parámetros que monitorizamos para enviarlos a la Raspberry.

- Porcentaje de baterías: Se obtiene mediante la función “leer_voltios()”.

```
void leer_voltios()
{

bateria= (analogRead(A3));
if (bateria > 940)
  bateria = 940;
porcentaje= map (bateria,879,940,0,100);

}
```

Código 9:Método de obtención de porcentaje de baterías

El porcentaje de las baterías lo obtenemos a partir de un divisor de tensión situado en nuestro PCB Central. Lo que hacemos es medir el voltaje entregado por la batería, y para ello hacemos un divisor de tensión, para garantizarnos que la entrada analógica con la que medimos ese voltaje entregado no supera los 5 voltios, ya que si esto ocurriera se estropearía el Arduino.

Como vemos en la siguiente figura, el divisor de tensión está compuesto por dos resistencias y un valor de entrada, lo que nos proporciona un voltaje de salida, que es el que entrará al Arduino como entrada analógica.

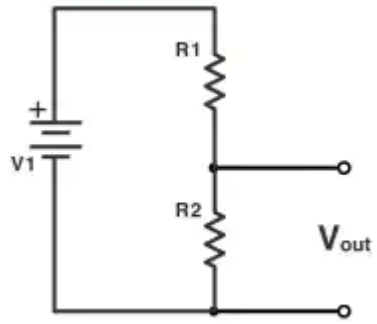

Figura 56: Esquemático divisor de tensión

El valor de R1 es 8200 Ω, el valor de R2 es 4700 Ω y el valor de voltaje de entrada es de 15 V.

Las baterías que se utilizan en este proyecto solo pueden entregar como máximo 12 V, pero por precaución, se ha aumentado ese valor a 15 V.

Entonces utilizando como base las siguientes ecuaciones obtenemos una serie de valores.

$$V_{out} = \frac{V_1 \times R_2}{(R_1 + R_2)} \quad (1)$$

Siendo V_{out} el voltaje a la salida del divisor de tensión, V_1 el parámetro variable del voltaje de entrada asociado a la capacidad de las baterías, $R_1 = 8200\ \Omega$ y $R_2 = 4700\ \Omega$.

$$A_3 = \frac{V_{out} \times 1023}{5} \quad (2)$$

Siendo A_3 el valor analógico que lee la entrada del Arduino, V_{out} obtenido de la ecuación (1), 1023 corresponde al limite superior de la lectura analógica del Arduino y 5 corresponde al valor de voltaje máximo que puede soportar el Arduino.

Teniendo en cuenta esta relación de porcentaje y capacidad.

Estimación de la capacidad según el voltaje	
Voltaje de los conectores	**Capacidad aproximada**
12,65 V	100 %
12,45 V	75 %
12,24 V	50 %
12,06 V	25 %
11,89 V	0 %

Figura 57: Relación voltaje-capacidad [38]

Sustituimos los valores para los dos límites de carga en la ecuación (1), utilizando 12.6 V para el 100% y 11.9 V para el 0%.

- Para 12.6 V y el valor de las resistencias definido anteriormente:

$$V_{out} = \frac{12.6 \times 4700}{(8200 + 4700)} = 4.6\,V \qquad (1.1)$$

$$A_3 = \frac{4.6 \times 1023}{5} = 940 \qquad (2.1)$$

- Para 11.9 V y el valor de las resistencias definido anteriormente:

$$V_{out} = \frac{11.9 \times 4700}{(8200 + 4700)} = 4.3\,V \qquad (1.2)$$

$$A_3 = \frac{4.3 \times 1023}{5} = 879 \qquad (2.2)$$

De esta manera se justifica el código descrito anteriormente con los cálculos realizados.

- Valor de la corriente: Se obtiene a partir del controlador del motor con un pin conectado a una entrada analógica de cada Arduino.

```
analogReference(INTERNAL);
cs=analogRead(CSENSE);
current1=((cs*1.1/1023)-0.05)*100;
```

Código 10: Método de obtención de valor de corriente

Para obtener el valor de la corriente, leemos el puerto analógico de nuestro Arduino declarado como CSENSE.

El pin sensor de corriente del conductor, CS, produce un voltaje proporcional a la corriente del motor. El voltaje de salida es de unos 10 mV/A más un pequeño desvío, que suele ser de unos 50 mV. [6]

Esa es la conversión que realizamos y asignamos a la variable “current1”.

- Modo de control: Obtenemos el tipo de control aprovechando la rutina de funcionamiento de nuestro botón descrita anteriormente aquí.

```
if (serie_manual == HIGH){
    Captura_Datos_Pot();
    Modo="1";
    }
   else {
      Captura_Datos_Serie();
      Modo="0";
      }
```

Código 11: Determinación del modo de control empleado

Esto significa que, cuando el interruptor representado por la variable “serie_manual” esté abierto (estado lógico ‘0’), la variable “Modo” pasará a valer cero, lo que representa que estamos capturando los datos por el puerto serie. Y si está con valor lógico ‘1’ nuestra variable “serie_manual”, es decir, cerrado el circuito para el interruptor; la variable “Modo” pasará a valer uno, que representa que estamos capturando datos manualmente.

6.4 CONEXIÓN DE CONTROL ENTRE ARDUINO Y CONTROLADOR POLOLU

Para comenzar a explicar este apartado, mostramos en la siguiente figura el esquemático específico necesario para conectar el controlador al Arduino. Los tres pines que vemos conectados son los que se utilizan en este proyecto para el control del motor y obtener parámetros.

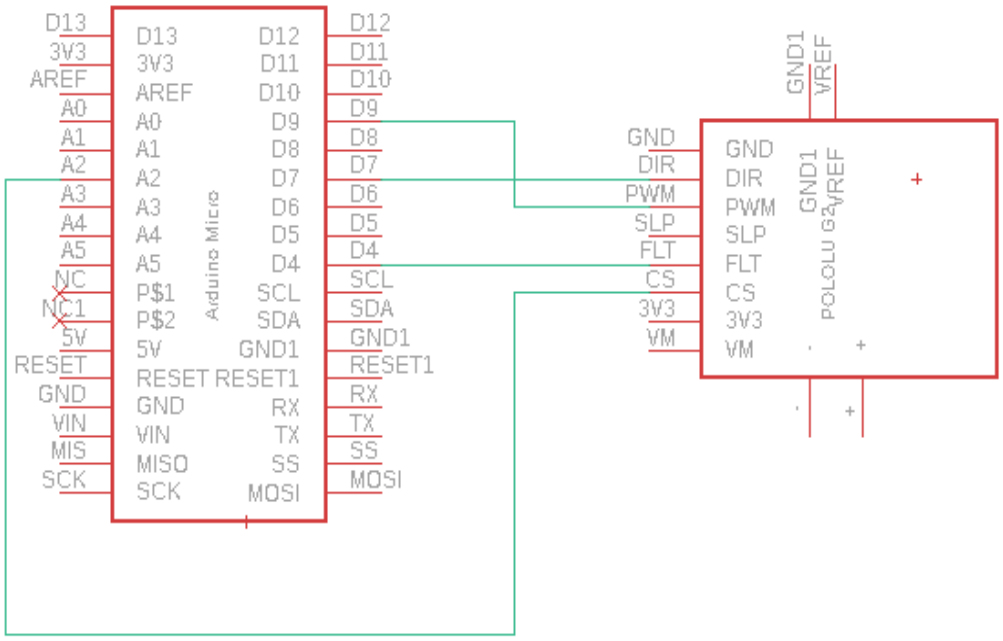

Figura 58: Conexionado Arduino-Pololu en EAGLE

Respecto a los parámetros, en el apartado anterior ya se ha descrito el método de obtención del único de ellos; el valor de la corriente con el pin CS. Por esa razón vamos a centrarnos en describir puramente el control del motor, sus cambios de dirección y potencia asignada.

En la siguiente figura observamos la tabla de verdad y el conjunto de asignaciones que se tienen que llevar a cabo para controlar el sentido del motor, que bien puede ser en sentido horario, antihorario, o parado.

Tabla de verdad del controlador de motor

PWM	DIR	OUTA	OUTB	Operación
H	H	H	L	Adelante
H	L	L	H	Contrarrestar
L	X	L	L	Freno

Figura 59: Tabla de verdad controlador del motor [13]

Esta tabla de verdad la hemos implementado en el programa de la siguiente manera.

```
if (current1>=0 && current1 <= 60) {
    if (Speed_Set1 > 511){
      digitalWrite(DIR1,HIGH);
      Timer1.pwm(PWM_motor1, (Speed_Set1*2)-1023);
      PWM_motor1_int= (Speed_Set1*2)-1023; //-- duty 0 to 1023.

}
    else if (Speed_Set1<511){
      digitalWrite(DIR1,LOW);
      Timer1.pwm(PWM_motor1, 1023-Speed_Set1*2);
      PWM_motor1_int= 1023-Speed_Set1*2; //-- duty 0 to 1023.

    }

     }
else {
     Timer1.pwm(PWM_motor1, 0);
     PWM_motor1_int=0;
    }


  // FLT - Activation : Se parara la generacion de PWM

    if ( digitalRead(FLT) == LOW){
      Timer1.pwm(PWM_motor1, 0);
      }
```

Código 12: Control del motor

En primer lugar, para proteger el motor establecemos un umbral de corriente siguiendo las recomendaciones del fabricante. Este umbral está comprendido entre los valores 0 y 60 A. Mientras la corriente del motor no esté dentro de esos valores, automáticamente el motor estará en estado parado y no se pondrá en funcionamiento.

Seguidamente comprobamos el valor analógico de la velocidad del motor, y en función de este valor pondremos el pin DIR a un nivel lógico alto o bajo. De esta manera, previo acuerdo con la parte de programación de la Raspberry que estará conectada al puerto serie, esta parte del código nos sirve para el control manual y el control automático, ya que establecemos como parado el valor '511'. De tal forma que el valor '0' sería el correspondiente a la máxima potencia para la operación contrarrestar de la tabla de verdad de la figura anterior, y el valor '1023' sería el correspondiente a la máxima potencia para la operación adelante de dicha tabla de verdad. Esta potencia irá disminuyendo para ambos sentidos progresivamente a medida que se acercan al valor '511'.

Por último, como pin de detección de errores tenemos el FLT. El controlador del motor puede detectar varios estados de falla que reporta al poner la clavija del FLT en posición baja; esta es una salida de drenaje abierto que debe ser llevada al voltaje lógico del sistema. Las fallas detectables incluyen cortocircuitos en las salidas, bajo voltaje y sobretemperatura. Todos los fallos deshabilitan las salidas del motor, pero no están enganchados, lo que significa que el controlador intentará reanudar el funcionamiento cuando se elimine la condición de fallo (o después de un retraso de unos pocos milisegundos en el caso del fallo de cortocircuito). El fallo de sobretemperatura proporciona una débil indicación de que la placa está demasiado caliente, pero no indica directamente la temperatura de los MOSFET, que suelen ser los primeros componentes en recalentarse, por lo que no se debe contar con este fallo para evitar los daños causados por las condiciones de sobretemperatura. [13]

7. SIMULACIÓN Y RESULTADOS

Una vez desarrollada la parte de software y hardware, realizamos las comprobaciones y simulaciones de todos los mecanismos desarrollados para comprobar su correcto funcionamiento antes de hacer pruebas en entorno acuático.

7.1 COMPROBACIÓN DE LA CONEXIÓN ENTRE AMBOS CONTROLADORES

Procedemos a comprobar la comunicación I2C.

Desde el maestro podemos observar que recibimos comunicación por el puerto I2C, sin ningún tipo de problema.

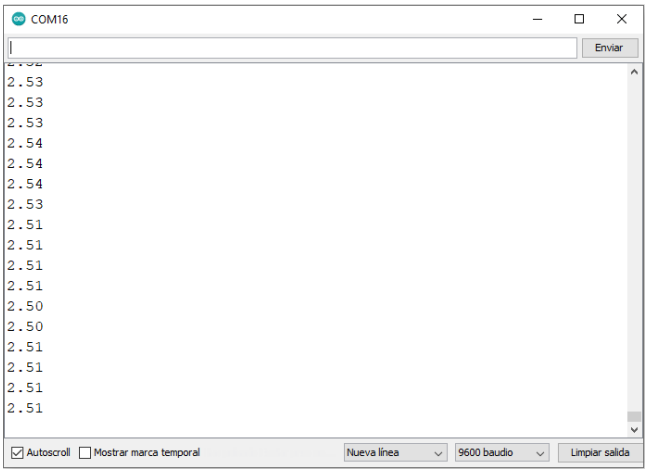

Figura 60: Recepción de datos de corriente vía I2C

Desde el esclavo vamos a comprobar que recibimos la Speed_Set sin ningún tipo de problema también. Conectamos el esclavo por el puerto serie a nuestro ordenador.

Activamos el control manual teniendo el Arduino Maestro encendido, y cambiamos los valores de Speed_Set desde el potenciómetro correspondiente al segundo motor y así verificamos como varía, ya que desde el puerto serie no podría cambiarlos al no tener el Arduino Master conectado al ordenador.

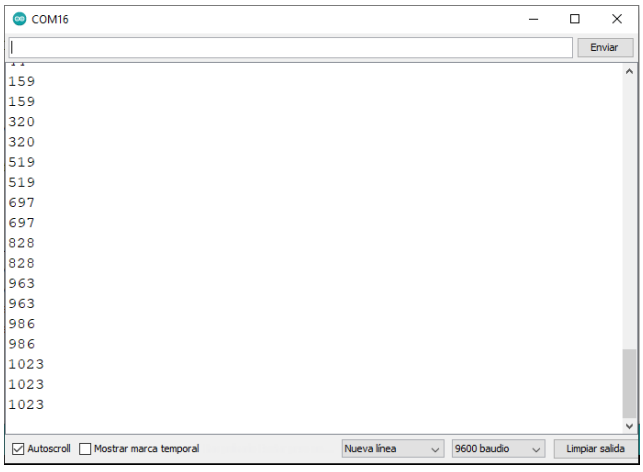

Figura 61: Recepción de valores de Speed_Set vía I2C

7.2 COMPROBACIÓN DE LA CONEXIÓN VIA SERIAL CON ORDENADOR

Comprobamos que lo que mando por el puerto serie lo identifica de manera correcta la rutina de control serial. El modo se mantiene a 0.

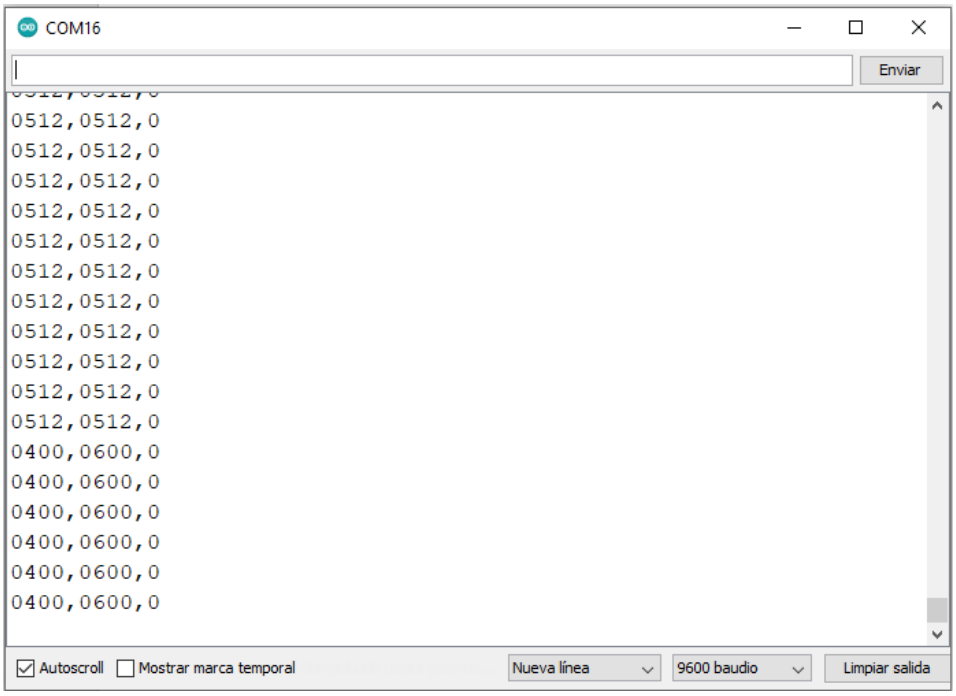

Figura 62: Recepción de datos por puerto serie

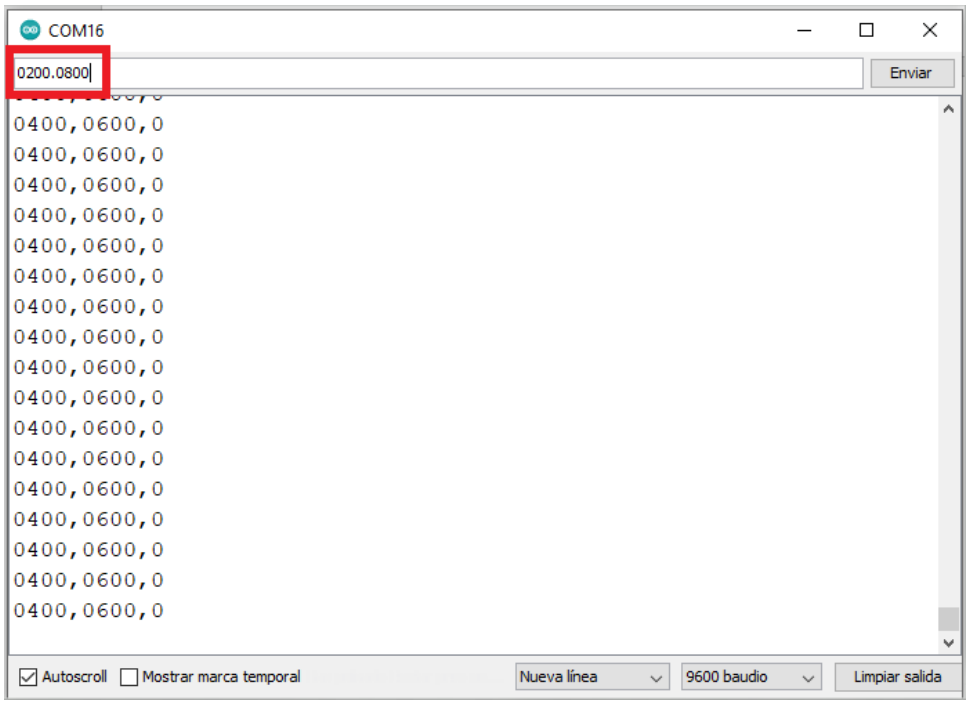

Figura 63: Recepción de datos por puerto serie

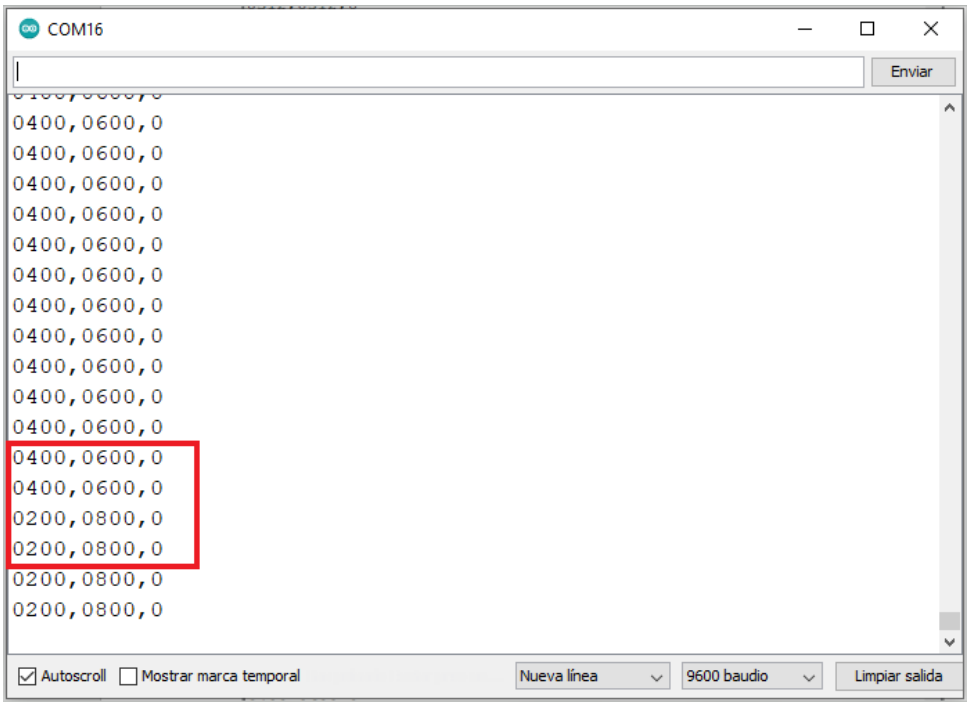

Figura 64: Recepción de datos por puerto serie

Esta comprobación se realiza para verificar el código empleado previamente a utilizarlo con la Raspberry Pi.

Correspondiendo (siguiendo la separación mediante una coma) los cuatro primeros caracteres a la Speed_Set1, los cuatro siguientes a Speed_Set2, y el último al Modo de control.

7.3 COMPROBACIÓN DE LA CONEXIÓN ENTRE RASPBERRY PI Y CONTROLADOR

Primeramente, antes de encender el sistema hay que asegurarse de:

- Sistema de navegación conectado a un puerto USB
- Arduino conectado a un puerto USB
- Motor de arranque SSD conectado a un puerto USB

Una vez iniciada la Raspberry, el sistema de arranque está configurado para iniciar automáticamente las aplicaciones que vamos a utilizar, las cuales son OpenCPN y VNAS (piloto automático).

En primer lugar, creamos una ruta en la aplicación OpenCPN.

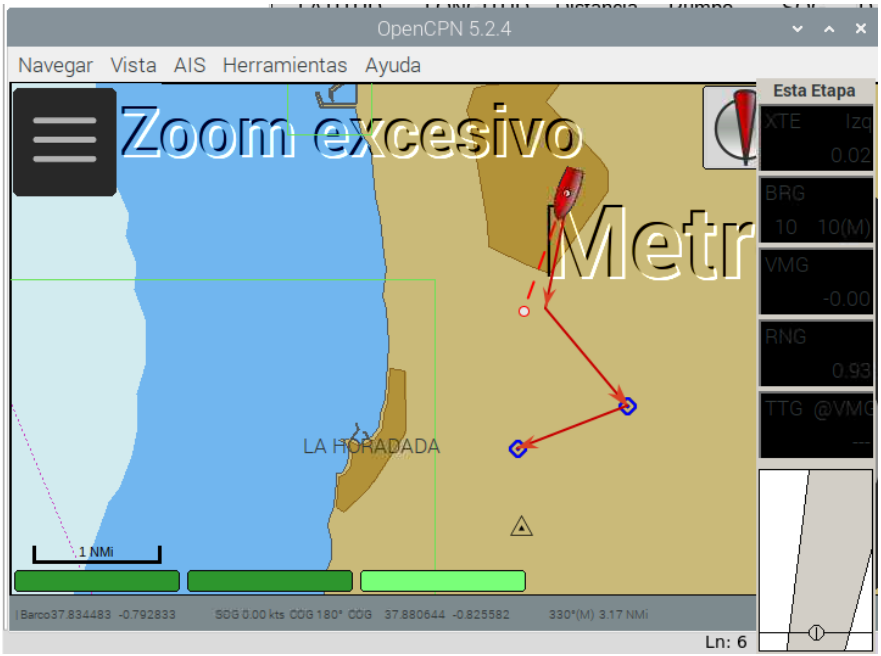

Figura 65: Creación de ruta en OpenCPN

A continuación, activamos el wavepoint al que queremos llegar, para que la Raspberry calcule el rumbo y la potencia necesaria.

Figura 66: Activación de ruta en OpenCPN

Una vez definida y activada la ruta, iniciamos el viaje desde la aplicación VNAS. Hay que destacar que, sin tener una ruta definida y activada, el VNAS no dejará iniciar el viaje de ninguna forma.

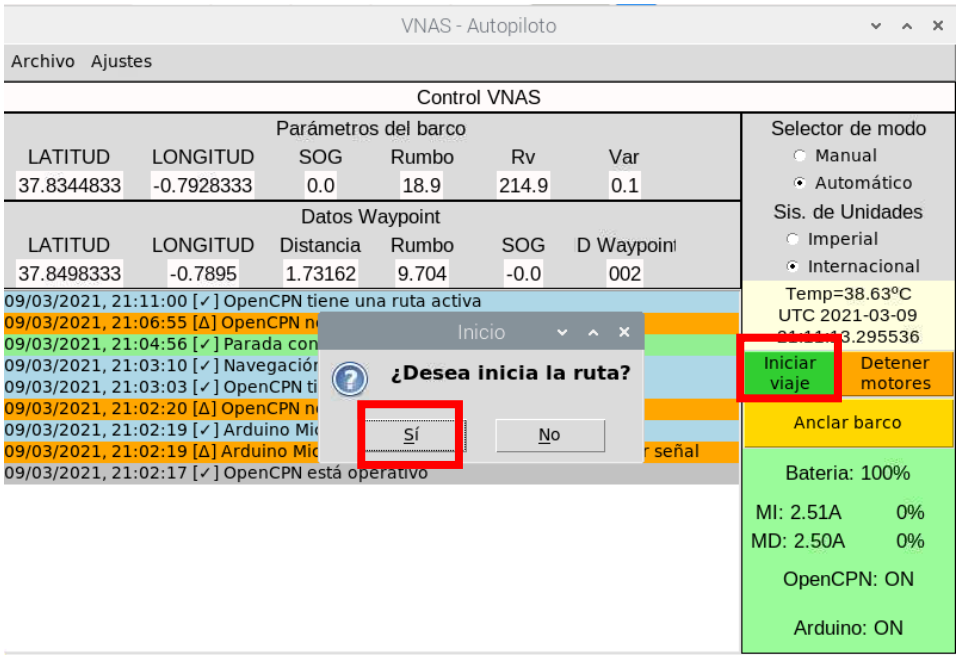

Figura 67: Iniciar viaje desde piloto automático

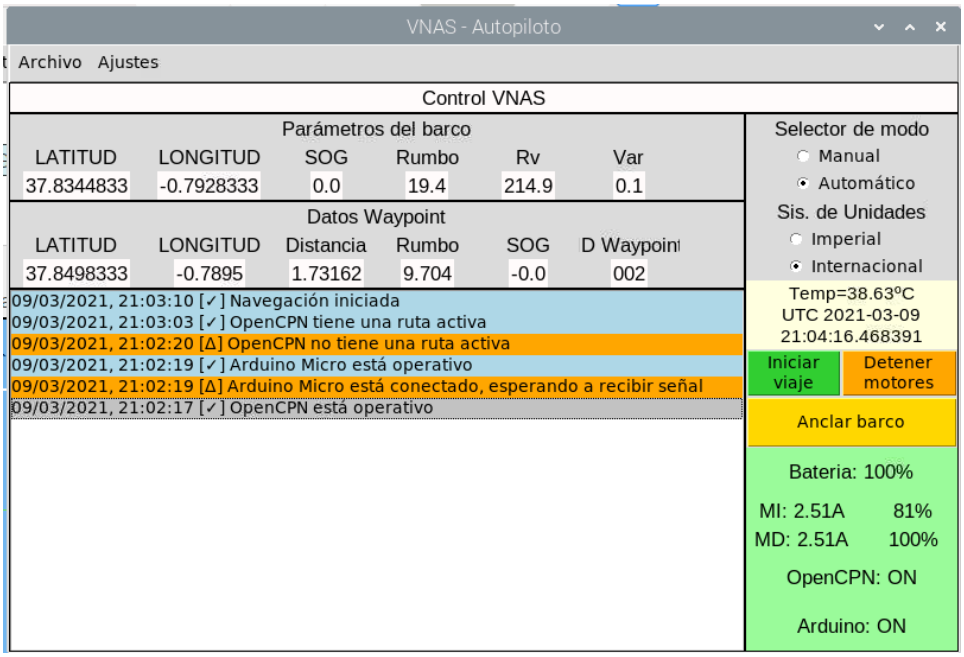

Figura 68: Viaje en curso desde piloto automático

Como vemos, comprobamos que la comunicación funciona, al ver que los datos representados por la Raspberry son los acordados y recibe perfectamente desde el puerto serie del controlador. También podemos observar que la ruta está activa y los motores en movimiento por el porcentaje que aparece al lado del consumo de cada motor. Ese porcentaje está referido a la potencia que desarrolla cada motor en tiempo real.

Seguidamente comprobamos que los motores se paren cuando se lo ordenemos mediante la interfaz de la aplicación VNAS.

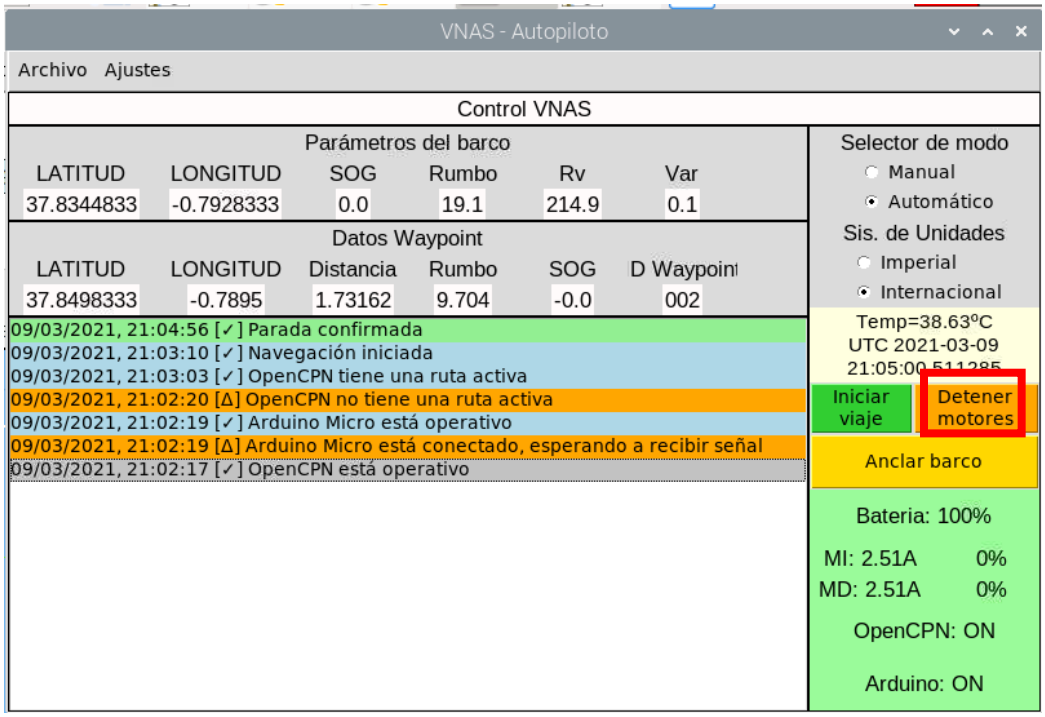

Figura 69: Parada desde piloto automático

Y como vemos, nos confirma la parada y el porcentaje de potencia que desarrolla cada motor está a su valor inicial, que es cero.

Si quisiéramos parar el VNAS y anclarlo en la posición que se encuentre actualmente, seleccionamos la opción de anclar barco y vemos lo que nos muestra a continuación.

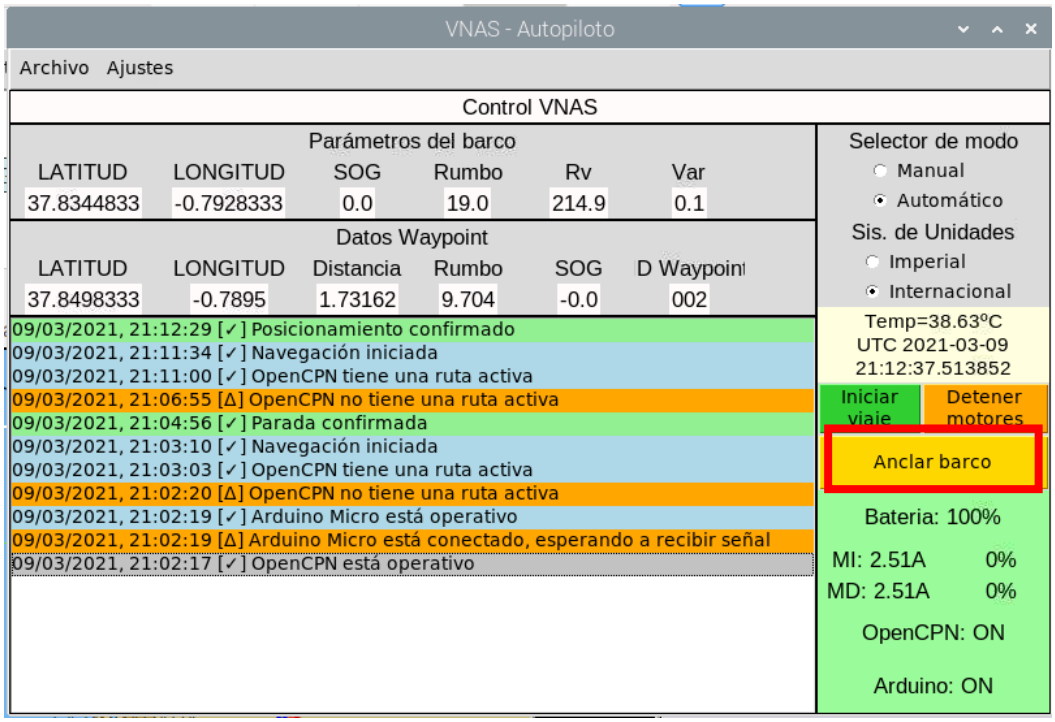

Figura 70: Anclado desde piloto automático

7.4 COMPROBACIÓN DEL FUNCIONAMIENTO MEDIANTE CONTROL MANUAL

En este apartado debemos observar el modo a 1. Para realizar la comprobación de la recepción de los datos de cada uno de los potenciómetros, dejamos fijo el valor de uno y variamos el otro y viceversa, así observamos su cambio con mayor nitidez.

- Primer potenciómetro

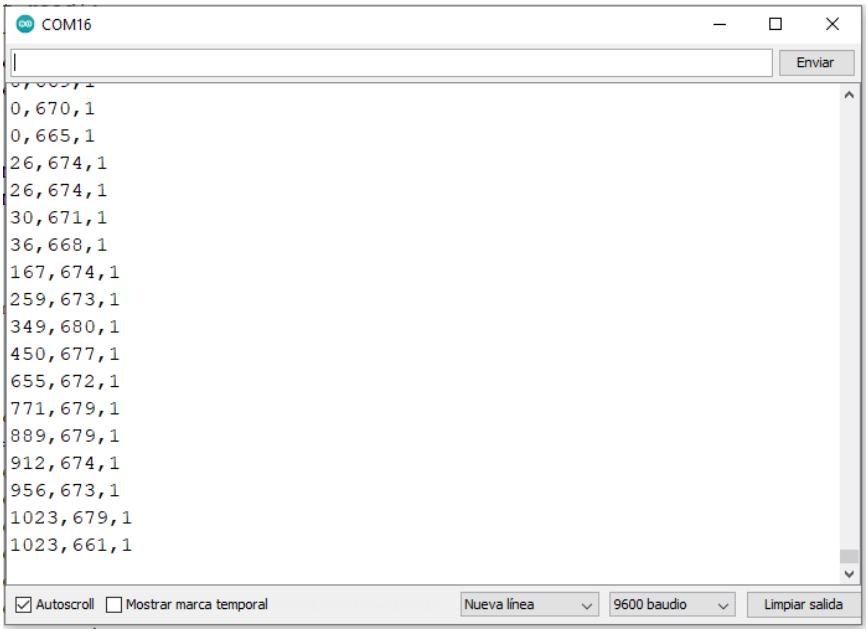

Figura 71: Recepción de datos primer potenciómetro

- Segundo potenciómetro

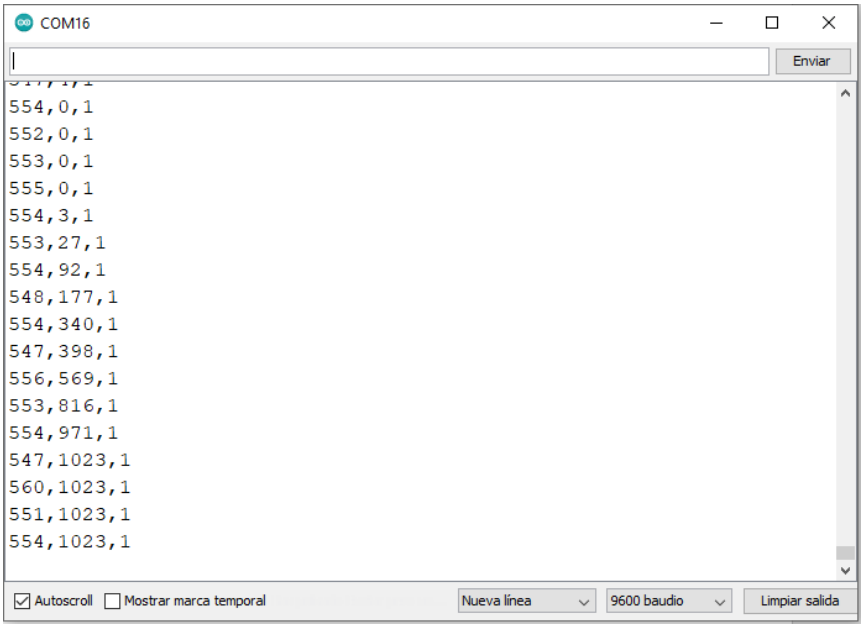

Figura 72: Recepción de datos segundo potenciómetro

Correspondiendo (siguiendo la separación mediante una coma) los cuatro primeros caracteres a la Speed_Set1, los cuatro siguientes a Speed_Set2, y el último al modo de control.

7.5 COMPROBACIÓN DE CAMBIO DE SENTIDO DE LOS MOTORES

Como únicamente tenemos el Arduino maestro conectado vía serial, lo comprobaremos directamente con su motor, mostrando por pantalla el pin DIR, que indica el sentido.

Como sabemos por los apartados descritos anteriormente, vemos que 511 es nuestro umbral para cambiar el pin DIR, por eso obtiene ese valor.

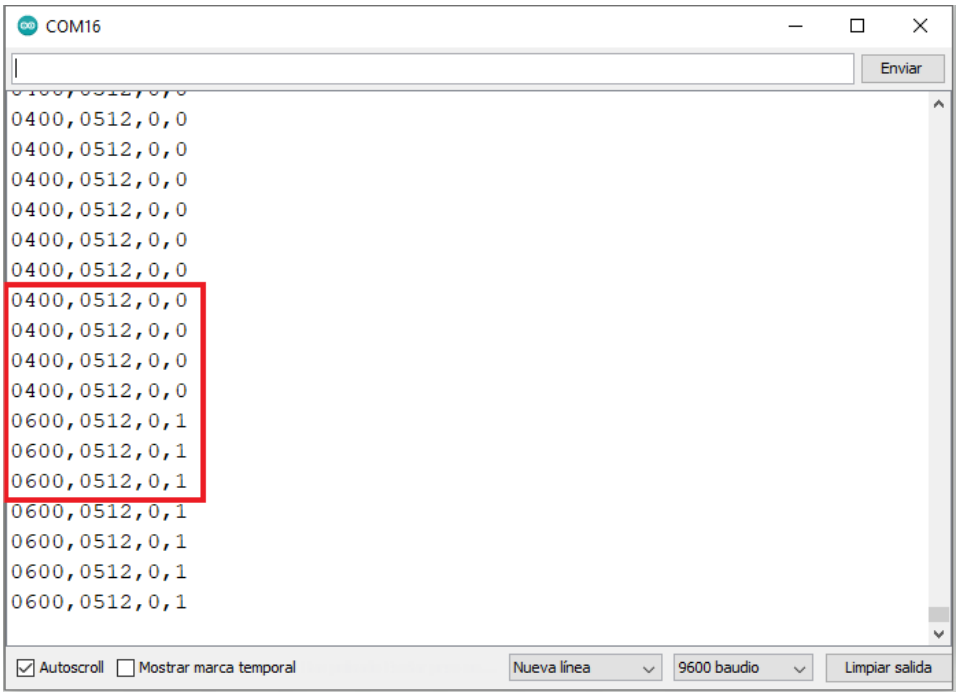

Figura 73: Comprobación de cambio de sentido de giro del motor

Pero lo vamos a ver aún más detallado y conciso.

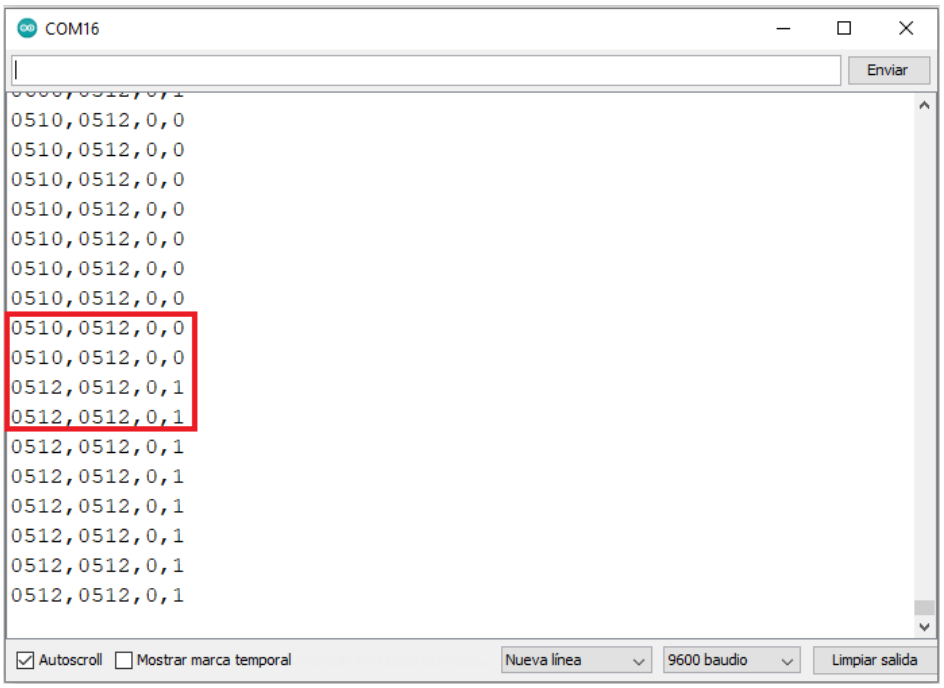

Figura 74: Comprobación de cambio de sentido de giro del motor

Correspondiendo (siguiendo la separación mediante una coma) los cuatro primeros caracteres a la Speed_Set1, los cuatro siguientes a Speed_Set2, el penúltimo al modo de control y el último al pin DIR.

Esta comprobación, una vez implementado el sistema de navegación sólo se podrá llevar a cabo con el control manual activado, ya que la Raspberry no controla el cambio de sentido de los motores, realiza los giros y cambios de dirección alternando la potencia de los motores en un sentido fijo, es decir, parando a uno de ellos en función del giro calculado.

8. CONCLUSIONES

El objetivo de este TFG era desarrollar el sistema de control de los motores del VNAS y adaptarlos para el entorno acuático y realizar sus posteriores pruebas y simulaciones en ese entorno, pero debido a la situación de Pandemia durante este último año y la situación de estado de alarma y restricciones perimetrales debido al virus SARS-CoV-2, se han atrasado los plazos deseados.

A pesar de esta situación, se ha conseguido desarrollar con éxito el sistema de control de los motores, conformado por el desarrollo de las tarjetas de circuito impreso (PCB), la programación de los controladores de ambos motores y los protocolos de comunicación entre los dos controladores, y entre la CPU central y nuestro controlador principal denominado Arduino Maestro. Todas las pruebas realizadas en vacío se han ejecutado exitosamente.

9. LÍNEAS FUTURAS

Como líneas de desarrollo futuras se deben contemplar las siguientes opciones:

- Adaptar las baterías para poder realizar conexión en paralelo y doblar su capacidad, ya que disponemos de dos baterías.

- Optimizar la obtención de carga de las baterías.

- Realizar pruebas en agua una vez realizados los apartados anteriores para asegurar la infraestructura.

- Ajustar consumo de los motores en agua, ya que no se puede determinar de una manera real el consumo con las pruebas realizadas hasta ahora en vacío.

BIBLIOGRAFÍA

[1] https://www.significados.com/telemetria/

[2] http://landing.sitrack.com/telemetr%C3%ADa-y-sus-aplicaciones

[3] http://www.practicasconarduino.com/manualrapido/conexiones_analgicas.html

[4] http://www.practicasconarduino.com/manualrapido/un_caso_especial_seales_pwm.html

[5] https://educarparaelcambio.com/arduino/reto-5-comunicandonos-con-arduino-puerto-serie/

https://exceltotal.com/que-es-excel/

[6] https://voilesetvoiliers.ouest-france.fr/industrie-nautique/chantier/architecture-navale/saildrone-premier-voilier-inhabite-a-traverser-l-atlantique-dans-les-deux-sens-4ee02eae-0abc-11ea-adb7-776ec54bacaf?fbclid=IwAR11APAbo6ijwgm6pdwuQ1UB-9J5UDAnkXRtdjofygPv6K8oBlgBKYnRAWo

[7] https://www.nauticayyates.com/barcos/novedades/x-shore-barco-electrico/amp/

[8] https://www.farodevigo.es/economia/2019/12/10/aister-entrega-navantia-primer-barco-15437742.html

[9] PFC's de otros proyectos

Alejandro González Redel "Implementación de un sistema de radiocomunicaciones para la transmisión de la telemetría de un barco autónomo."

Hassan Bahari Rhetassi "Desarrollo de una aplicación web de telemetría para el control de un barco autónomo."

José Carlos Urrea Celdrán "Avances en el desarrollo de un sistema de control para la navegación de un catamarán de forma autónoma"

[10] https://es.rs-online.com/web/p/raspberry-pi/1373331/

[11] https://www.arrow.com/es-mx/research-and-events/articles/an-overview-of-the-arduino-micro

[12] https://content.arduino.cc/assets/Pinout-Micro_latest.pdf

[13] https://www.pololu.com/product/2994

[14] https://www.pololu.com/product/2994/pictures#lightbox-picture0J7104

[15] https://www.amazon.co.uk/Jago-Electric-Outboard-Indicator-Inflatable/dp/B07QNYDHC9

[16] https://www.yuasa.es/batteries/automocion-ver-todas/ybx5000-silver-high-performance-smf-batteries/ybx5019.html

[17] https://www.amazon.es/Yizhet-Convertidor-Regulador-Ajustable-Alimentaci%C3%B3n/dp/B0823P6PW6/ref=pd_bxgy_img_3/259-1597227-6190351?_encoding=UTF8&pd_rd_i=B0823P6PW6&pd_rd_r=f71fc920-1eeb-4542-9cff-61f19205fe61&pd_rd_w=jcz2Q&pd_rd_wg=nwnE9&pf_rd_p=12d945b9-5148-4c92-8c2a-6d4b36ee4de4&pf_rd_r=MSNS6TWNB7EZ2XS808GS&psc=1&refRID=MSNS6TWNB7EZ2XS808GS

[18] http://ipowerelectronics.com/potenciometros/2339-potenciometro-lineal-deslizable-2-canales-10-k-arduino-pic.html

[19] https://rayte.com/componentes-electronicos-interruptores-interruptores/11771-interruptor-luminoso-azul-estanco-8434426150384.html

[20] https://www.cetronic.es/sqlcommerce/ficheros/dk_93/productos/421352085-1.jpg

[21] https://es.rs-online.com/web/p/contactos-de-crimpado/7620702/

[22] https://es.rs-online.com/web/c/conectores/carcasas-de-cables-y-conectores-para-pcb/contactos-de-crimpado/

[23] https://es.rs-online.com/web/p/conectores-macho-para-pcb/4838461

[24] https://es.rs-online.com/web/p/carcasas-de-cables-y-conectores-macho/6795287/

[25] https://es.rs-online.com/web/p/conectores-macho-para-pcb/5355197/

[26] https://es.rs-online.com/web/p/carcasas-de-cables-y-conectores-macho/8201175

[27] https://es.rs-online.com/web/p/conectores-macho-para-pcb/4838483/

[28] https://es.rs-online.com/web/p/carcasas-de-cables-y-conectores-macho/6795388/

[29] https://es.rs-online.com/web/p/conectores-de-horquilla/1355142/

[30] https://es.rs-online.com/web/p/conectores-de-horquilla/1354514/

[31] https://stockseguridad.com/es/cables/284-bobina-100-metros-de-cable-apantallado-4-hilos.html

[32] https://es.rs-online.com/web/p/montajes-para-pcb/8930586/

[33] https://es.rs-online.com/web/p/fusibles-de-cartucho/0563380/

[34] https://www.grupodebiase.com/tuberia-y-accesorios/5223-tuberia-y-accesorios-prensaestopa-plastica-30-35-mm-pg42-legrand-098018.html

[35] https://descubrearduino.com/lenguaje-arduino/

[36] https://es.qaz.wiki/wiki/EAGLE_(program)

[37] https://www.altium.com/es/solution/what-is-a-pcb

[38] https://es.wikipedia.org/wiki/Bater%C3%ADa_de_autom%C3%B3vil

ANEXOS

ANEXO 1

CÓDIGO ARDUINO MAESTRO

```
#include <TimerOne.h>
#include <Wire.h>
#define PIN_SELECT_MODO 12
unsigned long t;
boolean Modo_S_M = false;
int serie_manual = 0;
int pot_read1=0;
int pot_read2=0;
int Speed_Set1=512;
int Speed_Set2=512;
String Motor1;
String Motor2;
int PWM_motor1_int;
int PWM_motor2_int;
String PWM_motor1_aux;
String PWM_motor2_aux;
String Serie;
int pwm=0;
String Modo;
String current2_A;
String current1_A;
float current1;
int cs=0;
#define POT1 A0
#define PWM_motor1 9
#define DIR1 7
#define POT2 A1
#define CSENSE A2
#define FLT 4
String Speed_i2c;
char current2 [4];
int porcentaje;
int bateria;
String Carga_bateria;

void setup() {
  // put your setup code here, to run once:
  Wire.begin();
  pinMode(A3, INPUT); //BATERIA
  pinMode(A2, INPUT); // CS CONTROLADOR
  pinMode(POT1,INPUT); //SLIDER 1
  pinMode(7,OUTPUT); // DIR1 pins
  pinMode(POT2,INPUT); //SLIDER 2
  pinMode(PIN_SELECT_MODO,INPUT); // BOTON
  pinMode(FLT, INPUT_PULLUP);
  pinMode(2,INPUT_PULLUP);
  pinMode(3,INPUT_PULLUP);
  Timer1.initialize(50);
  Serial.begin(9600);
}
```

```
void Captura_Datos_Serie(void){
  // Recepcion comando SERIAL

  if (Serial.available() > 0)
  {
    t=millis();
    String str = Serial.readStringUntil('\n');
    Motor1=str.substring(0,4);
    Speed_Set1=Motor1.toInt();
    Motor2=str.substring(5,9);
    Speed_Set2=Motor2.toInt();
    Speed_i2c= String (Speed_Set2);
    //Manda_Speed();
  }
  else {
    if ((millis()-t)>=10000) {//SI NO RECIBO DATOS TEMPORIZO Y PONGO
SPEED A 0
      Speed_Set1=511;
      Speed_Set2=511;
      Speed_i2c= String (Speed_Set2);
      //Manda_Speed();
    }
  }
}
void Captura_Datos_Pot(void){
    pot_read1=analogRead(A0);
    pot_read2=analogRead(A1);
    //pwm=map(pot_read1,0,1023,0,255); //Esta capado a 911 el slider
como valor maximo que lee.
    if (pot_read1 <540 && pot_read1>480) {
    Speed_Set1=511;
  }
   else{
    Speed_Set1 = pot_read1;
   }
   if (pot_read2 <540 && pot_read2>480) {
    Speed_Set2=511;
  }
   else{
    Speed_Set2 = pot_read2;
   }
   Speed_i2c = String (Speed_Set2);

}

void leer_voltios()
{

bateria= (analogRead(A3));
if (bateria > 940)
  bateria = 940;
porcentaje= map (bateria,879,940,0,100);

}
```

```
void loop() {
//-------------ENVÍO SPEED SET A ESCLAVO---------
    Wire.beginTransmission(10);//inicializamos la transmision i2c al
esclavo 1
  for(byte i=0;i<=Speed_i2c.length();i++){
    Wire.write(Speed_i2c[i]); //le enviamos caracter por caracter
  }
  Wire.endTransmission(10);

  serie_manual = digitalRead(PIN_SELECT_MODO);
   if (serie_manual == HIGH){
    Captura_Datos_Pot();
    Modo="1";
    }
   else {
      Captura_Datos_Serie();
      Modo="0";
      }


  cs=analogRead(CSENSE);
  current1=((cs*1.1/1023)-0.05)*100;
    if (current1>=0 && current1 <= 60) {
     //if (current1 <= 60) {
      if (Speed_Set1 > 511){
      digitalWrite(DIR1,HIGH);
      Timer1.pwm(PWM_motor1, (Speed_Set1*2)-1023);
      PWM_motor1_int= (Speed_Set1*2)-1023; //-- duty 0 to 1023.

}
        else if (Speed_Set1<511){
        digitalWrite(DIR1,LOW);
        Timer1.pwm(PWM_motor1, 1023-Speed_Set1*2);
        PWM_motor1_int= 1023-Speed_Set1*2; //-- duty 0 to 1023.

    }
          else {
  Timer1.pwm(PWM_motor1, 0);
    }

     }
   else {
  Timer1.pwm(PWM_motor1, 0);
  PWM_motor1_int=0;
    }
 // FLT - Activation : Se parara la generacion de PWM

  if ( digitalRead(FLT) == LOW){
    Timer1.pwm(PWM_motor1, 0);
    //Serial.print("FAULT");
```

```
//--------------RECEPCION DE DATOS I2C------------
    Wire.requestFrom(10,4);
     byte j=0;
  while(Wire.available()>0){
    current2[j] = Wire.read();
    j++;
  }

  current2_A= String(current2);
  String current_rep= current2_A.substring(0,4);
  current1_A=String(current1);
  //PWM_motor1_aux = String (PWM_motor1_int);
  //PWM_motor2_aux = String (PWM_motor2_int);
  PWM_motor1_aux = String (pot_read1);
  PWM_motor2_aux = String (pot_read2);
  leer_voltios();
  Carga_bateria= String(porcentaje);
  //String Serie =
(PWM_motor1_aux+","+cs+","+current1_A+","+Modo+","+Carga_bateria);
  String Serie=
(current1_A+","+current_rep+","+Modo+","+Carga_bateria+","+PWM_moto
r1_aux+","+PWM_motor2_aux);
  Serial.println(Serie);
}
```

DESCRIPCIÓN DE VARIABLES Y PARÁMETROS ARDUINO MAESTRO

- **Valores enteros**

serie_manual→ Valor que se pone a '1' o '0' cuando se detecta un cambio de posición en el interruptor de cambio de modo.
pot_read1→ Valor analógico comprendido entre 0 y 1023 que representa los datos recibidos del potenciómetro manual del primer motor.
pot_read2→ Valor analógico comprendido entre 0 y 1023 que representa los datos recibidos del potenciómetro manual del segundo motor.
Speed_Set1→ Valor analógico comprendido entre 0 y 1023 que representa los datos recibidos de la Raspberry Pi por el puerto serie del primer motor.
Speed_Set2→ Valor analógico comprendido entre 0 y 1023 que representa los datos recibidos de la Raspberry Pi por el puerto serie del segundo motor.
PWM_motor1_int→ Valor analógico comprendido entre 0 y 1023 que representa el valor del primer motor teniendo en cuenta el sentido de giro de dicho motor.
PWM_motor2_int→ Valor analógico comprendido entre 0 y 1023 que representa el valor del segundo motor teniendo en cuenta el sentido de giro de dicho motor.
Cs→ Valor de lectura del pin del controlador del motor que muestra la relación de corriente frente a voltaje en el motor.
Batería→ Valor de lectura del divisor de tensión a partir de una entrada analógica comprendido entre 0 y 1023.
Porcentaje→ Valor en tanto por cien, obtenido en función del mapeo de los valores máximos y mínimos del valor de batería.

- **Valores en formato cadena**

Motor1→ Cadena obtenida de la información recibida por el puerto serie que representa los datos del primer motor.
Motor2→ Cadena obtenida de la información recibida por el puerto serie que representa los datos del segundo motor.
Serie→ Cadena de datos que debemos enviar a la Raspberry por el puerto serie.
Modo→ Pin que representa '0' si el método de obtención de datos es vía serial, o '1' si el método de obtención de datos es manual.
current2_A→ Valor de la corriente del segundo motor que está a cargo del Arduino Esclavo recibido por comunicación I2C.
current1_A→ Valor de la corriente del motor en formato cadena preparada para enviarla como substring por el puerto serie dentro de la cadena "Serie" descrita anteriormente.
Speed_i2c→ Cadena que contiene el valor entero "Speed_Set2" preparada para enviarlo mediante I2C al Arduino Esclavo.
Carga_bateria→ Porcentaje de la batería en formato cadena.

- **Definición de pines**

POT1→ Referido al pin de entrada analógico A0.
POT2 → Referido al pin de entrada analógico A1.
PWM_motor1→ Referido a la salida digital 9.
DIR1 → Referido al pin que controla la dirección de giro del motor.
CSENSE → Referido al pin que nos facilita el valor de la corriente.
FLT → Referido al pin de falla del controlador del motor.

- **Señal para contador, valor decimal y carácter**

t→ Variable donde se guarda el valor del contador de la función “millis”.
current1→ Valor decimal de la corriente en amperios.
current2 [4]→ Valor de corriente convertido a carácter de 4 espacios, incluyendo la coma y decimales.

CÓDIGO ARDUINO ESCLAVO

```
#include <Wire.h>
#include <TimerOne.h>
int cs;
float current_2;
// RECIBIR COMUNICACION I2C
char mensaje1 [4];
int Speed_Set2=511;
String Speed_Set;
//ENVIAR COMUNICACION I2C
String current_mA;

#define DIR2 8
#define PWM_motor2 10
#define FLT  4
#define CSENSE A0
int pwm;

void setup()
{

  pinMode(A0,INPUT);
  pinMode(8, OUTPUT); //-- DIR2 pins
  pinMode(FLT, INPUT_PULLUP); // FLT pin.
  Timer1.initialize(50);  // 50 us = 20 kHz
  Wire.begin(10);
  pinMode(2,INPUT_PULLUP);
  pinMode(3,INPUT_PULLUP);
  Wire.onReceive(datoRecibido);
  Wire.onRequest(MandaCorriente);
  Serial.begin(9600);
}


//-----------------------RECIBIR DATOS I2C DEL MAESTRO--------------
-----
void datoRecibido()
{
  //if {
    while(!Wire.available()){
      Speed_Set2=512;
    }
    while(Wire.available()>0)    {
    for (byte i=0;i<=4;i++){
    mensaje1[i] = Wire.read(); // Recibe la palabra del i2c tx
    }
    Speed_Set = String (mensaje1);
    Speed_Set2= Speed_Set.toInt();
     }
}
```

```
//----------------------ENVIAR DATOS I2C A  MAESTRO---------------
void MandaCorriente()
{
  //Wire.beginTransmission(1);
  for(byte i=0;i<=current_mA.length();i++){
    Wire.write(current_mA[i]);//Envio caracter por caracter
}
  //Wire.endTransmission();
}


void loop() {
  //datoRecibido();

  //----------------------------------------
  //              CURRENT SENSE
  analogReference(INTERNAL);
  cs = analogRead(CSENSE);
  current_2 = ((cs*0.01/1023)+0.025)*100;
  //if ( current_2 >= 0 && current_2 <= 60) {
    if (current_2 <= 60) {
    if (Speed_Set2 > 511) {
      digitalWrite(DIR2, HIGH);
      Timer1.pwm(PWM_motor2, (Speed_Set2*2)-1023); //-- duty 0 to
1023.
    }
    else if (Speed_Set2 < 511) {
      digitalWrite(DIR2, LOW);
      Timer1.pwm(PWM_motor2, 1023-Speed_Set2*2); //-- duty 0 to
1023.
    }
    else {
  Timer1.pwm(PWM_motor2, 0);

    }

    }
    else {
      Timer1.pwm(PWM_motor2, 0);
    }

    // FLT - Activation : Se parara la generacion de PWM

    if ( digitalRead(FLT) == LOW) {
      Timer1.pwm(PWM_motor2, 0);
    }

    current_mA = String(current_2);
    //MandaCorriente();
    //Serial.println(Speed_Set2);


  }
```

DESCRIPCIÓN DE VARIABLES Y PARÁMETROS ARDUINO ESCLAVO

- **Valores enteros, decimales y en formato carácter**

Cs→ Valor de lectura del pin del controlador del motor que muestra la relación de corriente frente a voltaje en el motor.
Speed_Set2→ Valor analógico comprendido entre 0 y 1023 que representa los datos recibidos por I2C del segundo motor.
current_2→ Valor decimal de la corriente en Amperios.
mensaje1 [4]→ Valor del motor convertido a carácter de 4 espacios, recibido por comunicación I2C.

- **Valores en formato cadena**

Speed_Set→ Valor en formato de cadena comprendido entre 0 y 1023 que representa los datos recibidos por I2C del segundo motor.
current_mA→ Valor en formato de cadena de la corriente en Amperios.

- **Definición de pines**

DIR2→ Referido al pin que controla la dirección de giro del motor.
PWM_motor2→ Referido a la salida digital 10.
FLT → Referido al pin de falla del controlador del motor.
CSENSE→ Referido al pin que nos facilita el valor de la corriente (A0).

ANEXO 2
PRESUPUESTO

En la siguiente tabla se muestran el listado de componentes y el presupuesto total empleado para llevar a cabo este proyecto. En algunos casos, el vendedor exigía una cantidad mínima de cada componente que superaba las necesarias para el proyecto.

Pos.	Concepto/Descripción	Cantidad	Precio unitario(€)	Importe (€)
1	Arduino Micro	2	16.98	33.96
2	Pololu G2 High-Power Motor Driver 18v25	2	32.88	65.76
3	Motor Jago Electric Outboard	2	166.61	333.22
4	Batería YBX5019 12V 100Ah 900A Yuasa	2	165.29	330.58
5	Convertidor DC/DC de 12 a 5V	1	6	6
6	Potenciómetro deslizante (10kΩ)	2	1	2
7	Interruptor de cambio de modo de control	1	2.16	2.16
8	Seta de parada de emergencia	1	10.25	10.25
9	Contacto de crimpado hembra	100	0.038	3.80
10	Portafusibles para montaje en PCB	10	0.777	7.77
11	Fusible de cartucho 1 A (5x20mm)	10	0.156	1.56
12	Conector macho para PCB (2 pines)	10	0.219	2.19
13	Carcasa de conector hembra (2 pines)	10	0.088	0.88
14	Conector macho para PCB (3 pines)	10	0.092	0.92
15	Carcasa de conector hembra (3 pines)	10	0.43	4.3
16	Conector macho para PCB (4 pines)	10	0.268	2.68
17	Carcasa de conector hembra (4 pines)	10	0.134	1.34
Total				**809.37**

ANEXO 3

GUÍA DE INICIO DE CONTROL REMOTO DE RASPBERRY

El sistema lleva incorporado una pantalla táctil, pero si se desea utilizar otro dispositivo, ya sea móvil, tablet o un propio ordenador, nos podemos conectar remotamente a través de la red WiFi propia que está configurada en la Raspberry Pi con el uso del software VNC Server, que deberemos descargar en nuestro dispositivo utilizado como remoto.

En primer lugar, debemos iniciar la raspberry con la pantalla táctil y el teclado incorporados conectados. Para asegurarnos de que el servicio remoto esté activado, debemos realizar los siguientes pasos:

- Nos situamos en el menú principal
- Seleccionamos la pestaña de Accesorios, y a su vez abrimos la Terminal de Linux
- Una vez abierta esta terminal escribimos el comando “vncserver”

Llegados a este punto podemos desconectar la pantalla táctil de la Raspberry y el teclado y dirigirnos a nuestro dispositivo remoto.

Desde el dispositivo remoto nos conectamos a la red WiFi. Los datos de acceso son:

SSID: VNAS

Pass: politecnica

(Canal 6)

Completado el paso anterior, abrimos el software VNC Server descargado previamente y realizamos una nueva conexión. En el cliente del software VNC la IP de acceso al sistema es siempre 10.10.10.1, en el puerto 1.

Las claves son:

Usuario: pi

Contraseña: raspberry

Una vez completado este proceso tendremos control remoto total sobre nuestra Raspberry Pi. Tendremos una sesión de escritorio nueva, desde la cual se debe abrir la aplicación OpenCPN y el archivo situado en el escritorio “vnas.py” y seleccionar en la barra superior la opción de “Run→Run Module”. Seguidamente se iniciará la aplicación de piloto automático para comenzar la navegación.