UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería Informática

Banco de pruebas para motores sin escobillas

Test stand for brushless motors

Realizado por
Agustín Tejero Marín

Tutorizado por
José Francisco Martín Canales

Departamento
Electrónica
UNIVERSIDAD DE MÁLAGA

MÁLAGA, diciembre de 2021

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Graduado en Ingeniería Informática

Banco de pruebas para motores brushless

Test stand for brushless motors

Realizado por
Agustín Tejero Marín

Tutorizado por
José Francisco Martín Canales

Departamento
Electrónica

UNIVERSIDAD DE MÁLAGA
MÁLAGA, DICIEMBRE DE 2021

Fecha defensa: diciembre de 2021

Agradecimientos

En primer lugar quería dar las gracias a mi familia, a mi madre y mi hermana pero sobre todo y en especial a mi padre por haberme ayudado tanto, haber estado apoyándome desde siempre y por haber confiado en mí.

A todos los amigos que he hecho durante estos 4 años de etapa universitaria, que me han ayudado mucho a crecer tanto académicamente como personalmente y que sin ellos no estaría donde estoy y a mi tutor de este proyecto por haber estado siempre disponible cada vez que he necesitado su ayuda.

Resumen

El objetivo del proyecto es la creación de un entorno de trabajo para motores sin escobillas que permita probar las diferentes características de éstos y su respuesta ante diferentes situaciones de forma que sea capaz de ayudar a gestionar, identificar y evaluar la forma en la que debe ser implementado en diferentes sistemas. Para la gestión de este entorno se hará uso de dos aplicaciones distintas. La primera encargada de la gestión de todos los sensores que se usarán para registrar las distintas propiedades del motor, esta aplicación se ejecutará sobre una placa Arduino que al mismo tiempo deberá enviar todos estos datos mediante un protocolo de comunicaciones. La segunda aplicación se trata de un programa en Java cuyo objetivo es la comunicación y gestión de esta placa Arduino. Este otro programa permitirá gestionar el acelerador del motor que se está probando a la vez que se pueden los datos en tiempo real, ya sea en cuadros de texto o sobre una gráfica y permitirá ejecutar una prueba que probará todos los niveles del acelerador registrando todos los datos y su evolución.

La amplia implantación de los vehículos aéreos no tripulados (UAV) en su variedad de dron ha disparado el uso de los motores sin escobillas o motores brushless con respecto a los motores eléctricos convencionales debido a la gran cantidad de ventajas que éstos ofrecen como mayor eficiencia, mayor vida útil o un menor ruido. La gran variedad de uso y aplicaciones de estos vehículos hace que las características de los motores usados puedan ser muy diferentes entre los distintos modelos. De ahí, la necesidad de poder evaluar las características de estos motores antes de su adaptación al diseño final en estos drones. Aunque la motivación principal del proyecto esté relacionada con el uso hoy día de motores brushless sobre drones también es necesario realizar estas mismas pruebas antes de la implementación de estos motores sobre otros sistemas industriales, informáticos, etc.

Palabras clave: Modbus, motores sin escobilla, monitorización de sensores, Arduino

Abstract

The main goal of the project Is the creation of two separated software system. The first one dedicated to the management of all the sensors that we will be using to record the different properties of the motor on an Arduino board while at the same time Is sending all of this data through the communication channel using a communication protocol.
The second software system is a Java program for the communication and control of the program inside the Arduino board and the management of all the data collected by the sensors from the other program. This Java system will allow us to manage the throttle of the engine that we are testing while we could see all Its data in real time, either in boxes or graphs. It will also let us run a test that will go through all the throttle levels recording all these data and its evolution.

The wide implementation of unmanned aerial vehicles (UAV) in its drone variety has shot up the use of brushless motors regarding the conventional electric motors due to the large number of advantages they offer. The great variety of uses and applications of these vehicles means that the characteristics of the motors they use may be very different between all these different models. Due to all of this appear the need to be able to evaluate the characteristics of these motors before their adaptation to a final design, whether in drones, in industrial systems, computer science, etc.

Keywords: Modbus, brushless motors, monitoring sensors, Arduino

Índice

1

Introducción

Se introducirán las razones sobre las cuáles se eligió y desarrolló este proyecto además de hablar brevemente sobre las metodologías y tecnologías utilizadas en su implementación.

1.1 Motivación

Los motores de corriente continua sin escobillas son uno de los tipos de motores que mayor popularidad ha ido ganando en los últimos años. Este tipo de motores se emplean en muchos tipos de sectores industriales tales como en el sector automovilístico con motos y coches eléctricos, el sector médico o para equipos de automatización e instrumentación, sin embargo, una de las principales razones por las que se ha popularizado este tipo de motores es debido a su uso sobre drones en el sector aeronáutico.

Los drones o vehículos aéreos no tripulados (UAV) tienen su origen en quizá Nikola Tesla quién ya habría imaginado máquinas o barcos que pudiesen ser controlados a distancia mediante ondas, impulsos o radiaciones. Todo esto estaba reflejado en su patente “Method of and apparatus for controlling mechanism of moving vessels or

vehicles" [1] concedida el 8 de noviembre de 1898. Sin embargo, no fue hasta la primera guerra mundial en la que Archibald Montgomery Low, capitán de la Royal Flying Corps británica [3], supervisó la construcción de aviones dirigidos por control remoto y equipados con explosivos. El mismo año Elmer Ambrose Sperry, uno de los inventores del giroscopio, junto con Peter Hewitt desarrolló una plataforma de aeronaves no tripuladas que llevaban incorporada una catapulta para lanzar torpedos [2][4]. Estos fueron los primeros precursores de los vehículos aéreos no tripulados. A partir de entonces diferentes países han ido aportando sus propios programas en la evolución de los drones, hasta el día de hoy, en el que muchos han terminado por adoptar un papel más comercial.

Desde que en 2013 Jeff Bezos anunciase su programa de drones para el envío y reparto de paquetes de Amazon el número de permisos de drones comerciales aprobados por la Administración Federal de Aviación ha aumentado considerablemente, desde los 4,100 permisos aprobados en 2016 hasta los 342,357 aprobados en 2021 [5]. Estos drones comerciales son usados en muchísimos sectores distintos ya sea en la agricultura, cine y fotografía, obras civiles y un largo etcétera como puede verse en la Figura 1.

Figura 1. Sectores productivos actuales en los que se emplean los drones. Fuente: [1]

Este crecimiento que se está dando en este sector no es algo que parece que vaya a ir frenándose con el paso de los años, sino que seguirá en auge y permitirá la creación de miles de puestos de empleo en esta área.

1.2 Objetivos

El objetivo del proyecto es la creación de una plataforma que sea capaz de ayudar a caracterizar los motores brushless susceptibles de ser utilizados en vehículos UAV (aviones, drones) junto con un firmware encargado del control y la gestión de los elementos hardware del sistema y una aplicación de gestión usada como interfaz de usuario y de la comunicación con la plataforma para la recopilación de información, datos generados por el sistema y envío de comandos para su control.

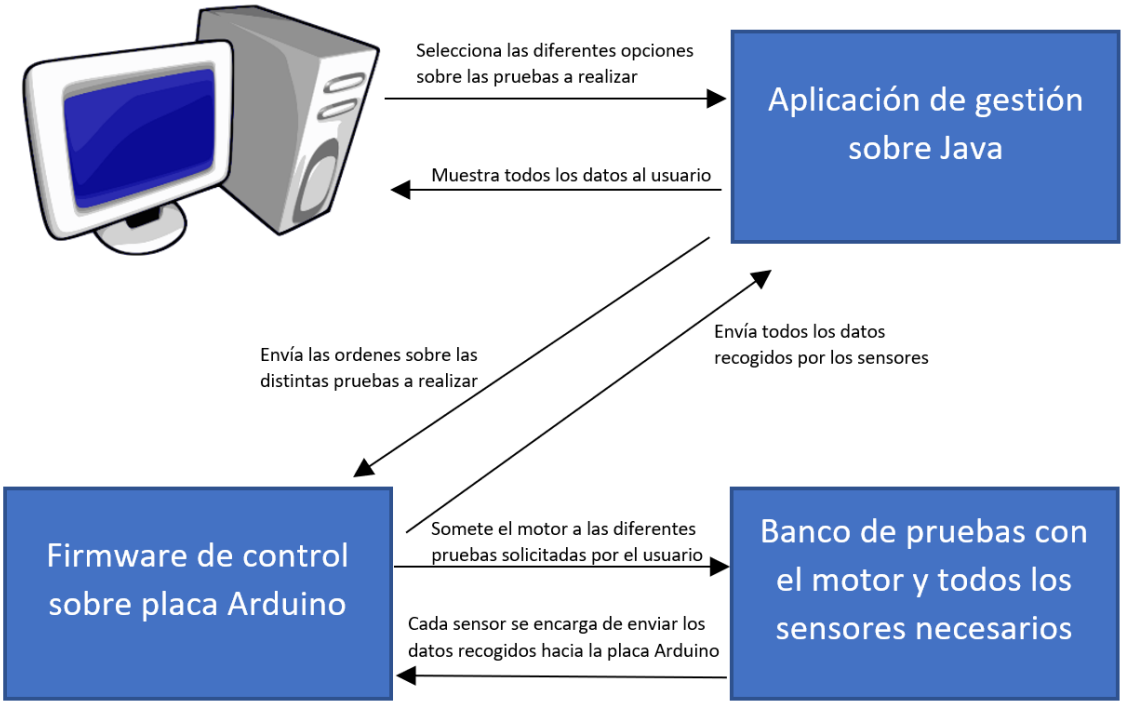

Figura 2. Diagrama de bloques general del proyecto.

El dispositivo consiste en un banco de pruebas donde colocar un motor sin escobilla (en adelante, brushless), al que se le acoplará una hélice y al hacerlo funcionar recopilar diferentes datos acerca de este funcionamiento. Para hacer esto posible, el

dispositivo contará con diferentes sensores que serán gestionados por el firmware para su correcto funcionamiento. Estos sensores consistirán en una galga extensiométrica para medir el empuje que da el motor con la hélice seleccionada, un sensor de revoluciones para medir las revoluciones por minuto (rpm) del motor y un sensor de voltaje e intensidad para medir su consumo durante el proceso. El funcionamiento de un motor brushless es gestionado mediante un dispositivo llamado ESC o *"Electronic Speed Controller"* incluido en el sistema y necesario para el control de la velocidad de giro del mismo (acelerador).

El elemento principal del sistema ("Heart of the system") será un módulo Arduino nano basado en el procesado AtMega328P (Figura 3) que contendrá el firmware de control que estará desarrollado en C++. Sobré él, recaerán todas las labores de configuración, gestión y control de los sensores indicados con anterioridad, así como controlar el desarrollo de las pruebas ideadas para la caracterización del motor objeto y la recopilación de cuantos datos sean necesarios.

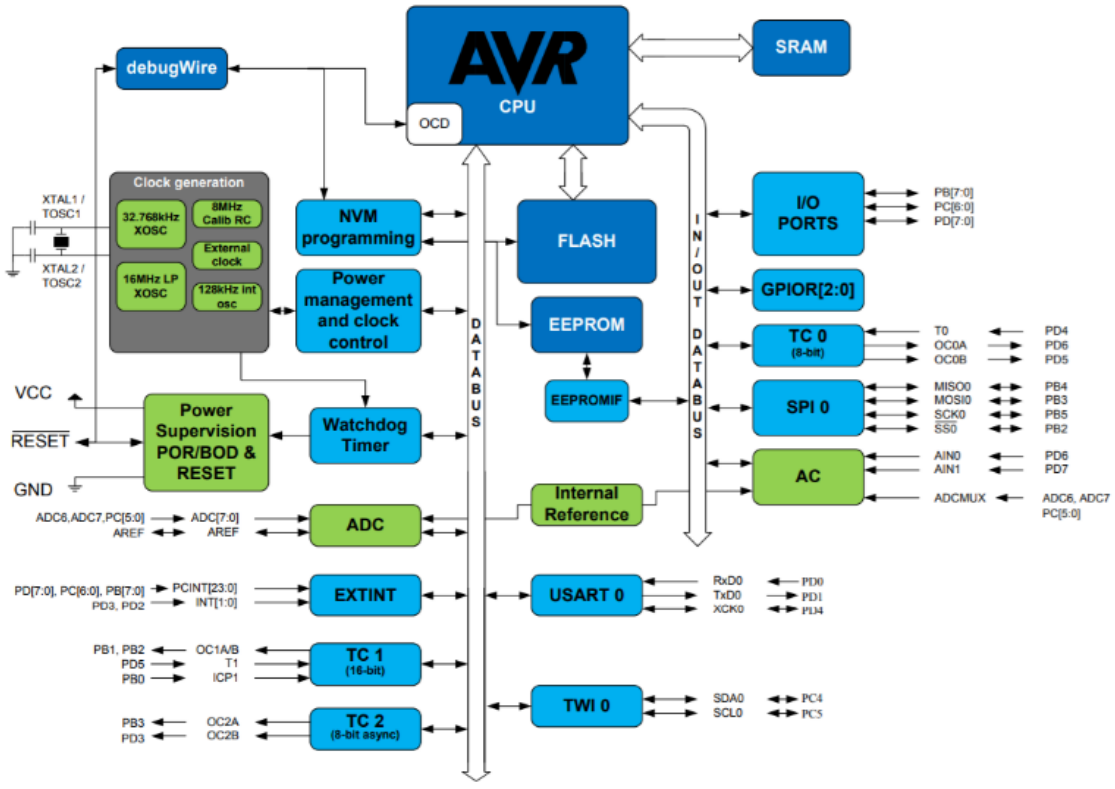

Figura 3. Arquitectura del procesador ATmega328P

Por último, la aplicación de gestión, como ya ha quedado indicado antes, es una interfaz gráfica de usuario o GUI por sus siglas en inglés *"Graphical User Interface"* que deberá ser capaz de comunicarse con el firmare de control mediante los protocolos de comunicaciones Modbus o telemetría implementados en el sistema, con el objetivo de recopilar los datos generados en las distintas pruebas además de ser capaz de mostrarlos al usuario mediante cuadros de texto o gráficos. También será usada para enviar comandos de control al sistema, necesarios para su correcto funcionamiento.

Con estos datos un experto puede ser capaz de dar un diagnóstico acerca de cualquier problema que pudiese tener el motor, saber sus límites y ayudar en la elección correcta del mismo en función de la aeronave, y la aplicación a la que esté destinada.

1.3 Tecnologías usadas

Durante el desarrollo del proyecto se han usado diferentes sensores, librerías y lenguajes de programación:

- **C++:** Es un lenguaje de programación de alto nivel orientado a objetos que surgió de un intento de extender el lenguaje de programación C con mecanismos que permitiesen la manipulación de objetos [13]. Se ha usado este lenguaje para la programación del firmware de control que se ha implementado en la placa Arduino.

- **Java:** Es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems [14]. Su sintaxis procede en gran parte de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos.

- **Eclipse:** Es una plataforma de Software compuesta por un conjunto de herramientas de código abierto para el desarrollo de entornos de desarrollo integrados o IDE por sus siglas en inglés *"Integrated Development Environment"* como el IDE de Java llamado *"Java Development Toolkit"*. La principal razón por la que se decidió el uso de eclipse para el desarrollo de las distintas partes del

código del proyecto sobre otros entornos de programación como pueden ser "Visual Studio Code" o "Intellij" es debido a la gran cantidad de entornos de desarrollo integrado que existen para eclipse como pueden ser "Sloeber" o "WindowBuilder". Además de esto eclipse es famoso por su entorno de desarrollo integrado de Java y es bastante sencillo a la hora de instalar las distintas bibliotecas que serán necesarias a lo largo del desarrollo del proyecto.

- **Sloeber:** Es un IDE de Eclipse gratuito y de código abierto para facilitar y agilizar todos los procesos a la hora de trabajar sobre una placa Arduino, procesos tales como compilar y cargar el programa en la placa o acceder a las comunicaciones que se están produciendo por el puerto serie para mostrar los mensajes correspondientes [15].

- **WindowBuilder:** Es una herramienta que puede ser instalada fácilmente en eclipse. Se trata de una herramienta muy potente a la hora de crear interfaces gráficas sobre Java, que permite no tener que escribir grandes cantidades de código y ver de forma visual los resultados. Se ha usado para construir las diferentes interfaces de cada uno de los menús de la aplicación Java que se explicarán más adelante.

1.4 Metodología

Por las características del proyecto, se ha elegido como metodología de trabajo para el desarrollo de las distintas partes del proyecto, un modelo en cascada. Este modelo, se basa en que el paso de una fase a la siguiente, se realizará solo tras haber finalizado la anterior. Cada una de fases ha sido planificada con anterioridad, y para ser implementada en un tiempo concreto se ha documentado cuidadosamente por escrito. Se ha elegido esta metodología debido a que es un proceso secuencial y fácil de implementar que puede llegar a ser muy útil a la hora de desarrollar un proyecto cuyas dimensiones sean parecidas a las de este. A continuación, se describe brevemente en qué han consistido cada una de las fases del proyecto.

1. **Análisis de requisitos a implementar:** Evaluación de las distintas funciones a implementar y por tanto compra de los distintos materiales que serán necesarios para el desarrollo del proyecto véase placa Arduino o los distintos sensores que se usarán.
2. **Comprensión y preparación de los sensores:** Será necesario aprender a procesar y manejar cada sensor y adaptar el uso de cada una de las librerías a las necesidades que vayan surgiendo del proyecto.
3. **Diseño del firmware de control en C++:** Una vez esté claro el manejo de cada sensor será necesario implementar un programa en C++ que será el encargado de la gestión de todos los sensores y el motor, a la vez que guarda toda esta información en un banco de registros para su posterior transmisión al sistema gestor de estos datos.
4. **Diseño del programa de gestión en Java:** Con el firmware de control ya en funcionamiento comienza el desarrollo de la aplicación de gestión en Java para la comunicación con el programa en C++ mediante el protocolo elegido, ya sea Modbus o telemetría. Esta aplicación en Java se encargará de la recopilación de datos recibidos desde la plataforma hardware y mostrar todos aquellos datos que puedan ser útiles a la hora de realizar las distintas pruebas. También transmitirá comando para gestionar el funcionamiento del motor y realizar pruebas sobre éste.
5. **Fase de pruebas:** Con todo esto ya hecho se procedería a la realización de todas las pruebas correspondientes para eliminar cualquier tipo de fallo que pueda encontrarse y en caso que se desee aumentar, modificar los test realizados sobre el motor preparar la aplicación para que sea fácilmente reconfigurable.

2

Motores brushless y sensores utilizados

Se introducirán las razones por las que se decidió comenzar a desarrollar este proyecto, además de hablar brevemente sobre las metodologías y tecnologías utilizadas.

2.1 ¿Qué es un motor brushless?

Los motores brushless son motores conmutados electrónicamente y alimentados por una fuente de corriente continua a través de un controlador externo llamado ESC(Electronic Speed Controller). A diferencia de los motores eléctricos convencionales los motores brushless necesitan de este controlador externo para lograr la conmutación. La conmutación es el proceso de conmutar la corriente en las distintas fases del motor con el objetivo de generar movimiento. En los motores eléctricos convencionales esto se consigue mediante el uso de escobillas que generan rozamiento. Cuando cada bobina de una dínamo pasa por una escobilla se invierte la corriente en dicha bobina, proceso que ocurre dos veces por rotación, el cual se puede visualizar en la Figura 4. Por otra parte, los motores brushless no poseen de estas escobillas, por lo

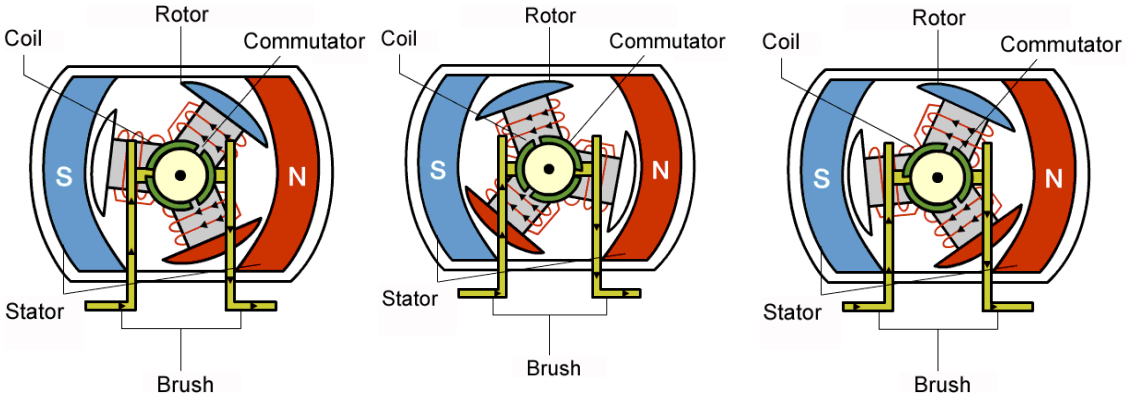

Figura 4. Fases de un motor con escobillas. Fuente: [24]

que para saber cuándo deben realizar la conmutación requieren de otros métodos electrónicos.

Los motores brushless pueden venir en distintas configuraciones, ya sean monofásicas, bifásicas y trifásicas siendo la configuración trifásica la más común de todas estas. En esta configuración, el número de fases coincide con el número de bobinados que posee el estátor que es la parte fija del motor tal y como puede verse en la Figura 5.

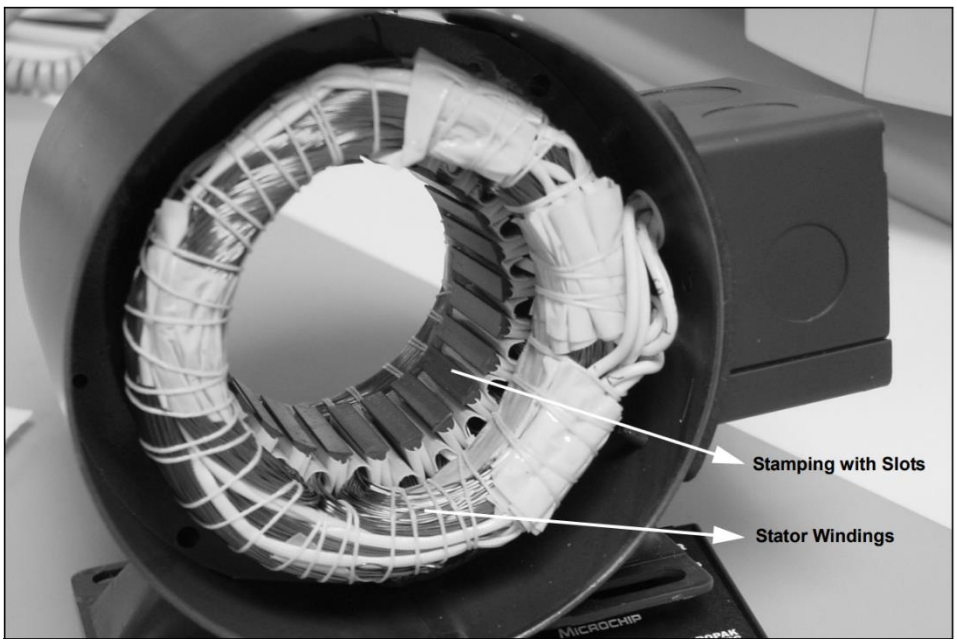

Figura 5. Estátor de un motor brushless. Fuente: [18]

Por otra parte el rotor es la parte móvil del motor, está hecha comúnmente de imanes permanentes y pueden variar de dos a ocho pares de polos de forma alternada, Norte (N) y Sur (S).

Para el control de los motores brushless existen tres tipos de algoritmos, aunque en este proyecto solo se mencionará brevemente la conmutación trapezoidal basada en sensores hall. Para más información acerca de los algoritmos de control puede acudirse a la referencia [17]. Para el correcto funcionamiento de estos algoritmos también es necesario poseer una señal de modulación por ancho de pulsos o PWM (Pulse *Width* Modulation) por sus siglas en inglés para controlar la velocidad del motor, un sistema para conmutar el motor y un procedimiento para detectar la posición del rotor.

La conmutación trapezoidal se caracteriza porque solo dos de las tres fases están activas en cada momento generando un patrón de conmutación de 6 pasos tal y como se aprecia en la Figura 6.

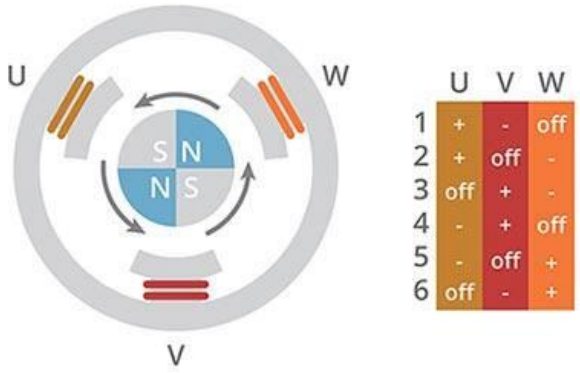

Figura 6. Patrón de conmutación en 6 pasos. Fuente: [6]

La Figura 6 se trata de un motor de tres fases, U, V y W, con un rotor con dos pares de imanes permanentes alternados tal y como se indicó anteriormente. La Figura 7 muestra cada una de las 6 fases por las que el motor pasará durante una revolución completa y la dirección de la corriente.

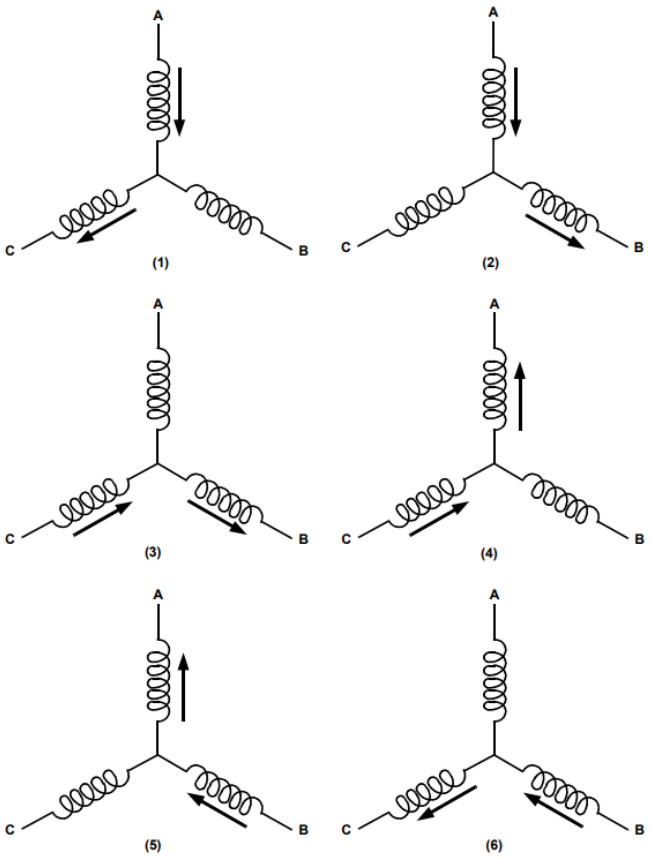

Figura 7. Secuencia de conmutación en 6 pasos. Fuente: [18]

2.2 Electronic Speed Controller o ESC

El electronic speed controller o ESC es un circuito electrónico encargado de realizar de forma correcta la conmutación de cada una de las 3 fases del motor brushless además de controlar su dirección y su velocidad de giro. El ESC hace uso de la modulación por ancho de pulsos o PWM para controlar la velocidad. Esta técnica consiste en modificar el periodo de una señal senoidal o cuadrada con el objeto de transmitir información, en este caso para transmitir la velocidad a la que se desea que gire el motor. Un ESC sigue el siguiente esquema conectado a un Arduino, ver figura 7.

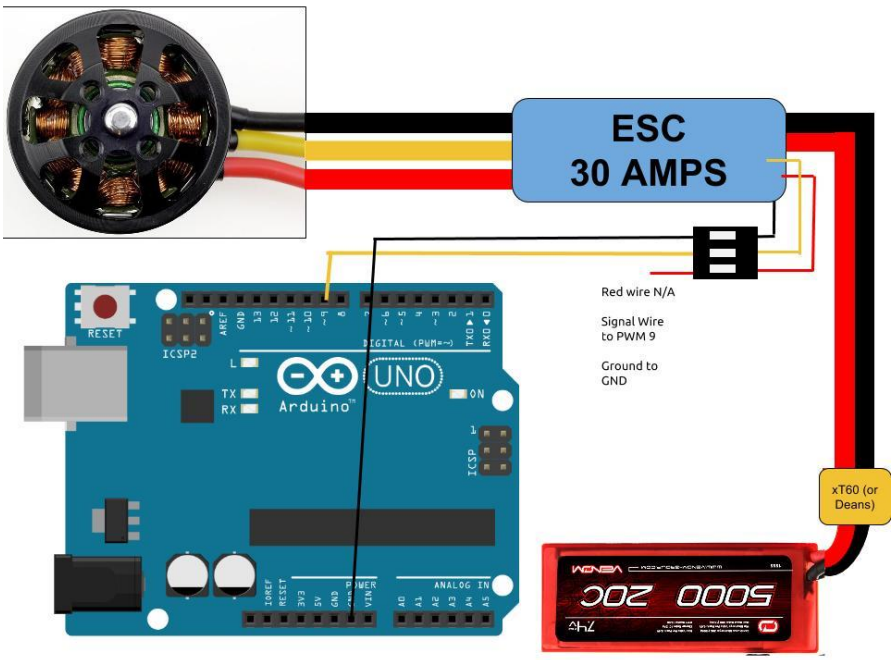

Figura 8. Diagrama de un ESC conectado a una placa Arduino. Fuente [19]

En el diagrama de la Figura 8 se aprecia como salen tres cables del ESC hacia el motor, estos cables son los encargados de la regulación de cada una de las tres fases por las que pasa el motor. Por la otra parte se aprecian otros 5 cables, los dos más anchos son la alimentación del motor mientras que los otros 3 son los que se usan para la regulación por ancho de pulsos y por tanto los usados por el sistema de control. El cable rojo fino no es necesario que vaya conectado ya que ya se obtiene la alimentación por la batería a la que está conectada, el cable negro es la toma de tierra y el cable amarillo es por donde debe emitirse la señal PWM desde el dispositivo Arduino.

2.3 Galga extensiométrica

Según Wikipedia " Una galga extensiométrica o extensómetro es un sensor que mide la deformación, presión, carga, par, posición, etcétera, y se basa en el efecto piezorresistivo, que es la propiedad que tienen ciertos materiales de cambiar el valor nominal de su resistencia eléctrica cuando se les somete a ciertos esfuerzos mecánicos que causan deformación.” [8]. Esta deformación, es directamente dependiente de la dirección de la que provienen las fuerzas ejercidas, es decir, no se obtendrá el mismo valor de aplicar la misma fuerza desde un lado de la galga que desde arriba de ésta.

Aunque el principio en que se basan todas es el mismo, medir los cambios en ciertas variables del material cuando es sometido a deformaciones, el mecanismo usado para medir esta deformación puede variar. Existen diversos tipos de galgas en función de cómo se haga este proceso. En este proyecto solo se hablará de las galgas por resistencia como la que puede verse en la Figura 9.

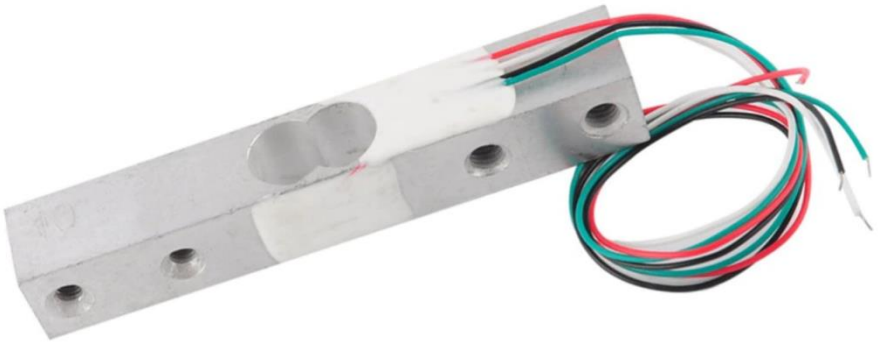

Figura 9. Galga extensiométrica por resistencia similar a la usada en el proyecto.

Este tipo de galgas se trata de un conductor eléctrico que, al ser deformado, debido a que los conductores que forman el material se vuelven más largos y finos provoca que aumente la resistencia de éste. El objetivo es medir esta resistencia mediante un puente de Wheatstone del que se entrará en profundidad más adelante acerca de su funcionamiento. Midiendo esta resistencia se puede establecer una relación lineal entre la deformación y la resistencia obtenida mediante un factor llamado factor de galga [8].

Para mayor información acerca del funcionamiento de las galgas extensiométricas se puede acudir a la referencia [25].

El Puente de Wheatstone

Un puente de Wheatstone se trata de un circuito eléctrico cerrado que se utiliza para medir de forma precisa resistencias desconocidas. También puede ser usado para la calibración de distintos instrumentos y sensores como voltímetros o amperímetros o para determinar cambios relativos en la resistencia. Es esta última aplicación, la de determinar cambios relativos en la resistencia, la que es interesante por su aplicación

sobre las galgas extensiométricas ya que permite medir con elevada exactitud cambios relativos en una resistencia.

El circuito consiste en cuatro brazos o ramales formados por 4 resistencias, de R1 a R4 en la Figura 10 con dos puntos a los que se llamará 'A' y 'B' para designar las conexiones de la tensión de alimentación del puente, Vs, y otros dos puntos que se designarán como 'C' y 'D' para indicar la tensión de salida del puente, Vo, es decir, la señal de medida [9].

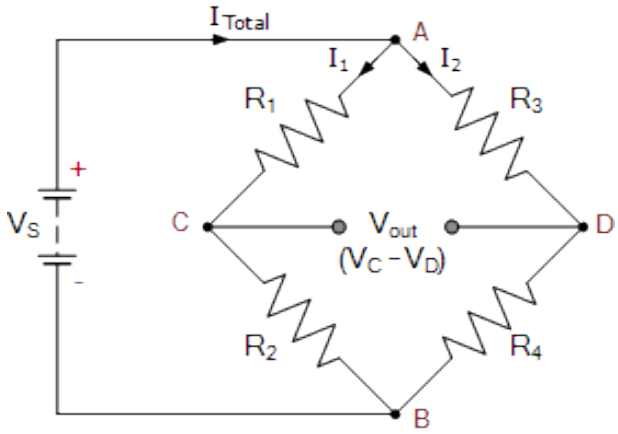

Figura 10. Circuito electrónico puente de Wheatstone. Fuente: [9]

En el momento que se ha aplicado una tensión de alimentación Vs a los puntos A y B del puente esta tensión se verá dividida entre las dos mitades del puente, R1-R2 y R3-R4 de forma proporcional a las resistencias correspondientes, por tanto, cada mitad del puente está formando un divisor de tensión.

Al deformarse la galga estas resistencias cambian provocando que el puente no esté equilibrado. Para calcular esto puede hacerse mediante la siguiente fórmula:

$$V_{out} = V_s \left(\frac{R_1}{R_1 + R_2} - \frac{R_4}{R_3 + R_4} \right)$$

Por tanto, en caso de que el puente esté equilibrado deberá cumplirse la siguiente igualdad:

$$\frac{R_1}{R_2} = \frac{R_4}{R_3}$$

Donde $V_{out} = 0$.

A partir de aquí se puede, mediante estas ecuaciones y los cambios en las resistencias, ser capaz de medir el factor de galga y la deformación que está causando el cambio en las resistencias de la galga ya que estos cambios en la resistencia tienen una relación directa con la deformación causada.

2.4 Sensor de voltaje e intensidad

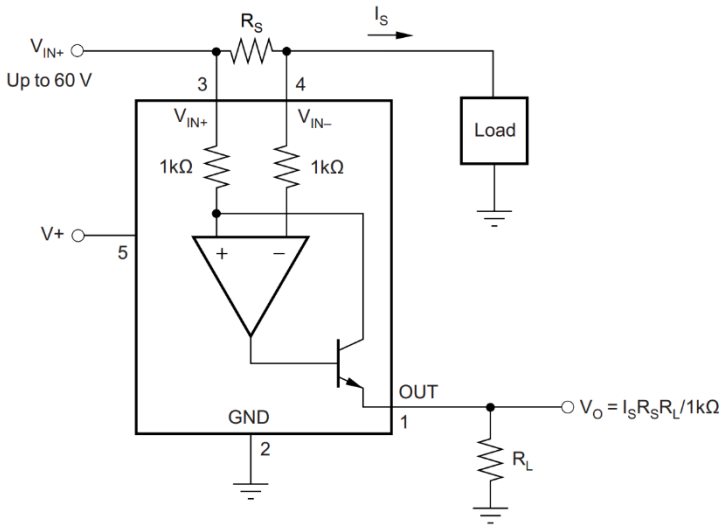

Figura 11. Circuito típico chip INA1x9

Para este proyecto, por su disponibilidad en el mercado, se ha decidido usar como sensor de voltaje e intensidad, APM 2.5, basado en el chip INA169 cuyo circuito puede ser apreciado en la Figura 101, cuyo funcionamiento básico es el siguiente:

La corriente de todo el sistema pasa por la resistencia R_s lo que provocará una caída de la tensión entre los puntos 3 y 4 y por la ley de Ohm se puede establecer una relación entre esta caída de potencial y la intensidad que circula a través de R_s. Para no influir en el valor nominal de la tensión proporcionada por la batería, el valor de la resistencia usada R_s deber ser muy pequeña. El chip usa un amplificador diferencial cuya

salida es proporcional a la diferencia de potencial entre los dos terminales del mismo para medir la pequeñísima caída de tensión provocada por la resistencia R_S [10].

2.5 Sensor de revoluciones

El sensor de revoluciones usado para este proyecto es el sensor rpm hobbywing como el que se puede apreciar en la Figura 12.

Figura 12. *RPM hobbywing ESC de alto voltaje, Fuente: [20]*

El funcionamiento del sensor de revoluciones es el siguiente: Por el diagrama de la Figura 13 se puede apreciar que el circuito usado posee dos entradas (rojo). Estas dos entradas deben ir conectadas a dos cables cualesquiera de los 3 que salen del ESC hacia motor que se esté probando. El sensor lo que hará será detectar los cambios de tensión en los cables conectados al ESC y dará como salida un pulso por cada uno de estos cambios. El número de pulsos detectados, será proporcional a las revoluciones del motor. Así, para obtener el número real de revoluciones por minuto, habría que contar la cantidad de pulsos que se obtienen en un plazo de tiempo, por ejemplo, de un segundo, y multiplicar este número de pulsos obtenido por 60 para saber, de esta forma, cuántos habría aproximadamente en un minuto. Por último, este número debe ser dividido entre el número de cambios de tensión que se producen en una vuelta.

Figura 13. Diagrama sensor de revoluciones. Fuente: [20]

3

Desarrollo y fases del proyecto

En este capítulo se explicará cuáles han sido las diferentes fases del proyecto en las que se ha trabajado, atendiendo también a la metodología empleada y se justificarán las distintas elecciones tomadas durante el desarrollo.

3.1 Comprensión del proyecto a realizar y objetivos finales

El primer paso es comprender cuáles son exactamente las dimensiones del proyecto a realizar, es decir cuáles son exactamente los objetivos finales y estudiar cuáles debían ser los requisitos debe cumplir el proyecto. El objetivo de este análisis de requisitos es especificar todas las características operacionales que poseerá y describir más concretamente cuál va a ser el plan trazado para su desarrollo, es decir, en qué orden se implementará cada parte, cuánto tiempo requerirá y cómo deberá hacerse.

Antes de decidir en qué consistirá cada una de las fases es necesario saber cuáles serán los objetivos finales del trabajo:

- **Construcción del banco de pruebas para motores brushless** usados en drones (copters) con los correspondientes sensores completamente operativo.

- **Desarrollo de programa en C++** sobre una placa Arduino que se encargará del control y gestión del sistema en todas aquellas acciones que resulten imprescindible para para el desarrollo de los test, tales como el accionamiento del motor, configuración y lectura de la información generada por los sensores, así como su transmisión y registro. En definitiva, todos aquellos que sea necesario o importante para el manejo del banco de pruebas.

- **Desarrollo de programa en Java** que se comunicará con la aplicación C++ y se encargará de mostrar todos los datos recopilados por el dispositivo de control mediante cuadros de texto o gráficos. Así como del envío de comandos generados de la interacción con el usuario

Dadas las características del proyecto a realizar se puede llegar a la conclusión de que la manera más sencilla de realizar este desarrollo es utilizando un modelo en cascada. Tal y como se explicó en el punto 1.4, un modelo en cascada se basa en realizar el desarrollo de las fases de manera secuencial solo empezando la siguiente tras haber terminado la anterior. Esto permite durante el análisis de requisitos definir en qué consistirá cada una de las partes del proyecto y definir una planificación de tiempo de cada una de ellas. Tras este análisis y siguiendo la planificación propuesta en el punto 1.4 se especificarán aún más en qué consistirá cada una de las fases del proyecto.

- **Comprensión y preparación de los sensores:** Se debe ser capaz de hacer funcionar cada sensor de forma correcta por separado, esto implica también comenzar a trabajar con cada una de las librerías involucradas en el manejo de estos sensores, aprender acerca de su estructura interna y mediante todo esto aprender también acerca de cómo obtener datos generados.

- **Diseño del programa en C++:** Con el uso y manejo de los sensores bajo control, es necesario implementar todos ellos a la vez en el programa en C++. Esto conlleva lidiar con ciertos problemas que van a surgir a la hora de compatibilizar todos estos sensores y sus respectivas librerías con las comunicaciones que se realizan a través del puerto serie debido al manejo de las interrupciones.

- **Diseño del programa en Java:** La interfaz que vaya a crearse en Java debe ser capaz de mostrar todos los datos relevantes que puedan derivase de los datos recogidos. Podrá hacerlo mediante cuadros de texto y gráficamente. Para ello es necesario que la interfaz tenga distintas ventanas donde seleccionar cada una de estas opciones.

- **Fase de pruebas:** Esta última fase se centrará en la construcción de forma física del banco de pruebas donde colocar el motor. Para ello se creará con ayuda del tutor una placa que incluya todos los componentes electrónicos esenciales de cada uno de los sensores y mediante una impresora 3D se creará el soporte donde se colocará el motor.

3.2 Preparación del proyecto, sensores y librerías usadas

La primera fase es la preparación del proyecto. El objetivo de ésta, como bien se mencionó en el punto 3.1 es aprender acerca del funcionamiento de cada uno de los sensores por separado y aprender acerca de cómo implementarlos de forma individual en la aplicación en C++.

ESC

El primer paso de esta fase es ser capaz de hacer operar el motor a través de la placa Arduino mediante el uso del ESC ("electronic speed controller"), también llamado variador. Este dispositivo deberá ser capaz de controlar la velocidad del motor además

de su sentido de giro. El ESC posee tres entradas siendo dos de ellas alimentación y tierra, y una tercera que será la entrada activa con la que se realiza el control efectivo del motor. Para ello, se utiliza una señal PWM (Modulación por ancho de pulso) que será generada por nuestro dispositivo Arduino (Figura 14) y como se explicó en el punto 2.2. El funcionamiento es sencillo, el procesador genera una un pulso en la señal PWM con un ancho que puede variar entre 1 y 2 milisegundos. En 1 milisegundo, el motor estará parado, y conforme se vaya aumentando el tiempo de este ancho de pulso irá también aumentando la velocidad hasta alcanzar a un ancho de pulso de 2 milisegundos donde el motor girará a máxima velocidad.

La elección del ESC usado, no se realiza al azar, ya que estos dispositivos se caracterizan por la mayor intensidad de corriente que son capaces de soportar y por supuesto, a mayor intensidad de corriente soportada, su precio suele ser mayor. Lo habitual, cuando se elige un variador para ser usado en un dron, es elegir un que tenga unas prestaciones que se encuentren por encima de los requerimientos del motor usado. En este caso, se elegirá un variador (ESC) con unas características lo suficientemente amplias, como para poder usar en el banco de pruebas una gran variedad de motores

Figura 14. Ejemplo de un ESC visto desde fuera. Pueden apreciarse los 3 pines para el control del ESC. Negro tierra, rojo alimentación y blanco por donde debe ir las señales PWM. Fuente: [21]

Galga extensiométrica

En el apartado 2.3 ya se analizó el funcionamiento de una galga extensiométrica y del puente de Wheatstone, sin embargo, estos cambios en la resistencia provocados por la deformación del material deben ser procesados y convertidos en algo que sea inteligible por los sistemas digitales. Para ello, se usará un módulo HX711 (Figura 15) con interfaz entre la galga extensiométrica y el sistema procesador (Arduino Nano). Este módulo se encarga de realizar una conversión analógico-digital de 24 bits y comunicar el resultado al procesador mediante una comunicación serie usando de líneas (PD_SDK y DOUT).

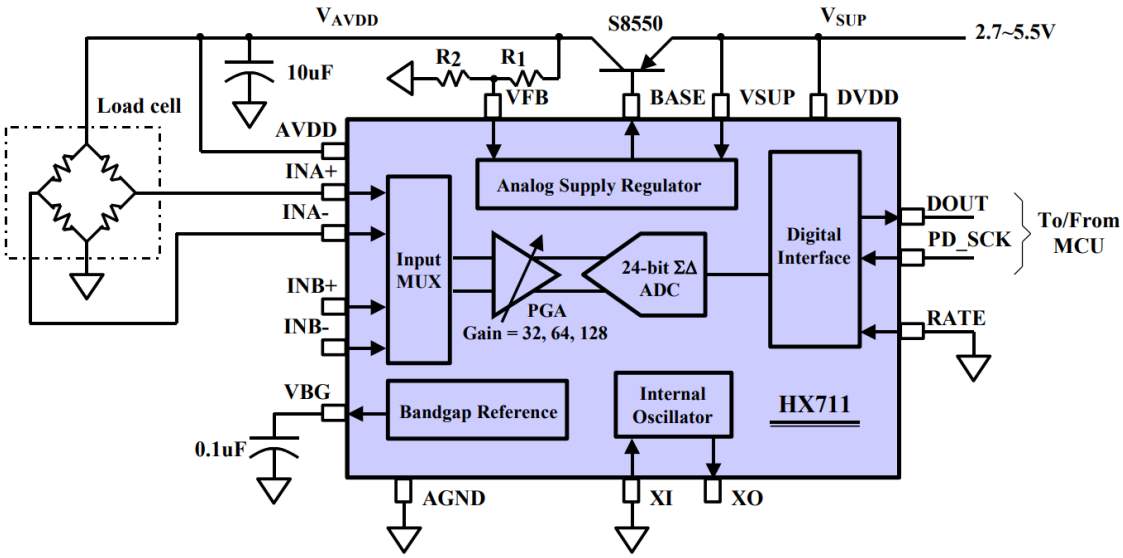

Figura 15. Diagrama de bloques del conversor analógico-digital HX711. Fuente: [22]

El módulo HX711, como se ha visto en el párrafo, usa como interface de comunicaciones, un interface serie diseñado por el fabricante, sin reglas estandarizadas como las del estándar/protocolo I2C. Para ello, usa los pines , DOUT o Serial Data Output y PD_SCK o Power Down and Serial Clock Input, que se conectarán con los pines del dispositivo arduino A0 y A1 (PC0, PC1) donde la librería de referencia implementará el protocolo diseñado por el fabricante.

El modo en el que el HX711 transmite los datos es el siguiente: Cuando los datos de salida no están listos para ser transmitidos el pin de salida DOUT permanece activo

mientras que el pin de salida PD_SCK permanece inactivo. En el momento en el que los datos están listos para su transmisión el pin DOUT pasa a estar inactivo. Tras esto el pin PD_SCK pasa a enviar entre 25 y 27 pulsos de reloj y es mediante estos pulsos como el pin DOUT transmite los datos. Por cada pulso que emite el pin PD_SCK el pin DOUT emite un bit de datos, empezando por el bit más significativo, hasta emitir los 24 bits de datos que necesita emitir. Tras esto el pulso número 25 del reloj se usará para que el pin DOUT vuelva a su estado activo como al principio. El canal de entrada y la ganancia son controlados mediante el número de pulsos tal y como puede apreciarse en la Figura 16.

PD_SCK Pulses	Input channel	Gain
25	A	128
26	B	32
27	A	64

Figura 16. Input Channel and Gain Selection, Fuente: [22]

Una vez está claro el funcionamiento básico del conversor HX711 se pasará a explicar brevemente el código usado para procesar estos datos. Arduino ya cuenta con una librería (HX711_Arduino_library) cuyo objetivo es exactamente este, leer los datos provenientes del conversor analógico-digital HX711. Esta librería cuenta con diferentes funciones de las cuáles solo explicaremos las más importantes para el proyecto:

- *tare():* Se encarga de reiniciar el nivel de la escala a 0. Puede ejecutarse en cualquier momento y puede ser muy útil a la hora de recalibrar la balanza a mitad del programa.
- *set_scale(m):* Fija el factor de calibración, es decir, el valor usado para convertir los valores obtenidos por la galga a valores en una medida de peso como gramos, miligramos o libras.
- *"get_value(t)"* y *"get_units(t)":* ambas incluyen una variable t o “times” en su llamada. Devuelven la media de sus mediciones, siendo t el número de veces que realizan estas mediciones para realizar esta media.

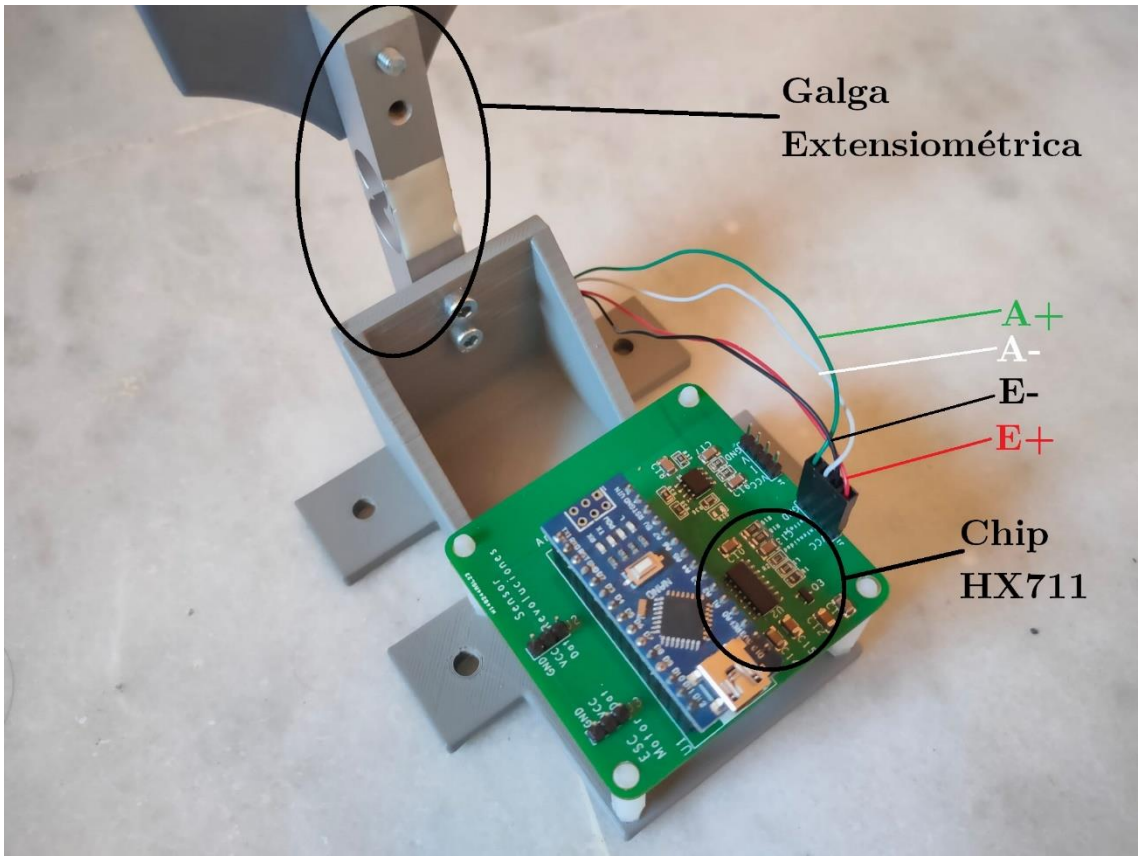

Figura 18. Galga extensiométrica conectada con el chip HX711 en la placa de control diseñada para el proyecto.

El diseño funcional realizado para el proyecto, junto con la conexión a la galga puede verse en la Figura 18 . El circuito de conexión con el que se ha interconectado la galga con el procesador, usando el HX711 se muestra en la Figura 17.

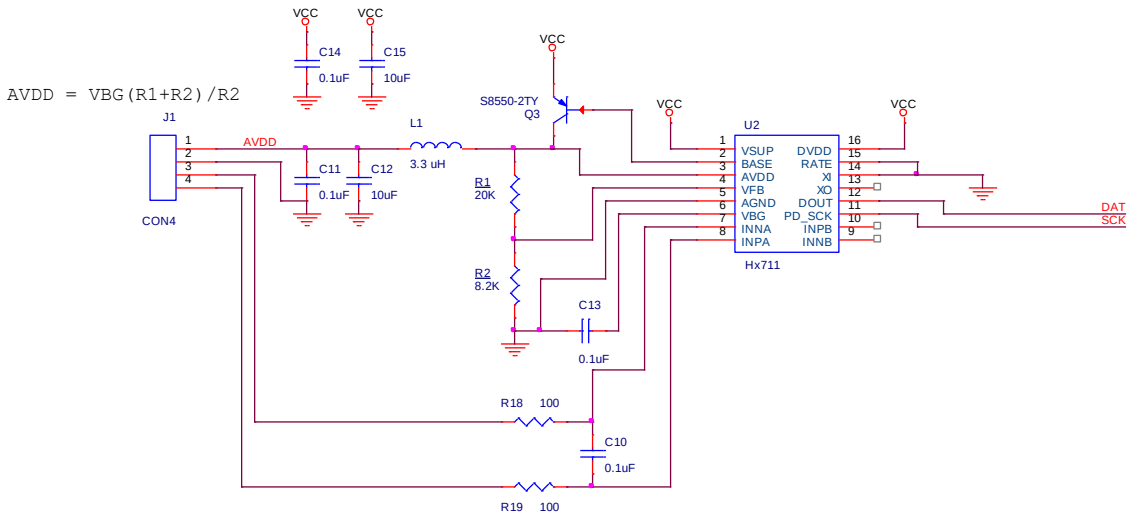

Figura 17. Circuito interface entre la galga extensiométrica y el procesado, usando el HX711

Una vez conocidas estas funciones de la librería, hacer un programa básico que ponga en funcionamiento la galga extensiométrica es muy intuitivo, sin embargo, a la hora de implementarla junto a un protocolo de comunicaciones, cuya longitud de palabra de datos es de 16 bits, obliga a usar dos palabras para la comunicación de cualquier dato. Si se toma como base uno de los protocolos usados en el proyecto, Modbus, el cual, aunque existen variantes con longitud de palabras de datos de 32 y 64 bits, el elegido para este proyecto es de 16 bits. En este sentido, Modbus usa para el almacenamiento de datos un banco de registros tipo *"unsigned int"* de 16 bits. Esto ha sido gestionado usando dos registros consecutivos para el almacenamiento de los datos obtenidos del dispositivo HX711 (24 bits). El siguiente fragmento de código de C++ perteneciente a la aplicación en Arduino, concretamente a la clase *"BeTeBa.cpp"* muestra la gestión de los datos obtenidos del HX711 y su almacenamiento en los registros consecutivos asignados para tal efecto en el protocolo ModBus.

```
case S_HX711:
    int32_t valorT = hx711.get_value(2);
    *Sensor->regModbus = valorT >> 16;
    *Sensor->regModbusH = valorT & 0xFFFF;
    break;
```

Sensor de voltaje

Como ya se ha indicado, para este proyecto, por su disponibilidad en el mercado, se ha decidido usar como sensor de voltaje e intensidad, APM 2.5, mostrado en la Figura 19.

Figura 19. Sensor de voltaje APM 2.5

El módulo posee un conector de seis pines, dos dedicados a alimentación (5V), dos de toma de tierra y dos salidas analógicas con los valores de la tensión y la intensidad. Estas señales, serán conectadas a los pines A2 (intensidad) y A3 (Voltaje) del procesador. El procesador AtMega328P usado en el módulo arduino Nano dispone de seis conversores analógicos-digitales de 10 bits, que para los requerimientos del sistema proporciona una resolución más que suficiente. Las especificaciones del sensor indican que tiene una capacidad de medición de 60 Amperios, lo que nos proporciona una resolución aproximadamente de 60 miliamperios. En la Figura 20 se muestra el sensor de voltaje e intensidad conectado a módulo de control diseñado para el proyecto.

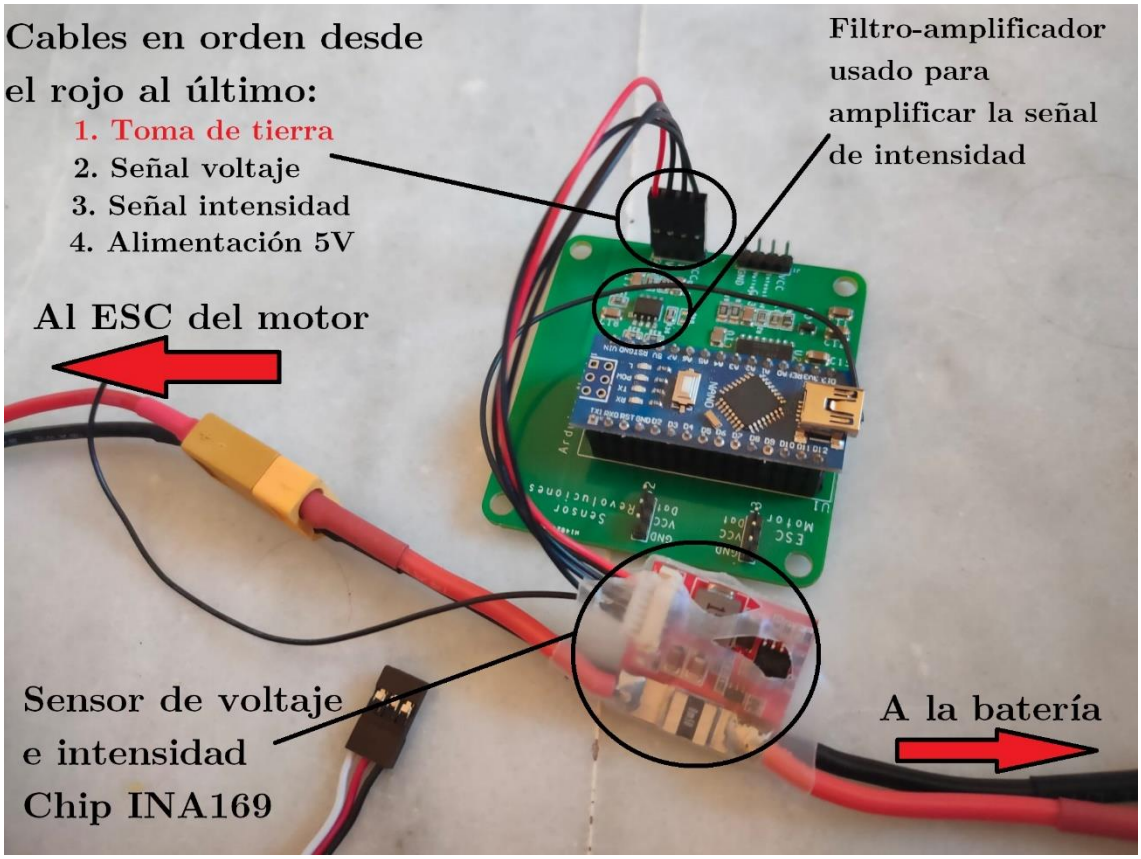

Figura 20. Sensor de intensidad y su conexión con el módulo de control

Para reducir las oscilaciones en las medidas realizadas en el intensidad, debido al ruido producido por los distintos elementos del sistema, se incluyó un circuito con un filtro activo paso de baja como el que se muestra en la Figura 21. En la configuración elegida, puesto que la resistencia R40 no se montó, la ganancia será 1 y la frecuencia de corte aproximadamente 159 Hz.

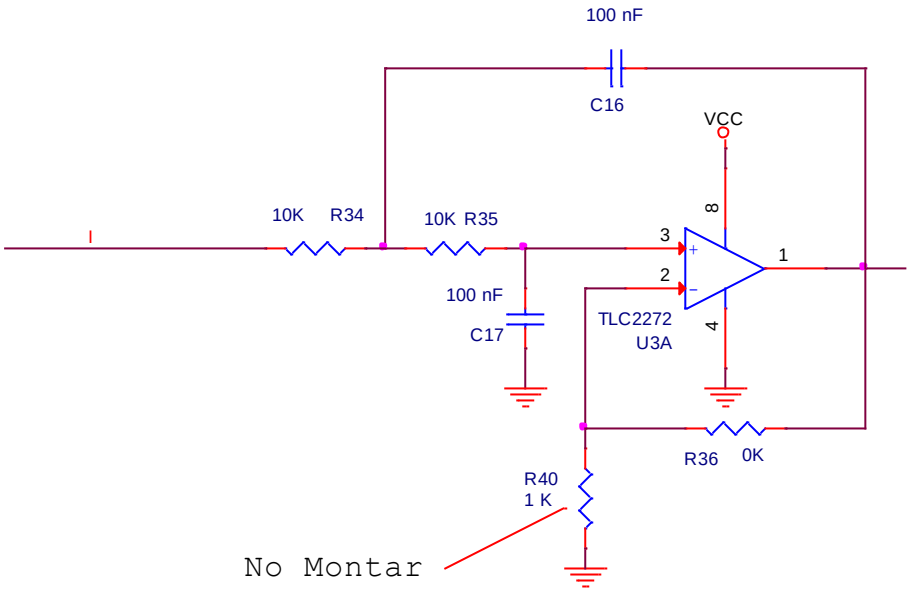

Figura 21. Filtro activo paso de baja usado para filtrar la señal de intensidad

Para realizar los cálculos anteriores, la ganancia del circuito de la Figura 21 viene dada por:

$$A_v = 1 + \frac{R_{36}}{R_{40}}$$

y la frecuencia de corte por:

$$f_c = \frac{1}{2\pi\sqrt{R_{34}R_{35}C_{16}C_{17}}}$$

Para la calibración de los valores de voltaje e intensidad proporcionados por el sensor, se usará un analizador de potencia de alta precisión como el de la Figura 22. Las señales de voltaje e intensidad deben ser conectadas a las entradas analógicas del módulo arduino, el cual, como ya se ha dicho está basado en el procesador AtMega328P que incorpora 6 convertidores analógico-digitales de 10 bits, de los que se usarán dos para el sensor de voltaje-intensidad. Debido a que los convertidores analógicos-digitales son de 10 bits, se puede asumir que el rango de valores que podrá devolver el sensor estará comprendido entre 0 y 1023. Se realiza ahora una medición cualquiera con el sensor y se obtiene un valor del voltaje de, por ejemplo, 223. Usando ahora el analizador de potencia de alta precisión se obtiene que este 223 equivale a un voltaje de 11,79V.

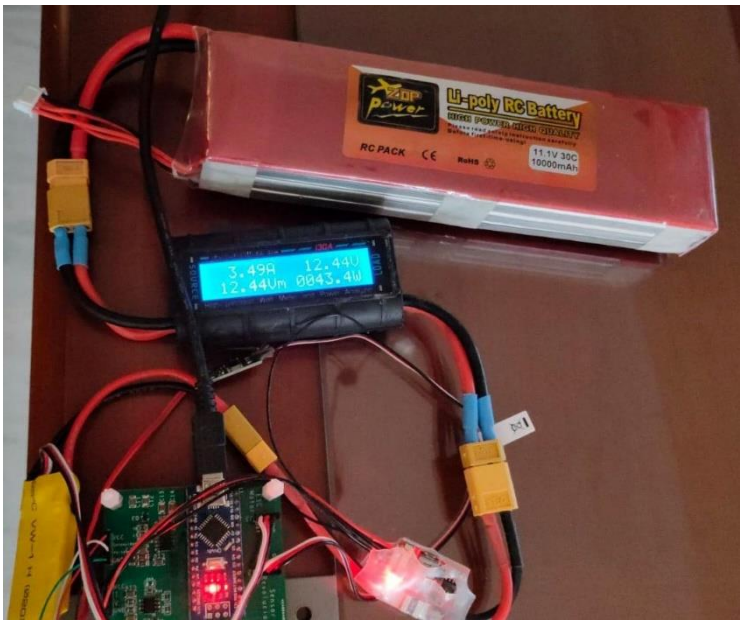

Figura 22. Módulo y conexión usada para la calibración de la tensión soportada y la intensidad consumida.

Debido a que el valor dado por el sensor crece linealmente desde 0 hasta el 223 que se ha obtenido en la medición, puede entonces obtenerse el valor por el que hay que dividir ese 223 para obtener el voltaje real. Para este ejemplo sería aproximadamente de 18,914. También se obtiene el voltaje máximo que puede dar el sensor que sería 1023/18,914 y por tanto el valor máximo medible sería de aproximadamente unos 54 voltios.

Una vez calibrado el sensor de voltaje, debe realizarse el mismo proceso para el sensor de intensidad, sin embargo, aquí se encuentra un problema y es que el sensor

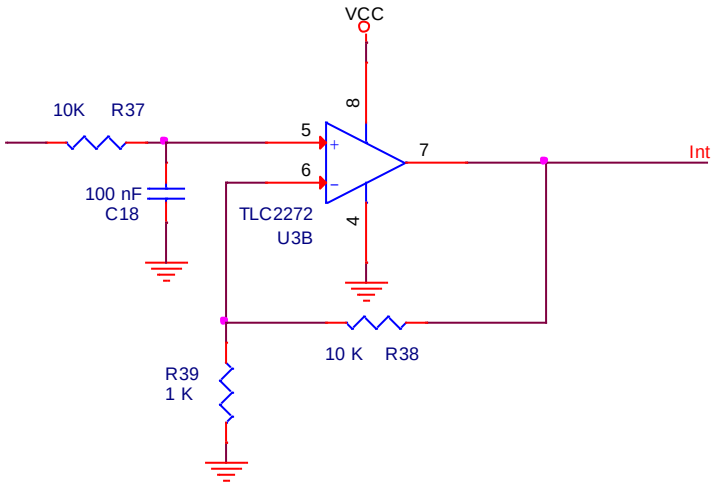

Figura 23. Circuito amplificador usado para la amplificación de la señal de intensidad

que es usado cuando los motores son pequeños, el consumo es de éstos es bajo y los valores que proporciona son demasiado pequeños, no siendo suficiente la resolución del convertidor como para que las variaciones en la intensidad, cuyo orden es de décimas de amperio, sean apreciables. Para solucionar esto se introduce en el circuito un amplificador como el que se muestra en la Figura 23.

La ganancia del circuito mostrado en la Figura 23 viene dada por la expresión:

$$A_v = 1 + \frac{R_{38}}{R_{39}}$$

Los valores de las resistencias R38 y R39 no son fijos y pueden variar en función de las necesidades de la prueba.

Si ponemos en cascada las dos etapas de los circuitos mostrados en la Figura 21 y la Figura 23, obtendremos el circuito mostrados en la Figura 24 y que se ha indicado en la Figura 20 montado en la placa de control del proyecto.

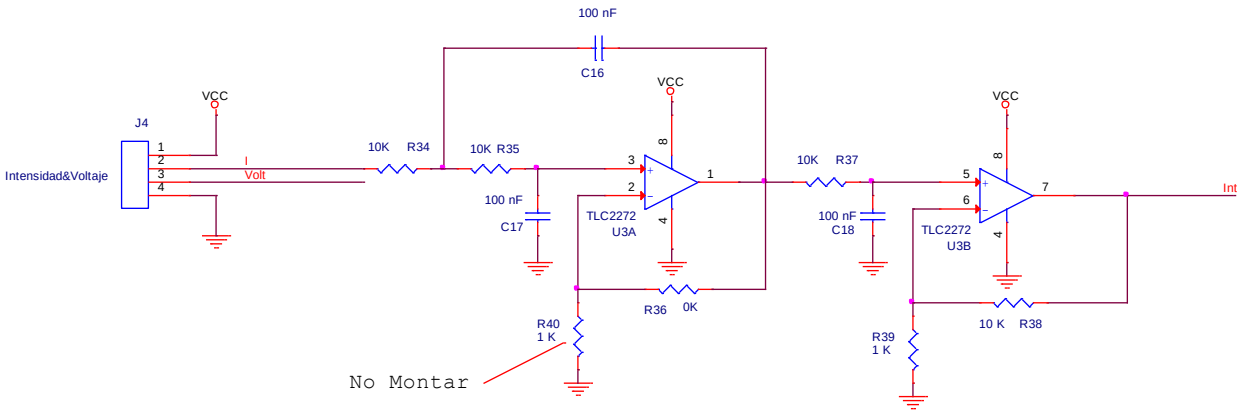

Figura 24. Circuito de filtrado y amplificación de la señal de intensidad

Sensor de revoluciones

Por último, para el manejo del sensor de revoluciones (Figura 12) no es necesaria ninguna biblioteca, únicamente conociendo el funcionamiento del hardware del dispositivo que va conectado al ESC puede ponerse en funcionamiento y comenzar a recopilar datos. Este dispositivo tiene un pin de datos que será el que se usará para saber

las revoluciones. Como ya se explicó en el punto 2.5, el sensor dará como salida un pulso por cada cambio de voltaje en los cables conectados al ESC que controlan cada una de las fases del motor. Esto quiere decir, que el número de revoluciones por minuto, no es el número de pulsos por minuto que da el sensor a través del puerto de datos, sino que es otro número que debe ser calculado mediante estos pulsos. El sensor da un pulso por cada cambio en el voltaje, esto quiere decir que en realidad está dando un pulso por cada par de imanes permanentes que usa el motor brushless objeto de test. Por tanto, el número real de revoluciones en un minuto es el número de pulsos dado por el sensor en un minuto dividido entre el número de pares de imanes que posee el motor en cuestión. Esto provoca que la calibración de este sensor sea distinta para cada motor, ya que cada motor poseerá un número distinto de pares de imanes permanentes.

3.3 Modelado del firmware de control en C++

En esta sección se mostrará el proceso de implementación de las partes más relevantes del código en C++, la gestión de cada uno de los sensores y de cada uno de los protocolos de comunicaciones. También se mostrarán los problema surgidos durante el desarrollo fruto de la incompatibilidad entre la generación de los pulsos para hacer funcionar el ESC del motor y las comunicaciones a través del puerto serie, junto con la solución propuesta para su solución.

Como se ha ido indicando a lo largo de esta memoria, el dispositivo de control del sistema estará basado en una placa de desarrollo "Arduino Nano", basada en el microcontrolador ATmega328P, la cual ha sido integrada en una placa desarrollada para el proyecto, donde realizar las conexiones de los elementos que forman parte del banco de prueba (Figura 26).

Para el desarrollo de aplicaciones en la placa arduino nano usaremos las librerías arduino que nos permite un rápido control sobre todos los componentes/periféricos que integran el microcontrolador. Por este motivo, lo que el lenguaje de programación usado para el desarrollo del firmware será el C++.

```
typedef enum
{
        S_DIGITAL,
        S_ANALOGICO,
        S_HX711
}TSensorType;

typedef struct
{
        byte    pin;
        byte    estado;
}tsSalida;

typedef struct
{
        byte                    pin;                    //Pin asignado al sensor
        int                     valor;                  //Valor leido
        int                     valor_Df;               //Valor Sensor por defecto
        bool                    Activo;                 //Activar/Desactivas sensor
        TSensorType             sensorType;             //Tipos de sensor
ANALÓGICO/DIGITAL
        char                    *name;
        uint16_t                *regModbus;
        uint16_t                *regModbusH;
}TSensor, *PTSensorPtr;

typedef struct
{
        byte                    pin;
        bool                    Activo;                 //Activar/Desactivas sensor
        uint32_t                rpm_count;
        uint32_t                rpm_micros;
        uint16_t                *regModbus;
}Trpm, *TPrpm;

/*****************************************************************************/
/***                         Macro Definitions                             ***/
/*****************************************************************************/
#define  EN_485             0       // Bit to enable RS485 Conmunications
#define  PIN_RPM            2       // Pin Sensor de revoluciones
#define  PIN_THROTTLE       9       // Pin Control Engine throttle

#define  HX711_CLK          A0
#define  HX711_DOUT         A1
```

Figura 25. Definición de las estructuras usadas para el control de los sensores y actuadores

Como se ha mencionado con anterioridad, entre otras cosas, este dispositivo se va a encargar de la gestión y control de todo lo relacionado con los sensores y actuadores integrados en el sistema. Es por lo que para el desarrollo de la aplicación lo primero que hemos implementado, son unas estructuras generales para describir los

distintos elementos que forman parte del sistema: Sensores, actuadores, etc. (Figura 25).

El siguiente paso, ha sido definir una clase en la que centralizar todo el intercambio de información con el sistema, tanto a nivel de datos como de control. Esta clase ha sido llamada BeTeBa (Brushless Engine Test Bank) (Figura 27)

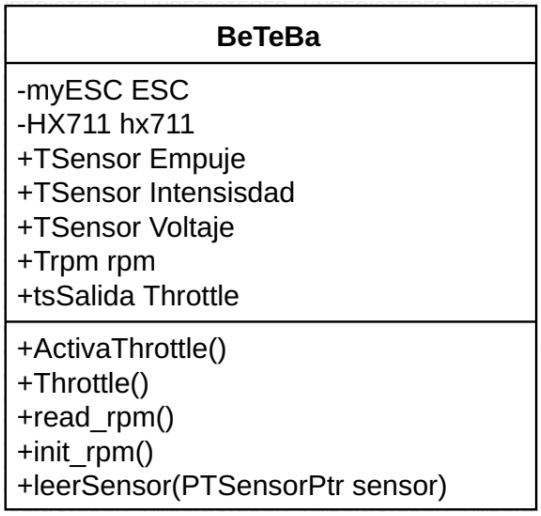

Figura 27. BeTeBa, Clase para comunicación con el dispositivo de control

BeTeBa, se encargará de realizar la inicialización de los sensores e incluye funciones para realizar su lectura; Y por supuesto, también permite controlar la velocidad de giro del motor, mediante la función Throttle.

Un programa Arduino se compone siempre de dos partes principales. Una primera donde se implementan las configuraciones iniciales de sistema, *"void setup()"* y otra donde se implementa el bucle principal de la aplicación, *"void loop()"*. El uso de la clase BeTeBa, permite desarrollar una aplicación muy concisa y clara, como se muestra en la Figura 28.

A la hora de leer los datos ambos protocolos realizan el mismo proceso. En la función *loop()* se encuentra la función *"leerSensores(BeTeBa *btb)"* que es la función encargada de leer cada uno de los sensores. Dentro de esta función lo único que se

encuentra es un conjunto de llamamientos a la función *"leerSensor(PTSensorPtr Sensor)"*, cuyo valor de entrada es un puntero a la instancia del sensor que quiera leerse y por ello, hay un llamamiento a esta función por cada sensor en funcionamiento.

```
//The setup function is called once at startup of the sketch
void setup()
{
	//Inicia/Configura Telemetría
	telemetry->configure(SERIAL_BPS,SERIAL_PARITY);
	telemetry->setGestionCmd(BeTeBa_Multiwii_Cmds);

	//Inicia/Configura BeTeBa
	beteba.activaThrottle();
	beteba.setNotifyResults(telemetry->send_msp);

	DEBUGLNF("BANCO DE PRUEBAS BRUSHLESS");
}

// The loop function is called in an endless loop
void loop()
{
	EXECUTELOOP(){
		UPDATELOOP();

		//LeerSensores
		BeTeBa::leerSensores(&beteba);

		telemetry->read_data();
	}
}
```

Figura 28. Inicialización y bucle principal de la aplicación del dispositivo de control

El dispositivo de control, también es responsable de establecer y mantener una comunicación fluida con el host donde se aloja la interfaz gráfica de usuario (GUI), usada por el usuario para mantener siempre bajo control al sistema. Para este efecto han sido implementados, por distintos motivos dos protocolos de comunicaciones, ModBus y multiwii.

En primer lugar se eligió Modbus por ser un protocolo de comunicaciones abierto, muy utilizado en entornos industriales, y cada vez más en entornos domésticos para sistemas de automatización domiciliaria, utilizado para la transmisión de información a través de interfaces serie. Además, ya se tenía cierta experiencia en su utilización, puesto que fue muy utilizado en una de las asignaturas cursadas durante los estudios de grado.

Con Modbus, podemos construir una red de comunicaciones de la que formen parte varios dispositivos. En este sentido, si el objetivo del proyecto fuese la construcción de un banco de pruebas para una aplicación mas industrial en la que fuese necesario mantener activo la toma de datos de varios motores a la vez esta opción podría ser la correcta, sobre todo, porque en un principio, se podría conectar desde un solo dispositivo hasta 32.

Por otra parte, y teniendo en cuenta la motivación inicial de proyecto, en la que solo se planteó la construcción de un banco de pruebas de uso particular, desde el punto de vista, que su utilización estaba orientado a usuarios de drones que necesitan un argumento objetivo para la selección de distintos motores, según la aplicación de UAV, se decidió implementar un segundo protocolo de comunicaciones, además de ser mas directo y basado en los sistemas de telemetría usados por los propio drones, podría ser muy útil, teniendo en cuenta la parte formativa del actual proyecto, y se eligió, Multiwii. Multiwii, no es un nombre comercial, sino que está basado en el nombre de un proyecto open source para la construcción de un dron (http://www.multiwii.com).

Modelado del protocolo Modbus

Para el modelado del protocolo Modbus es necesario mantener un banco de registros que se encargue de almacenar, para su envío, todos los datos del sistema que puedan llegar a ser relevantes. Cada uno de los sensores guardará sus datos en un registro de 16 bits, excepto en el caso del chip HX711 que requerirá de dos registros ya que sus datos ocupan 24 bits. Existen distintos bancos de registros dependiendo del tipo de dato que se esté guardando, pueden ser datos de entrada o de salida y por otra parte también pueden ser datos analógicos o digitales. En el siguiente fragmento de código

```
/*****************************************************************************/
/***        Variables Locales                                              ***/
/*****************************************************************************/
/* First step MBS: create an instance */
ModbusSlave mbs;

uint16_t    Cregs[MB_O_COILS];      //Registros para "Dicrete Output Coils"
uint16_t    Dregs[MB_I_CONTATCS];   //Registros para "Dicrete Input Contacts"
uint16_t    Aregs[MB_A_REGS];       //Registros para "Analog Output Holding Registers"
uint16_t    Eregs[MB_E_REGS];       //Registros para "Analog Input Registers"
```

Figura 29. Bancos de registros del programa en C++. Fuente: Autor

de la Figura 29 puede apreciarse cada uno de los bancos de registros que utiliza el programa.

Para su implementación, usamos como base la clase la clase ModbusSlave, que implementa todas las funcionales para la gestión del protocolo. Para la adaptación al proyecto, se ha implementado una serie de funciones que pueden ser encontradas en los ficheros (Gest_Modbus.cpp/Gest_Modbus.h), donde, sobre todo, se incluye la función para inicializar el protocolo (Figura 30) y la base para la lectura y escritura de la respuesta del protocolo a través de puerto serie (Figura 31).

```
void Init_RTU_Modbus()
{
       /* configure modbus communication
        * 19200 bps, 8E1, two-device network */
       /* Second step MBS: configure */
       /* the Modbus slave configuration parameters */
       const unsigned char       SLAVE   = ADDR_SLAVE;
       //Address SLAVE
       const long                BAUD    = SERIAL_BPS;
       const char                PARITY  = SERIAL_PARITY;
       const char                TXENPIN = EN_485;

       //Inicialmente configuramos 485 para recibir
       //digitalWrite(EN_485, LOW);

       //Para la conexión 485/ModBus usamos
       Serial485 = &Serial;

       //Configuramos bancos de registros ModBus (Analógicos/Digitales)
       A_Regs  = Aregs;
       C_Regs  = Cregs;
       D_Regs  = Dregs;
       E_Regs  = Eregs;
       N_ARegs = MB_A_REGS;
       N_CRegs = MB_O_COILS;
       N_DRegs = MB_I_CONTATCS;
       N_ERegs = MB_E_REGS;


       mbs.configure(SLAVE,BAUD,PARITY,TXENPIN);

       //mbDomoBoard = new MBDomoBoard(&domoboard);

       //Config_interruptor1();

       //Configura acción a realizar cuando se actualizan los registros
       (Discrete Output          Coils)
       mbs.writecoil = writecoil;
       mbs.writesingleregister = writesingleregister;
```

Figura 30. Función para la inicialización del protocolo ModBus

```
void RTU_ModBus()
{
	unsigned long wdog = 0;		/* watchdog */

	if(mbs.update()){
		wdog = millis();

		if ((millis() - wdog) > 5000)  {	// no comms in 5 sec
			//regs[MB_CTRL] = 0;	// turn off led
		}
	}
}
```

Figura 31. Función para la gestión de la lectura/escritura de las tramas ModBus a través de puerto serie

Modbus usa un modelo de protocolo de tipo maestro-esclavo en el que el maestro es siempre quien inicia las comunicaciones mientras que el esclavo solo le responde. En el caso del programa en C++ se trata del esclavo y solo enviará los datos almacenados en el banco de registros cuando el dispositivo maestro se lo indique.

El modelado completo del protocolo de comunicación ModBus puede ser consultado en el documento

(https://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf)

publicado por la organización encargada de velar por las normas que definen el estándar del protocolo (www.modbus.org).

Modelado del protocolo basado en telemetría

A diferencia del protocolo Modbus este protocolo no mantiene la estructura maestro-esclavo, sino que mantiene un canal de comunicaciones bidireccional por el que ambas partes son capaces de enviar información en cualquier momento. La trama usada por el protocolo de comunicaciones Multiwii es la que se muestra en la Figura 32

$M>	N_bytes	OpCode	[data]	Checksum

Figura 32. Trama del protocolo Multiwii

Como puede observarse en la Figura 32, cualquier trama de este protocolo debe iniciarse con la cabecera *"$M>"*, a partir de donde se envía en número de bytes (N_bytes) que va a contener el mensaje, el código de operación (OpCode), o comando contenido en el mensaje. Habitualmente, este comando está relacionado con el sensor dispositivo con el que está relacionado el mensaje (Figura 34). A continuación se incluyen los datos (Data) que estén relacionados con el comando, tantos como sea necesario hasta un máximo de 64. Por último, para salvaguardar la integridad del mensaje se envía el checksum, generado, durante la creación del mensaje

```
#define MSP_IDENT     100       //out message    multitype + multiwii version +
                                                  protocol version + capability variable
#define MSP_STATUS    101       //out message    cycletime & errors_count & sensor
                                                  present & box activation & current setting
                                                  number
#define MSP_ENGRPM    102       //out message    Engine RPM
#define MSP_THROTTLE  103       //in message     Engine throtle
```

Figura 34. Ejemplos de OpCode usados en el protocolo Multiwii

```
/**
 * telemetry based on multiwii serial protocol
 */
class multiwii {
private:
	_serial_state		c_state = IDLE;

	uint8_t			checksum = 0;
	uint8_t			commandMW = 0;
	uint8_t			offset = 0;
	uint8_t			dataSize = 0;
	uint8_t			inBuf[INBUF_SIZE];

	TNotifyResults		notifyCmd = NULL;

public:
	//constructor
	multiwii(){};

	void	configure(long baud, char parity);
	void	read_data();
	void	setGestionCmd(TNotifyResults gCmd);

	static  void send_msp(uint8_t opcode, uint8_t * data,
uint8_t n_bytes);
```

Figura 33. clase para implementar el protocolo Multiwii

Para la gestión, tanto del envío, como de la recepción de datos se ha desarrollado una clase que contiene todas las variables y funciones necesaria para la correcta aplicación del protocolo (Figura 33). Esta clase, hace que el bucle principal de la aplicación sea extremadamente compacto. En la Figura 28 se puede observar este bucle, donde, en setup se inicializa BeTeBa para trabajar con MultiWii

```
//Inicia/Configura BeTeBa
beteba.activaThrottle();
beteba.setNotifyResults(telemetry->send_msp);
```

Figura 35. inicialización de BeTeBa para funcionar con MultiWii

Como MultiWii no tiene una configuración maestro-esclavo, tanto el dispositivo como el host puede iniciar la transmisión, en este sentido se ha actualizado la clase BeTeBa, para incluir un evento que sea llamado, cada vez que se produzca un cambio en alguno de los sensores. Este evento, como puede apreciarse en la Figura 35, envía los nuevos valores registrados. La función para enviar nuevos datos ha sido incluida en la

```
void multiwii::send_msp(uint8_t opcode, uint8_t * data, uint8_t n_bytes)
{

  uint8_t checksum = 0;

  Serial.write((byte *)"$M>", 3);
  Serial.write(n_bytes);
  checksum ^= n_bytes;

  Serial.write(opcode);
  checksum ^= opcode;

  for(int i = 0; i < n_bytes; i++){
        Serial.write(data[i]);
        checksum ^= data[i];
  }

  Serial.write(checksum);
}
```

Figura 36. Función para enviar nuevos datos usando MultiWii

Figura 36.

Del mismo modo, en la Figura 28, podemos ver que el bucle principal incluye la función para leer (Figura 37) a través del puerto serie, si se ha recibido algún dato desde el host que requiera atención.

```
void multiwii::read_data(){

        while (Serial.available()) {

                byte c = Serial.read();

                if (c_state == IDLE) {
                        c_state = (c=='$') ? HEADER_START : IDLE;
                }
                else if (c_state == HEADER_START) {
                        c_state = (c=='M') ? HEADER_M : IDLE;
                }
                else if (c_state == HEADER_M) {
                        c_state = (c=='>') ? HEADER_ARROW : IDLE;
                }
                else if (c_state == HEADER_ARROW) {

                        if (c > INBUF_SIZE) {  // now we are expecting the payload size
                                c_state = IDLE;
                        }
                        else {
                                dataSize = c;
                                offset = 0;
                                checksum = 0;
                                checksum ^= c;
                                c_state = HEADER_SIZE;
                        }
                }
                else if (c_state == HEADER_SIZE) {
                        commandMW = c;
                        checksum ^= c;
                        c_state = HEADER_CMD;
                }
                else if (c_state == HEADER_CMD && offset < dataSize) {
                        checksum ^= c;
                        inBuf[offset++] = c;
                }
                else if (c_state == HEADER_CMD && offset >= dataSize) {
                        if (checksum == c) {
                                if(notifyCmd != NULL){
                                        notifyCmd(commandMW, inBuf, dataSize);
                                }
                        }

                        c_state = IDLE;
                }
        }
}
```

Figura 37. Función para recibir datos usando MultiWii

myESC

Inicialmente, para la generación de los pulsos PWM que van hacia el ESC, con objeto de controlar la velocidad del motor, con objeto de usar librerías estándar, siempre que fuera posible, se decidió usar la librería de Arduino Servo que cuenta con las funciones para generar estos PWM. Esta librería, genera los pulsos del control del ESC mediante una interrupción provocada por el desbordamiento timer 1 integrado en el procesador ATmega328P. Por otra parte, la librería arduino que gestiona el puerto serie, para la sincronización en la transmisión de datos, deshabilita las interrupciones

durante este periodo. Como ya se ha indicado durante la memoria, el pulso que gestiona la velocidad angular del motor debe tener una anchura entre 1 ms y 2 ms para parado y máxima. Esto hace muy crítico la anchura del pulso para mantener constante la velocidad de giro del motor. En este sentido, usando el procedimiento anterior, se producían aceleraciones inesperadas del motor. Esto era debido a que la atención de la interrupción que manejaba la anchura del pulso se podía encontrar deshabilitada en ciertos momentos.

Para solucionar este problema se decidió prescindir de la librería Servo y crear una librería propia que crease los PWM usando los temporizadores del procesador, pero sin dar como salida una interrupción, usando un modo de operación distinto.

El dispositivo Arduino usado en el proyecto utiliza el procesador ATmega328p, cuyo diagrama de bloques puede apreciarse en la Figura 3. Este procesador cuenta con 3 temporizadores o TC por sus siglas en inglés Timer/Counter. Dos de ellos de 8 bits correspondientes a los TC-0 y TC-2 y uno de 16 bits correspondiente al TC-1. El temporizador TC-0 es usado en múltiples funciones que vienen ya programadas dentro de la librería Arduino como puede ser la función *"millis()"* que devuelve el número de milisegundos que han transcurrido desde que el dispositivo Arduino comenzase a ejecutar el programa actual. Para la generación de los PWM del ESC se decidió usar el temporizador de 16 bits, el TC-1, no usado por ninguna otra librería de las que integran el proyecto. El diagrama de bloques mostrado en la Figura 38 muestra la funcionalidad del temporizador de 16 bits. El funcionamiento de este temporizador, explicado de una forma sencilla es el siguiente. El temporizador posee un contador que funciona de acuerdo con los tics del procesador, aunque cuenta con un *"preescaler"* y que en el diagrama de bloques está representado como *"TCNTn".* Este valor que posee el contador se compara con los valores que hay en los registros *"OCRnA"* y *"OCRnB"* y cuando alguna de estas comparaciones se cumple es cuando el temporizador genera automáticamente una interrupción.

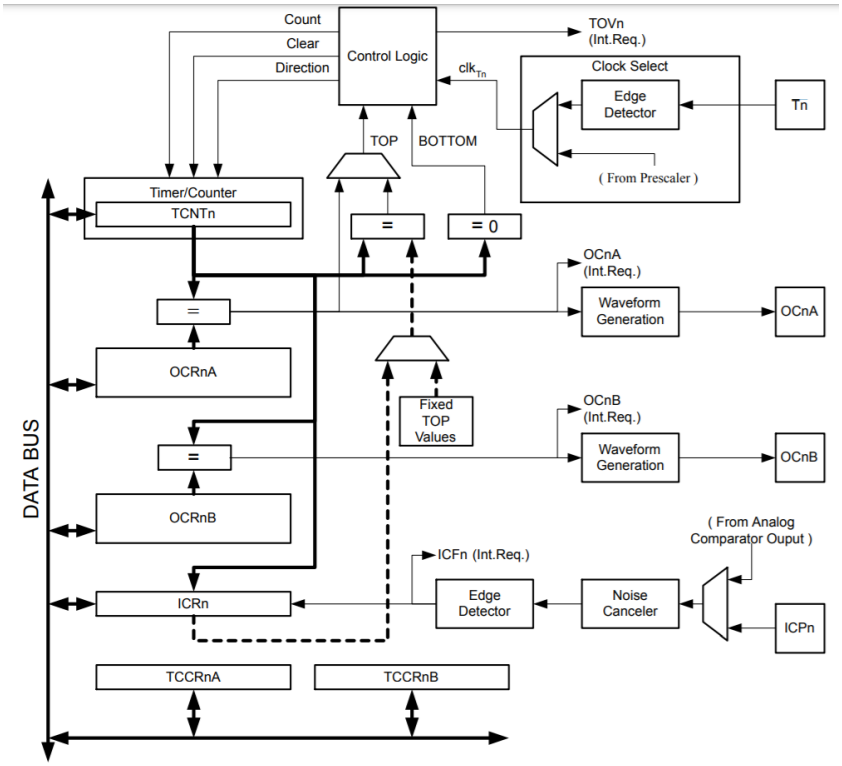

Figura 38. Diagrama de bloques del temporizador de 16 bits TC-1. Fuente [23]

Es aquí donde pasan a ser relevantes los diferentes modos de operación del temporizador y las salidas *"OCnA"* y *"OCnB"*. Cada uno de los temporizadores del procesador posee varios modos de operación, el modo normal, el modo CTC por sus siglas en inglés *Clear Timer on Compare Match*, el modo *Fast PWM, el modo corrector de fase* y muchos otros que no se mencionan. En este caso se hablará principalmente del modo Corrector de fase (Fase correct). Toda esta información puede ser ampliada en [23].

Los modos Phase Correct Pulse Width Modulation o Phase Correct PWM (WGM1[3:0]= 0x1, 0x2, 0x3, 0xA y 0xB) proporcionan una opción de generación de forma de onda PWM de alta resolución y fase correcta. En el modo corrector de fase, el contador cuenta repetidamente de BOTTOM (0x0000) a TOP y luego de TOP a BOTTOM. En el modo Comparar salida sin invertir, la comparación de salida (OC1x) se borra cuando hay una coincidencia entre TCNT1 y OCR1x mientras se cuenta hacia arriba, y se activa, se establece esta coincidencia cuando se cuenta hacia abajo. Al invertir el modo de comparación de salida, la operación se invierte. La operación de doble pendiente tiene una frecuencia de operación máxima más baja que la operación de una sola

pendiente. Sin embargo, debido a la simetría característica de los modos PWM de doble pendiente, estos modos son los preferidos para aplicaciones de control de motores.

Cuando se realiza un reset sobre el procesador ATmega328P con las librerías arduino, el estado inicial de los registros TCCR1A y TCCR1B son respectivamente 0x81 y 0x03, como se muestra en la Figura 40 y la Figura 39 respectivamente.

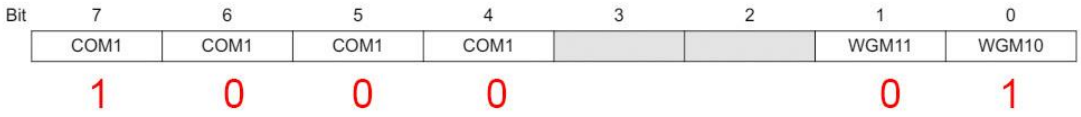

Figura 40. Contenido registro TCCR1A después del reset

Figura 39. Contenido del registro TCCR1B después del reset

Según estos valores, el preescalar del timer estará programado a 64:

CS12 CS11 CS10 --> 0 1 1

El modo de comparación de salida (Compare Output Mode) es:

COM1A1 COM1A0 --> 1 0

Según el cual, el bit OC1A se pone a cero en "Comprare Match", es decir en el momento en el que el contenido de los registros TCNT1 y OCR1A coinciden.

Por otra parte, el modo de operación del timer será el 1, es decir, el modo corrector de fase de 8 bits:

WSM13 WSM12 WSM11 WSM10 --> 0 0 0 1

Más información acerca de este modo de funcionamiento se puede encontrar en [23].

En estas condiciones, teniendo en cuenta que la frecuencia de reloj del procesador ATmega328P en el módulo arduino nano es de 16 MHZ y que el valor de preescalado del timer es de 64, encontramos que la frecuencia efectiva del reloj del timer será:

16 MHz/64 --> 250000 Hz

Y puesto que en el modo de operación "Phase Correct PWM", el contador cuenta de abajo hacia arriba y de arriba hacia abajo, la frecuencia del ciclo de generación, se verá reducida en 2

$$f_{ciclo} = 250000 / 2 = 125000 \text{ Hz}$$

por tanto, el periodo del ciclo será

P = 1/125000 = 8E-6 Sec.

Según el modo de operación "Phase Correct PWM", inicialmente, el valor del pin OCR1A es 1 (alta), el registro TCNT1 se incrementa desde cero, y cuando alcanza el valor del registro OCR1A, el pin OCR1A es puesto a cero. TCNT1 continúa incrementándose hasta que alcanza el valor 0xFF, momento en el que comienza a decrecer y cuando nuevamente alcanza el valor del registro OCR1A, el pin OCR1A es puesto de nuevo a uno hasta que el registro TCNT1 alcanza de otra vez el valor cero. A partir de ahí, el ciclo se repite. De aquí, podemos deducir que el tiempo (T), expresado en segundos, durante el que el pin OCR1A permanecerá a 1, vendrá dado por la expresión:

8E-6 x OCR1A = T Sec.

De donde se puede deducir el contenido con el que se debe cargar el registro OCR1A para un ancho de pulso deseado.

Para la gestión de todo este proceso se ha definido una clase myESC, mostrada en la Figura 42.

```
#ifndef MYESC_H_
#define MYESC_H_

#include "Arduino.h"


class myESC{
public:
	//Constructor
	myESC(){};

	void init();
	void writeMotors(uint16_t ms);
};

extern myESC ESC;


#endif /* MYESC_H_ */
```

Figura 42. Clase myESC para la gestión del pulso usado como acelerador del motor

donde la función que modifica el registro OCR1A que produce la señal de aceleración efectiva del motor viene dado por la Figura 41

```
/*************************************************************************/
/******  Writes the Motors values to the PWM compare register  *********/
/*************************************************************************/
void myESC::writeMotors(uint16_t ms) { // [1000;2000] => [125;250]
	OCR1A = ms >> 3; //  pin 9
}
```

Figura 41. Función para modificar el valor del ancho de pulso del acelerador

Como se está refiriendo al TC-1 se necesita buscar los pines a los que están asociadas las salidas OC1A y OC1B aunque para el proyecto solo se usará la salida OC1A. En el diagrama de la figura 22 se aprecia que el pin de entrada asociado a la salida OC1A es el pin 15.

Placa creada para el proyecto

Durante el transcurso del proyecto cada vez que se quería probar algún sensor era necesario era necesario ayudarse de una placa de pruebas o protoboard, para posteriormente conectarla a los pins correspondientes de Arduino. Todo este procedimiento se hacía tedioso, además de introducir en el sistema cierto grado de

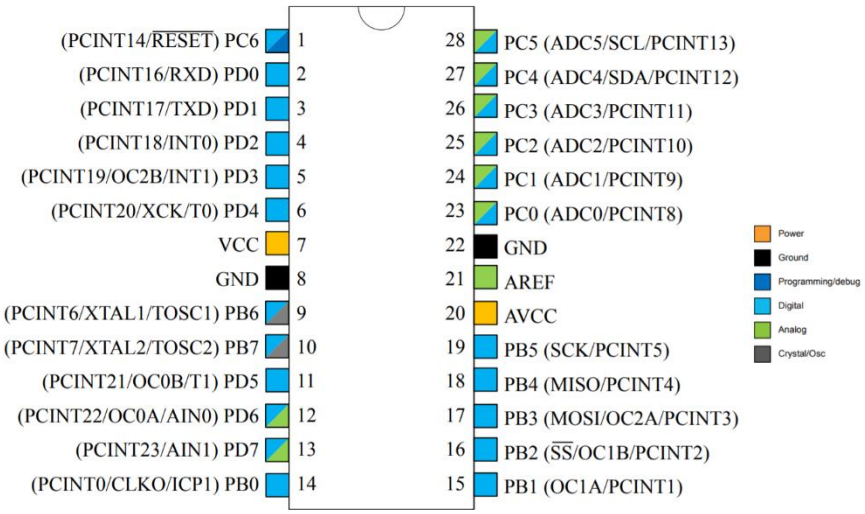

Figura 43. Pin.Out y sus tipos del procesador ATmega328p. Fuente [23]

inestabilidad, además un una fuente de ruido adicional. Con el fin de evitar todo este proceso de montaje, que además, requería de bastante tiempo, se creó en colaboración con el tutor del proyecto la siguiente placa en la que integrar todas la conexiones, junto con algunos de los circuitos usados durante el proyecto. El resultado final, es mostrado en la Figura 44

Figura 44. Placa para banco de pruebas brushless, anverso y reverso.

Esta placa incluye un zócalo para la conexión del núcleo del sistema, el módulo Arduino Nano en el que, como ya se ha indicado con anterioridad, va cargado el firmware de control. La placa cuenta ya con el circuito HX711 para la conexión de la célula de carga y con el filtro-amplificador para el acondicionamiento de la señal de intensidad intensidad. Además del chip HX711 y de los sensores de voltaje e intensidad, se incorpora los conectores para la conexión del control del ESC del motor y el sensor

de revoluciones, lo que aportará mayor estabilidad y fiabilidad a las conexionado del prototipode evitando así la protoboard y cables extras de conexiónado (Figura 45).

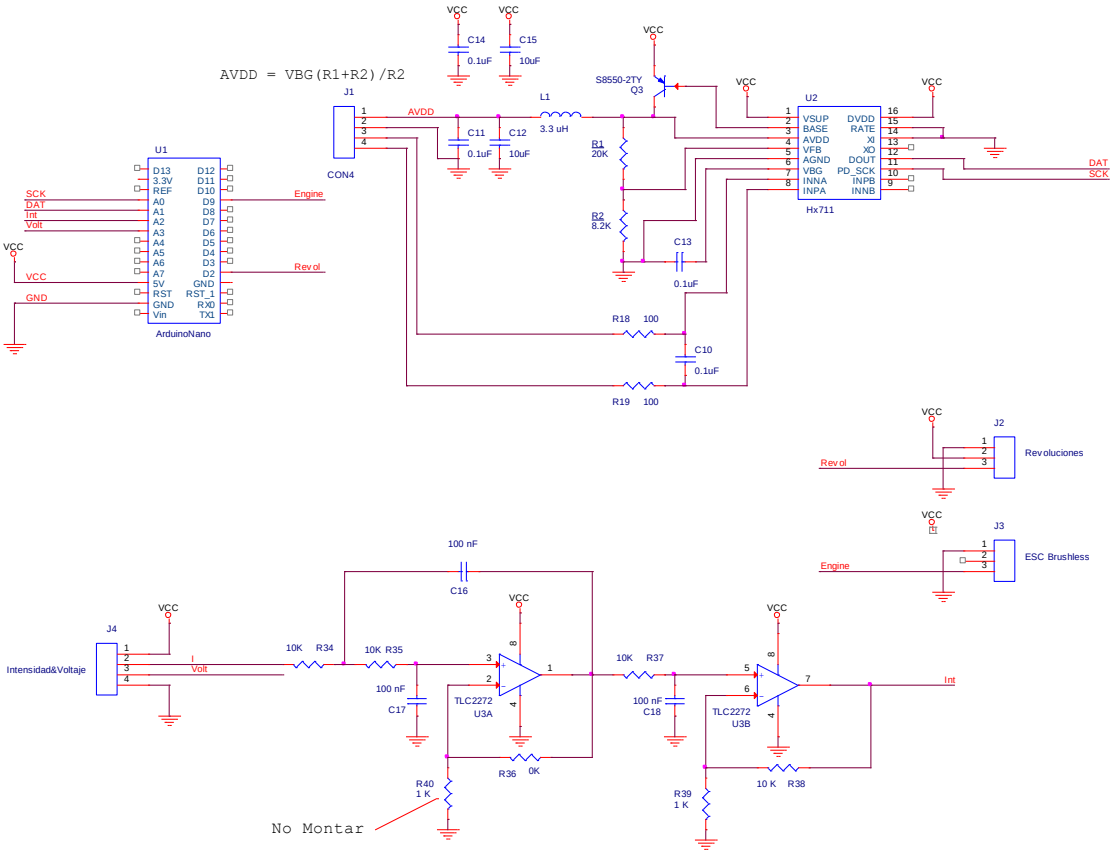

Figura 45. Circuito electrónico de la placa para banco de pruebas brushless.

3.4 Modelado del programa de gestión para la visualización de los datos recogidos por los sensores

En esta sección se mostrará en qué consiste exactamente la parte del proyecto realizada sobre Java, qué tecnología se ha usado y la evolución que ha seguido el programa.

Esta parte del proyecto realizada en Java, consiste en una interfaz gráfica de usuario que se comunica con el firmware de control cargado en la placa Arduino para obtener todos los datos de los sensores con el objetivo de mostrarlos al usuario gráficamente, además de permitir realizar cambios en los parámetros del motor. Esta

comunicación se hace a través del puerto serie y mediante un protocolo de comunicaciones. La aplicación se encarga de mantener una copia del banco de registros guardado en la placa Arduino y de actualizarla cada cierto tiempo o cuando realiza algún cambio.

Cuando se inicia la aplicación se abre una ventana donde aparecen todas las opciones. El primer paso es seleccionar el puerto serie por el que se realizarán las comunicaciones, ya que es necesario para que el programa pueda saber por dónde debe enviar y pedir los datos. Esto se hace desde el panel desplegable de comunicaciones (Figura 46), seleccionando en la pestaña "*Serie*" el puerto COM deseado.

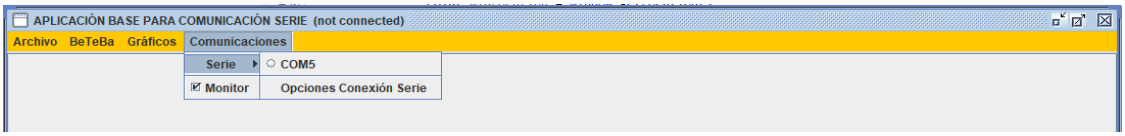

Figura 46. Captura de pantalla de la aplicación donde puede apreciarse los puertos series disponibles.

Una vez hecho esto, con el firmware de control corriendo sobre la placa Arduino y conectado al programa Java mediante el puerto serie se puede explorar las distintas ventanas que conforman la interfaz. Comenzando por la ventana "*Archivo*", (Figura 47) la cual solo posee un botón para cerrar la aplicación, aunque esta puede cerrarse manualmente.

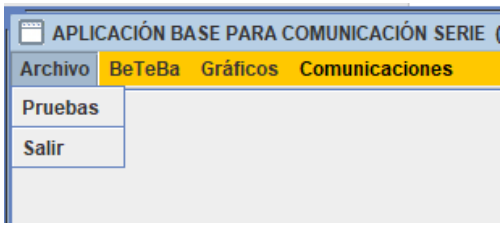

Figura 47. *Captura de pantalla de la aplicación donde se aprecia el botón "Salir". Fuente: Autor*

Luego se tiene la pestaña "BeTeBa" (Figura 48), encargada de mostrar todos los datos que llegan desde la placa Arduino en un formato de texto. La pestaña tiene en primer lugar un deslizador para controlar el acelerador del motor, luego tiene los valores de la fuerza de empuje actual del motor, el voltaje, la intensidad y el consumo de éste y por último tiene una rueda de medición para las revoluciones por minuto a las que está sometida el motor. El objetivo de este panel es que un experto pueda apreciar en tiempo

real los cambios que van ocurriendo en las distintas variables del motor de manera instantánea a medida que se cambie el valor del acelerador.

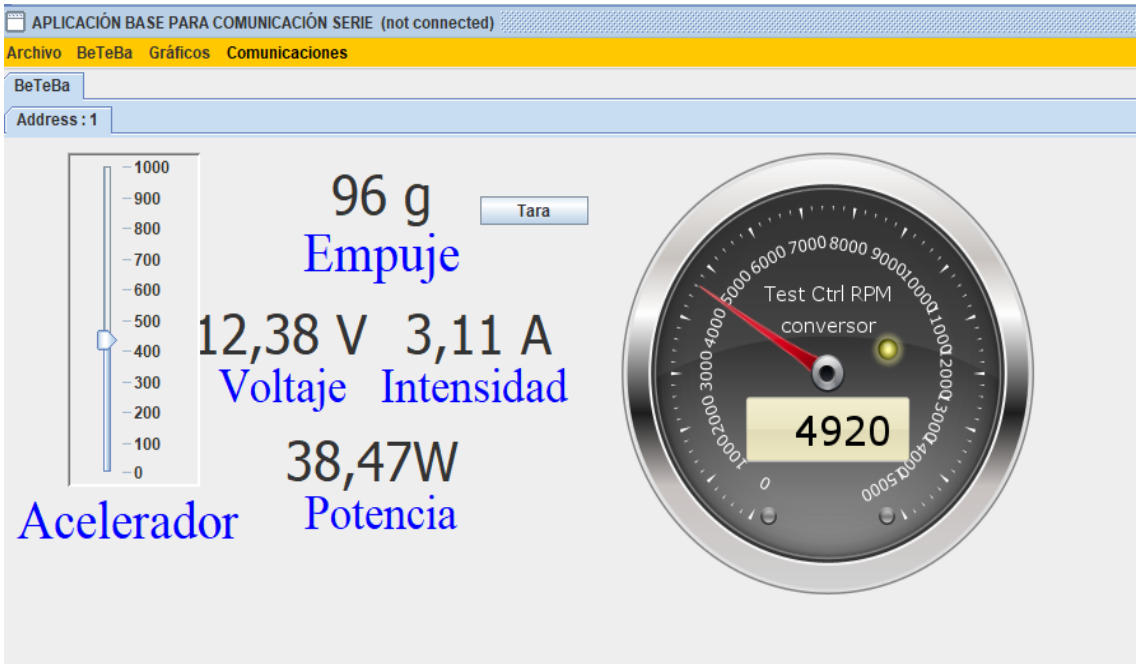

Figura 48. Captura de la pestaña BeTeBa donde se aprecian un resumen de los datos recogidos por la aplicación sobre el motor. Fuente: Autor

La siguiente pestaña, es la relacionada con el apartado gráfico, y posee dos paneles distintos: el primero dedicado a gráficos en tiempo real (Figura 50) y el segundo dedicado a pruebas específicas del motor (Figura 49). En el primer apartado, al igual que en la pestaña BeTeBa puede apreciarse una barra deslizante con la que se controla el acelerador del motor, sin embargo, a diferencia del caso anterior, se muestra una gráfica en la que pueden apreciarse los datos sobre el empuje, el consumo y las revoluciones por minuto de los últimos 30 datos recogidos por el sensor sobre el eje Y, mientras que la cantidad de datos viene representada por el eje X. Este panel está hecho para ver las distintas respuestas del motor a lo largo del tiempo y hacer comparaciones con valores anteriores y a medida que el valor del acelerador va cambiando.

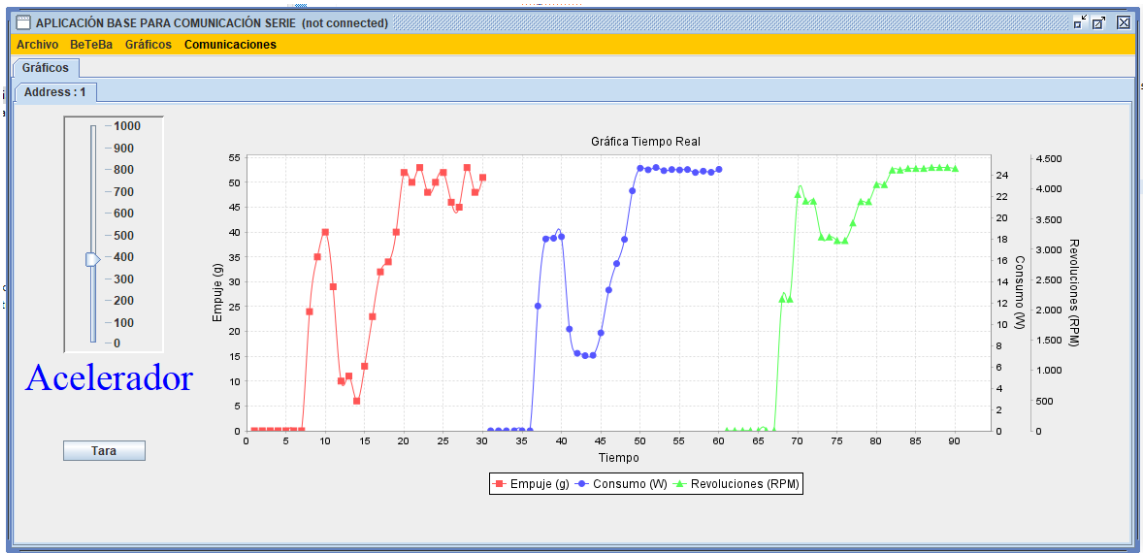

Figura 50. Captura de la pestaña de gráficos en tiempo real. Fuente: Autor

Por último, el panel del apartado gráfico dedicado a las pruebas específicas del motor. Aquí, a diferencia de los anteriores, no cuenta con una barra deslizante para el control del acelerador del motor. Por el contrario, se dispone de unos botones para calibración y ejecución de la prueba configurada. El botón de calibrar el motor se encarga de ver cuál es el ancho de pulso mínimo a partir del cual el motor comienza a dar una respuesta. Este ancho de pulso puede variar de un ESC a otro, por lo que se estima necesario la implementación de esta calibración. Una vez calibrado el motor el panel cuenta con una prueba, aunque como margen de mejora del proyecto, proponemos el desarrollo de nuevas pruebas que pongan al motor en diferentes

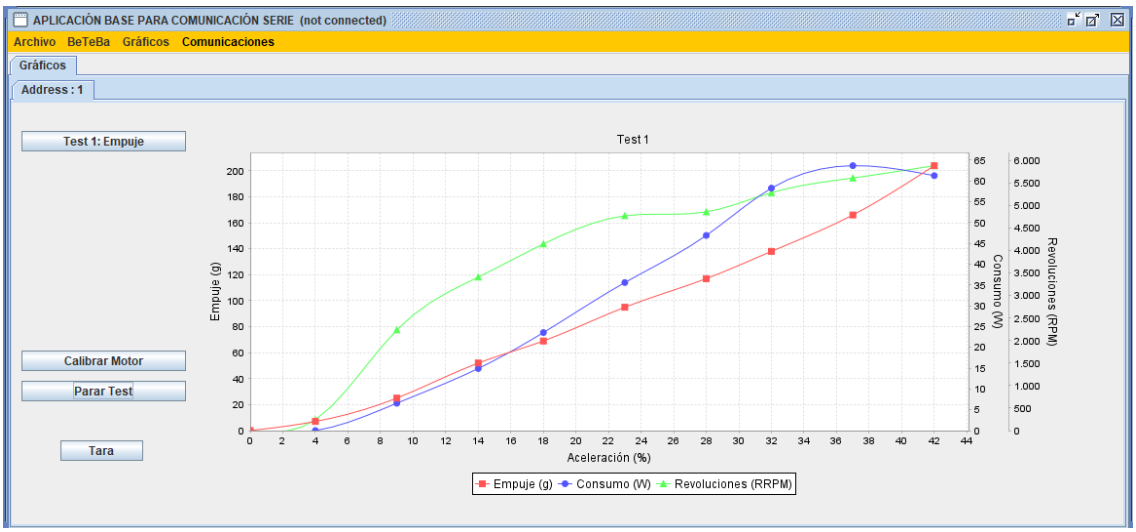

Figura 49. Captura de la pestaña de gráficos en pruebas. Fuente: Autor

situaciones. En este caso, al pulsar sobre el botón “Test 1: Empuje” el acelerador del motor comenzará a incrementarse de manera gradual, registrando también todos los datos de empuje, revoluciones y consumo hasta que el motor alcanza el 100% de su capacidad. En este momento, la prueba se detiene y muestra todos los datos recogidos en una gráfica.

Una vez vista la forma de uso del programa se explicará cómo está realizado el programa, las librerías usadas para cada uno de los paneles y cómo está implementado el algoritmo de comunicaciones sobre él.

La principal herramienta que se ha usado a la hora de desarrollar la interfaz ha sido "WindowBuilder". WindowBuilder es una herramienta de software que puede ser instalada fácilmente en eclipse. Se ha usado para construir las diferentes interfaces de cada uno de los menús explicados anteriormente, pero sin la necesidad de escribir grandes cantidades de código.

Se ha usado la librería *"SteelSeries"* para implementar el medidor radial usado para medir las revoluciones por minuto en la pestaña *"BeTeBa"*.

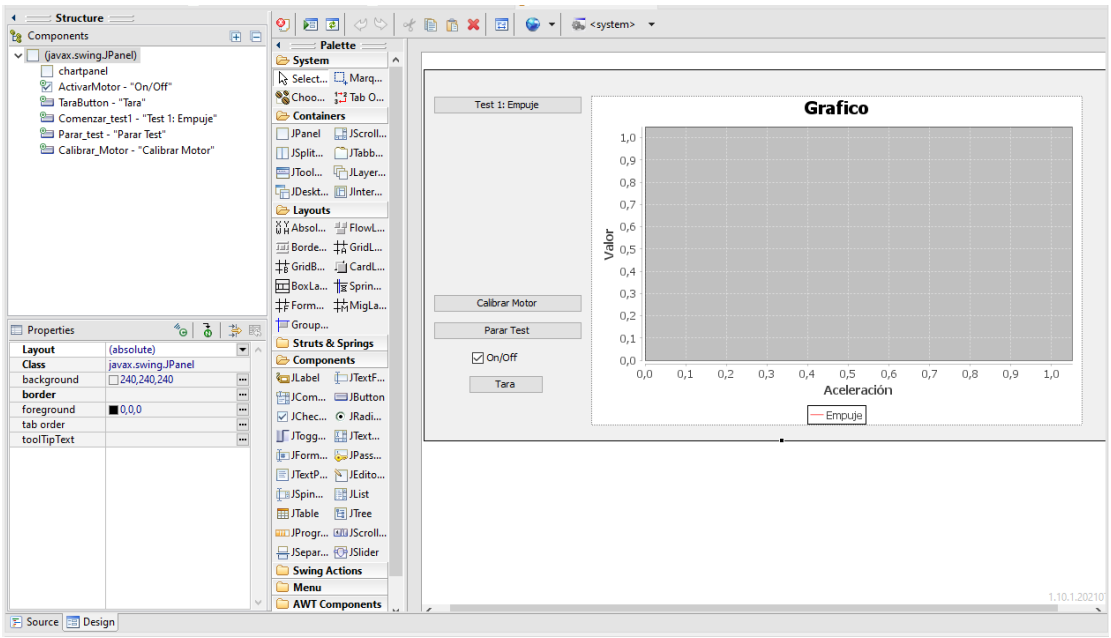

Figura 51. *Captura de pantalla con la herramienta "WindowBuilder" en funcionamiento. Fuente: Autor*

Para las comunicaciones en la aplicación Java usamos la librería "CommTransport", desarrollada e implementada durante los estudios de informática, que a su vez usa distintas librerías de comunicaciones, con objeto de generalizar el interface de comunicaciones independientemente del medio y la librería soporte. Concretamente, para este proyecto, se ha usado la librería soporte JSSC, por sus siglas en inglés de Java Simple Serial Conector que gestiona la comunicación a través del del puerto serie. Para la implementación del protocolo ModBus se ha usado la librería "myModBus" también desarrollada durante los estudios de informática. Y por último, para la implementación de las comunicaciones usando el protocolo de telemetría todo el proceso ha sido implementado usado el diagrama de clase mostrado en la Figura 56.

Las librerías *"JfreeChart"* y *"xChart"* fueron usadas para la creación de los distintos gráficos sobre sus respectivos paneles. También se ha usado la librería "JasperReport" para la generación de documentos pdf.

3.5 Protocolos de comunicaciones usados

3.5.1 Modbus

El protocolo principal sobre el que se pensó todo el proyecto desde un principio y sobre el que se ha ido desarrollando en su mayoría es Modbus, en concreto la versión Modbus RTU por lo que será el primero sobre el que se hablará.

El protocolo Modbus es un protocolo de comunicaciones basado en mensajes y en la arquitectura cliente/servidor o maestro/esclavo, en el caso de este proyecto quien está actuando como servidor o esclavo es la aplicación C++ corriendo en la placa Arduino mientras que como cliente o maestro la aplicación Java, que es quien envía las órdenes. Las razones por las que se eligió Modbus como protocolo de comunicaciones principal para el proyecto fueron, en primer lugar, que es un protocolo muy fácil de implementar para comunicaciones por puerto serie y no requiere apenas de desarrollo para funcionar correctamente, el protocolo es público y gratuito y se adapta muy bien a las necesidades

del proyecto ya que maneja muy bien los bloques de datos como los que se usan en este proyecto para guardar los datos de los sensores.

El funcionamiento del protocolo es el siguiente. Se tiene un equipo principal que será el encargado de enviar las órdenes, se llamará a este equipo de ahora en adelante maestro y se tienen por otra parte uno o varios equipos que se encargarán de recibir esas órdenes y actuar conforme a ellas, estos equipos se les conocerá como esclavos. Son los equipos esclavos los que están atentos permanentemente al canal de comunicaciones, en el caso de este proyecto el canal de comunicaciones es el puerto serie por el que se realizan las comunicaciones, pero podría tratarse de cualquier otro medio capaz de enviar datos. Es entonces el maestro quien se encarga de enviar las instrucciones a los esclavos que tras recibir la orden actuarán en consecuencia, tras esto los esclavos están obligados a enviar una respuesta al maestro con el objetivo de que éste sepa que se ha recibido la orden. En caso de que esta respuesta tarde demasiado en llegar o directamente no llegue el protocolo dicta que se lanzará una excepción en el maestro.

Todas estas comunicaciones deben realizarse utilizando un modelo de trama común definido por el protocolo. En primer lugar, el maestro que va a iniciar la comunicación por Modbus construye la unidad de datos de protocolo o como se conoce por sus siglas en inglés, PDU, y posteriormente añade el resto de los campos para la correcta formación de la trama. En la Figura 52 pueden apreciarse los diferentes campos y el orden de cada uno de ellos en la trama.

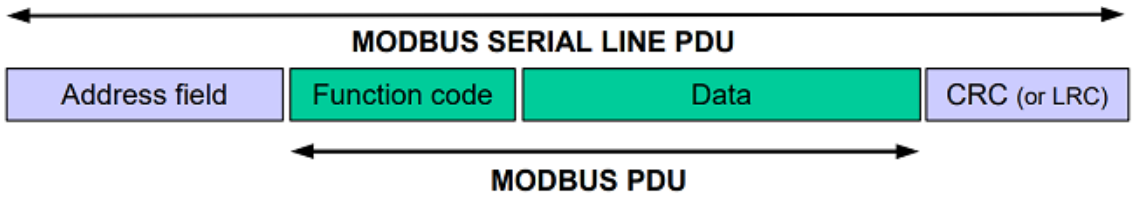

Figura 52. Estructura del datagrama en el protocolo Modbus. Fuente: [11]

El rango de direcciones válidas que pueden ser colocadas sobre el campo "Address field" va desde el 0 hasta el 247 en decimal. La dirección 0 está reservada para

el envío por difusión, es decir, a todos los equipos de la red al mismo tiempo por lo que cada equipo esclavo puede adoptar una dirección desde el 1 hasta el 247. Un equipo maestro puede enviar un mensaje a un equipo esclavo poniendo en el campo “Address field” la dirección de susodicho equipo esclavo. Una vez el equipo esclavo ha recibido este mensaje al enviar la respuesta éste pone su propia dirección para indicar al equipo maestro quién está enviándole esa respuesta.

El campo “Function code” indica al esclavo qué tipo de acción debe realizar y viene definido por la forma de programar estos esclavos. Este código de función puede o no ir acompañado por un campo de datos que puedan ser relevantes para la acción que vaya a tomar el equipo esclavo.

Por último, la trama va acompañada por un código de verificación de redundancia cíclica, o por sus siglas en inglés CRC, que es un código calculado a partir del contenido del mensaje y ue se usa para asegurar la integridad de éste. En caso de que el CRC no coincida con el calculado quiere decir que la integridad de la trama podría estar comprometida. [11][12]

Pueden apreciarse ahora los tamaños de cada campo en la Figura 53 donde puede apreciarse que el tamaño máximo para una trama Modbus es de 256 bytes.

Slave Address	Function Code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low | CRC Hi

Figura 53. Tamaños de cada campo de datos de un datagrama perteneciente al protocolo Modbus. Fuente: [11]

Cada trama debe ir separada de la siguiente por un tiempo en el que no se producen comunicaciones de al menos 3,5 veces el tiempo que se tarda en enviar un carácter. Si entre paquete y paquete no ha pasado ese tiempo ambos paquetes son descartados y declarados incompletos ya que este tiempo es la forma de declarar que

un paquete ha terminado de enviar datos. Todo el paquete debe ser retransmitido en una cadena continua de caracteres. Cada byte está representado por dos caracteres en hexadecimal, si entre cualquiera de los caracteres de un mismo paquete pasan al menos 1,5 veces el tiempo que se tarda en enviar un carácter, este paquete también será marcado como incompleto. Pueden apreciarse ejemplos sobre estos dos casos en la Figura 55 y Figura 54.

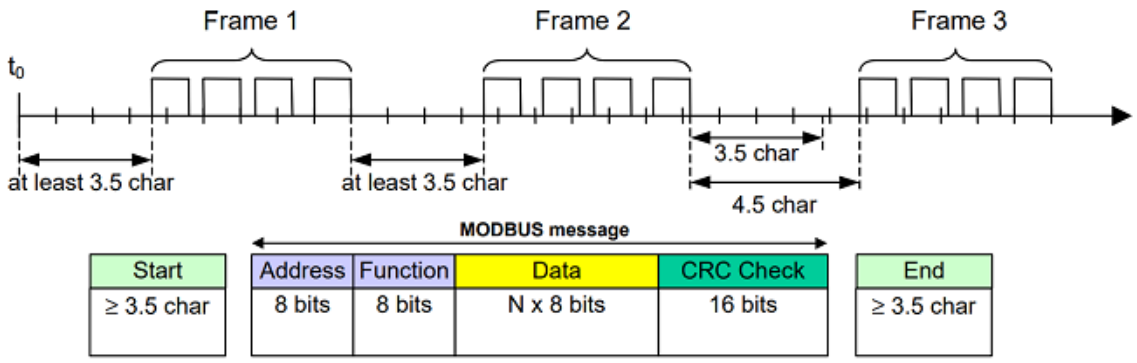

Figura 55. Tiempos de envío necesarios antes y después de un datagrama perteneciente al protocolo Modbus. Fuente: [11]

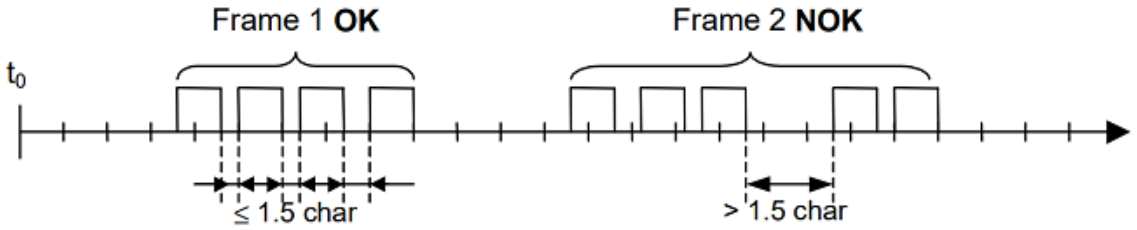

Figura 54. Ejemplos de un datagrama enviado correctamente y otro no válido pertenecientes al protocolo Modbus. Fuente: [11]

3.5.2 Protocolo basado en Telemetría

Durante el desarrollo del proyecto se fue barajando la posibilidad de utilizar otro protocolo de comunicaciones más orientado al hecho de solo estar usando un único banco de pruebas, eliminando así la posibilidad de trabajar con varios bancos de pruebas al mismo tiempo. De esta forma se estaría trabajando con un protocolo distinto al protocolo Modbus, aunque esto no tendría relevancia para el desarrollo del proyecto. De aquí surgió la idea de usar otro protocolo de comunicaciones basado en telemetría el cual, además, podría ser muy útil a nivel formativo. La trama de este protocolo ya se

mostró en la Figura 32. Además, es muy simple y no requiere de confirmación de mensaje recibido. Cada uno de los dispositivos se limitan a enviar una trama como un mensaje, que puede ser un comando o una información. El diagrama de clase de la implementación en java se muestra en la Figura 56.

El uso de este protocolo tiene ciertas ventajas y desventajas frente al protocolo Modbus que se explicarán a continuación.

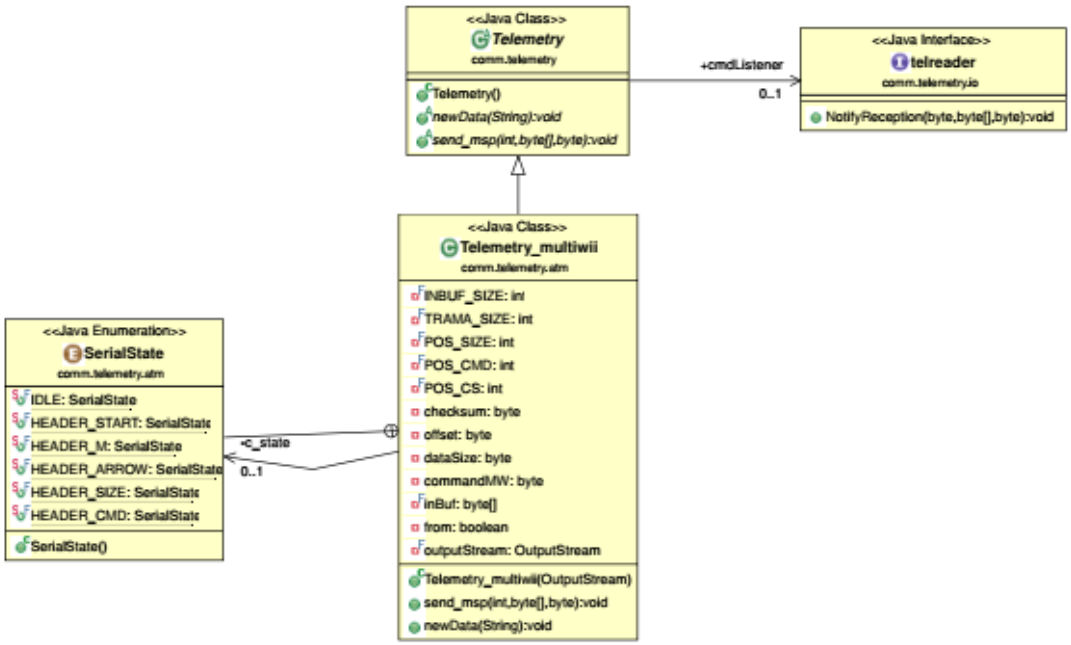

Figura 56. Diagrama de clase para la implementación del protocolo Multiwwi

Ventajas:

- Mayor facilidad a la hora de programar el protocolo. Su funcionamiento es bastante más sencillo que en Modbus.

- No requiere de un banco de registros para guardar la información por lo que el volumen de datos que ocupa en ambas partes es menor en el protocolo basado en telemetría.

Desventajas:

- No posee control de errores. En caso de que algún paquete se pierda de alguna forma o que la comunicación falle no existe ninguna forma de notificar al emisor de que su paquete no ha llegado ya que una vez enviado no se espera ninguna respuesta por parte del receptor.

- Mayor tráfico de paquetes. Por cada cambio detectado debe enviarse un paquete notificando de este cambio, a diferencia de antes que podían actualizarse todos los registros mediante una sola instrucción.

- No permite más equipos dentro de la red. A diferencia de Modbus que contempla la posibilidad de tener más equipos esclavos este protocolo basado en telemetría está orientado únicamente al intercambio de datos entre dos equipos.

3.6 Evaluación e interpretación final

En este apartado se expondrán ejemplos sobre las distintas conclusiones que pueden llegar a obtenerse a partir de la ejecución del proyecto sobre distintas situaciones.

En primer lugar, serviría para comprobar el correcto funcionamiento de un motor, es decir, si realmente el motor se comporta como debería. Esto incluye que las revoluciones vayan aumentando correctamente a medida que el valor del acelerador sube o comprobar que el motor realmente está consumiendo la cantidad de energía que debería.

También pueden medirse el impacto que tienen ciertos tipos distintos de hélices sobre el empuje que proporciona el motor o medir qué valor del acelerador se necesita para dar cierta cantidad de empuje. Esto por ejemplo puede ser muy útil en el ámbito

de las pruebas con drones, para saber la potencia necesaria para levantar su propio peso y cuánto peso podría llegar a levantar como máximo.

Conclusiones del proyecto

Durante el transcurso de este proyecto se han alcanzado la mayoría de los objetivos inicialmente propuestos:

- Creación de un programa para Arduino que permita controlar y monitorizar todos los sensores que se colocasen y que además fuese ampliable a cuantos sensores se quisieran. Esta aplicación tampoco está limitada a un solo tipo de motor y controlador sino que pueden usarse cualquiera siempre estos sean modulados mediante un ancho de pulso.

- Creación de una interfaz cómoda de usuario, que no fuese difícil de operar por cualquier persona y que implementase diversas maneras de mostrar todos los datos recogidos por los sensores además de ofrecer diversas opciones para realizar sobre el control del motor.

- Implementación de un protocolo de comunicaciones (Multiwii). Además del protocolo de comunicaciones Modbus, para este proyecto se ha Implementación el protocolo basado en telemetría explicado anteriormente en la memoria.

- En definitiva se ha completado la creación completa de una plataforma de pruebas para motores brushless de forma completamente operativa.

- Se ha hecho uso de los conocimientos adquiridos durante la carrera y se ha implementado un sistema donde se han usado varios lenguajes de programación además de una forma para su interconexión. Trabajando tanto a alto nivel con el programa en Java como a un nivel más cercano de la máquina como ha sido el caso del programa en C++, en el que se ha tenido que trabajar directamente con los registros del procesador.

Líneas futuras del proyecto

- Adaptación de la aplicación para trabajar con múltiples motores al mismo tiempo. El uso del protocolo Modbus permite que en un futuro puedan operarse al mismo tiempo diversos dispositivos esclavos, es decir, diversos bancos con sus respectivos sensores y motores lo que puede ser muy útil a la hora de probar el comportamiento de todos los motores de alguna aeronave al mismo tiempo.

- Implementación de otros tipos de pruebas con otras características que puedan resultar útiles para poner según las prescripciones del usuario.

- Ampliación y conexión de la aplicación con bases de datos para la gestión de resultados y accesos a distintos motores y hélices.

Referencias

[1] N. Tesla, *Method of and apparatus for controlling mechanism of moving vessels or vehicles*. Accesible en la Oficina de Patentes y Marcas de Estados Unidos por su número de patente: 613809

[2] *Historia de los drones ElDrone*. URL: http://eldrone.es/historia-de-los-drones/

[3] *Archibald Low*, Wikipedia - La Enciclopedia Libre. URL: https://en.wikipedia.org/wiki/Archibald_Low

[4] *Hewitt-Sperry Automatic Airplane*, Wikipedia - The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/Hewitt-Sperry_Automatic_Airplane

[5] *La Administración Federal de Aviación estadounidense*. URL: https://www.faa.gov/uas/resources/by_the_numbers/

[6] Jason Kelly, *¿Cuál es la manera más eficaz de conmutar un motor de CC sin escobillas?*, 2017. URL: https://www.digikey.es/es/articles/what-is-the-most-effective-way-to-commutate-a-bldc-motor

[7] Sergio Andrés Reyes Sierra, *Control híbrido de motores DC sin escobillas usando FPGA*, 2013. Repositorio: https://inaoe.repositorioinstitucional.mx/jspui/bitstream/1009/241/1/ReyesSSA.pdf

[8] *Galga Extensiométrica*, , Wikipedia - La Enciclopedia Libre. URL: https://es.wikipedia.org/wiki/Galga_extensiom%C3%A9trica

[9] *El puente de Wheatstone Aspectos básicos y teoría de funcionamiento*. URL: https://www.hbm.com/es/7163/el-puente-de-wheatstone-galgas-extensometricas/

[10] Texas Instruments, *INA1x9 High-Side Measurement Current Shunt Monitor datasheet*, 2000 - revised february 2017.

[11] Ozeki Ltd, Modbus RTU. URL: https://ozeki.hu/p_5854-modbus-rtu.html

[12] *The Modbus Organization*. URL: http://www.modbus.org/

[13] *C++*, Wikipedia - La Enciclopedia Libre. URL: https://es.wikipedia.org/wiki/C%2B%2B

[14] Java Oracle, *¿Qué es la tecnología Java?* URL: https://www.java.com/es/download/help/whatis_java.html

[15] *Arduino Eclipse IDE Sloeber*. URL: https://eclipse.baeyens.it/

[16] *Arduino library to interface the Avia Semiconductor HX711 24-Bit Analog-to-Digital Converter (ADC) for Weight Scales.* URL: *https*://github.com/bogde/HX711

[17] Jaime Eduardo Jimbo Tacuri, *Caracterización del funcionamiento de un motor eléctrico de corriente continua sin escobillas brushles con 1000 watts de potencia, 2015*

[18] P.Yedemale y M.T. Inc, *Brushless DC (BLDC) Motor Fundamentals*, 2003

[19] Vaclav Mach, Stanislav Kovav, Jan Valouch y Mgrk. Milan Adámek, *Brushless DC Motor Control on Arduino Platform*, 2018

[20] *RPM Sensor For High-Voltage ESC*. URL: https://www.hobbywingdirect.com/products/rpm-sensor

[21] *UG LAND INDIA ESC (electronic speed controller) ESC 30 amp 30A Brushless Motor Speed Controller RC BEC ESC*. URL: https://www.amazon.in/UG-LAND-INDIA-electronic-controller/dp/B07VG27225

[22] Avia Semiconductor, *24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales datasheet*. URL: https://www.mouser.com/datasheet/2/813/hx711_english-1022875.pdf

[23] Atmel Corporation, *ATmega328P 8-bit AVR Microcontroller with 32K Bytes In.System Programmable Flash Datasheet, 2015.*

[24] Renesas, *What are Brushless DC Motors*. URL: https://www.renesas.com/us/en/support/engineer-school/brushless-dc-motor-01-overview

[25] Garcia Casado, FJ. Galgas Extensiométricas: Funcionamiento y Circuitos de Medida, 2010. URL: http://hdl.handle.net/10251/7747

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA