UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE VELOCIDAD PARA UN MOTOR DE CONTINUA HANPOSE 755 BASADO EN MICROCONTROLADOR

AUTOR: ALBERTO MONFORTE GÓMEZ

TUTOR: ALBERTO PÉREZ JIMÉNEZ

Curso Académico: 2020-21

RESUMEN:

Este trabajo consiste en el diseño e implementación de un sistema de control de velocidad para un motor de corriente continua Hanpose 755 basado en un microcontrolador. Hemos utilizado un Arduino Mega 2650 para llevar a cabo este proyecto, en el cual hemos implementado una señal de control del tipo PID para regular la velocidad del motor.

Se ha realizado un montaje con todos los elementos necesarios para el correcto funcionamiento de nuestro control de velocidad. Uno de nuestros objetivos ha sido optimizar al máximo la utilización de material disponible siempre y cuando el sistema mantuviera un funcionamiento adecuado.

Para realizar correctamente el trabajo hemos tenido que poner en práctica múltiples habilidades adquiridas en varias asignaturas durante el grado, como, informática industrial, tecnología automática, tecnología electrónica, circuitos eléctricos, ingeniería gráfica y proyectos. Además, hemos aprendido como funciona Arduino, no solo en su incorporación en el montaje si no todo su lenguaje de programación y funcionamiento de su entorno de desarrollo.

Palabras clave: Motor DC, Arduino, PID, Control, Encoder, Driver L298n.

RESUM:

L'objectiu d'aquest projecte consisteix a dissenyar i implementar un sistema de control de velocitat per a un motor de corrent continu Hanpose 755 basat en un microcontrolador. Hem utilitzat un Arduino Mega 2650 per a dur a terme aquest projecte, en el qual hem implementat un senyal de control del tipus PID per a regular la velocitat del motor.

S'ha realitzat un muntatge amb tots els elements necessaris per al correcte funcionament del nostre control de velocitat. Un dels nostres objectius ha sigut optimitzar al màxim la utilització de material disponible sempre que el sistema mantinguera un funcionament adequat.

Per a realitzar correctament el treball hem hagut de posar en pràctica múltiples habilitats adquirides en diverses assignatures durant el grau, com, informàtica industrial, tecnologia automàtica, tecnologia electrònica, circuits elèctrics, enginyeria gràfica i projectes. A més, hem aprés com funciona Arduino, no sols en la seua incorporació en el muntatge si no tot el seu llenguatge de programació i funcionament del seu entorn de desenvolupament.

Paraules clau: Motor DC, Arduino, PID, Control, Encoder, Driver L298n.

ABSTRACT:

This work consists of the design and implementation of a speed control system for a Hanpose 755 DC motor based on a microcontroller. We have used an Arduino Mega 2650 to carry out this project, in which we have implemented a PID type control signal to regulate the speed of the motor.

We have made an assembly with all the necessary elements for the correct operation of our speed control. One of our objectives has been to optimize to the maximum the use of available material as long as the system maintained a proper operation.

To perform the work correctly we have had to put into practice multiple skills acquired in various subjects during the degree, such as industrial informatics, automatic technology, electronic technology, electrical circuits, graphic engineering and projects. In addition, we have learned how Arduino works, not only in its incorporation in the assembly but all its programming language and operation of its development environment.

Key words: Motor DC, Arduino, PID, Control, Encoder, Driver L298n.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

1. INTRODUCCION

1.1. Motivación

La elección de este tema en torno a la programación está basada en motivos personales ya que no solo se pretende poner en práctica los conocimientos adquiridos a lo largo de nuestros estudios sino también profundizar en aspectos prácticos relacionados con la programación, el montaje de circuitos y control automático. Con relación a esto último, para este trabajo se va a utilizar un microcontrolador Arduino Mega 2560 que vamos a aprender a usar desde cero ya que este no se ha abordado en ninguna de las asignaturas del grado. Consideramos de gran utilidad para un alumno de Ingeniería Industrial con inquietudes en los automatismos el hecho de trabajar y familiarizarse con un microcontrolador, puesto que sin lugar a duda se va a enfrentar a estos cuando comience su vida laboral. Así pues, este TFG se plantea principalmente desde un punto de vista práctico tratando con elementos tangibles, aunque desarrollando también una parte teórica que fundamente todo lo presentado a lo largo de todo el proyecto.

1.2. Objetivos

El objeto principal de este trabajo consistiría en diseñar e implementar un sistema de control de velocidad de un motor de corriente continua utilizando un microcontrolador Arduino, poder visualizar el consumo de este y modificar la velocidad en tiempo real. Todo esto intentando optimizar al máximo posible tanto el código creado, como el uso de materiales disponibles.

Como objetivos secundarios nos proponemos poner en práctica los máximos conocimientos posibles adquiridos durante la carrera usando todas las herramientas que nos facilita la universidad politécnica de Valencia, ya sea a nivel de software o de documentación escrita. También queremos adquirir la máxima experiencia posible con la realización de este trabajo situándonos en un contexto real, como si estuviéramos realizando un proyecto profesional.

1.3. Estructura

La estructura que va a seguir nuestro trabajo es la que siguiente:

- Introducción. En esta parte se realiza una breve exposición del porqué se ha decidido realizar este trabajo. A continuación, se plantean los objetivos y seguidamente se explica la estructura del trabajo.
- Estado del arte. Se procede a mostrar los inicios de los controladores PID, así como su utilización en motores de corriente continua. Por otro lado, también se trata la creación del Arduino y sus diferentes usos actuales.
- Análisis del problema. Presentamos la problemática con la que nos enfrentamos en este trabajo y buscamos una solución óptima a esta.
- Diseño e investigación. Hablaremos tanto del software como del hardware utilizado y estudiaremos en detalle el funcionamiento de cada parte implicada.
- Desarrollo e implementación de la solución. La parte práctica en la que expondremos el montaje paso por paso y veremos de forma detallada el funcionamiento del mismo.
- Pruebas. Veremos los diferentes montajes y etapas que hemos necesitado para llegar al funcionamiento correcto.
- Conclusiones. Puesta en común del trabajo y resultado final.
- Presupuesto. Realizar una estimación del coste de este proyecto.
- Referencias. Aquí encontraremos todas las fuentes de las cuales hemos sacado información y gracias a las cuales hemos podido realizar este trabajo.

2. ESTADO DEL ARTE

2.1. Historia del PID

El avance exponencial de la tecnología en el siglo XX derivó en la creación del PID para poder regular y controlar sus acciones. Principalmente se podría decir que empezaron como simples limitadores de velocidad hasta 1911 donde Elmer Sperry creó el PID tal y como lo conocemos con sus tres acciones, proporcional, integral y derivado. El objetivo de Sperry era el control de dirección de buques de la armada estadounidense. No obstante, Sperry no llegó a profundizar en el análisis teórico y, por lo tanto, fue Nicolas Minorsky en 1922 quien público el primer análisis teórico de este control, describiendo su comportamiento con una ecuación matemática la cual sigue siendo hoy en día la base de nuestros cálculos.

Siendo la ecuación tal que:

$$señal\ de\ control = K_p(e(t) + \frac{1}{T_i}\int_0^t e(\tau)\,d\tau + T_d\frac{de(t)}{dt}) \qquad \text{(Ecuación 1)}$$

No vamos a entrar en el análisis detallado de cada parte de la ecuación en este apartado, ya que más adelante explicaremos en profundidad el funcionamiento y realizaremos los cálculos necesarios.

Hoy en día, el control por PID está presente en prácticamente todos los procesos industriales y nos es de gran utilidad. Uno de sus puntos fuertes es la simplicidad que requiere su montaje, solo se necesita disponer de un sensor que determine el estado del sistema, un actuador que modifique el sistema de manera controlada y un controlador que suministre la señal al controlador. El ejemplo más cercano a nuestro trabajo es el control de velocidad en la automoción, lo que hoy en día se denomina velocidad de crucero. Esta consiste en fijarle una velocidad a tu vehículo y que este se encargue de mantenerla de manera autónoma. Esto se puede ver claramente cuando conduces un vehículo y activas la velocidad de crucero, viendo que cuando llegas a una cuesta el coche acelera y por el contrario al llegar a una rampa el coche frena. De esta manera, mantiene siempre la velocidad que se le ha fijado. Como nuestro trabajo consiste en el control de un motor eléctrico podemos hacer referencia a los coches

eléctricos ya que estos se están haciendo cada vez más hueco en el mercado actual, abanderados sobre todo por la famosa marca Tesla. También, los podemos encontrar de la mano de procesos de carácter más puramente industriales como en un torno o una fresadora, los cuales te permiten modificar su velocidad de rotación, las revoluciones por minuto, de manera exacta e inmediata. [1]

2.2. Historia de Arduino

Arduino es una plataforma de hardware libre basada en un microcontrolador pensada para realizar control de procesos de forma sencilla y muy barata. Este tiene un entorno de desarrollo (IDE) libre y lenguaje de programación propio y simplificado.

El Arduino nació en 2005 en la universidad de Ivrea, Italia. Este fue llevado a cabo por unos estudiantes del Instituto de diseño Interactivo de Ivrea. La necesidad de crear un microcontrolador como Arduino surgió porque ellos estaban utilizando principalmente un microcontrolador BASIC Stamp el cual tenía un coste elevado. Esto los llevó a diseñar un microcontrolador de acceso libre y barato que pudiera usar cualquier persona que quisiera, o bien iniciarse en el mundo de la programación y control, o bien que ya tuviera experiencia y que no necesitara gastar una gran suma de dinero en un controlador ya que podría resultar sobredimensionado en su proyecto [2].

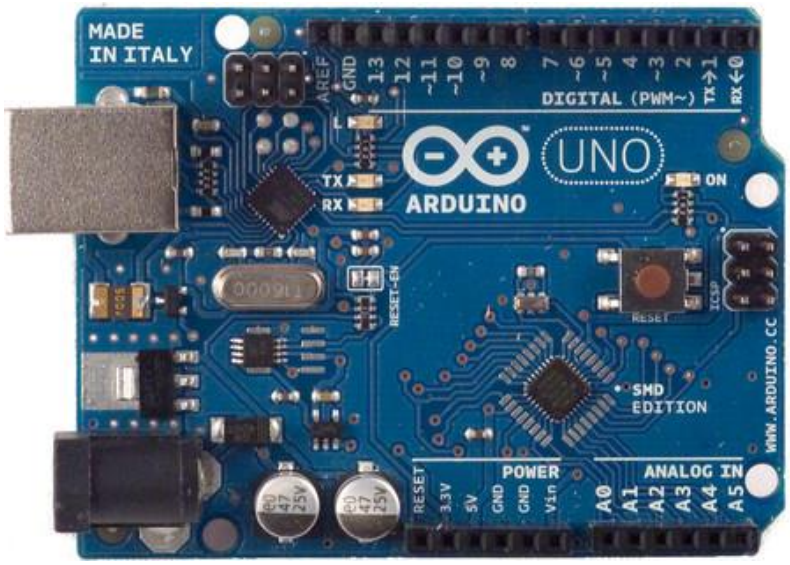


Figura 1 Arduino Uno
Fuente: [3]

En la figura 1 podemos observar un Ardunio modelo UNO. Este es el modelo más básico de la línea comercial. Es interesante dedicarle un momento a ver las diferentes conexiones que tiene el microcontrolador para recalcar la facilidad de uso. Se comunica con un ordenador vía USB, tiene posibilidad de alimentación externa con un conector EIAJ y tiene todos los pines de entradas y salidas disponibles para introducir un cable directamente. Posteriormente, hablaremos del modelo de Arduino utilizado en el trabajo y profundizaremos en la explicación de cada parte del mismo.

Hoy en día el uso de Arduino está muy extendido y podemos encontrarlo en cantidad de dispositivos. Vamos a comentar brevemente un par de elementos controlados por este que hoy en día son muy utilizados y todo el mundo conoce. En primer lugar, podemos encontrar un buen número de controladores MIDI que llevan integrado este microcontrolador, el MIDI se utiliza para conectar múltiples instrumentos y para la música electrónica, el cual hoy en día se ha apoderado por completo de la industria musical. En segundo lugar, tenemos impresoras 3D que cuentan con la tecnología Arduino para su control. Es evidente que gran parte del sector industrial tiene el punto de mira en estas impresoras y siguen muy de cerca su evolución por su indiscutible utilidad. En tercer y último lugar, Arduino también es muy utilizado en máquinas de control numérico por computadora (CNC), pese a ser este el caso más típico no podemos obviar la importancia de estos elementos de control en el sector Industrial durante estos tiempos.

3. ANALISIS DEL PROBLEMA

3.1. Presentación de la problemática

Nuestro objetivo principal es poder regular las revoluciones por minuto de nuestro motor de corriente continua. Por lo tanto, vamos a exponer todos los problemas que podemos encontrar a simple vista con el fin de poder ejecutar una solución de manera rápida y eficiente. Consideramos extremadamente importante dedicarle tiempo extra a reflexionar sobre los problemas iniciales de nuestro proyecto, ya que comenzar a trabajar sobre una base sólida nos va a dar seguridad a la hora de resolver contratiempos que surgirán durante la elaboración. Señalamos esto último, ya que sabemos con certeza que por muy bien que planeemos la resolución de nuestra problemática, vamos a tener que enfrentarnos a múltiples problemas a nivel de programación y montaje. Dicho esto, damos paso a los problemas principales sobre los que vamos a construir la base de nuestra solución.

Partiendo de que tenemos un motor DC de 12 voltios, ¿qué problemas nos encontramos para controlarlo?

En primer lugar, el obstáculo más evidente es que no tenemos manera de contar las revoluciones por minuto del motor y por lo tanto nos será imposible de controlar si no conseguimos de alguna forma poder medir esa variable.

En segundo lugar, suponiendo que tuviéramos las revoluciones controladas necesitamos de alguna manera regular la corriente que le suministramos al motor para poder modificar su velocidad de rotación. Algún elemento que por así decirlo haga de válvula o de grifo de los 12 voltios a los que se conecta el motor.

En tercer lugar, consideramos que poder controlar el consumo o la corriente demandada por el motor nos puede ser de extrema utilidad a la hora de proteger nuestro equipo y poder detenerlo antes de recibir daños permanentes.

En cuarto lugar, pensando en el usuario, vemos necesario una pantalla de adquisición de datos en la cual poder mostrar las variables sobre las que trabajamos para que pueda estar al corriente del funcionamiento del sistema.

Por último y quinto lugar, necesitamos un dispositivo que pueda reunir toda la información recopilada en los puntos anteriores y gestionarla correctamente, de manera que pueda dar las ordenes pertinentes a cada dispositivo y controlar el funcionamiento de cada uno de ellos de forma concreta y precisa.

A continuación, vamos a enumerar en forma de listado los principales obstáculos que tenemos que superar con el fin de controlar exitosamente nuestro motor y que hemos mencionado anteriormente:

- Conteo de RPM
- Control de tensión
- Corriente demandada
- Pantalla de adquisición de datos
- Dispositivo controlador

3.2. Soluciones propuestas

Una vez expuestos todos los obstáculos que debemos superar para iniciar el proyecto, vamos a proceder a buscar soluciones para cada uno de ellos. Cabe destacar que en este apartado vamos a buscar la solución general. Hablaremos de los elementos que vamos a necesitar para cumplir con todas las necesidades expuestas, pero no especificaremos exactamente el modelo que vamos a usar, ya que eso se hará de manera más detallada en apartados posteriores, concretamente en la arquitectura del sistema.

Una vez dicho todo esto comenzamos con las soluciones propuestas.

3.2.1. Conteo RPM

A la hora de controlar las revoluciones por minuto de un motor, uno de los elementos más utilizados y accesibles del mercado es sin duda alguna el encoder óptico. En nuestro caso vemos óptimo utilizar un encoder de cuadratura, ya que nos puede resultar muy interesante no solo controlar la velocidad sino también el sentido del giro. Este tiene un funcionamiento muy simple, se acopla mediante una unión sólida al eje del motor un disco ranurado de manera uniforme. Luego se colocan dos sensores ópticos desfasados 90 grados el uno con el otro y sus haces atravesando el disco. Conociendo el número de

ranuras del disco podemos determinar las vueltas que ha dado gracias a las veces por minuto que se interrumpa el sensor. Luego, como los dos sensores están desplazados entre sí, sabiendo cuál se activa primero podremos determinar el sentido del giro.

En la figura 2 se puede ver un esquema simplificado del funcionamiento de un encoder óptico.

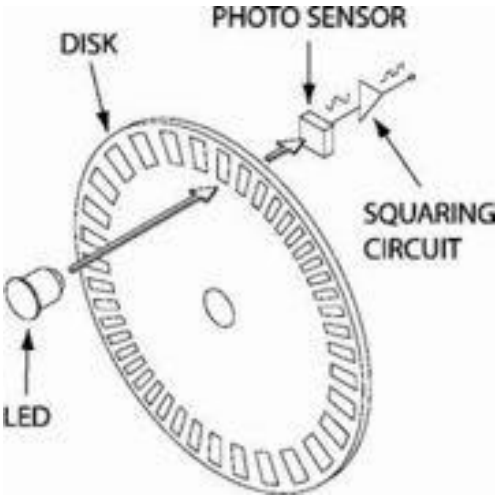


Figura 2 Encoder óptico.
Fuente: [4]

3.2.2. Control de corriente suministrada

Para poder controlar la corriente suministrada vamos a utilizar un Driver para motor DC. Este elemento es muy fácil de conseguir y suelen ser bastante económicos. Se pueden encontrar integrados en módulos con varios elementos de forma que solo tengas que realizar el cableado. En esta placa encontraremos todos los elementos auxiliares necesarios para que funcione correctamente, como por ejemplo un difusor de calor, condensaros, etc. Pese a que generalmente para nombrar a este elemento se use la palabra driver, nosotros en castellano lo llamaríamos controlador de motor, o bien, en sentido literario conductor de motor, ya que va a ser él quien le dé órdenes directas. Básicamente, su funcionamiento es como el de una válvula de agua. Con pequeñas corrientes podemos controlar el grado de apertura de la válvula y dejar pasar más o menos corriente. En nuestro caso, tendremos, por un lado, el driver conectado a los 12 voltios que puede soportar el motor y, por otro lado, estará conectado a nuestro microcontrolador. La manera que tiene de funcionar es amplificando la señal de control

de baja corriente generada por el microcontrolador y convertirla en una señal de alta corriente que pueda alimentar el motor. [5]

3.2.3. Corriente demandad

Para poder medir la corriente demandada por el sistema utilizaremos un sensor de corriente que sea compatible con nuestras características de carga. Necesitamos un sensor cuyo rango de medida esté dentro de nuestro rango de funcionamiento y que sea compatible con nuestro montaje y microcontrolador. Cabe destacar que, por norma general, para medir la corriente hay que realizar una conexión en serie y por tanto resulta un elemento intrusivo en el sistema, ya que si por algún motivo, no se pudieran desmontar los cables que alimentan al motor habría que cortar uno para poder conectar el sensor. Por suerte, existen cantidad de sensores en el mercado a precios muy económicos. En nuestro caso, vamos a buscar algún modulo que contenga el sensor y todos los elementos necesarios para su correcto funcionamiento de tal modo que solo sea necesario realizar el cableado y programar la lectura.

3.2.4. Pantalla de adquisición de datos

Este punto posiblemente sea el más familiar para todos, ya que hoy en día estamos completamente rodeados por este tipo de pantallas. A la hora de seleccionar una para nuestro proyecto vamos a valorar el tamaño y el precio. Los datos que queremos mostrar hasta ahora, como bien hemos señalado son pocos, las revoluciones por minuto y la corriente, por lo tanto, no necesitamos una pantalla de gran tamaño; nos basta con una pantalla pequeña. Tampoco necesitamos que disponga de ningún color de escritura concreto. Dicho esto, solo queda buscar alguna pantalla que sea compatible con nuestro microcontrolador y que tenga un precio asequible.

3.2.5. Dispositivo controlador

Este apartado es uno de los más importantes ya que esta pieza va a ser completamente indispensable para nuestro montaje y va a determinar el lenguaje en el que vamos a programar nuestro código. Para empezar, como hemos mencionado en apartados anteriores, vamos a utilizar un microcontrolador y sin duda es la mejor opción para

gestionar proyectos de este tamaño y presupuesto. Hoy en día, el mundo de los microcontroladores está gobernado por dos grandes marcas, conocidas por todo interesado en la electrónica o programación: Arduino y Raspberry Pi. En la figura número seis podemos ver una Raspberry Pi y la placa de Arduino aparece representada en la figura número uno.

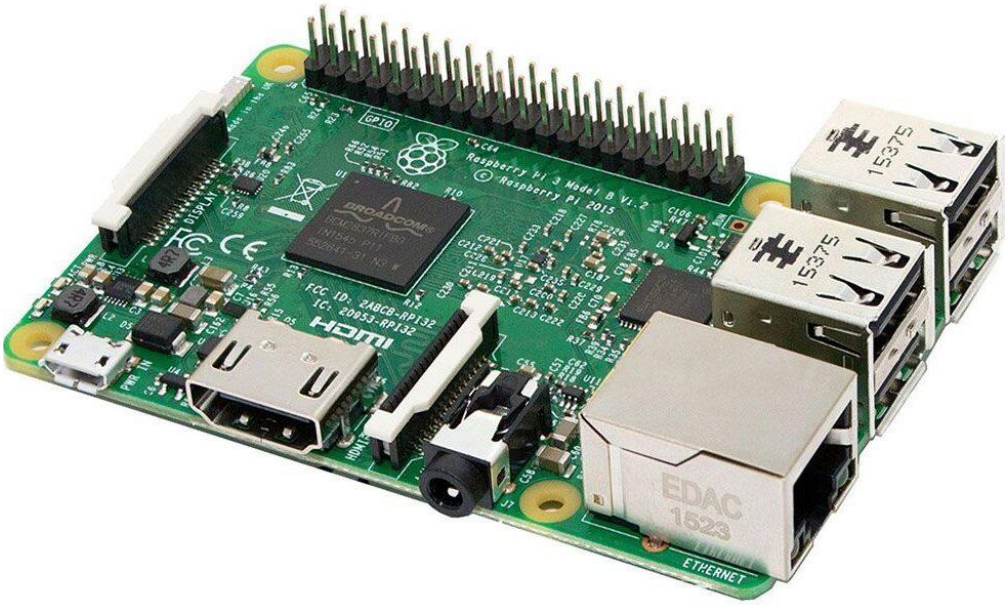


Figura 3 Raspberry Pi.
Fuente: [6]

Pese a que en este apartado no estamos entrado en detalle de la elección de cada elemento, para este caso concreto, se sabe por lo detallado en el apartado estado del arte, que nuestra elección es utilizar una placa Arduino. La elección de Arduino está basada en:

- Su bajo precio, de manera que hace el microcontrolador muy asequible.
- Su potencia de funcionamiento, tenemos capacidad suficiente para controlar nuestro proyecto. Si usáramos una Raspberry estaríamos como se diría de manera coloquial “matando moscas a cañonazos”.
- La cantidad de dispositivos compatible y pensados para funcionar con Arduino.
- Y, por último, la cantidad de información sobre su programación y montajes, tanto a través de su web, como publicados por su comunidad en internet.

4. DISEÑO E INVESTIGACIÓN

4.1. Arquitectura del sistema

En el punto anterior hemos hecho una puesta en común de todas las necesidades que debemos cubrir para realizar el proyecto y hemos presentado los elementos necesarios para ello. En este apartado vamos a pasar a explicar de manera concreta y detallada cada elemento y la razón de su elección. Vamos a puntualizar, que parte del material elegido es material que ya teníamos disponible en el laboratorio, de tal manera que igual hubiera sido posible buscar otros elementos que se acoplasen estrictamente a nuestras necesidades, pero consideramos que principalmente, íbamos a aprovechar lo que tuviéramos a nuestra disposición con el fin de no gastar de manera innecesaria. Esto último, debe tenerlo en cuenta cualquier persona que quisiera replicar este trabajo. Así pues, se le invita a utilizar primero el material que tenga a su disposición siempre y cuando satisfaga todas las necesidades planteadas anteriormente. Dicho todo esto vamos a pasar a la explicación detallada de los componentes.

4.1.1. Encoder

Ya hemos mencionado el tipo de encoder que estábamos buscando. Queríamos un encoder de cuadratura que nos permitiera acoplarlo a la rotación del motor y medir su velocidad y dirección. Para esto, necesitamos que disponga de dos sensores desfasados entre sí 90 grados. Vamos a utilizar en este caso el encoder llamado “Speed Sensor Encoder Coded” de la marca Hailangniao.

Figura 4 Encoder de cuadratura.
Fuente: [7]

Este encoder de procedencia china cumple con todos nuestros requisitos; consta de dos sensores desfasados y un precio muy económico. Como se puede apreciar en la imagen, nuestro encoder consta de dos partes diferentes, el disco ranurado, que se debe unir al eje del motor y el sensor que debe detectar las ranuras del disco. A simple vista, resulta evidente que vamos a necesitar una estructura que nos permita colocar de manera fija el sensor en la posición correcta de lectura siendo atravesado por el disco. Se debe tener en cuenta también, si es posible unir el disco al eje del motor solo con la pieza que nos proporcionan ellos o no. Las características técnicas más importantes del encoder son las siguientes:

- El disco consta de 100 ranuras, por lo tanto, ya sabemos las interrupciones que debe tener el sensor en una vuelta completa.
- El sensor tiene un total de 4 cables, alimentación, masa, sensor uno y sensor 2.
- El sensor está diseñado para microcontroladores tipo Arduino y por lo tanto, el cable de alimentación se conecta a 5 voltios directamente a Arduino.
- Los dos sensores tienen las señales también compatibles con Arduino y por lo tanto su activación es fácilmente gestionable con este.

En la figura siguiente vamos a mostrar el comportamiento que tiene el encoder en cuento a la señal que leída.

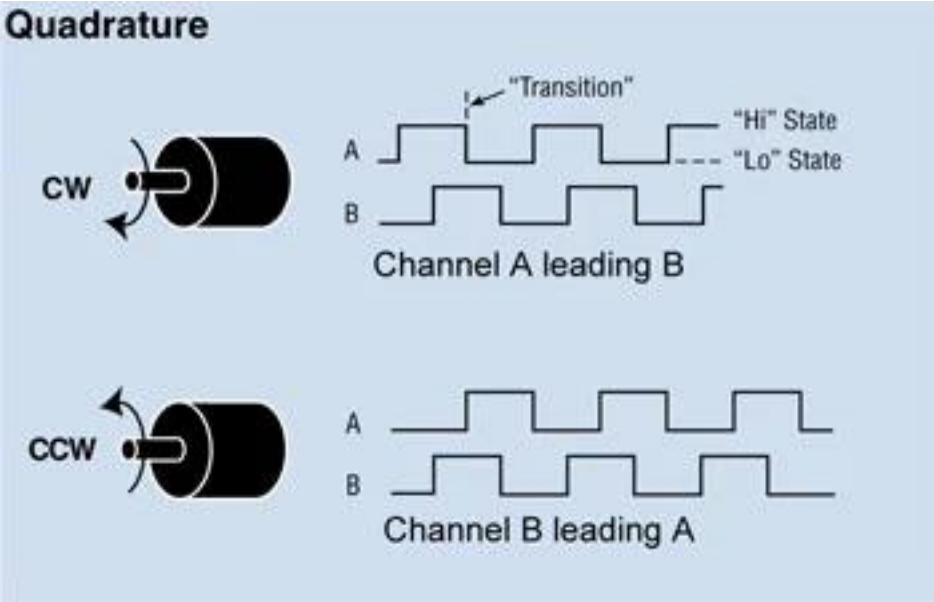


Figura 5 Encoder de cuadratura.
Fuente: [8]

Tal y como se muestra en la figura superior podemos ver que nos bastara con verificar el estado de por ejemplo el sensor "B" dentro de la rutina de interrupción del sensor "A". El código estará diseñado de manera que, asociaremos la rutina de interrupción al flanco de bajada del sensor "A", dentro de esa rutina verificaremos el estado del sensor "B". Si el estado es "HIGH" sabemos que giramos en el sentido de las agujas del reloj si el estado en "LOW" giramos en sentido contrario. Gracias a esto podemos implementar una rutina que cuente las vueltas en un sentido pero que si por algún casual giramos en sentido contrario el motor reste el valor correspondiente.

4.1.2. Motor DC

El motor que vamos a utilizar en nuestro proyecto es un "Hanpose 775 motor DC 12v 24v 80W". Todo nuestro trabajo gira en torno a este motor, ya que es la pieza que teníamos desde el momento inicial y de la cual nos interesa poder tener control de su velocidad de manera precisa. Este es un motor que se usa con bastante frecuencia en máquinas de control numérico o impresoras 3D, ya que tiene un amplio rango de

modelos para diferentes potencias y un par motor muy alto; es un motor con mucha reducción y por lo tanto fuerza. En nuestro caso concreto nos interesa poder controlar la velocidad porque es el motor usado en las máquinas de control numérico económicas como la CNC1820 o CNC3018 utilizada para crear prototipos de placas de circuitos impresos, así como en taladros a batería, amoladoras, dremmels, etc…

En la figura siguiente podemos ver por primera vez la forma que tiene nuestro motor y nos permite plantearnos los elementos que vamos a necesitar para construir el soporte del motor.

Figura 6 Motor Hanpose 775.
Fuente: [9]

En primer lugar, vamos a ver las especificaciones de diseño exterior del motor para saber las dimensiones y tamaño que debe tener el soporte.

Tabla 1 Especificaciones de diseño de motor Hanpose775.
Fuente: [10]

Diámetro del Eje	5 mm
Tamaño agujero de montaje	M4
Agujeros de montaje	2

Fijándonos en el diámetro del eje, vemos que la pieza que viene en el encoder para unir el disco con el eje del motor no tiene el diámetro suficiente para poder acoplarse y, por lo tanto, debemos pensar en alguna solución para este acoplamiento. También, si fijamos nuestra atención en la figura número cinco, vemos que el eje del motor no solo sale por la delantera, sino que se prolonga hasta la parte trasera, lo cual nos permite acoplar el disco del encoder sin interferir en su funcionamiento. Luego, observamos que nuestro motor dispone de dos agujeros roscados de métrica 4, los cuales nos pueden ser de mucha utilidad para atornillar el motor a una base. En la siguiente figura, se pueden ver los dos agujeros roscados alineados horizontalmente con el eje del motor.

Figura 7 Motor hanpose 775 vista de la cara frontal.
Fuente: [10]

Una vez vistas sus características de diseño exterior procedemos a tratar sus características de funcionamiento. Como ya hemos comentado existen varios modelos con diferentes potencias, en nuestro caso utilizamos el modelo de 80W.

Tabla 2 Especificaciones de funcionamiento de motor Hanpose 775.
Fuente: [10]

Potencia del motor(W)	Voltage (V)	Corriente Máxima (A)	Par Máximo (KG)	Velocidad Máxima (RPM)
80	12	6	1.8	4000
	24			8000

En nuestro caso, vamos a utilizar una fuente de alimentación de 12 voltios y por lo tanto la única característica que nos cambia con respecto a 24 voltios, es a velocidad máxima. Dicho esto, ya sabemos que con 12 voltios y 6 amperios nuestro motor alcanzará 4000 rpm como máximo.

4.1.3. Soporte de motor y encoder

En este apartado, conseguimos solucionar a la vez los tres problemas que teníamos a la hora de sujetar el motor, acoplar el disco del encoder y sujetar el sensor. Hemos construido el soporte a partir de una chapa metálica. Esta ha sido cortada en forma de "L" de manera que la parte larga hace de base para sujetar el peso del motor y la parte corta tiene tres agujeros taladrados con su respectivo tamaño para dos tornillos de métrica 4 y el eje del motor. Se le ha añadido la parte de la base que hace contacto con el suelo un adhesivo de corcho para absorber las vibraciones del motor y minimizar el ruido.

Las dimensiones del soporte son las siguientes:

- Ancho de todo el soporte 5 cm
- Altura de la parte corta 5 cm
- Longitud de la base o parte larga 7.5 cm
- La chapa esta doblada a 90 grados

Véase en la siguiente figura una vista lateral del soporte.

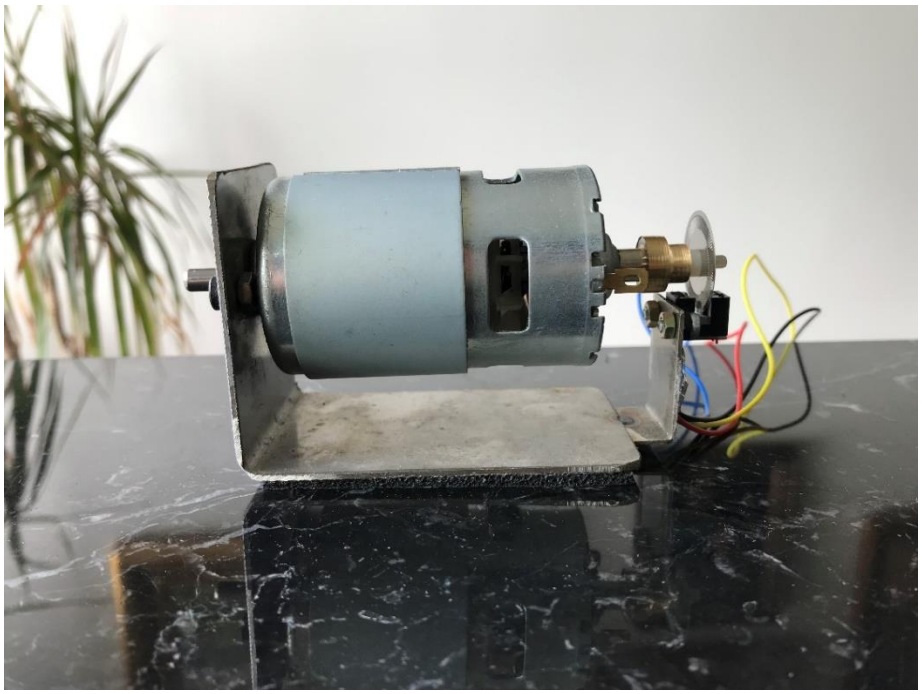


Figura 8 Vistal lateral de soporte y motor Hanpose.

En el anexo se puede ver el boceto en 2D del diseño del soporte.

Gracias a la figura numero 8 podemos ver una vista frontal de nuestro soporte y como quedan los tornillos alineados con el eje, el motor sujeto y suspendido en el aire.

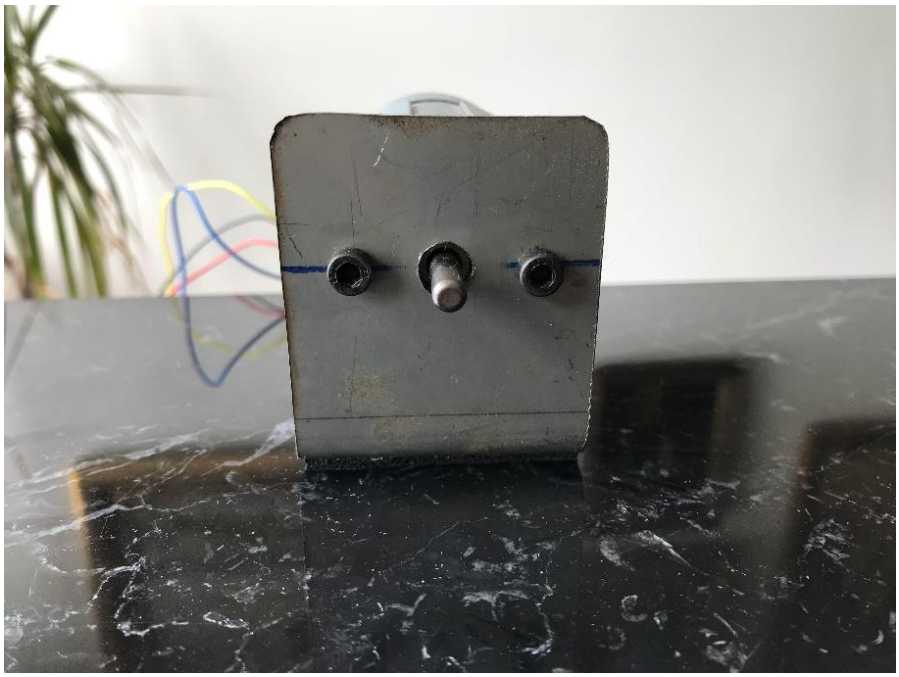


Figura 9 Vista frontal del soporte y motor Hanpose

Para conseguir una unión sólida entre el disco del encoder y la parte trasera del eje del motor hemos unido dos piezas en una. Por un lado, utilizamos una pequeña pieza metálica con dos aperturas de diferentes tamaños, una más grande que la otra. Con la ayuda de un torno, conseguimos ajustar en la apertura grande al eje del motor y añadimos unas gotas de pegamento para asegurarnos su unión. Por otro lado, en la apertura pequeña encajamos la pieza blanca que nos venía con el encoder de serie para acoplarlo con el eje. De este modo conseguimos que el disco gire solidario a nuestro eje. En la figura siguiente se ve con bastante claridad la pieza en cuestión.

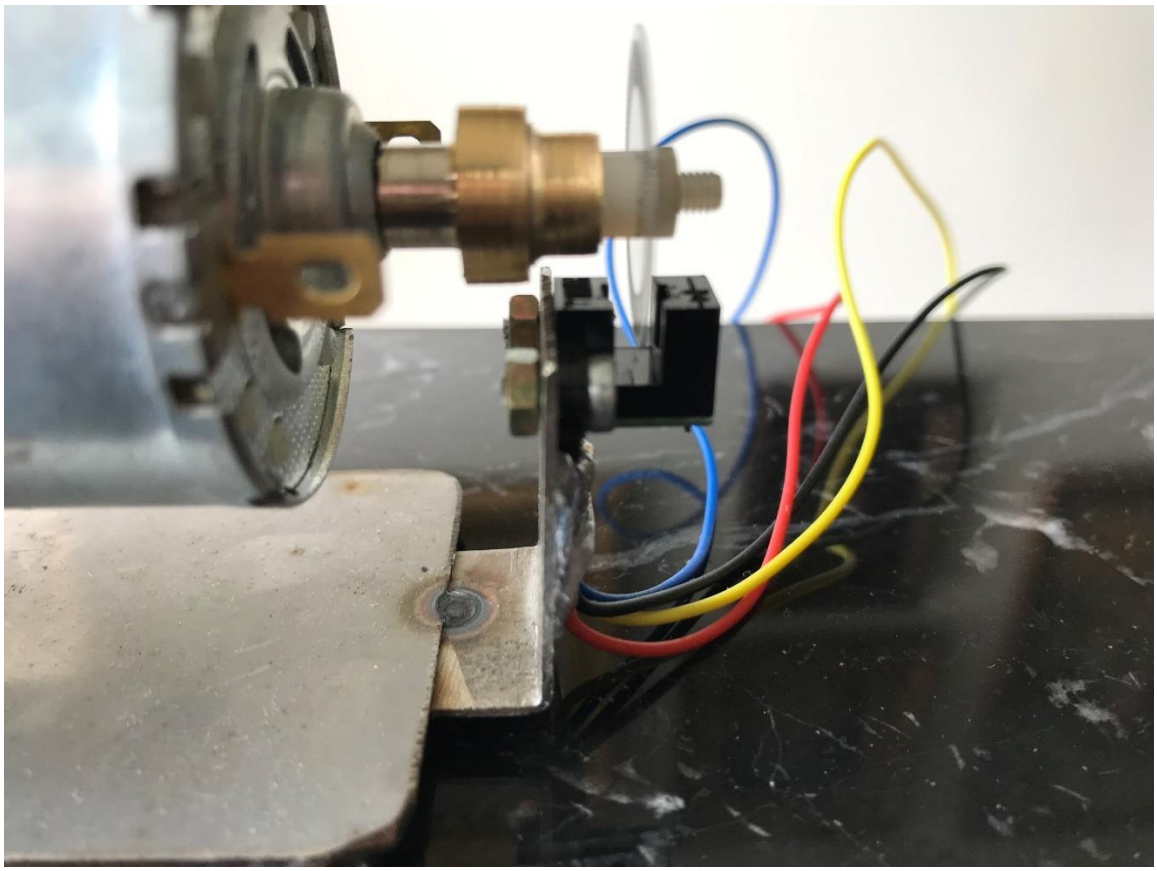


Figura 10 Vistal lateral del soporte con zoom en el encoder

Solo nos queda solucionar el soporte para el sensor. Como se puede observar en la figura número nueve hemos añadido otra pieza de metal en forma de “L”, esta vez más pequeña. La pieza ha sido unida a la base del soporte con un punto de soldadura la cual queda completamente fijada en inmóvil. En la parte superior, se le han realizado dos agujeros de tal manera que el sensor queda acoplado con dos simples tornillos y tuercas.

Las dimensiones del soporte del sensor del encoder son las siguientes:

- El ancho de todo el soporte de 2 cm.
- Parte corta unida a la base mediante soldadura de 1 cm.
- Altura de la parte larga 3 cm.
- Placa doblada a 90 grados.

Véase el plano detallado en el anexo.

4.1.4. Fuente de alimentación

La fuente de alimentación que usamos es de origen china de la marca CCDSTAR. La elección de este elemento ha sido simplemente por el hecho de tenerla disponible en el laboratorio y cumplir nuestras demandas de funcionamiento.

Figura 11 Fuente de alimentación.
Fuente: [11]

Se trata de un transformador de corriente alterna a corriente continua, se puede conectar entre 100 y 240 Voltios y la salida es corriente continua a 12 voltios y 2.5 amperios. Quizás la fuente vaya un poco corta de potencia, pero en nuestras pruebas en las que el motor no lleva carga nos sirve

Disponemos también de un adaptador para la salida de la fuente de alimentación que nos permite conectar cables de montaje directamente. Véase en la siguiente figura.

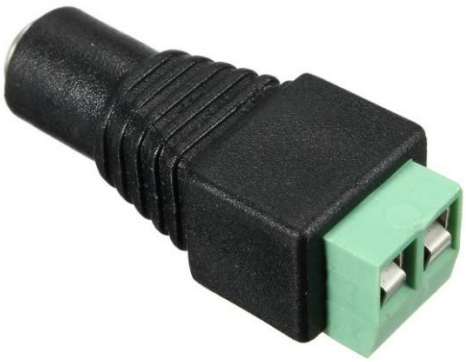


Figura 12 Adaptador de salida de corriente para cableado.
Fuente: [11]

4.1.5. Motor Driver

El driver que hemos elegido es "Full-Bridge-L298N". Este módulo está compuesto por todos los elementos necesarios para el correcto funcionamiento del dispositivo. Cuenta con un regulador LM7805 que suministra 5 voltios a la parte lógica del integrados L298N. También dispone de tres jumpers, uno para determinar el rango de voltaje de funcionamiento y dos para habilitar las salidas del módulo, ya que este módulo cuenta con dos salidas para poder controlar dos motores simultáneamente de hasta 2 amperios. Como bien hemos mencionado antes, uno de los jumpers nos permite determinar el rango de voltaje de funcionamiento. Los rangos de voltajes de funcionamiento en corriente continua son:

- Si el jumper de selección de 5 voltios esta activo el rango de funcionamiento es de 6 a 12 voltios y no será necesario alimentar el módulo con 5 voltios extras, ya que el regulador LM7805 se encargará de enviárselos a la parte lógica del driver.
- En el caso de que el jumper de selección de 5 voltios no estuviera activo pasamos a un rango de funcionamiento de 12 a 35 voltios y con la diferencia de que esta vez sí que será necesario alimentar el módulo con 5 voltios extra, ya que el regulador LM7805 no será capaz de transformar esos altos voltajes a 5 voltios. [12]

Véase en la siguiente figura una imagen del Driver con sus diferentes entradas y salidas señalizadas y jumpers.

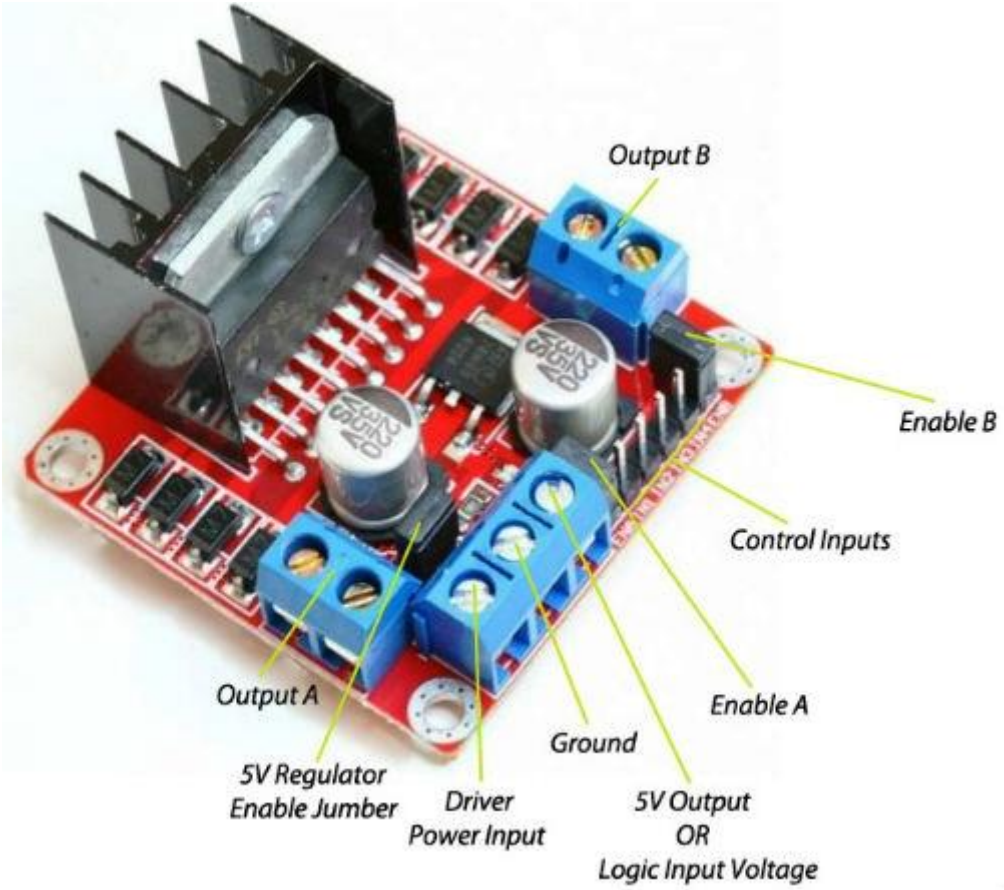


Figura 13 Modulo Driver L298N.
Fuente: [13]

Con respecto a la elección de este módulo, debemos comentar que lo teníamos a nuestra disposición en el laboratorio. Debería funcionar sin ningún problema con nuestro motor, aunque la limitación de 2 amperios nos resulta preocupante. Si apareciera algún problema con él, buscaríamos alguna solución enfocándonos siempre en el funcionamiento y el aspecto económico. Tal y como hemos comentado anteriormente, uno de nuestros objetivos es sacar el máximo rendimiento al material que tenemos a nuestra disposición antes de optar por comprar todo nuevo.

4.1.6. Sensor de corriente

El sensor de corriente que vamos a utilizar es el "ACS712ELCTR-05B-T". Este trabaja con un sensor de efecto Hall, el cual detecta el campo magnético que se produce por inducción de la corriente que circula por la línea medida. Nos entrega una salida de voltaje proporcional a la corriente mesurada. Nuestro modelo concreto tiene un rango de funcionamiento de -5 a 5 voltios y una sensibilidad de 185 mV/A.

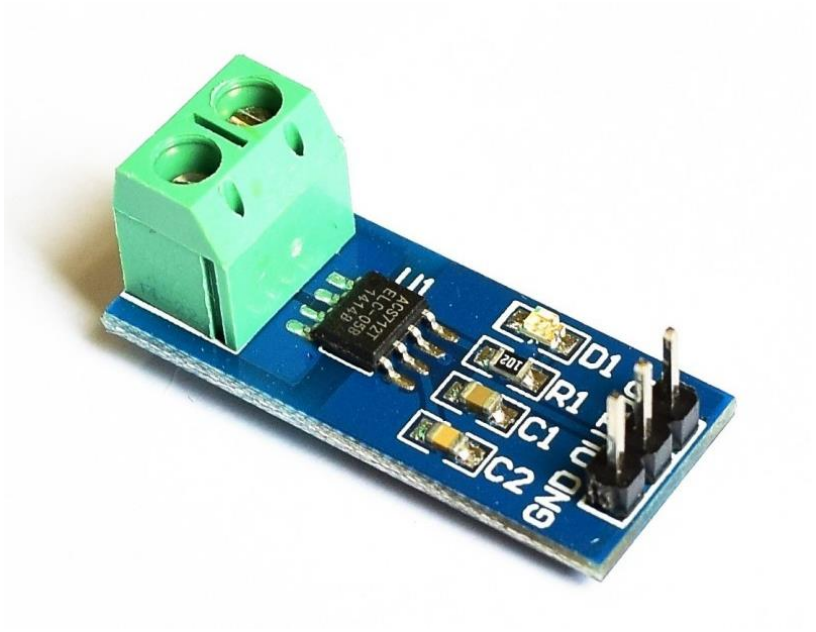


Figura 14 Sensor de corriente ACS712.
Fuente: [14]

Se puede ver en la figura número trece que nuestro sensor viene acoplado en un módulo que contiene todos los elementos necesarios para su funcionamiento y solo requiere de cableado.

Vamos a explicar brevemente como se debe proceder a la hora de calcular la corriente con el sensor. Este nos entregará 2.5 voltios para una corriente de 0 amperios e irá incrementando proporcionalmente en función de su sensibilidad.

$$V = sensibilidad * I + 2.5 \qquad (Ecuación\ 2)$$

Por lo tanto, la ecuación que nos da directamente la corriente en función del voltaje leído sería:

$$I = \frac{V-2.5}{sensibilidad} \qquad (Ecuación\ 3)$$

Es muy importante tener en cuenta el modelo del sensor ACS712 que estamos utilizando ya que la sensibilidad cambia con cada uno y nos saldrían los cálculos erróneos

4.1.7. Pantalla de adquisición de datos

La pantalla de adquisición de datos que hemos decidido utilizar es el modelo "Oled SSD1306" esta pantalla usa diodos orgánicos de emisión de luz, tiene un bajo consumo de energía y un diseño tan fino como ligero. Las siglas OLED vienen de "Organic Light-Emitting Diode" que traducido sería diodo orgánico de emisión de luz. [15]

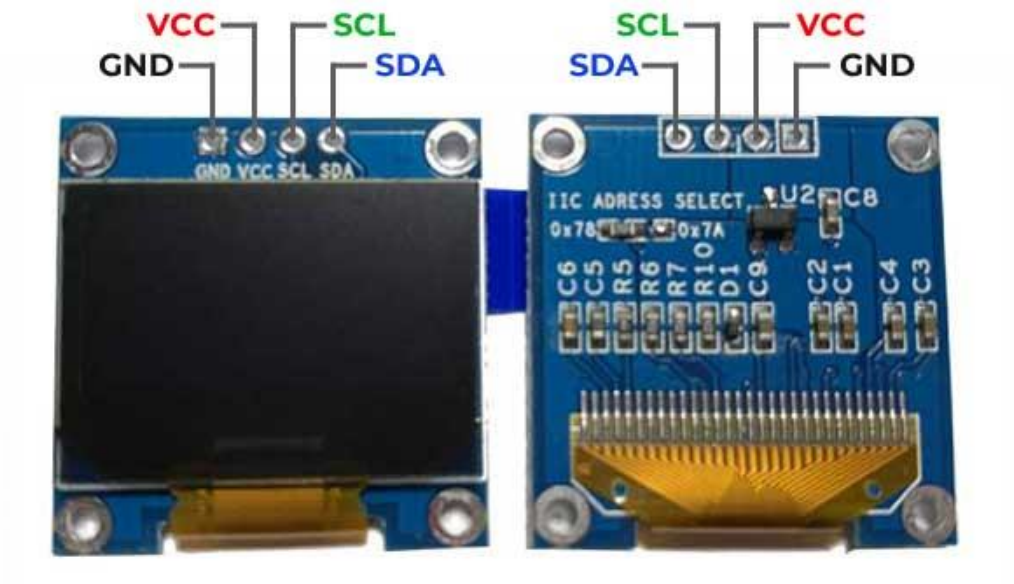


Figura 15 Oled SSD1306.
Fuente: [15]

Esta pantalla no solo destaca por su imagen nítida y brillante sino por su simplicidad a la hora de conectarte. Simplemente hay que alimentarla desde Arduino, conectarla a masa y utilizar los dos pines específicos denominados como “SCL” y “SDA” para establecer la comunicación con ella. Por todos estos aspectos resulta claramente superior a su hermana la pantalla LCD.

4.1.8. Protoboard

La protoboard es un elemento auxiliar que nos facilita en gran medida la conexión de los cables de nuestro proyecto.

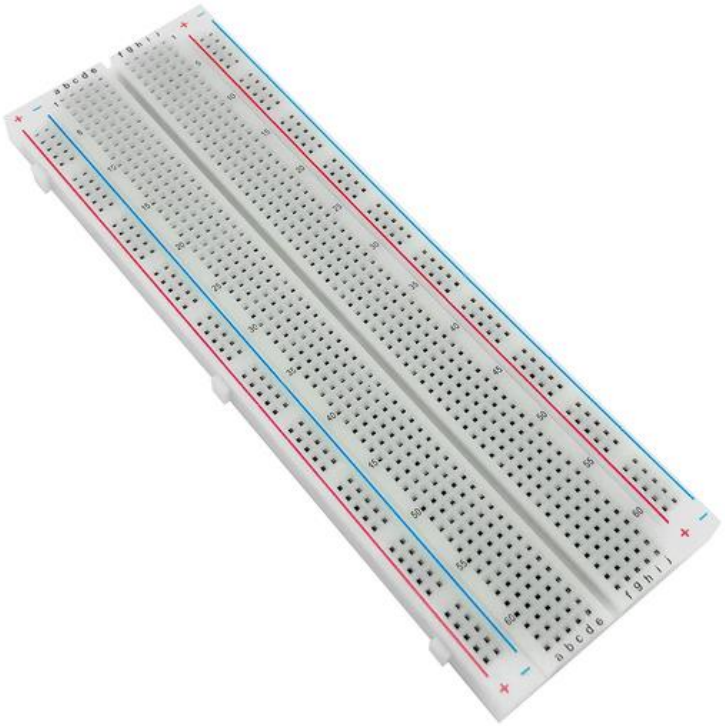


Figura 16 Protoboard.
Fuente: [16]

El funcionamiento de la protoboard es muy simple, en ambos extremos tenemos dos líneas marcas con un “+” y un “- “, estas están unidas por un material conductor a lo largo de toda la tabla. Por otro lado, las cavidades interiores denominadas como “abcde” y “fghij” están unidas entre sí, respectivamente y de manera perpendicular a las lines de “+” y “- “. Un punto fuerte a recalcar del uso de esta placa es la facilidad de poder poner las masas de todos los elementos al mismo nivel, ya que si no lo hacemos podremos ocasionar problemas e incluso dañar nuestro material.

4.1.9. Placa Arduino Mega 2560

Es cierto que todos y cada uno de los elementos utilizados en el proyecto resultan de carácter indispensables, pero el Arduino Mega 2560 es posiblemente la pieza más relevante de nuestro montaje ya que asume el papel de cerebro en nuestro proceso. El Arduino Mega 2560 es una placa de desarrollo basada en un microcontrolador ATmega2560, esta placa está compuesta de 54 pines de entradas y salidas, 16 entradas analógicas, cristal oscilador de 16MHz, conexión USB, jack de alimentación, conector ISCP y botón de reset. Cuenta con una memoria flash de 256K y la velocidad del reloj es de 16MHz

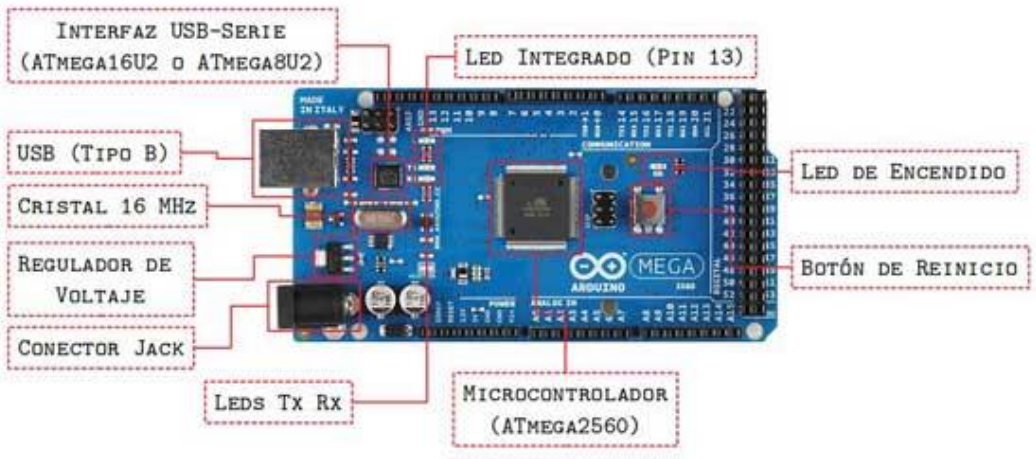


Figura 17 Arduino Mega 2560.
Fuente: [17]

Para la elección de esta placa hemos tenido en cuenta dos factores importantes. El primero de ellos, es que pese a no haber trabajado nunca de manera constante con Arduino, sí que es cierto que durante la carrera hemos aprendido a programar en C, siendo este la base del entorno de desarrollo de Arduino. Por lo tanto, decidimos optar por este microcontrolador confiando en que nos sería familiar la programación del mismo. El segundo factor es determinante en relación al modelo utilizado de Arduino. ¿Por qué hemos decidido utilizar el Arduino Mega 2650 y no cualquier otro modelo?

Pues la respuesta a esta pregunta es clara y concisa. Hemos optado por este modelo porque es el que teníamos disponible en el laboratorio y trabajando acorde a nuestro

objetivo de optimizar el material ya disponible, consideramos oportuno quedarnos con este modelo. Es cierto que seguramente habría bastado con usar su hermano pequeño, el Arduino Uno, pero hubiera requerido comprarlo específicamente y no estaríamos cumpliendo el objetivo propuesto.

4.2. Código detallado

En este apartado vamos a seccionar el código creado en Arduino IDE. Esta es una de las partes principales del trabajo y posiblemente a la que se le ha dedicado la gran parte del tiempo invertido en este trabajo. Todo el código se ha pensado de forma que no solo funcione correctamente, sino que optimice al máximo nuestros recursos y sea claro y sencillo de entender.

4.2.1. Librerías"

Pese a que la parte de librerías está compuesta de tres "#includes", realmente, podríamos considerar que estamos utilizando dos librerías, ya que dos de ellas están destinadas a trabajar con el mismo elemento, pero hay que añadirlas por separado.

```
#include <TimerOne.h>
#include <Adafruit_SSD1306.h>
#include <Wire.h>
```

Figura 18 Librerías usadas en Arduino

La primera librería llamada "TimerOne.h" la utilizaremos para poder asociar interrupciones con el tiempo.

Esta nos será realmente útil a la hora de determinar un tiempo de muestreo y facilitarnos el cálculo del PID. Profundizaremos más cuando analicemos el código destinado al diseño del PID.

Las otras dos librerías llamadas “Adafruit_SSD1306.h” y “Wire.h” son las que nos permiten utilizar la pantalla Oled SSD1306, utilizada para mostrar datos por pantalla. Como hemos comentado anteriormente, esta pantalla necesita de estas dos librerías para funcionar y, por lo tanto, pese a ser dos, podríamos considerarlas como una sola unidad. Estas librerías nos permiten mostrar datos en la pantalla oled utilizando funciones creadas por la propia empresa que las comercializa. De esta manera, la utilización de la pantalla se simplifica en gran medida y nos exime de tener que estudiar a fondo como está diseñada para utilizarla.

4.2.2. Declaración de variables

Una vez incluidas las bibliotecas, pasamos a declarar todas las variables que vamos a necesitar en nuestro programa. La declaración de variables está compuesta de tres partes.

```
#define EncoderA 2
#define EncoderB 7
#define ANCHO 128
#define ALTO 64
#define OLED_RESET 4

Adafruit_SSD1306 oled(ANCHO, ALTO, &Wire, OLED_RESET);

int IN3 = 5;    // Input3 conectada al pin 5
int IN4 = 4;    // Input4 conectada al pin 4
int ENB = 8;    // ENB conectada al pin 8 de Arduino
float voltageSensor; // voltage leido por el sensor ACS712
float corriente;     // Corriente consumida por el motor
volatile long int cont=0;  //Cuenta interrupciones del encoder
volatile long int rpm=0; //Convierte las interrupciones del encoder en rpm
volatile long int setPoint=0; //Las rpm que le pediremos fijar
long int Cm=0; //Para muestrear cada 200 milisegundos
volatile long int U=0; //Valor de la señal PID
volatile float P=0;    //Parte proporcional
volatile long int error=0;  //error
volatile float I=0;// Parte integral
long int T=200; // 200 milisegundos
float Kp=0.02;  //Constante de la parte proporcional P del PI
float Ki=0.0001;  // Constante de la parte proporcional I del PI
```

Figura 19 Declaración de variable en Arduino

En la primera parte, utilizamos “#define” para fijar ciertas variables que sabemos que no vamos a querer modificar, o bien porque queremos que estén en esos pines concretos y que no sean modificados, como el caso de “EncoderA” y “EncoderB”, o bien, porque son valores fijos como “ANCHO” y “ALTO” que dependen de las dimensiones físicas de nuestra pantalla oled. No vamos a prestar atención a la variable “OLED_RESET”, ya que es una variable utilizada en el modelo oficial, pero nuestra pantalla no dispone de ella ya que se trata de una imitación. Pese a no disponer de ella es necesario declararla para inicializar nuestra pantalla correctamente.

Por otro lado, en la segunda parte, es cierto que no estamos declarando una variable como tal, pero, estamos inicializando nuestra pantalla oled asociándole los valores previamente declarados y dejándola lista para su funcionamiento.

Por último, la tercera parte consta del mayor número de variables declaradas en este programa. Todas estas variables son susceptibles de ser, o bien, modificadas por el programa, o bien, manualmente en caso de que fuera necesario. Las tres primeras “IN3”, “IN4” y “ENB” asocian las entradas del módulo Driver L298N a los pines 5, 4 y 8 de Arduino respectivamente. La razón por la cual no hemos añadido estas variables con los “#define” es por organización. Los pines asociados a los encoders son específicos para recibir interrupciones, y, por lo tanto, no deben ser modificados. En cambio, los pines asociados a “IN3”, “IN4” y “ENB” no requieren nada más que tener habilitada la señal PWM, yendo estos del pin 2 al 13.

Es por esta flexibilidad de conexión, que hemos decidido declararlos en esta parte del código. Las variables “voltageSensor” y “corriente” almacenan la lectura del sensor de corriente ACS712 y la conversión en corriente hecha por el programa. A continuación de estas dos, tenemos “cont” que se usará para contar el número de interrupciones leídas por el encoder. Seguidamente, aparecen las variables asociadas al PID; “rpm” almacena las revoluciones por minuto actuales de nuestro motor, “setPoint” tomará el valor que nosotros le demos, siendo este las revoluciones que queremos fijar con nuestro programa. “U” será la señal de control que generemos. Este será fruto de la suma de “P” e “I” siendo estas dos variables el cálculo de la parte proporcional e integral de nuestro PID respectivamente. La parte proporcional se calcula usando la variable “Kp” así como la integral usando “Ki”. La base del cálculo de ambas partes del PI es “error” la cual

almacenará la diferencia entre “setPoint” y “rpm”. La variable “Cm” la usaremos para mostrar los datos con una frecuencia concreta y la variable “T” determinará el tiempo de muestreo en milisegundos.

4.2.3. Inicialización

En Arduino es indispensable la fase de inicialización para poder trabajar con él. En este apartado se les asocia una dirección concreta a todas las que estén asociadas a un pin. También se ponen en marcha las funciones que vamos a necesitar activas durante nuestro bucle principal.

```
void setup()
{
 pinMode (ENB, OUTPUT);
 pinMode (IN3, OUTPUT);
 pinMode (IN4, OUTPUT);
  pinMode (EncoderA, INPUT);
 pinMode (EncoderB, INPUT);
 Wire.begin();
 oled.begin(SSD1306_SWITCHCAPVCC, 0x3C);
 Serial.begin(57600);
 Timer1.initialize(T*1000); // Esta en microsegundos
 Timer1.attachInterrupt(PID); //Una interrupcion condicionada por el tiempo para calcular las rpm
 attachInterrupt(0,rutina,FALLING);  //Interrupcion del encoder inicializada
}
```

Figura 20 Inicialización del programa de Arduino

Los “pinMode”, simplemente, determinan si el pin que has conectado va a enviar o a recibir información, nombrando “OUTPUT” a los que envían e “INPUT” a los que reciben. Tanto “Wire.begin()” y “oled.begin” sirven para inicializar la pantalla oled y que permanezca a la espera de recibir órdenes. Utilizamos “Serial.begin” para activar el monitor serial que usaremos para introducir el valor de revoluciones por minuto que queremos fijar y en caso de que fuera necesario, podríamos mostrar por pantalla el valor de alguna variable. “Timer1.initialize” nos sirve para determinar el intervalo de tiempo en el que queremos que esta función se active. “Timer1.attachInterrupt” nos permite asociar una interrupción al tiempo determinado en “Timer1.initialize”, y por lo tanto, controlar el tiempo de muestreo. Por último, “attachInterrupt” nos permite asociar al pin 2 una rutina de interrupción que se activaáa cuando el sensor del encoder esté en flanco de bajada.

4.2.4. Funciones

En nuestro programa, usamos solamente dos funciones. La primera, es la rutina de interrupción asociada al sensor del encoder para contar las revoluciones por minuto, y la segunda, es la que calcula el PID.

4.2.4.1. Función rutina

```
void rutina(){ // Interrupcion del encoder, cuenta las veces que lee un aspa
  if(digitalRead(EncoderB)== 0){
    cont--;
  }
  else{
    cont++;
  }

}
```

Figura 21 Función rutina en Arduino

Esta función es una rutina de interrupción asociada al pin número dos. Tiene un funcionamiento simple pero efectivo; cada vez que recibe la interrupción, el programa verifica el estado del otro sensor del encoder y suma o resta uno en la variable "cont" dependiendo del sentido de giro; de esta forma sabemos cuántas aspas del disco ha leído. Gracias a un simple cálculo realizado, en la siguiente función podemos determinar las revoluciones por minuto de nuestro motor.

4.2.4.2. Función PID

La Función PID es, sin duda alguna, la parte principal de este trabajo. Esta función es la que se encarga de controlar y regular el funcionamiento de nuestro motor de manera completamente automática. Véase en la siguiente figura.

```
void PID(){     //Interrupcion de cada segundo Mostrando las rpm(Correccion para el muestro a 20 milisegundos)
 rpm=((60.0*cont)/(T/1000.0))/100;
 cont=0;
      //Empezar calculos PI
 error=setPoint - rpm;
 P=Kp*error;// Leer calcular error, calcular accion de control y enviar.
 I=(error*T*Ki+I);  // integrador, (T es el tiempo de muestreo)
 U=P+I;  // Señal de control PI, Proporcional e Integrador
 if(U>0){      // Modificamos el sentido de giro en funcion del signo de mi señal de entrada,
    digitalWrite (IN3, HIGH);            // positivo a derechas y negativo izquierdas
    digitalWrite (IN4, LOW);
 }
 else{
   digitalWrite (IN3, LOW);
    digitalWrite (IN4, HIGH);
    U=abs(U);
 }
 if(U>255){
   U=255;
 }
 analogWrite(ENB,U);

}
```

Figura 22 Función PID en Arduino

Las dos primeras líneas de la función se encargan de convertir con un rápido calculo la variable “cont” generada por la primera función en revoluciones por minuto. Como podemos observar, es determinante saber con qué frecuencia se va a realizar el cálculo para que la conversión tenga sentido. Aquí es donde vemos la utilidad de haber asociado esta interrupción a un temporizador concreto y conocido. La influencia de este dentro del cálculo se ve representado por la variable “T”. Nada más realizar el cálculo se pone “cont” a cero para que pueda volver a almacenar lecturas lo antes posible.

Antes de seguir desglosando del código, vamos a realizar una breve explicación de cómo hemos interpretado el término proporcional e integral con el fin de poder implementarlos en Arduino. En la ecuación siguiente, podemos ver la forma matemática que tiene una señal PI, que es la que vamos a implementar en nuestro Arduino.

$$señal\ de\ control = K_p(e(t) + \frac{1}{T_i}\int_0^t e(t)\,dt$$

En primer lugar, aparece el término proporcional que es Kp por el error en un tiempo determinado. Este término es realmente sencillo a la hora de plasmarlo en Arduino puesto que simplemente es una multiplicación.

El segundo término, como podemos observar, se trata de una integral, elemento que es difícilmente representable en Arduino como tal. ¿Cuál es la solución propuesta para esta situación? Sabiendo que el termino integral representa el área bajo la curva de nuestro sistema y que nosotros tomamos datos con un tiempo de muestreo determinado y no de manera continua, la solución a nuestro problema es discretizar la curva con el fin de facilitar el cálculo. La siguiente figura muestra un ejemplo de discretización.

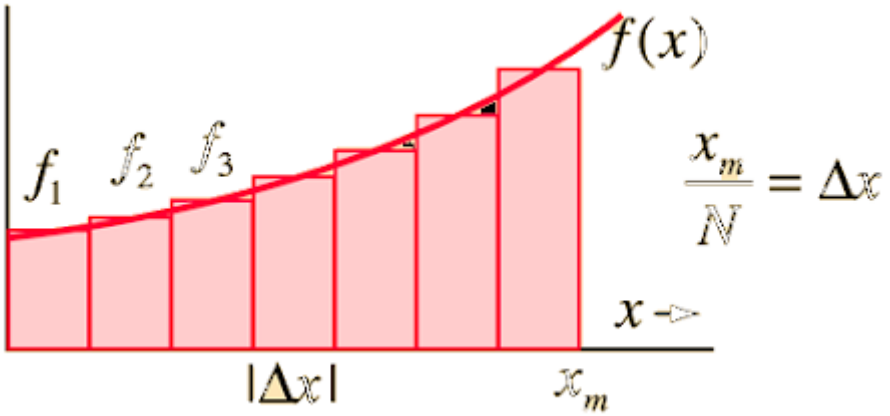


Figura 23 Curva discretizada.
Fuente: [18]

En nuestro caso, pondríamos como eje de abscisas el tiempo transcurrido y como eje de ordenadas el error calculado. Podemos ver como al discretizar, conseguimos dividir el área bajo la curva en una suma de rectángulos, por lo tanto, para calcularla bastará con sumar el área de todos los rectángulos generados. A la hora de realizar el cálculo, volvemos a encontrar de extrema utilidad el hecho de haber ligado nuestra función PID a una interrupción temporal determinada, ya que podemos observar que la base de nuestros rectángulos siempre será el tiempo de muestreo fijado, en nuestro caso 200 milisegundos, y la altura será el error calculado en ese mismo instante. Dicho todo esto ya podemos pasar a comentar nuestro código con pleno conocimiento de sus orígenes.

Siguiendo con el código, el orden en el que se han escrito las siguientes líneas es muy importante e incluso determinante. La primera línea almacena en la variable “error” la diferencia entre “setPoint” y “rpm”. La siguiente línea, calcula el término proporcional siendo este una simple multiplicación tal y como se comentó anteriormente. A continuación, pasamos a calcular el término integral y podemos ver que consta de dos

partes; la primera, resulta ser el cálculo del área del rectángulo en ese instante de tiempo ya comentado y el segundo término, es una suma de sí mismo para ir almacenando el valor de todas las áreas calculadas. Seguido de todo esto, tenemos la variable "U" que resulta ser la suma de "P" e "I".

A continuación, tenemos dos "if". El primero, invierte el sentido del giro del motor en función del signo de "U" y en caso de ser negativo, le aplica valor absoluto para que siempre sea positivo. El segundo, acota el límite superior de "U" a 255 ya que es el valor máximo que puede tomar nuestra señal PWM en Arduino. Por último, enviamos el valor de "U" al driver para que dé la corriente correspondiente al motor.

4.2.4.3. Bucle principal

El bucle principal es la última pieza del programa, comúnmente se declara como "void loop()". Este se llama bucle ya que es la parte del programa que se va a repetir indefinidamente mientras el Arduino reciba alimentación.

```
void loop()
{

 if(Serial.available() != 0){     //Para escribir por el monitor serial, verifica si le envias algo
      String str = Serial.readStringUntil('\n');  //Para leer un numero de mas de una cifra hay que leerlo como string
       setPoint = str.toInt();      //Convertimos la string en un entero para poder leerlo más addelante en el analogWrite


 }

if(millis()-Cm>=T){
  Cm=millis();
  voltageSensor = analogRead(A0)*5.0/1023.0;
  corriente= (voltageSensor - 2.5)/0.185;
  oled.clearDisplay();
  oled.setTextColor(WHITE);
  oled.setCursor(0,0);
  oled.setTextSize(2);
  oled.print (rpm);
  oled.print(" rpm");
  oled.setCursor(0,40);
  oled.setTextSize(2);
  oled.print(corriente);
  oled.print(" A");
  oled.display();
}
```

Figura 24 Bucle principal de Arduino

Este bucle tiene dos utilidades bien definidas. La primera la podemos encontrar fijándonos en la parte superior del bucle, el "if(Serial.avaible())"; con este "if lo que conseguimos es que nuestro Arduino este pendiente de que escribamos en el monitor serial un valor el cual se almacenará en "setPoint". La segunda parte, simplemente, se encarga de imprimir en la pantalla oled las variables que queremos visualizar y está también dentro de un "if" para que solo se actualice con la frecuencia del tiempo de muestreo.

4.3. Tecnología utilizada

Anteriormente hemos presentado todo el hardware utilizado en el proyecto y a continuación procedemos a mostrar los softwares que han hecho posible el correcto funcionamiento de este.

4.3.1. Windows 10

Es cierto que durante todo el trabajo hemos obviado que hace falta tener al alcance un ordenador para poder programar en Arduino, pero este apartado nos saca de dudas en caso de que las hubiera. La gran ventaja de Arduino es que tiene un ambiente de desarrollo compatible con cualquier sistema operativo, y por lo tanto, no tendremos ninguna limitación en ese sentido. En nuestro caso concreto, hemos utilizado Windows 10 porque es el sistema operativo que tiene instalado nuestro equipo de serie, y por lo tanto, estamos familiarizados con su funcionamiento. Poco hay que decir sobre Windows que no se sepa ya, puesto que es sin lugar a dudas uno de los sistemas operativos más usados hoy en día. Por lo tanto, solo queda señalar que la decisión no se basa en ninguna directriz, y por lo tanto, cualquier otro sistema operativo sería completamente válido para la realización de este proyecto con éxito.

4.3.2. Arduino IDE

El entorno de desarrollo integrado (IDE) de Arduino es una aplicación diseñada en Java que se utiliza tanto para escribir como subir programas a placas Arduino o compatibles con Arduino. Como ya hemos mencionado anteriormente, esta aplicación multiplataforma en compatible con los tres grandes sistemas operativos, Windows,

macOS y Linux. Por lo tanto, cualquier persona con acceso a un ordenador no debería tener ningún problema a la hora de trabajar con ella. Otra de las grandes ventajas sin duda es que es un software libre y por lo tanto puedes descargarlo gratuitamente de la página web oficial. El IDE de Arduino te permite utilizar tanto el lenguaje C como el C++ pero utilizando reglas especiales de estructuración de códigos.

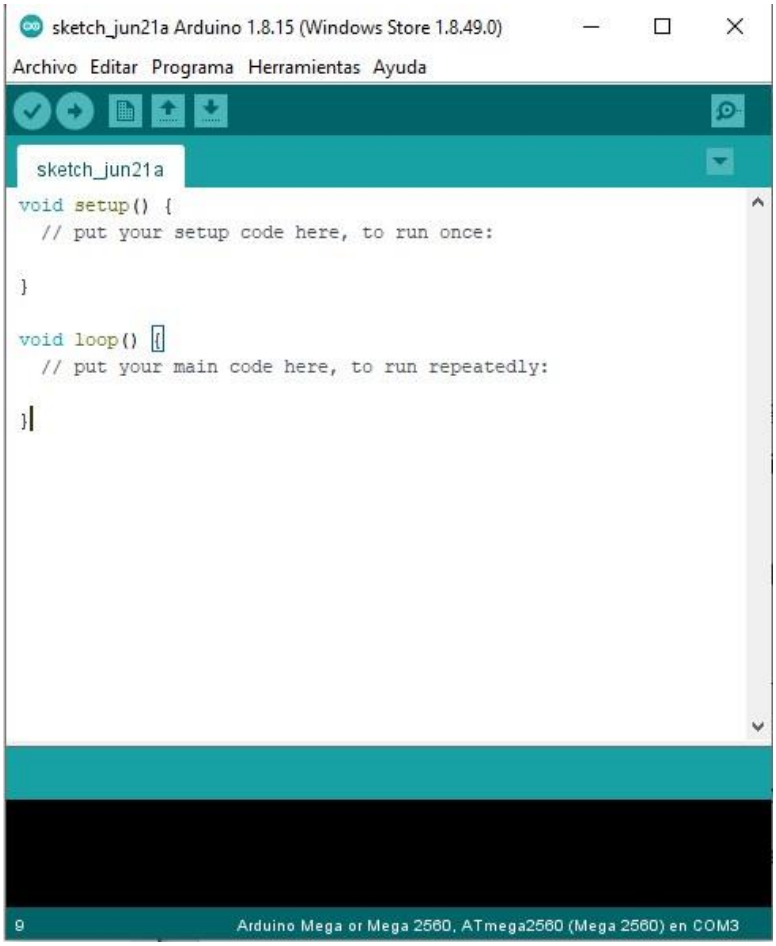


Figura 25 Arduino IDE

La figura superior nos muestra la pantalla de inicio cuando abres un nuevo archivo en Arduino IDE. Esto nos da una visión general de la estructura que suele tener un programa en este entorno. Esta estructura está compuesta por dos partes principales, la primera, es la inicialización de las variables en la parte de " void setup() " y la segunda parte es el bucle que va a repetir el microcontrolador mientras la alimentación siga conectada, siendo esta la de " void loop() ". [19]

4.3.3. Excel

Por último, necesitábamos un software que nos permitiera volcar y gestionar todos los datos recopilados desde Arduino para poder realizar los cálculos y controles pertinentes. Cuando se habla hoy en día de gestión de datos, Excel es el software más conocido. Este es posiblemente el más utilizado por los estudiantes e incluso por los profesionales. Hemos elegido utilizar Excel para la gestión de nuestros datos por varias razones:

- La primera sería por haber trabajado anteriormente con él en múltiples trabajos y estar familiarizados con su funcionamiento.
- La segunda, es que, gracias a la Universidad Politécnica de Valencia, contamos con acceso gratuito a todos los softwares académicos de Microsoft y por lo tanto debemos aprovechar esta oportunidad.
- Y la tercera, es que consideramos que Excel tiene un funcionamiento superior al resto de competidores que tenemos a nuestro alcance.

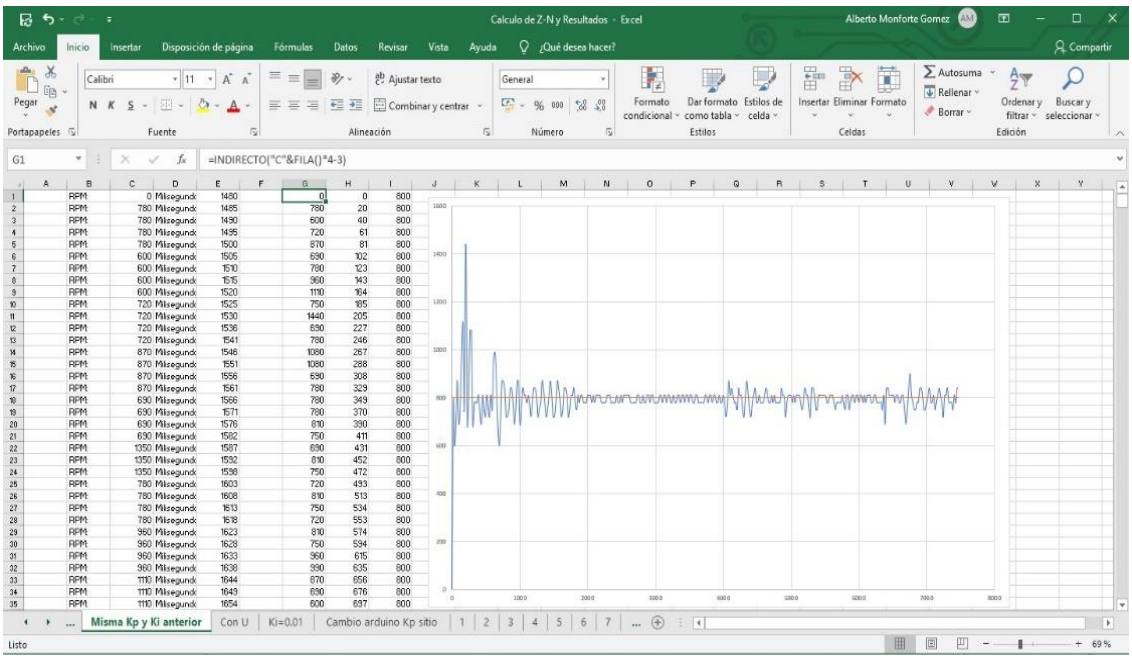


Figura 26 Pagina de Excel con una grafica

En la figura superior, podemos ver un ejemplo de volcado de datos sacados del monitor serial de Arduino y cómo podemos utilizar el Excel para dibujar una gráfica. Se puede ver de manera evidente que gracias a este software, la interpretación y gestión de nuestros resultados se convierte en una tarea mucho más sencilla.

5. DESARROLLO E IMPLEMENTACION DE LA SOLUCIÓN

Una vez vistos todos los componentes que vamos a utilizar, nos adentramos en la parte práctica, es decir, el montaje. Vamos a proceder a mostrar paso por paso la conexión de todos y cada uno de los componentes presentes en nuestro montaje. Estas etapas de montaje, hemos decidido reproducirlas con un software en el ordenador con el fin de poder mostrar de manera más clara y organizada cada una de las conexiones. En la parte final, mostraremos el montaje físico que hemos realizado de tal manera que se podrá apreciar con claridad tanto el resultado final como el proceso. Debemos destacar que en este apartado vamos a mostrar directamente el montaje de manera correcta y concisa. Durante toda esta parte se hará alusión de manera recurrente a las pruebas e intentos realizados antes de llegar a la conexión mostrada. Todas estas pruebas e intentos fallidos se podrán seguir con detalle en el siguiente punto llamado “PRUEBAS”. De esta manera, se presenta el trabajo con un funcionamiento correcto y en caso de querer profundizar más en el proceso podría consultarse el apartado de pruebas de manera simultánea.

5.1. Montaje

El primer paso con el que iniciamos el montaje es simplemente conectar a la protoboard los 5 voltios de alimentación y la masa de Arduino.

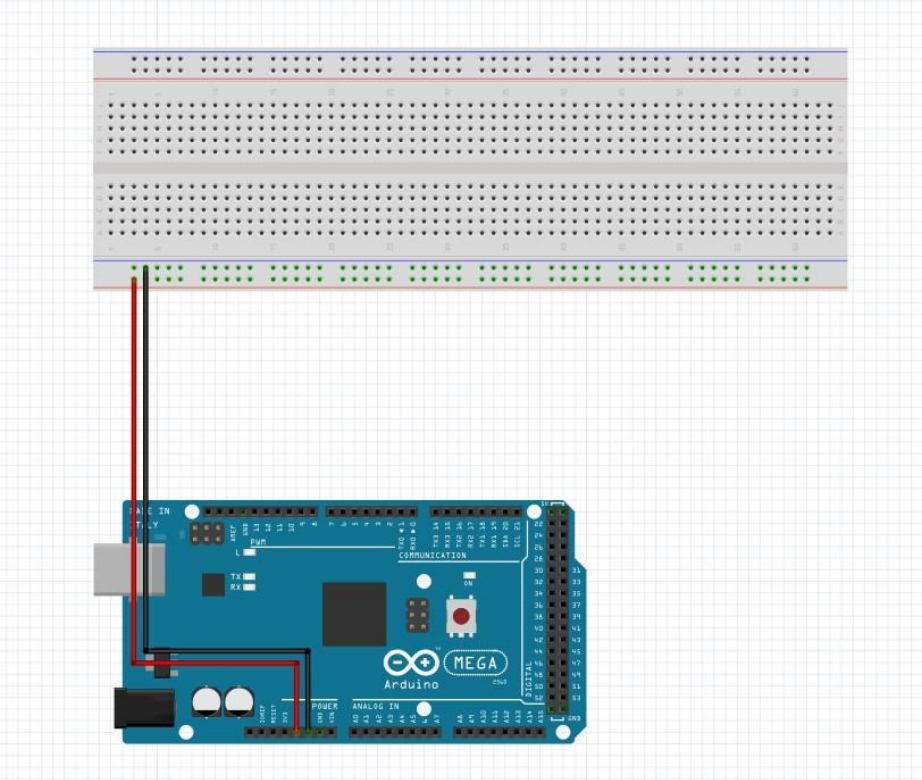


Figura 27 Conexión de alimentación de Arduino a protoboard

Esto nos facilitará la alimentación de los diferentes elementos y conseguirá que se obtenga una masa común en todo el sistema. Una vez establecida la alimentación, se procede a conectar el módulo driver L298N. Durante la fase de pruebas, observamos que el motor demandaba más corriente de la que podía suministrar una sola salida de nuestro driver, por lo tanto, se sobrecalentaba y dejaba de funcionar. La solución por la que optamos fue la de conectar las dos salidas del puente H en paralelo tal y como nos muestra su datasheet, la cual se muestra a continuación.

Figure 7 : For higher currents, outputs can be paralleled. Take care to parallel channel 1 with channel 4 and channel 2 with channel 3.

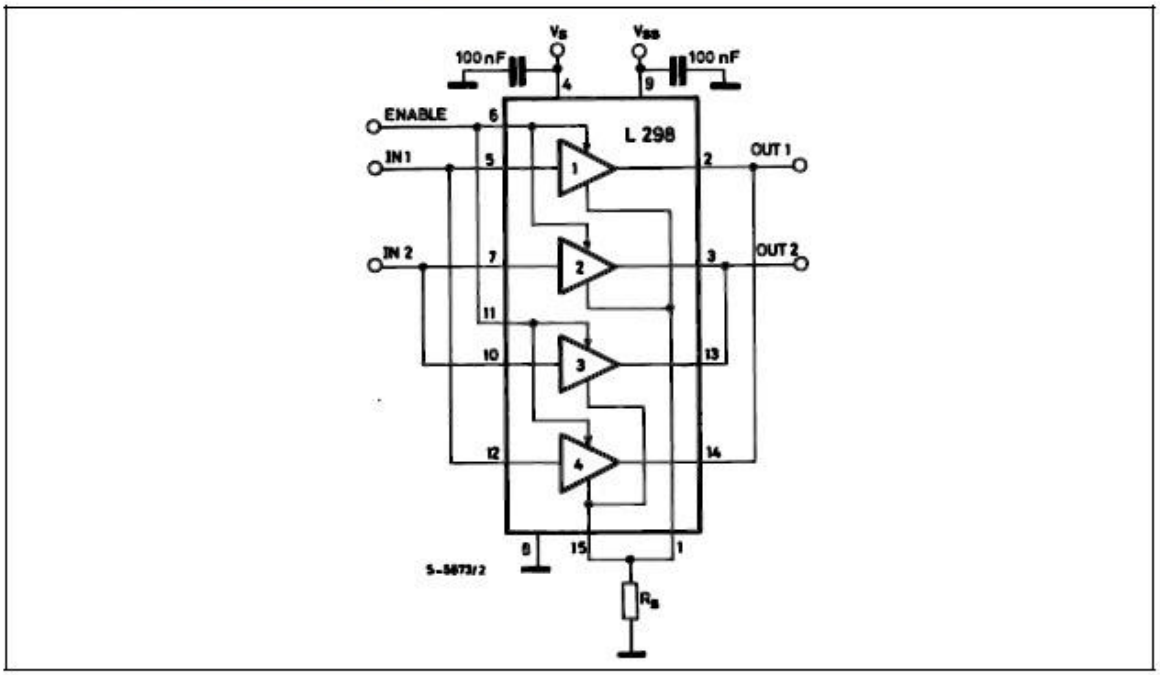


Figura 28 L298 H Bridge DATASHEET pagina 7.
Fuente: L298 H Bridge DATASHEET

Como se puede observar, este esquema nos muestra cómo se pueden conectar las diferentes entradas y salidas entre sí para adaptar el módulo a un motor con requerimientos más potentes. Para poder realizar la conexión correctamente, es necesario consultar la página 2 del datasheet en la cual se muestra el nombre de cada entrada o salida asociada a su número de conexión. Véase en la siguiente figura.

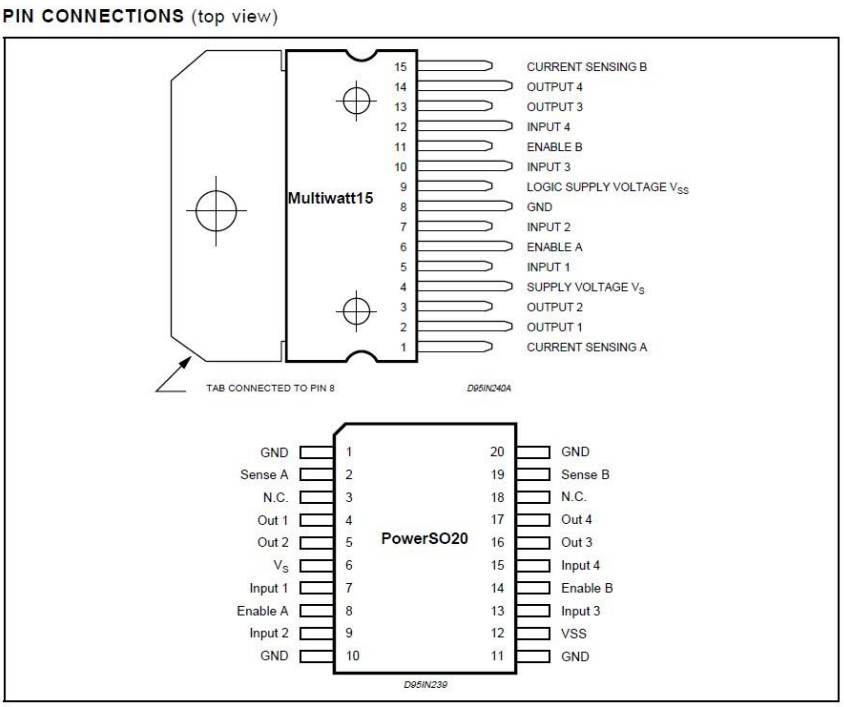


Figura 29 L298 H Bridge DATASHEET pagina 2, conexiones de pines.
Fuente: L298 H Bridge DATASHEET

Gracias a estos dos documentos tenemos todo el conocimiento necesario para realizar la conexión en paralelo correctamente. En la siguiente figura mostramos de manera esquemática como quedarían las conexiones en el módulo L298n.

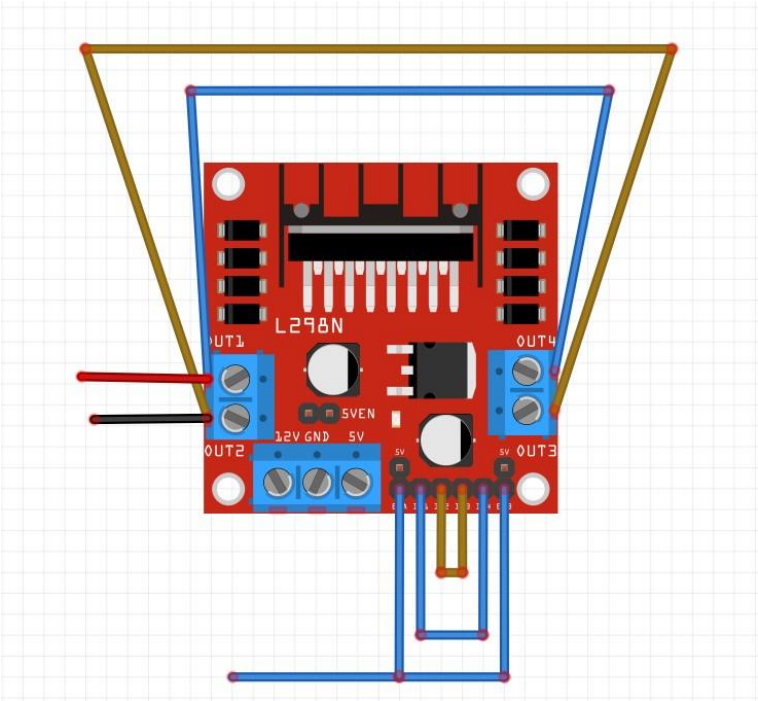


Figura 30 Modulo L298n conectado en paralelo

Una vez tenemos el módulo conectado correctamente pasamos a incorporarlo a nuestro montaje.

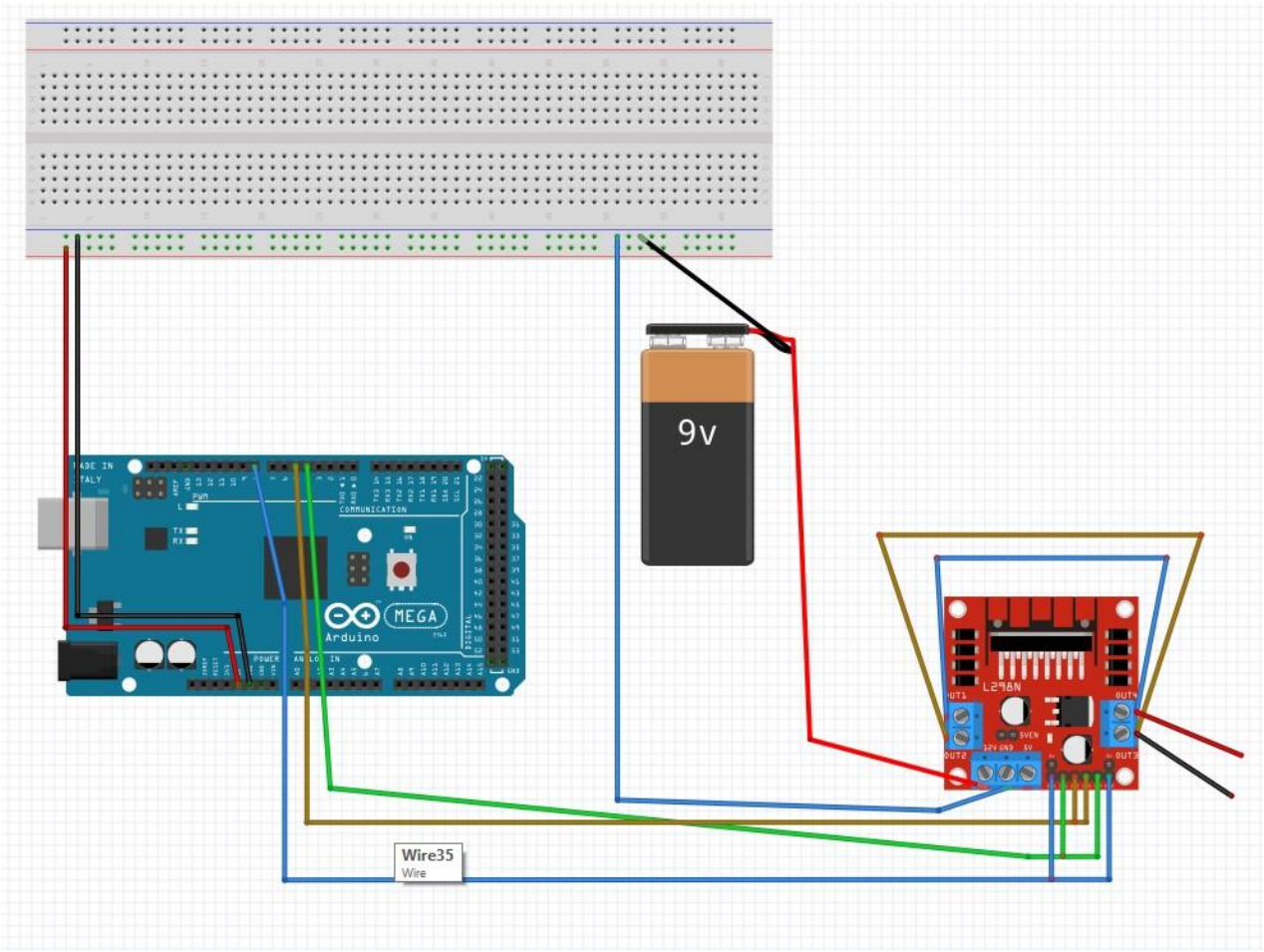


Figura 31 Montaje con Driver y Alimentacion

Tal y como se ve en la figura superior, no solo hemos aprovechado para conectar el driver, sino también la fuente de alimentación. No hemos encontrado una fuente de alimentación de 12 voltios en el software, y por lo tanto la hemos representado con una pila de 9 voltios. Podemos apreciar como los pines conectados a Arduino son los pines de entrada de nuestro módulo. Contamos con la fuente de alimentación conectada a la entrada de 12 voltios del driver y la masa puesta en común a través de la protoboard. Se puede observar que a la derecha del módulo quedan dos cables libres, uno rojo y otro negro. Estos son los que envían la corriente controlada por el driver y se encargaran de alimentar al motor. Entre el motor y nuestro driver deberemos conectar en serie el sensor de corriente por lo que este va a ser nuestro siguiente paso.

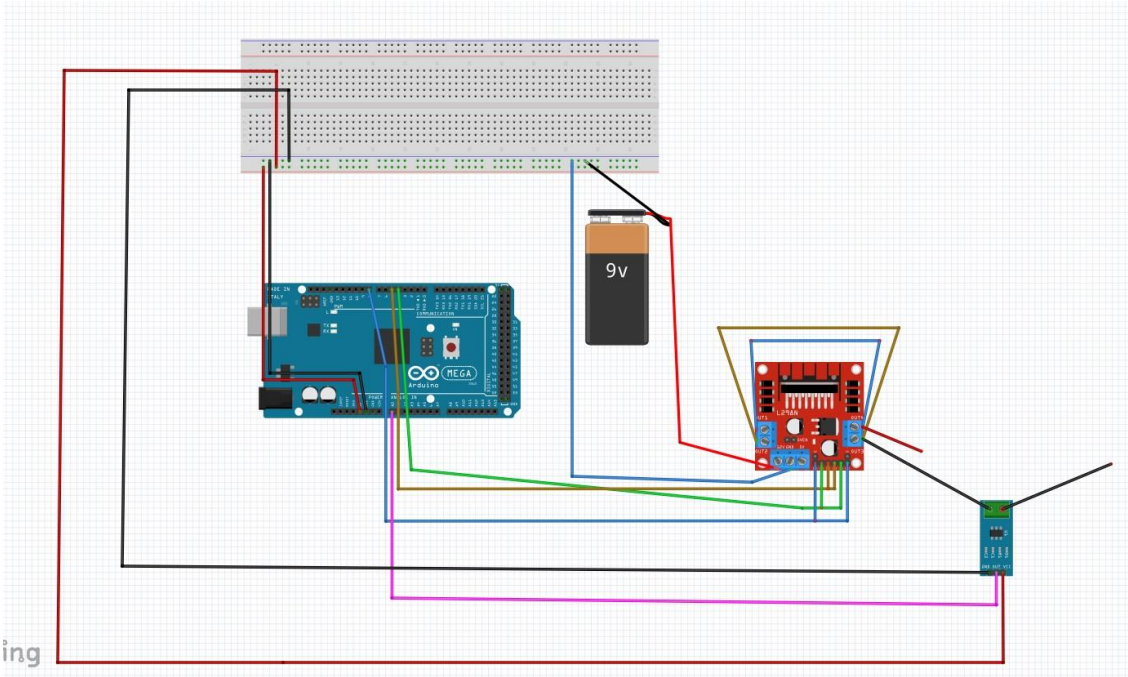


Figura 32 Montaje con el sensor de corriente incorporado

Una vez esté el sensor ACS712 incorporado al circuito y como se ha comentado previamente, lo conectaremos en serie con la salida del driver. El circuito interno del sensor se alimenta con 5 voltios desde Arduino y está conectado a la masa común. La señal recibida del sensor en analógica y la hemos conectado directamente al pin A0 de Arduino. El siguiente paso consiste en la adición de dos elementos, el motor de corriente continua y el encoder. Este es posiblemente uno de los pasos más delicados por lo que debemos prestar mucha atención a la hora de colocarlo todo correctamente.

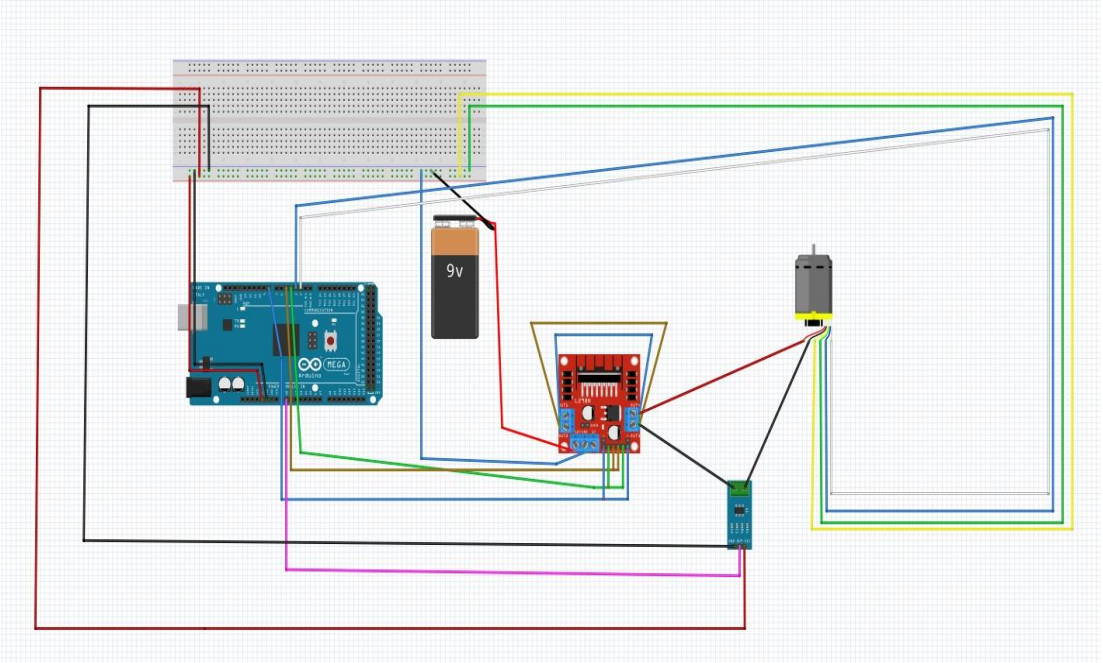


Figura 33 Montaje con motor y encoder incorporados

Una vez incorporados todos los elementos, podemos apreciar que el esquema comienza a estar bastante sobrecargado de cables y nos hacemos una idea de lo caótico que puede quedar el montaje real. Por esta razón, hemos considerado importante poder mostrar las conexiones pieza por pieza de manera esquemática.

Por último, solo queda incorporar la pantalla de adquisición de datos Oled SSD1306 tal y como se muestra en el esquema siguiente.

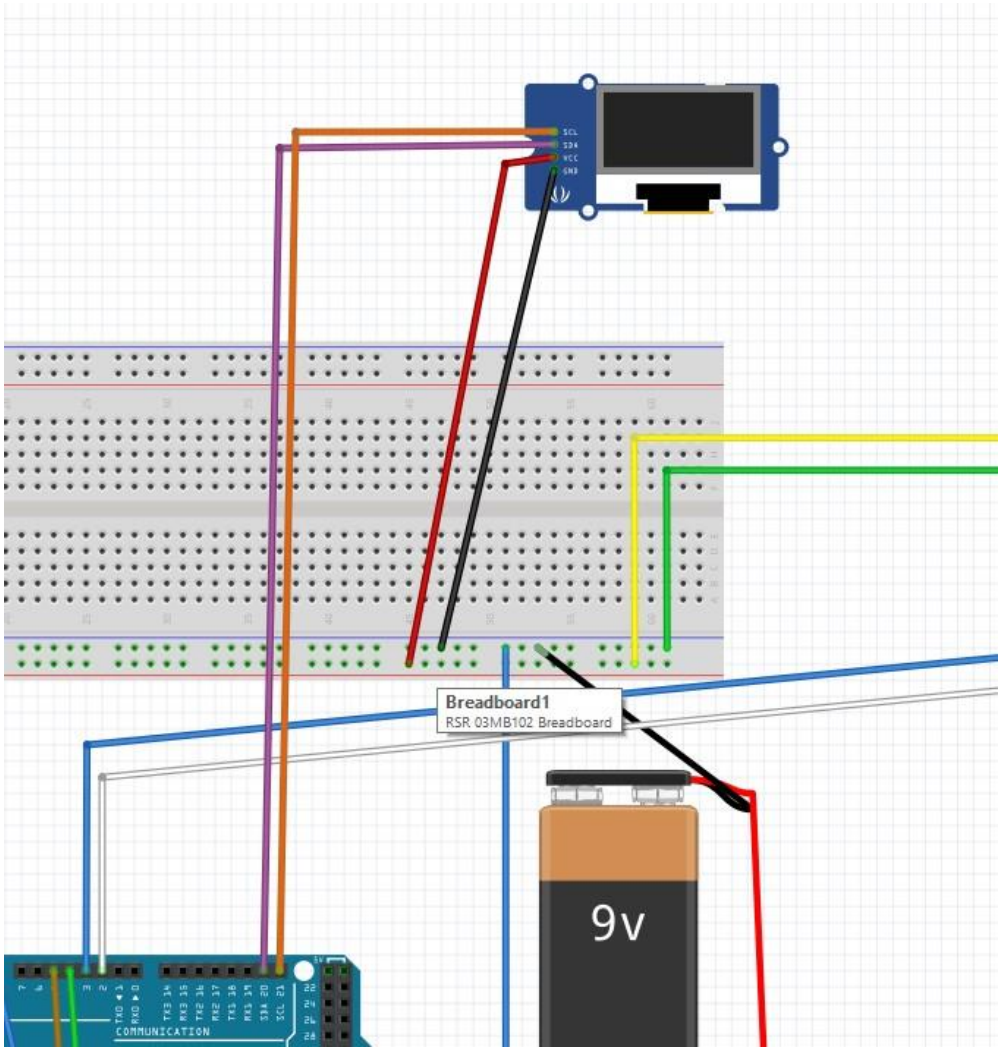


Figura 34 Vista ampliada de montaje con oled

Nos hemos decantado por una vista ampliada en este paso, ya que con la dimensión del montaje apenas se apreciaban las conexiones si se optaba por una vista global. La pantalla consta de cuatro conexiones, dos conexiones de alimentación, las cuales conectamos en la protoboard y dos conexiones de comunicación con Arduino, “SDA” y “SCL”; por suerte nuestro Arduino Mega tiene dos pines exclusivamente preparados para este tipo de conexiones el pin “20” y el “21”.

Finalmente, vamos a presentar el montaje completo esquemático y lo vamos a contrastar con nuestro montaje real.

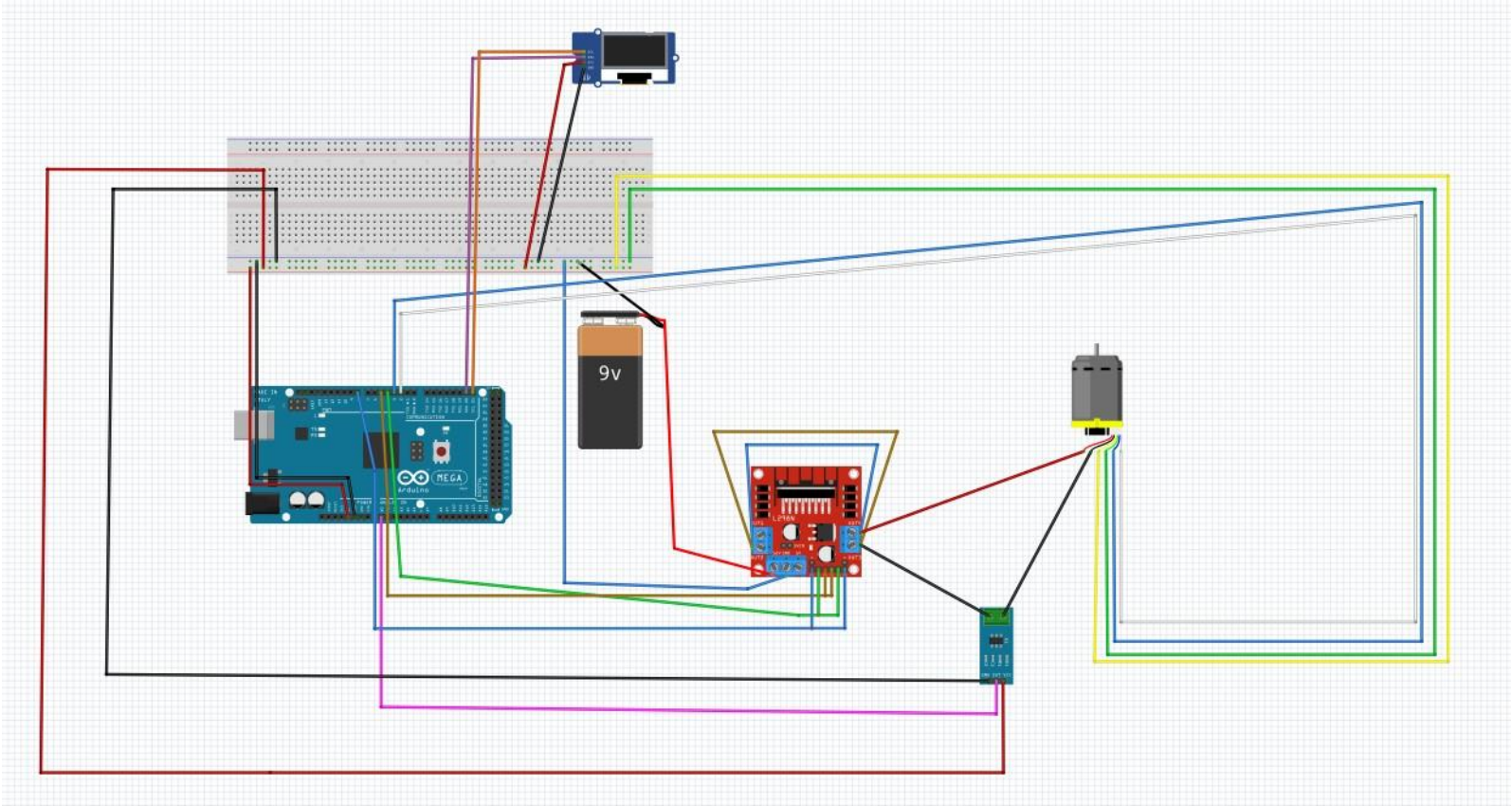


Figura 35 Montaje completo esquemático

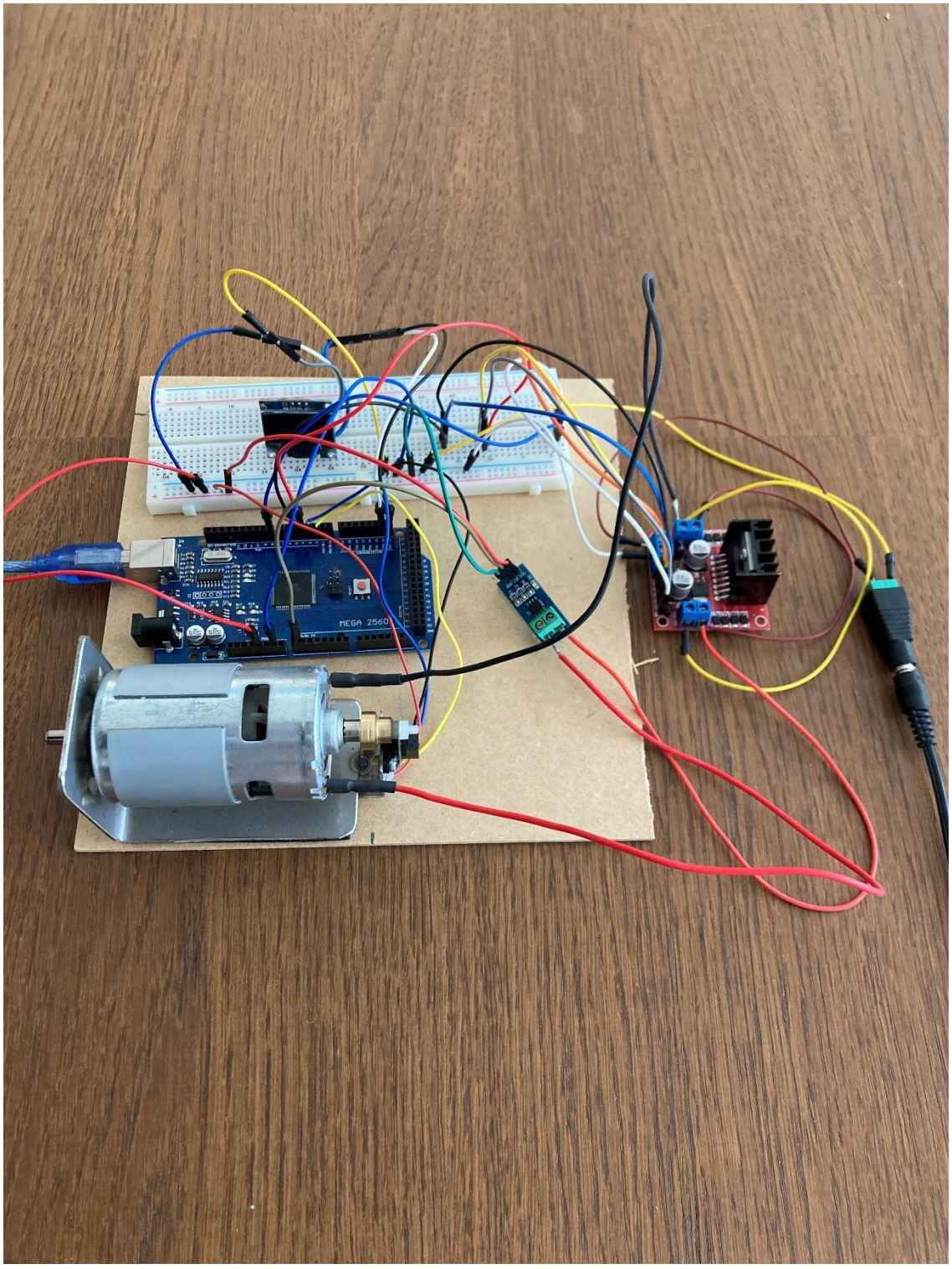


Figura 36 Montaje real definitivo

Se puede apreciar a simple vista que el montaje esquemático aporta mucha claridad y orden ya que en nuestro montaje real no nos es posible organizar los cables de manera tan visual. Pese a todo esto, podemos observar que nuestro montaje está compuesto por todos los elementos anteriormente descritos. Nos puede llamar la atención que la protoboard real tiene más conexiones que la que aparece en el esquema y esto es debido a que a la hora de conectar el driver en paralelo la única forma que teníamos de unir tanto los pines de "Input" como de "Enable" era conectándolos en la protoboard.

En definitiva, la figura muestra el montaje final de nuestro proyecto y el cual nos permite llevar a cabo con éxito el control de velocidad del motor Hanpose 775.

5.2. Método de Ziegler-Nichols

El siguiente paso que debemos realizar para desarrollar la solución es calcular el valor de nuestras constantes "Kp" y "ki" para conseguir una respuesta del PID lo más ajustada posible. El método de Ziegler-Nichols nació en 1942 y desde entonces es uno de los métodos de sintonización más difundido y utilizado. Se trata de unos cálculos realizados a través de una toma de datos de nuestro sistema. Por lo tanto, para realizar este método necesitamos poder controlar nuestro sistema de manera manual.

El procedimiento de cálculo es muy simple. Debemos poner en marcha nuestro sistema con el PID ya diseñado y programado en el microcontrolador. Vamos a realizar una serie de pruebas buscando lo que llamaremos nuestro valor "Kc" y "Tc". Para encontrar "Kc" deberemos poner nuestro "Ki" a cero e ir probando valores en "Kd" hasta que veamos que la respuesta de nuestro motor oscile de manera constante. En este paso llevamos el sistema al borde total del desequilibrio y por lo tanto hay que tener en cuenta que dependiendo de la maquinaria controlada puede resultar muy peligroso e incluso dañino para el sistema y más importante, para las personas que lo rodean. Una vez detectemos cual es la "Kp" que lleva nuestro sistema a esa inestabilidad constante, deberemos calcular el periodo "Tc" de esta e utilizar la siguiente tabla.

Tabla 3 Calculo de constante por Ziegler-Nichols

Control	Kp	Ki	Kd
P	0.50*Kc		
PI	0.45*Kc	0.54*Kc/Tc	
PD	0.80*Kc		0.075*Kc*Tc
PID	0.59*Kc	1.18*Kc/Tc	0.075*Kc*Tc

Nosotros deberemos fijarnos en la línea correspondiente a PI ya que para controlar nuestro motor nos bastará simplemente con implementar la parte proporcional e integral del PID.

Pasamos a mostrar los cálculos realizados en nuestro sistema. Después de realizar varias pruebas de las cuales se habla en su apartado correspondiente como ya mencionamos, decidimos que la inestabilidad más constante del sistema de encuentra con un "Kp" igual a 0,04.

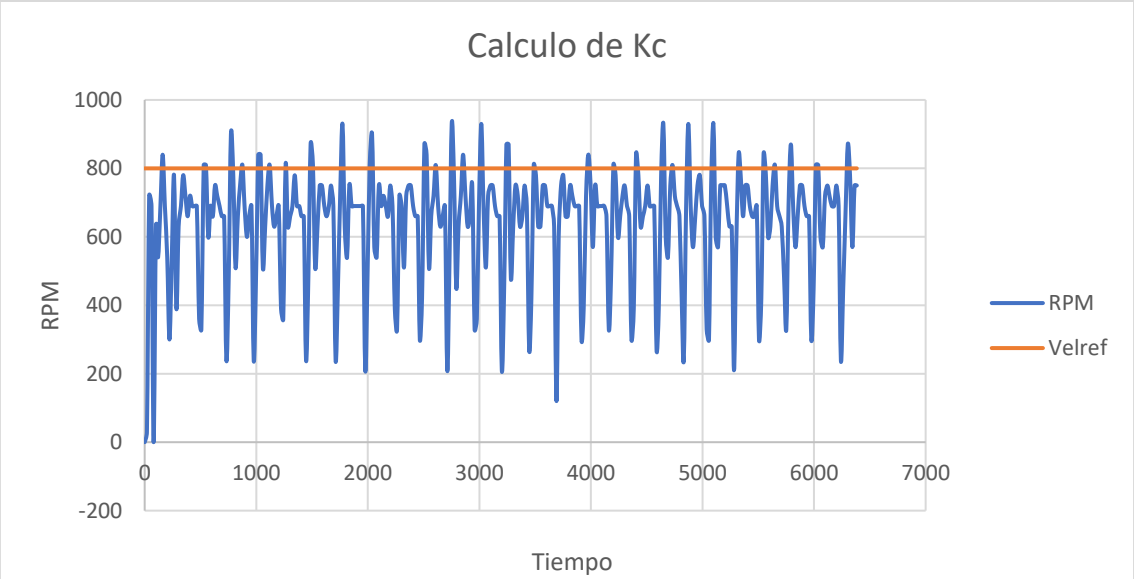


Figura 37 Toma de datos para cálculo de Kc

Por lo tanto, establecemos nuestro "Kc" igual a 0,04 y realizamos los cálculos pertinentes:

$$Kp = 0.45 * 0.04 = 0.018 \qquad (Ecuación\ 4)$$

Por otro lado, sacamos el periodo el periodo "Tc" siendo este igual a 201. Con estos datos ya podemos calcular la constante "Ki".

$$Ki = \frac{0.54*0.04}{201} = 0.000107 \qquad (Ecuación\ 5)$$

Ya tenemos el valor de nuestras constantes. Estos valores no son exactos y simplemente se usan para tener una referencia de donde partir. Por lo tanto, ajustándolos un poco se consigue mejor funcionamiento de este. Los valores finales son los siguientes:

$$Kp = 0.02 \qquad Ki = 0.0001 \qquad (Ecuación\ 6)$$

Finalmente conseguimos como resultado final el siguiente comportamiento en nuestro sistema.

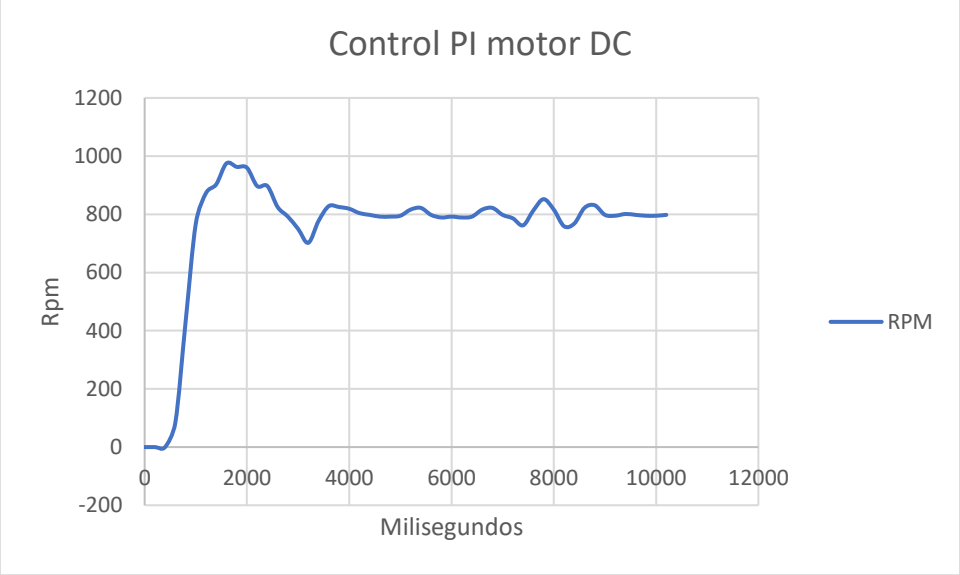


Figura 38 Control PI de nuestro motor DC

Consideramos este un buen resultado, ya que pese a tener ciertas oscilaciones momentáneas, no son extremadamente elevadas. A continuación, se muestra una gráfica en la que aparece el error en porcentaje de nuestro ajuste de revoluciones por minuto.

Figura 39 Error de control con respecto del tiempo

Constatamos que una vez se establece la velocidad, rara vez sobrepasa un error del 4% lo cual consideremos bastante asumible. Debemos añadir que durante este trabajo nos hemos dado cuenta de que el cuello de botella de nuestro montaje es el encoder. Este no tiene muy buena precisión y por lo tanto nos genera oscilaciones con el tiempo. Se señala que en la parte de pruebas profundizaremos más en los ensayos realizados con él.

6. PRUEBAS

6.1. Cálculo de Kp y Ki

Como ya hemos mencionando para calcular el valor de las constantes “Kp” y “Ki” hemos utilizado el método de Ziegler-Nichols. Decidimos que entre las diferentes tomas de datos que llevamos a cabo, la que más se aproximaba a lo que estábamos buscando era la de Kc igual a 0.04 pero procedemos a mostrar las diferentes graficas que obtuvimos para los diferentes Kc probados.

En primer lugar, probado con Kc igual a 0.1, aparece el siguiente resultado.

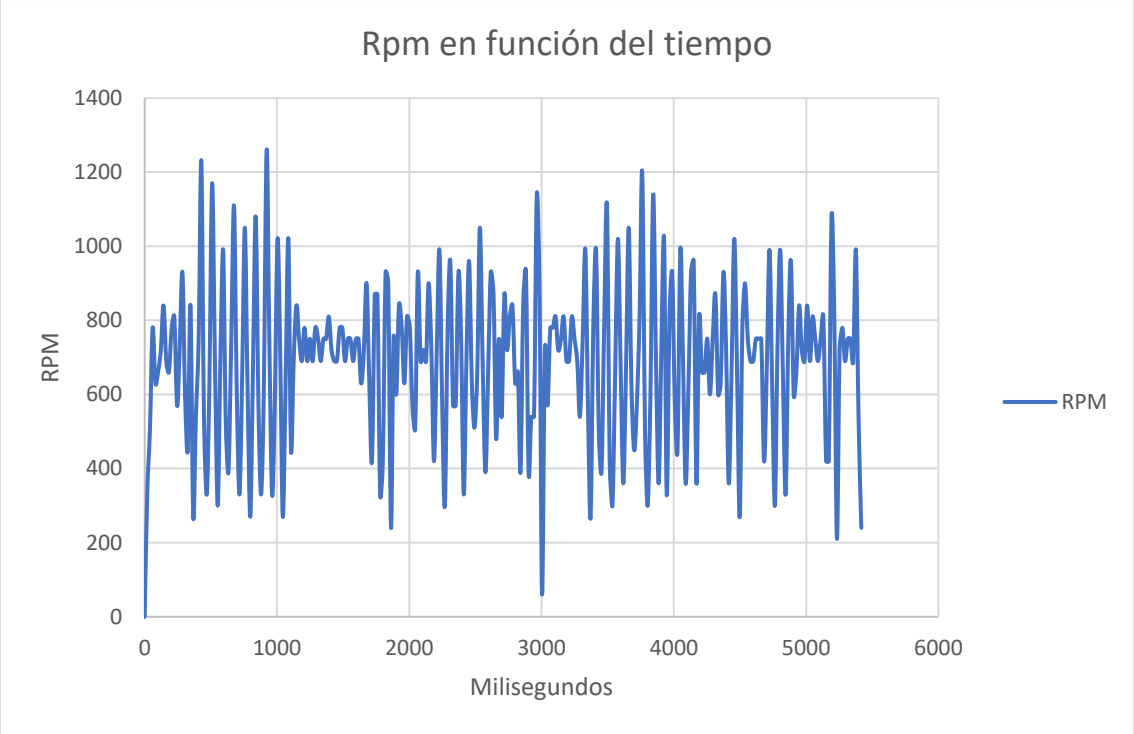


Figura 40 Toma de datos para Kc=1

Observamos una curva muy irregular con comportamientos espontáneos. Decidimos comenzar a bajar de 0.01 en 0.01 hasta conseguir un comportamiento más estable. A continuación, vamos a presentar de manera continuada las gráficas obtenidas hasta llegar a un resultado significativo.

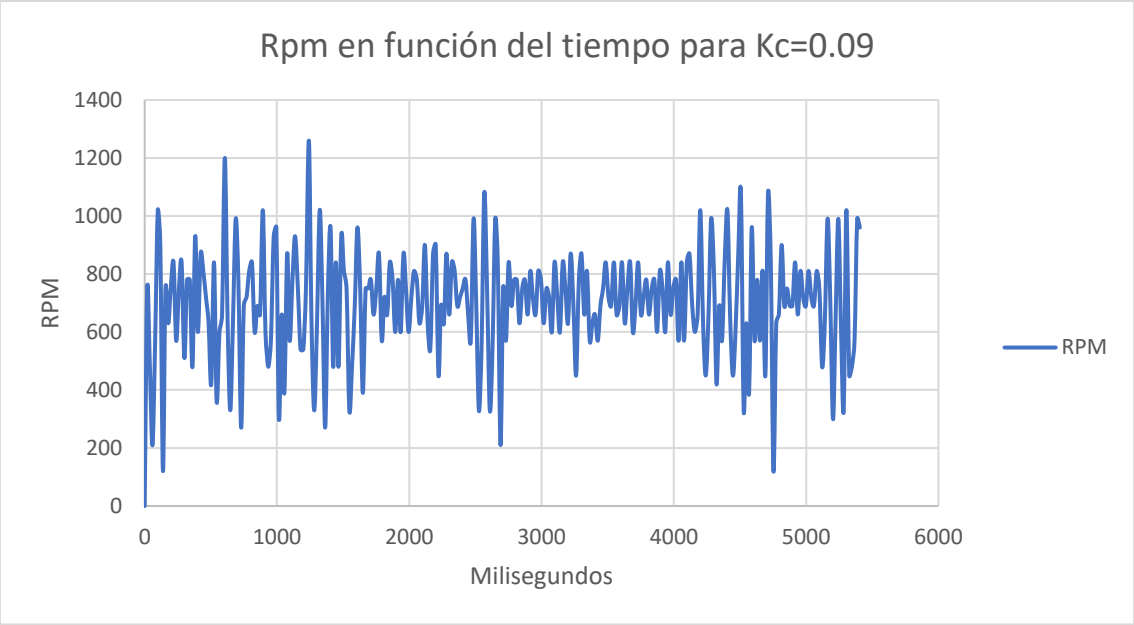


Figura 41 Toma de datos para Kc=0.09

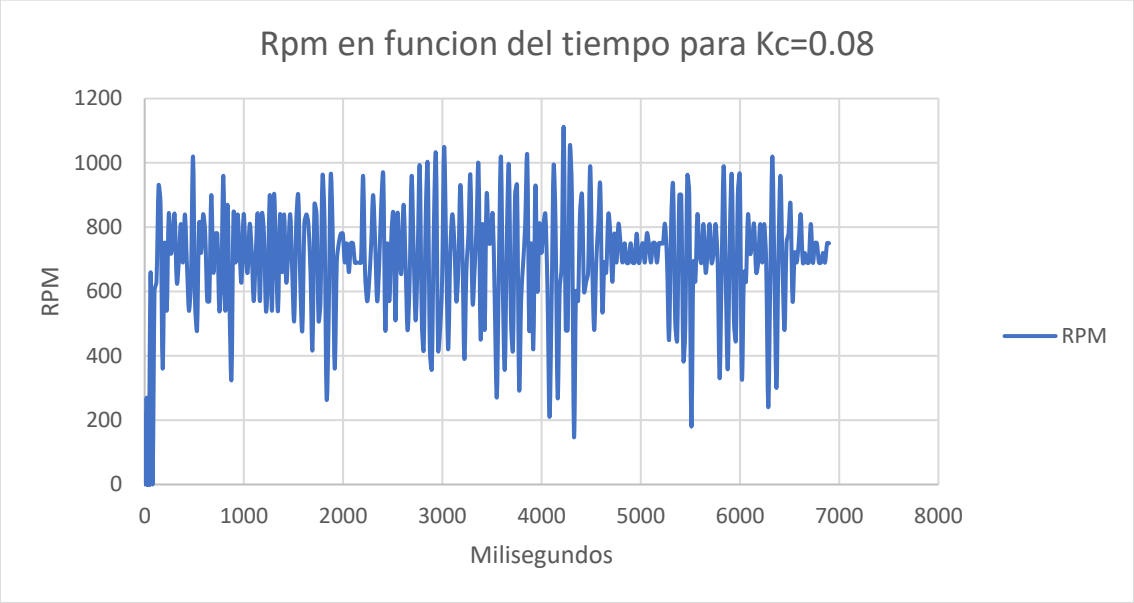


Figura 42 Toma de datos para Kc=0.08

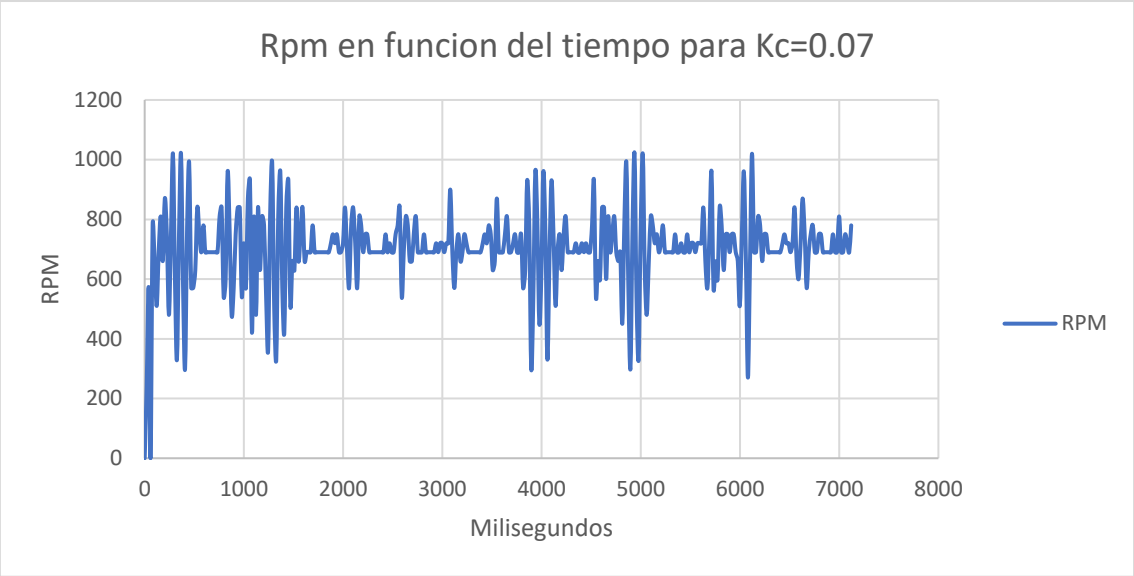


Figura 43 Toma de datos para Kc=0.07

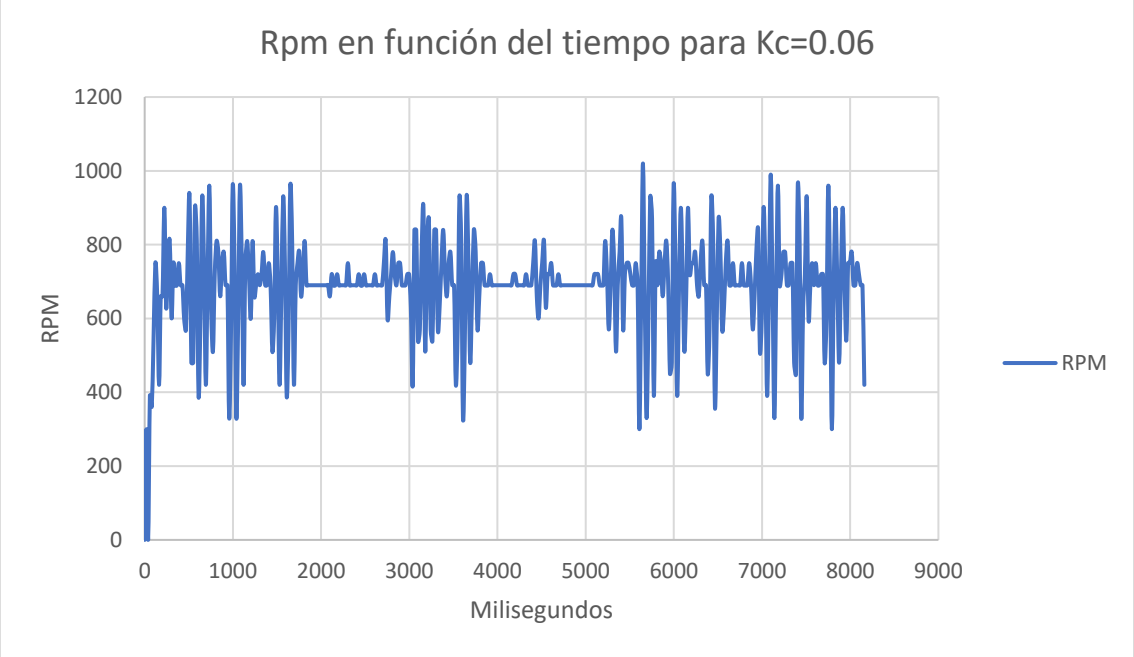


Figura 44 Toma de datos para Kc=0.06

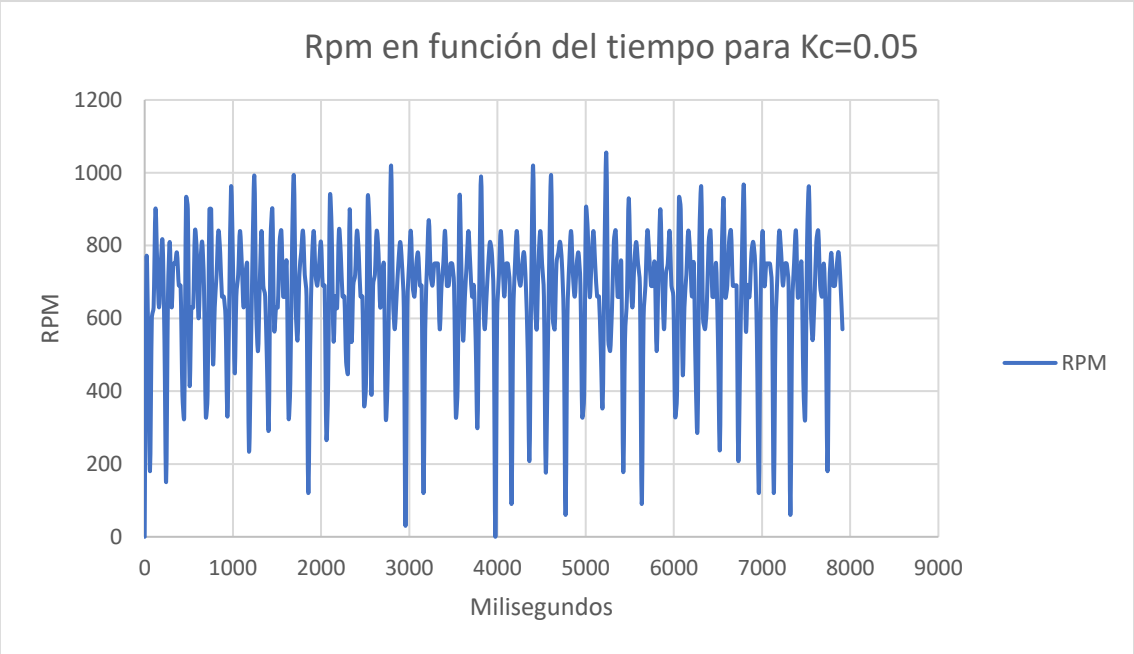


Figura 45 Toma de datos para Kc=0.05

En este último gráfico empezamos a ver algo diferente. Apreciamos que dejan de haber saltos irregulares y parece que oscila con un patrón más constante. Antes de seguir reduciendo probamos con Kc igual a 0.055 para ver su comportamiento.

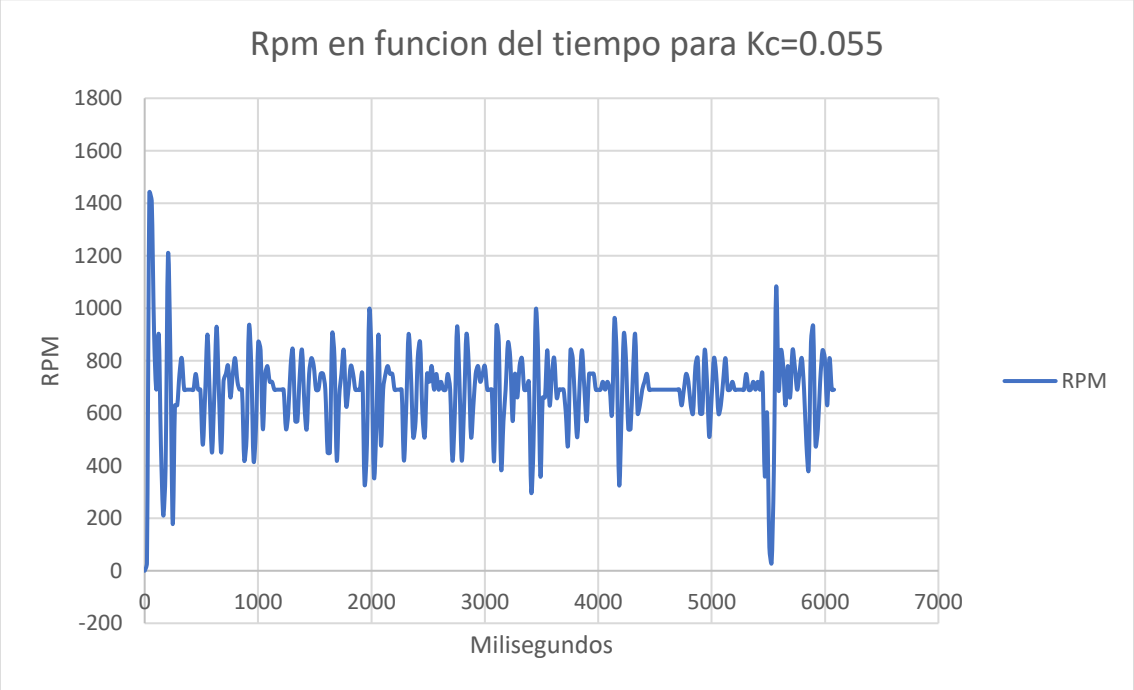


Figura 46 Toma de datos para Kc=0.055

Podemos observar que vuelven a aparecer las irregularidades por lo que decidimos seguir reduciendo el valor. Por lo tanto, pasamos a 0.04 el resultado que ya conocemos puesto que es el que hemos utilizado para la resolución final. Véase el grafico en la figura 35.

6.2. Driver L298n

Durante los primeros montajes del sistema, utilizábamos el módulo driver con la conexión estándar para solo un motor lo cual nos dejaba la mitad de este sin funcionar ya que tiene capacidad para conectar dos motores de manera simultánea. Enseguida nos dimos cuenta de que a los pocos segundos de ponerlo en marcha, el motor iba perdiendo velocidad hasta quedarse sin movimiento. En ese momento realizamos una exploración de la conexión por si algo había fallado y lo que descubrimos es que las aletas de refrigeración del L298n estaban a una temperatura muy elevada. La primera causa que nos vino a la mente fue que quizás nuestra fuente de alimentación estaba dando voltaje en exceso. Por ello probamos la combinación del driver que permite recibir hasta 32 voltios. En esta combinación simplemente debemos quitar el jumper del pin de 5 voltios y alimentar su entrada correspondiente con Arduino. El resultado obtenido fue exactamente el mismo que el anterior. Necesitábamos otra vía por la que continuar explorando. Leyendo la datasheet del módulo, descubrimos que estaba diseñado para controlar motores de hasta 2 amperios y que si queríamos aumentar ese límite existe una conexión en paralelo de ambas salidas. Esto nos llevó a pensar que seguramente ese iba a ser nuestro problema. Decidimos buscar las especificaciones de nuestro motor Hanpose 775 y efectivamente, al ser un motor diseñado para procesos que requieren un par muy elevado, nuestro motor demanda mucho amperaje, de hecho, puede trabajar hasta un máximo de 6 amperios, lo que sobrepasa en gran medida nuestro límite de 2 amperios. A raíz de estos descubrimientos, llegamos a la solución mostrada en el desarrollo. Conectamos el módulo en paralelo y con eso conseguimos solventar la limitación de corriente de nuestro driver.

6.3. Encoder

Como bien hemos comentado, el encoder es la parte más limitante de nuestro proyecto ya que no tiene un comportamiento muy preciso. Esto ha derivado en uno de los principales problemas que hemos tenido a la hora de ajustar nuestro PID. Inicialmente, nuestro controlador tenía un tiempo de muestreo de 20 milisegundos. Nos pusimos a trabajar con ese intervalo y por mucho que lo intentáramos no conseguíamos un comportamiento estable. Véase en las siguientes figuras.

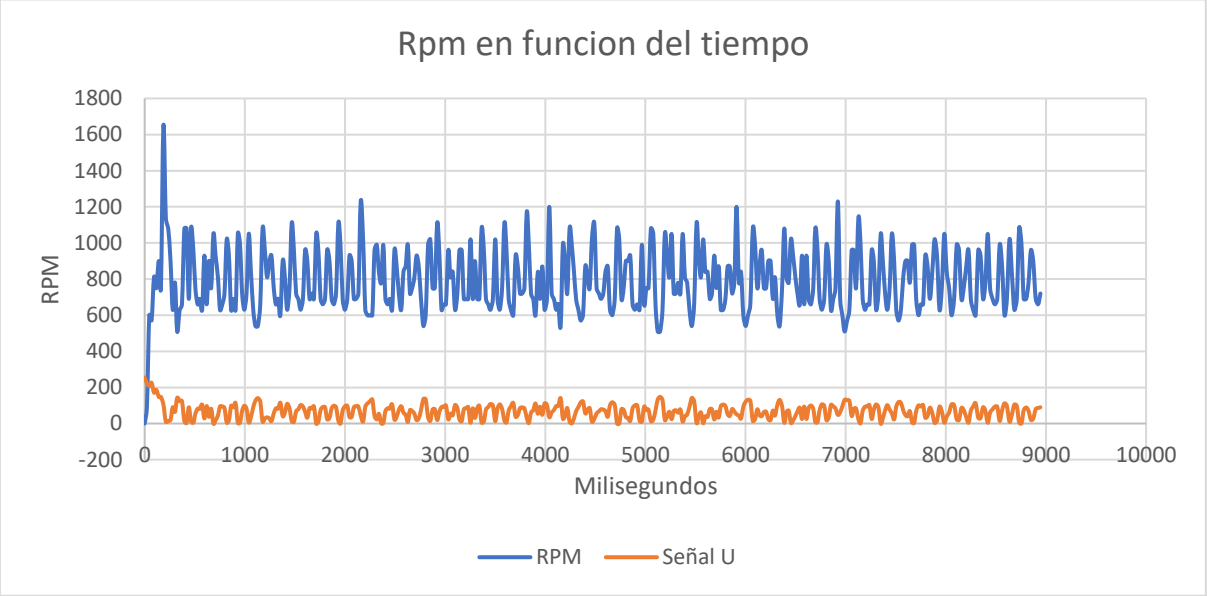


Figura 47 Prueba 1 PID

Tal y como se ve en la figura anterior decidimos monitorizar también la señal de control "U" para intentar descubrir de dónde venía el error de precisión. Hicimos varias pruebas más como las que se pueden ver a continuación, intentando ajustar los valores de "Kp" y "Ki".

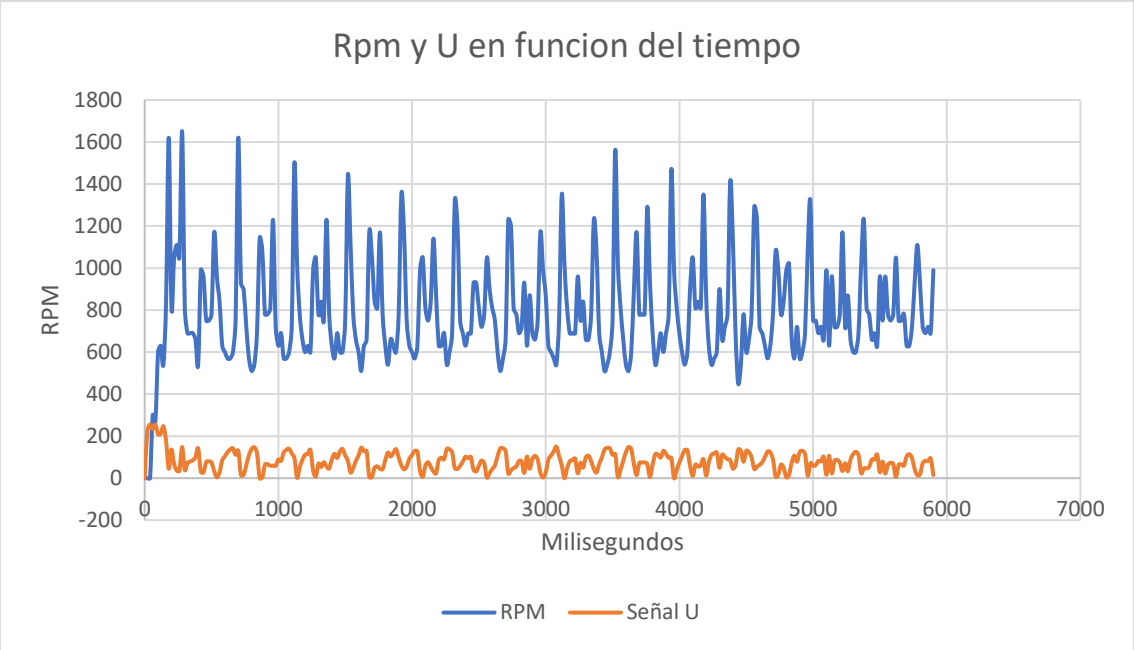


Figura 48 Prueba 2 PID

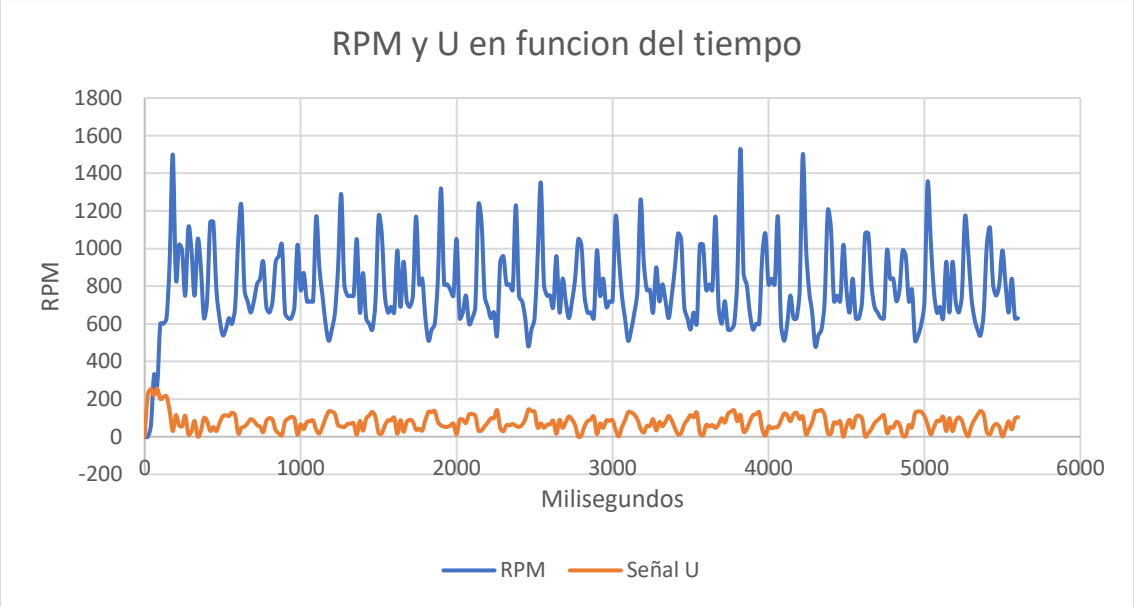


Figura 49 Prueba 3 PID

Tras un gran número de pruebas con los mismos resultados que los mostrados anteriormente, empezamos a pensar que debía haber algún error o algún elemento limitándonos. Revisamos minuciosamente el diseño de nuestro PID y no encontramos ningún fallo en el código y por lo tanto se nos ocurrió realizar unas pruebas de precisión

con el encoder. Nos dimos cuenta que pese a mandar una señal constante al motor, si mediamos sus revoluciones por minuto con un intervalo de 20 milisegundos el resultado era bastante dispar. Este error de medida estaría causando seguramente gran parte de la inestabilidad de nuestro PID. Fuimos subiendo poco a poco el tiempo de muestreo hasta que conseguimos una lectura de constante. Esta se consiguió con 200 milisegundos, y por lo tanto, tuvimos que establecer ese como intervalo de nuestro sistema. Somos conscientes de que este intervalo es un poco alto para que el sistema tenga una respuesta con la velocidad y precisión deseada, sobre todo en procesos que requieran un tiempo de reacción muy rápido. No obstante, puesto que nuestro objetivo principal es utilizarlo en una máquina de control numérico, no necesitamos una respuesta extremadamente rápida. Por lo tanto, asumimos los 200 milisegundos como un tiempo de muestreo aceptable en nuestras condiciones.

El segundo problema que tuvimos también está relacionado con el encoder, pero esta vez, relacionado con su geometría. Resulta que los dos sensores no están solo desfasados 90 grados entre sí, sino que se encuentran a diferentes alturas. Uno de ellos se encuentra a mayor profundidad que el otro. Lamentablemente, a la hora de fijar el encoder en el soporte, comprobamos que el sensor colocado más cerca de la superficie leyera correctamente pero el segundo sensor queda fuera de alcance y no lee de manera correcta. Es cierto que esto resulta un problema menor en nuestro proyecto, ya que como hemos mencionado, para su uso en máquinas de control numérico no consideramos imprescindible poder determinar su sentido de giro. Por esa razón y por haber sellado los cables al soporte con un termoplástico por miedo a que se tocaran y cortocircuitaran, hemos preferido funcionar con un solo sensor y evitar el riesgo de dañar algún cable desmontándolo. El aspecto positivo de esto, es que dejamos recursos todavía por explotar en el caso de que quisiéramos realizar algún trabajo futuro o darle alguna otra utilidad.

7. CONCLUSIONES

Una vez expuesto todo el material y los resultados obtenidos en el trabajo, podemos dar paso a las conclusiones. En primer lugar, estamos bastante satisfechos con el resultado final. Hemos conseguido controlar la velocidad de manera estable, manteniéndonos por debajo de un error del 5%. Pese a haber tenido varios problemas y limitaciones con el material inicial, hemos conseguido profundizar es su funcionamiento y adaptarlo para que cubra nuestras necesidades. Tenemos un código de Arduino muy sólido, el cual sin duda alguna podría ser utilizado en muchos otros proyectos de control PID. Hemos ideado un diseño ergonómico de nuestro soporte del motor y un montaje compacto y funcional. Hemos puesto en práctica un gran número de habilidades y conocimientos adquiridos en la carrera como el montaje de circuitos eléctricos, la programación, el modelado de piezas en Inventor, la gestión de datos con Excel, así como el cálculo de presupuestos y realización de un proyecto. A parte de llevar todo esto a la práctica, hemos tenido la suerte de seguir aprendiendo durante todo este proceso ya que nos hemos enfrentado por primera vez a un microcontrolador como es el Arduino, hemos aprendido poco a poco a dominarlo y sacarle un buen rendimiento. Estimamos de gran utilidad la experiencia que hemos adquirido enfrentándonos por primera vez a un trabajo académico de este calibre y lo consideramos sin lugar a dudas una aportación favorable.

Por otro lado, somos conscientes de que hay ciertos puntos en nuestro montaje que podrían mejorarse. Durante la elaboración del trabajo hemos ido detectando y comentando las partes que nos limitaban. Como bien hemos comentado anteriormente, la mayor parte de ella las hemos conseguido solucionar, en especial, las que afectaban al correcto funcionamiento del sistema. En este proceso, hemos detectado ciertos cuellos de botella que nos impiden llevar este trabajo a un nivel superior. En primer lugar, como ya mencionamos tenemos el encoder. Esta una pieza extremadamente importante para la precisión y fiabilidad de nuestro proyecto por lo que debería ser la primera inversión que se hiciera para mejorar el rendimiento. Por último, sabemos que nuestro motor puede trabajar hasta 6 amperios con carga, y nuestra fuente de alimentación suministra hasta 2,5 amperios. Como en nuestro prototipo el motor

trabaja sin carga no es determinante esa falta de corriente, pero, si se le quisiera dar una aplicación real habría que modificar esa pieza con el fin de poder suministrar más amperaje.

En definitiva, repetimos que el resultado de nuestro proyecto es más que satisfactorio. Hemos extraído el máximo partido a nuestro material y cumplido todos nuestros objetivos con éxito. Además de todo esto, consideramos un punto positivo el haber detectado las partes más limitantes de nuestro montaje, ya que el en caso de que quisiéramos mejorarlo o incorporarlo en algún proyecto sabríamos directamente que vías deberíamos seguir con el fin de mejorar su rendimiento. Además, el trabajo contempla todos los puntos y da todas las pautas necesarias para replicarlo en su totalidad.

8. PRESUPUESTO

MATERIAL				
DESCRIPCIÓN	**UNIDAD**	**PRECIO UNITARIOR**	**CANTIDAD**	**IMPORTE**
Motor Hanpose 775	€	13,99 €	1	13,99 €
Encoder de Cuadratura	€	2,14 €	1	2,14 €
Chapa metálica para soporte Motor	€	16,14 €/m2	0,00705m2	0,11 €
Driver L298n	€	1,69 €	1	1,69 €
Pantalla Oled SSD1306	€	1,42 €	1	1,42 €
Sensor de corriente ACS712	€	0,89 €	1	0,89 €
Protoboard	€	1,14 €	1	1,14 €
Ardunio Mega 2560	€	11,93 €	1	11,93 €
Fuente de alimentación CCDSTAR	€	5,90 €	1	5,90 €
Cables	€	1,00 €	1	1,00 €
			TOTAL	**40,21 €**

IINVESTIGACIÓN				
DESCRIPCIÓN	**UNIDAD**	**PRECIO UNITARIOR**	**MEDICIÓN**	**IMPORTE**
Aprendizaje de Arduino	€/h	20,00 €	12 días(6h)	2.040,00 €
Estudio de los diferentes componentes del montaje	€/h	20,00 €	7 días(6h)	840,00 €
Estudio sobre PID	€/h	20,00 €	5 días(6h)	600,00 €
Estudio sobre PID implementado en Arduino	€/h	20,00 €	7 días(6h)	840,00 €
			TOTAL	**4.320,00 €**

IMPLEMENTACIÓN Y MONTAJE				
DESCRIPCIÓN	**UNIDAD**	**PRECIO UNITARIOR**	**MEDICIÓN**	**IMPORTE**
CODIGO DE ARDUINO	€/h	20,00 €	9 días (6h)	1.080,00 €
MONTAJE	€/h	20,00 €	7 días (6h)	840,00 €
CALIBRACIÓN DE RESULTADOS	€/h	20,00 €	5 días(6h)	600,00 €
EDICIÓN DE TEXTO	€/h	10,00 €	11 días (6h)	660,00 €
			TOTAL	**3.180,00 €**

PRESUPUESTO TOTAL	
DESCRIPCIÓN	**IMPORTE**
MATERIAL	40,21 €
INVESTIGACIÓN	4.320,00 €
IMPLEMENTACIÓN Y MONTAJE	3.180,00 €
PEM	7.540,21 €
IVA (21%)	1.583,44 €
PRESUPUESTO BASE DE LICITACIÓN	9.123,66 €

9. BIBLIOGRAFÍA

[1] NOVUS, «novusautomation,» [En línea]. Available: https://www.novusautomation.com/site/default.asp?Idioma=34&TroncoID=053663&SecaoID=0&SubsecaoID=0&Template=../artigosnoticias/user_exibir.asp&ID=638091#.

[2] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Arduino.

[3] Xataka, «Xataka,» [En línea]. Available: https://www.google.com/search?q=arduino+imagenes&rlz=1C1CHBD_esES947ES947&sxsrf=ALeKk03vWuzENrJVuey4eCijF9Gb0HA-2A:1622535734062&tbm=isch&source=iu&ictx=1&fir=0wlHqbbX3kH9NM%252Cl-9qb4XiBdV8tM%252C_&vet=1&usg=AI4_-kTuVVQ3btID5yeYz7C4M1QJ1xDXKQ&sa=X&ved=2a.

[4] technoreeze. [En línea]. Available: http://www.technoreeze.com/es/2012/01/25/como-hacer-un-encoder-optico-usando-partes-de-un-mouse/.

[5] aprendiendoarduino, «aprendiendoarduino,» [En línea]. Available: https://aprendiendoarduino.wordpress.com/tag/motores/.

[6] educaciontrespuntocero, «educaciontrespuntocreo,» [En línea]. Available: https://www.educaciontrespuntocero.com/noticias/raspberry-pi-educacion/.

[17] aliexpress, «aliexpress,» [En línea]. Available: https://es.aliexpress.com/item/32848024192.html?spm=a2g0o.productlist.0.0.51691189uh5HbT&algo_pvid=8c3a96a5-34a4-4eda-879c-359e0554de28&algo_expid=8c3a96a5-34a4-4eda-879c-359e0554de28-2&btsid=2100bb4716170998256044917e987d&ws_ab_test=searchweb0_0,searchwe.

[18] DYNAPAR, «dynapar.com,» [En línea]. Available: https://www.dynapar.com/Technology/Encoder_Basics/Quadrature_Encoder/.

[19] amazon, «amazon,» [En línea]. Available: https://www.amazon.es/HANPOSE-torsi%C3%B3n-grande-potencia-rodamiento/dp/B07RTS3XJK.

[10] banggood, «banggood,» [En línea]. Available: https://nz.banggood.com/HANPOSE-775-Motor-DC-12V-24V-80W-150W-288W-DC-Motor-Large-Torque-High-Power-DC-Motor-Double-Ball-Bearing-Spindle-Motor-p-1441968.html?cur_warehouse=CN&ID=585307&rmmds=buy.

[11] Aliexpress, «Aliexpress,» [En línea]. Available: https://es.aliexpress.com/item/32793575014.html.

[12] electronilab, «electronilab,» [En línea]. Available: https://electronilab.co/tutoriales/tutorial-de-uso-driver-dual-l298n-para-motores-dc-y-paso-a-paso-con-arduino/.

[13] C. Veloso, «electrontools,» 6 mayo 2016. [En línea]. Available: https://www.electrontools.com/Home/WP/puente-h-con-driver-l298/.

[14] naylampmechatronics, «naylampmechatronics,» [En línea]. Available: https://naylampmechatronics.com/blog/48_tutorial-sensor-de-corriente-acs712.html.

[15] L. d. V. Hernández, «programarfacil.com,» [En línea]. Available: https://programarfacil.com/blog/arduino-blog/ssd1306-pantalla-oled-con-arduino/.

[16] 330ohms, «330ohms,» [En línea]. Available: https://www.330ohms.com/products/protoboard-grande.

[17] J. G. Carmenate, «programarfacil.com,» [En línea]. Available: https://programarfacil.com/blog/arduino-blog/arduino-mega-2560/.

[18] M. T. Díaz, «Fisicotrónica,» [En línea]. Available: http://fisicotronica.com/discretizacion-el-salto-cualitativo/.

[19] wikipedia, «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Arduino_IDE.

10. ANEXO

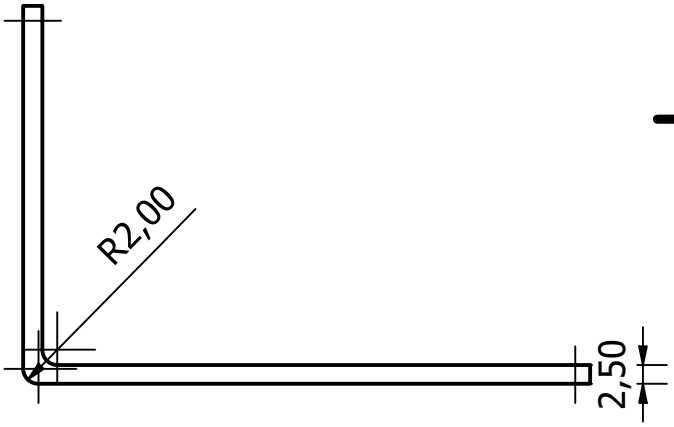
10.1. Código PID para Arduino

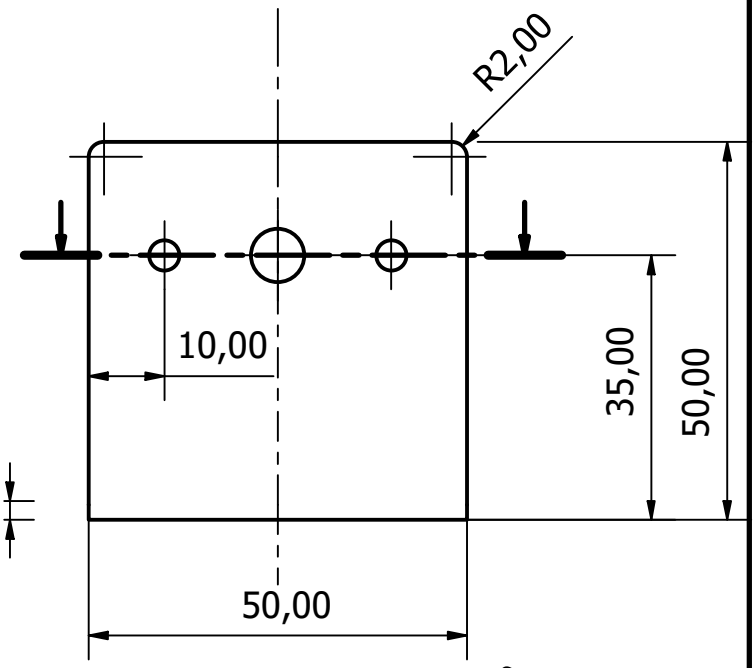
```
#include <TimerOne.h>
#include <Adafruit_SSD1306.h>
#include <Wire.h>

#define EncoderA 2
#define EncoderB 7
#define ANCHO 128
#define ALTO 64
#define OLED_RESET 4

Adafruit_SSD1306 oled(ANCHO, ALTO, &Wire, OLED_RESET);

int IN3 = 5;     // Input3 conectada al pin 5
int IN4 = 4;     // Input4 conectada al pin 4
int ENB = 8;     // ENB conectada al pin 8 de Arduino
float voltageSensor; // voltaje leído por el sensor ACS712
float corriente;     // Corriente consumida por el motor
volatile long int cont=0;  //Cuenta interrupciones del
encoder
volatile long int rpm=0; //Convierte las interrupciones del
encoder en rpm
volatile long int setPoint=0; //Las rpm que le pediremos
fijar
long int Cm=0; //Para muestrear cada 200 milisegundos
volatile long int U=0; //Valor de la señal PID
volatile float P=0;    //Parte proporcional
volatile long int error=0;  //error
volatile float I=0;// Parte integral
long int T=200; // 200 milisegundos
float Kp=0.02;  //Constante de la parte proporcional P del
PI
float Ki=0.0001;  // Constante de la parte proporcional I
del PI

void setup()
{
 pinMode (ENB, OUTPUT);
 pinMode (IN3, OUTPUT);
 pinMode (IN4, OUTPUT);
  pinMode (EncoderA, INPUT);
 pinMode (EncoderB, INPUT);
 Wire.begin();
 oled.begin(SSD1306_SWITCHCAPVCC, 0x3C);
 Serial.begin(57600);
```

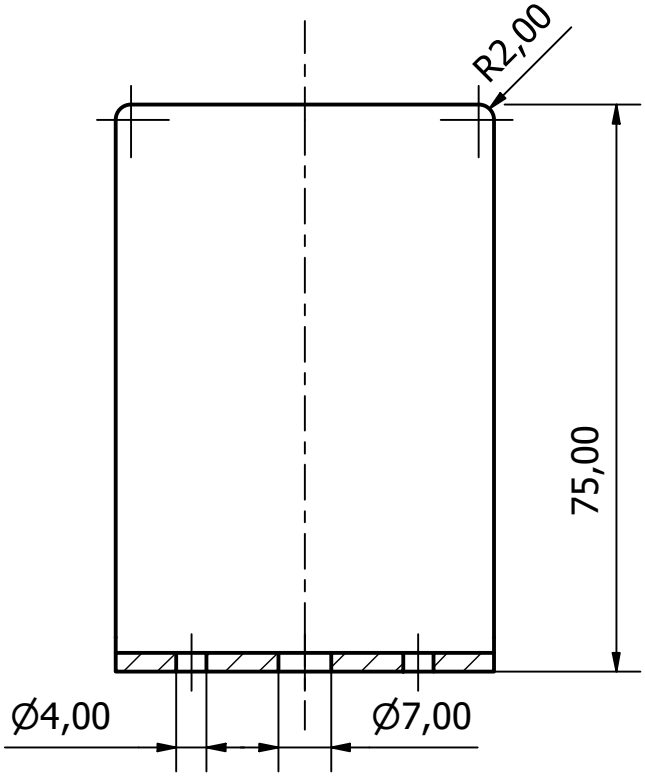
```
 Timer1.initialize(T*1000); // Esta en microsegundos
 Timer1.attachInterrupt(PID); //Una interrupcion
condicionada por el tiempo para calcular las rpm
 attachInterrupt(0,rutina,FALLING);  //Interrupcion del
encoder inicializada
}
void loop()
{

 if(Serial.available() != 0){     //Para escribir por el
monitor serial, verifica si envias algo
      String str = Serial.readStringUntil('\n');  //Para
leer un numero de más de una cifra hay que leerlo como
string
       setPoint = str.toInt();      //Convertimos la string
en un entero para poder leerlo más addelante en el
analogWrite


 }

if(millis()-Cm>=T){
  Cm=millis();
  voltageSensor = analogRead(A0)*5.0/1023.0;
  corriente= (voltageSensor - 2.5)/0.185;
  oled.clearDisplay();
  oled.setTextColor(WHITE);
  oled.setCursor(0,0);
  oled.setTextSize(2);
  oled.print (rpm);
  oled.print(" rpm");
  oled.setCursor(0,40);
  oled.setTextSize(2);
  oled.print(corriente);
  oled.print(" A");
  oled.display();
}




}

void rutina(){ // Interrupcion del encoder, cuenta las
veces que lee un aspa
  if(digitalRead(EncoderB)== 0){
    cont--;
  }
  else{
    cont++;
```

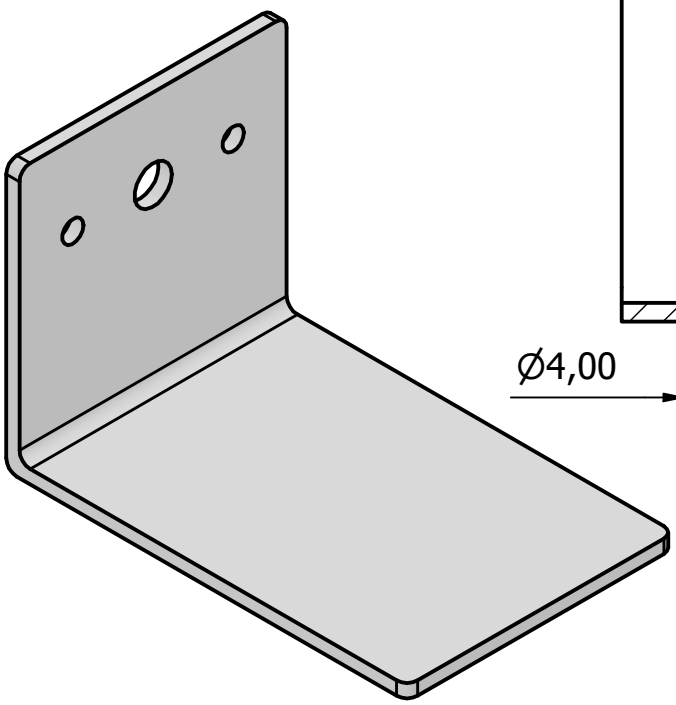
```
  }

 }



 void PID(){     //Funcion de control PID
  rpm=((60.0*cont)/(T/1000.0))/100;
  cont=0;
       //Empezar calculos PI
  error=setPoint - rpm;
  P=Kp*error; //parte proporcional
  I=(error*T*Ki+I);  // parte integral, (T es el tiempo de
muestreo)
  U=P+I;  // Señal de control PI, Proporcional e Integrador
  if(U>0){      // Modificamos el sentido de giro en
funcion del signo de mi señal de entrada,
     digitalWrite (IN3, HIGH);    // positivo a derechas y
negativo izquierdas
     digitalWrite (IN4, LOW);
  }
  else{
    digitalWrite (IN3, LOW);
     digitalWrite (IN4, HIGH);
     U=abs(U);
  }
  if(U>255){
    U=255;
  }
  analogWrite(ENB,U);

 }
```

10.2. Plano del soporte



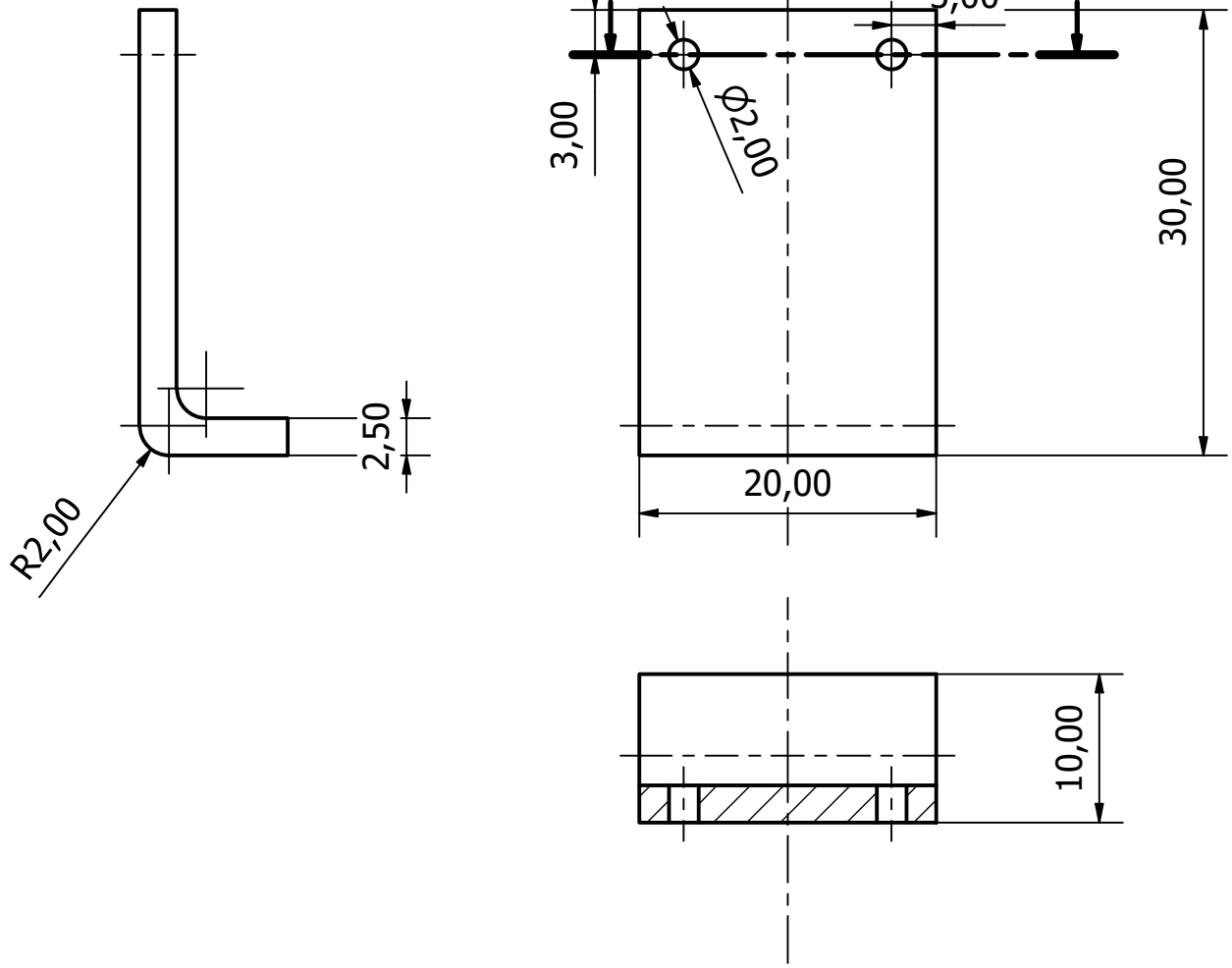


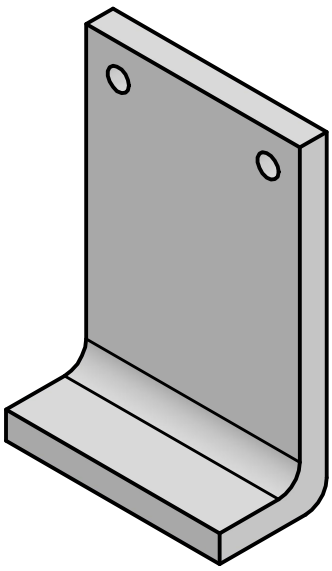




Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
Alberto Monforte					29/06/2021	

Diseño de soporte para motor Hanpose 775	Escala: 1:1		
	Soporte motor plano	Edición	Hoja 1 / 1





Diseño de	Revisado por	Aprobado por	Fecha		Fecha	
Alberto Monforte					29/06/2021	

Diseño de soporte para el encoder	Escala: 2:1		
	Soporte sensor plano	Edición	Hoja 1 / 1