UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE VELOCIDAD PARA MOTORES DE CONTINUA BASADO EN MICROCONTROLADOR

AUTORA: WEN YI SUN

TUTOR: ALBERTO JOSÉ PÉREZ JIMÉNEZ

Curso Académico: 2018-19

AGRADECIMIENTOS

A mis padres y a mi hermano, por ser mi inspiración en la vida y porque gracias a ellos soy quien soy ahora.

A mi tutor, por sus ánimos, su ayuda y haber sido mi guía.

A mis amigos, compañeros y profesores, por su acompañamiento en este recorrido académico.

Y en especial, a mi novio, por su apoyo incondicional.

RESUMEN

En el presente trabajo se ha realizado el diseño e implementación de un sistema de control que permite regular la velocidad de rotación de un motor de continua, mediante un microcontrolador, concretamente el Arduino MEGA 2560, del que se comentarán sus características más adelante.

Para ello, se pretende desarrollar un controlador PID digital en el Arduino, implementando el código que corresponde a esa acción de control y transmitirlo al motor de continua. Cabe destacar la previa conversión de la ecuación de control analógica a digital.

Además, se incluirán otros componentes, como un encoder rotativo y una pantalla LCD, que ayudarán al usuario a introducir la velocidad deseada y los parámetros requeridos para el controlador PID. Con ello también se podrá visualizar los resultados de una manera más sencilla e inmediata. También ha sido imprescindible el uso de un controlador de motor para la alimentación de dicho motor de continua.

Por otra parte, el motor no posee ninguna carga en su eje de salida. No obstante, si se frena el motor de manera forzada por algún factor externo, se ha incluido un sensor de corriente que servirá para medir la intensidad que se está suministrando al motor para que trate de llegar a la velocidad de consigna.

Finalmente, se ha diseñado una caja, con el programa Inventor, para poder situar todos los elementos conectados de una manera más conjunta y atractiva. Tras ello, se ha elaborado la caja físicamente con una impresora 3D.

Palabras clave: control, PID, microcontrolador, motor DC, velocidad, Arduino, programación, automática, Inventor, diseño.

RESUM

En el present treball s'ha realitzat el disseny i implementació d'un sistema de control que permet regular la velocitat de rotació d'un motor de contínua, mitjançant un microcontrolador, concretament el Arduino MEGA 2560, del qual es comentaran les seues característiques més endavant.

Per això, es pretén desenvolupar un controlador PID digital en l'Arduino, implementant el codi que correspon a eixa acció de control i transmetre-ho al motor de contínua. Cal destacar la prèvia conversió de l'equació de control analògica a digital.

A més, s'inclouran altres components, com un encoder rotatiu i una pantalla LCD, que ajudaran a l'usuari a introduir la velocitat desitjada i els paràmetres requerits per al controlador PID. Amb allò també es podrà visualitzar els resultats d'una manera més senzilla i immediata. També ha sigut imprescindible l'ús d'un controlador de motor per a l'alimentació del motor de contínua mencionat.

D'altra banda, el motor no té cap càrrega en el seu eix d'eixida. No obstant, si es frena el motor de manera forçada per algun factor extern, s'ha inclòs un sensor de corrent que servirà però mesurar la intensitat que s'està subministrant al motor perquè tracte d'arribar a la velocitat de consigna.

Finalment, s'ha dissenyat una Caixa, amb el programa Inventor, per poder situar tots els elements connectats d'una manera més conjunta i atractiva. Després d'això, s'ha elaborat la Caixa físicament amb una impressora 3D.

Paraules clau: control, PID, microcontrolador, motor DC, velocitat, Arduino, programació, automàtica, Inventor, disseny.

ABSTRACT

In this project, it has been made a design and the implementation of a control system which allows to adjust the rotation speed of a DC Motor, through a microcontroller, specifically the Arduino MEGA 2560, whose features will be remarked later.

To do this, the aim is to develop a digital PID controller in the Arduino, implementing the code which corresponds to that control action and then, to send to the DC Motor. It should be noted the previous conversion of the analog to digital control equation.

In addition, other components will be included, such as a rotary encoder and an LCD screen, which will help the user to input the wanted speed and the parameters required for the PID controller. Also, results will be seen in a simpler and faster way. It has been indispensable to use a driver for powering the DC Motor.

On the other hand, the motor does not have any engine load on its output shaft. However, if the motor is slowed down in a forced way by some external factor, a current sensor will measure the intensity that is being supplied to the motor, which is trying to make the motor reach the set speed.

At last, a box has been designed with a program called “Inventor” to be able to place all the connected elements in a more joint and attractive way. After that, the box has been physically created with a 3D printer.

Keywords: control, PID, microcontroller, DC motor, speed, Arduino, programming, automatic, Inventor, design.

Este escrito consta de los siguientes documentos:

DOCUMENTO I: MEMORIA

DOCUMENTO II: PRESUPUESTO

DOCUMENTO III: PLANOS

ANEXO I: CÓDIGO PROGRAMACIÓN

DOCUMENTO I: MEMORIA

ÍNDICE DE LA MEMORIA

ÍNDICE DE LAS ILUSTRACIONES

1. INTRODUCCIÓN

1.1. Objetivos

El principal objetivo del presente Trabajo Fin de Grado es diseñar e implementar un sistema de control de velocidad para motores de continua, es decir, crear una interfaz donde el usuario pueda regular la rotación de un motor DC a un estado final deseado.

Concretamente, lo que se pretende es:

- Conocer el manejo del programa Arduino (2019) y varias de sus funciones básicas.
- Diseñar una estructura, con todos los componentes necesarios, con la que el usuario pueda comunicarse con el Arduino e interpretar los resultados de una manera sencilla física y visualmente.
- Desarrollar un controlador PID digital, con su previa conversión de la ecuación analógica a digital, y sintonizarlo de manera aceptable para su funcionamiento.
- Conseguir que el motor DC alcance la velocidad de referencia deseada de manera rápida y suave (idealmente sin oscilaciones).
- Diseñar una caja para englobar todo y proporcionar más estética al montaje final.

Finalmente, es importante mencionar todos los conocimientos que se pueden adquirir gracias a la realización de este trabajo, ya que abarca muchos campos estudiados durante el grado: automática, electrónica, eléctrica, mecánica, diseño, informática, etc.

1.2. Antecedentes

Este trabajo surge de la necesidad de controlar un motor de continua de forma leve y rápida. Para conseguirlo se ha optado por implementar un controlador PID, ya que es la herramienta más usada en la industria, haciendo uso de ella para más de 95% de procesos de control en lazo cerrado.

A su vez, este tipo de control surgió por razones bélicas. Nicolas Minorsky fue quien lo desarrolló con la finalidad de crear la navegación automática de la flota americana en 1922. Desde entonces, se fue introduciendo en la industria y mejorando hasta hoy en día que, a pesar de todos los métodos de control avanzados y alternativas que existen, se sigue usando frecuentemente. La razón de ello reside en que es un sistema robusto (tiene buen comportamiento ante distintas situaciones) y sencillo (fácil de comprender su funcionamiento).

Por otra parte, para llevar a cabo el control del motor, se ha usado un Arduino, un microcontrolador fácil de usar y barato. Por tanto, se puede decir que se aborda el problema de una manera económica, lo cual es importante también, haciendo que este sistema sea accesible para cualquier usuario.

Cabe destacar que, aunque en este trabajo el dispositivo a manejar es un motor de continua, el proyecto puede ser útil para otro tipo de motores y aplicaciones. Por lo que es un controlador bastante versátil.

1.3. Motivación

El principal motivo por el que se ha decidido realizar este trabajo es el gran interés académico que promovían las asignaturas relacionadas con informática y automática. Desde el primer curso con "Informática", pasando por "Sistemas Automáticos", "Tecnología Automática", "Tecnología Informática Industrial" e incluso la optativa de "Desarrollo de Aplicaciones Móviles", despertaron una fascinación a nivel personal con la que era inevitable no realizar un Trabajo Fin de Grado de este ámbito.

Asimismo, como ya se ha comentado en apartados anteriores, estos no han sido los únicos campos explorados durante la realización de este proyecto. Se han aplicado también, aunque no con la misma importancia, otros entornos atractivos como la electrónica (muy relacionada con la automática en muchas aplicaciones) y el diseño (con la caja creada en el programa Inventor).

Por tanto, el carácter multidisciplinar de este trabajo sumado al interés personal ha hecho posible la elección de esta temática para este Trabajo Fin de Grado.

1.4. Justificación

Como ya se ha dicho anteriormente, académicamente hablando, las asignaturas que tienen que ver con informática y automática son atractivas de estudiar. No obstante, a pesar de tener varias asignaturas de esos ámbitos, no se ha llegado a profundizar mucho en el punto práctico y profesional.

Por tanto, esta ha sido una buena oportunidad de, por una parte, contemplar de una forma más cercana el funcionamiento del controlador PID y, por otra parte, de escribir un código más elaborado y práctico en lenguaje C++.

Hay que hacer hincapié también en el interés industrial y tecnológico que tiene un controlador PID, por razones ya comentadas. Por lo que implementar uno debe ser un punto muy favorable estudiantil y profesionalmente hablando.

Por otra parte, se va a trabajar con un Arduino, uno de los microcontroladores más populares en la actualidad, por su versatilidad, facilidad de uso y precio. Fue diseñado para ser más accesible, lo que permite que cualquiera pueda comprar uno y empezar a crear sus proyectos interactivos, como mover un motor, regar automáticamente las plantas del hogar y un largo etcétera. Todo con un coste muy asequible.

2. IMPLEMENTACIÓN DEL CONTROLADOR PID

Blasco, Martínez, Senent y Sanchis (2000) redactaron que "El objetivo que se persigue con el control de un proceso es que dicho proceso tenga el comportamiento deseado" (p.9).

Por tanto, lo que se busca es implementar un controlador que alcance o mantenga un estado final deseado por el usuario. Para ello se usará un regulador tipo PID, el más común para esta aplicación, por su robustez y sencillez, como se ha comentado anteriormente.

El PID consta de tres acciones: Proporcional, Integral y Derivativo. A continuación, se explicará de forma breve cada una de las partes (Herrero, 2013). Pero antes de entrar en detalle, cabe destacar la definición de error.

El error es la diferencia entre el valor de referencia y el valor medido. El valor de referencia, llamado también "set point" o punto de consigna, es aquel que introduce el usuario, el punto final que se desea que se alcance.

Para el funcionamiento del controlador PID (2019), se necesita al menos un sensor que determine cuál es el estado del sistema, un controlador (PID) que genere la señal para corregir dicho sistema y un accionador para que mande la señal generada (variable manipulada) al sistema:

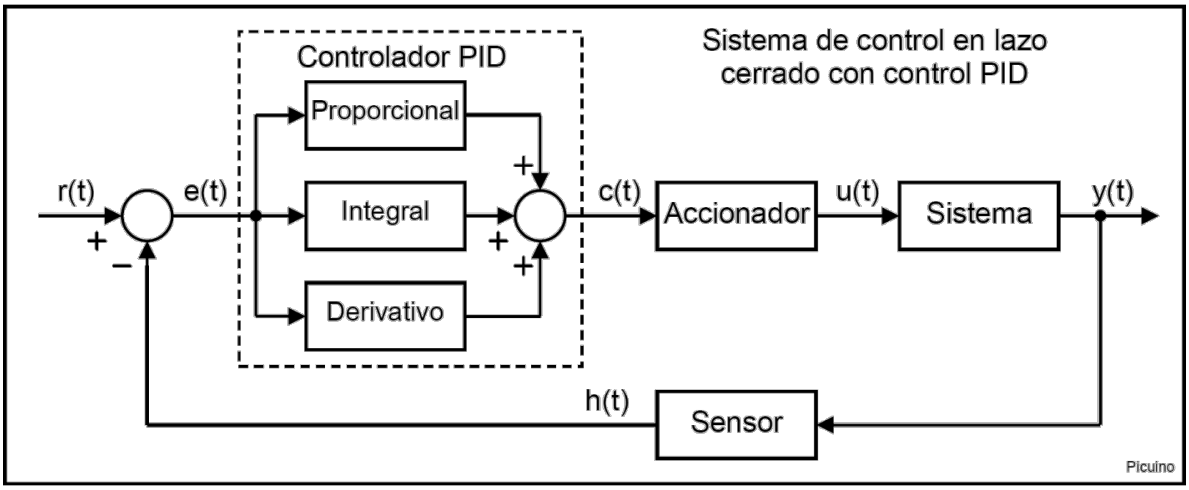


Ilustración 1. Funcionamiento del controlador PID

Para nuestro proyecto, el sensor será un encoder que está incorporado en el motor. El controlador PID es el que se introducirá mediante el Arduino. Y el accionador será el controlador de motor, mandando la potencia necesaria al motor de continua.

2.1. Acción proporcional

Es la acción cuya salida da una señal proporcional al error. Es instantánea, influye en el presente y no considera el tiempo. La expresión que la define es:

$$u_p(t) = K_p \cdot e(t) \quad (1)$$

Cuanto mayor sea la K_p, más rápida será la respuesta y menor error en régimen permanente tendrá, pero el sistema será más inestable. Por lo que, si se ajusta la K_p lo suficientemente bien en cuanto a rápido y estable, se podría usar un control tipo P. Sin embargo, lo usual es añadirle un componente integral o derivativo.

2.2. Acción integral

La integral sí considera el tiempo, concretamente el pasado. Su objetivo es reducir el error permanente a cero. Por tanto, lo que realiza es un control sobre el error acumulado. Su expresión es la siguiente:

$$u_i(t) = K_i \cdot \int_0^t e(t)dt \quad (2)$$

A mayor K_i, se disminuye el error permanente de manera más rápida, pero con oscilaciones que podrían aportar inestabilidad al sistema. Por tanto, hay que encontrar una constante adecuada para que ese error acumulativo sea nulo, pero sin sobreoscilar el sistema.

2.3. Acción derivativa

Este componente es el que predice el error futuro. Su finalidad es conocer la velocidad con la que cambia la señal de error y controlarla de una manera más rápida para no permitir que ese error aumente, es decir, que se adelantará al error que se producirá y realizará cambios significativos con tal de mantener el error al mínimo. Su expresión es:

$$u_d(t) = K_d \cdot \frac{de(t)}{dt} \quad (3)$$

Aumentar la K_d implica una mayor estabilidad del sistema y un poco de disminución de la velocidad de este. En cambio, si se aumenta demasiado, tiende a desestabilizar y, si es demasiado pequeño, produce muchas oscilaciones. Por otra parte, otro de los problemas en los que reside en la acción derivativa es que es sensible al ruido y amplifica sus señales, al mínimo error, esta ganancia aumenta de una manera significativa y esto provocaría muchas oscilaciones. Por esta razón, en algunas aplicaciones se prescinde de este término.

La suma de las tres acciones compondrá lo que es el controlador PID, quedando la expresión de la siguiente manera:

$$u(t) = K_p \cdot e(t) + \frac{K_p}{T_i} \cdot \int_0^t e(t)dt + K_p \cdot T_d \cdot \frac{de(t)}{dt} \quad (4)$$

2.4. PID industrial

Blasco et al. (2000) escribieron:

> Sobre la base de este regulador se suelen realizar una serie de modificaciones que permiten la correcta implementación en un proceso industrial. Las modificaciones se refieren tanto a aspectos de implementación (diferentes estructuras, implementación digital, etc.) como a la resolución de los problemas del algoritmo frente a situaciones comunes en la industria (saturaciones, señales ruidosas, etc. (p.189).

La ecuación de control anterior es una de las formas comunes de expresar el algoritmo básico de un PID, llamada "Estándar ISA". Sin embargo, para este trabajo será necesario realizar una implementación digital para poder incluirlo en Arduino.

2.5. Implementación digital

Para desarrollar un controlador PID en un procesador digital, hay que realizar una conversión aproximada del algoritmo de control a ecuaciones en diferencias. Existen varias alternativas, pero la implementación más típica es el método del rectángulo posterior.

Partiendo de la ecuación (4), se tiene que:

$$u(t) = u_p(t) + u_i(t) + u_d(t) \quad (5)$$

Se realiza la discretización de cada uno de los componentes, resultando:

- Acción proporcional

$$u_p(t) = K_p \cdot e(t) \Rightarrow u_p(k) = K_p \cdot e(k) \quad (6)$$

- Acción integral

$$u_i(t) = K_i \cdot \int_0^t e(t)dt \Rightarrow u_i(k) = u_i(k-1) + \frac{K_p \cdot T}{T_i} \cdot e(k) \quad (7)$$

- Acción derivativa

$$u_d(t) = K_d \cdot \frac{de(t)}{dt} \Rightarrow u_d(k) = K_p \cdot T_d \cdot \frac{e(k) - e(k-1)}{T} \quad (8)$$

La suma de estas tres componentes discretizadas resultaría la acción de control que se debe transmitir al proceso. Esta es:

$$u(k) = u_p(k) + u_i(k) + u_d(k) \quad (9)$$

En este trabajo, por las razones anteriormente comentadas, conviene prescindir del componente derivativo, no se desea ningún cambio brusco y no aportaría mejora alguna. Además, en la asignatura de "Sistemas automáticos", dada en la carrera, se comentó una regla a la hora de elegir controlador: escoger el más simple posible. Por lo que finalmente bastaría con introducir un controlador PI:

$$u(k) = u_p(k) + u_i(k) \quad (10)$$

Anteriormente se ha expresado la gran importancia que tiene elegir correctamente las constantes de control. Por ello surge la necesidad de ajustarlas y tratar de encontrar el mayor equilibrio entre ellas. Existen distintas formas de ajuste o sintonía. Y pueden ser de forma analítica o empírica.

Con la primera habrá un mayor control sobre el ajuste y permite que se conozca el comportamiento del sistema ante variaciones de las constantes. No obstante, es más compleja, ya que se ha de modelizar matemáticamente el sistema.

Por otra parte, la segunda manera evita la modelización matemática, lo cual supone un ahorro de cálculo, aunque no se vaya a tener tanto control sobre el conocimiento de su comportamiento frente a cambios.

Dentro de los métodos de ajuste empírico también hay varios tipos, uno de los más populares en la industria es el de Ziegler-Nichols (2019). Consiste en, a partir de una gráfica, obtener los parámetros necesarios que, al sustituirlos en la siguiente tabla, resultan los valores de las constantes proporcional, integral y/o derivada que formarán el controlador deseado.

A su vez, existe el ajuste en lazo abierto y en lazo cerrado. En este trabajo se usará el método de sintonización en lazo cerrado, ya que es el estudiado. En esta condición, las expresiones con las que se obtendrán los valores de las constantes proporcional, integral y/o derivada, según el tipo de controlador deseado, son:

	Kp	Ti	Td
P	$0{,}5 \cdot Kcr$		
PI	$0{,}45 \cdot Kcr$	$\frac{Pcr}{1{,}2}$	
PID	$0{,}6"Kcr$	$\frac{Pcr}{2}$	$\frac{Pcr}{8}$

Tabla 1. Ajuste Ziegler-Nichols en lazo cerrado

Siendo Kcr la ganancia crítica y Pcr el periodo de oscilación crítico.

Para obtener el valor de la ganancia crítica (Kcr) basta con aumentar el valor solamente de Kp hasta que el sistema empiece a oscilar de forma uniforme. El periodo de oscilación crítico (Pcr) será el tiempo que transcurre de pico a pico.

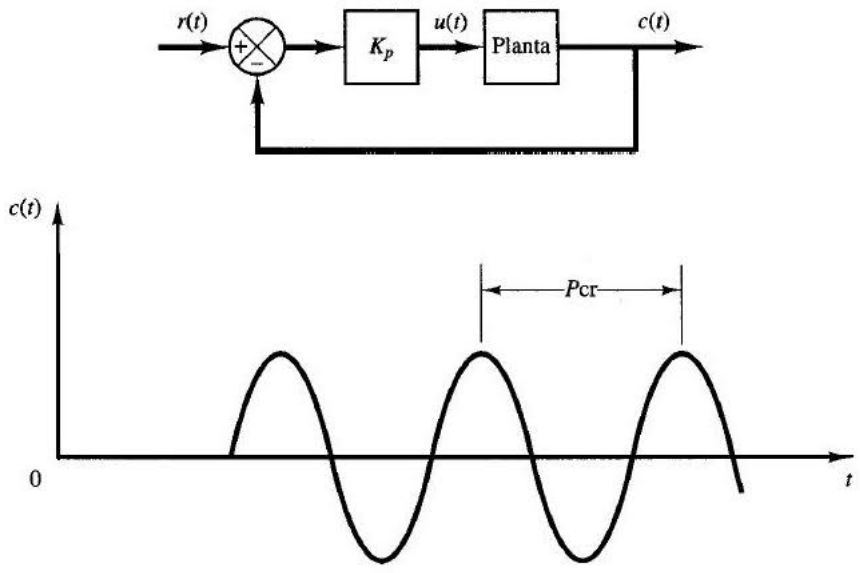


Ilustración 2. Kcr y Pcr con el método de ajuste Ziegler-Nichols en lazo cerrado.

3. HARDWARE

3.1. Componentes

3.1.1. Arduino MEGA 2560

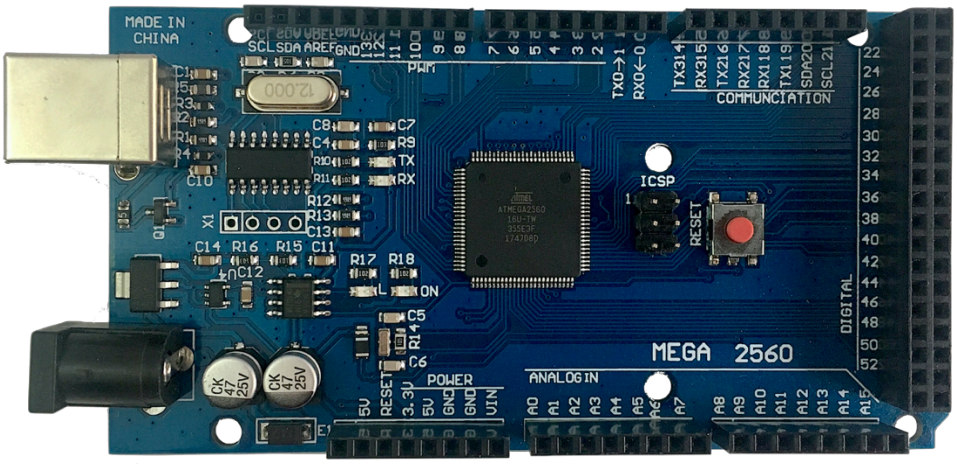


Ilustración 3. Arduino MEGA 2560

Una placa electrónica con un microcontrolador incorporado. La ventaja en este tipo de placas rige en que tanto su hardware como su software son libres, por los que son flexibles y fáciles de usar, adaptándose a las necesidades del usuario. Otra ventaja que destacar es su bajo coste, objetivo por el cual se inventó este dispositivo, ya que fue un proyecto enfocado a estudiantes de electrónica que usaban microcontroladores bastante más caros que estos.

Algunas de sus principales características son:

- 54 pines digitales de entrada/salida (15 de las cuales se pueden usar como salidas PWM)
- 16 pines de entradas analógicas
- 4 puertos serie
- Voltaje operativo: 5 V
- Límite de voltaje de entrada: 6-20 V (recomendado entre 7-12 V)
- Reloj: Cristal oscilador de 16 MHz

Además, consta de una conexión a USB, donde se produce la comunicación de la placa con el ordenador en el que se introduce el código de programación para enviarle las instrucciones que se desea, esta conexión es la que le proporcionará la alimentación necesaria. Teniendo también un botón de reset y otra entrada alternativa para la alimentación de la placa (con una clavija Jack hembra).

Se hará uso de una de las salidas PWM, por lo que sería importante realizar una breve descripción de ellas. Son aquellas que se usarán como una alternativa en caso de que se requiera utilizar una salida analógica ya que la placa Arduino no es capaz de proporcionarla. Por tanto, habrá que simularla de alguna forma mediante señales digitales, haciendo posible las regulaciones de voltaje y, a su vez, de velocidad, por ejemplo.

Las señales PWM (pulse-width modulation, en español, modulación de ancho de pulso) son una de las formas más simples con el fin de realizar esa conversión digital-analógica. Las señales digitales pueden ser 1 o 0 y, para poder regular a un valor entre todo y nada, el funcionamiento de la simulación de las señales analógicas consiste en mantener activa la salida digital durante parte de un tiempo periódico y apagarla durante el resto del periodo.

3.1.2. Arduino Sensor Shieldv5.0

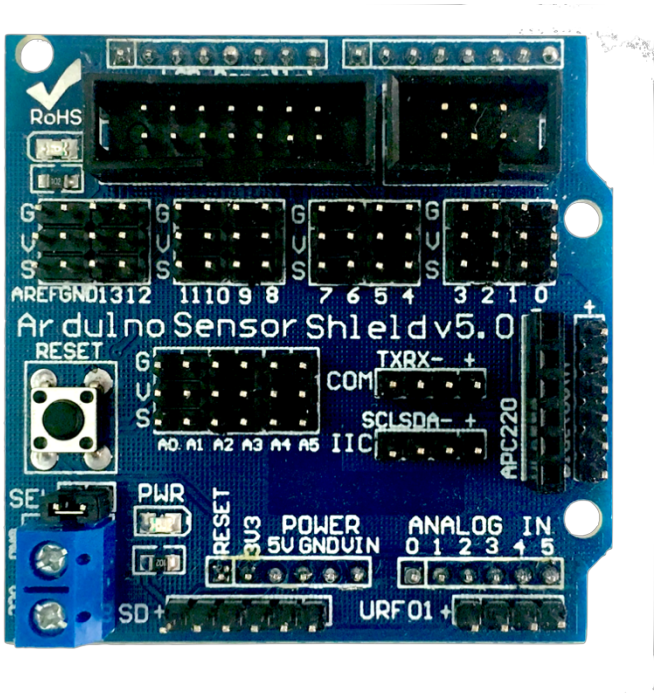


Ilustración 4. Arduino Sensor Shieldv5.0

Una placa de expansión, que agrega circuitería, sensores y módulos de comunicación externos. En este caso, ha sido de gran utilidad para poder organizar todo el cableado de los distintos componentes de una manera más ordenada y distintiva. Otra de las razones por las que se ha utilizado este dispositivo es que cada una de las entradas analógicas y entradas/salidas digitales posee 3 pines: G (Tierra), V (Alimentación) y S (Señal), siendo posible enviar o recibir señales a la vez que se alimenta el dispositivo que se conecte, todo desde la misma placa. Esto supone una gran ventaja ya que no haría falta añadir ninguna alimentación externa para cada dispositivo añadido.

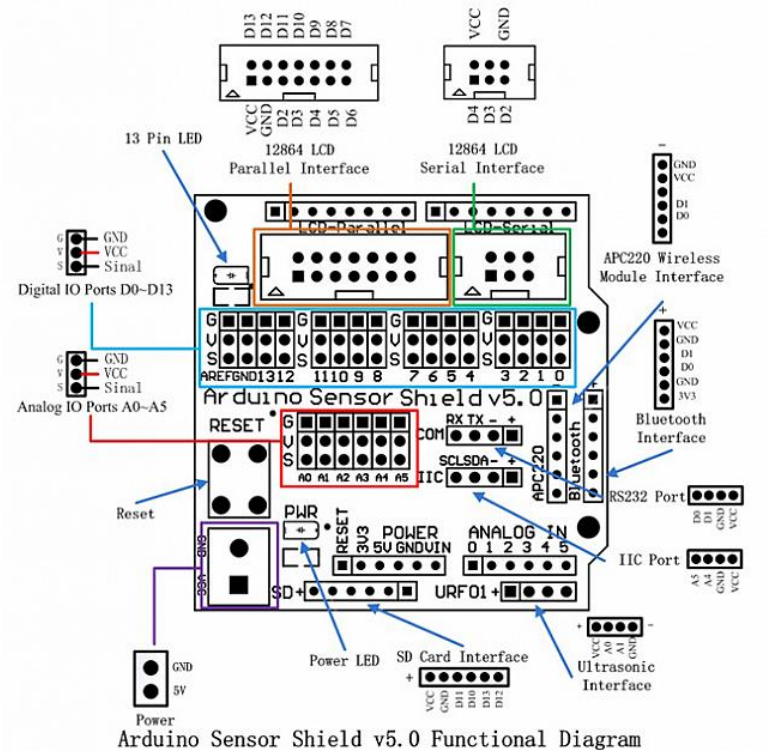


Ilustración 5. Descripción de los pines del Arduino Sensor Shield v5.0

La forma de conectarlo a la placa Arduino es muy sencilla, solo hay que encajar los pines macho del Shield con los pines hembra del Arduino, de modo que no quede ningún pin macho sin conectar. Quedando así:

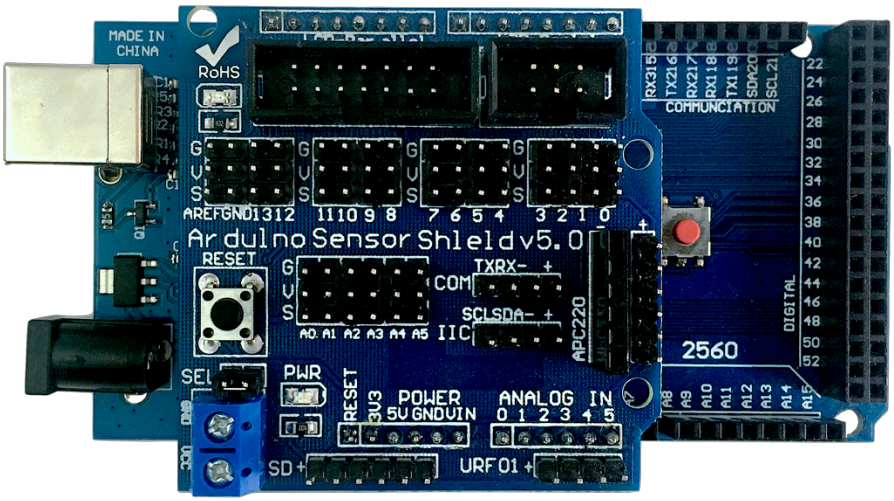


Ilustración 6. Arduino Sensor Shieldv5.0 conectado a la placa base

3.1.3. Controlador de motor (o Driver) L298N

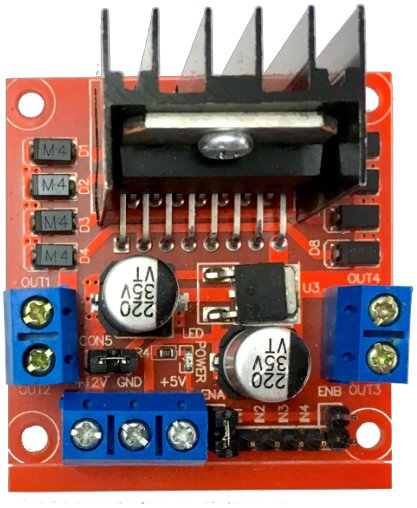


Ilustración 7. Controlador de motor L298N

Este módulo posee dos puente-H que permiten controlar la velocidad y la dirección de dos motores de continua o un motor paso a paso.

Un puente-H es un conjunto de cuatro transistores que, al estar conectados de la forma en que se muestra en la siguiente fotografía, permite cambiar el sentido del voltaje de un motor, y por tanto el sentido de la intensidad, haciendo posible que el motor pueda rotar en ambos sentidos.

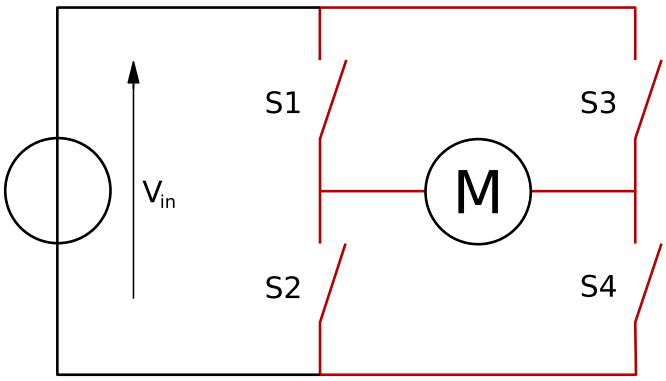


Ilustración 8. Circuito Puente-H

Se usará para poder controlar un motor de continua. Con un adaptador de corriente, se convertirá los 230 Voltios de corriente alterna en 12 Voltios de corriente continua que alimentarán al controlador. A su vez, este dará una salida de 12 Voltios DC para poder poner en marcha el motor y, gracias al regulador de tensión que contiene, otra salida de 5 Voltios para poder alimentar al

Arduino. Más adelante se explicará con más detalle la conexión entre estos elementos para hacer posible esa conversión.
Además, cabe destacar los jumpers que contiene y sus funciones: uno que controla la alimentación y otros dos que habilitan las salidas analógicas para dos motores DC.

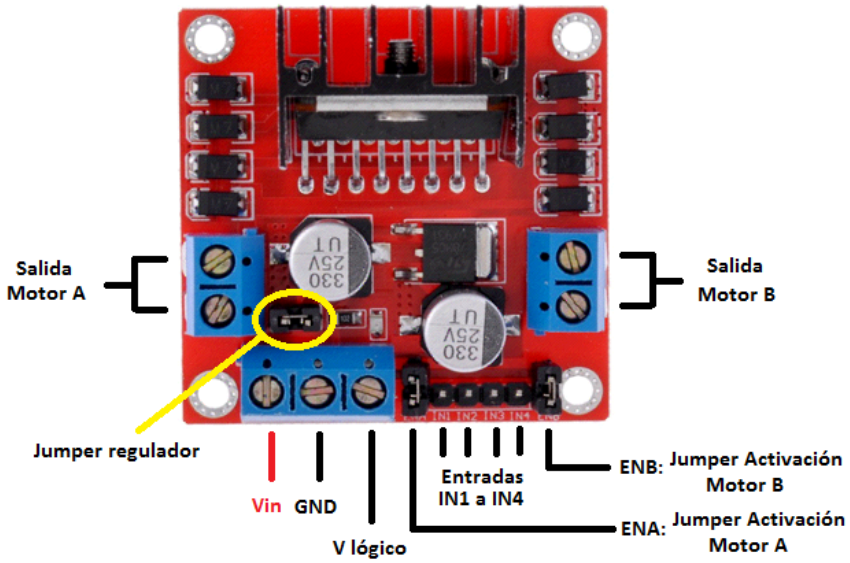


Ilustración 9. Descripción de las entradas, salidas y jumpers del controlador L298N

Si el jumper regulador está activo, es decir, que tiene el jumper puesto, significa que el controlador permitirá una alimentación de entre 6 y 12 Voltios en DC por el pin que señala “+12”, y el pin que marca “+5V” tendrá una salida de voltaje de 5 Voltios DC, (que en este caso servirá para alimentar la placa de Arduino).

Si por el contrario ese jumper está inactivo (sin poner), el controlador permitirá una alimentación de 12 a 35 Voltios DC (pin “+12”) y hará falta una alimentación externa para alimentar su parte lógica con 5 Voltios (pin “+5V”).

Para el caso del jumper que habilita la salida analógica del motor (jumper activación), este es útil para controlar la velocidad de dicho motor, en otras palabras, para obtener una velocidad variable dentro de un rango en lugar de que solo vaya a máxima o nula velocidad. Si se deja puesto será todo o nada (señal digital) y, si se quita se podrá conectar el pin a una salida PWM, que emulará la señal analógica de tensión requerida para regular dicha velocidad.

3.1.4. Sensor de corriente ACS712

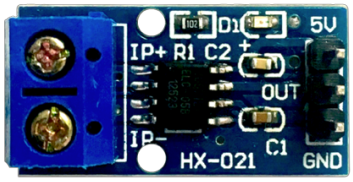


Ilustración 10. Sensor de corriente ACS712

Este sensor sirve para medir la corriente que atraviesa un conductor, se conecta en serie entre el controlador y el motor. En este trabajo se ha incluido con la finalidad de saber, de forma económica, la intensidad que se consume en caso de que el motor esté siendo frenado por alguna carga exterior. El valor de dicha intensidad se mostrará mediante una pantalla.

El modelo de este sensor es "ACS712ELCTR-05B-T". Hay un dato relevante que se encuentra en sus especificaciones que será imprescindible: la sensibilidad que, dependiendo del rango de corriente que mide, será de un valor u otro. Para el aparato usado cuyo rango es de [-5,5] Amperios, su valor es 185mV/A. (Naylamp Mechatronics, s.f.)

Trabaja con efecto Hall, que consiste en la aparición de una tensión a través de la separación de cargas que se produce mediante la circulación de una corriente en presencia de un campo magnético. Mide corrientes en alterna y en continua. Tiene incluido un offset, cuando la intensidad que pasa por el conductor es 0 A, el voltaje de salida es 2,5 Voltios. Al estar la referencia centrada, permite que se pueda medir intensidades positivas y negativas. A partir de ese punto, existe una relación proporcional entre el voltaje de salida y la intensidad. La ecuación que las relaciona sería: $V = m \cdot I + 2.5$, donde la m es la sensibilidad. Despejando la intensidad se tiene: $I = (V-2.5) / 0.185$.

Por último, se debe comentar que el microcontrolador lee las entradas analógicas de forma que convierte sus valores a números entre 0 y 1023, por lo que, para interpretar este sensor, cuyos valores de voltaje van desde 0 a 5, habrá que realizar una regla de tres. Para ello, el valor leído con la función "AnalogRead()" se multiplicará por: (5.0/1023.0), el decimal es para precisar más, por lo que ese número será de tipo "float".

3.1.5. Encoder rotativo

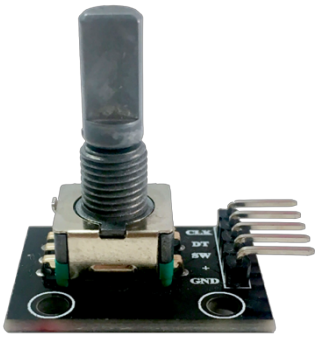


Ilustración 11. Encoder (visto de frente)

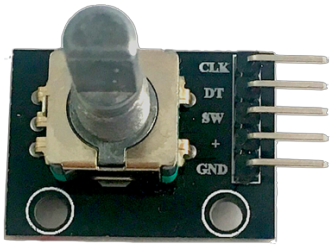


Ilustración 12. Encoder (visto desde arriba)

Este dispositivo se usa en conjunto con la pantalla LCD que se describirá en el siguiente punto. Se lee su movimiento, que dependerá del usuario, mediante el Arduino y ello causará un efecto en la pantalla, haciendo posible el movimiento por la pantalla para configurar los parámetros necesarios para el control del motor.

El encoder consiste en una especie de mezcla entre botón y rueda. Se tiene la posibilidad de pulsar, para confirmar algún parámetro, por ejemplo, y girar en sentido horario o antihorario, que en este caso se usará para recorrer el menú programado y cambiar valores de algunos parámetros.

Tiene 5 pines para su funcionamiento, tres son funciones clave para poder leer esos movimientos descritos:

- +: Voltaje
- GND: Tierra
- CLK: CLOCK. Es la señal de reloj.
- DT: DATA. Es la señal de datos.
- SW: SWITCH. Señal de presión.

Las señales que transmiten son cuadradas (encoder en cuadratura), digitales, tienen flancos de subida o de bajada, y desfasadas entre ellas 90 grados. Con las señales CLK y DT se pretende averiguar qué giro está realizando el encoder. Para ello, habrá que conocer en cuál de los dos se lee primero la señal de subida. Con la siguiente imagen se observa claramente la diferencia:

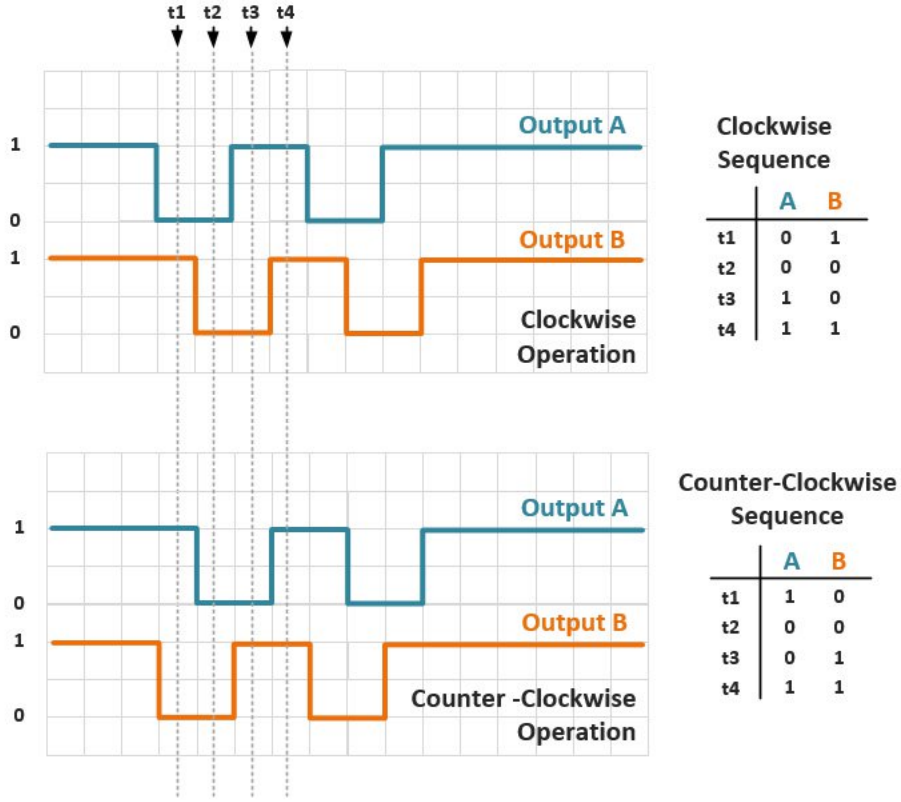


Ilustración 13. Descripción señales CLK y DT del encoder rotativo

Cuando se lee primero a la señal de subida del reloj CLK (en la imagen sería A) y luego la de datos DT (en la imagen sería B) quiere decir que el giro es horario (clockwise). Por el contrario, si se lee primero la señal DT y después la de CLK, el giro es antihorario (counter-clockwise). (Dejan, s.f.)

Para leer la señal de presión (SW) es sencillo. En este caso, la señal está en 1 mientras no se interactue con el pulsador y cuando se presiona el encoder, la señal pasa a ser de bajada (0).

3.1.6. Pantalla LCD

LCD: Liquid Crystal Display (Pantalla de Cristal Líquido).

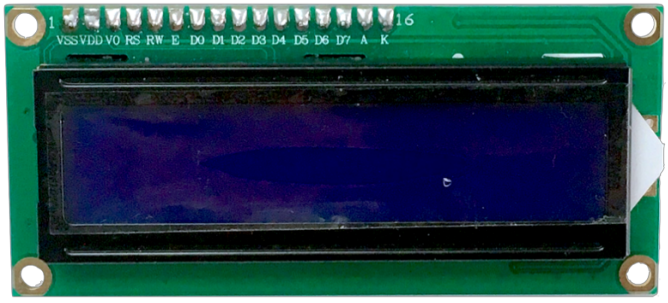


Ilustración 14. Pantalla LCD

Como recién se ha comentado, este es el dispositivo que se usará para mostrar y configurar algunos parámetros. Consiste en una pequeña pantalla donde se puede introducir 16 caracteres por fila en dos filas, es decir, 32 caracteres en total. Cuando está encendido, tiene el fondo de color azul con las letras blancas.

Sus ventajas son que son baratas, consumen poco, son muy prácticas y útiles para mostrar todo aquello que se quiere mostrar sin tener que recurrir al Monitor Serie de Arduino.

Hay que alimentarlo con 5 Voltios, por lo que se conectará al Arduino para ello. No es muy grande, sus dimensiones son de 80x36x12 (mm), pero es más que suficiente para la aplicación de este trabajo.

Se le ha incorporado un adaptador IIC/I2C para una fácil conexión con el microcontrolador, solo habría que conectar 4 pines: 2 de alimentación y otros 2 para la comunicación, mediante el bus I2C. Sin este, se tendría que conectar 16 pines: los 2 de alimentación y el resto para la comunicación. Esto dificultaría la conexión, por lo que supone una gran ventaja utilizar el adaptador. Para ello, lo primero es soldar los 16 pines al adaptador por la parte trasera. De forma que quedará de la siguiente forma:

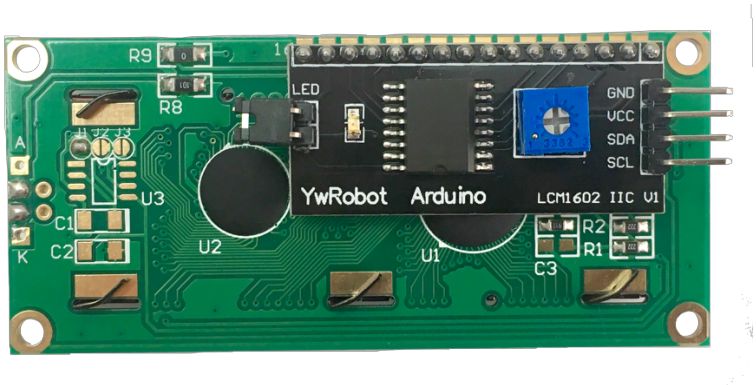


Ilustración 15. Pantalla LCD (vista trasera)

Así ya solo quedaría conectar los 4 pines. VCC y GND se conectan a cualquier V y G, respectivamente, de la placa de Arduino. Para los de SDA y SCL existen unos pines específicamente para su conexión, en el caso del Arduino MEGA2560, son los que marcan “SDA20” y “SCL21”, haciendo posible su comunicación con el microcontrolador. Como se puede apreciar también, tiene un potenciómetro incorporado que sirve para regular el contraste de la pantalla (el pequeño cubo azul).

3.1.7. Motor de continua

Ilustración 16. Motor de continua

Este es el elemento donde se plasmará este control. Es una máquina que convierte la energía eléctrica en mecánica, por medio de campo magnético. Y, a pesar de que actualmente este tipo de motores no son tan comunes como los de corriente alterna, todavía sigue vigente su utilización gracias a su posibilidad de alterar la velocidad.

Serrano y Martínez (2017) afirman:

> El motor de continua realiza de forma automática y correcta la inversión de corriente necesaria en sus conductores, cualquiera que sea su velocidad, a diferencia del motor síncrono en el que, una vez fijada la frecuencia de alimentación, sólo es posible su funcionamiento para una única velocidad (la de sincronismo) (p.259)

De forma muy resumida, se puede decir que la velocidad del motor es directamente proporcional a la tensión continua. “La máquina de continua está específicamente indicada para su utilización como motor en accionamientos industriales a velocidad variable.” (Serrano y Martínez, 2017, p.260).

Además, otra de sus ventajas reside en que pueden servir para abaratar costes y para aplicaciones donde los valores de voltaje son bajos. Siendo ambas principales razones por las que se usa esta máquina en este trabajo.

A continuación, se va a proceder a mostrar algunas de las características del motor utilizado. (TranThinh, s.f.)

El motor tiene 6 cables:

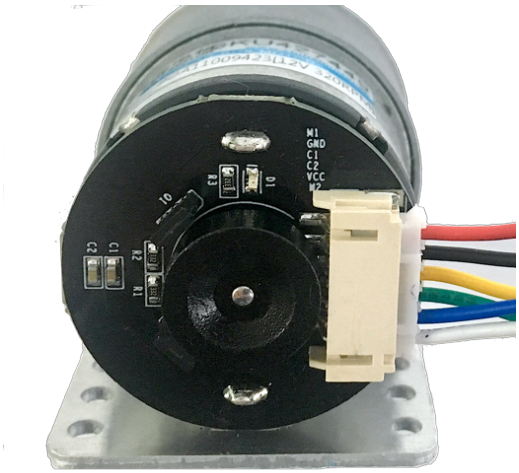


Ilustración 17. Encoder y cables del motor

- 2 para alimentar el motor. Señalados con “M1” y “M2” (de color rojo y blanco, respectivamente). Estos se conectan al controlador o driver que se ha descrito anteriormente, proporcionando 12 Voltios.
- 2 para alimentar el encoder. Señalados con “VCC” y “GND” (de color azul y negro, respectivamente). Se conectan al microcontrolador con alimentación de 5 Voltios, a los pines “V” y “G”, respectivamente, de alguna de las salidas digitales.
- 2 que servirán para enviar señales del encoder. Señalados como “C1” y “C2” (de color amarillo y verde, respectivamente). También serán conectados al microcontrolador, pero en algún pin “S” de alguna de las entradas digitales.

El encoder, que se halla en su interior, sirve para medir la velocidad a la que está rotando y saber en qué dirección está girando. Para ello, tiene un sensor óptico y dos señales digitales que se crean a partir de ranuras equidistantes de un disco situado en el eje del motor. Funciona de una forma similar al encoder rotativo descrito antes.

El proceso, dentro del motor cuando gira, es que hay un led que emite una luz y atraviesa las ranuras del disco. El sensor óptico detecta una señal de subida cuando recibe luz y una de bajada cuando se bloquea el paso de dicha luz. Para leer ambas direcciones, se dispone de dos sensores ópticos y dos leds para obtener dos señales digitales. De esta forma, funcionará de la misma manera que se ha comentado anteriormente con el encoder rotativo. Como solo se desea conocer la velocidad y el sentido de giro, ha sido suficiente con este tipo de encoder, conocido como codificador relativo o incremental.

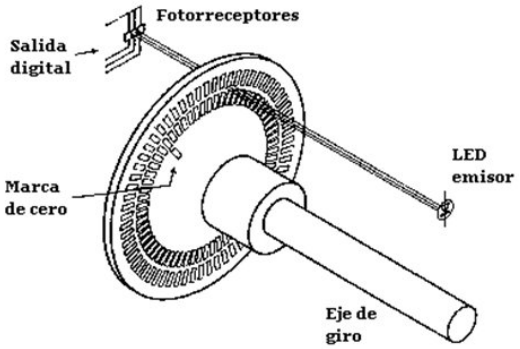


Ilustración 18. Descripción del interior del encoder del motor

Para saber la velocidad hay que realizar ciertas medidas y cálculos. Primero, se mide cuántas interrupciones hay por vuelta. Después, hay que conocer cuántas interrupciones hace por minuto, entonces así se hallará cuántas revoluciones por minuto realiza. Se especificará valores posteriormente.

Por último, solo cabe comentar que contiene un reductor de 30:1 en cuyo eje de salida se proporcionará un máximo de 320 revoluciones por minuto (a tensión nominal). Y también, que cada vuelta genera 330 pulsos.

3.1.8. Cable USB

Se dispone de un cable USB que sirve para realizar la comunicación entre el ordenador y el Arduino. Aunque sea un elemento muy básico, sin él, el sistema no funcionaría. Es un poco corto, pero es más que suficiente para trabajar con él.

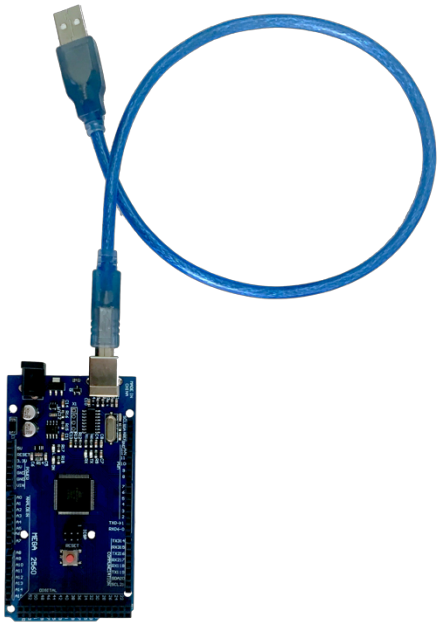


Ilustración 19. Cable USB del Arduino

3.1.9. Cableado

Por supuesto, hay que señalar que se hará uso de cables para conectar todos estos componentes entre sí. Son los llamados cables "Dupont" o cables de puente para prototipos. Son unos cables de tamaño variable consistentes en hilos conductores envueltos con aislantes de colores diferentes.

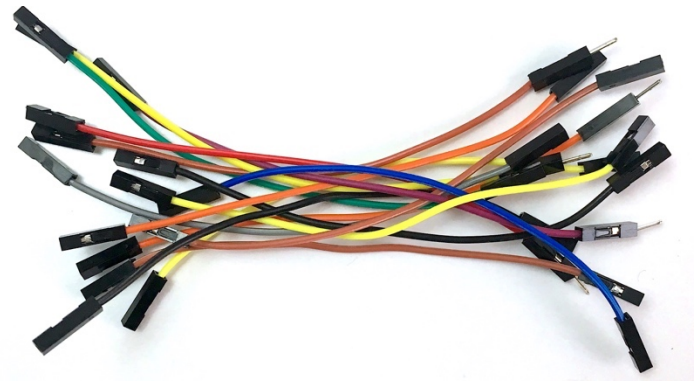


Ilustración 20. Cables de puente para prototipos

Poseen unos conectores en sus extremos (macho o hembra), incluidos manualmente, que sirven para poder enlazarse fácilmente a la placa, de Arduino en este caso, sin necesidad de soldar. También, son flexibles y prácticos para cambiarlos de lugar si se desea, por lo que es muy cómodo modificar y reutilizarlo para otras conexiones. Se ha usado con terminales macho-hembra y hembra-hembra.

3.1.10. Adaptador de corriente

Para alimentar el controlador, como ya se ha comentado, se hará uso de un adaptador de corriente que convierte los 230 Voltios en corriente alterna de la red a 12 Voltios en corriente continua que servirán para alimentar el driver.

Ilustración 21. Adaptador de corriente

3.2. Montaje

Para explicar el montaje hay que ir por partes:

Para empezar, se ha conectado el Arduino Sensor Shieldv5.0 al Arduino, es un proceso fácil, solo hay que asegurarse que están todos los pines bien colocados, de forma que no quede ningún pin macho sin ser conectado. Como ya se ha comentado, esto ha servido para facilitar las conexiones de todos los dispositivos necesarios para este montaje.

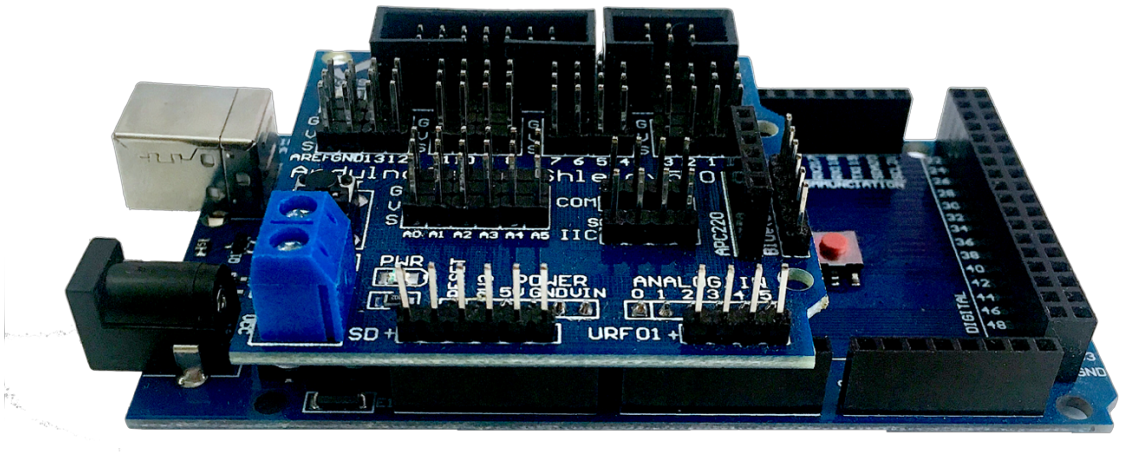


Ilustración 22. Montaje Arduino Sensor Shieldv5.0

El siguiente paso ha sido conectar el controlador con el motor (Electronilab, 2014) y el Arduino Sensor Shieldv5.0. Para empezar, hay que saber cuáles son los jumpers que se dejan puestos y cuáles no en este trabajo. El jumper de la alimentación de 12 Voltios se queda conectado para poder dar una salida de 5 Voltios al Arduino y no tener que introducir una entrada de dichos 5 Voltios. Por otra parte, solo hay que desactivar el jumper del pin “ENB” y “+5V”, ya que solo se hará uso de un motor y su salida PWM (emulando una salida analógica).

Para conectar el motor, se han conectado los cables pertenecientes a M1 y M2 del motor a los pines OUT3 y OUT4. Se ha escogido estas salidas en lugar de OUT1 y OUT2 por comodidad de cableado. Para controlar el giro del motor se ha conectado los pines IN3 e IN4 a las salidas digitales número 9 y 8 del Arduino, respectivamente. Y para poder regular la velocidad del motor, se ha tenido que conectar el pin ENB a la salida PWM número 7.

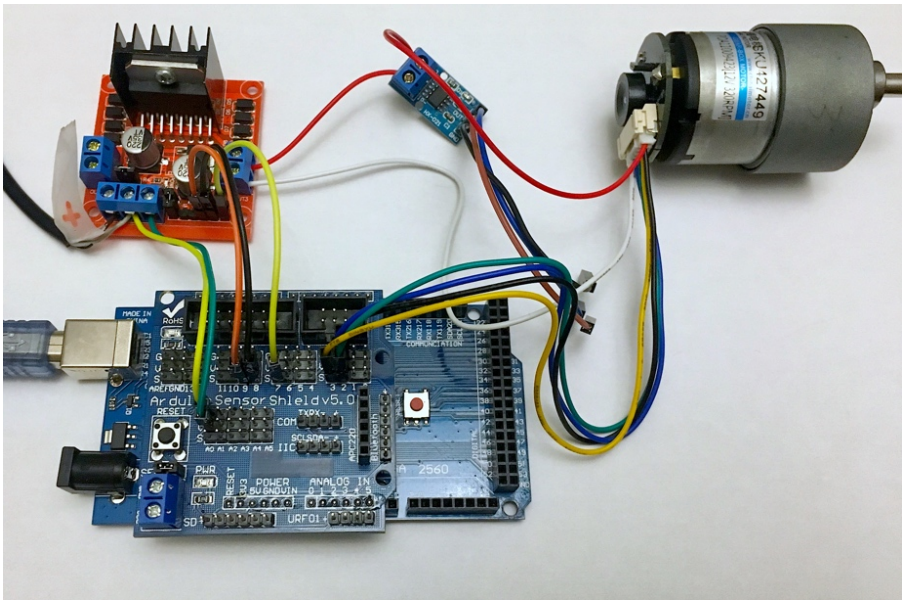


Ilustración 23. Montaje de motor + controlador + Arduino

Por otra parte, uno de los cables del motor, el que está conectado al OUT4 del controlador (el cable rojo), ha sido cortado para que sea posible la conexión del sensor de corriente. Por tanto, se está leyendo la intensidad que pasa por ese conductor rojo. La señal analógica para medir el voltaje se conecta en la entrada analógica A3. Además, la alimentación de este instrumento también se hace a través del Arduino, conectando sus pines "5V" y "GND" a cualquier "V" y "G" de Arduino, respectivamente.

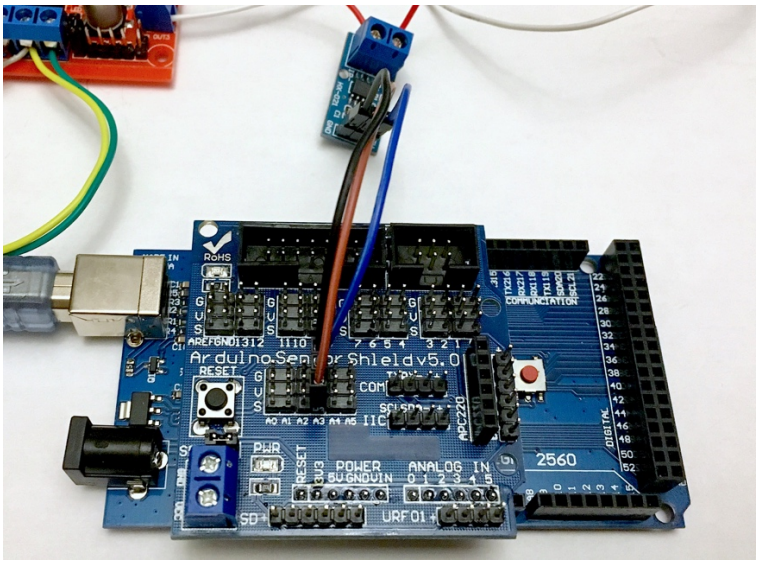


Ilustración 24. Montaje de Arduino + sensor de corriente

La pantalla LCD, como se ha comentado en su descripción, tiene 4 pines a conectar: "VCC" y "GND" se conectan a cualquier "V" y "G", respectivamente, de la placa de Arduino, de esta forma es como se alimentará la pantalla. Y para los de "SDA" y "SCL" existen unos pines específicamente para su conexión, en el caso del Arduino MEGA2560, son los que marcan "SDA20" y "SCL21", haciendo posible su comunicación serie con el microcontrolador.

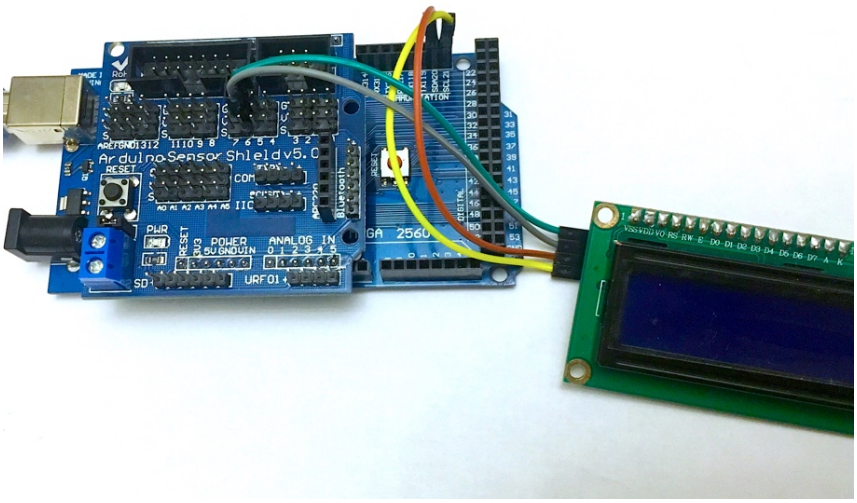


Ilustración 25. Montaje de Arduino + pantalla LCD

El encoder rotativo tiene 5 pines: 2 de alimentación (5V y tierra) que se conectarán a cualquier "V" y "G" de la placa y otros 3 pines que, resumiendo y recordando la información ya dada, proporcionarán señales digitales necesarias al Arduino y se traducirán como información necesaria para saber los movimientos que el usuario desea realizar en la pantalla LCD.

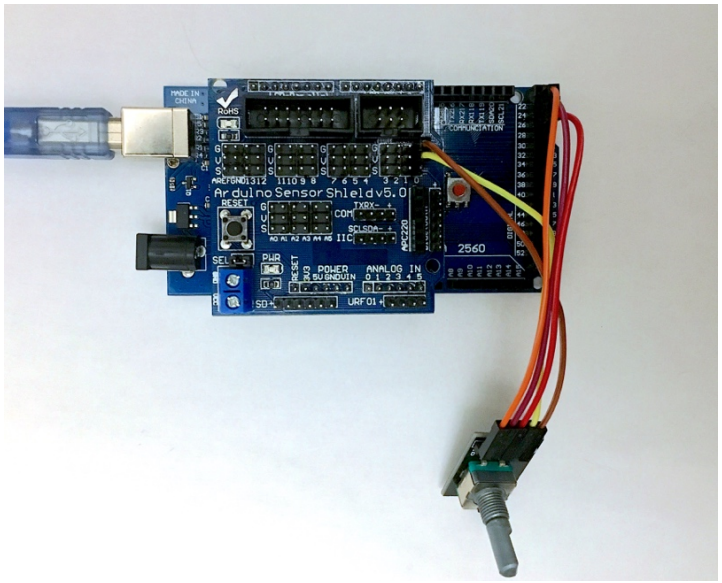


Ilustración 26. Montaje de Arduino + encoder rotativo

Se han aprovechado las conexiones de entradas digitales de la propia placa de Arduino para permitir más espacio entre el cableado. De esta forma, se han conectado los pines "CLK", "DT" y "SW" del encoder a los señalados "22", "24" y "26" de la placa, respectivamente.

Para que el usuario pueda recibir y/o enviar información al Arduino, hará falta un cable USB que conecte con él. El cabezal de USB se conecta a un puerto del ordenador del usuario y el otro cabezal se conecta a una entrada del Arduino. Además de ser un conductor de información, puede servir como alimentación (5 Voltios para los aparatos conectados a la placa).

Finalmente, con todo el circuito montado y para mayor facilidad a la hora de transportar y guardar todo este montaje, se ha decidido fijar todo sobre una chapa de metacrilato. Para ello, se han hecho varios agujeros, marcados con un rotulador para asemejar la situación de los dispositivos, con un taladro de columna de uno de los talleres de la universidad. Estos han servido para colocar unos tornillos con tuercas que actúan como soporte para cada componente, quedando de la siguiente manera:

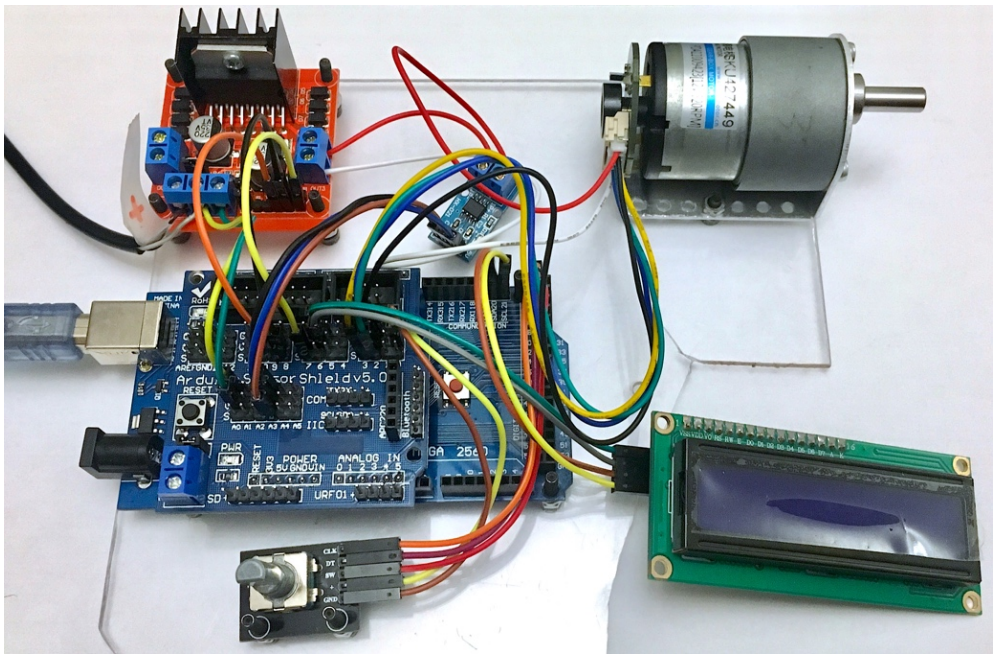


Ilustración 27. Montaje final con la plancha de metacrilato

Cabe destacar que posteriormente a esto se ha diseñado una caja para este trabajo, por lo que esto ha sido un montaje provisional para facilitar su manejo durante la realización del trabajo. El diseño de dicha caja está más ampliamente explicado en el Apartado 7 (“Diseño de la caja”).

4. PROGRAMACIÓN

4.1. Software

Para programar el control de velocidad se ha usado el programa Arduino. Es un programa bastante intuitivo, nada más abrirlo ya se puede empezar a escribir. Usualmente, es un programa que se usa para crear acciones automatizadas. El lenguaje de programación está basado en C++, además de todas las funciones básicas desarrolladas en su página web: https://www.arduino.cc/reference/en/

Además, es una plataforma de hardware y software libres. Que el hardware sea libre quiere decir que Arduino ha dejado en manos de cualquier usuario la información suficiente para poder crear su propia placa. Y, que el software sea libre significa que cualquier usuario tiene información de las funciones y cómo programar. En su página web o incluso en otras páginas, se puede contar con

descripciones y tutoriales que ayudan al usuario a programar. Cuantas más funciones se conozca, más fácil es realizar el código deseado.

Por otra parte, el software consta de dos partes: el IDE (un entorno de desarrollo) y el bootloader (un sencillo programa al arrancar el Arduino).

El IDE (Integrated Development Environment) es el entorno de desarrollo integrado, es una herramienta imprescindible para poder escribir las instrucciones en su lenguaje y compilarlo para realizar las acciones deseadas, en otras palabras, será la ventana donde se escribirá el código programado.

El bootloader es un sencillo programa de arranque creado e incorporado por defecto. Sirve para poder ejecutar los programas que se suben. Si no se ha subido ningún programa nuevo, lo que hace es ejecutar el último programa almacenado. Por tanto, su función es la de controlar y almacenar los "sketches" que sube el usuario para poder ejecutarlos.

El software es descargado desde su página web. No obstante, se puede escribir, compilar y/o ejecutar el código online, con el Arduino Web Editor que se encuentra en la página oficial de Arduino, concretamente en https://create.arduino.cc/.

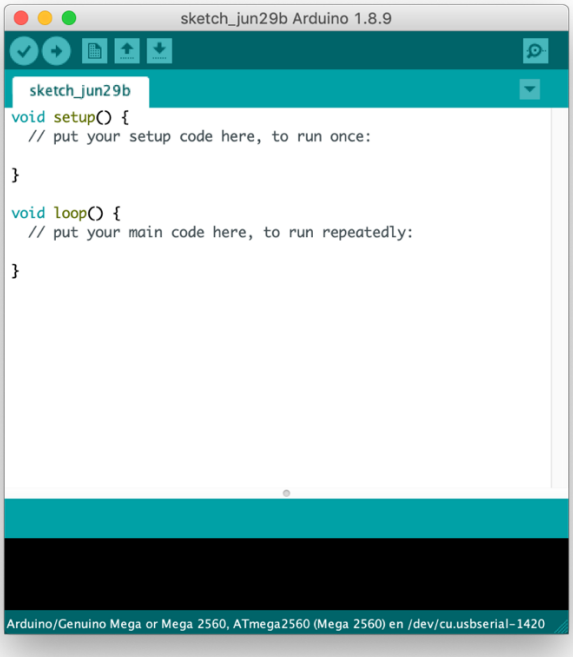


Ilustración 28. Aplicación de escritorio "Arduino"

Ilustración 29. Arduino Web Editor (online)

Como se observa en las imágenes anteriores, al abrir el programa, aparecen dos funciones por defecto: la función setup y loop.

Función setup(): Se llama a esta función al inicio del programa. Solo se llamará una vez, siempre que se pulse la opción de "Subir" (el botón de la flecha) o cada vez que se reinicie la placa de Arduino (con el botón de reset). Habitualmente, se usa para inicializar variables de forma global, configurar los modos de los pines (pinMode), iniciar la comunicación con un puerto serie para transmitir datos indicando su velocidad en baudios (Serial.begin(velocidad)), poner un mensaje inicial, etc.

Función loop(): Esta es la función donde se situará el código principal que se ejecutará en bucle, siempre que la placa se encuentre alimentada o hasta que se arranque otro sketch (programa). La repetición se producirá de forma instantánea, a no ser que se le incluya un retraso de los milisegundos que el usuario desee introducir (delay(milisegundos)). Aquí es donde se realizará el control PID, se hará la lectura del encoder rotativo, se mostrará repetitivamente los parámetros, etc.

4.2. Librerías

Son un conjunto de códigos ya escritos que facilitan la programación, es decir, que contienen las funciones necesarias programadas para comunicar con ciertos dispositivos y/o permitir las acciones deseadas. Esto supone un ahorro significante de tiempo a la hora de programar.

Se puede encontrar librerías ya incorporadas en la aplicación de Arduino (creadas por su propio equipo), se pueden descargar de otras páginas web (creadas por otras personas) o incluso el propio usuario las puede crear. Para ello, son necesarios dos ficheros: uno con el código (.cpp) y otro con

la información de lo que contiene la librería (.h), aunque también se pueden añadir otros ficheros opcionales como "keywords.txt" cuya función es colorear las palabras clave que se introduzcan en dicho fichero, esto puede ser muy útil visualmente y para saber si se están introduciendo correctamente las funciones incluidas en la librería. Todos estos ficheros se comprimen en formato .zip y ya pueden usarse.

La forma de introducirlas es sencilla, solamente hay que ir a la pestaña de *Programa*, ponerse sobre la opción de *Incluir Librería* y aparecen varias sub-opciones como *Administrar Bibliotecas*, donde se puede buscar dentro de un repertorio que tiene Arduino (sea una librería creada por su equipo o no), o *Añadir biblioteca .ZIP*, en el que se puede introducir una librería que haya sido descargada desde otra página o que haya sido creada por el propio usuario.

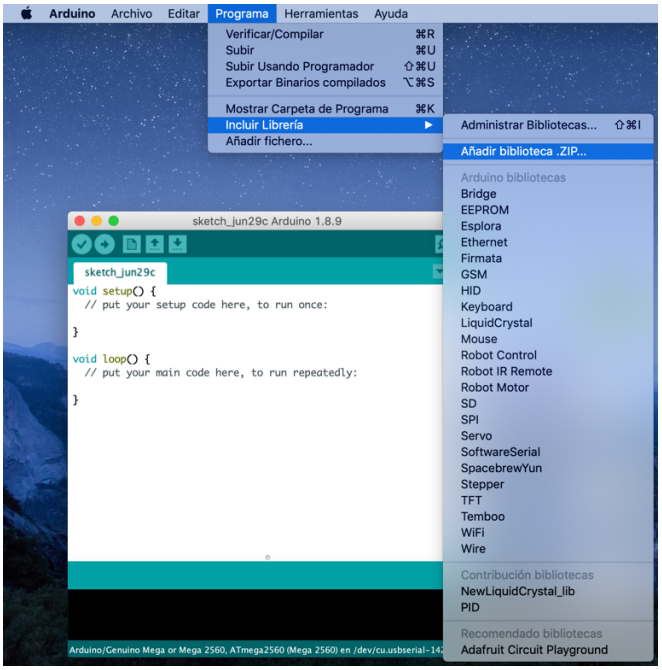


Ilustración 30. Pasos para introducir una librería .zip en Arduino

4.3. Código por partes

Debido a que a cada componente le pertenece un código distinto, a continuación, se explicará por partes todo el código programado.

Cabe destacar que, antes de realizar el código completo, se ha estado realizando varios tutoriales de cada componente por separado para saber cómo funcionan y qué funciones son útiles para lo que se quiere realizar. Tras ello, se ha configurado lo que es el código para este trabajo.

4.3.1. Instrucción "#include"

Para empezar, se ha incluido las librerías necesarias para poder ejecutar el código. Las tres primeras corresponden al funcionamiento de la pantalla LCD y la cuarta a la memoria:

```
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#include <EEPROM.h>
```

Ilustración 31. Librerías incluidas para el proyecto

- Wire.h: Permite la comunicación por los pines 20 (SDA) y 21 (SCL) en el caso del Arduino Mega 2560, que es el que está en uso en este trabajo.
- LCD.h: Ha sido descargada de internet. Es la librería que contiene varias funciones cuya utilidad es relevante a la hora de manejar la pantalla LCD, como por ejemplo la función de *clear()* que limpia la pantalla, es decir, que la deja en blanco.
- LiquidCrystal_I2C.h: Esta también ha sido descargada de internet a pesar de que hay una librería en Arduino llamada *LiquidCrystal*, la diferencia es que esta es una versión más actual. Sirve para configurar la pantalla LCD.
- EEPROM.h: Permite el almacenamiento de valores de los parámetros que el usuario desee. En el caso de la placa Arduino Mega 2560, puede guardar hasta 4 KB.

Por otra parte, antes de empezar con las funciones por defecto de la aplicación, se han inicializado algunas variables, de forma global: La configuración de la pantalla LCD, la asignación de los números de entradas/salidas a sus pines correspondientes, etc. Se ha creado una matriz creando el icono de una flecha, el cual será útil para el desplazamiento por el menú de la pantalla LCD. Se han configurado varias características del menú principal. También, se ha inicializado las variables para almacenar los valores necesarios de los parámetros deseados en la memoria EEPROM. Y, por último, se ha programado una función que contará las interrupciones que realiza el encoder interno del motor al girar, para averiguar su velocidad.

4.3.2. Funciones creadas

En este punto se explica de forma detallada el código. Es conveniente leer estas descripciones a la vez que se observa el anexo del código. No obstante, por la complejidad que puede suponer estar hojeando, se han incluido, en formato imagen, algunas partes del código correspondientes capturadas o al menos, las partes más relevantes de las funciones más extensas.

Boton leerBotones()

Para leer el encoder se ha creado esta función particular. Para empezar, se ha inicializado los estados de los pines mediante “LOW” y “HIGH”. Se ha realizado las lecturas de los pines con la función digitalRead().

```
/*Lee el movimiento del encoder*/

Boton leerBotones()
{
  static boolean oldDT = HIGH;
  static boolean pinDT = LOW;
  static boolean pinCLK = LOW;

  botonPres = Boton::Unknown;
  pinDT = digitalRead(DT);
  pinCLK = digitalRead(CLK);

  if ( !oldDT && pinDT )
  {
    botonPres = !pinCLK ? Boton::Right : Boton::Left;
    delay(50);
  }
  else if ( !digitalRead(SW) )
  {
    while (!digitalRead(SW));
    botonPres = Boton::OK;
    delay(50);
  }

  oldDT = pinDT;
  return botonPres;
}
```

Ilustración 32. Función leerBotones()

Con el primer *if*, se configura cuál es el modo de "Boton" en "Left" (giro antihorario) y cuál en "Right" (giro horario). De modo que, cuando se produce un cambio de flanco con el pin DT, se entra en el if y si el pin CLK todavía no ha producido un flanco de subida (que está en "LOW"), significa que el giro está siendo antihorario (Boton::Left), y será horario (Boton::Right) si ocurre lo contrario.

Con el segundo *if* (*else if* concretamente), se lee si se está pulsando el encoder o no. Como ya se ha comentado en el apartado de descripción del encoder, la señal siempre está en 1 cuando permanece en reposo y pasará a 0 cuando se presione. Por tanto, cuando se produzca un flanco de bajada el *botonPres* pasará al estado "Boton::OK".

Se les ha añadido cierto retraso para que al programa le de tiempo leer las señales y reaccionar.

void readConfiguration()

```
/*Lee y configura la memoria EEPROM*/
void readConfiguration()
{
  for ( int i = 0 ; i < sizeof(memory.d) ; i++  )
    memory.b[i] = EEPROM.read(i);

  if ( memory.d.initialized != 'Y' )
  {
    memory.d.initialized  = 'Y';
    memory.d.sentidomotor = 1;
    memory.d.rpmmotor     = 0;
    memory.d.milikp       = 0;
    memory.d.militi       = 0;
    memory.d.t            = 0;
    memory.d.pararmotor   = 0;
    writeConfiguration();
  }
}
```

Ilustración 33. Función readConfiguration()

Esta ha sido programada para poner en marcha la memoria EEPROM, dándole los valores por defecto a los parámetros que se usarán más adelante en caso de que no haya ningún valor anterior almacenado. No obstante, si hay datos almacenados, se leerá y adjudicará cada valor ya asignado anteriormente a su parámetro correspondiente. Será llamada cuando el usuario pulse la opción de solamente "Salir".

void writeConfiguration()

```
/*Escribe la memoria EEPROM*/

void writeConfiguration()
{
  for ( int i = 0 ; i < sizeof(memory.d) ; i++  )
    EEPROM.write( i, memory.b[i] );
}
```

Ilustración 34. Función writeConfiguration()

Esta función es la que sobrescribirá los valores elegidos por el usuario a través del menú. Será llamada cuando el usuario, tras modificar los valores que ha deseado, haya pulsado la opción de "Guardar y salir".

void abrirMenu()

En esta es donde se ha configurado el menú donde el usuario podrá modificar los parámetros que crea conveniente, que aparecerá cuando se presione el botón del encoder mientras se encuentra en la pantalla principal.

Se ha programado limpiar la pantalla en cuanto se llame a esta función y, mientras no se salga del menú, se leerán los botones del encoder para conocer qué opción se está eligiendo. La primera opción sería la 0, la segunda sería la 1 y así sucesivamente.

Para ello, se ha creado una variable llamada "idxMenu", cuyo valor siempre se encuentra dentro del rango 0-7 (el tamaño del menú), para contar cuántas posiciones se está desplazando. Por tanto, si se mueve el encoder hacia la "izquierda" (giro antihorario) se restará uno a la variable "idxMenu" y si se gira hacia la "derecha" (giro horario) se sumará uno al valor de dicha variable.

En cuanto "Boton" pase a "OK" querrá que decir que se ha pulsado el botón y, según el valor de "idxMenu", se abrirá un Submenú que corresponde a la opción deseada llamando a la función "openSubMenu" con 5 parámetros, en los dos últimos casos ("Guardar y salir" o "Salir") simplemente se volverá a la pantalla principal tras realizar la acción programada.

```
else if ( botonPres == Boton::OK )
{
  switch ( idxMenu )
  {
    case 0: openSubMenu( idxMenu, Screen::Flag, &memory.d.sentidomotor, 0, 1); break; //Configurar sentido motor
    case 1: openSubMenu( idxMenu, Screen::Number, &memory.d.rpmmotor, 0, 320); break; //Configurar RPM Referencia
    case 2: openSubMenu( idxMenu, Screen::Number, &memory.d.milikp, 0, 5000); break; //Configurar Kp
    case 3: openSubMenu( idxMenu, Screen::Number, &memory.d.militi, 0, 5000); break; //Configurar Ti
    case 4: openSubMenu( idxMenu, Screen::Number, &memory.d.t, 0, 200); break; //Configurar t
    case 5: openSubMenu( idxMenu, Screen::Flag, &memory.d.pararmotor, 0, 1); break; //Parar motor
    case 6: writeConfiguration(); salirMenu = true; break; //Guardar y salir
    case 7: readConfiguration();  salirMenu = true; break; //Salir (sin cambios)
  }
  forcePrint = true;
}
```

Ilustración 35. Instrucción de la función abrirMenu() al pulsar el encoder

También se ha configurado la impresión de los caracteres necesarios para conocer las opciones que existen. La posibilidad de desplazarse por la pantalla con el encoder ha sido gracias a los bucles for programados. Cabe destacar que la flecha creada al principio es utilizada en este menú, ayudando visualmente al usuario para saber qué elección pretende realizar.

```
if ( !salirMenu && (forcePrint || botonPres != Boton::Unknown) )
{
  forcePrint = false;

  static const byte endFor1 = (iMENU + filas - 1) / filas;
  int graphMenu     = 0;

  for ( int i = 1 ; i <= endFor1 ; i++ )
  {
    if ( idxMenu < i * filas )
    {
      graphMenu = (i - 1) * filas;
      break;
    }
  }

  byte endFor2 = graphMenu + filas;

  for ( int i = graphMenu, j = 0; i < endFor2 ; i++, j++ )
  {
    lcd.setCursor(1, j);
    lcd.print( (i < iMENU) ? MENU[i] : "                    " );
  }

  for ( int i = 0 ; i < filas ; i++ )
  {
    lcd.setCursor(0, i);
    lcd.print(" ");
  }
  lcd.setCursor(0, idxMenu % filas );
  lcd.write(idflechas);
}
```

Ilustración 36. Instrucción para la impresión de la función abrirMenu()

void openSubMenu(…)

Esta función es llamada desde el menú cuando se ha pulsado el encoder en alguna de las opciones con la intención modificar algún parámetro. Para facilitar el código de los submenús, al llamar a esta función, se reciben los 5 argumentos desde la función abrirMenu(), estas son:

```
void openSubMenu( byte menuID, Screen screen, int *value, int minValue, int maxValue )
```

Ilustración 37. Argumentos que recibe la función openSubMenu()

- byte menuID: El número de identificación del menú. Para saber a qué opción se ha accedido.
- Screen screen: Un enumerador tipo "Screen". Creada al principio del programa. Sirve para averiguar el tipo de valores que hay que mostrar en pantalla: "SI/NO" (Screen::Flag) o un número entero (Screen::Number).
- int *value: El valor correspondiente, según el caso, almacenado en la memoria.
- int minValue: El mínimo valor límite del parámetro.
- int maxValue: El máximo valor límite del parámetro.

El planteamiento del submenú es bastante parecido al menú que se acaba de describir. Lo primero que se hace al cambiar de menú es limpiar la pantalla y, mientras no se salga del submenú, se llamará a "leerBotones()", otra vez, para saber qué movimiento quiere realizar el usuario.

Si se gira en sentido antihorario se restará valor y, si se gira en sentido horario, se sumará. Ña cantidad que se resta o se añade dependerá del parámetro que se está modificando, esto se ha configurado mediante varios "if/else if".

Nuevamente, la última parte del código está enlazada a la impresión de los caracteres necesarios para crear este submenú. La diferencia respecto al menú anterior es que en este está asegurando que, según la opción seleccionada y gracias al enumerador Screen, se muestre los valores "SI/NO" o un valor numérico.

```
if ( screen == Screen::Flag )
{
  lcd.setCursor(columnas / 2 - 1, 1);
  lcd.print(*value == 0 ? "NO" : "SI");
}
else if ( screen == Screen::Number )
{
  lcd.setCursor(columnas / 2 - 1, 1);
  lcd.print(*value);
  lcd.print(" ");
}
```

Ilustración 38. Instrucción con el enumerador Screen

Cuando se pulse el encoder, se saldrá del submenú, ya que "exitSubMenu" es "true", por lo que se limpia la pantalla y se vuelve al menú anterior.

4.3.3. Función inicial

void setup()

Como ya se ha comentado antes, al ejecutar un programa, comienza con la función setup().

En esta, se ha llamado a la función que inicia el *Monitor Serie*, indicando también la velocidad con la que se transmiten los datos, en este caso son 57600 baudios (bits por segundo).

```
Serial.begin(57600);
```

Ilustración 39. Instrucción que inicia el Monitor Serie

Se ha configurado los modos de los pines según si son entradas o salidas.

```
pinMode(SW,  INPUT_PULLUP);
pinMode(DT,  INPUT_PULLUP);
pinMode(CLK, INPUT_PULLUP);

pinMode (ENB, OUTPUT);
pinMode (IN3, OUTPUT);
pinMode (IN4, OUTPUT);

pinMode (2, INPUT);
pinMode (3, INPUT);
```

Ilustración 40. Configuración de modos de los pines

Se ha incluido la función *attachInterrupt()*, en la que cada vez que exista una interrupción en el encoder interno del motor, llamará a la función *cuenta()* encargada de contar las interrupciones producidas, algo necesario para calcular la velocidad del motor, para ello también se han conectado los cables encargados de transmitir esas señales de interrupción a los pines 2 y 3, dos de los cuales están habilitados para esta función, según la descripción de *attachInterrupt()* en la página web de Arduino.

```
attachInterrupt(digitalPinToInterrupt(2), cuenta, RISING);
```

Ilustración 41. Instrucción attachInterrupt()

```
void cuenta() {
  if (digitalRead(3) == HIGH)
    valor_cuenta++;
  else
    valor_cuenta--;
}
```

Ilustración 42. Función cuenta()

Se ha cargado la configuración de la memoria EEPROM con la función *readConfiguration()*.

Finalmente, se ha iniciado la pantalla LCD, se ha creado el símbolo de la flecha, se ha introducido un mensaje de inicio que estará por poco tiempo (cuestión de milisegundos) y la función acaba con limpiar la pantalla para ser rellenada con los siguientes caracteres que se formarán en las siguientes órdenes.

```
// Inicia el LCD
lcd.begin(columnas, filas);
lcd.createChar(idflechas, flechas);

lcd.setCursor(0, 0);
for ( int i = 0 ; i < columnas ; i++ )
{
  lcd.print(".");
  delay(100);
}
lcd.clear();
```

Ilustración 43. Inicialización de la pantalla LCD e impresión de un mensaje de inicio

4.3.4. Función en bucle

void loop()

Ahora para explicar mejor la parte de la función principal *loop()*, hay que fraccionarla en distintos trozos de código:

Para empezar, se han inicializado las variables principales del control PID y se ha creado la variable t0 para contar los milisegundos que ha tardado desde que empezó el bucle, con la función *millis()*.

Después, para leer la intensidad se han declarado las variables correspondientes, 3 *float* exactamente. Al primero, llamado "Sensibilidad", se le ha asignado un valor de 0.185 V/A ya que así lo indica las especificaciones del sensor. Al segundo, llamado "voltajeSensor" se le ha adjudicado la expresión que realiza la lectura analógica correspondiente al sensor y lo convertirá a un rango comprendido en [0,5] Voltios. Con el tercer *float*, se ha creado la variable "I" que será igual a la ecuación que obtendrá la corriente, ya comentada anteriormente de forma más detallada.

```
float Sensibilidad = 0.185; //sensibilidad en Voltios/Amperio para sensor de 5A
float voltajeSensor = analogRead(A3) * (5.0 / 1023.0); //lectura del sensor
float I = (voltajeSensor - 2.5) / Sensibilidad; //Ecuación  para obtener la corriente
```

Ilustración 44. Instrucciones para leer la intensidad

Tras ello, se ha impreso por pantalla. La forma en que se hace se explicará más adelante juntamente con las demás variables a imprimir.

A continuación, se han recuperado los datos de la memoria EEPROM para darles valor a algunas de las variables recientemente creadas. Cabe destacar que "rpm_ref" (el valor de la velocidad de referencia introducida por el usuario) será negativo si se ha configurado que no se quiere el sentido horario en el menú, y será positivo en caso contrario.

```
sentido = memory.d.sentidomotor;
rpm_ref = (sentido == 0 ? -1 : 1 ) * memory.d.rpmmotor;
milikp = memory.d.milikp;
militi = memory.d.militi;
t = memory.d.t;
stopmotor = memory.d.pararmotor;
```

Ilustración 45. Instrucciones para la recuperación de datos de la memoria EEPROM

También se ha declarado la variable *botonPres* para *leerBotones()* para averiguar en qué estado se encuentra el encoder. Si el estado es que el *Boton* es *OK*, significa que se ha pulsado el encoder entonces se abrirá el Menú.

Como ya se ha comentado anteriormente, para leer la velocidad medida es necesaria la función "void cuenta()", tras contar las interrupciones, se traduce a revoluciones por minuto con la siguiente expresión: rpm_medida = (valor_cuenta)*182/T. El contador de valor_cuenta se pone a 0 nada más acaba la conversión a rpm_medida para una correcta lectura.

Seguidamente, se ha colocado el código de la implementación del PID. El primer paso consiste en calcular el error con los datos necesarios, este resulta de la diferencia entre la velocidad de referencia y la velocidad medida. Luego, se introduce la expresión del PID, con la implementación digital que se ha comentado en su apartado correspondiente.

En el código, de una forma resumida, la componente proporcional resulta ser: "Up = Kp * e" y la integral: "Ui = Ui0 + (Kp*T/Ti)*e". Al escribirlo en Arduino se ha adaptado según las variables creadas.

```
e = rpm_ref - rpm_medida;
up = (milikp / 1000.0) * e;
ui = ui0 + (((float)milikp) / (militi)) * (T / 1000.0) * e;
```

Ilustración 46. Implementación del PID

Por otra parte, como Ti está en el denominador y cabe la posibilidad de que sea 0, en este caso, se ha programado que Ui sea 0 también, evitando así problemas matemáticos con un denominador nulo.

```
if (militi == 0) {
  ui = 0;
}
```

Ilustración 47. Corrección de la componente ui

El control que se manda al Arduino es la suma de las dos componentes anteriores: U = Up + Ui. Si este es mayor que 0, el motor girará en sentido horario y, si es menor que 0, girará en sentido antihorario.

```
control = up + ui;
if (control > 0) {
  digitalWrite (IN3, HIGH);
  digitalWrite (IN4, LOW);
}
else {
  digitalWrite (IN3, LOW);
  digitalWrite (IN4, HIGH);
}
```

Ilustración 48. Instrucciones para mandar el control al controlador de motor

Posteriormente, se ha limitado el valor de rpm_salida, aquella que será transmitida de forma "analógica" (mediante señal PWM) al controlador para que mueva y regule el motor. Como el valor máximo de la función analogWrite() es 255 (máximo voltaje), ese ha sido el valor máximo al que se le ha acotado. Por el contrario, el valor mínimo de dicha función es 0 (voltaje nulo), que se le asignará a "rpm_salida" cuando el usuario haya elegido la opción de "Parar motor" o la velocidad de referencia elegida sea nula.

```
rpm_salida = fabs(control) > 255 ? 255 : fabs(control);

if ( stopmotor == 1) {
  rpm_salida = 0;
  memory.d.rpmmotor = 0;
  writeConfiguration();
}
analogWrite(ENB, rpm_salida);
```

Ilustración 49. Transmisión del control mediante la señal PWM

Para programar cómo plasmar los valores de las velocidades de referencia y medida en la pantalla principal, se ha tenido que llamar a la función "sprintf" que imprime los caracteres que se desean además de valores no fijos de esas variables.

En el caso de la velocidad de referencia, "referencia" es el texto que se quiere plasmar, "rpm_ref" es el valor que se escribirá en lugar de "%3d" y "buf" es la matriz donde se almacenará toda la frase. Tras esto, se transmite a la pantalla LCD con "lcd.print(buf)", indicando con "lcd.setCursor()" la posición en la que se debe encontrar el principio de la frase.

```
static char buf[10];
char* referencia = "REF %3d ";
sprintf(buf, referencia, rpm_ref);
```

Ilustración 50. Instrucciones para la impresión del texto de la velocidad de referencia

Exactamente lo mismo ocurre con la velocidad medida, básicamente sería cambiar la palabra "referencia/ref" por "medida".

Algo similar ocurre a la hora de querer plasmar la intensidad, aunque existe la diferencia de que esta es una variable tipo float, ya que se pretende leer la corriente con 3 decimales. Al ser así, se

ha de crear otra matriz de tipo char para guardar el valor en formato texto y llamar a la función dtostrf() para realizar la conversión de float a char. Tras la conversión, el proceso de mostrarlo en pantalla es igual que con las velocidades.

```
char buf3[16];
char valorI[6];
dtostrf(fabs(I), 1, 3, valorI);
char* intensidad = "I = %s A     ";
sprintf(buf3, intensidad, valorI);
```

Ilustración 51. Instrucciones para la impresión del texto de la intensidad

Para que estos valores no parpadeen tan rápidamente en la pantalla, se ha configurado un método de retrasar la impresión con la variable “slow”. Esto hace que se llame a la función “lcd.print()” correspondiente solamente después de haber recorrido 5 bucles.

```
if (slow == 0) {
  lcd.setCursor(8, 0);
  lcd.print(buf2);

  lcd.setCursor(0, 0);
  lcd.print(buf);

  lcd.setCursor(0, 1);
  lcd.print(buf3);
  slow = 5;
}
slow--;
```

Ilustración 52. Instrucciones para retrasar la impresión de parámetros

Tras realizar todas estas acciones, se lee otra vez el tiempo que ha pasado en milisegundos con la función “millis()” y se guarda en la variable “t1”. Se calcula la diferencia entre “t1” y “t0” para saber cuánto tiempo le ha costado al programa ejecutar esas funciones (dt) y se le resta al periodo de muestreo (T) seleccionado en la función de “delay()”. A pesar de que dt es un valor muy bajo (alrededor de 1), de esta forma será más preciso, siempre que dt<T.

```
t1 = millis();
dt = t1 - t0;

if (dt < T) {
  delay(T - dt);
}
else if ( dt >= T) {
}
```

Ilustración 53. Instrucciones para introducir un periodo al bucle

5. RESULTADOS OBTENIDOS

En este apartado se va a comentar la sintonización que se ha realizado en este trabajo.

Aunque el objetivo de este trabajo simplemente era crear una interfaz donde el usuario pueda controlar cualquier otro motor de continua que quiera enlazar, incluyendo los parámetros que crea convenientes, también se ha realizado un ajuste manual (empírico) para el motor usado en este trabajo, para conocer qué constantes son lo suficientemente idóneos y confirmar que el controlador funciona.

Se van a realizar las siguientes pruebas con un punto de consigna de 100 rpm y un periodo (T) de 50 ms.

El primer paso, como se ha indicado teóricamente en el apartado correspondiente al método de ajuste de Ziegler-Nichols, es aumentar la Kp hasta observar que el motor oscila constantemente, sin llegar al punto de consigna y sin que las variaciones se incrementen. Con el motor usado en este trabajo, esa Kp, que será la ganancia crítica, son 4,850.

Por tanto, $Kcr = 4{,}850$.

Mediante el Monitor Serie de Arduino, se ha recopilado los datos de la velocidad medida. Con el programa Microsoft Excel, se ha construido la gráfica que nos proporcionará el periodo de oscilación crítico (Pcr).

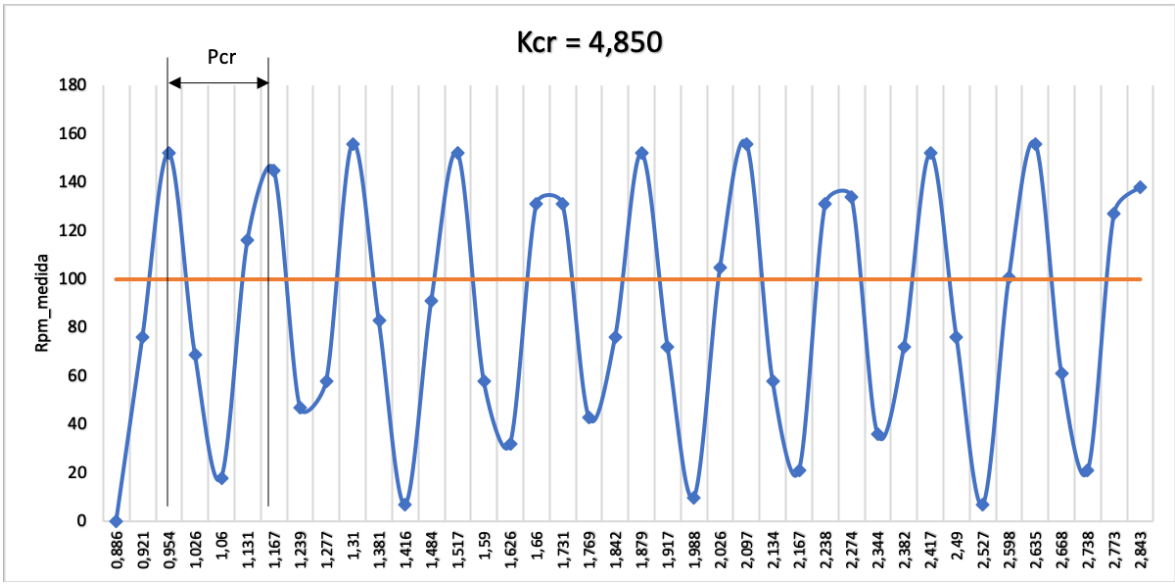


Ilustración 54. Gráfica para obtener el periodo de oscilación crítico

Como se observa, el periodo entre cada pico de dos oscilaciones consecutivas es de:
$1{,}166 - 0{,}954 = 0{,}212\ segundos$.

Por tanto, $Pcr = 212\ milisegundos$.

Sustituyendo dichos valores en la tabla 1 donde se muestran las expresiones necesarias para obtener las constantes proporcional e integral (PI), se tiene que:

$$Kp = 0{,}45 \cdot Kp = 0{,}45 \cdot 4{,}850 = 2{,}1825 \approx 2{,}200.$$

$$Ti = \frac{Pcr}{1{,}2} = \frac{212}{1{,}2} = 176{,}\hat{6} \approx 180\ milisegundos.$$

Por lo tanto, se probará con introducir en "Configurar Kp" el valor de 2200 y en "Configurar Ti" el valor de 180. Resultando de la siguiente manera:

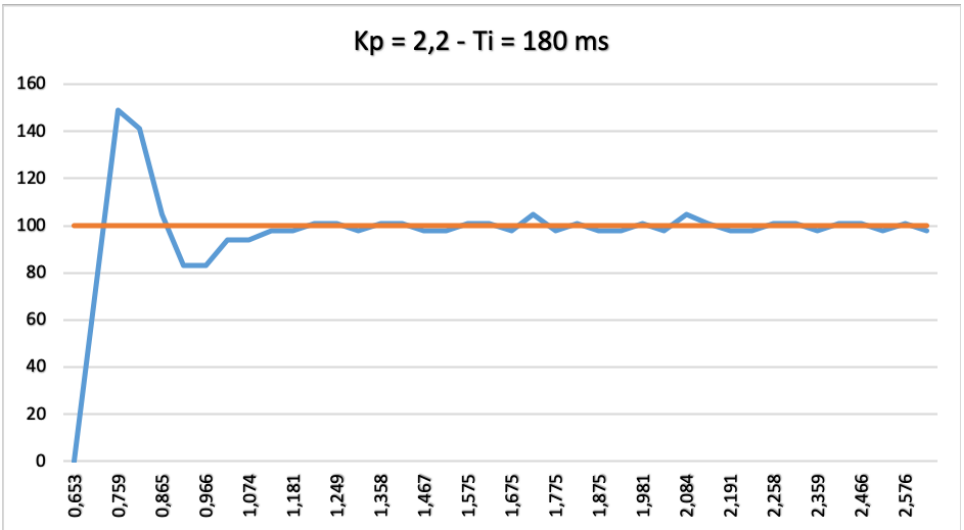


Ilustración 55. Gráfica con el resultado del control con Kp = 2,2

Se observa que se llega a la velocidad de consigna de una forma rápida, pero tal vez no de una forma muy estable. Por ello, se reduce el Kp a la mitad para ver cómo reacciona (como ya se ha comentado, a mayor Kp el proceso es más inestable). El Ti no se modifica ya que parece corregir bastante bien el error.

Probando con Kp = 1,1 y manteniendo el Ti = 180 ms (se introduce 1100 en Kp y 180 en Ti), el resultado es:

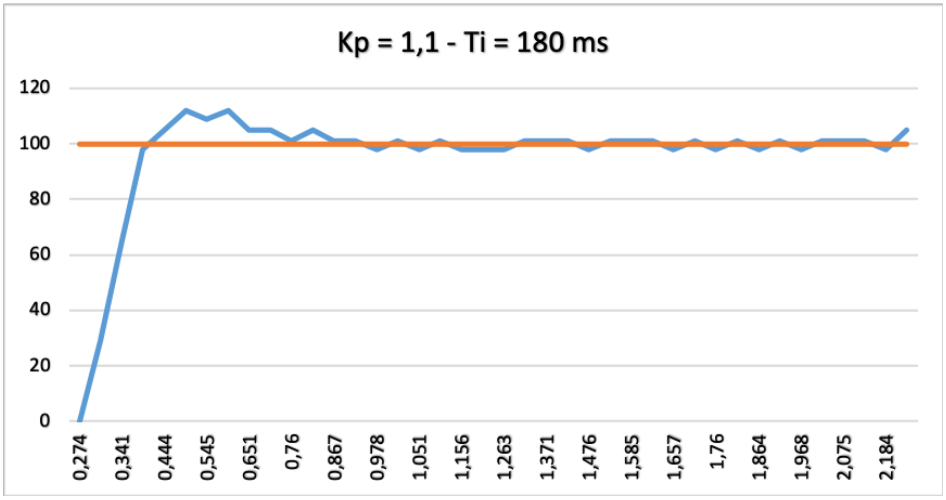


Ilustración 56. Gráfica con el resultado del control con Kp = 1,1

Se puede ver que ahora se ha llegado a la velocidad de consigna con un arranque un poco más lento, pero también, con una apariencia mucho más estable.

Por tanto, el valor de la constante proporcional resultante (Kp) estaría entre 1,1 y 2,2. Según las especificaciones que se quieran buscar, será un valor más grande o más pequeño. Y los valores más adecuados del tiempo integral (Ti) y el periodo (T) son 180 ms y 50 ms, respectivamente.

En conclusión, se ha ajustado manualmente de una manera rápida y que, aunque tal vez se pueda realizar con más exactitud mediante un ajuste analítico, es bastante aceptable para este motor.

6. MANUAL DEL USUARIO

Todo el apartado 4 (“Programación”) ha sido como un manual para que un programador pueda desarrollar el mismo código que el realizado en este trabajo. Por ello, también se ha redactado este manual o guía para que cualquier usuario pueda hacer uso de este sistema de una forma más rápida y sencilla.

Para el funcionamiento de este proyecto, solo se necesita un ordenador con puerto USB, una toma de corriente y seguir los siguientes pasos:

Nota: Puede haber diferencias entre las imágenes incorporadas respecto a su funcionamiento, debido al posible distinto sistema operativo usado.

1. Descargar aplicación de Arduino. En la propia página de Arduino se puede descargar dicha aplicación. Concretamente en: https://www.arduino.cc/en/Main/Software. Se selecciona en qué sistema operativo se quiere descargar y listo, solo hay que seguir los pasos que te indican al pinchar sobre el zip descargado. Otra forma también sería trabajar online en el “Arduino Web Editor” (https://create.arduino.cc/), aunque es recomendable la aplicación.

2. Una vez descargada la aplicación, conectar el cable USB de color azul al Arduino y a un puerto USB del ordenador. Y también, introducir el adaptador negro a un enchufe con toma de corriente.

3. En la pestaña de "Herramientas", comprobar que aparece el puerto conectado en Arduino y que la placa y el procesador seleccionados son los correctos. En caso contrario, clicar sobre los correspondientes. De forma que quede de la siguiente manera (el nombre del puerto no tiene porqué ser igual):

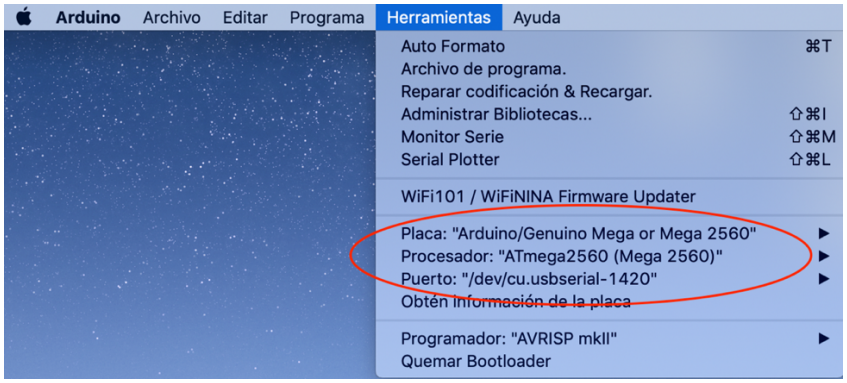


Ilustración 57. Configuración de la placa, procesador y puerto en Arduino

4. Abrir el programa e incluir la librería de "NewLiquidCrystal_1.5.1." (en caso de que no esté incluida) descargada en https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads/). Se recuerda que la forma de incluirla es:

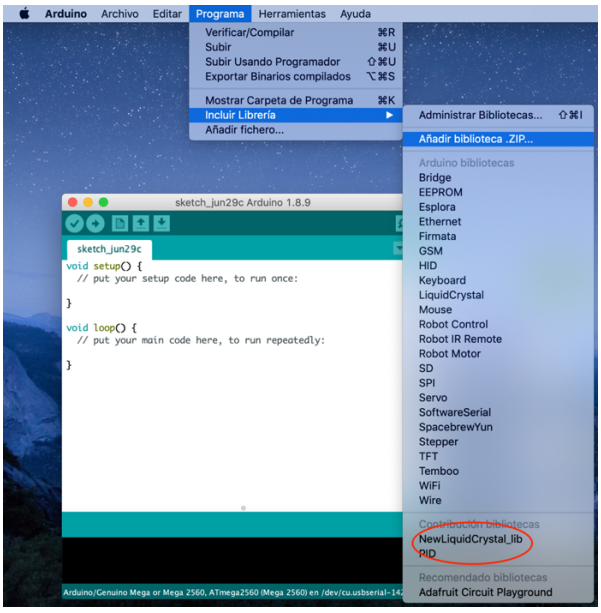


Ilustración 58. Pasos para incluir una librería en Arduino

Cuando esté incluida, aparecerá de la misma forma que en la imagen.

5. Compilarlo para asegurarse que no hay error. Es la opción que contiene un "tic":

Ilustración 59. Botón de Compilar/Verificar

6. Una vez compilado sin errores, subir/ejecutar el programa. Es clicar sobre el botón que contiene una flecha:

Ilustración 60. Botón de Ejecutar/Subir

7. Se encenderá la pantalla, aparecerá un mensaje durante un breve instante, y seguidamente, en la pantalla se verá la intensidad cambiar junto con los valores de velocidad de referencia y velocidad medida:

Ilustración 61. Pantalla inicial

8. Pulsar encoder para entrar en el menú. Se observarán las distintas opciones configurables. Para realizar modificaciones, con la ayuda de una flechita que aparece, se va girando el encoder hasta la elección deseada y se pulsa otra vez. Entonces, aparecerá un submenú donde se cambiarán los valores. Los distintos parámetros y sus valores cambiables son:

   - "Sentido horario" (SI/NO). "SI" (Giro horario), "NO" (Giro antihorario).
   - "Velocidad motor" (Entre 0 y 320). En revoluciones por minuto.
   - "Configurar Kp" (Entre 0 y 5000). Cabe destacar que este valor se dividirá entre 1000, es decir, que el rango de Kp está entre 0 y 5.
   - "Configurar Ti" (Entre 0 y 5000). En milisegundos.
   - "Configurar t" (Entre 0 y 200). En milisegundos.
   - "Parar motor" (SI/NO). Sirve para poner el motor a 0 rpm. Se ve afectado en el valor de velocidad de referencia, es decir, que, para volver a conectar el motor, hay que ponerle un valor de nuevo en la opción de "Velocidad motor".
   - "Guardar y salir". Se clica esta opción si se quiere guardar los cambios realizados.
   - "Salir". Se sale del menú sin guardar los cambios realizados.

Los valores predefinidos de las configuraciones son:

- Sentido horario: SI
- Velocidad motor: 0
- Configurar Kp: 1100
- Configurar Ti: 180
- Configurar T: 50
- Parar motor: NO

Posibles fallos

Podría pasar que, a pesar de seguir todas las instrucciones comentadas, no se haya alcanzado con éxito el proceso. Esto puede ser por varias razones. Comprobar:

- Si las luces LED de todos los dispositivos están encendidos. Si no lo están, es que hay una mala conexión del USB o del adaptador de corriente.
- Si se ha conectado bien el puerto USB con la revisión comentada en la *Ilustración 57.*
- Si hay algún fallo de conexión en el cableado. Tirar suavemente de los cables a ver si hay alguno suelto, presionar sobre los conectores para que encajen mejor con los pines y/o apretar más los tornillos que sujetan los cables.
- Si no compila. Esto puede resultar si el software no está conectado. En el caso de la aplicación online, el icono del "Arduino Web Editor" ha de estar de color negro, en caso contrario, hay que clicar sobre él y darle a "Open Plugin". Para la aplicación descargada, basta con comprobar la conexión del puerto.

Ilustración 62. "Open plugin"

7. DISEÑO DE LA CAJA

A medida que el proyecto estaba finalizando y ya estaban todos los componentes fijados en un sitio concreto de la chapa, se ha diseñado una caja con el programa Inventor para poder introducir todo el montaje en ella.

En la ilustración 64 se refleja cómo ha quedado conjuntamente y en los planos se verá con más detalle el boceto de todos los componentes descritos.

De esta forma, el sistema es más fácil de transportar y seguro ya que, tanto el cableado como los dispositivos, quedan más resguardados. Además de quedar mucho mejor estéticamente.

La caja consiste en:

- Una base: donde se introduce y se apoya el montaje. Contiene 4 agujeros en sus laterales. Uno sirve para poder conectar el USB al Arduino, otro para conectar otra vía de alimentación del Arduino, otro para poder pasar el cable que alimenta el controlador y otro para poder observar el motor.
- Una tapa: para proteger los elementos. Contiene 2 aberturas. Una para ver la pantalla LCD y otra para poder interactuar con el encoder rotativo.
- Una rueda para el encoder: para introducirlo en el pulsador del encoder y que sea más estético y fácil de manejar.
- Una pieza que sostiene el encoder: sirve para poder introducir el encoder dentro de la base y que se mantenga a una cierta altura, para así poder girar y apretar el pulsador con más comodidad.

Al finalizar el diseño, se ha procedido a imprimirlo en 3D. En concreto, se ha usado la impresora 3D "Original Prusa i3 MK2.5". Para llevarlo a cabo, se ha tenido que usar el programa PrusaSlicer, descargado en Internet (Prusa Research, 2019), para convertir los archivos generados en "Inventor" (.stl) a otro tipo de archivo que la impresora pueda leer (.gcode).

El filamento usado es PLA. Sean Rohringer (s.f.) lo define como:

> El ácido poliláctico (PLA) es un termoplástico biodegradable, hecho a base de recursos renovables como el almidón de maíz o la caña de azúcar. Aparte de usarse en la impresión 3D, lo podemos ver principalmente en implantes médicos, envases de alimentos y vajillas desechables. El principal beneficio que presenta el filamento PLA es que es fácil de extruir.

Ilustración 63. Impresora Prusa 3D

La forma de la caja tendrá un aspecto similar a la de la imagen siguiente:

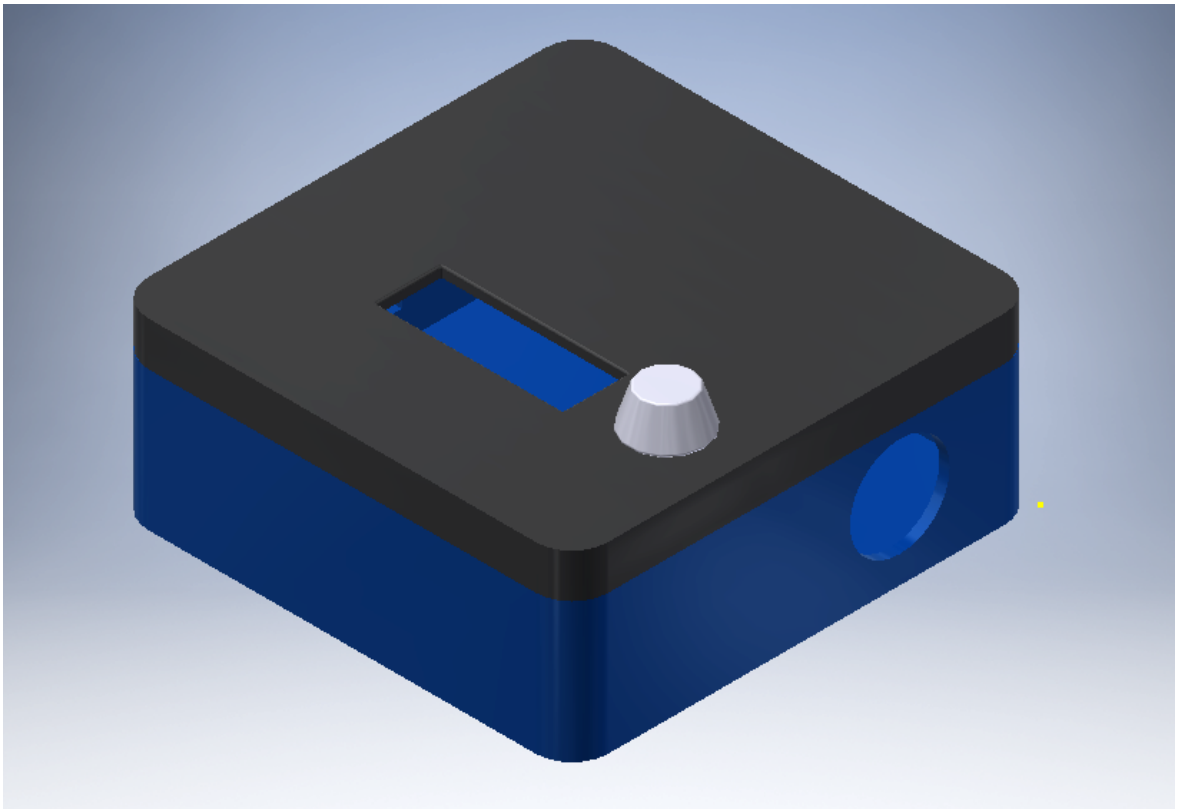


Ilustración 64. Captura de la caja diseñada con Inventor

8. CONCLUSIONES

Para finalizar, se va a proceder a comentar las conclusiones que se han podido deducir de este trabajo.

Primero, se observa si se ha cumplido con los objetivos que se ha decidido perseguir al principio. Recordemos que eran los siguientes:

- Conocer el manejo del programa Arduino y varias de sus funciones básicas.
- Diseñar una estructura, con todos los componentes necesarios, con la que el usuario pueda comunicarse con el Arduino e interpretar los resultados de una manera sencilla física y visualmente.
- Desarrollar un controlador PID digital, con su previa conversión de la ecuación analógica a digital, y sintonizarlo de manera aceptable para su funcionamiento.
- Conseguir que el motor DC alcance la velocidad de referencia deseada de manera rápida y suave (idealmente sin oscilaciones).
- Diseñar una caja para englobar todo y proporcionar más estética al montaje final.

La programación ha resultado y sus funciones han resultado correctas. El sistema es fácil de manejar y comprender. El PID, a pesar de que puede ser mejorable, funciona bastante bien con el motor usado y alcanza la velocidad de consigna adecuadamente. Y, por último, la caja está diseñada de manera acertada con sus justas medidas. Se puede afirmar, por tanto, que se han conseguido todos los objetivos.

Cabe señalar también que, a pesar de tener cierta base de lenguaje de programación, gracias a las asignaturas de “Informática” y “Tecnología Informática Industrial”, en este trabajo se ha aprendido mucho más todavía.

Saber manejarse ante los fallos de programación al final tiene un resultado favorecedor tanto a nivel académico como particular, a pesar de toda la angustia que pueda provocar.

Personalmente, creo que es un proyecto interesante, ya que engloba muchos campos y he aprendido mucho más de cada uno de ellos.

Al principio, no sabía ni explicar lo que podía hacer un Arduino o para qué servían las señales PWM, o las de reloj, por ejemplo. O incluso no saber cuándo usar un motor de continua. Por no mencionar la satisfacción que se obtiene al ver funcionar un controlador PID implementado en una aplicación real.

Además, ha sido todo un placer poder realizar este trabajo, ya que la programación y el control son dos ámbitos que me gustan mucho, por lo que ha sido didáctico a la vez que entretenido y ameno. Tanto despertaron mi interés por ellos que la especialización que deseo realizar en el máster es el de “Control de procesos, automatización y robótica”.

9. BIBLIOGRAFÍA

Blasco, F. X., Martínez, M. A., Senent, J. S., Sanchis, J. (2000). *Sistemas automáticos*. Valencia: Universidad Politécnica de Valencia.

Serrano, L., Martínez, J. A. (2017). *Máquinas eléctricas* (4ª ed.). Valencia: Universidad Politécnica de Valencia.

Arduino (2019). Recuperado 14 marzo 2019, de https://www.arduino.cc/

Controlador PID (2019). Recuperado 2 abril 2019, de https://www.picuino.com/es/arduprog/control-pid.html

Método de Ziegler-Nichols (2019). Recuperado 22 mayo 2019, de https://www.picuino.com/es/arduprog/control-ziegler-nichols.html

Herrero, J.M. (2013) *Unidad Didáctica 6: Realimentación y control realimentado.* Manuscrito no publicado, Departamento de Ingeniería de Sistemas y Automática, Universidad Politécnica de Valencia, Valencia, Comunidad Valenciana.

Electronilab (2014) *Tutorial: Uso de Driver L298N para motores DC y paso a paso con Arduino.* Recuperado 4 abril 2019, de https://electronilab.co/tutoriales/tutorial-de-uso-driver-dual-l298n-para-motores-dc-y-paso-a-paso-con-arduino/

TranThinh (s.f.) *Motor with Encoder, How to Read Input Value From Encoder*. Recuperado 15 abril 2019, de https://www.instructables.com/id/Motor-With-Encoder-How-to-Read-Input-Value-From-En/

Naylamp Mechatronics (s.f.) *Tutorial sensor de corriente ACS712*. Recuperado 20 abril 2019, de https://naylampmechatronics.com/blog/48_tutorial-sensor-de-corriente-acs712.html

Dejan (s.f.). *How Rotary Encoder Works and How To Use It with Arduino.* Recuperado de https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/

Prusa Research (2019). Drivers y manuales. Recuperado 12 junio 2019, de https://www.prusa3d.es/drivers-y-manuales/

Sean Rohringer (s.f.). PLA vs ABS: comparación de filamentos para impresión 3D. Recuperado 30 junio 2019, de https://all3dp.com/es/filamento-abs-filamento-pla-comparacion-impresion-3d/

DOCUMENTO II: PRESUPUESTO

ÍNDICE DEL PRESUPUESTO

1. Introducción

En el presente documento se va a reflejar el presupuesto de todo el Trabajo Final de Grado.

Primero, se va a calcular el precio de mano de obra, tanto el de las horas dedicadas por el ingeniero industrial (tutor) como el de las del ingeniero técnico en prácticas (estudiante).

Después, se reflejarán los costes de los materiales necesarios para fabricar el sistema en sí y también los costes de las herramientas, proporcionadas por la Universidad, para el montaje del proyecto.

Finalmente, se hará el presupuesto final con la suma de todas las partes, incluyendo también los gastos generales, el beneficio industrial y el IVA. Indicando sus desgloses correspondientes.

2. Mano de obra

En este apartado, se va a calcular el coste que supone las horas de trabajo empleadas en el proyecto. Para ello, se van a hacer las siguientes suposiciones:

- El ingeniero industrial (tutor de la UPV) tiene un salario anual de alrededor 50.000€/año.
- El ingeniero técnico en prácticas (estudiante que realiza el Trabajo Final de Grado) tiene un salario anual aproximadamente de 20000€/año.
- Un año tiene 250 días laborables y cada día se trabaja 8 horas.

Realizando los cálculos correspondientes se hallará el coste por hora de cada uno:

$$50.000\ \frac{€}{año} \cdot \frac{1\ año}{250\ días} \cdot \frac{1\ día}{8\ horas} = 25€/h$$

$$20.000\ \frac{€}{año} \cdot \frac{1\ año}{250\ días} \cdot \frac{1\ día}{8\ horas} = 10€/h$$

Resumiendo, se tiene que:

	Coste (€/h)
Ingeniero industrial	25
Ingeniero técnico en prácticas	10

Tabla 1. Coste de mano de obra

Sabiendo el coste de cada ingeniero, se va a proceder a reflejar las horas dedicadas, por el estudiante en este trabajo, en el siguiente desglose:

	Horas dedicadas (h)
Reuniones	20
Planificación	10
Búsqueda de información	40
Montaje	30
Diseño del sistema	20
Programación	80
Diseño de la caja	40
Elaboración de documentos	130
Total	**370**

Tabla 2. Horas empleadas por el estudiante

Se supone que el tutor dedicará las horas correspondientes a las reuniones.

Por tanto, el presupuesto de la mano de obra será:

Unidad	Descripción	Cantidad	Precio	Importe
h	Ingeniero industrial	20	25	500
h	Ingeniero técnico en prácticas	370	10	3.700
			Total	**4.200**

Tabla 3. Presupuesto parcial de Mano de Obra

3. Costes materiales de fabricación

A continuación, se detallarán solo los costes de los materiales usados para la creación del sistema de este proyecto en concreto.

Cabe decir que, para la caja, que se imprimirá en 3D, se usará cierta cantidad de filamento PLA. Esta la calcula el programa "Prusa Slicer" cuando se convierten los archivos generados en "Inventor" a otro tipo de archivos para que la impresora 3D los pueda leer. La cantidad usada es de 353,34 gramos, que son 0,35334 kilogramos.

Por otra parte, también se hará uso de una plancha de metacrilato, de tamaño 0,2 x 0,2 m_2.

Comentado esto, se procede a calcular el presupuesto de este apartado:

Unidad	Descripción	Cantidad	Precio	Importe
ud	Arduino MEGA 2560 + cable USB	1	7,99	7,99
ud	Controlador L298N	1	14,22	14,22
ud	Motor de continua	1	13,9	13,9
ud	Pantalla LCD + Adaptador IIC/I2C	1	3,95	3,95
ud	Encoder rotativo	1	1,68	1,68
ud	Arduino shield v5	1	2,09	2,09
ud	Pack de cables de puente	1	1,16	1,16
m_2	Plancha de metacrilato	0,04	17,60	0,71
ud	Adaptador de corriente	1	2,73	2,73
kg	PLA	0,35334	24,99	8,83
			Total	57,26

Tabla 4. Presupuesto parcial de Materiales

4. Costes de las herramientas para el montaje

En este apartado se especificarán los costes de la maquinaria usada para montar el sistema. Se calcula de forma separada al punto anterior ya que no han sido herramientas con el único propósito de realizar este trabajo, sino que ya estaban disponibles en la Universidad. Se podría decir que es el coste de inversión para este proyecto y otros futuros.

Se calcula el presupuesto de inversión en maquinaria:

Unidad	Descripción	Cantidad	Precio	Importe
ud	Impresora 3D Prusa MK 2.5	1	769	769
ud	Taladro de columna	1	129	129
ud	Lote de tornillos (3mm)	1	3,98	3,98
ud	Destornillador de tuercas	1	2,78	2,78
ud	Lote de tuercas (3mm)	1	1,82	1,82
ud	Destornillador pequeño	1	0,25	0,25
ud	Lote llaves Allen	1	2,36	2,36
			Total	909,19

Tabla 5. Presupuesto parcial de Maquinaria

5. Presupuesto total

Ahora se calcula todo el presupuesto conjunto. Se suman todos los presupuestos parciales anteriores y con ello, se obtiene el Presupuesto de Ejecución Material del proyecto. Si a ello se le suman los gastos generales (13%) y el beneficio industrial (6%), se obtiene el Presupuesto de Ejecución por Contrata. Y, por último, se le añade el IVA (21%), resultando así el Presupuesto Base de Licitación:

Presupuesto parcial de Mano de Obra	4.200
Presupuesto parcial de Materiales	57,26
Presupuesto parcial de Maquinaria	909,19
PRESUPUESTO EJECUCIÓN MATERIAL	5166,45
Gastos Generales (13%)	671,64
Beneficio Industrial (6%)	309,99
PRESUPUESTO EJECUCIÓN POR CONTRATA	6148,08
IVA (21%)	1291,10
PRESUPUESTO BASE DE LICITACIÓN	7439,18

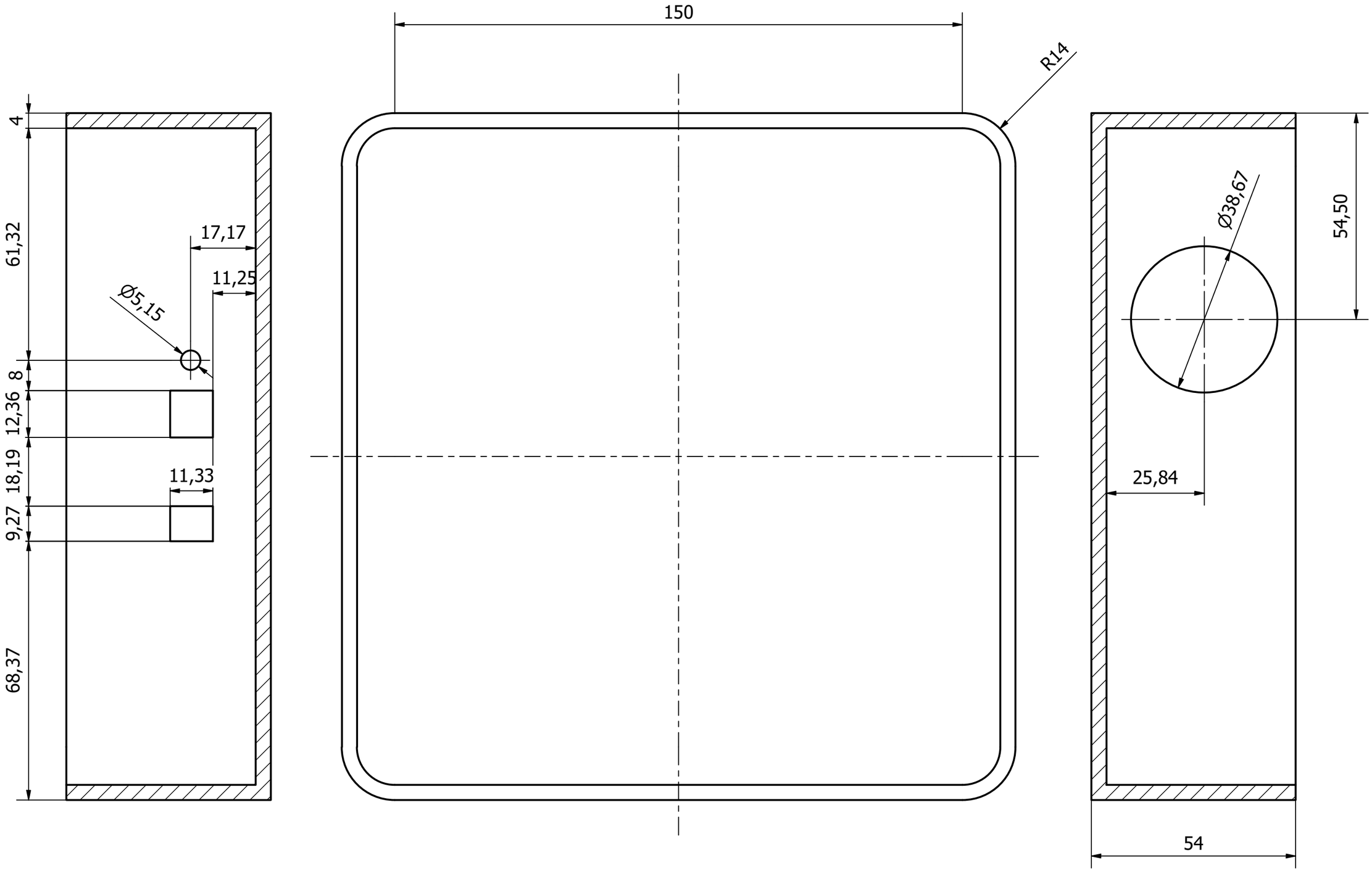
Tabla 6. Presupuesto final

Finalmente, se tiene que, el Trabajo tiene un coste de 7.439,18 €.

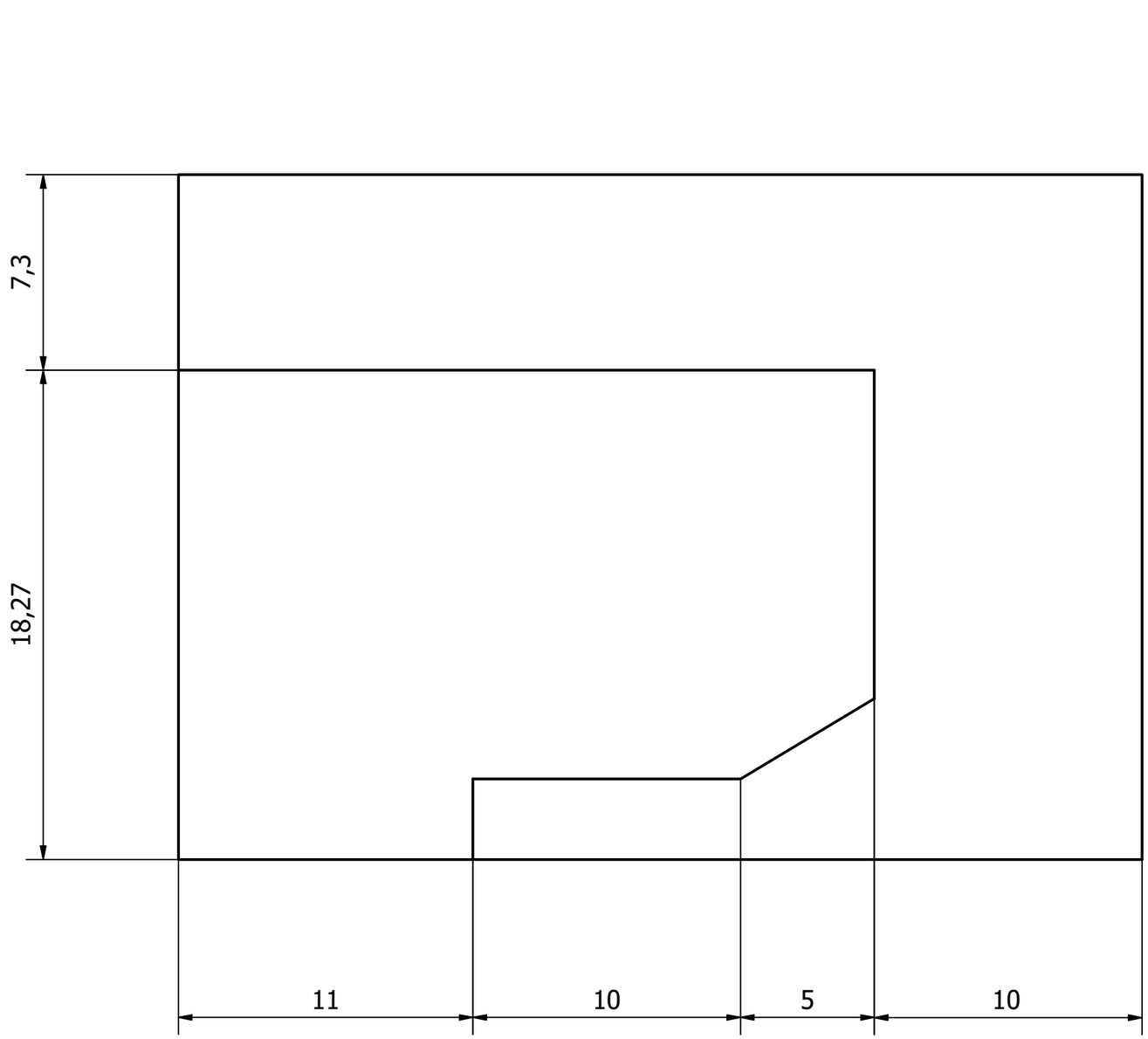
(SIETE MIL CUATROCIENTOS TREINTA Y NUEVE EUROS CON DIECIOCHO CÉNTIMOS).

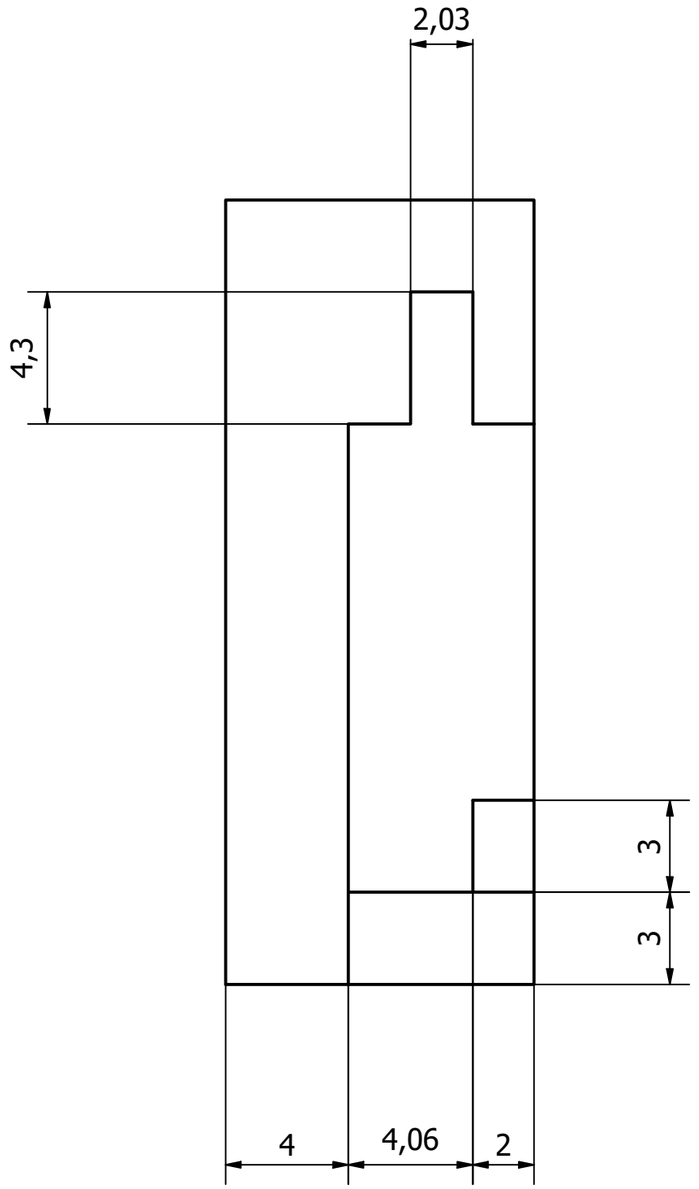
DOCUMENTO III:
PLANOS

ÍNDICE DE LOS PLANOS



TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES UNIVERSITAT POLITÈCNICA DE VALÈNCIA — ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA	Proyecto: DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE VELOCIDAD PARA MOTORES DE CONTINUA BASADO EN MICROCONTROLADOR	Plano: Base Autor: Wen yi Sun	Fecha: Julio 2019 Escala: 1:1	Nº Plano: 1





TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Proyecto: DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE VELOCIDAD PARA MOTORES DE CONTINUA BASADO EN MICROCONTROLADOR

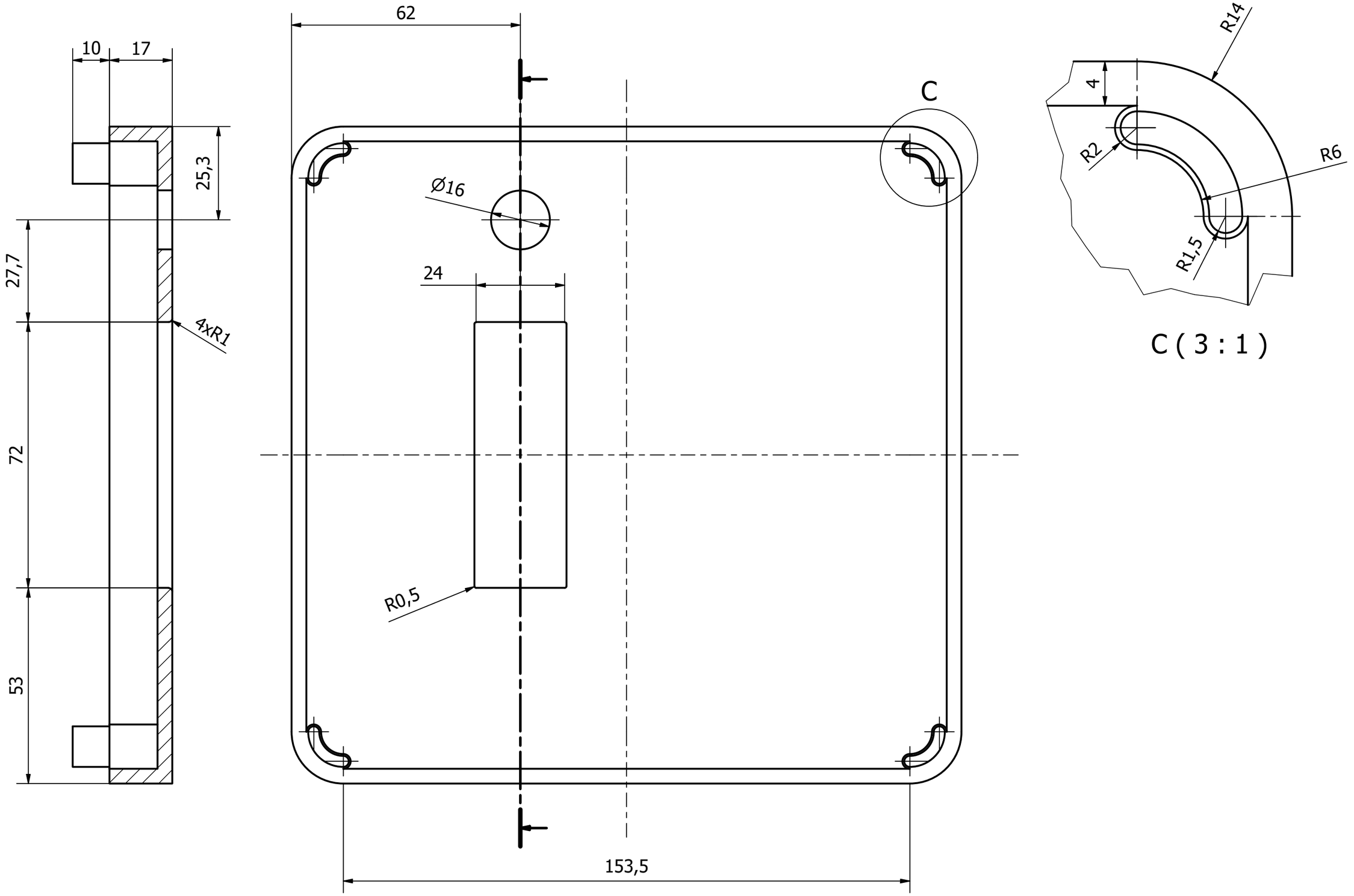
Plano: Soporte encoder

Autor: Wen yi Sun

Fecha: Julio 2019

Escala: 5:1

Nº Plano: 2



TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Proyecto:
DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE VELOCIDAD PARA MOTORES DE CONTINUA BASADO EN MICROCONTROLADOR

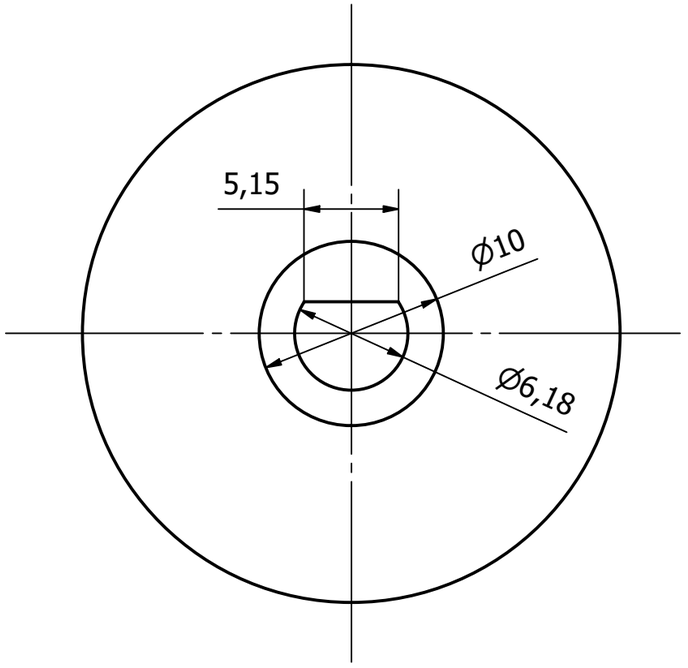
Plano: Tapa

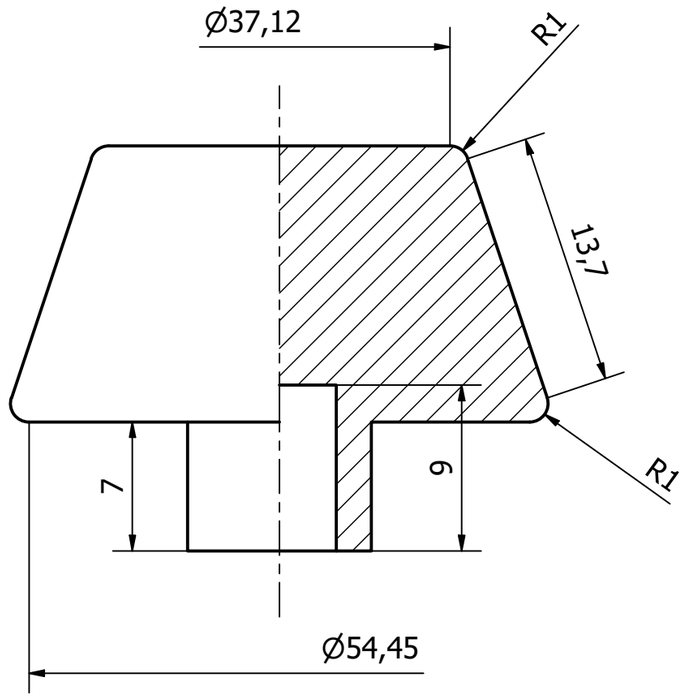
Autor: Wen yi Sun

Fecha: Julio 2019

Escala: 1:1

Nº Plano: 3





TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Proyecto:
DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE VELOCIDAD PARA MOTORES DE CONTINUA BASADO EN MICROCONTROLADOR

Plano: Rueda

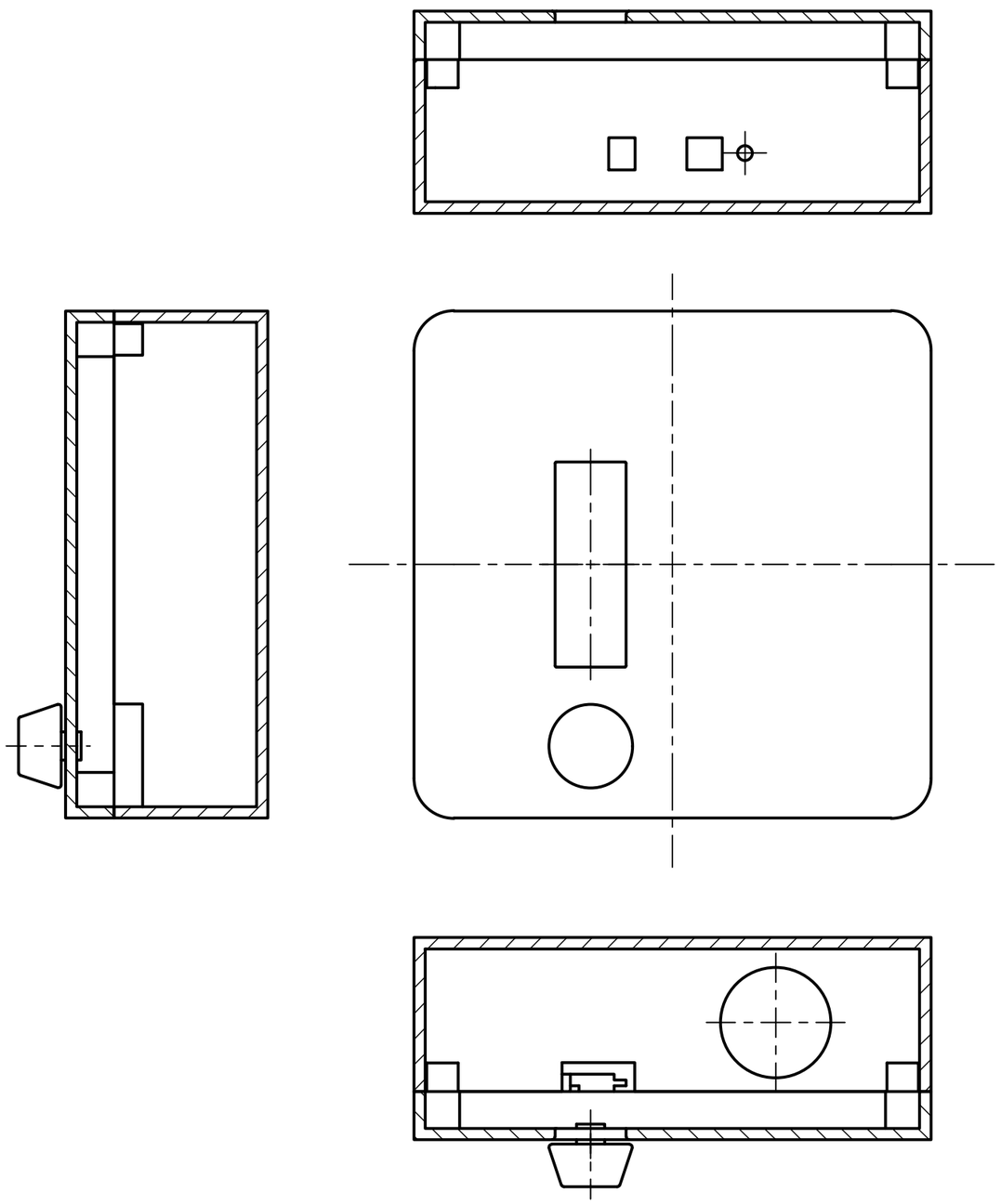
Autor: Wen yi Sun

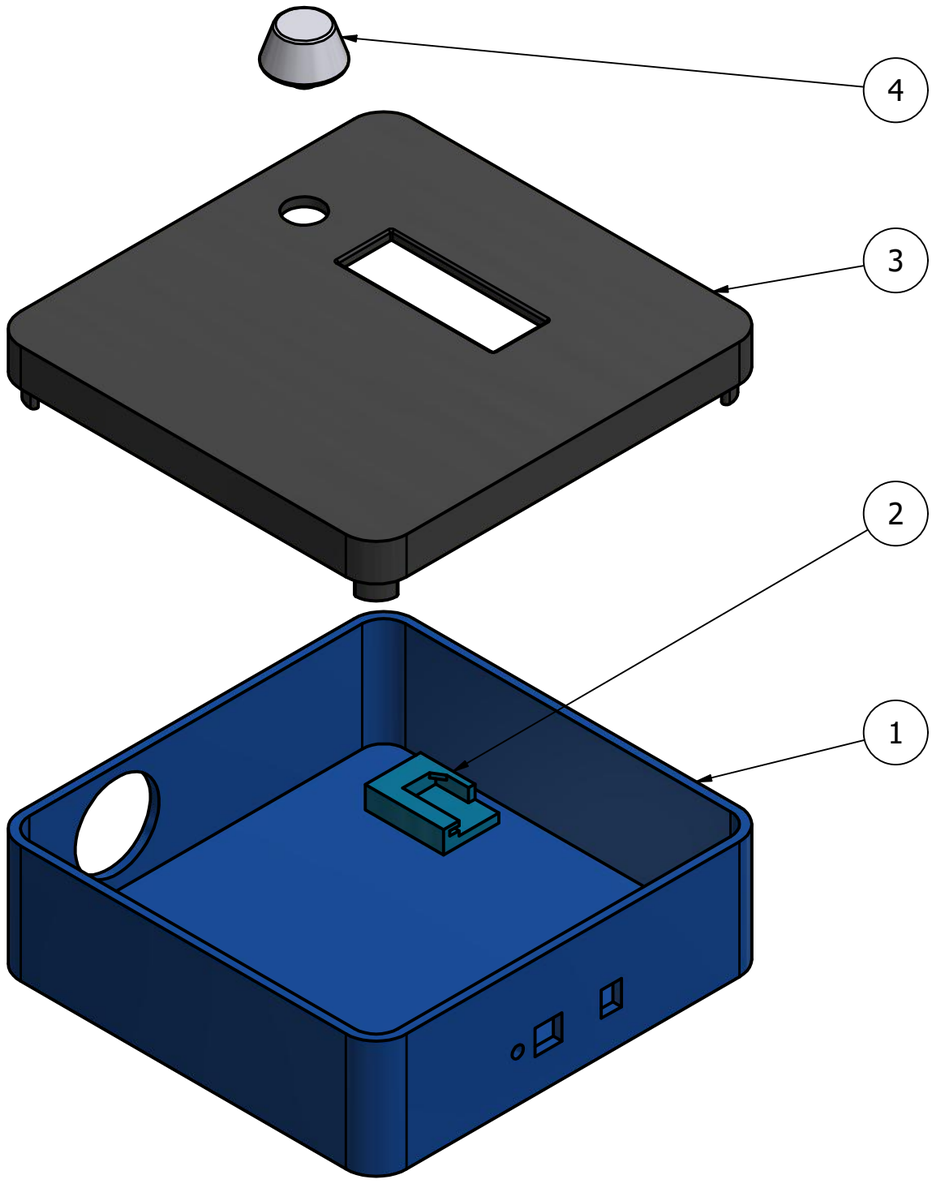
Fecha: Julio 2019

Escala: 3:1

Nº Plano: 4

LISTA DE PIEZAS		
ELEMENTO	CTDAD	Nº DE PIEZA
1	1	Base
2	1	Soporte encoder
3	1	Tapa
4	1	Rueda





TRABAJO FINAL DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Proyecto:
DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE VELOCIDAD PARA MOTORES DE CONTINUA BASADO EN MICROCONTROLADOR

Plano: Conjunto

Autor: Wen yi Sun

Fecha: Julio 2019

Escala: 1:2

Nº Plano: 5

ANEXO I: CÓDIGO PROGRAMACIÓN

```
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#include <EEPROM.h>

LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE); //Configuracion del LCD I2C

#define COUNT(x) sizeof(x)/sizeof(*x)              // Macro para contar el numero de elementos de un array
const byte SW   = 26;                              // Pin encoder SW
const byte DT   = 24;                              // Pin encoder DT
const byte CLK  = 22;                              // Pin encoder CLK
const byte filas    = 2;                           // Filas del LCD
const byte columnas = 16;                          // Columnas del LCD
byte idflechas     = 0;                            // ID icono flecha
byte flechas[]   = {                               // Bits icono flecha
  B00000,
  B00100,
  B00110,
  B11111,
  B00110,
  B00100,
  B00000,
  B00000
};

int IN3 = 9;     // Input3 conectada al pin 9
int IN4 = 8;     // Input4 conectada al pin 8
int ENB = 7;     // ENB conectada al pin 7

int valor_cuenta = 0;

enum Boton { Unknown, OK, Left, Right } botonPres;    // Enumerador con los diferentes botones disponibles
enum Screen { Flag, Number };                         // Enumerador con los distintos tipos de submenus disponibles

const char *MENU[] = {                                // Los textos del menú
  "Sentido horario",
  "Velocidad motor",
  "Configurar Kp  ",
  "Configurar Ti  ",
  "Configurar t   ",
  "Parar motor    ",
  "Guardar y salir",
  "Salir          "
};
const byte iMENU = COUNT(MENU);                       // Numero de opciones del menu principal

struct MYDATA {                                       // Estructura STRUCT con las variables que almacenaran los datos que se guardaran en la memoria EEPROM
  int initialized;
  int sentidomotor;
  int rpmmotor;
  int milikp;
  int militi;
  int t;
  int pararmotor;
};
union MEMORY {                                        // Estructura UNION para facilitar la lectura y escritura en la EEPROM de la estructura STRUCT
  MYDATA d;
  byte b[sizeof(MYDATA)];
}
memory;

void cuenta() {
  if (digitalRead(3) == HIGH)
    valor_cuenta++;
  else
    valor_cuenta--;
```

```
67  }
68
69  void setup() {
70
71    Serial.begin(57600);
72
73    pinMode(SW,  INPUT_PULLUP);
74    pinMode(DT,  INPUT_PULLUP);
75    pinMode(CLK, INPUT_PULLUP);
76
77    pinMode (ENB, OUTPUT);
78    pinMode (IN3, OUTPUT);
79    pinMode (IN4, OUTPUT);
80
81    pinMode (2, INPUT);
82    pinMode (3, INPUT);
83
84    attachInterrupt(digitalPinToInterrupt(2), cuenta, RISING);
85
86    // Carga la configuracion de la EEPROM
87    readConfiguration();
88
89    // Inicia el LCD:
90    lcd.begin(columnas, filas);
91    lcd.createChar(idflechas, flechas);
92
93    lcd.setCursor(0, 0);
94    for ( int i = 0 ; i < columnas ; i++ )
95    {
96      lcd.print(".");
97      delay(100);
98    }
99    lcd.clear();
100 }
101
102 void loop() {
103
104   unsigned long t0, t1, dt;
105
106   int stopmotor = 0;
107   int rpm_ref = 0;
108   int rpm_medida = 0;
109   long rpm_salida = 0;
110   byte sentido;
111   int e = 0;
112   int milikp = 0, militi = 0;
113   int T = 0;
114   int control = 0;
115   float kp = 0;
116   float ti = 0;
117   float up = 0;
118   float ui = 0;
119   static float ui0 = 0;
120   static int slow = 5;
121
122   t0 = millis();
123
124   float Sensibilidad = 0.185; //sensibilidad en Voltios/Amperio para sensor de 5A
125   float voltajeSensor = analogRead(A3) * (5.0 / 1023.0); //lectura del sensor
126   float I = (voltajeSensor - 2.5) / Sensibilidad; //Ecuación  para obtener la
      corriente
127
128   char buf3[16];
129   char valorI[6];
130   dtostrf(fabs(I), 1, 3, valorI);
131   char* intensidad = "I = %s A     ";
132   sprintf(buf3, intensidad, valorI);
133
134   sentido = memory.d.sentidomotor;
135   rpm_ref = (sentido == 0 ? -1 : 1 ) * memory.d.rpmmotor;
136   milikp = memory.d.milikp;
137   militi = memory.d.militi;
138   T = memory.d.t;
```

```
    stopmotor = memory.d.pararmotor;

    botonPres = leerBotones();

    if (botonPres == Boton::OK) {
      abrirMenu();
    }

    rpm_medida = (valor_cuenta) * 182.0 / T;
    valor_cuenta = 0;
    e = rpm_ref - rpm_medida;
    up = (milikp / 1000.0) * e;
    ui = ui0 + (((float)milikp) / (militi)) * (T / 1000.0) * e;

    if (militi == 0) {
      ui = 0;
    }
    control = up + ui;
    if (control > 0) {
      digitalWrite (IN3, HIGH);
      digitalWrite (IN4, LOW);
    }
    else {
      digitalWrite (IN3, LOW);
      digitalWrite (IN4, HIGH);
    }
    rpm_salida = fabs(control) > 255 ? 255 : fabs(control);

    if ( stopmotor == 1) {
      rpm_salida = 0;
      memory.d.rpmmotor = 0;
      writeConfiguration();
    }
    analogWrite(ENB, rpm_salida);

    Serial.println(rpm_medida);

    ui0 = ui;

    static char buf[10];
    char* referencia = "REF %3d ";
    sprintf(buf, referencia, rpm_ref);

    char buf2[20];
    char* medida = "MED %3d ";
    sprintf(buf2, medida, rpm_medida);

    if (slow == 0) {
      lcd.setCursor(8, 0);
      lcd.print(buf2);

      lcd.setCursor(0, 0);
      lcd.print(buf);

      lcd.setCursor(0, 1);
      lcd.print(buf3);
      slow = 5;
    }
    slow--;
    t1 = millis();
    dt = t1 - t0;

    if (dt < T) {
      delay(T - dt);
    }
    else if ( dt >= T) {
    }

    //Serial.println(dt);
  }

  /*Lee el movimiento del encoder*/
```

```
212  Boton leerBotones()
213  {
214    static boolean oldDT = HIGH;
215    static boolean pinDT = LOW;
216    static boolean pinCLK = LOW;
217
218    botonPres = Boton::Unknown;
219    pinDT = digitalRead(DT);
220    pinCLK = digitalRead(CLK);
221
222    if ( !oldDT && pinDT )
223    {
224      botonPres = !pinCLK ? Boton::Right : Boton::Left;
225      delay(50);
226    }
227    else if ( !digitalRead(SW) )
228    {
229      while (!digitalRead(SW));
230      botonPres = Boton::OK;
231      delay(50);
232    }
233
234    oldDT = pinDT;
235    return botonPres;
236  }
237
238  /*Lee y configura la memoria EEPROM*/
239  void readConfiguration()
240  {
241    for ( int i = 0 ; i < sizeof(memory.d) ; i++  )
242      memory.b[i] = EEPROM.read(i);
243
244    if ( memory.d.initialized != 'Y' )
245    {
246      memory.d.initialized  = 'Y';
247      memory.d.sentidomotor = 1;
248      memory.d.rpmmotor     = 0;
249      memory.d.milikp       = 0;
250      memory.d.militi       = 0;
251      memory.d.t            = 0;
252      memory.d.pararmotor   = 0;
253      writeConfiguration();
254    }
255  }
256
257  /*Escribe la memoria EEPROM*/
258
259  void writeConfiguration()
260  {
261    for ( int i = 0 ; i < sizeof(memory.d) ; i++  )
262      EEPROM.write( i, memory.b[i] );
263  }
264
265  /* Función de abrir menú */
266
267  void abrirMenu()
268  {
269    byte idxMenu       = 0;
270    boolean salirMenu   = false;
271    boolean forcePrint = true;
272
273    digitalWrite (IN3, LOW);
274    digitalWrite (IN4, LOW);
275
276    lcd.clear();
277    memory.d.pararmotor = 0;
278    writeConfiguration();
279
280    while ( !salirMenu )
281    {
282      botonPres = leerBotones();
283
284      if ( botonPres == Boton::Left && idxMenu - 1 >= 0 )
```

```
285       {
286         idxMenu--;
287       }
288       else if ( botonPres == Boton::Right && idxMenu + 1 < iMENU )
289       {
290         idxMenu++;
291       }
292       else if ( botonPres == Boton::OK )
293       {
294         switch ( idxMenu )
295         {
296           case 0: openSubMenu( idxMenu, Screen::Flag, &memory.d.sentidomotor, 0, 1);
                  break; //Configurar sentido motor
297           case 1: openSubMenu( idxMenu, Screen::Number, &memory.d.rpmmotor, 0, 320);
                  break; //Configurar RPM Referencia
298           case 2: openSubMenu( idxMenu, Screen::Number, &memory.d.milikp, 0, 5000);
                  break; //Configurar Kp
299           case 3: openSubMenu( idxMenu, Screen::Number, &memory.d.militi, 0, 5000);
                  break; //Configurar Ti
300           case 4: openSubMenu( idxMenu, Screen::Number, &memory.d.t, 0, 200); break;
                  //Configurar t
301           case 5: openSubMenu( idxMenu, Screen::Flag, &memory.d.pararmotor, 0, 1);
                  break; //Parar motor
302           case 6: writeConfiguration(); salirMenu = true; break; //Guardar y salir
303           case 7: readConfiguration();  salirMenu = true; break; //Salir (sin cambios)
304         }
305         forcePrint = true;
306       }
307
308       if ( !salirMenu && (forcePrint || botonPres != Boton::Unknown) )
309       {
310         forcePrint = false;
311
312         static const byte endFor1 = (iMENU + filas - 1) / filas;
313         int graphMenu     = 0;
314
315         for ( int i = 1 ; i <= endFor1 ; i++ )
316         {
317           if ( idxMenu < i * filas )
318           {
319             graphMenu = (i - 1) * filas;
320             break;
321           }
322         }
323
324         byte endFor2 = graphMenu + filas;
325
326         for ( int i = graphMenu, j = 0; i < endFor2 ; i++, j++ )
327         {
328           lcd.setCursor(1, j);
329           lcd.print( (i < iMENU) ? MENU[i] : "                    " );
330         }
331
332         for ( int i = 0 ; i < filas ; i++ )
333         {
334           lcd.setCursor(0, i);
335           lcd.print(" ");
336         }
337         lcd.setCursor(0, idxMenu % filas );
338         lcd.write(idflechas);
339       }
340     }
341
342     lcd.clear();
343   }
344
345   /*Función abrir submenú*/
346
347   void openSubMenu( byte menuID, Screen screen, int *value, int minValue, int maxValue )
348   {
349     boolean exitSubMenu = false;
350     boolean forcePrint  = true;
351
```

```
352     lcd.clear();
353
354     while ( !exitSubMenu )
355     {
356       botonPres = leerBotones();
357
358       if ( botonPres == Boton::OK ) {
359         exitSubMenu = true;
360       }
361
362       else if ( botonPres == Boton::Left && (menuID == 0 || menuID == 5) && (*value) -
        1 >= minValue ) {
363         (*value)--;
364       }
365
366       else if (botonPres == Boton::Left && (menuID == 2) && (*value) - 50 >= minValue
        ) {
367         (*value) -= 50;
368       }
369
370       else if ( botonPres == Boton::Left && (menuID == 3) && (*value) - 10 >= minValue
        ) {
371         (*value) -= 10;
372       }
373
374       else if ( botonPres == Boton::Left && (menuID == 1 || menuID == 4) && (*value) -
        5 >= minValue ) {
375         (*value) -= 5;
376       }
377
378       else if ( botonPres == Boton::Right && (menuID == 0 || menuID == 5) && (*value)
        + 1 <= maxValue ) {
379         (*value)++;
380       }
381
382       else if ( botonPres == Boton::Right && (menuID == 2) && (*value) + 50 <=
        maxValue ) {
383         (*value) += 50;
384       }
385
386       else if ( botonPres == Boton::Right && (menuID == 3) && (*value) + 10 <=
        maxValue ) {
387         (*value) += 10;
388       }
389
390       else if ( botonPres == Boton::Right && (menuID == 1 || menuID == 4) && (*value)
        + 5 <= maxValue ) {
391         (*value) += 5;
392       }
393
394       if ( !exitSubMenu && (forcePrint || botonPres != Boton::Unknown) )
395       {
396         forcePrint = false;
397
398         lcd.setCursor(0, 0);
399         lcd.print(MENU[menuID]);
400
401         lcd.setCursor(0, 1);
402         lcd.print("<");
403         lcd.setCursor(columnas - 1, 1);
404         lcd.print(">");
405
406         if ( screen == Screen::Flag )
407         {
408           lcd.setCursor(columnas / 2 - 1, 1);
409           lcd.print(*value == 0 ? "NO" : "SI");
410         }
411         else if ( screen == Screen::Number )
412         {
413           lcd.setCursor(columnas / 2 - 1, 1);
414           lcd.print(*value);
415           lcd.print(" ");
416         }
```

```
417
418         }
419
420       }
421
422       lcd.clear();
423     }
424
```