Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Eléctrica

Diseño e implementación del control de motor paso mediante dispositivos embebidos RIO

Autor:

Gómez Pindado, Javier

Tutor:

San Martín Ojeda, Moisés Luis

Departamento de Ingeniería Eléctrica

Valladolid, septiembre 2018.

Resumen y palabras clave

Este trabajo de fin de grado consiste en el diseño de un control de motor paso a paso, con un objetivo didáctico que permita conocer y entender mejor el funcionamiento de estos motores. Se realiza mediante dos botoneras, una física y otra digital. Para ello se utiliza un dispositivo Arduino, integrado en un circuito impreso, y la programación que va asociada a este, utilizando un código basado en interrupciones del microprocesador y en el cual se establece de manera manual y sin el uso de bibliotecas las secuencias de giro del motor. Para el control digital vía PC se utilizará el programa LabVIEW y la comunicación entre máquina y dispositivo se realizará mediante el puerto serie. Para ello se creará un protocolo de comunicaciones que evite la pérdida de información y que permita trabajar de manera simultánea al control físico y a la digital.

Palabras clave: motor, paso, Arduino, LabVIEW y serial.

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Diseño e implementación del control de motor paso a paso mediante dispositivos embebidos RIO

Javier Gómez Pindado

Grado en Ingeniería Eléctrica

Tutor: Moisés San Martín Ojeda

Código: 01003

Septiembre 2018

Índice Memoria

Anexo 1. Manual de uso

Anexo 2. Planos

Anexo 3 Hojas de datos

Anexo 4. Código

Índice de figuras

1. Introducción y objeto del TFG:

Con este trabajo se pretende realizar el control de un motor paso a paso mediante dispositivos embebidos de tiempo real a través de un enfoque didáctico que facilite la comprensión del funcionamiento de los motores paso a paso y de su control.

Por lo tanto, con el objeto de una visualización y manejos que resulten didácticos, se buscan los siguientes requisitos para el control, diferenciando el control físico o analógico y el control mediante una interfaz de ordenador:

- Control físico: se pide que el motor sea capaz de girar en ambos sentidos de manera autónoma y a velocidad fija, mediante una activación manual, y también que el motor pueda girar en ambos sentidos realizando la secuencia de alimentación de fases de manera manual.
- Control vía PC: se pide que el motor sea capaz de girar en ambos sentidos a velocidad variable. Además, se requiere que el dispositivo devuelva el estado del motor (sentido de giro y número de pasos realizados).

2. Descripción general

2.1 Motor paso a paso:

El motor paso a paso es un motor cuya característica principal es la de convertir la excitación de las distintas bobinas que lo componen en cambios en la posición del rotor mediante precisos incrementos. Este tipo de motores se caracteriza por tener dientes de material magnético tanto en el rotor como en el estator. El flujo magnético generado se cerrará por el entrehierro existente entre los dientes del rotor y los del estator. De esta manera los dientes de ambas partes experimentarán fuerzas cuyo objetivo sea minimizar el espacio libre entre ellos, apareciendo una fuerza normal y otra tangencial, figura 1, esta última será la que hará girar el motor. Así, alimentando mediante la secuencia adecuada las distintas bobinas el motor girará en un sentido o en otro.

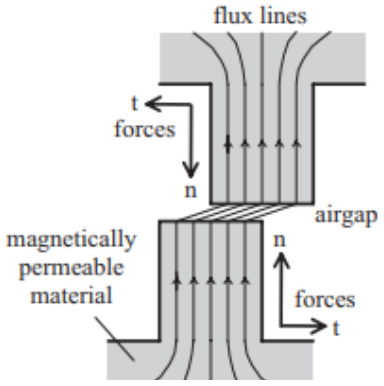

Figura 1. Acarnley(2002). Stapping Motors a Guide of Theory and Practice

Existen dos tipos de motor paso a paso los de reluctancia variable y los motores paso a paso híbridos. En los primeros el campo magnético es producido solamente por la energización de bobinas, mientras que en el segundo tipo la principal fuente de campo magnético es un imán permanente y se alimentan bobinas con corriente continua de manera que se conduzca el flujo por distintos caminos.

2.1.1 Motor paso a paso de reluctancia variable:

Motor paso a paso de reluctancia variable *multi-stack*:

Este motor está dividido axialmente en secciones magnéticas aisladas (*stacks*), cada una de estas secciones se excita de manera independiente (fase).

Cada sección incluye un estátor que contiene las bobinas a alimentar y el rotor que es una pieza común para todas las secciones y está fabricado de acero eléctrico (magnético). El estátor está dividido en polos alrededor de los cuales se enrollan distintas partes del cableado de la fase de manera que se produzca un campo magnético radial. El recorrido del flujo partirá de uno de los polos y a través del entrehierro cruzará el acero del rotor para llegar al polo opuesto y retornar por la envolvente del estátor. En este campo radial las fuerzas normales que se generan al intentar reducir el campo el entrehierro quedan anuladas quedando únicamente las tangenciales que originan el giro.

La posición en cada sección viene determinada por la necesidad de alineación entre los dientes del rotor y el estátor de manera que se vea reducida la reluctancia magnética.

La capacidad de giro de este tipo de motores yace en que mientras los dientes del rotor están alineados a lo largo de todo el eje (en cada una de las secciones), los polos de cada una de las secciones están ligeramente desalineados de forma que al alimentar de manera sucesiva cada una de las fases, que corresponde con cada una de las secciones), el rotor girará para alinear en cada caso sus dientes con los polos energizados, figura 2. Hay que tener en cuenta que el número de dientes del rotor corresponde con el número de polos del estátor. De manera que para hacer girar el motor en un sentido o en otro habrá que alimentar cada una de las fases (secciones) en secuencia positiva 1-2-3-1-2-3… o negativa 1-3-2-1-3-2…

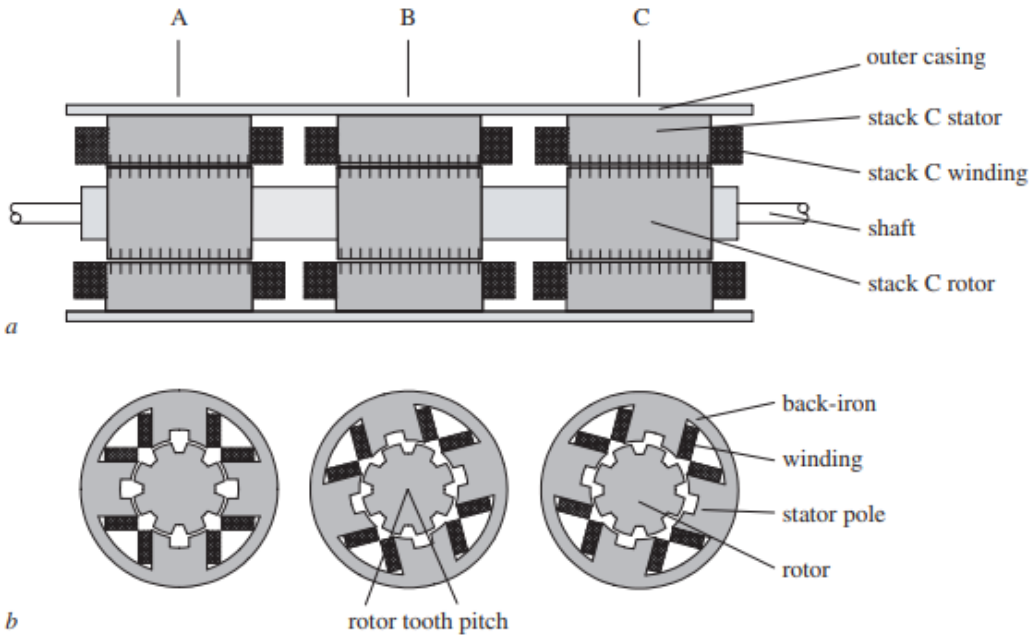

Figura 2. Acarnley(2002). Stapping Motors a Guide of Theory and Practice

La relación entre el número de dientes del rotor, las secciones y la longitud del paso será:

$$Longitud\ de\ paso(°) = \frac{360}{N \cdot p}$$

Donde N es el número de secciones y p el número de dientes del rotor.

Motor de reluctancia variable single-*stack*:

En este caso el motor consta de una sola sección y su rotor está construido de la misma manera que los *multi-stack* y el estátor está dividido en polos de la misma manera, pero con la diferencia de que los distintos pares de polos opuestos tienen enrollados cable de distintas fases. De tal forma que el funcionamiento de estos motores consistirá en alimentar las distintas fases para conseguir crear un campo que cree una fuerza que intente reducir el entrehierro y alinee los dientes del rotor con el par de polos. A diferencia de los *multi-stack* estos motores tienen un número de dientes en el rotor distinto al número de polos del estátor, figura 3.

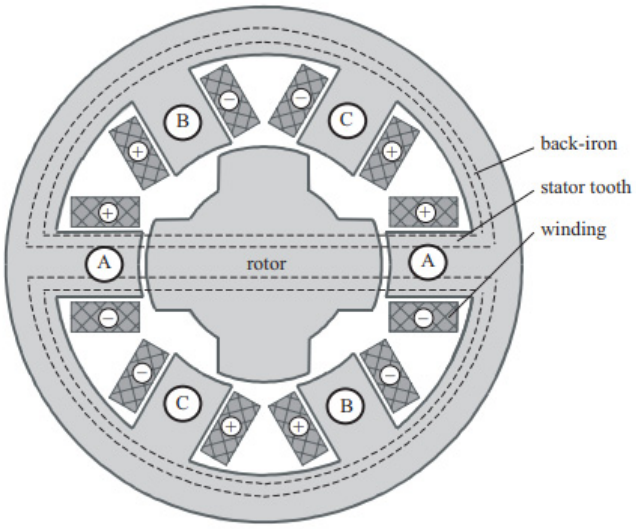

Figura 3. Acarnley(2002). Stapping Motors a Guide of Theory and Practice

La relación entre el número de dientes del rotor, las secciones y la longitud del paso será:

$$Longitud\ de\ paso(°) = \frac{360}{N \cdot p}$$

Donde N es el número de fases y p el número de dientes del rotor.

2.1.2 Motor paso a paso híbrido:

Este tipo de motores se apoya en la combinación de bobinas y un imán permanente para crear el circuito magnético. Las bobinas se encuentran situadas en polos del estator y el imán permanente se sitúa en el rotor. De manera que el recorrido del flujo magnético sea: polo norte del imán, cabezal de hierro del rotor, a través del entrehierro pasa a los polos del estátor de la sección X, pasa por la estructura de hierro del estátor hasta los polos del estátor de la sección Y, a través del entrehierro pasan al cabezal metálico y finalmente acaban en el polo sur del imán, figura 4.

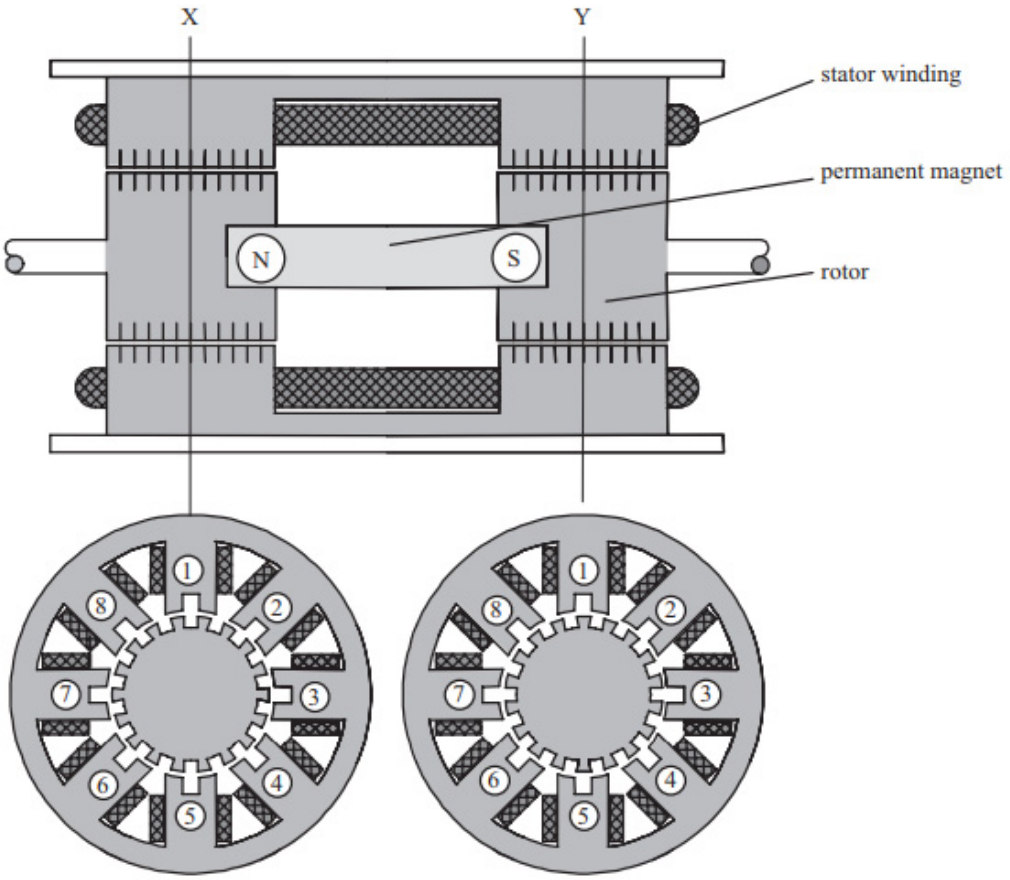

Figura 4. Acarnley(2002). Stapping Motors a Guide of Theory and Practice

Por norma general el estátor es de ocho polos. Estos polos están rodeados por bobinas que sirven para habilitar o deshabilitar el paso del flujo por los distintos polos en función de la posición del rotor. Estas bobinas se alimentan mediante dos fases de manera que cada fase alimente cuatro de estas bobinas de manera alterna, una fase, llamémosla A, alimentará a los polos impares (1,3,5,7) y la otra bobina, B, alimentará a las bobinas de los polos pares (2,4,6,8). De esta manera los polos sucesivos serán alimentados de manera opuesta de manera que dos bobinas dejarán pasar el flujo en un sentido y las otras dos en el contrario, habilitando así que el flujo pase por dos de ellas en la sección X y por las otras dos en la sección Y.

Tabla 1. Acarnley(2002). Stapping Motors a Guide of Theory and Practice

Winding	Current direction	Pole field direction	
		Radially outward	Radially inward
A	positive	3, 7	1, 5
A	negative	1, 5	3, 7
B	positive	4, 8	2, 6
B	negative	2, 6	4, 8

Para conseguir una rotación continua del motor simplemente habrá que realizar una excitación secuencial de las dos fases. Para obtener un giro en sentido horario la secuencia será: A+, B+, A-, B-, A+, B+, … Y para un giro en sentido antihorario la secuencia será: A+, B-, A-, B+, A+, B-, …

La longitud de cada uno de los pasos (pequeños incrementos en la posición del rotor) está relacionada con el número de dientes del rotor, p. Un ciclo completo de excitación en el motor híbrido consiste en 4 estados que originarán sendos pasos, de esta manera cada cuatro pasos el estado de excitación será el mismo, así que la alineación de los dientes del estátor con el rotor se hará en el mismo polo del estátor. Por lo tanto 4 pasos corresponden a un movimiento del rotor de (360/p)º y en el caso del motor paso a paso híbrido:

$$Longitud\ del\ paso = \frac{90}{p}\ (°)$$

En el caso de nuestro motor de 200 pasos por vuelta, que corresponden a una longitud de paso de 1,8°, el rotor tiene 50 dientes, figura 5.

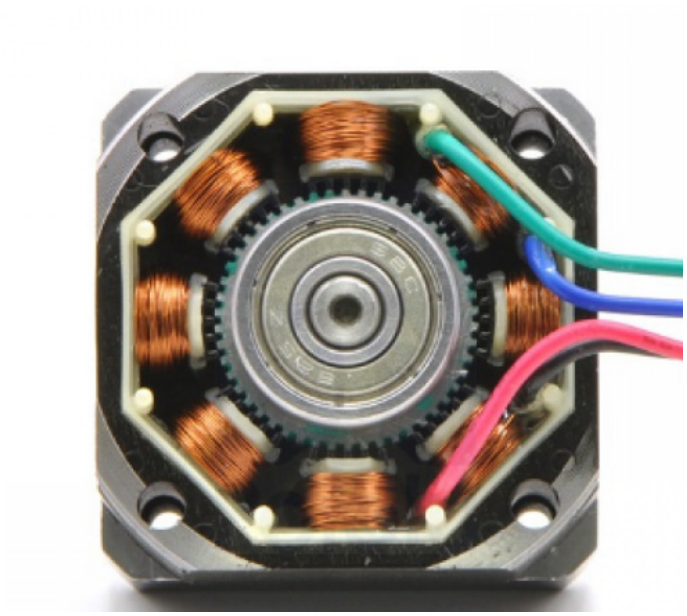

Figura 5.Polulu-Stepper motor (2018) Recuperado de https://www.pololu.com/product/1205

Dependiendo la manera en la que se alimenten los 8 polos exciten, podemos distinguir entre motores bipolares, los anteriormente descritos, y motores unipolares (motor de trabajo) que se caracterizan por alimentar las bobinas de los polos de dos en dos en vez de cuatro en cuatro como los bipolares. El sistema de funcionamiento de ambos es prácticamente el mismo diferenciándose únicamente en la manera de realizar la secuencia de alimentación.

Motor bipolar:

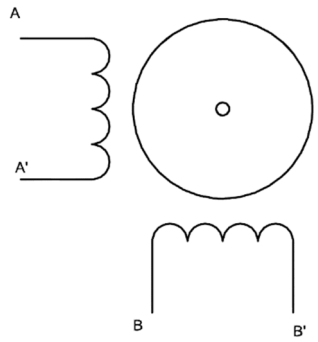

Figura 6 Motor bipolar. Fuente: elaboración propia

Para realizar la secuencia del motor bipolar, figura 6, se realiza la secuencia que se ha explicado anteriormente. Llamando a A+: A con el positivo y A' con el negativo, y al contrario para A-. y de la misma manera para B+ y B- con sus respectivos terminales homónimos.

Motor unipolar:

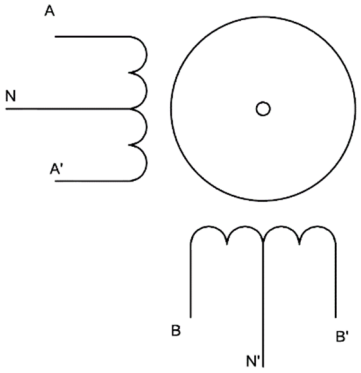

Figura 7 Motor unipolar. Fuente: elaboración propia

Como se puede observar, figura 7, este motor es igual que el bipolar, figura 6, podría funcionar de la misma manera, con la salvedad de que en la mitad del bobinado de cada fase se extrae un punto de conexión (N y N'). Para realizar la secuencia de alimentación del motor en primer lugar se ponen en común ambos puntos de conexión (N y N') y se conectan al positivo de la fuente de alimentación continua de manera que el circuito eléctrico se cerrará mediante cualquiera de los terminales de cada una de las fases.

Para realizar el giro horario la secuencia de terminales que cierran el circuito será: A, B, A', B', A, B, … Y para el sentido antihorario: A, B', A', B, A, B', …

Cabe destacar que estos motores unipolares se pueden encontrar de 5 hilos, 6 hilos (figura 7) u 8 hilos, dependiendo de si N y N' están unidas en el interior(5 hilos), si están separadas (6 hilos) o si tanto N como N' se corresponden con dos hilos separados (8 hilos) teniendo las siguientes conexiones: A-N, A'-N'', B-B' y B'-N'''.

2.1.3 Motor paso a paso de imán permanente:

En estos motores el eje está formado por un imán permanente, figura 8, y no está dentado, de tal forma que queda reducido a los dos polos del imán. El estátor tiene una estructura similar a la de los motores de reluctancia variable de una sección (single stack) de tal manera que el número de pasos por vuelta del motor dependerán del número de polos, dos por fase que tenga el estátor. En el ejemplo que se muestra a continuación el motor sería de 4 pasos de 90° cada uno.

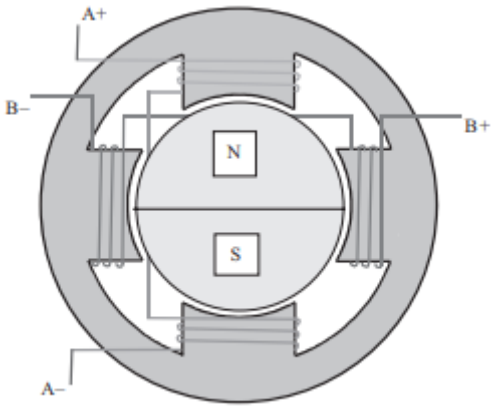

Figura 8. Acarnley(2002). Stapping Motors a Guide of Theory and Practice

2.1.4 Modos de giro en motores paso a paso:

En cualquiera de los tipos de motores anteriores cabe la posibilidad de alimentar más de una fase a la vez, de esta forma se generan fuerzas que dan como resultante posiciones del eje alternativas. Por ejemplo: si alimentamos dos fases consecutivas a la vez el eje se orientará en la dirección intermedia entre estas dos fases. De este modo, si queremos pasar de A a B y alimentamos según la secuencia: A, AB, B, en vez de un paso tendremos dos. Este es el método de medio paso o *half step*.

Yendo más allá, se podría variar la corriente de alimentación de una de las dos fases dando lugar a otra orientación más.

De esta manera, se puede modificar el número de pasos por vuelta del motor y conseguir mayor versatilidad a la hora de colocar el eje en una posición concreta, siempre a costa de unos dispositivos y un control más complejos.

(Acarnley, 2002)

2.2 Opciones de control del motor paso a paso

Para realizar el control de un motor paso a paso mediante el uso de Arduino y LabVIEW aparecen varias opciones, tanto de librerías de Arduino o LabVIEW a

utilizar como de placas de control y la utilización de módulos shield para dichas placas.

Desde un punto de vista de programación en Arduino, nos encontramos con las siguientes opciones:

1. Programación utilizando las funciones básicas de Arduino. Si se utiliza este tipo de programación, habrá que realizar un programa en el que se creen de forma manual las secuencias de alimentación de las bobinas correspondientes, así como programar el tiempo que transcurre entre ellas para así controlar la velocidad.

2. Librerías de Arduino. Arduino cuenta con librerías específicas para el control de motores paso a paso en las cuales solo habría que introducir valores del motor como los pasos por vuelta, velocidad y sentido de giro. Por ejemplo, nos encontramos con la librería Stepper cuyas funciones son:
   i. Stepper(steps, pin1, pin2) o Stepper(steps, pin1, pin2, pin3, pin4): en las que se ajusta el número de pasos por vuelta, y se establece que pines vana a alimentar el motor. Ej. Stepper myStepper(100,4,5).
   ii. setSpeed(rpm): esta función únicamente preestablece la velocidad a la que girará el motor.
   iii. step(steps): gira el motor un determinado número de pasos, se utilizará el valor positivo o negativo de la variable para modificar el sentido de giro.

   En los dos casos anteriores, dado que Arduino cuenta con una tensión y una corriente limitadas, habrá que utilizar un dispositivo que amplifique las señales provenientes del Arduino. Para este cometido existen distintos modelos de matrices Darlington, como son el ULN2003 (16 pines) o el ULN2803A (18 pines).
   (“Stepper”, 2018)
3. Utilización de una CNC shield. Estas shields, figura 10, están diseñadas para el control de cuatro motores de manera simultánea, cuenta con 2 finales de carrera por eje. El objetivo de estas placas es facilitar el control de una máquina CNC (Computador de Control Numérico) y está preparado para funcionar con drivers del tipo A4988 o DRV8825, figura9, que cuentan con la capacidad de realizar microstepping (1/16 o 1/32). La idea principal de este módulo es el funcionamiento con GRBL, que es un firmware de OpenSource creado para Arduino UNO cuya finalidad es convertir código-G, que es el utilizado por las máquinas CNC, en comando de control para motores paso a paso.

("Guía definitiva para dominar GRBL", 2018)

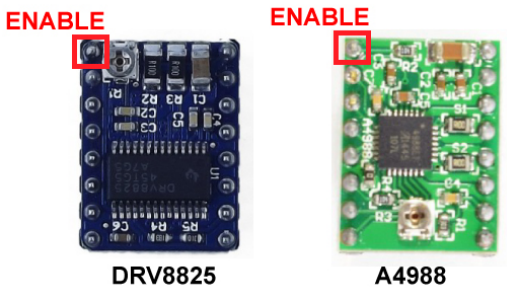

Figura 9 Shield CNC A4988 (2018). Recuperado de https://naylampmechatronics.com/arduino-shields/68-shield-cnc-a4988-grbl-.html

Esta placa requiere de un código concreto en el que los pines están predefinidos.

Figura 10 Shield CNC A4988 (2018). Recuperado de https://naylampmechatronics.com/arduino-shields/68-shield-cnc-a4988-grbl-.html

Desde el punto de vista del control del dispositivo desde LabVIEW, se encuentran las siguientes opciones:

1. Utilizar la librería de comunicación con el puerto serie incorporada en LabVIEW y que el programa de Arduino y el de LabVIEW funcionen de manera independiente, pero manteniendo comunicación. De esta forma, habrá que realizar dos programas, uno para Arduino y el otro para LabVIEW y diseñar un protocolo de comunicación entre ambos.
2. Cargar en la placa Arduino el firmware correspondiente a una de las siguientes dos librerías de LabVIEW de tal forma que Arduino dependa de manera permanente del programa LabVIEW. Realizando toda la programación en este último.
    i. Librería LINX: se carga el firmware en el Arduino a través de una herramienta que proporciona la implementación del programa a LabVIEW, figura 11.

Figura 11 Firmware LINX Fuente: elaboración propia (LabVIEW)

Una vez cargado el firmware, figura 11, se puede empezar a programar en LabVIEW utilizando los bloques de la librería, figura 12. A pesar de ser una programación por bloques, la programación es muy parecida a la que se utilizaría en Arduino, seleccionando que pines se quieren utilizar y de qué manera.

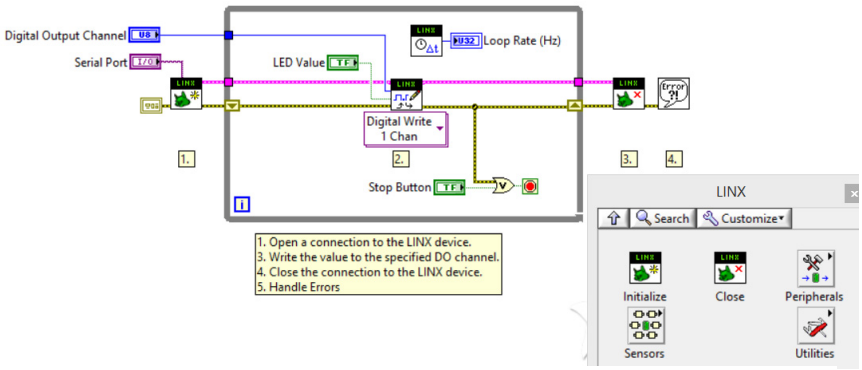

Figura 12 Programa LINX (Diagram block) Fuente: elaboración propia (LabVIEW)

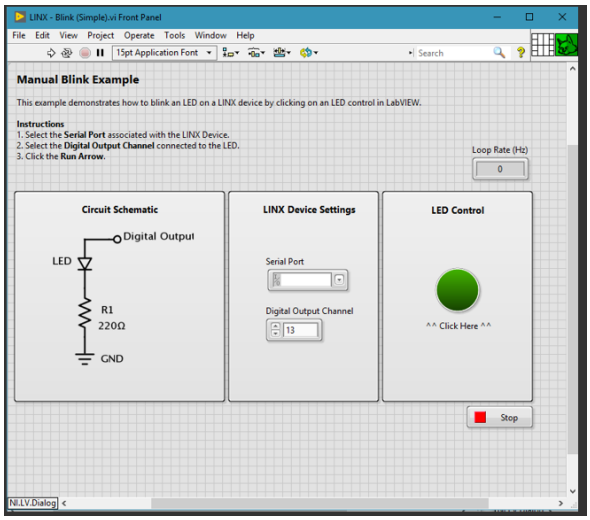

Figura 13 Programa LINX (Front panel) Fuente: elaboración propia (LabVIEW)

ii. Librería Arduino para LabVIEW: al descargar esta librería para LabVIEW, también se descargarán una serie de archivos, dentro de esos archivos aparece un programa .ino (sketch de Arduino) que es el firmware. Tan solo hay que abrir este sketch y subir el código a la placa y empezar a programar en LabVIEW.

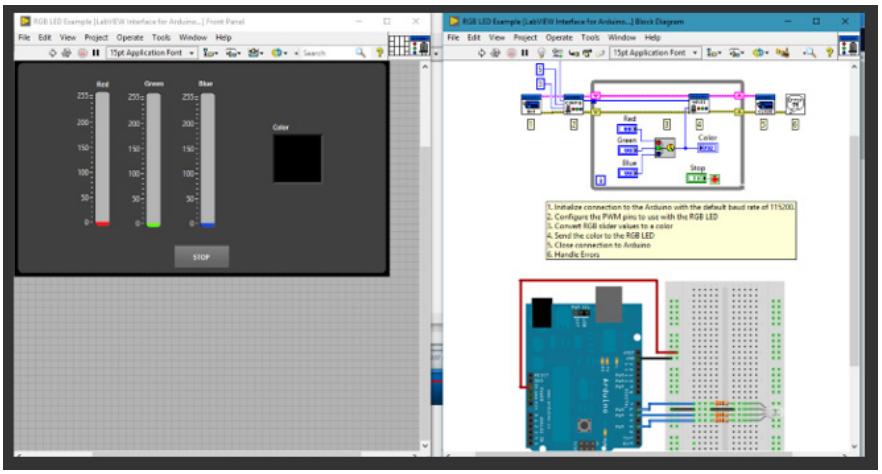

Figura 14 Programa librería Arduino Fuente: elaboración propia (LabVIEW)

Ambas librerías funcionan de manera similar y cuentan con programas ejemplo en los que se ilustra de manera sencilla su funcionamiento. No obstante, estas librerías tienen el inconveniente de necesitar constante comunicación con LabVIEW para que la placa de Arduino funcione.

2.3 Arduino:

Puesto que se va a utilizar un dispositivo Arduino con su correspondiente programación, a continuación, se explica el funcionamiento de Arduino y su software.

Esta programación consiste en programar un microcontrolador. Es decir, consiste en traducir tareas programadas a líneas de código a través de un entorno de programación sencillo como el que proporciona Arduino Software (IDE), que además cuenta con las herramientas para compilar el programa y

grabarlo en la memoria flash del microcontrolador. Este software cuenta con un sistema para gestionar placas y librerías muy práctico.

Figura 15 Logo Arduino (2018) Recuperado de www.arduino.cc

2.3.1 Lenguaje Arduino:

Arduino basa su programación en el lenguaje C++, es decir, es una adaptación que proviene de AVR Libc que proporciona una librería C para el uso con GCC (compilador de C y C++) en microcontroladores Atmel AVR. No obstante, las herramientas necesitadas para la programación del microcontrolador están incluidas en el software de Arduino. Además, el IDE incluye otras librerías que utilizamos sin necesidad de que se declaren. A parte de estas diferencias hay que destacar también la estructura básica del programa que se detalla más adelante. Aun así, el software acepta comandos de C++ estándar (si están incluidos en AVR libc).

2.3.2 Programa Arduino:

Un proyecto de Arduino o sketch, figura 16, consta de la siguiente estructura principal:

- setup() es la función encargada de la configuración. Se ejecuta una sola vez al empezar el programa. Se utiliza para configurar los modos de trabajo de los pines (entrada o salida) o configurar el puerto serie.
- loop() contiene el programa que se va a ejecutar, pudiendo haber llamadas a otras funciones. Este programa se repetirá cíclicamente (loop=bucle), de esta manera podrá estar leyendo o escribiendo continuamente.

Figura 16 Programa en blanco Arduino. Fuente: elaboración propia

Estas dos funciones son necesarias para que funcione el programa, incluso si una de ellas está vacía. Además de esta estructura puede incluir funciones a parte a las que se llamará en el código.

- Funciones: son bloques de código independientes que cuentan con un nombre y con unas tareas que se ejecutan cuando esta función es llamada. Estas funciones se utilizan para reducir el tamaño de los programas cuando son tareas repetitivas.

Dentro de cada una de estas funciones se utilizan los comandos propios de lenguaje de Arduino (C++ adaptado), tales como estructuras de control (if, while, for, …), operadores booleanos (&&, ||, …), …

(Evans, 2007)

2.4 LabVIEW:

El software LabVIEW de National Instruments es un potente programa enfocado a la integración de sistemas E/S (*I/O*) y plataformas de medida, su principal objetivo es acelerar la productividad de los ingenieros y científicos. Para ello cuenta con un potente entorno de programación gráfica que facilita tareas de ingeniería como la adquisición de datos, análisis de medidas, presentación de datos, crear sistemas de control embebido y crear aplicaciones de gran flexibilidad. Además, LabVIEW es capaz de interactuar con otros softwares, como por ejemplo Arduino, para poder añadir versatilidad al programa y tener un mayor rango de aplicación de las herramientas que ofrece y cuenta con más

de 10.000 *drivers* para distinta instrumentación para garantizar la compatibilidad de casi cualquier dispositivo con el sistema LabVIEW.

(Consejo Superior de Investigaciones Científicas [CSIC],2018; National Intruments,2018).

Figura 17 Logo LabVIEW (2018) National Instruments. Recuperado de www.ni.com

Entre sus principales características podemos encontrar:

- Lenguaje intuitivo.
- Herramientas y librerías de alto nivel para aplicaciones específicas.
- Interfaz gráfica muy eficiente y versátil.
- Control del código fuente.
- Gran número de funciones para control, análisis, E/S y presentación de datos.
- Gran número de ejemplos con programación abierta para adecuarlos a casos concretos.
- Ayudas integradas y tutoriales.

Áreas de aplicación:

- Análisis automatizado y plataformas de medida.
- Medidas industriales y plataformas de control.
- Diseño embebido y plataformas de prototipaje.

(Universidad de Cantabria [UC], 2018)

El programa ofrece varias posibilidades a la hora de comenzar la programación, desde un proyecto completo hasta un sencillo "VI" que consta de un diagrama de bloques (entorno de programación) y un panel frontal (interfaz de visualización y control), figura 18. Dentro de un proyecto se encuentran los "VI's", archivos de apoyo, aplicaciones y configuraciones de hardware.

Estructura de un "VI", figura 18:

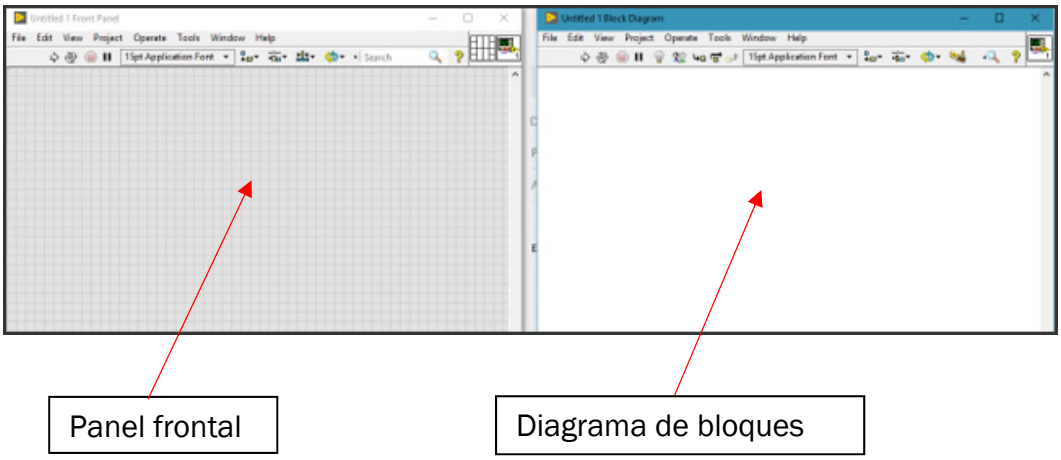

Figura 18 Programa en blanco LabVIEW. Fuente: elaboración propia

En el panel frontal se dispondrán los controles e indicadores (figura 19) tales como controles e indicadores numéricos, interruptores y botones (controladores booleanos), leds (indicador booleano), gráficos, …

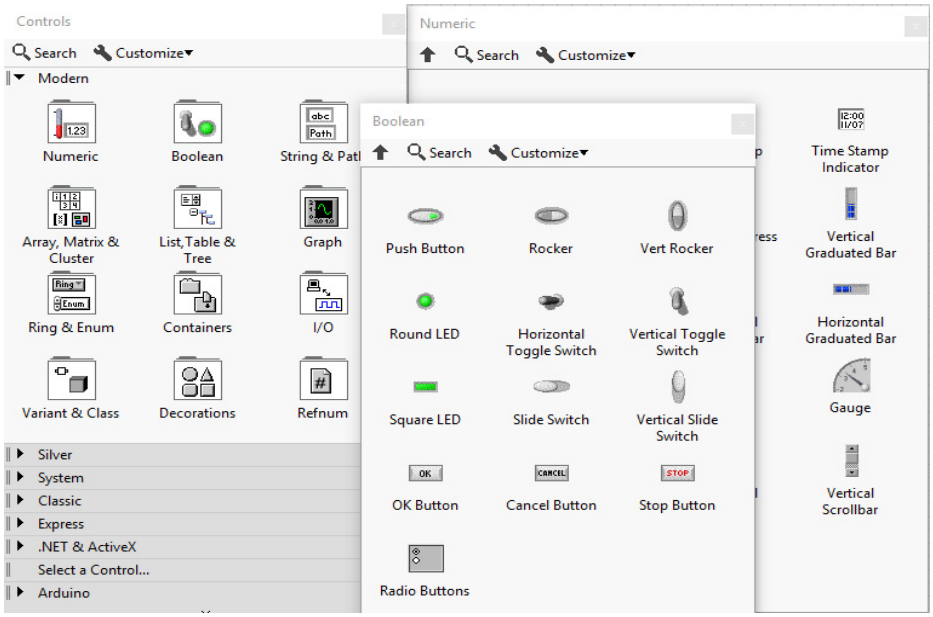

Figura 19 Controles LabVIEW. Fuente: elaboración propia

En el diagrama de bloques se realiza la programación, como su propio nombre indica esta programación se realiza mediante la utilización de bloques como operadores numéricos o booleanos, conversores de tipo de datos, etc. y estructuras semejantes a las estructuras de control del lenguaje C, figura 20. Estos bloques se unirán mediante "cables". Además, en esta ventana aparecerán los bloques correspondientes a los elementos generados en el panel frontal, con conexión de salida los controles y conexión de entrada los indicadores.

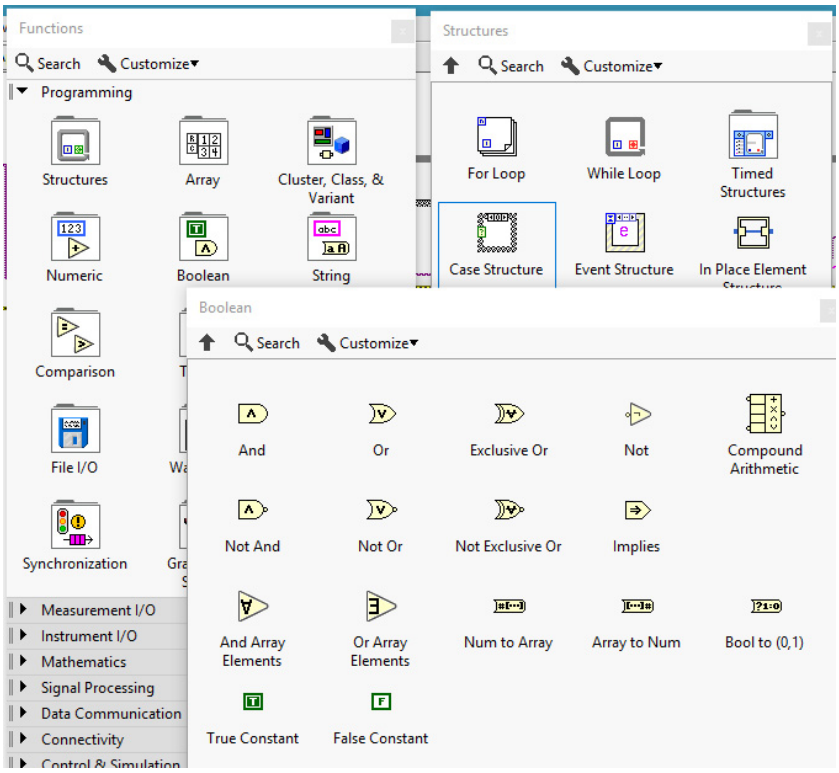

Figura 20 Bloques LabVIEW. Fuente: elaboración propia

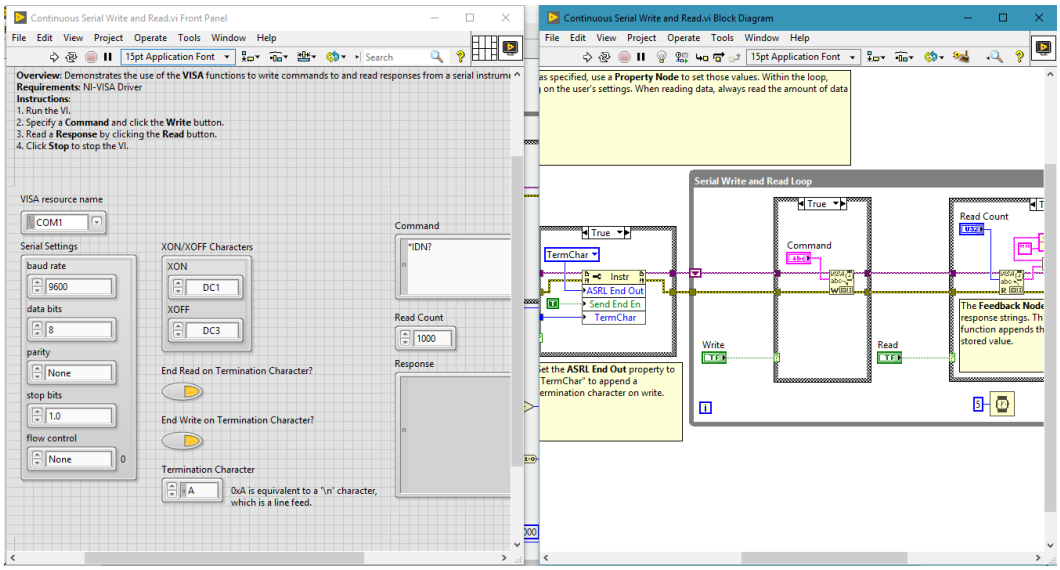

Figura 21 Programa ejemplo LabVIEW. Fuente: elaboración propia

Una vez creado el programa se puede ejecutar mediante los botones *Run*, *Run Continuously* y se detendrá mediante el botón *Abort Execution*. Además, el programa nos otorga la opción de pulsar el botón *Highlight Execution* para ver de una forma más lenta y detallada el funcionamiento de nuestro programa, figura 22.

Figura 22 Control de programa LabVIEW. Fuente: elaboración propia

2.5 Comunicaciones

Una de las principales ventajas de Arduino es que se dispone de casi cualquier tipo de comunicación con tan solo añadir un shield (placas accesorio) a nuestra placa o utilizando alguna placa Arduino más específica. A continuación, se va a exponer las distintas formas de comunicación con las que cuenta Arduino y se va a profundizar en la comunicación del puerto serie.

A. Comunicación Ethernet: se establece la comunicación con el dispositivo utilizando el protocolo ethernet. Añadiendo Arduino Ethernet Shield o utilizando la placa Arduino Leonardo Ethernet.

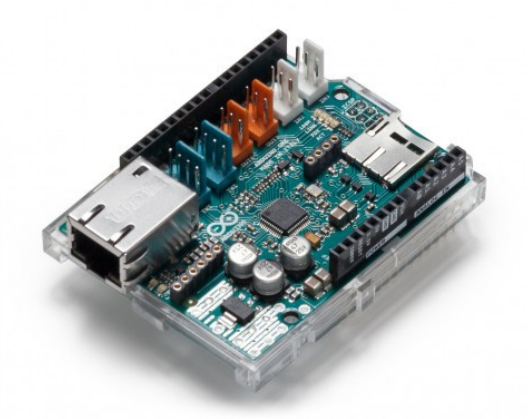

Figura 23 Arduino Ethernet Shield (2018) Recuperado de https://store.arduino.cc/arduino-ethernet-shield-2

Figura 24 Arduino Leonardo Ethernet (2018) Recuperado de https://store.arduino.cc/arduino-leonardo-eth

Además, Arduino cuenta con una librería específica para establecer esta comunicación.

B. Comunicación Wifi: para realizar esta comunicación existen múltiples vías, tales como:

Shield: esta shield (Arduino Wifi 101 Shield) está preparada para comunicar de manera inalámbrica con Arduino.

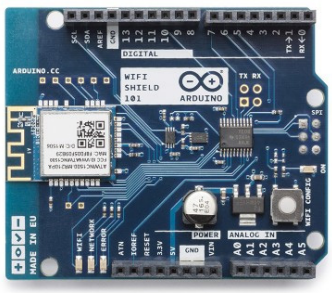

Figura 25 Arduino WiFi Shield(2018) Recuperado de https://store.arduino.cc/arduino-wifi-101-shield

Wifi integrado en el microcontrolador: mediante el uso de la placa Arduino MKR1000 o en la placa ESP8266 (no pertenece a Arduino).

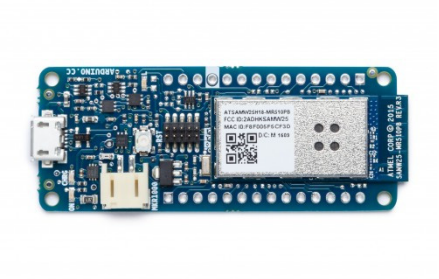

Figura 26 Arduino MKR1000 (2018) Recuperado de https://store.arduino.cc/arduino-mkr1000

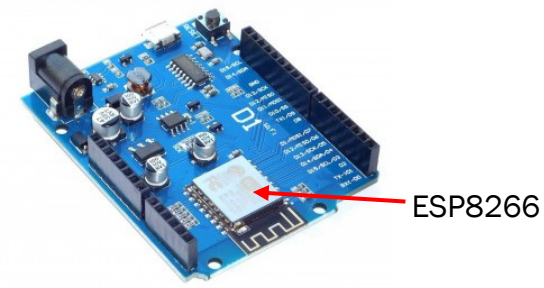

Figura 27 ARDUINO UNO COMPATIBLE WIFI BOARD BASED ON ESP8266EX (2018) Recuperado de https://store.qkits.com/arduino-uno-compatible-wifi-board-based-on-esp8266ex.html

El software de Arduino cuenta con librerías que sirven para conectar la placa con una red inalámbrica wifi sin configuración previa, tan solo es necesario el SSID (*Service Set Identifier*) y la contraseña.

C. Comunicación Bluetooth: consiste en la comunicación con el Arduino mediante el protocolo bluetooth, para ello se pueden utilizar varias formas, ya sea utilizar módulos externos, módulos integrados o shields específicas.

(“Comunicaciones Arduino”, 2018; “Arduino-Ethernet” 2018; “Arduino-WiFi101”, 2018)

D. Comunicación mediante el puerto serie: esta comunicación consiste en el envío de datos mediante el envío secuencial de paquetes de un bit, sobre un canal de comunicación.

Todas las placas de Arduino cuentan con un puerto serie disponible. Para ello utilizan los pines digitales 0 RX (datos) y 1 TX (reloj) compartido con el USB (estos pines no se podrán utilizar si está habilitada la comunicación serie). Algunas placas cuentan con puertos serie adicionales ubicados en otros pines, que no tienen por qué estar conectados al USB del Arduino.

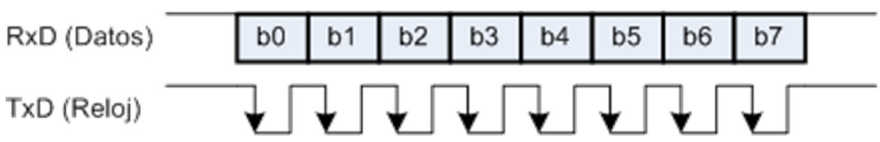

Figura 28 Aprendiendo Arduino (2018) Recuperado de https://aprendiendoarduino.wordpress.com/2016/07/02/comunicacion-serie-arduino/

Para recibir estos datos, el microcontrolador cuenta con un buffer que se encarga de ir almacenándolos hasta que mediante el comando read() son leídos y se vacían usando el método FIFO (*First In First Out*). Este buffer, tiene una capacidad limitada y cuando este se llena el resto de datos recibidos no se almacenan y se pierden.

Dentro del lenguaje de Arduino existen todos los comandos necesarios para la comunicación con el puerto serie. Las principales funciones de esta librería son:

- Serial.begin(): establece la velocidad en baudios para la comunicación. Además, se pueden configurar otros valores como el número de bits de datos, la parida y los bits de stop.
- Serial.available(): devuelve el número de bytes disponibles. Si hay bytes esperando a ser leídos devolverá un número mayor que 0.
- Serial.read(): se encarga de leer los datos almacenados en el buffer.
- Serial.print(): escribe datos en el puerto serie con formato de texto ASCII.
- Serial.write(): realiza la misma acción que Serial.print pero en binario.

(“Comunicación Serie Arduino”,2018)

3. Diseño e implementación del control (Desarrollo del TFG)

3.1 Consideraciones previas

Una vez expuestas las distintas opciones tanto de comunicación, programación (Arduino y LabVIEW) y placa de control, tipos de motor paso a paso y teniendo en cuenta los objetivos del proyecto, que serán:

> *Realizar un control del motor paso a paso en el cual tanto el código de programación como el propio control tengan una finalidad didáctica, permitiendo entender el funcionamiento y las posibilidades de uso de los motores paso a paso, así como el control a través de un PC.*

Para ello se fijan las siguientes funciones:

1. En el control físico se podrá girar en ambos sentidos de manera automática, así como parar el motor mediante el uso de tres pulsadores. A su vez se programará la secuencia automática de tal modo que con una lectura del código se entienda que está realizando el dispositivo Arduino y por qué el motor gira en un sentido u otro. Además, se dotará de otros cuatro pulsadores mediante los cuales se pueda alimentar las bobinas de manera manual, con el objeto de poder realizar una secuencia, tanto de paso completo como medio paso o incluso no realizar ninguna secuencia válida para el giro del motor, para poder comprender de mejor manera, como se orienta el eje del motor en función de las fases que se encuentren alimentadas.

2. En el control vía PC, se podrá hacer girar en ambos sentidos y parar el motor con tres botones que simulen a los tres botones físicos homónimos, ambos grupos de botones (físicos y digitales) podrán interactuar con el dispositivo de manera simultánea. Además, se podrá variar la velocidad de giro del motor, de manera que se ilustre un mayor número de posibilidades de uso.

3. Se situarán cuatro diodos led de tal forma que se iluminen en función de que fase del motor este alimentada en cada instante generando la sensación de giro para que se pueda visualizar la correspondencia entre las acciones de la placa controladora y el giro del motor.

Con el objeto de ganar versatilidad y una mayor visibilidad de los pasos del proceso de control, se selecciona una placa Arduino Nano y un método de programación por el cual se programe en el software de Arduino sin utilizar librerías y una comunicación entre Arduino y PC vía puerto serie, que hará que la programación en LabVIEW se realice usando los bloques básicos y los correspondientes a la comunicación con puerto serie.

3.2 Hardware

3.2.1 Descripción básica:

El hardware necesario para realizar este proyecto consistirá en un motor paso a paso, una placa de Arduino y una placa con circuito impreso en la cual se conectará el Arduino y los demás dispositivos necesarios para el control del motor y para la visualización de dicho control.

En este caso el motor será un motor paso a paso híbrido de la norma Nema17 de 200 pasos por vuelta y la placa Arduino será una placa Arduino Nano, debido a su pequeño tamaño y a que tiene funciones similares a las del Arduino UNO (placa básica).

Para el control físico del motor se colocarán 7 pulsadores y para la visualización de la alimentación de cada fase un diodo led con su correspondiente resistencia de protección. Además, puesto que la corriente proporcionada por el Arduino es muy baja, se utilizará un driver Darlington ULN2803A.

3.2.2 Esquema eléctrico:

Se asignan los pines del 6 al 12 para los pulsadores de control. Estos pulsadores se conectan directamente a tierra (GND) de tal forma que al pulsar se igualen potenciales entre el pin y la tierra y el dispositivo vea un 0. Se asignan los pines 2 al 5 para la alimentación de las fases del motor y se conectan con las entradas del driver Darlignton ULN2803A, a estos mismos puntos se conectan unos diodos led con su correspondiente resistencia para que al alimentar el motor se encienda un diodo. Se parean las entradas y salidas del ULN2803A con el objetivo de conseguir reducir la corriente que pasa por los transistores a la mitad. Se conectan las salidas del driver al motor. Ambos cables cuya conexión corresponde con la mitad de los bobinados se conectan a la alimentación, que a su vez se conectara con la pata correspondiente del driver. Finalmente se ponen todos los puntos de tierra en común.

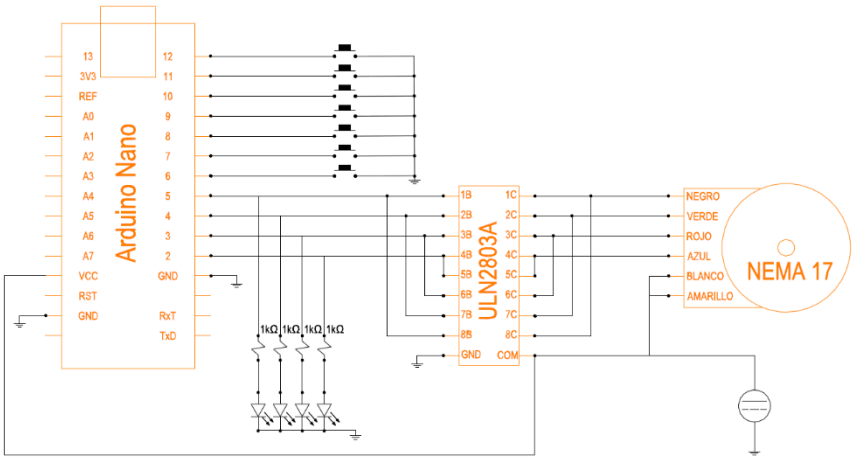

Figura 29 Esquema eléctrico. Fuente: plano esquema eléctrico (elaboración propia)

3.2.3 Motor paso a paso:

Debido a que se quiere unir la alimentación del motor con la del dispositivo Arduino, se selecciona un motor cuya tensión nominal se asimile a la tensión de alimentación del Arduino. Además, no se necesita un motor de grandes dimensiones puesto que se trata de un proyecto ilustrativo. Por lo tanto, el motor seleccionado es el siguiente:

El motor con el que vamos a trabajar es un Nema 17, SY42STH47-1206A, figura 30, que se corresponde con un motor paso a paso híbrido unipolar de 200 pasos por vuelta, tensión nominal 4V y corriente nominal 1.2A.

Figura 30 Motor paso a paso Nema 17 (2018) recuperado de https://tienda.bricogeek.com/motores-paso-a-paso/546-motor-paso-a-paso-nema-17-32kg-cm.html?search_query=nema+17&results=9

Como se puede observar, figura 31, es un motor unipolar de 6 hilos.

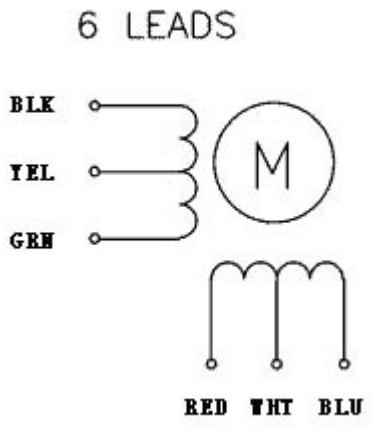

Figura 31 Esquema unifilar del motor. Fuente: Hoja de datos de SY42STH47-1206A

3.2.4 Controlador del motor ULN2803A:

Dado que las señales que emite el microcontrolador (Arduino) son de baja intensidad será necesario amplificar dichas señales y para ello se utiliza el ULN2803A, figura 32, que es una matriz de transistores Darlington. El dispositivo consiste en 8 pares Darlington npn. Cabe la posibilidad de conectar en paralelo estos pares para conseguir una capacidad de corriente superior. Se selecciona este modelo por encima del ULN2003 ya que al tener el ULN2803 un par Darlington más podemos obtener cuatro pareados y así llegar a la tensión requerida por el motor de manera más sencilla y provocando menor desgaste al dispositivo, ya que su corriente nominal es de 500 mA y el motor requiere una corriente cercana a 1A.

Figura 32 Driver Darlington 8-Canales ULN2803(2018) Recuperado de https://tienda.bricogeek.com/componentes/60-driver-darlington-8-canales-uln2803-dip.html

3.2.5 Microcontrolador Atmel AVR 328

Este microcontrolador, figura 33, se trata de un microchip de alto rendimiento que combina una memoria flash ISP de 32kB con capacidades de lectura/escritura simultánea. Cuenta con 23 E/S generales, 32 registros de trabajo general, 3 contadores/relojes flexibles con modos de comparación, interrupciones interna y externa, programación serie USART, 2 byte orientados a la comunicación serial, 6 canales de 10-bit convertidores de A/D. El dispositivo funciona entre 1,8-5,5 V.

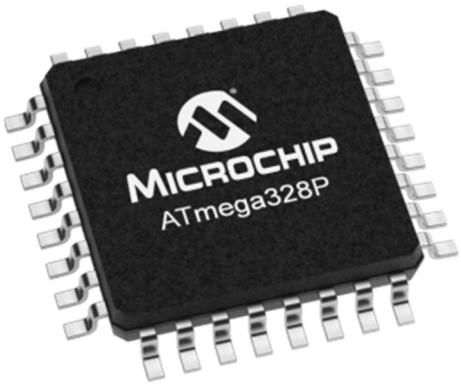

Figura 33 ATmega328P (2018) Recuperado de https://www.microchip.com/wwwproducts/en/ATmega328P

(“Microchip Technology”,2018)

3.2.6 Arduino Nano

Arduino Nano es una placa de pequeño tamaño basada en el microchip Atmega328 con características similares al Arduino UNO, figura 34. Esta placa se programa mediante el software Arduino Software (IDE). Sus características son las siguientes:

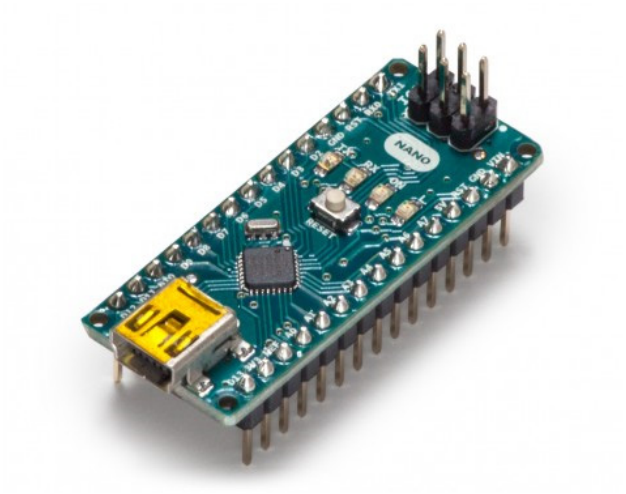

Figura 34 Arduino Nano (2018) Recuperado de https://store.arduino.cc/arduino-nano

- Microcontrolador: Atmel ATmega328 (ATmega168 versiones anteriores)
- Tensión de Operación (nivel lógico): 5 V
- Tensión de Entrada (recomendado): 7-12 V
- Tensión de Entrada (límites): 6-20 V
- Pines E/S Digitales: 14 (de los cuales 6 proveen de salida PWM)
- Entradas Analógicas: 8 Corriente máxima por cada PIN de E/S: 40 mA
- Memoria Flash: 32 KB (ATmega328) de los cuales 2KB son usados por el bootloader.
- SRAM: 2 KB (ATmega328) (1 KB ATmega168)
- EEPROM: 1 KB (ATmega328) (512 bytes – ATmega168)
- Frecuencia de reloj: 16 MHz
- Dimensiones: 18,5mm x 43,2mm

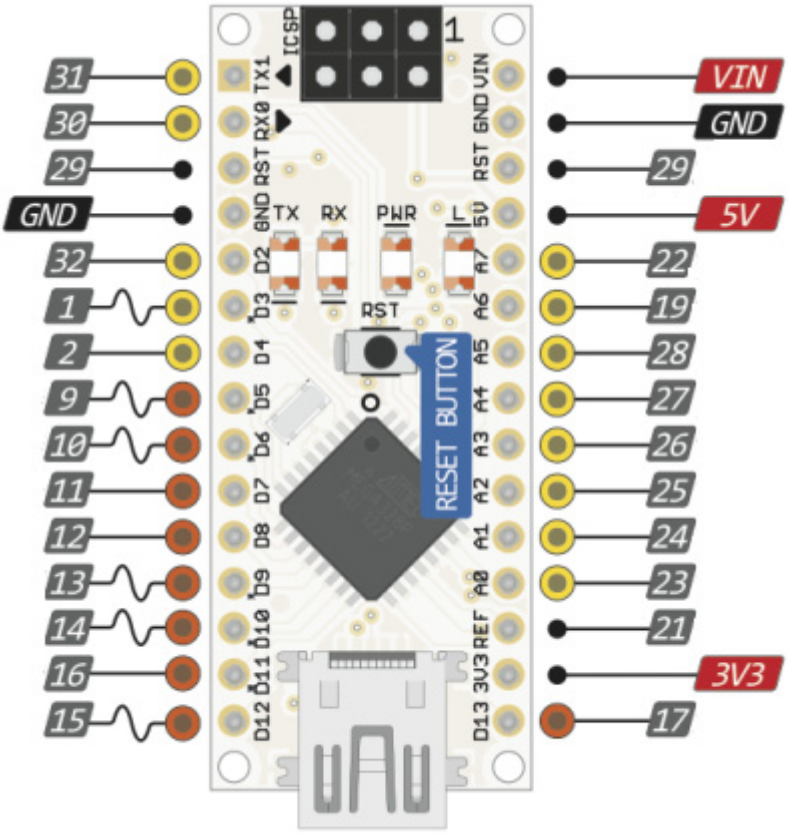

Figura 35 Arduino Nano V3(2018) Recuperado de https://electronilab.co/tienda/arduino-nano-v3-atmega328-5v-cable-usb/

El Arduino Nano se conecta mediante un cable USB Mini-B, además de establecer una conexión con el ordenador también se usa como alimentación para la placa, no obstante Arduino Nano puede ser alimentado de otras formas ya que posee una selección automática de la alimentación. Puede ser alimentado de las siguientes maneras:

- Vía USB Mini-B.
- Una fuente de alimentación no regulada 6-20V (pin30).
- Una fuente de alimentación de 5V (pin27).

Cuando se conecta vía USB la placa proporciona 3,3V. Mientras que al alimentarse por otras vías ese pin no se encuentra disponible.

("Arduino Nano V3 – ATmega328 5V + Cable USB Compatible", 2018)

3.2.7 Placa del circuito impreso:

Debido al gran número de conexiones necesarias para realizar el circuito eléctrico necesario para el control del motor, el montaje requiere excesivo cableado lo que dificulta mucho el manejo y la visualización de los distintos componentes utilizados. Por ello se busca crear un circuito impreso del cual se obtenga un montaje más compacto y portátil.

3.2.7.1 Diseño:

Para el diseño del circuito impreso en primer lugar se han de medir los distintos elementos que se van a utilizar mediante un calibre de precisión, puesto que el diseño de la placa se va a realizar mediante AutoCAD utilizando una escala de 1:1. Una vez medidos todos los componentes y se reproducen en AutoCAD con la misma escala (1:1), figuras 36, 37, 38 y 39.

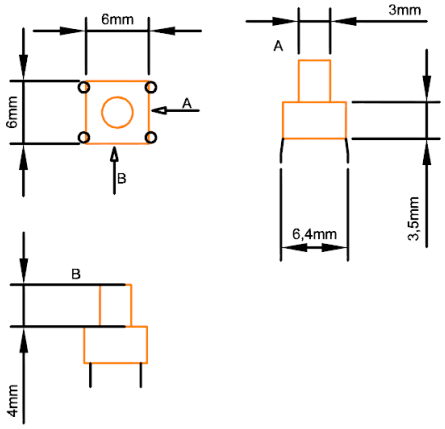

Figura 36 Pulsador. Fuente: plano piezas (elaboración propia)

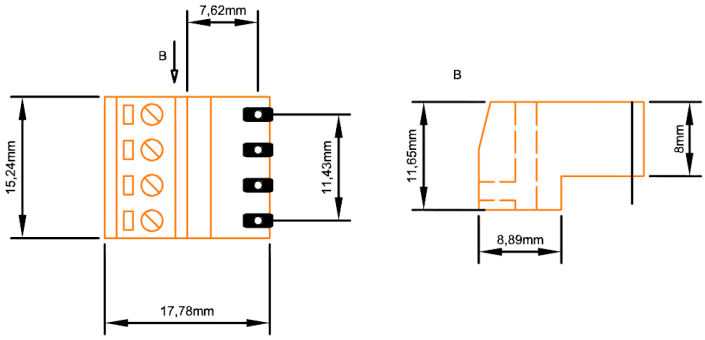

Figura 37 Conector PCB. Fuente: plano piezas (elaboración propia)

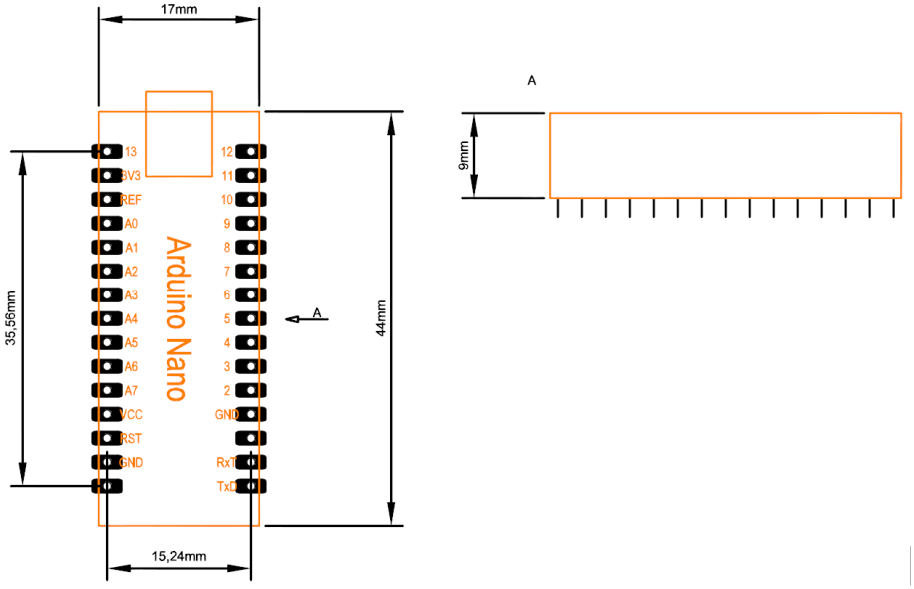

Figura 38 Arduino Nano+ Zócalo. Fuente: plano piezas (elaboración propia)

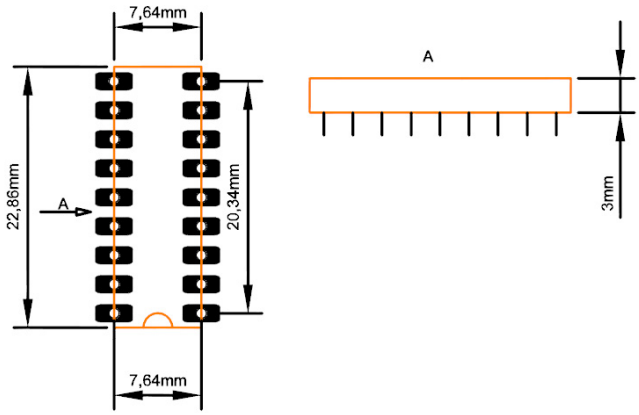

Figura 39 Zócalo 18DIP. Fuente: plano piezas (elaboración propia)

Previamente al diseño de la placa con el circuito impreso. Se han probado todos los componentes realizando el circuito mediante cableado y el uso de protoboards, figura 40.

Figura 40 Recuperado de https://www.amazon.es/ELEGOO-Breadboard-Prototipo-Soldaduras-Distribuci%C3%B3n/dp/B06Y3MNGVN/ref=sr_1_41?ie=UTF8&qid=1536571530&sr=8-41&keywords=protoboard

Además, en vez de utilizar el Arduino Nano se ha utilizado para estas pruebas una placa Arduino Uno Rev3, figura 41, de mayor tamaño pero que puede realizar las mismas funciones que el Arduino Nano, con la diferencia que el Uno está mejor adaptado al cableado.

Figura 41 Arduino (2018) Recuperado de https://store.arduino.cc/arduino-uno-rev3

A continuación, partiendo del esquema eléctrico se realiza un boceto en el que, teniendo en cuenta la posterior funcionalidad requerida (por ejemplo, todos los pulsadores juntos o los leds situados en las esquinas de un cuadrado), se realiza un boceto en el que se sitúan los componentes y de esa manera se estima el tamaño de la placa que será necesaria. A partir de las medidas estimadas de la placa y de la colocación de los distintos componentes se reproduce el esquema sobre la placa. Teniendo en cuenta que aquellas pistas que vayan a estar sometidas a mayor corriente como pueden ser la pista de alimentación, la de retorno de GND o las pistas que combinen dos salidas del ULN2803A.

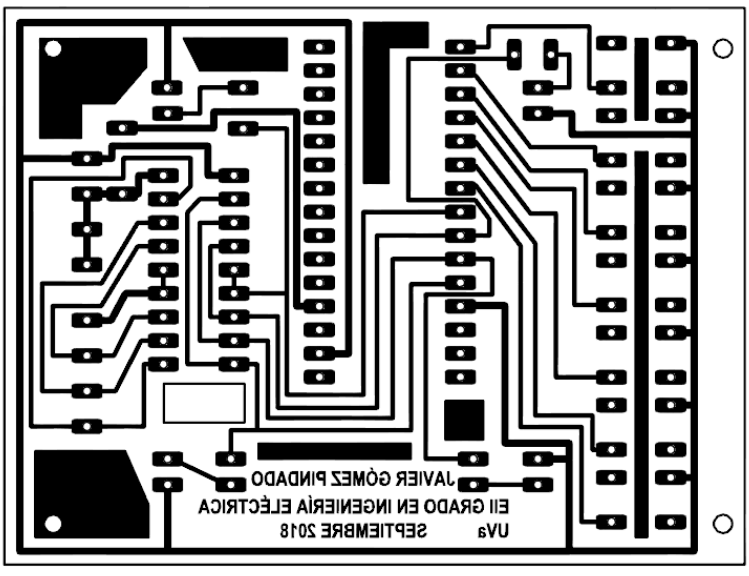

Figura 42 Circuito impreso. Fuente: plano placa circuito impreso (elaboración propia)

3.2.7.2 Fabricación:

El primer paso para la fabricación de la placa es imprimir en una lámina de acetato transparente la placa que se ha diseñado, así quedara una parte opaca que corresponde con el circuito a imprimir y una parte transparente. Una vez impresa la placa se procede a insolar la placa. Este proceso consiste en colocar el circuito impreso sobre la cara de cobre de una placa que está dotada de un barniz fotosensible (evitando que en este proceso esté expuesta a la luz), cuando el acetato y la placa estén alineados correctamente se expone la placa a luz ultravioleta en una insoladora durante 2'30'', quedando tapada la zona correspondiente al circuito impreso.

Figura 43 Insoladora. Fuente: elaboración propia

Una vez insolada la placa se introduce en un revelador, que consiste en sosa cáustica (NaOH) disuelta en agua. A los 30''-40'' se puede observar como el material fotosensible que ha estado expuesto a la luz.

Una vez retirado el material fotosensible se introduce la placa en la mezcla de dos disoluciones, una de HCl y otra de H_2O_2 para que reaccionen con el cobre que ha quedado expuesto y de esa forma lo retiren de la placa. Se produce la siguiente reacción.

$$CU + 2HCl + H_2O_2 = CUCl_2 + 2H_2O$$

Ahora, ha quedado sobre la placa solo el cobre correspondiente al circuito, se aclara la placa. A continuación, se realizan los agujeros en los que posteriormente se introducirán las patas de los componentes y mediante disolvente se retira el barniz restante de la placa.

("Revelado y atacado de nuestros circuitos impresos PCB", 2018)

Una vez que se ha comprobado que el esquema eléctrico y los componentes funcionan correctamente se procede a la unión de los componentes a la placa mediante soldadura blanda con estaño, así, se unen tanto física como eléctricamente, y se prueba para asegurarse de que el circuito impreso es el correcto.

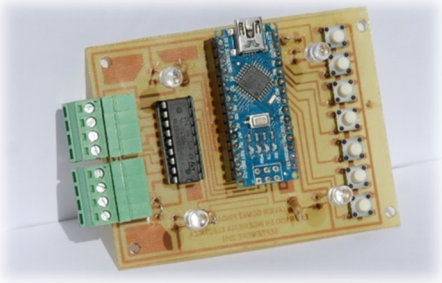

Figura 46 Placa de control1. Fuente: elaboración propia

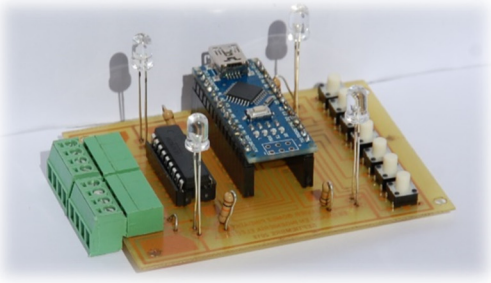

Figura 44 Placa de control2. Fuente: elaboración propia

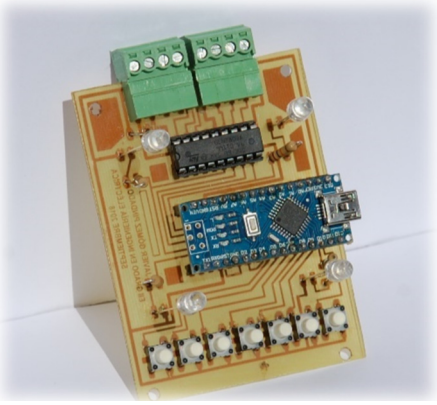

Figura 45 Placa de control3. Fuente: elaboración propia

3.2.8 Accesorios:

Además de los componentes anteriormente citados, se utilizan:

- 4 diodos led que servirán para identificar cuando está alimentado cada uno de los pares de bobinas. Se colocan en la placa formando las cuatro esquinas de un cuadrado de tal manera que al encenderse además de señalizar que la fase está alimentada, simulen el giro que está realizando el motor.
- 7 pulsadores para controlar de manera física el motor. En sentido ascendente se encuentran: paro, marcha en sentido horario, marcha en sentido antihorario y los cuatro últimos se corresponden con pulsadores que cierran los distintos circuitos de alimentación del motor, tomando como referencia el color del cableado del motor, en sentido ascendente se activarán los siguientes circuitos: negro, verde, rojo y azul.
- 4 resistencias de 1kΩ conectadas en serie con los diodos led para evitar el cortocircuito cuando pase corriente por ellos.

- 2 zócalos de 1 fila y 15 columnas para montar el Arduino Nano.
- 1 zócalo de 2 filas y 9 columnas para montar el ULN2803A.
- 2 conectores PCB externos de 4 puntos de conexión cada uno para realizar la conexión con los terminales del motor y con la fuente de alimentación.
- Fuente de alimentación de corriente continua de 5V y 2A.

3.2.9 Envolvente:

Con el objetivo de proteger los componentes de la placa de circuito impreso, se va a diseñar una caja envolvente de metacrilato.

Diseño:

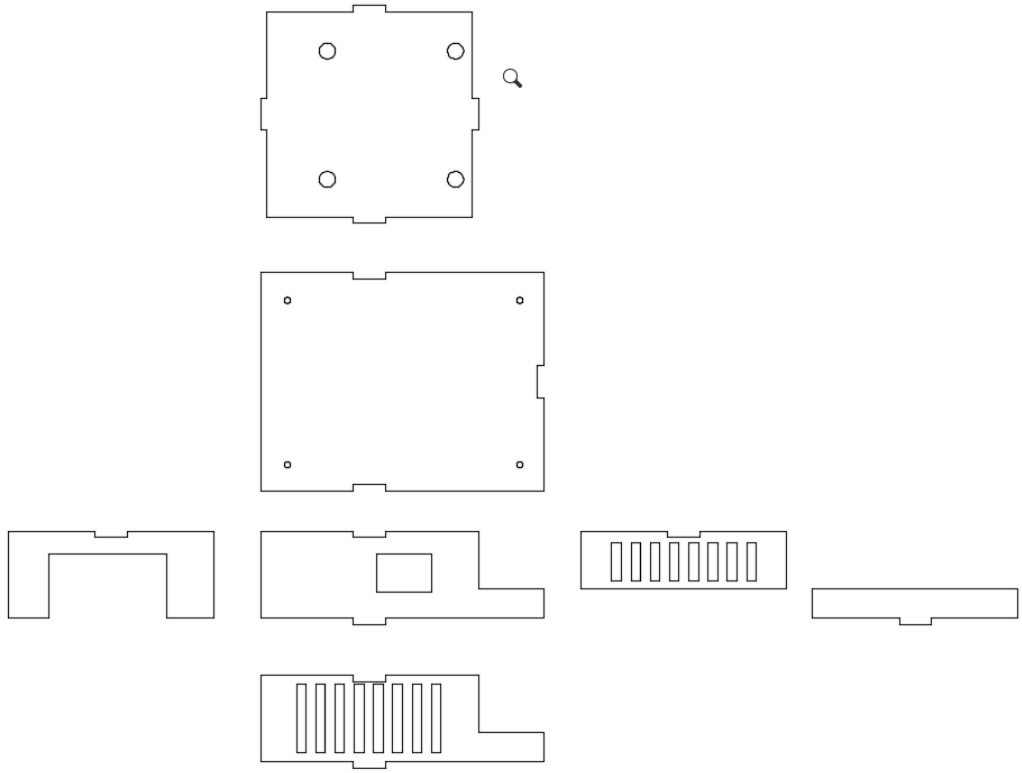

Figura 47 Piezas envolvente. Fuente: elaboración propia

Fabricación:

Se cortan y se ensamblan las distintas partes de la envolvente.

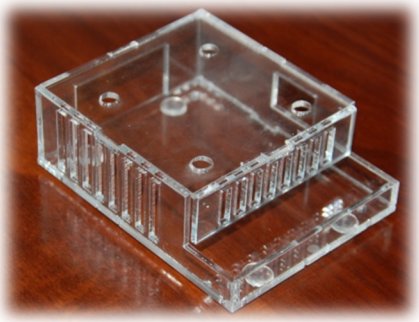

Figura 48 Envolvente. Fuente: elaboración propia

3.3 Software

Puesto que se ha elegido como placa de control una placa Arduino Nano, se utilizará el software de Arduino para la programación del dispositivo. Además, el control vía PC se realizará utilizando el software de LabVIEW. Como se pretende que el dispositivo pueda funcionar, tanto si está conectado al ordenador, como si no lo está, en LabVIEW se programará simplemente realizando una lectura y escritura en el puerto serie, sin utilizar ninguna librería de control para Arduino.

En la programación de Arduino tampoco se utilizará ninguna librería específica para el control de los motores paso a paso, de esta manera se podrá ver el programa con detalle y se podrá explicar las secuencias de control y funcionamiento del motor paso a paso.

3.3.1 Protocolo de comunicación entre LabVIEW y Arduino

Dado que va a existir un flujo de comunicaciones en ambos sentidos, tanto Arduino como LabVIEW van a enviar y recibir información se tendrá que realizar un protocolo de comunicaciones que evite la pérdida de datos o la recepción de órdenes erróneas.

Para ello se programarán ambos programas de tal forma que estén leyendo datos de manera permanente y solo manden información cuando ocurra un evento, ya sea una interrupción en el microprocesador del Arduino o la pulsación de algún botón de la interfaz de LabVIEW.

El dispositivo Arduino se encontrará de manera permanente leyendo el buffer del puerto serie, en el momento en el que detecte un byte comenzará a leer los caracteres recibidos, si reconoce el byte de inicio “<” vaciará una variable llamado comandoTexto, a continuación, empezará a almacenar en dicha variable los caracteres leídos hasta que encuentre el byte de fin de comando “>”.

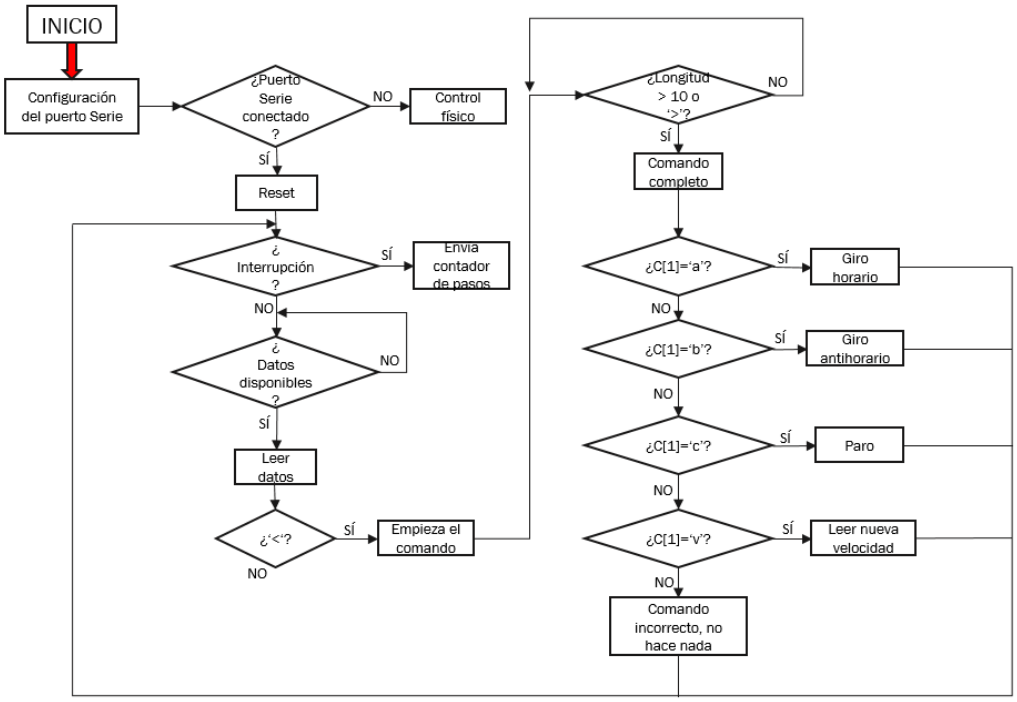

Diagrama 1 Protocolo comunicación Arduino. Fuente: elaboración propia

```
while(Serial.available()){
    char c = Serial.read();
    if(c=='<')                    //  Inicio de nuevo comando.
      sprintf(comandoTexto,"");
    else if(c=='>')               // Fin de nuevo comando.
      interpreta(comandoTexto);
    else if(strlen(comandoTexto)<10)  // Si comandoTexto tiene espacio,
                                      //se lee el carácter.
      sprintf(comandoTexto,"%s%c",comandoTexto,c);
                                // Si no, se ignora y se busca  '<' o '>'.
}
```

Código 1

Se configurará el microcontrolador para que cada cierto tiempo se produzca una interrupción en el programa, en ese momento el programa activará una variable que se encargará de dar la orden al dispositivo de enviar un dato (el número de pasos dados por el motor). Estas interrupciones se producen utilizando un contador interno que está programado para que avance en 1 cada 1024 ciclos, por lo que, llegando a un caso extremo, por cada dato enviado, podría leer 1024 comandos, de esta manera, se asegura que no haya pérdida de información.

Una vez leído el comando, se pasa a interpretarlo. El byte posterior al de inicio será el que especifique la que orden se le va dar al dispositivo. De esta forma tenemos:

- <a>: giro horario.
- <b>: giro antihorario.

- <c>: paro.
- <vxxxx>: este comando es el encargado de variar la velocidad, la v le indica al dispositivo que tiene que leer los otros 4 caracteres y conformar con ellos un valor numérico correspondiente al valor del contador para realizar la interrupción. Con el objetivo de que si la última cifra es 0 no se interprete como un valor nulo en la programación, esta velocidad se recibe invertida y se reconvierte dentro del programa.

```
void interpreta(char *comando){
  switch(comando[0]){
    case 'a':       // Giro horario.
      dir=2;
      if(bobina==0) bobina=2;
      break;
    case 'b':       // Giro antihorario.
      dir=1;
      if(bobina==0) bobina=2;
      break;
    case 'c':      // Paro.
      dir=0;
      break;
    case 'v':       // Cambiar velocidad (valores entre 391 y 9999)
      if (comando[4]>0){
      vel = ((comando[4]-'0')*1000+(comando[3]-'0')*100+ (comando[2]-'0')*10+(comando[1]-'0'));
      }
      else{
         vel = ((comando[3]-'0')*100+ (comando[2]-'0')*10+(comando[1]-'0'));
      }
      break;
 }
}
```

Código 2

A su vez, LabVIEW estará leyendo en cada ciclo del bucle while en el que se basa su funcionamiento el valor de los pasos que envía el dispositivo Arduino. Si en algún momento se pulsa alguno de los botones de control, el programa enviará un comando de la forma <XXX> y continuará leyendo. Para evitar la pérdida de datos recibidos, se fijará como velocidad máxima de las interrupciones (velocidad del motor) el intervalo de tiempo que tarda en repetirse cada ciclo del bucle while de LabVIEW (50ms).

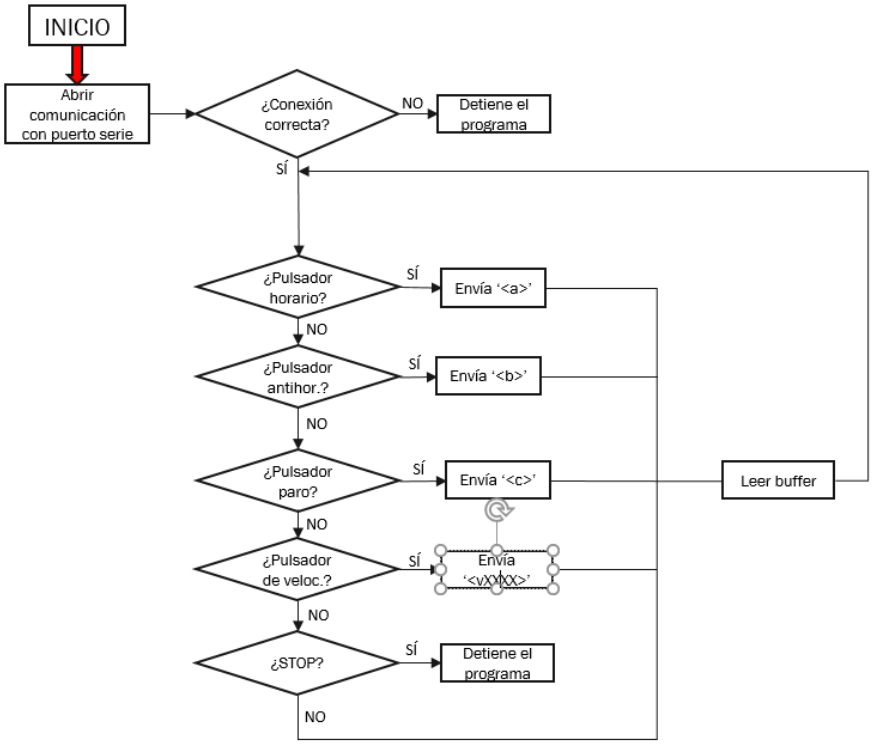

Diagrama 2 Protocolo comunicación LabVIEW. Fuente: elaboración propia

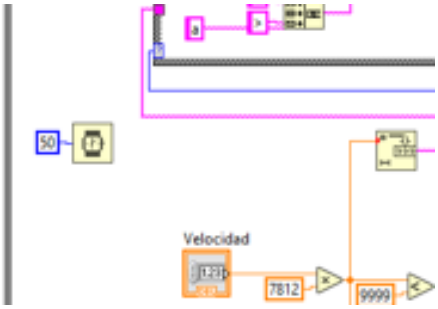

Figura 49 Retardo bucle while. Fuente: elaboración propia

Por lo tanto, en este caso el ordenador funcionará como maestro y el dispositivo Arduino como esclavo.

3.3.2 Programa de control del motor paso a paso (Arduino):

Se programará el dispositivo de tal forma que esté leyendo de manera permanente los estados exteriores, ya sean pulsadores o entradas al puerto serie y actúe en función de estas, el dispositivo solo enviará información al puerto serie tras una interrupción programada en el microprocesador, de forma que no interfiera con su capacidad de recibir datos. Podrá funcionar tanto independiente de un ordenador (sin conexión USB), como conectado al ordenador, en este caso el puerto serie estará disponible y el dispositivo enviará y recibirá información. Solo se podrá variar la velocidad mediante la comunicación por el puerto serie y el dispositivo solo registrará el número de pasos dados si actúa mediante la secuencia automática, ya sea activada

mediante pulsadores o mediante comunicación por puerto serie. Se podrá trabajar de manera simultánea con pulsadores y con comunicación serie.

Se decide programar utilizando interrupciones en vez de funciones de Arduino más sencillas como delay() o millis() ya que estas dos implican un funcionamiento erróneo a la hora de interactuar con lecturas o escrituras en el puerto serie, pues estas funciones implican tener al programa esperando y se podría perder información.

A continuación, se describirá el código de programación Arduino utilizado:

En primer lugar, se definen los distintos pines que se van a utilizar del Arduino y las distintas variables.

```
#define black 2     //Pin que alimenta la fase NEGRA
#define green 3     //Pin que alimenta la fase VERDE
#define red 4       //Pin que alimenta la fase ROJA
#define blue 5      //Pin que alimenta la fase AZUL
#define paro 6      //Pin que lee el pulsador de paro
#define horario 7   //Pin que lee el pulsador de giro horario
#define antihorario 8 //Pin que lee el pulsador de giro antihorario
#define negro 9     //Pin que lee el pulsador para la fase NEGRA
#define verde 10    //Pin que lee el pulsador para la fase VERDE
#define rojo 11     //Pin que lee el pulsador para la fase ROJA
#define azul 12     //Pin que lee el pulsador para la fase AZUL
```

Código 3

Variables del sketch:

```
int dir=0, cont;
int vel=7812;
char comandoTexto[10];
volatile boolean chivato=false;
int bobina=0;
boolean led = false;
```

Código 4

La variable dir corresponde con el sentido de giro teniendo tres opciones:

- 0=paro.
- 1=giro antihorario.
- 2=giro horario.

La variable será el contador que irá acumulando los pasos dados por el motor, para el sentido horario contara en positivo (0, 1, 2, ...) y para el sentido antihorario (0, -1, -2, ...), vel se corresponde con la velocidad (número de ciclos para establecer la interrupción), comandoTexto es el comando recibido por puerto serie, bobina se utiliza para realizar la secuencia de giro y led sirve para controlar la activación de pines en la interrupción.

A continuación, en el setup() se configura el modo de trabajo de los pines (pinMode) anteriormente definidos como entrada (INPUT_PULLUP) aquellos correspondientes a los pulsadores y como salidas (OUTPUT) los encargados de alimentar a las distintas fases del motor y se habilita el puerto serie y la velocidad de comunicación (Serial.begin).

```
void setup(){
  Serial.begin(9600);
  pinMode(black, OUTPUT);
  pinMode(green,OUTPUT);
  pinMode(red,OUTPUT);
  pinMode(blue,OUTPUT);
  pinMode(horario,INPUT_PULLUP);
  pinMode(antihorario,INPUT_PULLUP);
  pinMode(paro,INPUT_PULLUP);
  pinMode(negro,INPUT_PULLUP);
  pinMode(verde,INPUT_PULLUP);
  pinMode(rojo,INPUT_PULLUP);
  pinMode(azul,INPUT_PULLUP);
```

Código 5

Una vez configurados los pines de entrada y salida, se pasa a configurar las interrupciones y se cierra el setup().

Estas interrupciones de tiempo (timer) o de software consisten en una forma especial de programar por la que indicamos al Arduino que interrumpa la ejecución del programa principal y ejecute otro tipo de tareas. Gracias a esto, se pueden conseguir cosas como que el Arduino simule realizar dos tareas a la vez.

Las interrupciones dependen del reloj del micicrocontrolador, ya que serán configuradas en función de la frecuencia de dicho reloj. En el caso de Arduino esta frecuencia será de 16MHz.

Puesto que los microprocesadores no comprenden lenguajes como C/C++, que es el lenguaje con el que trabaja Arduino, cuando se va a trabajar con interrupciones no se deben utilizar determinadas funciones como millis() o delay(), ya que su implementación requiere el uso del timer del microprocesador y daría lugar a comportamientos no deseados.

A continuación, se van a explicar algunos conceptos necesarios para comprender el funcionamiento de los timers, encargados de las interrupciones:

- Contadores:
  Los contadores son registros que se van incrementando con cada ciclo del reloj, en función de su configuración. Estos contadores son la base de los timers. El microprocesador Atmel 328 cuenta con contadores de 8bits y de 16bits.

Si nuestro microprocesador tiene una frecuencia de 16MHZ, los contadores de 8 bits desbordan (overflow) 16.000.000/256=62.500 veces por segundo. Esto es porque solo pueden contar de 0 a 255. Y los de 16bits podrán contar de 0 a 65.535, con lo que desbordarán 16.000.000/65.535≈244 veces por segundo. En este caso se utilizará el timer1 que tiene un contador de 16 bits.
Puesto que estos dos tiempos no son prácticos es necesario buscar una solución que permita cambiar la forma en que incrementamos el registro del contador. Esta herramienta, son los pre-escaladores (*prescaler*).

- Pre-escaladores:
  Son registros que se utilizan para configurar los contadores de manera que el contador incremente su valor en 1 cada un número determinado de ciclos.

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0		1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	clkI/O/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)

Figura 50 Atmel (2018) ATmega328/P datasheet

Se ha programado el prescaler 1024 que corresponde con 16.000.000/1024=15625 tics o incremento cada segundo.

- Comparadores:
  Se configuran de tal manera que comparen un valor con el estado del timer (contador), si el valor es el configurado, se realizará una determinada acción.
  (“Interrupciones de timer con Arduino”, 2018)

- Registros:
  La configuración de estas interrupciones se realiza mediante la modificación de registros del microprocesador que son posiciones específicas de la memoria RAM. De tal manera que en función de que configuración se pretenda conseguir, habrá que modificar unos registros u otros.
  En este caso se van a utilizar los registros:

- TC1 Control Register A

Name: TCCR1A
Offset: 0x80
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	COM1	COM1	COM1	COM1			WGM11	WGM10
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

Figura 51 Atmel (2018) ATmega328/P datasheet

- TC1 Control Register B

Name: TCCR1B
Offset: 0x81
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	ICNC1	ICES1		WGM13	WGM12	CS12	CS11	CS10
Access	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

Figura 52 Atmel (2018) ATmega328/P datasheet

- Timer/Counter 1 Interrupt Mask Register

Name: TIMSK1
Offset: 0x6F
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
			ICIE			OCIEB	OCIEA	TOIE
Access			R/W			R/W	R/W	R/W
Reset			0			0	0	0

Figura 53 Atmel (2018) ATmega328/P datasheet

- OCR1A (*Output Compare Register*): valor del comparador.
- TCNT1: es el contador.

Para el programa de control del motor paso a paso, se va a configurar un modo de comparación que corresponde a las siglas CTC (*Clear Timer on Compare Match*), que cada vez que el contador llega al valor del comparador, vuelve a poner el contador a 0. De esta manera variando el valor del comparador se variará también la frecuencia en la que ocurren las interrupciones y con ello la velocidad del motor. Para ello hay que configurar los registros de la siguiente manera:

- Los registros de TCCR1A se sitúan en 0.
- Todos los registros de TCCR1B salvo el WGM12 (valor 1) se sitúan en 0.

Mode	WGM13	WGM12 (CTC1)[1]	WGM11 (PWM11)[1]	WGM10 (PWM10)[1]	Timer/ Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX

Figura 54 Atmel (2018) ATmega328/P datasheet

- En el registro TIMSK1 se sitúa el valor OCIE1A en 1 para habilitar el comparador del registro A (OCR1A).

```
  // Paramos todas las interrupciones antes de contigurar un timer.
  noInterrupts();
  // El registro de control A queda todo en 0.
  TCCR1A = 0;
  // Activamos el modo CTC en Timer1.
  TCCR1B = 0;
  TCCR1B |= (1 << WGM12);
  //Prescaler 1024: 16.000.000/1024=15625 tics/s.
  //Se genera un tic cada 1024 ciclos.
  TCCR1B |= (1 << CS12) | (1 << CS10);
  // Inicializamos el contador en 0
  TCNT1 = 0;
  // Registro comparador 7812 equivale a 0,5s.
  OCR1A = 7812;
  // Inicializamos el comparador para el registro A.
  TIMSK1 |= (1 << OCIE1A);
  // Activamos interrupciones nuevamente.
  interrupts();
}
```

Código 6

Antes de configurar las interrupciones tienen que ser inhabilitadas, para que no hay interferencias.

Previamente a programar en el loop() se van a programar las distintas funciones que aparecerán en la función principal.

- Función interpreta(char *comando) que interpreta el comando recibido por el puerto serie y actúa en función del valor recibido. La variable dir indica al programa en qué sentido debe girar el motor: 0 significa paro, 1 sentido horario y 2 sentido antihorario. La variable vel se utiliza para variar el valor del contador al que se producen las interrupciones y con ello la velocidad.

```
void interpreta(char *comando){
  switch(comando[0]){
    case 'a':       // Giro horario.
      dir=2;
      if(bobina==0) bobina=2;
      break;
    case 'b':       // Giro antihorario.
      dir=1;
      if(bobina==0) bobina=2;
      break;
    case 'c':      // Paro.
      dir=0;
      break;
    case 'v':       // Cambiar velocidad (valores entre 391 y 9999)
      if (comando[4]>0){
      vel = ((comando[4]-'0')*1000+(comando[3]-'0')*100+ (comando[2]-'0')*10+(comando[1]-'0'));
      }
      else{
         vel = ((comando[3]-'0')*100+ (comando[2]-'0')*10+(comando[1]-'0'));
      }
      break;
 }
}
```

Código 7

Se limita la velocidad a valores comprendidos entre 0,05 y 1,27 pasos partido segundo. Que equivale a 391 y 9999 tics respectivamente.

Además, una vez configuradas las interrupciones habrá que programar su función correspondiente:

- Interrupción ISR(TIMER1_COMPA_vect), esta interrupción se activa cuando el contador llega al valor de OCR1A.

```
ISR(TIMER1_COMPA_vect) {
chivato=true;
 }
```

Código 8

Cada vez que se produzca la interrupción se activará un chivato o *flag*, cuyo valor true o false se utilizará en el programa.

Por último, la función principal que se repetirá en cada ciclo del microprocesador, loop(), se compondrá del siguiente código:

```
void loop() {
if(digitalRead(7)==0){  //Giro horario
  dir=2;
  if(bobina==0) bobina=2;
}
if(digitalRead(8)==0){ //Giro antihorario
  dir=1;
  if(bobina==0) bobina=2;
}
if(digitalRead(6)==0) dir=0; //Paro
if(dir==0){
  digitalWrite(bobina,LOW);
  bobina=0;
}
```

Código 9

El dispositivo leerá el estado de los pines asociados a los pulsadores de paro, giro horario o giro antihorario en cada ciclo y modificará el valor de dir en función de cuál de ellos se haya activado.

A continuación, leerá el estado de los pulsadores correspondientes a la activación de la secuencia manual (NEGRO, VERDE, ROJO, AZUL). Aunque la señal proveniente de estos pulsadores indique que están pulsados, no se realizará ninguna acción siempre que no se haya pasado previamente por el paro.

```
//Activación de secuencia mediante pulsadores (previo paso por paro).
if(digitalRead(negro)==0 && dir==0) digitalWrite(black,HIGH);
if(digitalRead(negro)==1 && dir==0) digitalWrite(black,LOW);

if(digitalRead(verde)==0 && dir==0) digitalWrite(green,HIGH);
if(digitalRead(verde)==1 && dir==0) digitalWrite(green,LOW);

if(digitalRead(rojo)==0 && dir==0) digitalWrite(red,HIGH);
if(digitalRead(rojo)==1 && dir==0) digitalWrite(red,LOW);

if(digitalRead(azul)==0 && dir==0) digitalWrite(blue,HIGH);
if(digitalRead(azul)==1 && dir==0) digitalWrite(blue,LOW);
```

Código 10

Si el estado es pulsado, se alimentará la correspondiente fase, cuando no lo esté, se desactivará. Con el método manual se puede funcionar tanto en paso completo (*full step*) como en medio paso (*half step*).

Cuando la interrupción active el chivato, el dispositivo escribirá el número de pasos dados en el puerto serie y dejará de alimentar la fase actual. A continuación, dependiendo de la variable dir (sentido de giro) se realizará el siguiente paso de la secuencia que corresponda con dicho sentido de giro y se aumentará (+1) o se disminuirá (-1) el número de pasos acumulados.

```
if (chivato==true){ //Cuando la interrupción activa el chivato:

  Serial.println(cont); //Escribe el número de
                        //pasos dados por el puerto serie.
  digitalWrite(bobina,LOW);//Deja de alimentar la fase.
  if (dir==1){   //Secuencia automática giro antihoario.
    cont--;
       switch (bobina){
       case black:
       bobina=red;break;
       case green:
       bobina=blue;break;
       case red:
       bobina=green;break;
       case blue:
       bobina=black;break;}
     }
     if (dir==2){  //Secuencia automática giro horario.
      cont++;
       switch (bobina){
       case black:
       bobina=blue;break;
       case green:
       bobina=red;break;
       case red:
       bobina=black;break;
       case blue:
       bobina=green;break;}
     }
     digitalWrite(bobina,HIGH); //Alimenta la nueva fase.
     chivato=false; //Desactiva el chivato para que este código
                    //solo se reproduzca una vez.
}
```

Código 11

Al terminar de realizar el paso por la secuencia correspondiente, activa la nueva fase y desactiva el chivato para que esta parte del código se ejecute una sola vez tras haber activado el chivato la interrupción.

Las secuencias de giro serán:

- Sentido antihorario: Negro-Rojo-Verde-Azul- Negro-Rojo-Verde-Azul…
- Sentido horario: Negro-Azul-Verde-Rojo- Negro-Azul-Verde-Rojo…

Por último, si el puerto serie está disponible el dispositivo comenzará a leerlo, en el momento en el que detecte el carácter de inicio, en este caso “<”, comenzará a guardar los distintos caracteres en la variable comandoTexto, hasta llegar al límite de esta cadena de caracteres, 10, o hasta que aparezca el carácter de final de cadena, “>”. Si llega al límite de la cadena seguirá leyendo hasta que aparezca el carácter de final de cadena, pero no registrará más caracteres. En ese momento dará la cadena por terminada y llamará a la

función interpreta(char *comando) enviando como variable la cadena registrada comandoTexto.

```
while(Serial.available()){
    char c = Serial.read();
    if(c=='<')                    //  Inicio de nuevo comando.
      sprintf(comandoTexto,"");
    else if(c=='>')               // Fin de nuevo comando.
      interpreta(comandoTexto);
    else if(strlen(comandoTexto)<10)  // Si comandoTexto tiene espacio,
                                      //se lee el carácter.
      sprintf(comandoTexto,"%s%c",comandoTexto,c);
                                  // Si no, se ignora y se busca  '<' o '>'.
}


   OCR1A=vel; //El valor de la velocidad cambia el comparador para variar
              //el ritmo de las interrupciones.
}
```

Compilado

Código 12

Por último se iguala el valor de OCR1A con el de la velocidad (vel), reprogramando la interrupción y variando así la velocidad del motor y se cierra la función principal loop(), dando por finalizado el programa (sketch).

3.3.3 Programa de control del motor paso a paso (LabVIEW).

Este programa está programado de tal forma que este leyendo constantemente en el puerto serie el registro del buffer del dispositivo Arduino y refleje esta lectura tanto en el indicador del lector del buffer como en un indicador numérico que simula el giro del motor. Cuando se active uno de los cuatro pulsadores de control el programa enviará un comando con la forma <XXX> al dispositivo, que está programado para recibir información en todo momento y que solo procesa comandos con la forma enviada. Acto seguido al envío de este comando, el programa continuará leyendo los datos recibidos por el dispositivo Arduino.

Para poder comunicarnos mediante el uso del puerto serie utilizando LabVIEW, antes, debemos descargar la librería VISA para LabVIEW.

3.3.3.1 Panel frontal:

En el panel frontal se situarán los controles necesarios para la preconfiguración del programa para una buena comunicación con el puerto serie, así como los controles e indicadores que se van a utilizar para controlar y mostrar los distintos estados del motor a través del dispositivo Arduino:

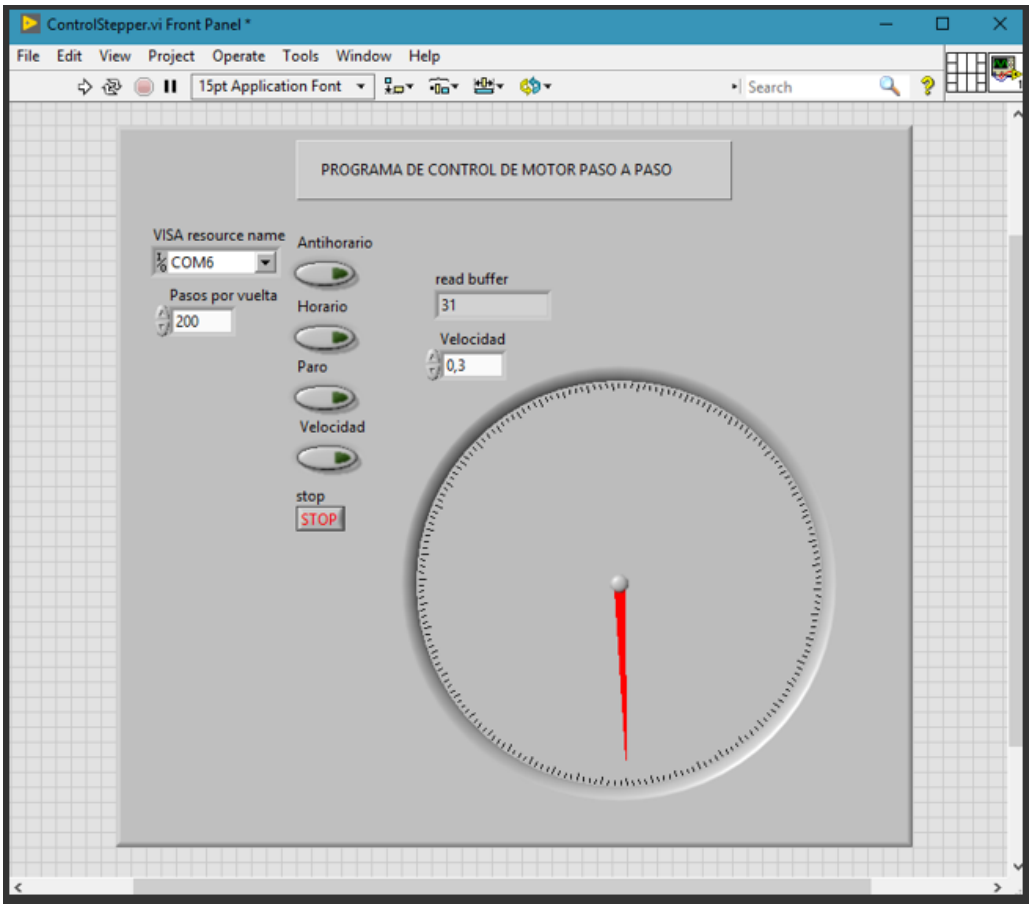

Figura 55 Panel frontal LabVIEW. Fuente: elaboración propia

Aparecen los siguientes elementos:

- Un selector de nombre VISA resource name cuyo objetivo es seleccionar el puerto COM con el que se va a comunicar el programa. Se debe seleccionar aquel COM que el ordenador haya asignado a nuestro dispositivo.

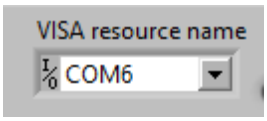

Figura 56 VISA resource name. Fuente: elaboración propia

- Cuatro pulsadores que realizan las siguientes tareas: giro en ambos sentidos, paro y enviar el nuevo valor de velocidad.

Figura 57 Botones de control. Fuente: elaboración propia

- Un control numérico para escribir el valor de la nueva velocidad, en pasos partido segundo.

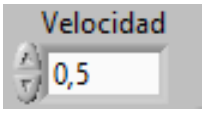

Figura 58 Control velocidad. Fuente: elaboración propia

- Un indicador que muestra la lectura del buffer del dispositivo.

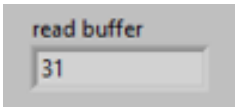

Figura 59 Lectura del buffer. Fuente: elaboración propia

- Un pulsador que finaliza el programa.

Figura 60 Parada del programa. Fuente: elaboración propia

- Un control numérico en el que se especifique el número de pasos por vuelta del motor a controlar.

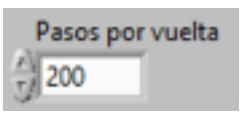

Figura 61 Pasos por vuelta. Fuente: elaboración propia

- Un indicador numérico que varía en función de la lectura del acumulado de los pasos que da el motor, de manera que simula el giro del motor.

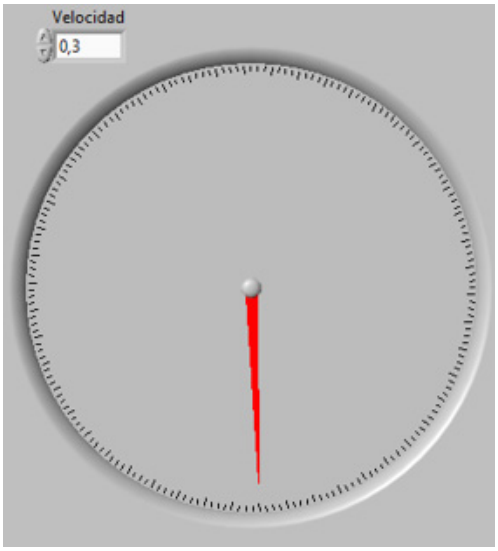

Figura 62 Indicador del motor. Fuente: elaboración propia

3.3.3.2 Diagrama de bloques:

Para analizar el diagrama de bloques vamos a dividirlo en dos partes, por un lado, la programación concerniente a la comunicación serie (VISA) y por el otro la programación que controla el envío de comandos y trata y muestra los datos adquiridos.

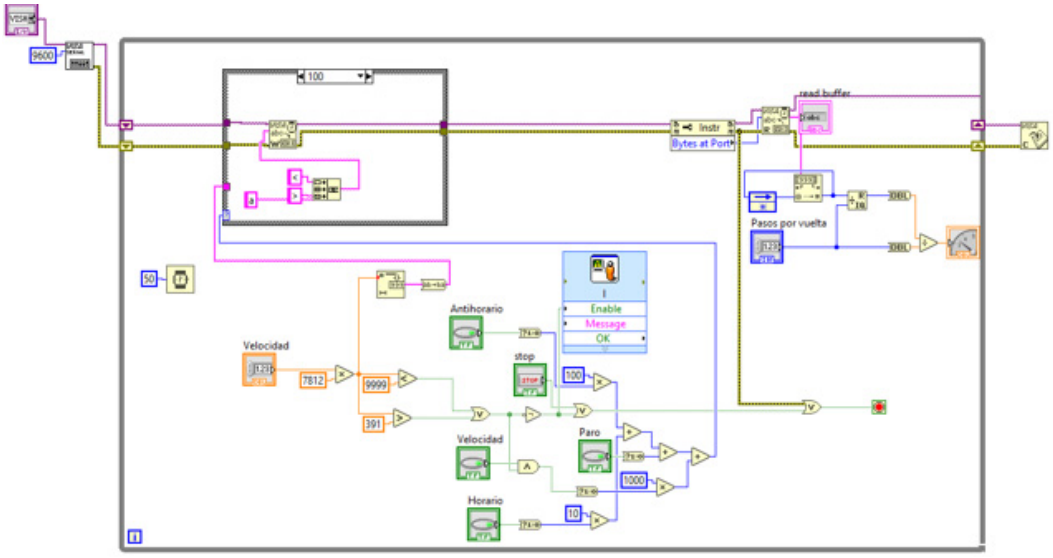

Figura 63 Diagrama de bloques LabVIEW. Fuente: elaboración propia

Comunicación con el puerto serie:

En primer lugar, se debe inicializar el puerto serie, para ello utilizamos el bloque llamado *VISA Configure Serial Port*, a pesar de que este bloque tiene un número elevado de entradas solo se modificarán dos valores: *VISA resource name* que es el encargado de seleccionar que puerto se quiere habilitar y *baud rate* que es la velocidad de comunicación del puerto serie (es importante que este valor

coincida con el valor programado en la placa de Arduino). Este bloque, figura 64, cuenta con dos salidas una de error y la otra que porta el nombre del puerto de comunicación. Estas dos salidas serán las que se cableen al resto de bloques relacionados con la comunicación.

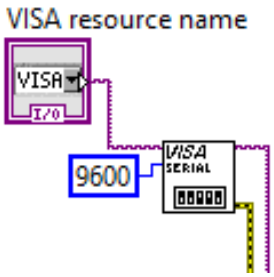

Figura 64 Bloque VISA resource name. Fuente: elaboración propia

A continuación, se configuran tanto el bloque de escritura como el de lectura. El bloque de escritura, llamado *VISA Write*, figura 65, cuenta con tres entradas las dos entradas comunes al resto de bloques, salientes del bloque anteriormente descrito, y una tercera en la cual se cablea la cadena de caracteres que corresponde con el comando que se va a enviar al dispositivo. Además, cuenta con las dos salidas comunes al resto de bloques de comunicación.

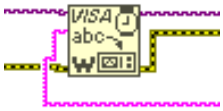

Figura 65 VISA Write. Fuente: elaboración propia

Previamente al bloque de lectura es necesario instalar un bloque llamado *VISA Bytes at Serial Port* que se encargue de configurar en el bloque de lectura el número de bytes que están configurados en el puerto serie.

El bloque de lectura, llamado *VISA Read*, figura 66, cuenta con tres entradas: las dos comunes y la correspondiente al número de bytes del puerto serie. Y cuenta con tres salidas, las dos comunes y una tercera que es una cadena de caracteres y se corresponde con la lectura del buffer del dispositivo Arduino.

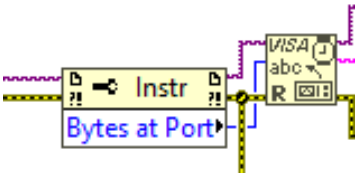

Figura 66 VISA Read. Fuente: elaboración propia

Por último, se coloca el bloque VISA Close, figura 67, que sirve para deshabilitar la comunicación con el puerto serie.

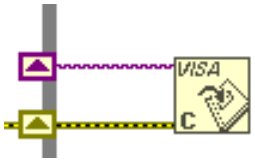

Figura 67 VISA Close. Fuente: elaboración propia

Programación de control y visualización:

Para tratar los cuatro pulsadores y los diferentes modos de actuación en función de cuál de ellos esté activado, se ha diseñado un sistema, figura 68, mediante el cual se asigna una posición a cada pulsador dentro de un número de cuatro cifras, por ejemplo: el pulsador de paro corresponde con las unidades, de tal forma que en función de que pulsador esté activo aparecerá un 1 en esa posición, dando lugar a los siguientes valores: 0, 1, 10, 100, 1000.

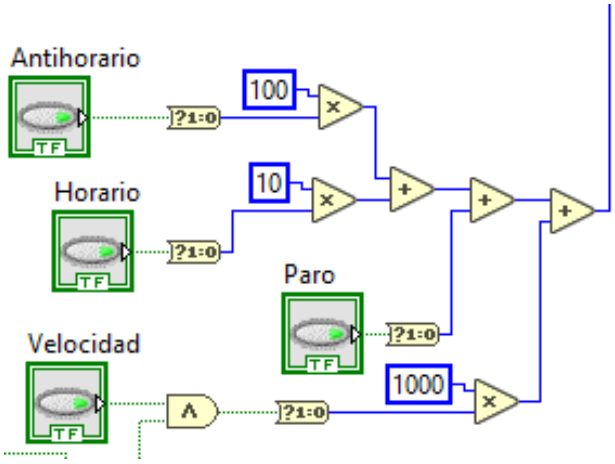

Figura 68 Control pulsadores. Fuente: elaboración propia

En función de estos 5 valores se activará una de las distintas opciones que contempladas en una estructura case:

- 0: este caso es también el caso predeterminado, es decir, si no es reconoce ninguno de los casos configurados, ejecuta este caso. En este caso el programa no realiza ninguna acción, pasando directamente a la lectura del buffer.

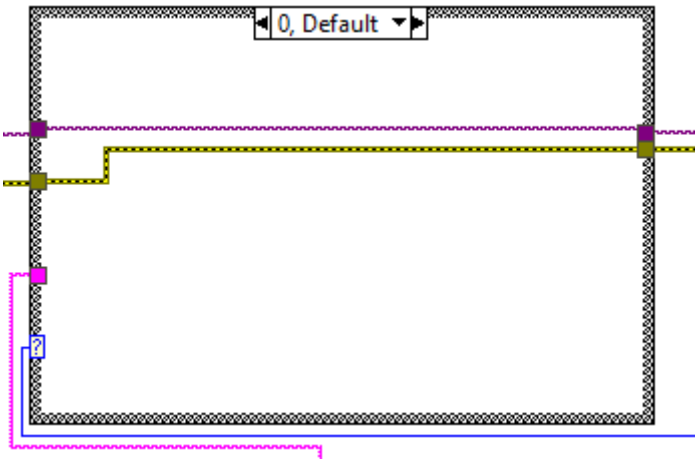

Figura 69 Caso 0. Fuente: elaboración propia

- 1: si llega el valor uno, significa que está pulsado el botón de paro. El programa enviará al dispositivo el comando “<c>”, que será reconocido como orden de parada.

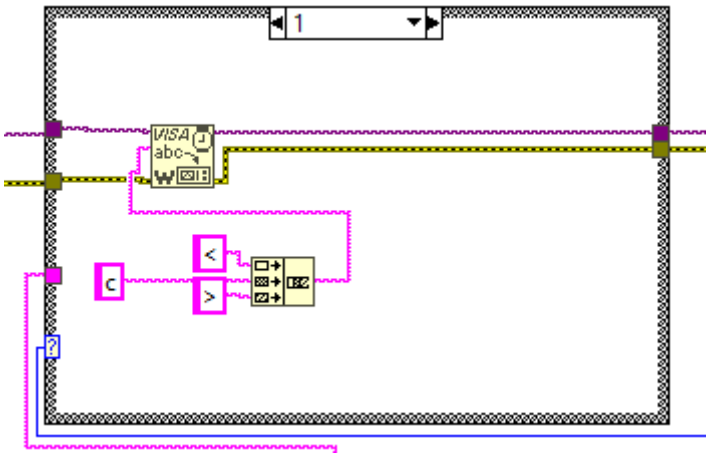

Figura 70 Caso 1. Fuente: elaboración propia

- 10: se activará si está pulsado el botón de giro antihorario. Se enviará el comando “<b>”, que corresponde con el giro antihorario.

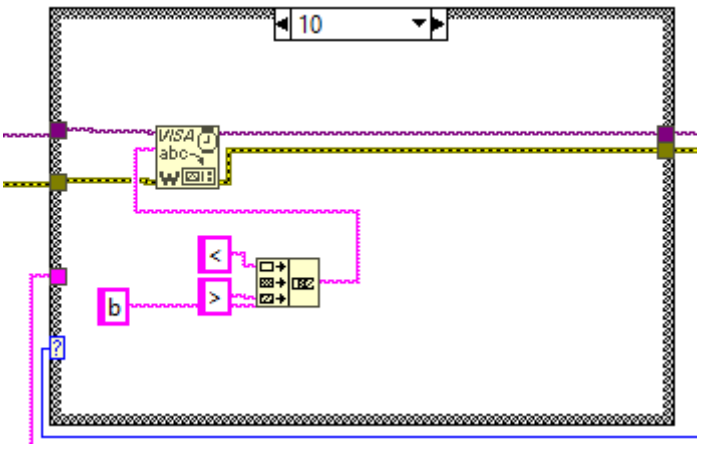

Figura 71 Caso 10. Fuente: elaboración propia

- 100: se activará si esta pulsado el botón de giro horario. En este caso se enviará el comando “<a>”, correspondiente con el giro horario.

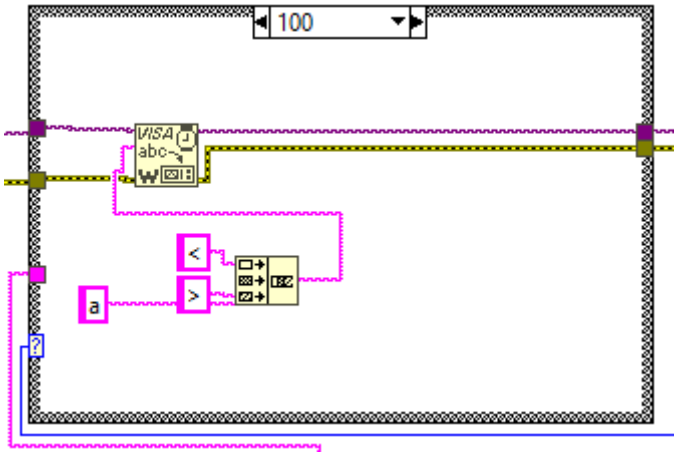

Figura 72 Caso 100. Fuente: elaboración propia

- 1000: en este caso se habrá activado el botón que envía el nuevo valor de velocidad y se enviará un comando del tipo <vXXX>. El dispositivo reconocerá “v” como orden de tratar los caracteres enviados a continuación y obtendrá el valor de la nueva velocidad.

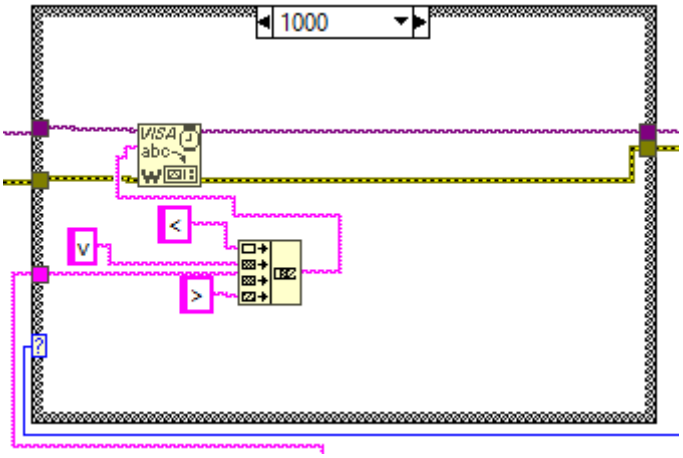

Figura 73 Caso 1000. Fuente: elaboración propia

Hay que destacar que el valor de la velocidad debe ser tratado previamente a ser enviado al dispositivo, ya que solo se pueden enviar cadenas de caracteres, por lo que el valor numérico se tendrá que convertir a cadena de caracteres. Además, el dispositivo lee este valor de derecha a izquierda, así que se invertirá la cadena obteniendo una nueva cadena simétrica. Dado que el dispositivo cambia la velocidad en función de la programación de la interrupción que depende del número de ciclos realizados por el procesador, para convertir los pasos partido segundo a un valor válido para la programación de la interrupción, se han realizado los siguientes cálculos:

Dado que el microprocesador tiene una velocidad de 16MHz y que hemos configurado el prescaler de la interrupción a un “tic” cada 1024 ciclos, obtendremos los siguientes “tics” por segundo:

$$\frac{16.000.000}{1.024} = 15625\ tics/s$$

Como cada paso corresponde a dos interrupciones, se tendrá que realizar una interrupción cada 7812,5≈7812 tics.

Por lo tanto, la ecuación que relaciona la velocidad (en pasos partido segundo) será:

$$tics = velocidad \cdot 7812$$

Siendo x el valor en pasos partido segundo.

Por lo tanto, si el dispositivo solo admite valores hasta 9999 tics el valor mínimo de velocidad será:

$$tics = velocidad \cdot 7812$$

$$\rightarrow$$

$$velocidad = \frac{9999}{7812} = 1{,}2799 \approx 1{,}27\ {}^{pasos}/_{s}$$

Si queremos fijar un valor máximo de velocidad, en este caso un paso cada 0,05 segundos, el valor mínimo de tics será:

$$tics = velocidad \cdot 7812$$

$$\rightarrow$$

$$tics = 0{,}05 \cdot 7812 = 390{,}6 \approx 391\ tics$$

Por último, como se ha limitado la velocidad máxima a un paso cada 0,05 segundos y la mínima a 1,27 pasos cada segundo, por lo que habrá que incluir un comparador, programar una ventana de error en caso de que el valor sea inferior y detener la ejecución del programa.

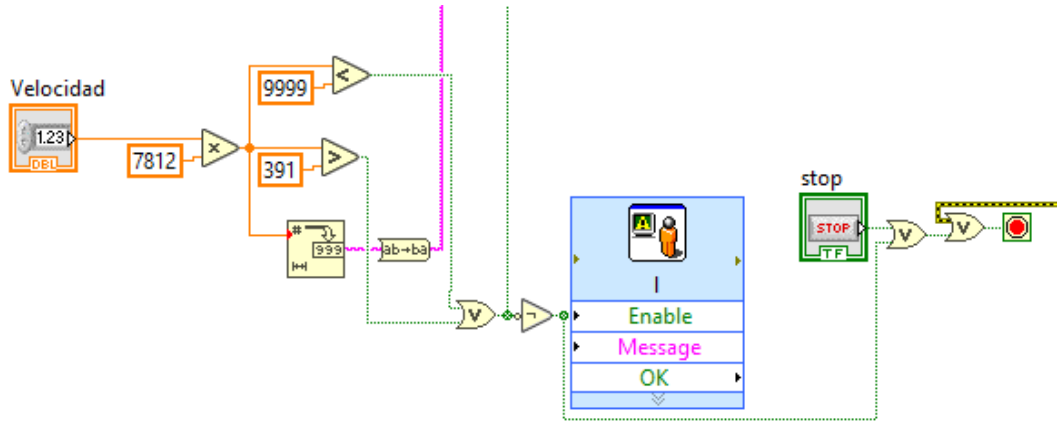

Figura 74 Control de velocidad. Fuente: elaboración propia

La cadena de caracteres que se obtiene de la lectura del buffer se cableará hacia un indicador que mostrará la lectura y hacia un indicador numérico que simulará el giro del motor.

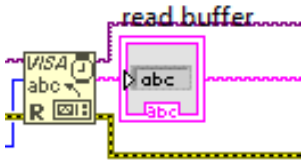

Figura 75 Indicador buffer. Fuente: elaboración propia

Puesto que el indicador numérico no acepta cadenas de caracteres se tendrá que convertir a valor numérico. Antes de introducir el valor de los pasos en el indicador, habrá que adaptar este valor para que una vez alcance el límite de pasos o el número de pasos sea negativo, siga girando. Para ello se dividirá el número de pasos entre el número de pasos por vuelta y se obtendrá el resto de esta operación. Después, se convertirá este valor a un valor numérico de doble precisión y se volverá a dividir entre el número de pasos por vuelta, obteniendo así un valor entre 0 y 1 de manera independiente al motor que utilicemos.

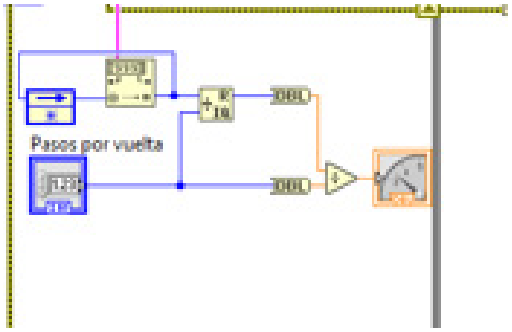

Figura 76. Indicador motor (bloque). Fuente: elaboración propia

Toda la estructura anterior, control e indicadores y comunicación serie, salvo el bloque VISA Configure Serial Port y el VISA Close, se introducen en una estructura while, para que el programa se esté ejecutando de forma continua. Esta ejecución se detendrá en caso de pulsar el botón de paro, introducir un valor incorrecto de velocidad o que exista un error en la comunicación con el puerto serie, figura 63.

4. Adaptación a otros sistemas (control de otros motores)

A. El dispositivo de control del motor paso a paso (Arduino Nano + placa de circuito impreso) están diseñados específicamente para el motor escogido por lo tanto se han tomado decisiones como unir la alimentación del motor y la

del Arduino, lo que implica que la tensión a la que funcione cualquier motor que se quiera conectar debe ser de 5V. No obstante, en la placa de circuito impreso existe un puente que une la alimentación del motor y la del Arduino, por lo que se podría retirar ese puente y colocar en su lugar una resistencia cuya caída de tensión hiciera que la alimentación del Arduino siguiera en 5V mientras que la del motor fuera superior.

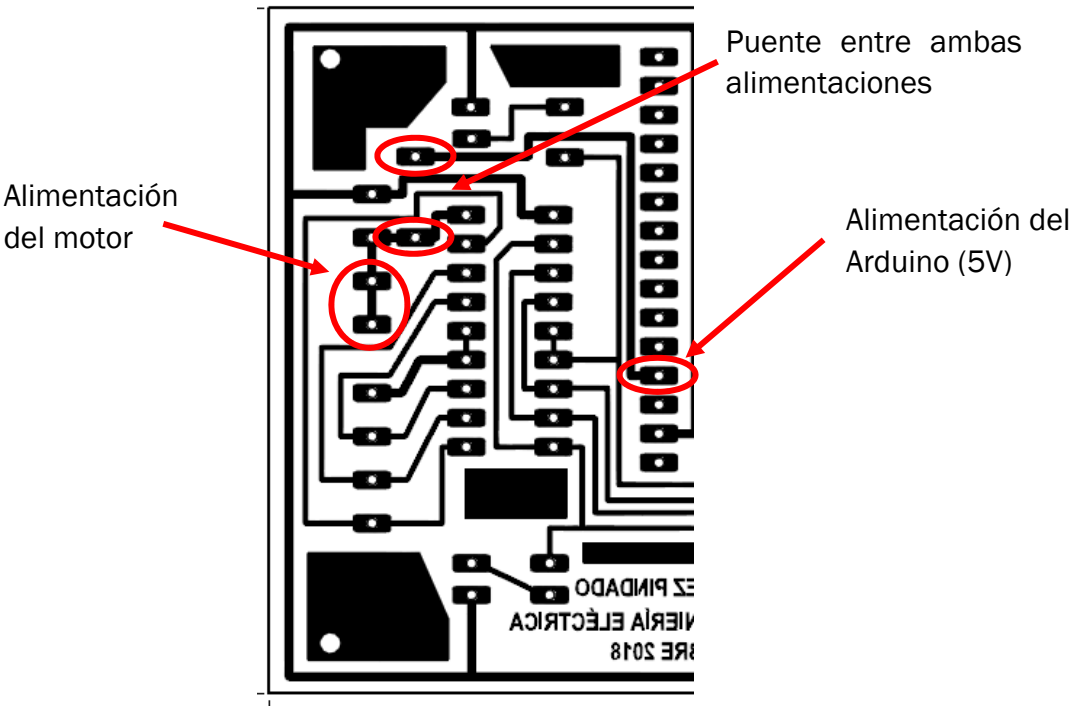

Figura 77 Conexión de alimentación. Fuente: elaboración propia

B. Además, se puede sustituir el motor por otro que tenga un número distinto de pasos por vuelta, ya que en el código de Arduino no intervienen el número de pasos y el programa de LabVIEW está preparado para modificar este valor.

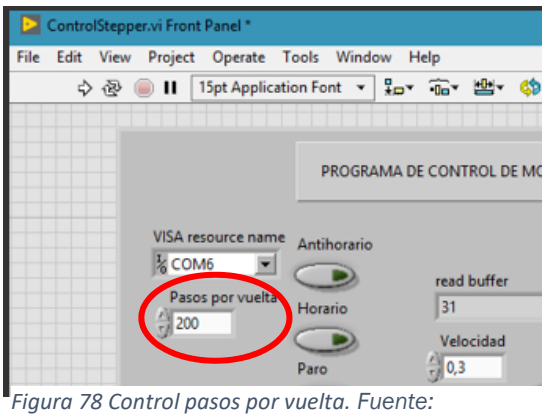

Figura 78 Control pasos por vuelta. Fuente: elaboración propia

C. Si se quisiera, se podría variar el código de Arduino para que el motor funcionase a distinta velocidad en el control físico.

```
int dir=0, cont;
int vel=7812;
char comandoTexto[10];
volatile boolean chivato=false;
int bobina=0;
boolean led = false;
```

Figura 79 Velocidad predeterminada código Arduino. Fuente: elaboración propia

D. Si se quisiera cambiar el método de comunicación, habría que recurrir a otro tipo de placas, ya que Arduino Nano solo comunica por puerto serie y no admite la conexión de módulos shield. Por lo tanto, habría que prescindir de la placa instalada y utilizar los zócalos en los que está montada para conectar el nuevo dispositivo.

Sin embargo, aparece el problema de que o bien otras placas no se adecúan al tamaño del zócalo, o si lo hacen, la posición de sus pines no se corresponde con la instalada. La única solución posible es realizar la conexión mediante cableado de forma que se conecten los pines de forma que correspondan. Por ejemplo, si se quisiera cambiar a conexión vía wifi:

Aparecen dos opciones:

1. Utilizar la placa Arduino MKR1000 y hacer la conexión mediante cableado ya que la posición de los pines no se corresponde.

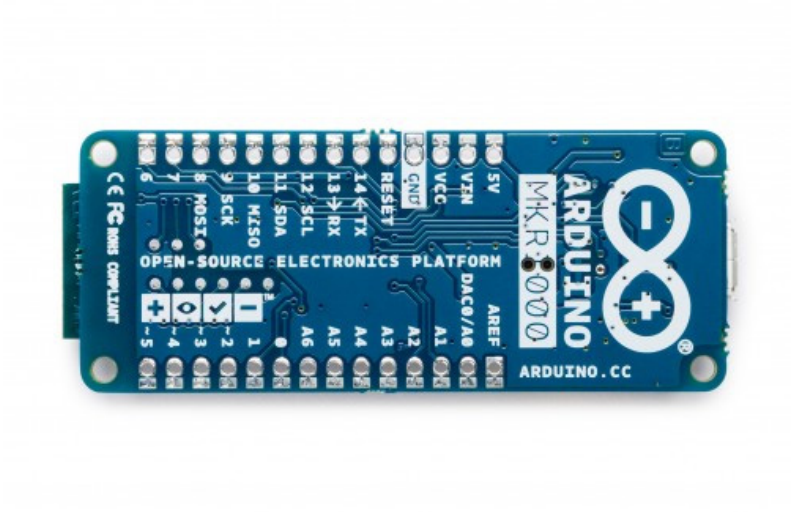

Figura 80 Arduino MKR1000, trasera (2018) Recuperado de https://store.arduino.cc/arduino-mkr1000

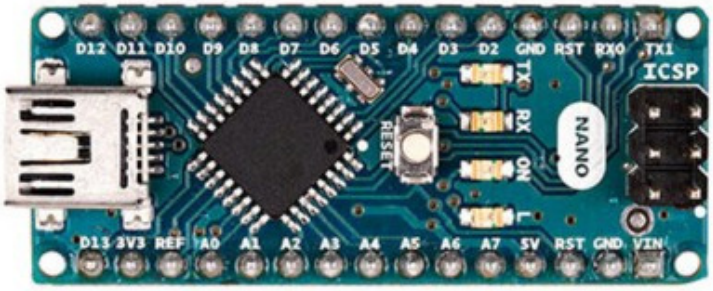

Figura 81 Arduino Nano, Frontal (2018) Recuperado de https://store.arduino.cc/arduino-nano

2. Utilizar Arduino UNO y añadirle el módulo Arduino WiFi 101 Shield y cablear de tal forma que se correspondan los pines con los del circuito impreso.

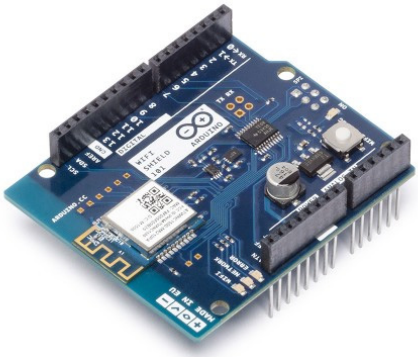

Figura 82 Arduino(2018) Recuperado de https://store.arduino.cc/arduino-wifi-101-shield

Figura 83 Arduino (2018) Recuperado de https://store.arduino.cc/arduino-uno-rev3

En ambos casos anteriores se debe cambiar el código que corresponde a la comunicación serie por el código de la librería de Arduino para wifi, sin modificar el protocolo de comunicaciones.

3. Utilizar el ESP8266 y unir sus pines con los del Arduino Nano de la siguiente manera:

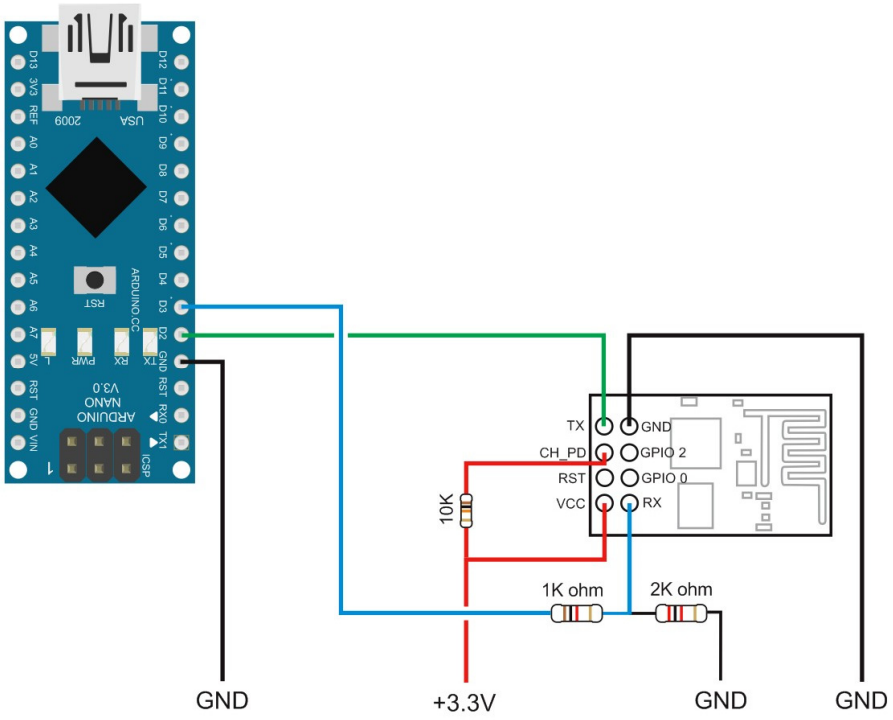

Figura 84 Connect an ESP8266 module to an Arduino Nano (2018) Recuperado de https://home.et.utwente.nl/slootenvanf/2018/03/17/connect-esp8266-control-blynk/

Una vez conectado, se configura el programa mediante el lenguaje específico que existe para funcionar con el módulo ESP8266 en el entorno de Arduino.

Si en vez de querer cambiar la comunicación a Ethernet o bluetooth habría que seguir un proceso similar al seguido con wifi, cambiando siempre la configuración correspondiente a las comunicaciones, pero manteniendo el protocolo de comunicaciones mediante comandos entre máquina y dispositivo

5. Presupuesto

Material	Unidades	Precio	Agregado
Placa Arduino Nano	1	25,29 €	25,29 €
Motor paso a paso SY42STH47-1206A (Nema 17)	1	21,66 €	21,66 €
Diodo led	4	0,30€	0,90 €
Pulsador	7	0,42 €	2,94 €

ULN2803A	1	2,50 €	2,50 €
Conector hembra Arduino 15 pines	2	0,75 €	1,50 €
Conector PCB 4 pines x2	1	8,99 €	8,99 €
Conector hembra para ULN2803A	1	0,50 €	0,50 €
Resistencia 1kΩ	4	0,25 €	1,00 €
Placa de cobre con resina fotosensible 60x80mm	1	8,99 €	8,99 €
Placa protoboard mini x6	1	7,99 €	7,99 €
Kit de cables para Arduino	1	7,59 €	7,59 €
Cable USB Mini-B	1	5,49 €	5,49 €
Caja envolvente de metacrilato	1	28 €	28 €
Hilo de estaño para soldar	1	3,99 €	3,99 €
Soldador con conexión USB 2.0A	1	14,99 €	14,99 €
Fabricación del dispositivo físico (horas)	5	35,00 €	175,00 €
Programación LabVIEW (horas)	30	35,00 €	1.050,00 €
Programación Arduino (horas)	30	35,00 €	1.050,00 €
Redacción de memoria y anexos	75	35,00 €	2.625,00 €
TOTAL			**5.042,32 €**

6. Conclusiones

Partiendo de los objetivos iniciales se ha creado un control que, si bien no alcanza todos ellos de una forma estricta, consigue realizar todas las funciones planteadas a pesar de que al pasar de alguna de ellas a otra surjan ciertos inconvenientes tales como:

- Al abrir el puerto serie la placa Arduino sufre un reseteo automático debido a las propias características del dispositivo.
- Al realizar el movimiento mediante la secuencia manual el dispositivo no es capaz de registrar el número de pasos dados.
- No se consigue compatibilizar el registro total de los pasos dados por el motor con la posibilidad de programar un número de pasos concretos durante el movimiento automático.

A pesar de estos inconvenientes, se ha conseguido crear una placa de control mediante la cual se consigue explicar y facilitar el entendí miento del funcionamiento de este tipo de motores.

7. Bibliografía

1. Paul Acarnley (2002). The Institution of Engineering and Technology. *Stepping Motors a guide to theory and practice 4th edition.*
2. Brian W. Evans (2007). *Arduino programming notebook*.
3. Atmel. *ATmega328/P datasheet*.
4. Arduino Nano V3 – ATmega328 5V + Cable USB Compatible (23 de agosoto de 2018). Recuperado de https://electronilab.co/tienda/arduino-nano-v3-atmega328-5v-cable-usb/
5. Programación Arduino - *Aprendiendo Arduino* (25 de agosto de 2018). Recuperado de https://aprendiendoarduino.wordpress.com/2017/01/23/programacion-arduino-5/
6. AVR Libc Home Page - Savannah.nongnu.org (25 de agosto de 2018). Recuperado de http://www.nongnu.org/avr-libc/
7. ATmega328 - *Microchip Technology* (24 de agosto de 2018). Recuperado de https://www.microchip.com/wwwproducts/en/ATmega328
8. Arduino Hobby (26 de agosto de 2018). *Interrupciones de timer con Arduino*. Recuperado de http://www.arduinohobby.com/interrupciones-timer-arduino/
9. Universidad de Cantabria- Servicio de informática (28 de agosto de 2018). *LabVIEW.* Recuperado de https://sdei.unican.es/paginas/servicios/software/labview.aspx
10. National Instruments (28 de agosto de 2018). *Software de Desarrollo de Sistemas NI LabVIEW*. Recuperado de http://www.ni.com/labview/products/esa/
11. Csic.es (28 de agosto de 2018). *LabVIEW*. Recuperado de http://sitios.csic.es/web/calculo-cientifico/labview
12. Carlitosspedia (15 de agosto de 2018). *Revelado y atacado de nuestros ciruitos impresos PCB*. Recuperado de https://www.carlitosspedia.info/revelado-y-atacado-de-nuestros-circuitos-impresos-pcb/
13. Staticboards (4 de septiembre de 2018). *Guía definitiva para dominar GBRL.* Recuperado de https://www.staticboards.es/blog/dominar-motor-paso-a-paso-con-grbl/
14. Aprendiendo Arduino (4 de septiembre de 2018). *Comunicación Serie Arduino*. Recuperado de https://aprendiendoarduino.wordpress.com/2016/07/02/comunicacion-serie-arduino/
15. Aprendiendo Arduino (4 de septiembre de 2018). *Comunicaciones Arduino*. Recuperado de

https://aprendiendoarduino.wordpress.com/2016/12/18/comunicaciones-arduino/

16. Arduino (5 de septiembre de 2018). *Stepper*. Recuperado de https://www.arduino.cc/en/Reference/Stepper
17. GRBL, S. (5 de septiembre2018). *Shield CNC A4988 - GRBL*. Recuperado de https://naylampmechatronics.com/arduino-shields/68-shield-cnc-a4988-grbl-.html

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Diseño e implementación del control de motor paso a paso mediante dispositivos embebidos RIO

(ANEXOS)

Javier Gómez Pindado

Grado en Ingeniería Eléctrica

Tutor: Moisés San Martín Ojeda

Código:01003

Septiembre 2018

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Anexo 1- Manual de uso

Anexo 1. Manual de uso