Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Escuela de ingeniería y arquitectura
Universidad de Zaragoza

Proyecto de final de carrera:

Entrenador de motores paso a paso

Autor:

Álvaro Barrios Rubia

Director:

Antonio Pardina

Titulación:

Ingeniería técnica industrial electrónica

Fecha de deposito:

20 de Mayo de 2015

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

PROPUESTA y ACEPTACIÓN DEL PROYECTO FIN DE CARRERA DE INGENIERÍA TÉCNICA

DATOS PERSONALES

APELLIDOS, Nombre
BARRIOS RUBIA, ALVARO
Nº DNI 17766681-D Dirección C/ CARMELO BETORÉ 2 CASA 6 5ºB
C.P. 50014 Localidad ZARAGOZA
Provincia ZARAGOZA Teléfono 615990338 NIA: 573831

Firma:
Alvaro Barrios

DATOS DEL PROYECTO FIN DE CARRERA

INGENIERIA TECNICA INDUSTRIAL, Especialidad ELECTRÓNICA
TITULO ENTRENADOR DE MOTORES PASO A PASO

DEPÓSITO EN: ZAGUAN (Obligatorio) ☒ y CD-ROM ☐ (si PFC es tipo B aplicación informática)

DIRECTOR ANTONIO PARDINA

VERIFICACIÓN EN SECRETARÍA

El alumno reúne los requisitos académicos **(1)** para la adjudicación de Proyecto Fin de Carrera

SELLO DEL CENTRO

EL FUNCIONARIO DE SECRETARIA

Fdo.: _____

SE ACEPTA LA PROPUESTA DEL PROYECTO **(2)**	SE ACEPTA EL DEPÓSITO DEL PROYECTO
En Zaragoza, a 26 de ENERO de 2.0__	En Zaragoza, a 5 de MAYO de 2.015
Fdo.: ANTONIO PARDINA CARRERA	Fdo.: ANTONIO PARDINA CARRERA
DIRECTOR DEL PFC	DIRECTOR DEL PFC

(1) Requisitos académicos: tener pendientes un máximo de 24 créditos o dos asignaturas para finalizar la titulación.

(2) Para que la propuesta sea aceptada por el Director, es imprescindible que este impreso esté sellado por la Secretaría de la EINA una vez comprobados los requisitos académicos.

Índice:

1.- Objetivo:

El objetivo de este proyecto es la realización de un entrenador de motores paso a paso con fines didácticos.

El trabajo realizado en este proyecto parte de la necesidad de construir un sistema capaz de medir y mostrar diferentes variables relacionadas con el funcionamiento de un motor paso a paso en distintas situaciones de trabajo para la correcta comprensión de su funcionamiento y manejo, así como la comparación de los resultados experimentales con los correspondientes teóricos para poder conocer las condiciones reales de funcionamiento y cuando dichas condiciones dejan de ser óptimas y el porqué de este funcionamiento defectuoso.

Las variables que se desean tanto controlar como visualizar son la velocidad de giro del motor, la posición de parada del motor, el tipo de paso con el que gira el motor y la forma de onda de corriente por las bobinas del motor. Para la introducción de los valores deseados se utilizará un teclado numérico, la visualización de la velocidad, la posición y el tipo de paso se realizará en un LCD integrado en el propio equipo.

El equipo deberá tener unas dimensiones lo más reducidas posibles para que pueda ser utilizado de modo portátil, e ira en el interior de una caja metálica o plástica diseñada con tal propósito o que ya exista en el mercado.

Este proyecto se ha realizado en dos etapas diferenciadas que se detallaran en profundidad más adelante:

1- En la primera fase se construyó un prototipo ya diseñado por el profesor responsable del proyecto Antonio Pardina para comprender las funciones básicas que debe realizar el equipo y que más tarde se debía realizar en la segunda fase de forma ampliada.

2- En la segunda fase se ha realizado una ampliación del prototipo anteriormente mencionado basando el funcionamiento del mismo en un Arduino DUE, aunque ha requerido de algunos componentes externos tales como un driver controlador de motores paso a paso, un integrado para contaje rápido y un encoder.

2.- Fases en el desarrollo del proyecto:

La realización de cualquier proyecto cuyo funcionamiento de base en microcontroladores se puede descomponer en varias fases, aunque no hay una separación nítida entre ellas.

En el desarrollo de cada fase hay que utilizar un conjunto de herramientas de hardware y software específicas.

En este proyecto, el proceso seguido ha sido el siguiente:

1- Definición clara del proyecto que se desea realizar, con sus características y prestaciones. En esta fase, se realiza la determinación de las necesidades de hardware, así como las partes que integrarán el programa.

2- Diseño del circuito físico del proyecto. Es decir, diseño del circuito físico que va a controlar el programa ejecutado en el microcontrolador.

3- Construcción del prototipo. En esta fase se adquieren todos los componentes que constituyen el proyecto y se realiza el circuito físico diseñado en la etapa anterior montándolo en una placa de pruebas. En esta fase, se debe de verificar el comportamiento del circuito en todas aquellas partes que sea posible.

4- Planteamiento del programa necesario. En primer lugar se plantea el diagrama de flujo general depurándolo hasta que sea óptimo. Si el proceso es complejo, este diagrama se debe descomponer en varios sub-diagramas para hacer más comprensible el funcionamiento.

5- Edición del programa de control del sistema utilizando para ello el lenguaje propio de Arduino (muy similar al C) en el editor exclusivo para Arduino llamado "Arduino IDE", que es un "Entorno de Desarrollo Integrado" (Integrated Development Environment) que se ejecuta bajo Windows y que incluye todas las utilidades necesarias para la realización de proyectos con cualquier tipo de placa Arduino, ya que permite editar el código del proyecto, además de compilarlo y comprobar que dicho código se ha escrito de forma correcta. Este software también permite la transferencia del código al microcontrolador del Arduino, aunque es un poco pobre en cuanto de herramientas de debug, ya que no se puede seguir la evolución de las variables para localizar un posible fallo en el código.

6- Grabación del programa en el microcontrolador una vez comprobado que la sintaxis utilizada es la correcta. Para ello se utiliza el "Arduino IDE". El fichero a grabar en la memoria del microcontrolador tiene la extensión ".ino", y esta grabación se realiza directamente desde el "Arduino IDE" tras la compilación del programa.

7- Comprobación del funcionamiento del prototipo con el programa ya transferido a la placa Arduino. Si no cumple con el funcionamiento previsto hay que revisar el código escrito para localizar el error y corregirlo. Para esto puede ser de gran ayuda utilizar algún programa de debug como el "Atmel Studio".

8- Depuración del programa y del circuito repitiendo el proceso hasta alcanzar los objetivos perseguidos.

9- Diseño de la placa de circuito impreso mediante el software "Protel DXP" para la realización del prototipo final.

10- Montaje del prototipo final en la placa de circuito impreso.

11- Verificación del comportamiento del circuito.

12- Documentación del proyecto.

3.- Antecedentes:

3.1.- Introducción:

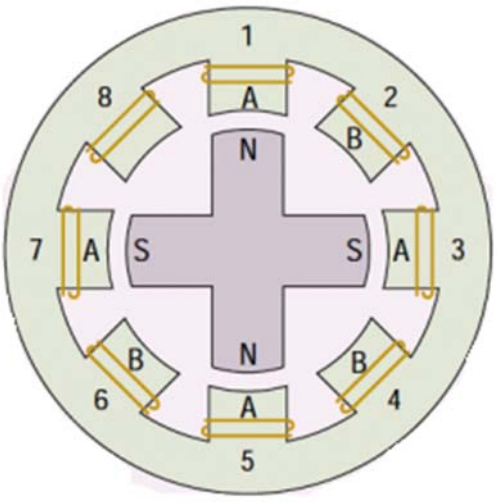
Un motor paso a paso es un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares, lo que hace que su movimiento sea a impulsos entre posiciones de equilibrio estable. Esto se logra modificando el sentido de circulación de la corriente por sus devanados.

Este tipo de motores se diferencia de los motores convencionales en que un motor paso a paso puede lograr una gran precisión y repetitividad al posicionarse, y sus principales aplicaciones son como motor de frecuencia variable, motor de corriente continua sin escobillas y como servomotor o motor controlados digitalmente, y también se unan como elementos motrices de máquinas tales como pequeñas fresadores o impresoras 3D.

3.2.- Evolución histórica:

El primer motor paso a paso fue desarrollado por la armada británica en 1933 y fue de tipo bidireccional y de reluctancia variable. Este motor fue usado como repetidor remoto de una brújula y punto de mira de un arma. Aunque este primer motor era bastante rudimentario, cumplía a la perfección con su cometido siempre y cuando se usase a velocidades constantes y lentas. Este sistema fue posteriormente adoptado por la armada de los Estados Unidos durante la segunda guerra mundial.

Durante la década de los 50, este tipo de motores comenzaron a usarse en otras aplicaciones, aunque de forma muy limitada debido a lo nuevo de esta tecnología y por lo tanto a lo poco evolucionada que se encontraba. A pesar de esto, los motores paso a paso de imanes permanentes poco a poco comenzaron a sustituir a los servomotores de lazo cerrado y a sus complejos sistemas digitales de control.



Motor paso a paso síncrono de principio de los 60

No fue hasta 1960 cuando apareció el motor paso a paso de imanes permanentes y comenzó a usarse de un modo mucho más extendido y se convirtió en el motor paso a paso más utilizado de la época. El hecho de que este tipo de motor se convirtiese en algo bastante popular no lo libraba de ciertos problemas como vibraciones en el eje o la resonancia que solo podían solucionarse deteniendo el motor e iniciando de nuevo su movimiento. Pero aunque hubiese algunos problemas, las ventajas hacían de estos motores unas máquinas únicas en lo que a posicionamiento se refería, ya que lograban una precisión del 5% en lazo abierto y sin acumulación del error.

Los sistemas de servomotor de escobillas (que trabaja en lazo cerrado) o con pequeños motores de corriente alterna de dos fases requerían una cuidadosa estabilización de la señal de retorno, tanto de posición como de velocidad, lo que hacía de esta tarea algo muy complicado en la década de los 60.

Las aplicaciones típicas de los motores paso a paso de la época eran el control de aviones no tripulados, indicadores de pesaje portátiles y altímetros diferenciales digitales.

En estos últimos 40 años la estructura básica de los motores paso a paso no ha variado, pero si lo ha hecho su precisión y métodos de control, de forma que pueden conseguirse posicionamientos muy precisos, lo que hace que este tipo de motores se usen en múltiples aplicaciones.

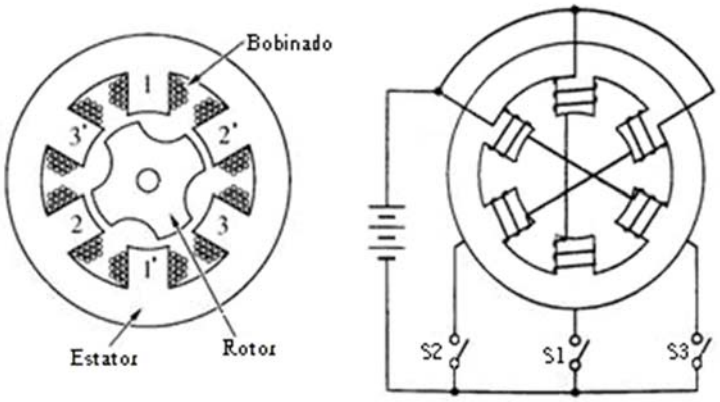
3.3.- Tipos de motor y funcionamiento:

Existen tres tipos de motores paso a paso, de imanes permanentes (unipolares y bipolares), de reluctancia variable (unipolares y bipolares) e híbridos.

- Los motores de imanes permanentes están constituidos por un rotor sobre el que se colocan varios imanes permanentes y por unas bobinas excitadoras bobinadas en su estator. Así, las bobinas forman parte del estator y el rotor es un imán permanente. Este tipo de motores tiene mayor par que el de reluctancia variable, pero no es capaz de alcanzar tanta velocidad.

- Los motores de reluctancia variable está constituido por un rotor de láminas ferromagnéticas no imantadas (de hierro dulce), formando un cilindro alrededor del eje. Éstas se encuentran ranuradas de forma longitudinal, formando dientes (polos del rotor).

La ranuración del rotor conlleva una variación de la reluctancia en función de su posición angular. Al igual que el rotor, el estator está formado por láminas de material ferromagnético no imantado, con una serie de ranuras longitudinales que albergan los bobinados de las fases y forman los polos del estator. El número de dientes del rotor es menor que el número de dientes del estator, de modo que sólo un par de polos del estator y su correspondiente par de polos del rotor pueden estar alineados por fase.

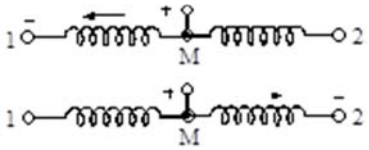


Sección de un motor paso a paso de reluctancia variable de tres fases

- Los motores híbridos son una combinación de los otros dos tipos de motores. Este motor consiste en un estator dentado y un rotor de tres partes (apilado simple). El rotor consta de dos piezas de polos separados por un imán permanente con los dientes opuestos desplazados en medio salto de diente para permitir una alta resolución de pasos.

- Motores unipolares:

  Los devanados unipolares están formados por dos bobinas en serie por polo, y se alimentan entre la toma intermedia (conexión serie de los dos devanados) y cada uno de los extremos.



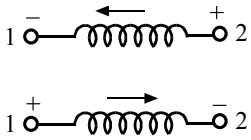
Devanados motor unipolar

  Al exterior salen los dos extremos y la toma intermedia.

  La corriente por las bobinas siempre circula en el mismo sentido, por lo tanto el campo magnético en un polo excitado siempre tendrá la misma polaridad. En la siguiente imagen se muestra una representación simplificada del funcionamiento de un motor paso a paso unipolar de alimentación doble (dos bobinas alimentadas al mismo tiempo).

- Motores bipolares:

  Este tipo de motor solo tiene accesibles los dos extremos de la bobina y mediante el cambio de polaridad se cambia el sentido de la corriente.



Devanados motor bipolar

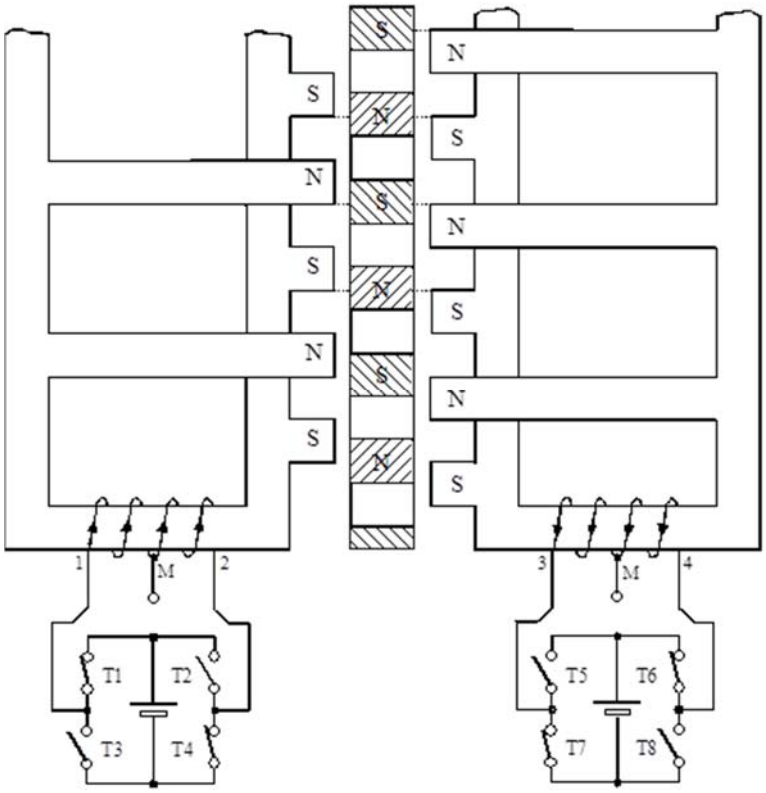
  En este caso, la corriente no siempre tiene el mismo sentido, por lo que el campo magnético generado en un polo cambia de polaridad en función del sentido de la corriente.

  Tienen un solo devanado por polo y por lo tanto el hilo es más grueso que en los unipolares. Por esa razón soportan mayor corriente y por consiguiente su par es mayor, este hecho hace también que este tipo de motor sea más barato que los unipolares, ya que los unipolares tienen más devanados constituidos con hilo más fino.

- Motor paso a paso real:

  En un motor paso a paso real se aumenta el número de polos para conseguir un ángulo de movimiento por paso mucho más pequeño (normalmente un paso completo equivale a 1.8º), pero en cambio el número de devanados es el mismo. En la siguiente figura se representa la construcción real de un motor paso a paso.

  En la siguiente imagen se muestra una representación simplificada del funcionamiento de un motor paso a paso bipolar de alimentación doble (dos bobinas alimentadas al mismo tiempo).



Representación de un motor paso a paso real

3.4.- Control de motores paso a paso:

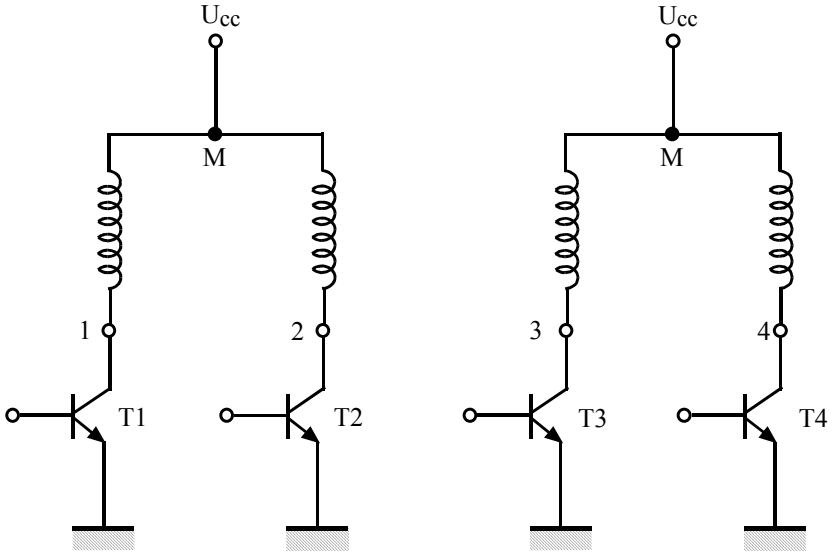
El funcionamiento básico de un motor paso a paso consiste en ir cambiando la polaridad de unos electroimanes mediante la apertura o cierre de unos interruptores (se usan transistores) para hacer girar el rotor del motor y lograr un posicionamiento lo más preciso posible.

Los motores paso a paso requieren de un driver de control que se encargue de ir realizando los disparos de los transistores de uno (motores unipolares) o dos (motores bipolares) puentes en H en el orden correcto según el tipo de motor, paso y frecuencia deseados.

La secuencia de disparo (entrada en conducción) de los transistores dependerá de si el motor es unipolar o bipolar y del tipo de paso que se desea dar. El tamaño mínimo del paso depende del driver utilizado, siendo un valor típico 1/8 o 1/16 de paso, pudiendo alcanzar resoluciones de 0.1125º por paso.

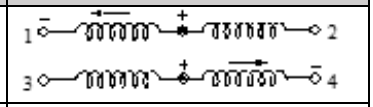
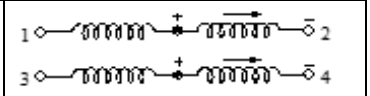
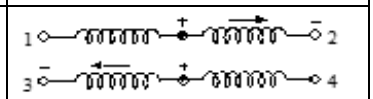
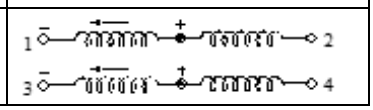
3.4.1.- Motor unipolar:

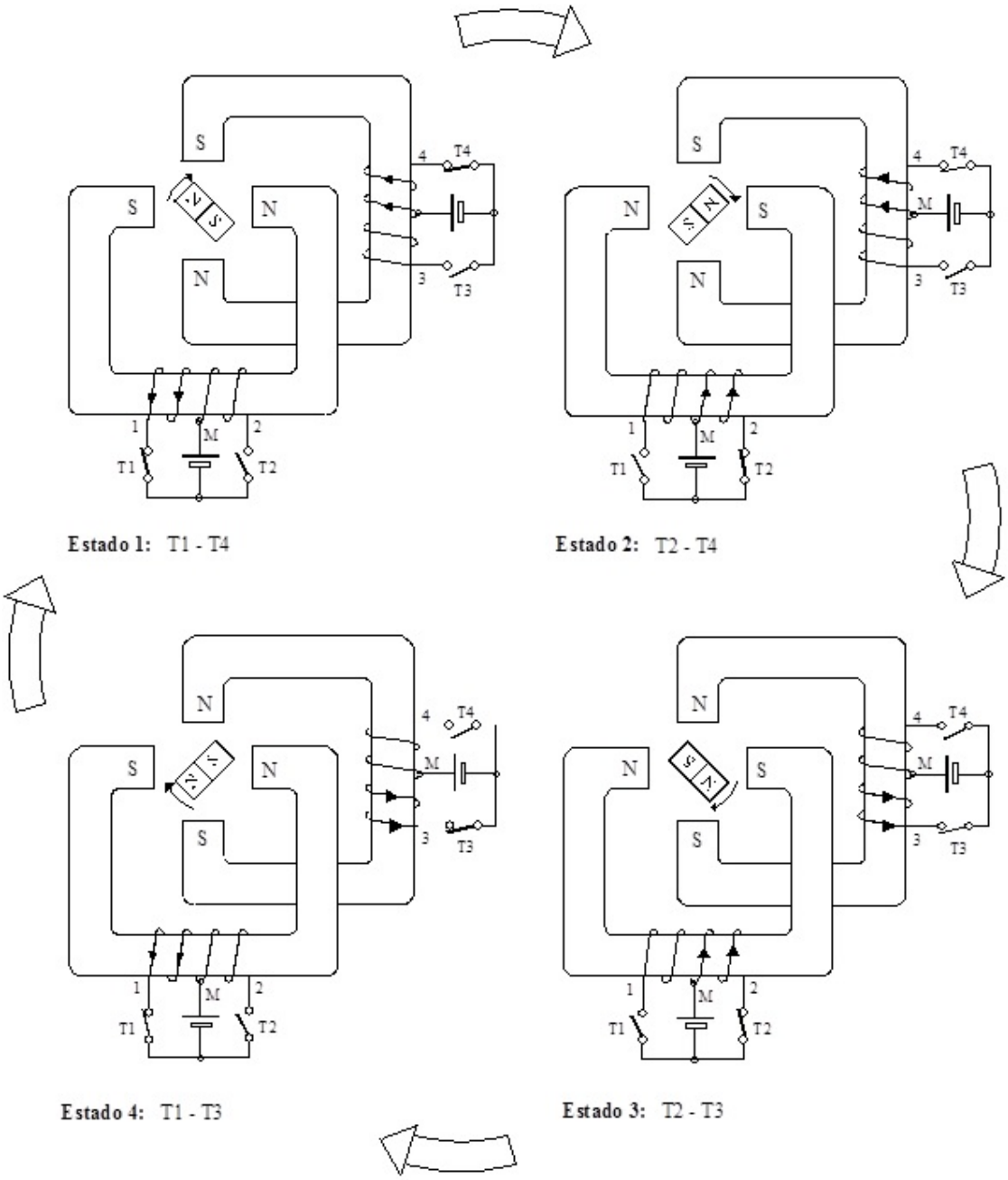
En esta imagen puede verse el circuito de accionamiento de un motor paso a paso con alimentación unipolar y dos bobinas excitadas simultáneamente:



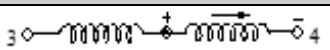
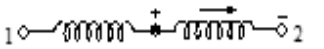
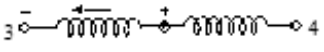
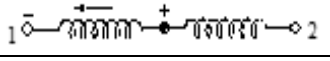
A continuación se muestra la tabla en la que se representa la secuencia de disparo correspondiente a un paso completo y a medio paso para un motor unipolar, así como una representación gráfica de la posición del motor:

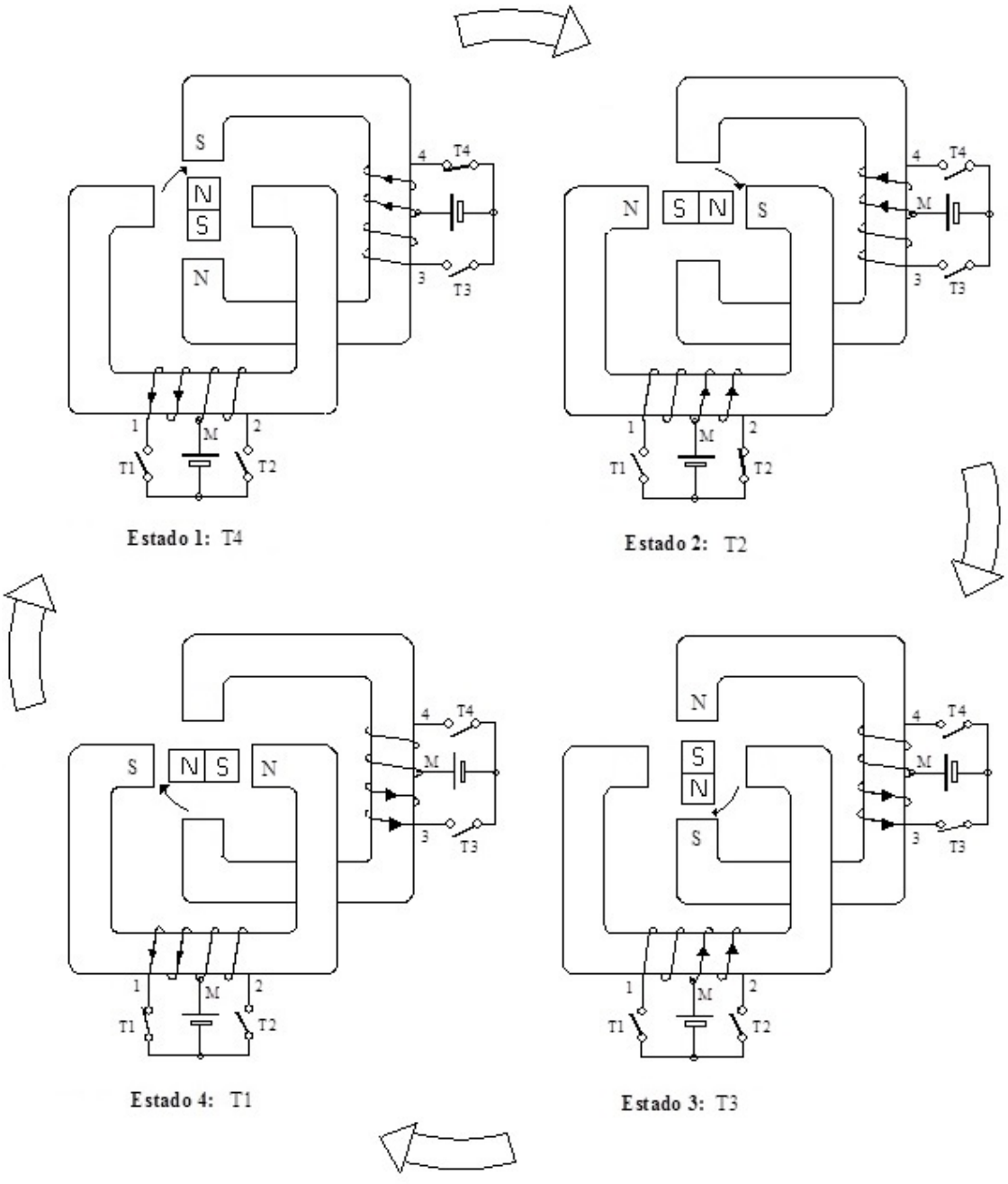
Para el paso completo existen dos opciones:

Opción 1					
Estado	**T1**	**T2**	**T3**	**T4**	**Sentido de las corrientes**
1	1	0	0	1	
2	0	1	0	1	
3	0	1	1	0	
4	1	0	1	0	



Representación gráfica de la opción 1 de paso completo

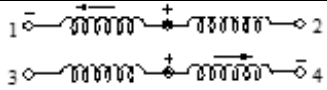
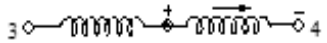
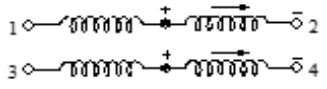
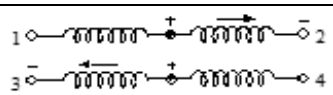
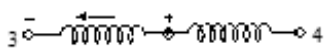
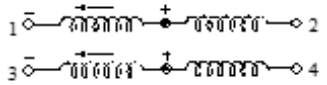
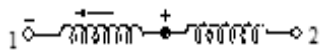
Opción 2					
Estado	**T1**	**T2**	**T3**	**T4**	**Sentido de las corrientes**
1	0	0	0	1	
2	0	1	0	0	
3	0	0	1	0	
4	1	0	0	0	

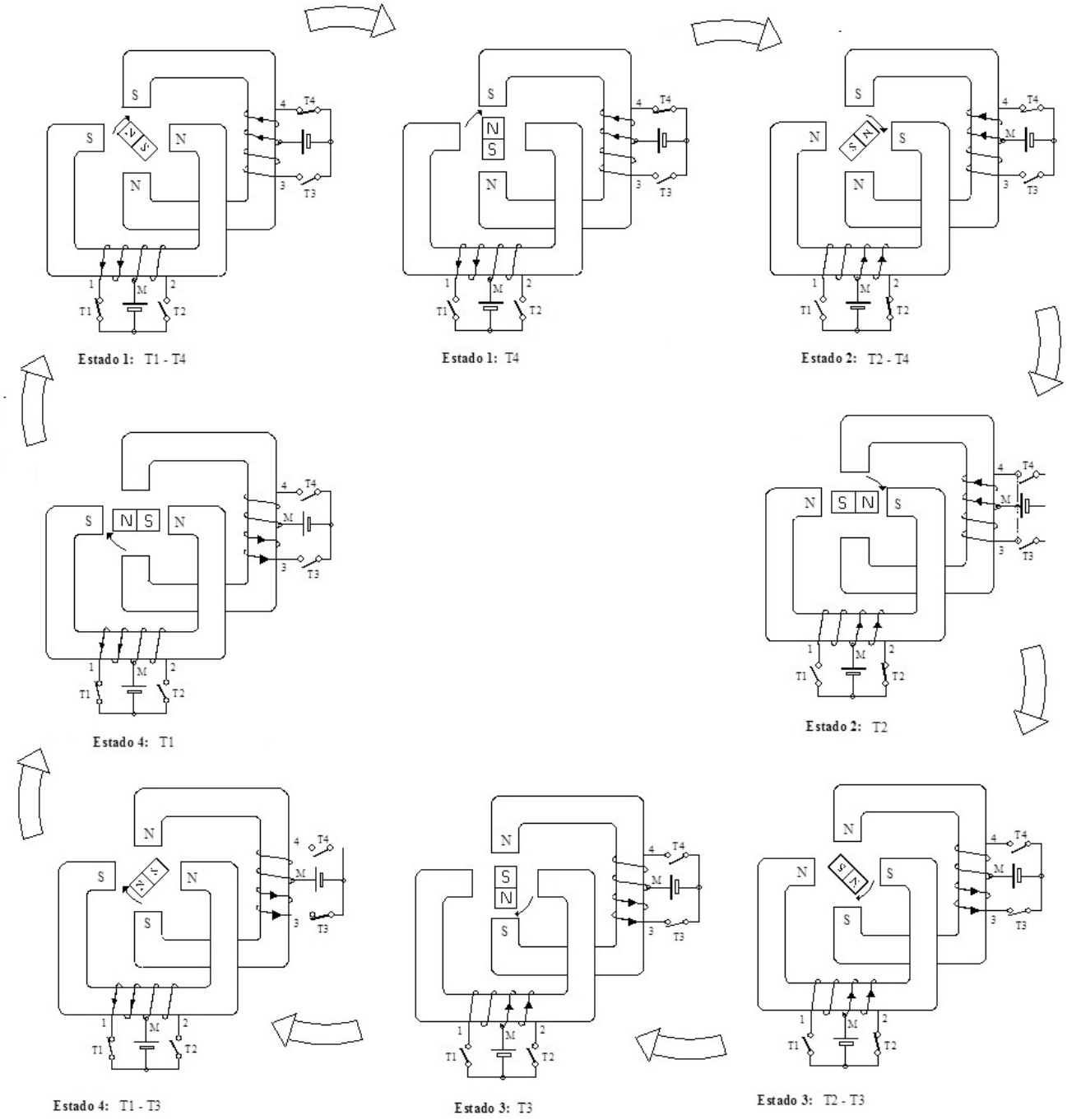


Representación gráfica de la opción 2 de paso completo

Aunque las dos opciones anteriores para el paso completo son válidas, es mejor la primera. Esto se debe a que como actúan a la vez dos imanes se consigue un mayor par que en el caso de la segunda opción, haciendo que el motor tenga más fuerza y por lo tanto sea más fácil que mantenga la posición.

Para obtener el medio paso hay que realizar una combinación de las dos opciones existentes para el paso completo, obteniendo así la siguiente secuencia de disparo:

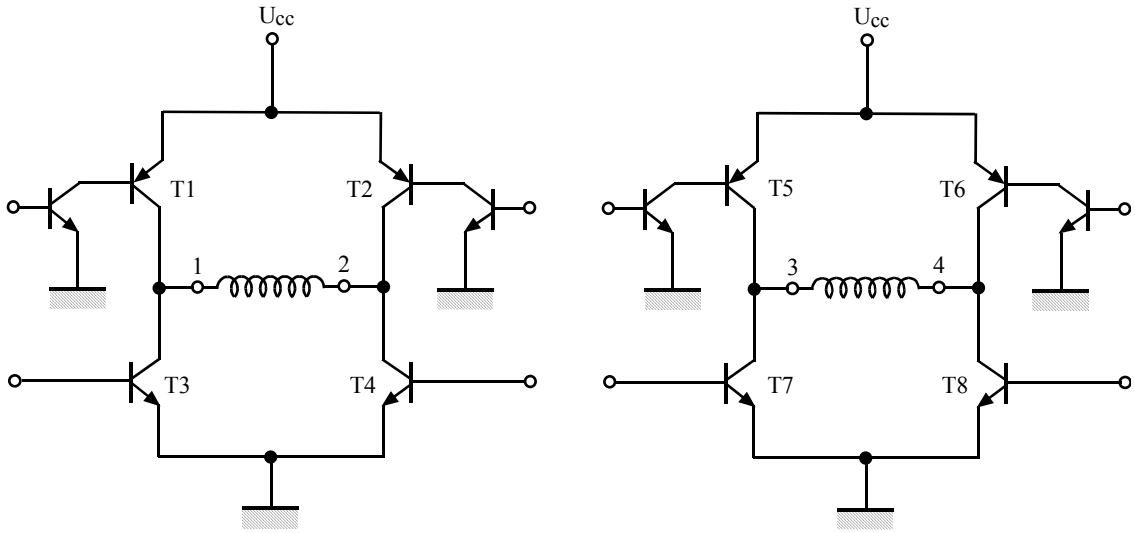
Estado	T1	T2	T3	T4	Sentido de las corrientes
1	1	0	0	1	
2	0	0	0	1	
3	0	1	0	1	
4	0	1	0	0	
5	0	1	1	0	
6	0	0	1	0	
7	1	0	1	0	
8	1	0	0	0	



Representación gráfica de medio paso para motor unipolar.

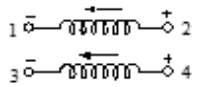
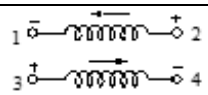
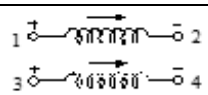
3.4.2.- Motor bipolar:

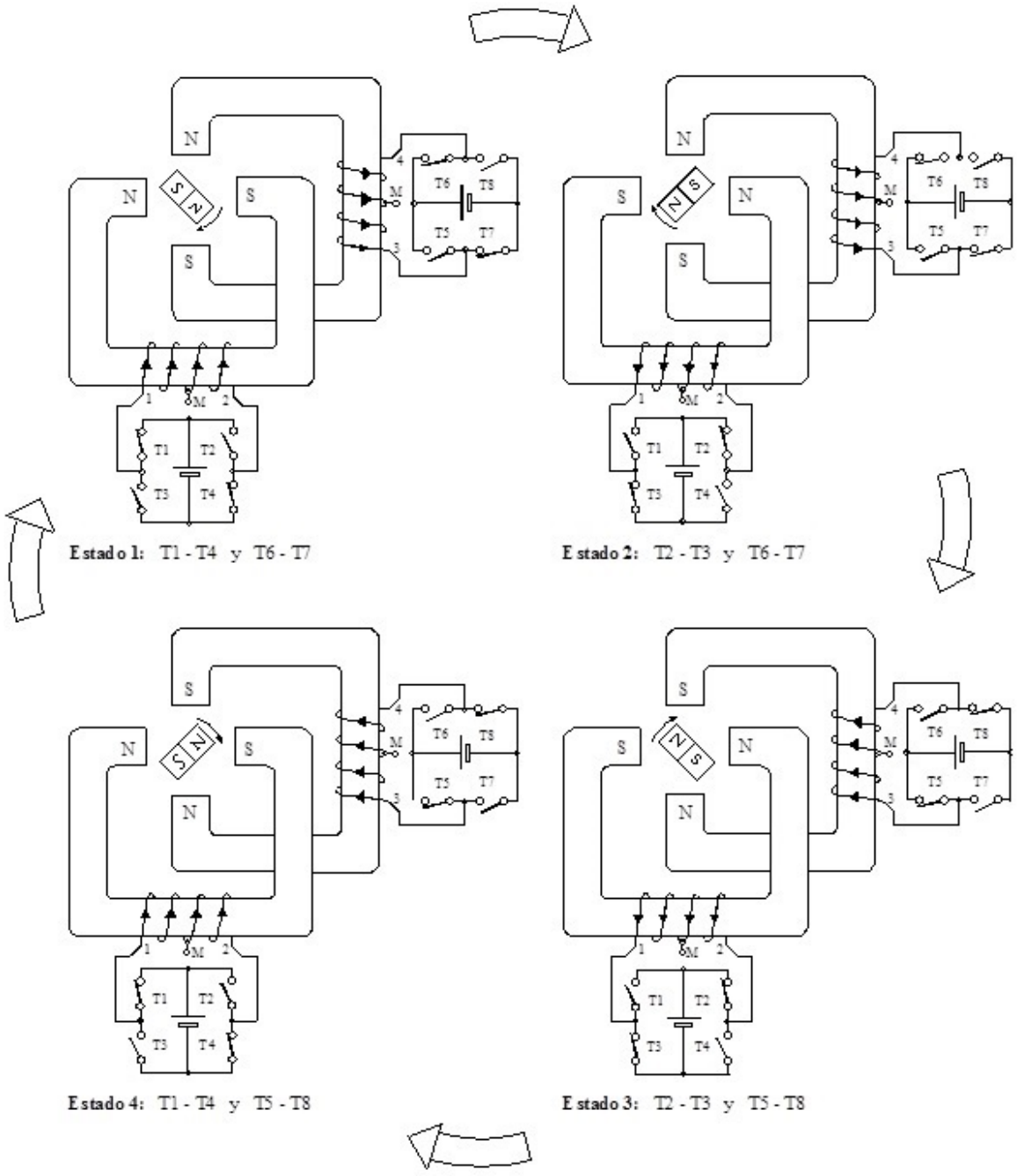
En esta imagen puede verse el circuito de accionamiento de un motor paso a paso con alimentación bipolar y dos bobinas excitadas simultáneamente:



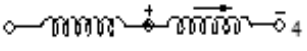
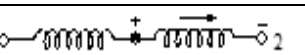
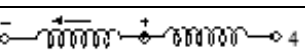
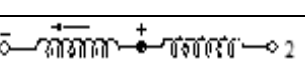
Al igual que en el caso anterior, para el motor bipolar también existen dos opciones a la hora de mover el motor con paso completo, y una para hacerlo con medio paso.

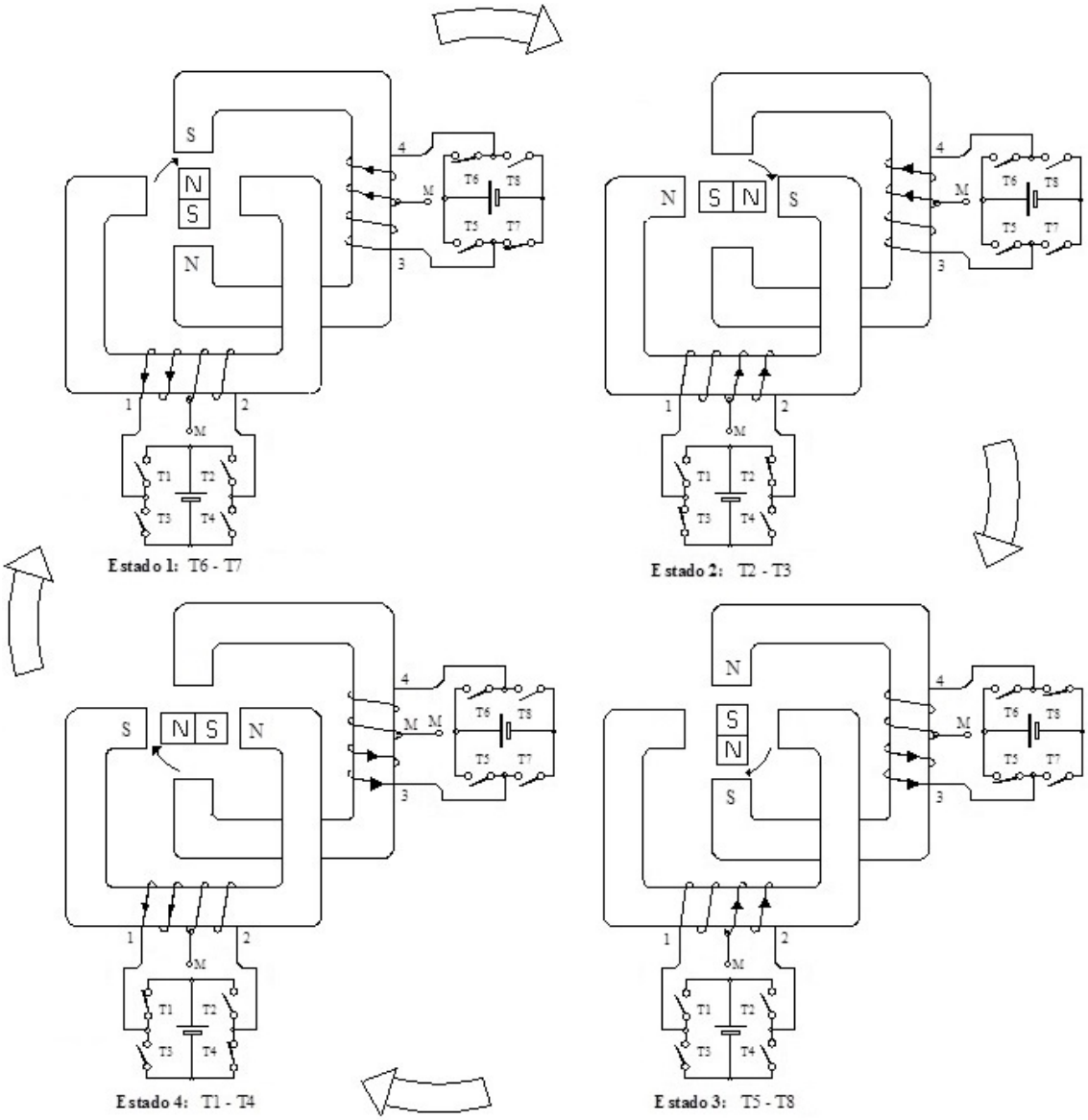
A continuación se muestran las dos secuencias de disparo correspondientes al paso completo:

Opción 1					
Estado	**T1-T4**	**T2-T3**	**T5-T8**	**T6-T7**	**Sentido de las corrientes**
1	1	0	0	1	
2	0	1	0	1	
3	0	1	1	0	
4	1	0	1	0	



Representación gráfica de la opción 1 de paso completo

Opción 2					
Estado	**T1-T4**	**T2-T3**	**T5-T8**	**T6-T7**	**Sentido de las corrientes**
1	0	0	0	1	
2	0	1	0	0	
3	0	0	1	0	
4	1	0	0	0	

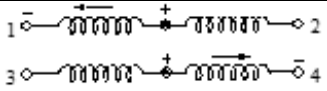
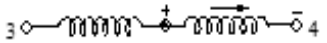
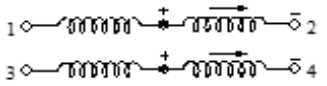
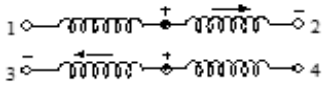
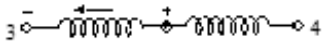
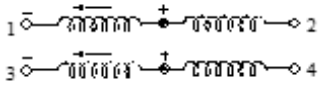
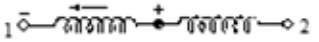


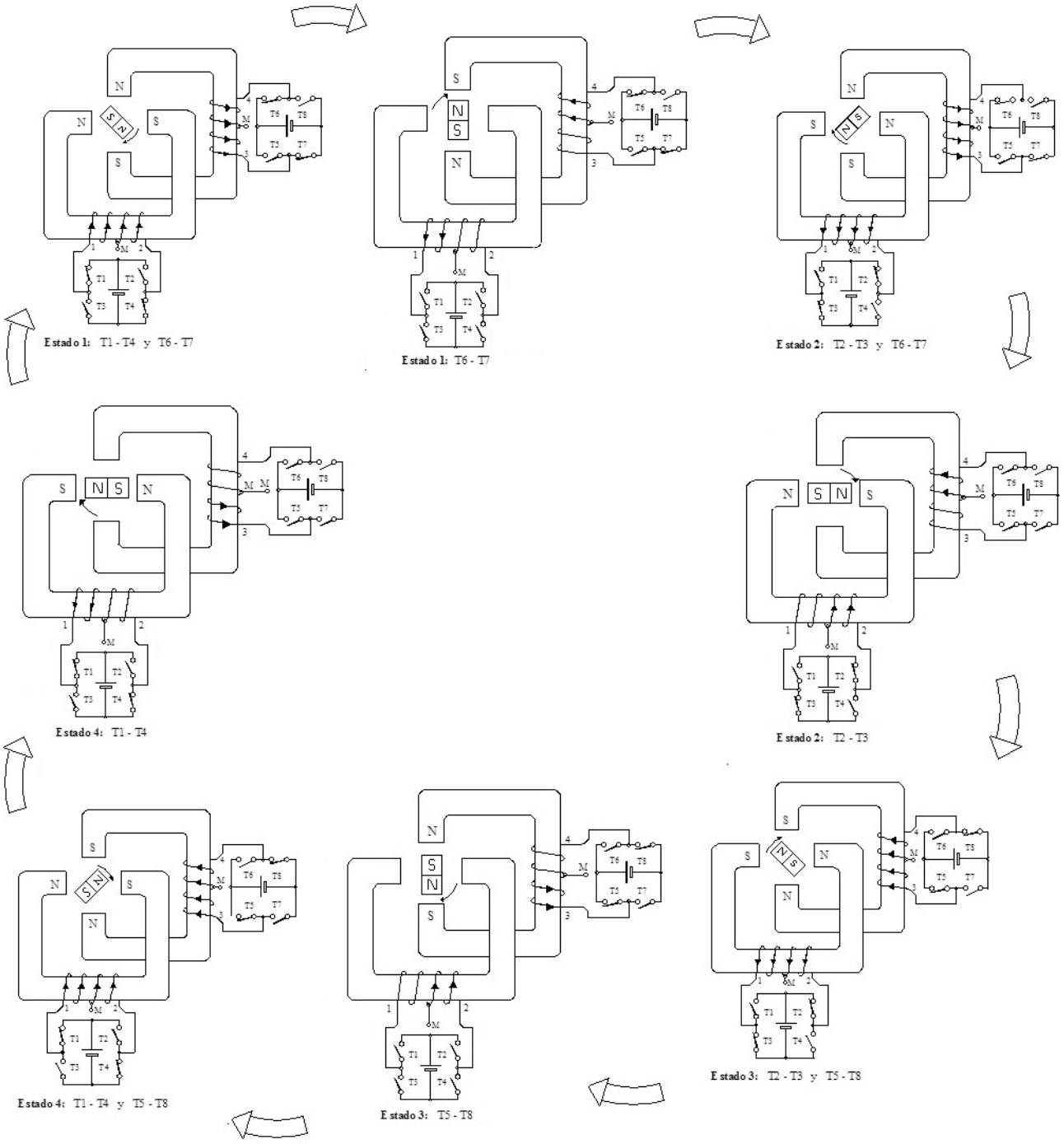
Representación gráfica de la opción 2 de paso completo

Al igual que en el caso anterior, las dos opciones son totalmente validas, pero es mucho mejor utilizar la primera opción, ya que actúan dos imanes al mismo tiempo y se logra mucho más par. Esta característica aunque no es imprescindible para este tipo de motores (ya que lo importante es la posición) sí que es importante para algunas aplicaciones que requieran de bastante fuerza por parte del motor.

Para este proyecto se ha utilizado un motor paso a paso bipolar, y se emplea la primera opción, ya que como se ha visto anteriormente, esta opción es bastante mejor que la segunda.

Al igual que en el caso del motor unipolar, para obtener el medio paso hay que realizar una combinación de las dos opciones existentes para el paso completo, obteniendo así la siguiente secuencia de disparo:

Estado	T1	T2	T3	T4	Sentido de las corrientes
1	1	0	0	1	
2	0	0	0	1	
3	0	1	0	1	
4	0	1	0	0	
5	0	1	1	0	
6	0	0	1	0	
7	1	0	1	0	
8	1	0	0	0	



Representación gráfica de medio paso para motor bipolar.

3.5.- Arduino:

3.5.1.- ¿Qué es arduino?:

Arduino es una plataforma de hardware libre, basada en una placa de circuito impreso con un microcontrolador (normalmente ATMEL) y un entorno de desarrollo. Todo diseñado con el objetivo de facilitar el uso de la electrónica en proyectos multidisciplinares.

El hardware consiste en una PCB con un microcontrolador Atmel AVR y varios puertos de entrada y salida. Los microcontroladores más usados son el Atmega168, Atmega328, Atmega1280, ATmega8 (dependiendo de la versión de arduino) por su sencillez y bajo coste, que permiten el desarrollo de múltiples diseños de forma mucho más sencilla que usando solo el microprocesador. Por otro lado el software utilizado para programar el arduino consiste en un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring y el cargador de arranque que es ejecutado en la placa.

Desde el 2012, arduino también incluye los microprocesadores CortexM3 de ARM de 32 bits, que aunque son mucho más potentes (32 bits) que los anteriormente mencionados, pueden programarse con el mismo software que las primeras versiones de arduino. A pesar de esto, estas placas funcionan con unos niveles lógicos inferiores, ya que si las primeras placas funcionaban a 5v, estas últimas lo hacen a 3.3v.

El último modelo (Arduino TRE) que será lanzado en el 2015, incluye dos microcontroladores, un Atmel ATmega32u4 y un Sitara AM335x que trabaja a 1 GHz. Uno de estos microcontroladores se encarga de gestionar el conjunto de entradas y salidas, mientras que el otro permite la instalación de un sistema operativo tipo Linux capaz de crear interfaces graficas que interactúen con el exterior. Debido a estas características, el arduino TRE se puede considerar como un mini ordenador capaz de interactuar con sensores y demás elementos externos.

La plataforma Arduino se programa mediante el uso de un lenguaje propio basado en el lenguaje de programación de alto nivel Processing. Sin embargo, es posible utilizar otros lenguajes de programación, debido a que Arduino usa la transmisión serie de datos soportada por muchos de los lenguajes existentes. Para los que no soportan el formato serie de forma nativa, es posible utilizar software intermediario que traduzca los mensajes enviados por ambas partes para permitir una comunicación fluida.

3.5.2.- Historia:

El proyecto de arduino se inició en el año 2005 como un proyecto para estudiantes en el Instituto IVREA (Italia). En un principio, los estudiantes usaban el microcontrolador BASIC Stamp, cuyo coste era de 100 dólares, lo que se consideraba demasiado costoso para ellos.

El nombre del proyecto proviene del nombre del Bar di Re Arduino (Bar del Rey Arduino) donde Massimo Banzi (uno de los principales creadores de arduino) pasaba muchas horas. En su creación, contribuyó el estudiante colombiano Hernando Barragán, quien desarrolló la tarjeta electrónica Wiring, el lenguaje de programación y la plataforma de desarrollo. Una vez concluida dicha plataforma, los investigadores trabajaron para hacerlo más ligero y económico, además de hacerlo disponible para la comunidad de código abierto (hardware y código abierto).

El instituto IVREA acabó cerrando, así que los investigadores, entre ellos el español David Cuartielles, promovieron la continuidad del proyecto, ya que creían en el gran potencial del proyecto. Banzi afirmó años después que el proyecto no surgió como una idea de negocio, sino como una forma de intentar evitar el inminente cierre del Instituto de diseño Interactivo IVREA, ya que al crear un producto de hardware abierto, éste no podría ser embargado.

Para la producción en serie de la primera versión del arduino (UNO) se fijó como norma que el coste no fuera mayor de 30 euros, que fuera ensamblado en una placa de color azul, que fuese Plug and Play y que fuese capaz de trabajar con todas las plataformas informáticas tales como MacOSX, Windows y GNU/Linux. Las primeras 300 unidades se distribuyeron entre los alumnos del Instituto IVRAE, con el fin de que las probaran y empezaran a diseñar sus primeros prototipos basados en arduino.

3.5.3.- Librerías Arduino:

Aparte de la integración en una placa de un microcontrolador con todos los elementos externos necesarios para su correcto funcionamiento y el fácil acceso a los pines de entrada y salida, lo que hace tan popular a Arduino, son sus librerías.

Las librerías son subrutinas que permiten realizar funciones complejas como el control de un display LCD (LiquidCrystal), el anti rebote de pulsadores (Bounce), el manejo de servomotores (Servo), la comunicación serie con distintos elementos externos (Software Serial) o el envío y recepción de datos sobre una red de dispositivos o sensores mediante Two Wire Interface (TWI/I2C) (Wire) o mediante el protocolo SPI (SPI).

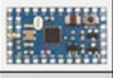
La mayoría de estas librerías están incluidas en el software de desarrollo de Arduino, aunque la posibilidad de crear nuevas librerías y el espíritu colaborativo del software libre hace que cualquiera que cree una librería pueda compartirla con el resto de desarrolladores.

3.5.4.- Versiones de Arduino:

En la actualidad existe una gran cantidad de versiones de Arduino, y aunque muchas de ellas comparten el mismo modelo de microcontrolador, varia el número de entradas y salidas, tanto analógicas como digitales o salidas de PWM (pulse width modulation) o la tensión de alimentación de la placa.

El hecho de que Arduino es una plataforma de hardware y software libre, hace posible que cualquier persona pueda desarrollar una placa de arduino con unas características que se adapten mejor a las necesidades de su proyecto, aunque luego esta quede disponible para cualquier otro usuario. De esta forma podemos encontrar otras placas similares a Arduino como Netduino, Rainbowduino, Digilent o DiamondBack.

En la siguiente tabla se puede ver una comparación de las principales placas de Arduino existentes:

								
Fabricante	Arduino	Arduino	Arduino	Arduino	Arduino	Arduino	Arduino	Netduino
Modelo	Pro Mini	Nano	Uno	Mega / Mega 2560	Leonardo	Micro	Due	Netduino 2
Microcontrolador	AVR Atmega 168 ó 328 8bits	AVR ATmega 168 ó 328 8bits	AVR ATmega 328 8bits	AVR ATmega2560 8bits	AVR ATmega 32u4 8bits	AVR ATmega 32u4 8bits	ARM SAM3X8E Cortex-M3 32bits	ARM STMicro STM32F2 Cortex-M3 32bits
Frecuencia	16Mhz	16Mhz	16Mhz	16Mhz	16Mhz	16Mhz	84Mhz	120Mhz
Memoria RAM	2KiB	2KiB	2KiB	8KiB	2.5KiB	2.5KiB	96KiB (64+32KiB)	60KiB
Memoria EEPROM	1KiB	1KiB	1KiB	4KiB	1KiB	1KiB	0	0
Memoria FLASH	16 ó 32KiB	16 ó 32KiB	32KiB	128 ó 256KiB	32KiB	32KiB	512KiB	192KiB
Pines digitales entradas/salidas	14/14	14/14	14/14	54/54	20/20	20/20	54/54	20/20
Tensión/corriente pines digitales	3.3v ó 5v 40mA	5v 40mA	5v 40mA	5v 40mA	5v 40mA	5v 40mA	3.3v 3~15mA (130mA entre todos)	3.3v~5v 25mA (125mA entre todos)
Pines analógicos entradas/salidas	6/0	8/0	6/0	16/0	12/0	12/0	12/2	6/0
Tensión/resolución pines analógicos	3.3v ó 5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	3.3v 12bits (4096 valores)	5v 12bits (4096 valores)
Pines con interrupción externa	2	2	2	6	2	2	-	-
Pines PWM	6	6	6	15	7	7	12	6
Conexiones Serial / UART	1	1	1	4	1	1	4	4
Conexiones I2C / TWI	1	1	1	1	1	1	2	1
Conexiones ISP / ICSP	1	1	1	1	1	1	1	1
Conexión USB	No (necesita adaptador externo)	Si	Si, USB-B	Si, USB-B	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB
Conexión USB de depuración	No	No	No	No	No	No	Si, MicroUSB	Si, MicroUSB
Conexión Ethernet	No	No	No	No	No	No	No	No
Conexión USB Host	No	No	No	No	No	No	Si	No
Almacenamiento por SD	No	No	No	No	No	No	No	No
Corriente en el pin de 5v	-	500mA	500~800mA	500~800mA	500~800 mA	500mA	800mA	-
Corriente en el pin de 3.3v	-	50mA	50mA	50mA	50mA	50mA	800mA	-
Voltaje de alimentación por el USB	3.3v ó 5v (sin usb)	5v	5v	5v	5v	5v	5v	5v
Voltaje de alimentación recomendado por el Jack	3.35 -12 V (modelo 3.3V) ó 5 - 12 V (modelo 5V)	7~12v	7~12v	7~12v	7~12v	7~12v	7~12v	7.5~9v
Voltaje de alimentación limite por el Jack	-	6~20v	6~20v	6~20v	6~20v	6~20v	6~20v	-

4.- Descripción general del sistema:

4.1.- Primer prototipo:

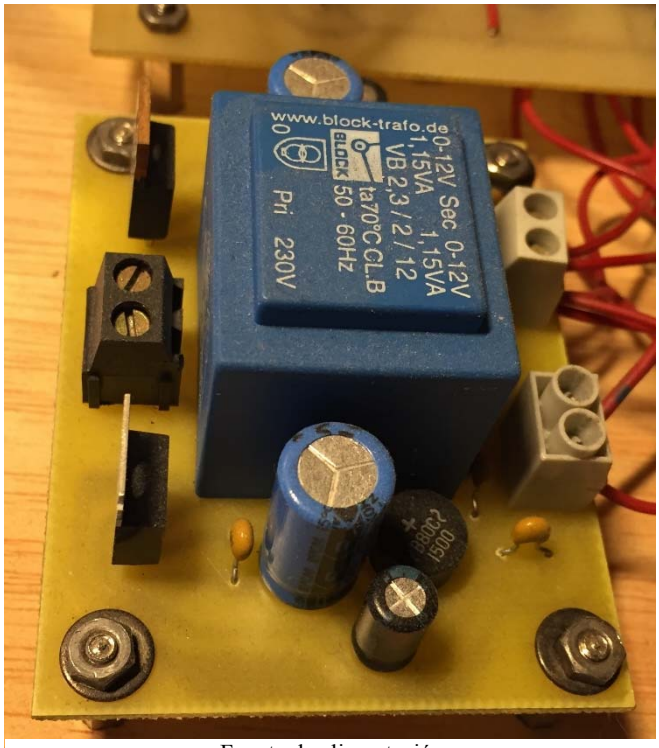
Como se ha dicho anteriormente, el profesor Antonio Pardina había diseñado un prototipo de entrenador de motores paso a paso basado en puertas lógicas que no había llegado a construirse, por lo que como paso previo a la realización de un entrenador más completo, se construyó dicho sistema para entender de forma correcta como funciona un motor paso a paso y las bases de lo que se quería lograr con este proyecto.

Las especificaciones de este equipo son las siguientes:

- Alimentación a partir de la red monofásica (230V).
- El motor realiza siempre pasos completos.
- Selección de frecuencia interna variable mediante un potenciómetro.
- Selección de frecuencia externa mediante un generador de señales.
- Posibilidad de paso único del motor.
- Selección de la dirección de giro del motor.

Este prototipo consta de varias partes diferenciadas (son distintas placas de circuito impreso) que interactúan entre si logrando el objetivo deseado.

4.1.1.- Fuente de alimentación:



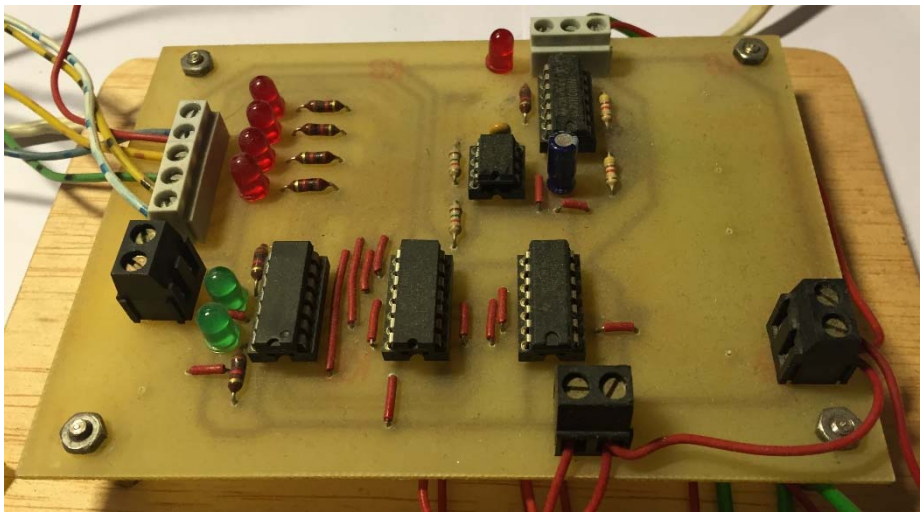
Fuente de alimentación

Esta es una fuente de alimentación doble (2 salidas de 5V) utilizada para alimentar todos los circuitos integrados y LEDS del prototipo, así como los dos puentes en H.

Consta de un transformador 230/12 V, dos puentes rectificadores de cuatro diodos, dos condensadores de filtrado y dos reguladores lineales de tensión de 5V (7805). En la fotografía se ven también otros cuatro condensadores que se encargan de eliminar el ruido que aparece en la tensión de salida.

Como se puede ver en la fotografía hay tres conectores, la entrada (230V) son los bornes negros y las salidas (2 salidas de 5V) son los bornes grises.

4.1.2.- Placa de control:



Placa de control

En la imagen superior se puede ver la placa de control utilizada en el prototipo inicial.

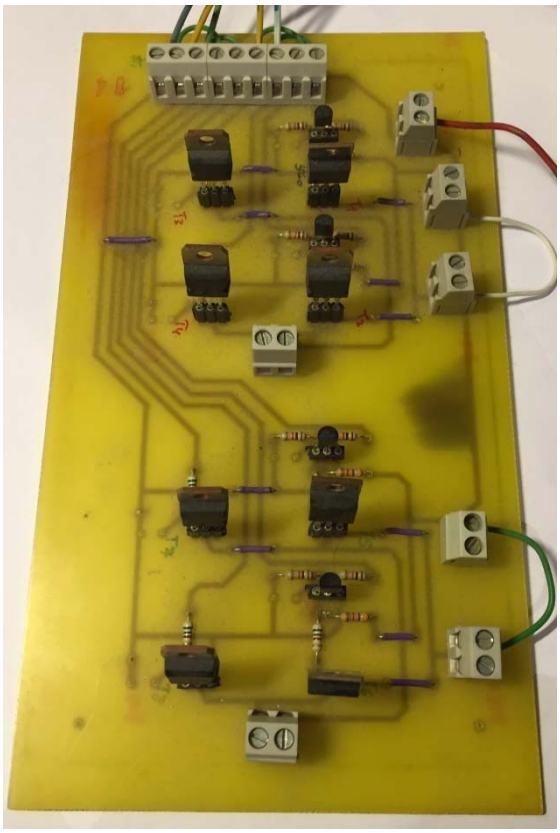
En ella pueden apreciarse varios circuitos integrados encargados de realizar los disparos que dan lugar a cada paso del motor, también podemos ver varios diodos LED que indican el estado de cada una de las cuatro salidas, y cuya combinación da lugar a los distintos estados del motor (tablas del apartado 3.4).

Los circuitos integrados utilizados en esta placa de circuito impreso son:

- 7474: Dos biestables.
- 7486: Cuatro puertas lógicas OR exclusivas.
- 7432: Cuatro puertas lógicas OR.
- 7408: Cuatro puertas lógicas AND.
- 7400: Cuatro puertas lógicas NAND.
- 7404: Seis inversores.
- 7402: Cuatro puertas lógicas NOR.

Hay también varios conectores, en el conector gris se encuentran las cuatro salidas correspondientes a cada transistor o pareja de transistores, así como una toma de tierra. En el conector negro situado al lado del gris se puede ver en código binario el número del estado en el que se encuentra el motor. El conector que se encuentra en la parte inferior de la fotografía se corresponde con la alimentación de 5V, y el de la derecha es la entrada de pulsos externos para hacer que el motor gire a una frecuencia generada con un instrumento externo al sistema.

4.1.3.- Puente en H:



Puente en H

En la imagen de la izquierda puede verse la placa de circuito impreso utilizada en el prototipo inicial como puente en H. En dicha placa hay dos puentes en H que trabajan de forma simultánea (como se ha visto en el apartado 3.4).

Un puente en H es un circuito electrónico que trabaja como convertidor de potencia, dichos puentes están disponibles como circuitos integrados, pero también pueden construirse a partir de componentes discretos (es el caso dela fotografía).

El término "puente en H" proviene de la típica representación gráfica del circuito, ya que su disposición recuerda a la letra H. Un puente en H se construye con 4 interruptores (mecánicos o mediante transistores) que conmutan por parejas transmitiendo potencia al motor paso a paso.

Si se observa la imagen superior, se puede ver un conector a la izquierda de la imagen, este conector es la entrada del puente en H, que se conecta a la salida de la placa de control. También existen cuatro conectores de dos hilos en la parte superior derecha, cuya función es la de medir la corriente de cada uno de los puentes mediante un sensor de corriente. Por ultimo en la parte derecha y superior izquierda de la placa se encuentran las salidas de sendos puentes en H, donde se conecta el motor paso a paso.

4.1.4.- Panel de control:

Panel de control

Este el panel desde el que se selecciona el comportamiento del motor paso a paso mediante diversos interruptores, un pulsador y un potenciómetro. En el panel de control podemos encontrar las siguientes opciones:

- Pulsos externos/internos: Mediante este interruptor se puede seleccionar si la frecuencia de entrada en conducción de los transistores del puente en H es la del oscilador interno del sistema o una externa generada mediante otro equipo (por ejemplo un generador de señales).

- Frecuencia manual: Si el interruptor se encuentra en esta posición, los pulsos se generan mediante un pulsador.

- Variador de frecuencia: Con el interruptor en esta posición se puede variar la frecuencia de los pasos mediante un potenciómetro.

- Giro izquierda/derecha: Mediante este interruptor se selecciona la dirección de giro del motor.

- Pulsos manuales: Este pulsador permite que el motor de un paso con casa pulsación. Para que esto funcione debemos seleccionar las opciones pulsos internos y frecuencia manual.

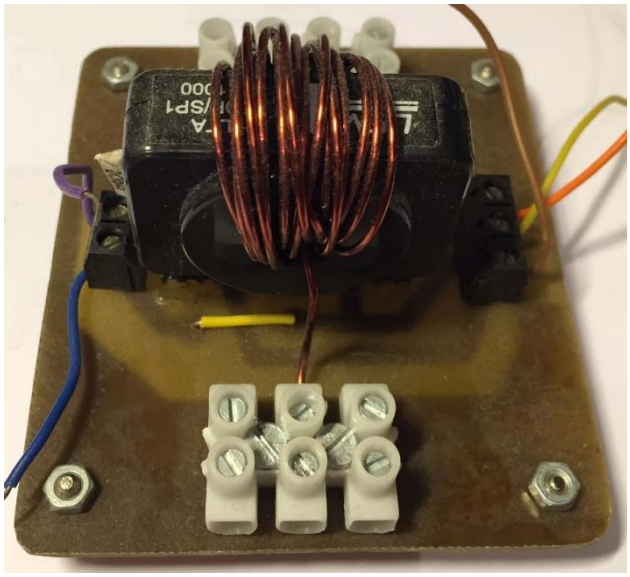
- Regulador de frecuencia: Este potenciómetro permite regular la frecuencia de los pasos del motor. Para que esta opción se encuentre activa, deben estar seleccionadas las opciones pulsos internos y variador de frecuencia.

4.1.5.- Sensor de corriente:

El dispositivo de la imagen es un sensor de corriente, cuya salida proporciona una tensión proporcional a la corriente que circula por la bobina. Esta relación es seleccionable en tres rangos dependiendo de la entrada seleccionada (en realidad hay tres arrollamientos con distinto número de vueltas, 5, 10 y 20).

Si se conecta la salida de este sensor a un osciloscopio, se puede ver la forma de onda de la corriente que circula por la bobina.

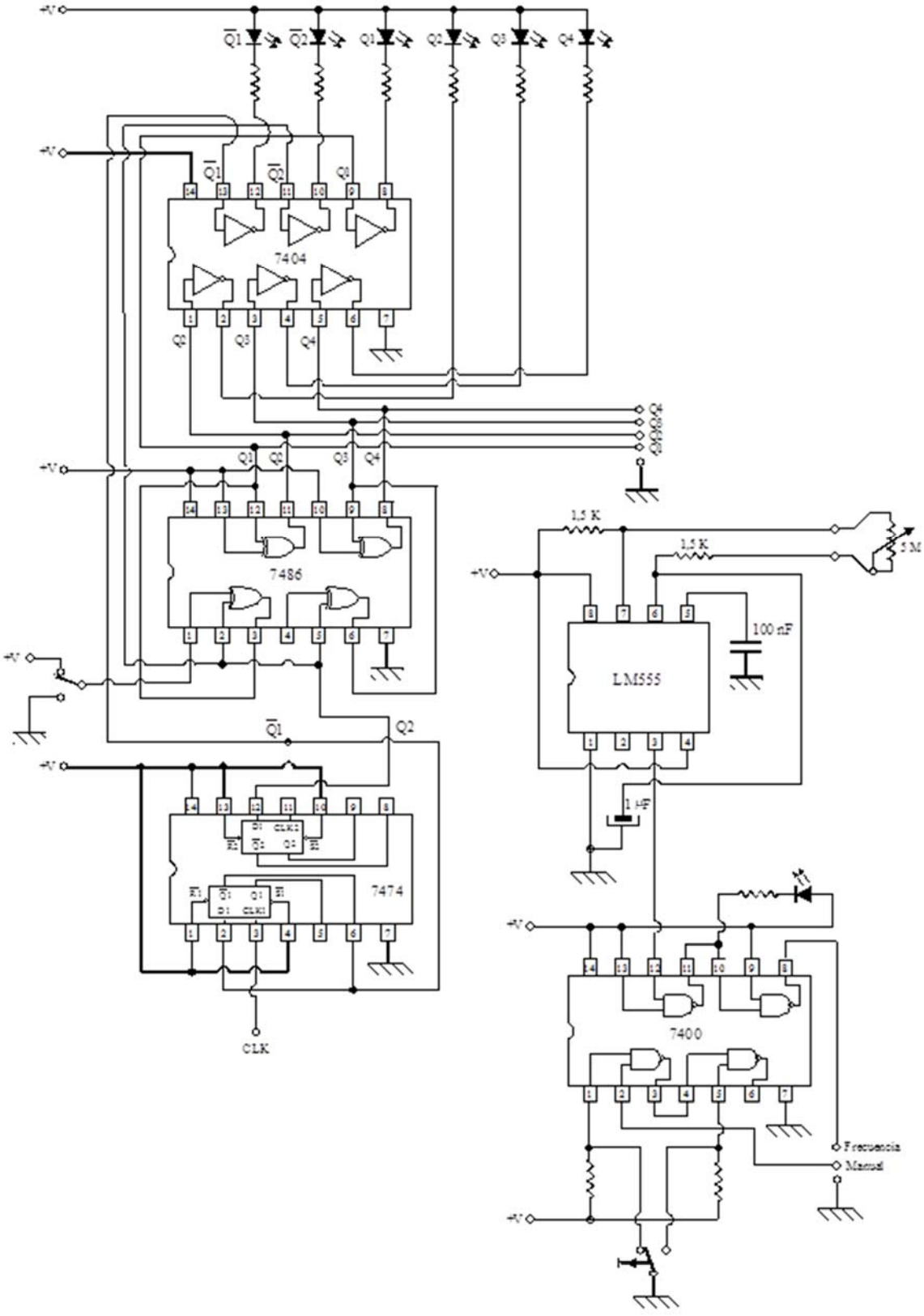
Este sensor se conecta al puente en H para poder ver la forma y el valor de la corriente que se entrega al motor paso a paso.



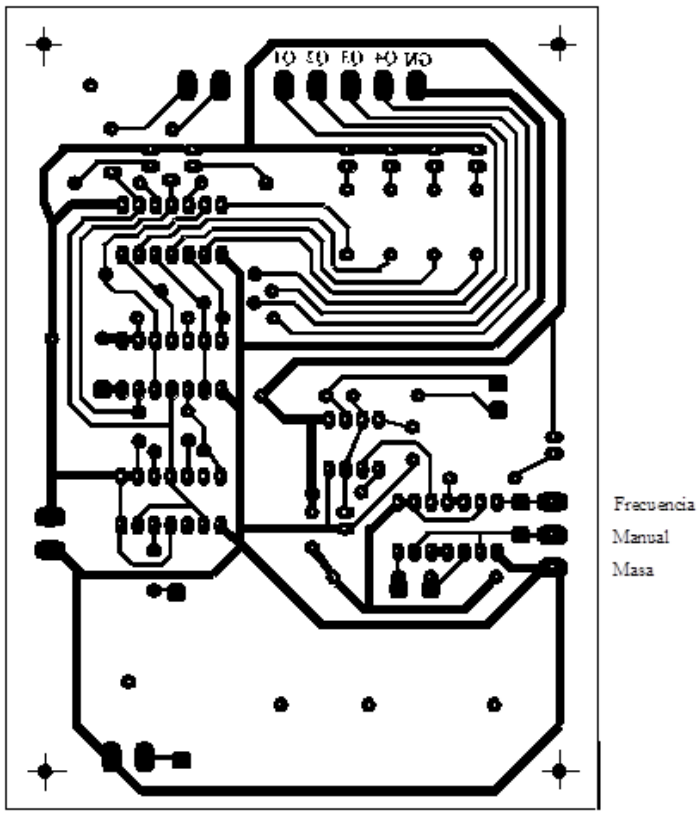
Sensor de corriente

4.1.6.- Esquemas y planos:

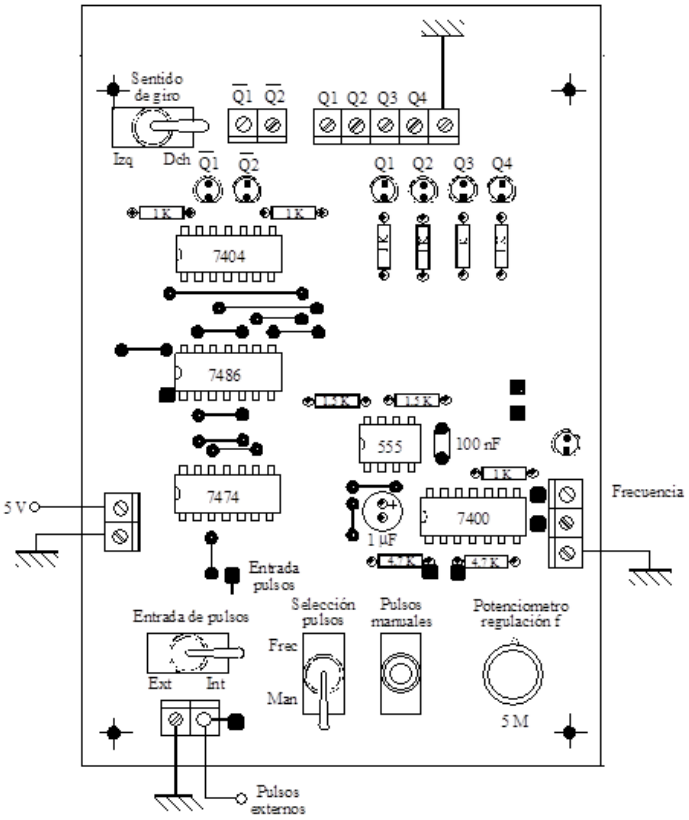
En las siguientes imágenes se pueden ver la interconexión entre los distintos componentes que forman el prototipo inicial, el plano de pistas de la PCB y el plano de serigrafía.



Esquema del prototipo inicial



PCB del primer prototipo.



Serigrafía del primer prototipo.

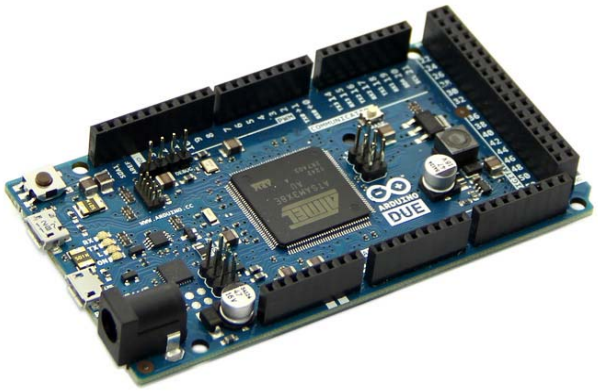
4.2.- Prototipo final:

Una vez realizado el prototipo inicial y observado su funcionamiento, se pasó a realizar el prototipo definitivo, que como ya se ha dicho anteriormente basa su funcionamiento en un Arduino due.

Aparte de este Arduino, este prototipo está compuesto por otros componentes, como un teclado, una pantalla LCD, una fuente de alimentación y un encoder, además del ordenador necesario para ver la forma de onda y el valor de la corriente por el motor paso a paso.

A continuación se pasa a detallar las partes que componen el prototipo.

4.2.1.- Placa Arduino due:



Placa Arduino due

La placa Arduino due es la más potente hasta el momento dentro de toda la gama de placas Arduino existentes en el mercado (Arduino Tre es la siguiente generación pero todavía se encuentra en fase beta).

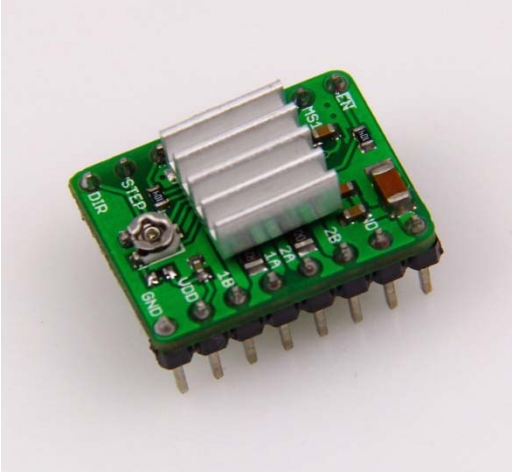
Esta placa se usa como elemento controlador de todo el equipo, ya que su microprocesador gestiona todas las entradas y salidas (tanto analógicas como digitales) necesarias para el correcto manejo de los distintos elementos que se le conectan.

Se ha decidido utilizar el Arduino Due en lugar de cualquier otra placa Arduino atendiendo a la velocidad del reloj, ya que este funciona a 84 MHz, mientras que el resto de placas trabajan a 16 MHz, y para este proyecto es necesaria bastante velocidad, ya que hay que leer el encoder y hacer girar el motor al mismo tiempo, y aunque se utiliza hardware específico para estas tareas, sigue siendo necesario una velocidad de trabajo bastante elevada

Esta placa de Arduino cuenta con las siguientes características:

- Microcontrolador: ATMEL AT91SAM3X8E
- Tensión de operación: 3.3 V
- Tensión de entrada recomendada: 7-12 V
- Límites de tensión de entrada: 6-16 V
- Entradas/salidas digitales: 54 (12 PWM)
- Entradas analógicas: 12
- Salidas analógicas: 2
- Corriente total de salida: 130 mA
- Corriente para el pin de 3.3 V: 800mA
- Corriente para el pin de 5 V: 800mA
- Memoria flash: 512kB
- SRAM: 96 kB
- Velocidad de reloj: 84 MHz

4.2.2.- Controlador del motor:

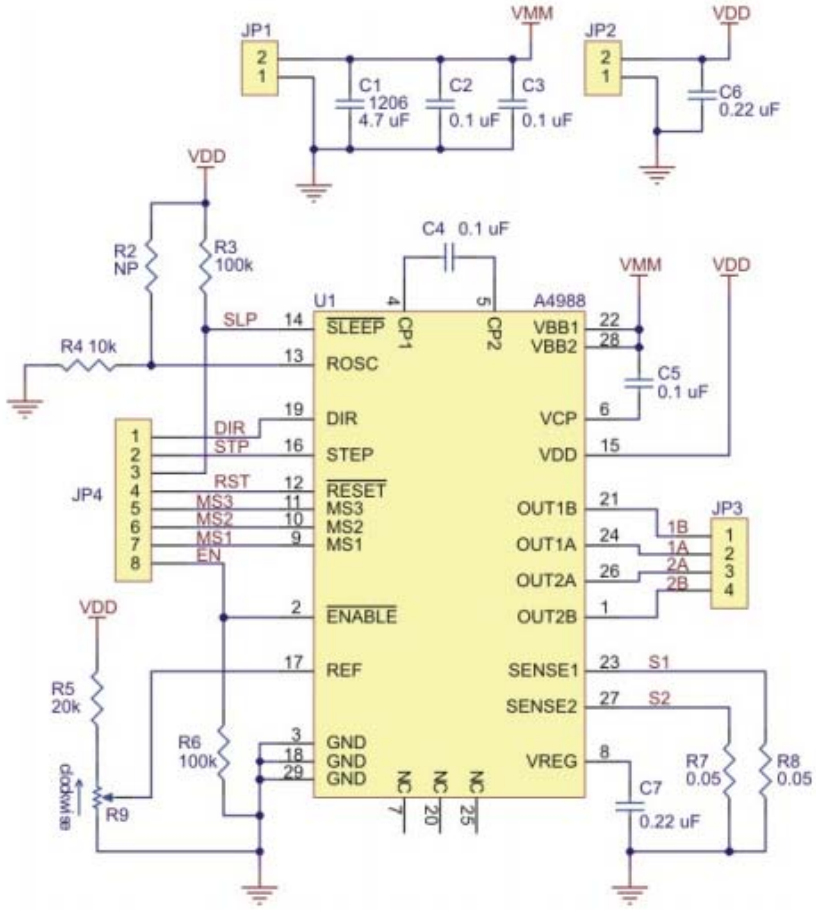


Controlador de motor paso a paso

Se ha utilizado el controlador A4988 de Allegro para manejar el motor paso a paso, ya que simplifica bastante el código del programa de arduino y aporta una mayor precisión en los movimientos del mismo.

Este circuito integrado viene montado en una pequeña PCB, que hace que su inclusión en la placa de circuito impreso general del proyecto sea mucho más simple, ya que al incluirse algunos componentes pasivos como resistencias o condensadores necesarios para su funcionamiento en esta pequeña PCB, se reducen las conexiones a realizar al circuito integrado respecto del caso un el que se usase el integrado sin esta pequeña PCB. Al usarse componentes SMD (Surface Mount Component) se consigue un tamaño muy reducido de la placa de circuito impreso.

En la siguiente imagen puede verse como se han conectado los distintos elementos de los que se ha hablado al circuito integrado A4988 de Allegro.



Conexionado del circuito integrado Allegro A4988

Este pequeño montaje reduce los 23 pines del circuito integrado a tan solo 16, los cuales se detallan a continuación:

- ENABLE: Es una entrada negada, y permite habilitar el controlador mediante un 0.
- MS1, MS2 y MS3: Son las entradas de control para seleccionar el tipo de paso del motor.
- RESET: Es una entrada negada, y mantiene el controlador en condiciones iniciales a no ser que se encuentre en estado alto.
- SLEEP: Es una entrada negada, y activa el modo sleep mientras esta entrada se encuentre a 0, reduciendo así el consumo del controlador.
- STEP: Es la entrada de reloj que marca la frecuencia de los pasos del motor.
- DIR. Es la entrada que indica el sentido de giro del motor.
- VMOT y GND: Son las entradas de alimentación del motor.
- 1A, 1B, 2A y 2B: Son las salidas que se conectan al motor.
- VDD y GND: Es la alimentación del controlador. Esta masa (GND) debe ser distinta que la del motor (VMOT).

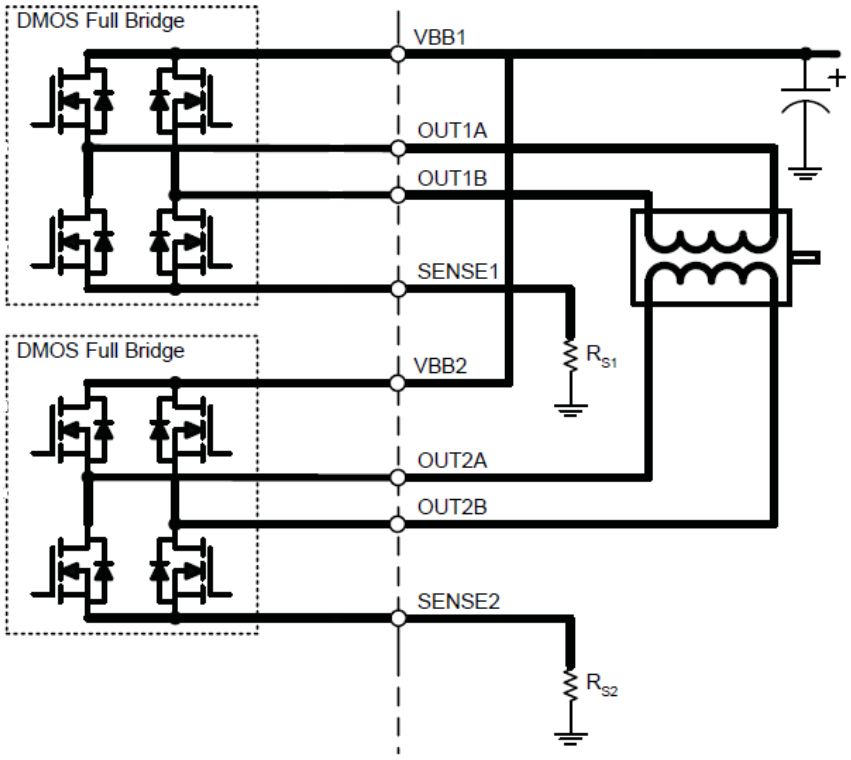
A pesar de que la pequeña PCB del controlador ya incluye varios componentes externos al integrado, es necesario colocar un condensador electrolítico de entre los pines VMOT y GND 100uF (terminal positivo en VMOT).

Este controlador de motores paso a paso necesita dos fuentes de alimentación, una de 5v para la alimentación del circuito integrado y de los componentes de la pequeña PCB en la que se encuentra el circuito integrado, y otra fuente de entre 8 y 35 v para alimentar el motor paso a paso conectado al controlador. La tensión de esta última fuente de alimentación depende de la tensión de funcionamiento del motor utilizado, en el caso de este proyecto es de 12 voltios.

El tipo de paso que debe dar el motor selecciona con la combinación de las entradas MS1, MS2 y MS3, dando lugar a las siguientes combinaciones:

MS1	MS2	MS3	Tipo de paso
0	0	0	Paso completo
1	0	0	Medio paso
0	1	0	Cuarto de paso
1	1	0	Octavo de paso
1	1	1	Dieciseisavo de paso

Este controlador también cuenta con los dos puentes en H necesarios para que el motor pueda girar. Los cuatro pines de cada puente se reducen a dos al conectarlos tal y como se ha mostrado en la imagen anterior (conexionado del circuito integrado Allegro A4988), por lo que solo es necesario conectar los pines de salida 1A, 1B, 2A y 2B. A continuación se muestra un extracto de la hoja de características del circuito integrado en el que se ve la configuración de los dos puentes en H.



Puentes en H del circuito integrado Allegro A4988

Por ultimo cabe destacar que debido a los dos puentes en H que se encuentran en el interior del circuito integrado, se maneja un nivel bastante elevado de corriente, lo que produce un calentamiento del mismo y hace totalmente imprescindible la colocación de un disipador que evite el sobrecalentamiento del circuito integrado y por consiguiente la destrucción del controlador.

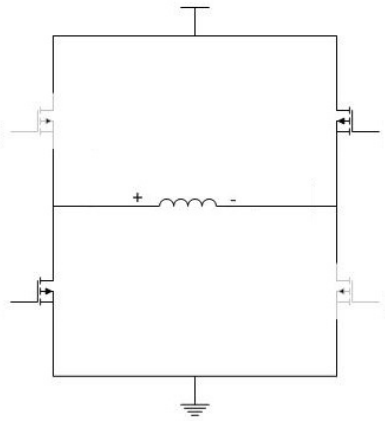
4.2.2.1.- Funcionamiento:

Las corrientes de cada uno de los dos puentes en H de salida, al igual que la corriente por los canales de los transistores de efecto campo (DMOS FET) están regulados por un circuito de control de modulación de anchura del pulso (PWM) con un periodo fijo. En cada paso, la corriente de cada puente, es regulada mediante su resistencia externa de control de corriente (RS1 y RS2), la tensión de referencia (V_{REF}) y la tensión de salida de su conversor digital/analógico**.** Las resistencias RS1 y RS2 ya están incluidas en la PCB en la que viene montado el controlador.

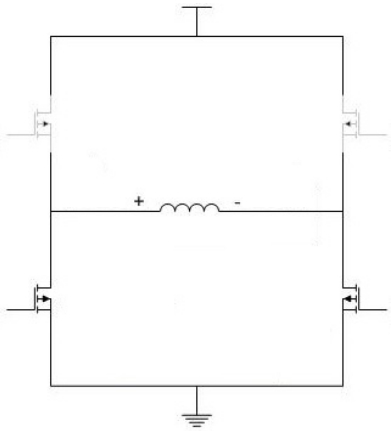
Al arrancar o resetear el controlador, los conversores digitales/analógicos y la polaridad de la corriente de fase vuelven a su estado inicial, y el regulador de corriente se configura con un modo de decaimiento mixto para ambas fases. Cuando hay un impulso en la entrada STEP, el relé de transmisión secuencia automáticamente la polaridad de la corriente y los conversores analógicos/digitales al siguiente nivel.

El modo de decaimiento es la forma de controlar la distorsión existente en las ondas debido a la oposición de las cargas inductivas a los cambios bruscos de corriente. Existen tres modos de decaimiento, el rápido, el lento y el mixto:

- En el modo rápido, se usan dos transistores opuestos del puente en H, consiguiéndose así una rápida reducción de la corriente a cero, pero también es el modo que más distorsión genera.
- En el modo lento, se usan los dos transistores del mismo segmento del puente en H. En este caso la corriente tarda bastante en hacerse cero, ya que la corriente se anula al recircular por un circuito resistivo, por el contrario, la distorsión es mucho menor que en el caso rápido.
- El modo mixto es una mezcla de los dos anteriores. Este modo une las ventajas de los dos anteriores, logrando una velocidad de anulación de la corriente bastante rápida (aunque más lenta que en el modo rápido) y una distorsión despreciable para la mayoría de las aplicaciones. Estas características hacen que el modo mixto sea el más utilizado cuando se ha de trabajar con cargas inductivas.



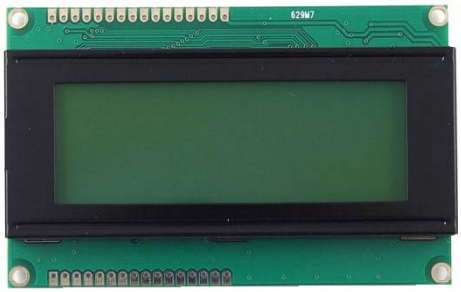
Decaimiento rápido



Decaimiento lento

Mientras el controlador está en funcionamiento, si los nuevos niveles de salida de los conversores analógicos/digitales son inferiores que los niveles de salida anteriores, el modo de decaimiento de los puentes activos se cambia a mixto. Si por el contrario los niveles de salida de los conversores son superiores o iguales a los anteriores, el modo de decaimiento de los puentes activos se cambia a lento. Esta selección automática del modo de decaimiento mejora el funcionamiento del control de los pasos reduciendo la distorsión de la forma de onda de la corriente resultante de la radiación electromagnética del motor.

4.2.3.- Pantalla LCD:


Display LCD

Una pantalla LCD (Liquid Cristal Display) es una pantalla formada por muchos pixeles monocromáticos que al encenderse forman letras o pequeños dibujos. Estos pixeles se agrupan en bloques de 5x8, dando lugar cada uno de estos bloques a un carácter. En este tipo de pantallas es necesario una luz posterior (retroiluminación) para hacer visibles los pixeles activos.

Este tipo de display tiene un tiempo de respuesta bastante bajo, de forma que si la información mostrada en ella cambia muy rápidamente, no se verán más que varios rectángulos negros. Por esto es importante que las imágenes o texto de la pantalla no se actualicen demasiado rápido, siendo perfectos para mostrar imágenes estáticas o información durante un periodo de tiempo largo.

Como se ha dicho anteriormente, se pueden crear pequeños dibujos en estas pantallas, para ello se puede elegir que pixeles concretos se encienden o apagan de cada bloque, dando así lugar a una imagen de un tamaño máximo de 8 x 5 pixeles.

Para este proyecto se ha utilizado un display de 4x20 (Cuatro líneas y veinte columnas) que permite visualizar distintos parámetros del motor en directo, así como seleccionar las condiciones de funcionamiento para el ensayo deseado.

Esta pantalla está controlada por el integrado LCM1602A de Hitachi (es el más utilizado en este tipo de dispositivos) o uno equivalente, ya que este tipo de displays se pueden manejar fácilmente mediante una librería de arduino, y además se reducen las conexiones a 12 pines, ya que mediante la librería podemos pasar de utilizar los 8 pines de transmisión de datos a la mitad (transmisión de 4bits), y de estos 12 pines solo es necesario conectar al arduino 6 de ellos.

El display utilizado cuenta con los siguientes pines:

- VSS: Es la conexión a masa.
- VDD: Es la alimentación del display, que puede ser de 3 a 5v, en este proyecto se conecta a 5V.
- V0: Es la regulación del contraste. Se conecta a un potenciómetro de 10kΩ para poder regularlo.
- RS: Es la selección de registro (Register Selecction), si está a cero se selecciona instrucción y si está a 1 se selecciona dato.
- R/W: Es la selección de lectura o escritura (Read/Write), y se conecta a GND porque en este caso solo se va a escribir.
- E: Habilitación (Enable), el display se habilita con un flanco descendente en esta entrada.
- DB0 a DB7: Son las líneas de envío de datos. La librería de Arduino utilizada para el manejo del display permite la transmisión mediante 4 bits en vez de los 8 bits necesarios si no se usa esta librería, por lo que solo se usan los pines del DB4 al DB7, pudiendo dejar libres 4 pines del microcontrolador.
- A y K: Son el ando y el cátodo de la retroiluminación del display. El ánodo (A) se conecta a 5 voltios a través de una resistencia de 15Ω, y el cátodo (K) se conecta a masa (GND).

4.2.4.- Encoder:

Un encoder es un dispositivo óptico que permite conocer en todo momento la posición angular del eje del motor, así como la velocidad de giro del mismo. Su funcionamiento se basa en el contaje de unas marcas (rayas) que giran junto con el eje del motor. Este dispositivo consta de dos partes:


Encoder ERN 1321

- Un disco rayado solidario al eje del motor, que determina la resolución del encoder. De esta forma, cuantas más marcas tenga el disco, mayor será la resolución del encoder y podrá aportar una mayor precisión al indicar la posición del eje del motor. El encoder utilizado en el proyecto es un Heidenhain ERN1321 que tiene un disco con 4096 marcas por vuelta.
- Un lector óptico que cuenta el número de marcas que van pasando, y sabiendo la distancia entre las mismas se puede determinar la cantidad que ha girado el eje del motor, y por lo tanto la posición del eje.

Dependo de distintos criterios como el tipo de señal de salida, el tipo de contaje o el tipo de movimiento a contabilizar, se pueden diferenciar distintos tipos de encoder:

- Atendiendo a la forma de onda de la señal de salida, se pueden diferenciar tres tipos de encoder. Así, podemos diferenciar entre encoders senoidales, voltio pico a pico (Vpp) o TTL.
- Atendiendo a la forma de contaje existen los encoders absolutos, en los que se conoce en todo momento la posición del encoder, y los incrementales, en los que la posición que se conoce es respecto de un punto ya conocido.
- Atendiendo al tipo de movimiento a medir, se diferencia entre encoders lineales (también llamados reglas) o rotativos.

Atendiendo a las características del proyecto, se ha elegido un encoder rotativo incremental TTL.

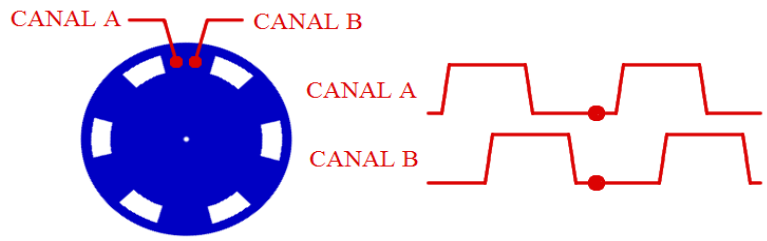
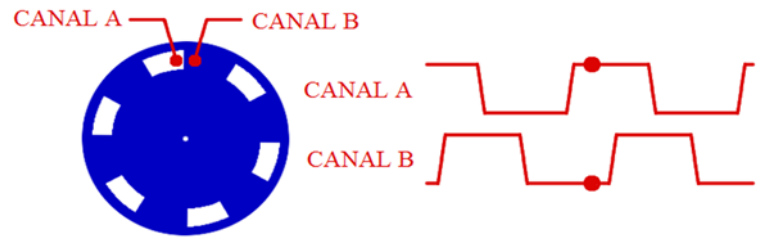
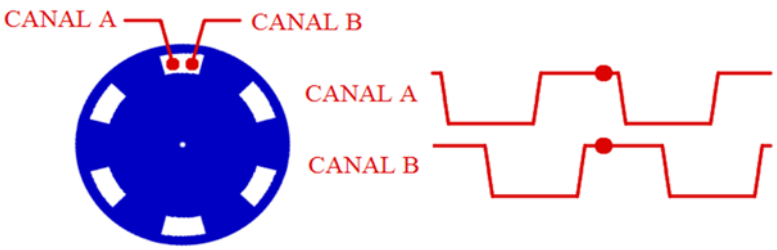
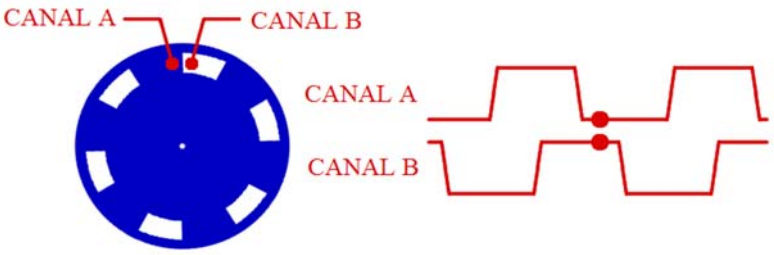
Hay que destacar que la utilización de este encoder se debe a que se disponía de uno, si no hubiese sido así, se habría elegido otro modelo, ya que el ERN 1321 tiene una resolución muy alta, y esta aplicación no la requiere. Podría usarse un encoder de 1024 pulsos o incluso algo menos y el resultado sería bastante parecido

4.2.4.1.- Funcionamiento:

Con el fin de aumentar la cantidad de pulsos que se pueden leer en una vuelta del encoder, se utilizan dos sensores se lectura de las marcas del disco rayado solidario al eje de rotación y que se encuentra en el interior del encoder. Estos dos sensores están desfasados entre sí, de forma que se produzcan dos señales (A y B) desfasadas entre si 90º. Mediante esta simple técnica se multiplican por dos los pulsos que proporciona el encoder en cada vuelta. Si se observan tanto los flancos ascendentes como descendentes de ambas señales, se pueden volver a multiplicar por dos los pulsos del encoder, por lo tanto el resultado de este desfase entre los dos sensores se traduce en una multiplicación por cuatro del número de pulsos del encoder, aumentando así notablemente la resolución del sensor.

En el caso de este proyecto, se pasa de 4.096 a 16.384 pulsos por vuelta, pudiendo alcanzarse una precisión de 0,0219º. Esto significa que cada vez que se recibe un flanco tanto ascendente como descendente de cualquiera de las señales (A o B), el eje del encoder ha girado 0,0219º.

A continuación se muestra el funcionamiento de contaje de los pulsos:

Estado	Descripción gráfica	Canal A	Canal B
1		0	0
2		1	0
3		1	1
4		0	1

Cuando el disco rayado del encoder gira en sentido horario, los cambios de estado son detectados primero por el canal A, y después por el B. Cuando gira en sentido anti horario, el canal B es el primero en detectar los cambios, y el A es el segundo en hacerlo. Este sistema es llamado "codificación de la cuadratura" porque como ya se ha dicho anteriormente, las formas de onda detectadas por los dos canales están desfasados 90º.

Aunque en este proyecto no las utilizamos, la mayoría de los encoders tienen cuatro señales más, los canales A y B negados, que hacen que la transmisión de las señales sea más segura y pueden utilizarse para diagnosticar una posible avería. También podemos

encontrar la señal de IO o index, que es el origen de contaje del encoder, de forma que se alimente el encoder habrá que buscar esa marca para referenciar el contaje que se haga posteriormente a este punto, en nuestro caso no es necesario ya que esta función se hará de forma manual. Por último, esta señal de IO también tiene su negada, cuya finalidad es la misma que las señales negadas de los canales A y B.

En el caso de este encoder, como puede ser montado en el interior de un motor, también están disponibles dos señales más que se usarían solo si se diese este caso, estas señales son la sonda de temperatura del motor, que como su propio nombre indica, proporciona información acerca de la temperatura interna del motor.

4.2.5.- LS7366R:

El circuito integrado LS7366R es un componente electrónico utilizado para realizar el contaje de señales provenientes de un encoder que proporcione una salida de onda cuadrada con niveles de 5 voltios (TTL).

Este circuito integrado es un contador CMOS de 32 bits con una interface serie directa para los relojes de codificación de la cuadratura de los encoders incrementales. También permite la detección de la marca de IO, aunque en este proyecto no se va a utilizar.

La comunicación con un microcontrolador o microprocesador se realiza mediante cuatro líneas de comunicación, usando el protocolo SPI. El funcionamiento de este bus de comunicaciones se explica en el apartado 5.

Este componente requiere una configuración previa a su uso, ya que hay que configurar el tipo de contaje, que puede ser de 1, 2, 3 o 4 bytes, cuantos ciclos de contaje hay por vuelta, el divisor del reloj, si es contaje libre o se reinicia cada vuelta, si existe señal de IO o no y en caso de haberla, que hacer al detectarla o si el IO es síncrono o asíncrono.

4.2.5.1.- Funcionamiento:

Como se ha dicho en el apartado anterior, la comunicación de este dispositivo con el microcontrolador o microprocesador se realiza mediante un protocolo de cuatro hilos llamado SPI, y cada transmisión se organiza en bloques de 1 a 5 bytes de datos. Cada ciclo de transmisión se inicia con una transición de alto a bajo (flanco descendente) de la entrada SS. El primer byte recibido de un ciclo de transmisión es siempre un byte de instrucción, y del segundo al quinto byte son interpretados como bytes de datos. Cada ciclo de transmisión termina con una transición de bajo a alto (flanco ascendente) de la entrada SS. Los bytes recibidos llegan al circuito integrado a través del pin MOSI, con el bit más significativo primero, coincidiendo con cada flanco ascendente del reloj SCK. La salida de datos se realiza a través de la salida MOSI, también con el bit más significativo primero y al producirse un flanco ascendente de SCK.

Es imprescindible que acabe una transmisión de datos para comenzar otra nueva, es decir, no se pueden enviar y recibir datos al mismo tiempo. Por ejemplo, si el integrado está enviado datos, ignorara cualquier orden de lectura de datos hasta que haya un flanco

ascendente y otro descendente en la entrada SS, momento en el que comenzara otra transmisión de datos en cualquiera de los dos sentidos (envío o recepción).

El contador puede configurarse para operar como un contador de uno, dos, tres o cuatro bytes. Una vez que se ha realizado esta configuración, los registros CNTR, DTR y OTR se configuran con el mismo número de bytes que hemos ya definido. El contenido de las instrucciones o datos se ajusta automáticamente según la configuración del número de bytes que hayamos definido. De esta forma si se ha configurado como un contador de 4 bytes, la instrucción de escribir en el registro DTR espera 3 bytes de datos después del byte de instrucciones. Una vez que se han transmitido los bytes indicados, se ignora cualquier otra instrucción posterior hasta que comience un nuevo ciclo de transmisión.

El contador se puede programar para trabajar en distintos modos, escribiendo las características del modo de operación en los registros MDR0 y MDR1. Para indicar si se quiere leer o escribir y el registro de origen o destino, hay que enviar un byte de configuración al registro IR.

B7	B6	B5	B4	B3	B2	B1	B0

B2, B1, B0 = XXX (da igual)

B5, B4, B3 = 000: No se selecciona nada
= 001: Seleccionar MDR0
= 010: Seleccionar MDR1
= 011: Seleccionar DTR
= 100: Seleccionar CNTR
= 101: Seleccionar OTR
= 110: Seleccionar STR
= 111: No se selecciona nada

B7, B6 = 00: Borrar registro
= 01: Leer registro
= 10: Escribir en el registro
= 11: Cargar registro

Por ejemplo, si se desea escribir en el registro MDR1, se debe mandar el byte 10010000 (B2, B1 y B0 podrían ser cualesquiera) al registro IR, este registro es en el que se escribe el primer byte que se envía en cada transmisión. Es muy normal que estos bytes se envíen en formato hexadecimal, ya que así queda una instrucción más corta. En este caso se enviaría un 90.

El siguiente paso es configurar el contador mediante los registros MDR0 y MDR1, enviando los bytes correspondientes. Es necesario hacer esto cada vez que se enciende el sistema, ya que estos registros se inicializan a cero cada vez que se arranca el dispositivo.

MDR0

B1, B0 = 00: Contaje sin cuadratura
= 01: Contaje con cuadratura x1
= 10: Contaje con cuadratura x2
= 11: Contaje con cuadratura x4

B3, B2 = 00: Contaje libre
= 01: Contaje en ciclo único
= 10: Contaje con rango limite
= 11: Contaje con modulo n

B5, B4 = 00: Sin index
= 01: Transferir DTR a CNTR al leer la marca de index
= 10: Borrar CNTR al leer la marca de index
= 11: Transferir CNTR a OTR al leer la marca de index

B6 = 0: Index asíncrono
= 1: Index síncrono

B7 = 0: Divisor del reloj por 1
= 1: Divisor del reloj por 2

MDR1

B1, B0 = 00: Contador de 4 bytes
= 01: Contador de 3 bytes
= 10: Contador de 2 bytes
= 11: Contador de 1 bytes

B2 = 0: Habilita el contaje
= 1: Deshabilita el contaje

B3 No usado

B4 = 0: Sin función
= 1: FLAG en IDX

B5 = 0: Sin función
= 1: FLAG en CMP

B6 = 0: Sin función
= 1: FLAG en BW

B7 = 0: Sin función
= 1: FLAG en CY

El sistema para enviar datos a estos dos registros es el mismo que para enviarlos al registro IR.

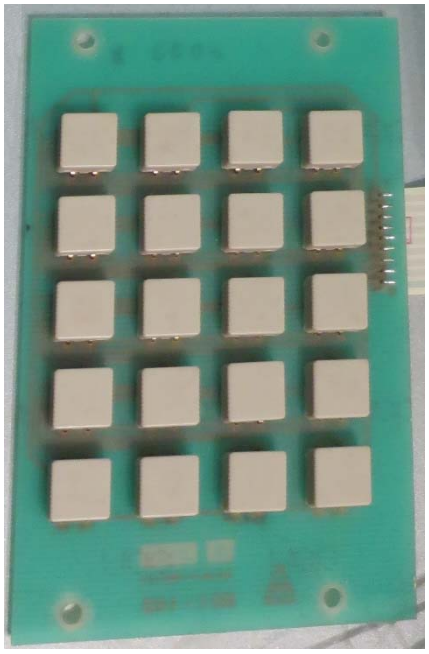
A continuación se van a describir los pines del circuito integrado:

- A y B: Son las salidas del encoder. En caso de encoders con contaje de cuadratura, A y B son dos señales desfasadas 90º, pero si el contaje no es de cuadratura, la señal A se utiliza para el contaje y la señal B indica la dirección de contaje.
- INDEX: Es la entrada a la que se conecta la señal de IO o index proveniente del encoder.
- fcki y fcko: Se debe conectar un cristal de cuarzo entre estas dos entradas para generar el reloj básico de filtrado. También se puede conectar esta entrada a un reloj externo generado por otro dispositivo. Este reloj puede dividirse mediante el valor del bit 7 del registro MDR0.
- SS: Es la entrada de habilitación de la comunicación, tanto envío como recepción de datos.
- CNT_EN: Habilita el contaje.
- LFLAG y DFLAG: Son dos salidas programables para gestionar el overflow del contador. La configuración de estas dos salidas se realiza en el registro MDR1.
- MOSI: Salida del maestro y entrada del esclavo. Es la entrada del dispositivo.
- MISO: Entrada del maestro y salida del esclavo. Es la salida del integrado.
- SCK: Serial clock. Es el reloj de sincronización de la transmisión de datos.

Por último, se van a describir el resto de los registros existentes en el dispositivo:

- DTR: Es un registro de entrada de datos configurable por software de 8, 16, 32 o 64 bits, en el que se puede escribir directamente desde la entrada MISO.
- CNTR: Es un registro configurable por software de 16, 32 o 64 bits. En este registro se almacena el contaje de los pulsos del encoder.
- OTR: Es un registro configurable por software de 16, 32 o 64 bits que puede ser leído por la salida MISO. Este registro se usa para poder leer el valor del contaje del encoder mientras que el registro CNTR sigue aumentando o disminuyendo su valor sin interferir en el contaje.

4.2.6.- Teclado:



Para la introducción de datos al sistema como la velocidad de giro o los grados que debe girar el motor, se utiliza un teclado de cuatro columnas y cinco filas, lo que hace un total de veinte teclas, aunque no se usan todas ellas.

Este teclado está construido de una forma muy sencilla. Los veinte pulsadores que lo componen, están soldados a una placa de circuito impreso, y las pistas de esta PCB conectan todos los pulsadores entre si formando una matriz de cuatro columnas y cinco filas. Los teclados que están construidos mediante esta técnica se denominan teclados matriciales, ya que como se ha dicho más arriba, la conexión entre todos los pulsadores forma una matriz de n filas * n columnas.

La ventaja de este tipo de teclados es que se reduce significativamente la cantidad de entradas necesarias en el microprocesador o microcontrolador respecto del caso de un teclado formado por varios pulsadores independientes, pasando así de veinte entradas (una por pulsador) a solo nueve entradas/salidas (cuatro de las columnas y cinco de las filas).

El inconveniente de este tipo de teclados es que es mal difícil de gestionar que uno formado por varios pulsadores independientes, aunque este problema se reduce significativamente al utilizar la librería keyboard de Arduino, que parametrizándolo de forma correcta asigna el valor de la tecla pulsada a una variable. Esta librería no se incluye por defecto en el software de desarrollo de Arduino, pero es fácil de conseguir a través de internet, tanto en la web oficial de Arduino como en foros o páginas web dedicadas al tema.

El uso de este teclado y no de otro de similares características, se debe a que es un elemento del que ya se disponía. Es un teclado reutilizado procedente de un plotter antiguo del que se han aprovechado varios componentes para distintas aplicaciones.

4.2.6.1.- Funcionamiento:

Como ya se ha dicho en el apartado anterior, el teclado utilizado es de tipo matricial, es decir, que los pulsadores que lo componen están conectados entre sí formando una matriz. En la siguiente imagen se representa de forma esquemática la conexión entre los veinte pulsadores del teclado.

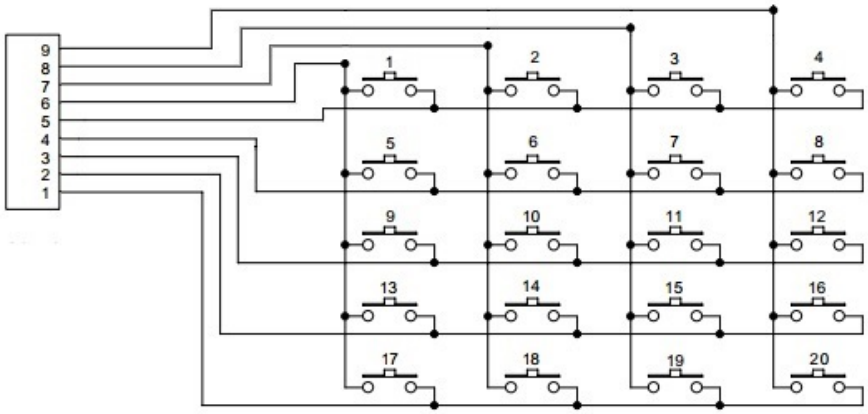


Diagrama de conexión de un teclado matricial de 4 x 5

Como se puede ver en el diagrama, cada vez que se activa uno de los pulsadores, se produce la unión de una fila y una columna, y analizando cuál de las uniones es la que se ha producido, se puede saber que tecla ha sido pulsada.

Para hacer este análisis, se configuran los pines a los que se han conectado las filas como salidas, y los pines de las columnas como entradas. Una vez hecho esto, hay poner un valor alto en uno de los pines definidos como salidas (filas) y un estado bajo en el resto, y después leer todos los pines configurados como entradas. Si alguna de estas entradas esta activa, significa que se ha pulsado uno de los pulsadores, y uniendo la información de la fila y columna activadas se deduce la tecla pulsada. Este proceso también puede realizarse poniendo en estado bajo una de las filas y en estado alto el resto

Este proceso de lectura del teclado debe repetirse en bucle para cada fila, "mirando" así continuamente si hay alguna tecla pulsada, el proceso de rotar el estado alto entre los pines de salida se denomina walking ones, y si lo que se rota es un estado bajo, se denomina walking zeros. Esta es la labor que realiza la librería keyboard para Arduino.

Este tipo de teclados no requieren alimentación ni conexión a masa, ya que la alimentación necesaria para llevar tensión hasta una de las entradas, la proporcionan las salidas a las que se han conectado las filas de la matriz. Esto hace que el número de pines a conectar al sistema que controla en teclado sea bastante reducido, concretamente es la suma del número de filas y columnas.

En la siguiente imagen se puede ver la conexión que se produce entre una fila y una columna al pulsar una de las teclas. También se representa el fuljo de corriente, indicando su sentido mediante flechas. En el caso de la imagen se activa el pin tres, y al leer las entradas, se detecta que la columna conectada al pin seis está activa, de lo que se deduce que se ha pulsado la tecla número nueve.

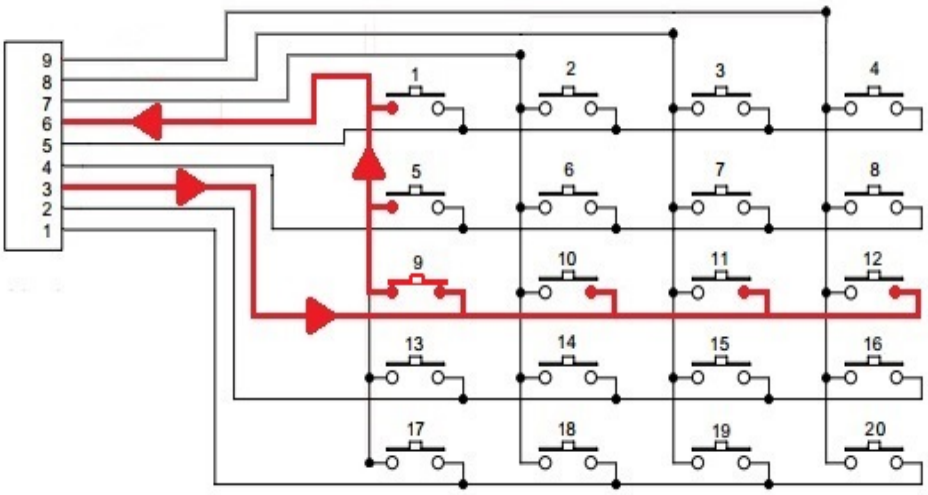


Diagrama de lectura de un teclado matricial de 4 x 5

Este sistema se puede aplicar a teclados de cualquier tamaño, siempre y cuando se mantenga la forma de conexión entre sus teclas, lo único que cambiara será el número de entradas y salidas a utilizar en función del número de filas y columnas de cada teclado.

4.2.7.- Motor paso a paso:

Para este proyecto se ha utilizado un motor paso a paso bipolar de 400 pasos por vuelta de la marca Sinano Electric. Esto significa que cada vez que el motor da un paso completo, su eje gira 0,9º.

La elección de este motor se debe a sus reducidas dimensiones (un cubo de 40 mm de lado y 30 mm de altura), al par que es capaz de proporcionar y al hecho de que el eje es pasante, es decir, que sobresale por los dos lados, lo que permite acoplar el encoder fácilmente mediante un acoplamiento elástico.

Este motor se ha escogido siguiendo los criterios ya mencionados entre cinco motores paso a paso recuperados de distintos equipos estropeados de los que se han extraído distintos componentes. En concreto, este motor se ha obtenido de una impresora antigua.

Este motor incluía originalmente una chapa en su parte frontal para la sujeción del mismo en la posición deseada dentro de la impresora. Para este proyecto se ha retirado esa chapa, ya que para esta aplicación cambia el amarre del motor, y se ha sustituido por otra en forma de "L" que permite sujetarlo a una base de madera además de actuar como disipador para el motor, ya que este genera una gran cantidad de calor tras un tiempo de trabajo continuado. El plano de esta nueva chapa se sujeción se encuentra en los anexos del proyecto.

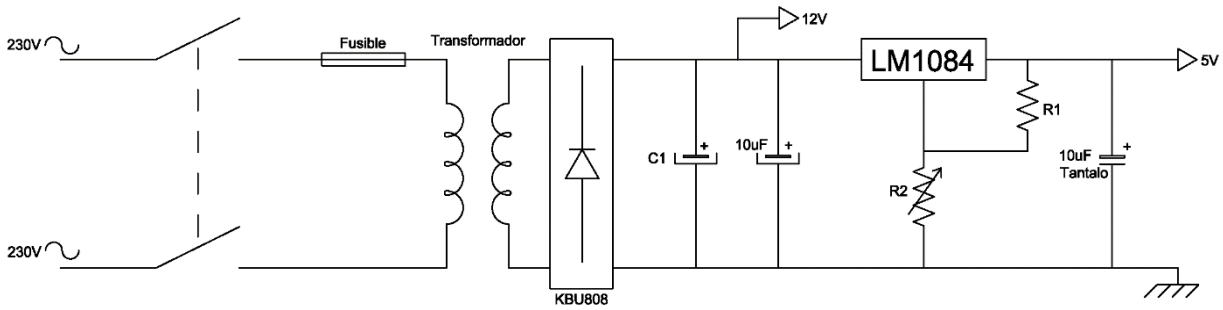
4.2.8.- Fuente de alimentación:

Para este proyecto se ha diseñado una fuente de alimentación con dos salidas a distintas tensiones, 5 y 12 voltios. Para esto se ha partido de una tensión alterna monofásica de 220 voltios y se ha reducido a unos 13 voltios mediante un transformador. Posteriormente se ha rectificado esta tensión alterna mediante un puente de cuatro diodos y se ha filtrado con un condensador electrolítico con el fin de conseguir una tensión casi continua. Por último se ha utilizado un regulador lineal de tensión variable para obtener la salida de 5 voltios, la de 12 voltios se obtiene directamente a la salida del filtrado mediante condensador, ya que esta tensión no tiene porqué ser perfectamente continua.

Aunque existen reguladores de tensión específicos para una tensión de salida de 5 voltios, se ha optado por la opción de un regulador de tensión variable porque estos son capaces de entregar más corriente que los primeros, ya que los reguladores de tensión fija no han variado prácticamente desde que se desarrollaron, mientras que los reguladores variables están construidos con una tecnología más moderna. El hecho de que pueda proporcionar más corriente, también hace necesario un disipador que elimine el calor generado por el componente para evitar así la destrucción del mismo.

Para un correcto funcionamiento de esta fuente, se han añadido unos condensadores de desacoplo a la entrada y salida del regulador variable de tensión. Esto se hace para añadir una baja impedancia a los caminos de retorno de los transitorios de conmutación desde el origen hasta el plano de masa. Esto es importante porque las fuentes de alimentación no pueden suministrar corriente a una velocidad tan rápida como lo requieren los sistemas conectados a ella, provocando una caída de tensión indeseada. Para eliminar el efecto de los cables o pistas de la placa de circuito impreso es muy importante colocar estos condensadores lo más cerca posible de los pines de entrada del regulador de tensión.

El valor de los componentes de la fuente se calcula en el apartado 6.5.



Esquema de la fuente de alimentación

5.- Protocolo SPI:

5.1.- Definición:

El bus SPI (Seria Peripheral Interface) es un enlace síncrono de datos entre varios dispositivos, que fue nombrado así por Motorola. Este sistema de transmisión de datos solamente puede usarse en distancias cortas, como en sistemas integrados, tarjetas SD o sensores.

Los dispositivos se comunican en modo maestro – esclavo, donde el maestro es el encargado de iniciar la comunicación entre todos los dispositivos conectados entre sí, y el esclavo puede tanto enviar como recibir información, pero siempre bajo las órdenes del maestro. El dispositivo maestro debe ser un microprocesador, pero el esclavo no tiene por qué serlo. Este sistema de comunicaciones permite la conexión de varios dispositivos esclavo a un mismo maestro, teniendo cada uno de ellos su propia línea de selección de dispositivo. El Arduino due permite la conexión de hasta tres esclavos, pero en este proyecto solo se utiliza uno.

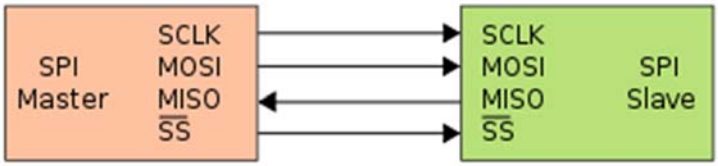
Este tipo de comunicación se realiza mediante cuatro líneas de transmisión de datos (lo que da nombre a esta comunicación como "four-wire"), que son las siguientes:

- **SCLK (Serial Clock):** Es el reloj que marca la velocidad de transmisión de la información, ya que con cada flanco de este reloj se produce un envío o recepción. Este reloj debe ser generado por el master. El uso de este reloj hace que este sistema de envío y recepción de datos sea de tipo síncrono.

- **MISO (Master Input, Slave Output):** Como su propio nombre indica, esta línea es la entrada del maestro y la salida del esclavo, por lo que se usa para transmitir la información del esclavo al maestro.

- **MOSI (Master Output, Slave Input):** Esta línea es la contraria a la MISO, ya que es la entrada del esclavo y la salida del maestro. Se usa para transmitir desde el maestro hasta el esclavo.

- **SS (Slave Select):** Es una salida del master, y es el encargado de seleccionar en cada momento el dispositivo esclavo con el que se desea trabajar. Esta selección se realiza mediante un nivel bajo a la salida, ya que si el estado es alto, este esclavo se ignora. Esto permite conectar varios dispositivos compartiendo las líneas MISO, MOSI y SCLK.

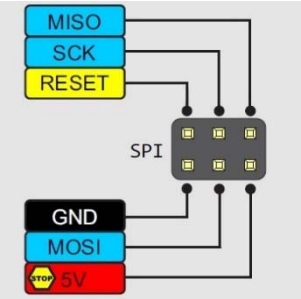


Bus SPI con un maestro y un esclavo

En el caso del Arduino due, estos cuatro pines se encuentran en un conector llamado SPI situado en la parte central de la placa. En este conector también se encuentran un pin de 5v y otro de reset.

Este conector es común a todas las placas de Arduino, lo que posibilita el diseño de un elemento externo que haga uso de este conector y que podría ser utilizado en cualquier placa de Arduino.

En el caso de la placa de Arduino due, es importante no confundir este conector llamado SPI con otro de igual aspecto pero con otra configuración de pines llamado ICSP, ya que aunque se trata de otro puerto para comunicación SPI, no es un estándar presente en todas las placas de Arduino.



Conector SPI placa Arduino

5.2.- Operación:

Antes de comenzar la transmisión de datos, hay que configurar varios parámetros de la comunicación. Lo primero que debe hacerse es configurar el reloj serie del sistema, cuya frecuencia no debe superar nunca la frecuencia máxima del microprocesador usado como maestro ni la velocidad máxima del esclavo (normalmente bastante inferior a la del maestro).

Básicamente existen cuatro tipos de transmisión de datos atendiendo a la polaridad y la fase del reloj (SCLK):

- La fase del reloj (CPHA) determina si los datos se envían o reciben de un dispositivo a otro con los flancos ascendentes o descendentes de la señal SCLK generada por el equipo maestro.

- La polaridad del reloj (CPOL) indica si este se encuentra en reposo en su estado alto o en su estado bajo.

La combinación de estos dos elementos da lugar a los siguientes cuatro modos de operación:

Modo	Polaridad del reloj (CPOL)	Fase del reloj (CPHA)
SPI_MODE0	0	0
SPI_MODE1	0	1
SPI_MODE2	1	0
SPI_MODE3	1	1

En el Arduino due esto se realiza mediante la instrucción “SPI.setClockDivider (SS, divisor)”, donde SS es el pin al que se ha conectado la selección del esclavo (pueden ser los pines 4, 10 o 52), y el divisor es el número por el que se divide la frecuencia del microprocesador del Arduino (en el due es 84 MHz), dando lugar a las siguientes posibles frecuencias, aunque el divisor puede ser cualquier número entre 1 y 255:

Divisor	Frecuencia SCLK
84	1 MHz
42	2 MHz
21	4 MHz

El siguiente paso es seleccionar el dispositivo al que conectarse, para ello debemos poner a cero la salida a la que se ha conectado el pin SS del dispositivo esclavo. En Arduino esto se hace de forma automática al dar la orden de inicio de transmisión. Para que esto se pueda realizar, es necesario haber indicado anteriormente a que pin se ha conectado la línea SS mediante la instrucción "SPI.begin (SS)" que debe introducirse al inicio del programa.

Por último, y solo en caso de que el esclavo lo requiera, hay que configurar el o los dispositivos esclavo. Esto se hace enviando uno o varios números a cada esclavo mediante la instrucción "PI.transfer (SS, número)".

5.3.- Ventajas y desventajas:

- Ventajas:

  • En el modo por defecto esta comunicación es de tipo full dúplex, es decir, se puede enviar y recibir información al mismo tiempo.

  • Proporciona una señal de buena calidad y a una gran velocidad.

  • El procesamiento de datos es mayor que en el protocolo I2C.

  • Configuración de hardware muy sencilla, ya que necesita niveles más bajos que otros protocolos similares. Además los dispositivos esclavos usan el reloj del maestro, evitando usar osciladores de precisión.

  • Usa solo cuatro líneas para la transmisión de datos.

  • Las señales son unipolares, lo que permite un fácil aislamiento galvánico.

- Desventajas:

  • Requiere mayor número de cables que otros tipos de comunicaciones.

  • No hay control de flujo de la información desde el esclavo.

  • No hay reconocimiento del esclavo, por lo que el master podría transmitir aunque no haya ningún esclavo conectado.

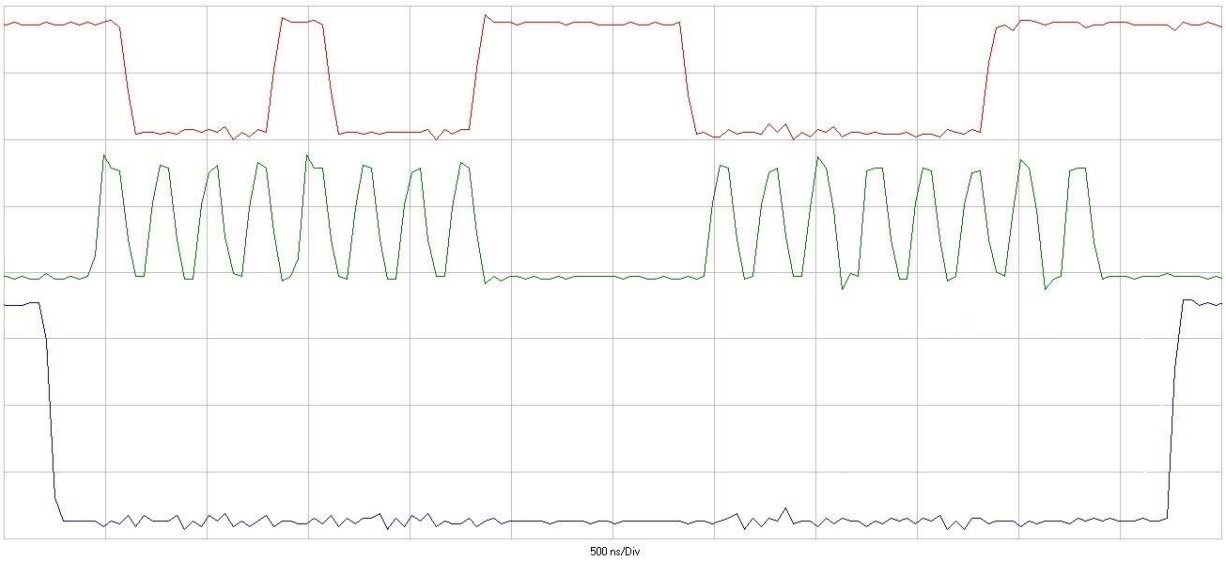
  • Solo puede haber un dispositivo maestro.

  • No existe un protocolo de información sobre errores.

- Solo es válido en distancias cortas.
- No soporta la conexión en caliente, es decir, no se pueden conectar esclavos al maestro con los dispositivos encendidos.

5.4.- Ejemplo de transmisión de datos:

A continuación se muestra una captura del osciloscopio durante la transmisión de configuración del dispositivo esclavo utilizado en el proyecto (LS7366R), donde pueden verse las señales de SS, SCLK y MOSI (envío desde el maestro hasta el esclavo):

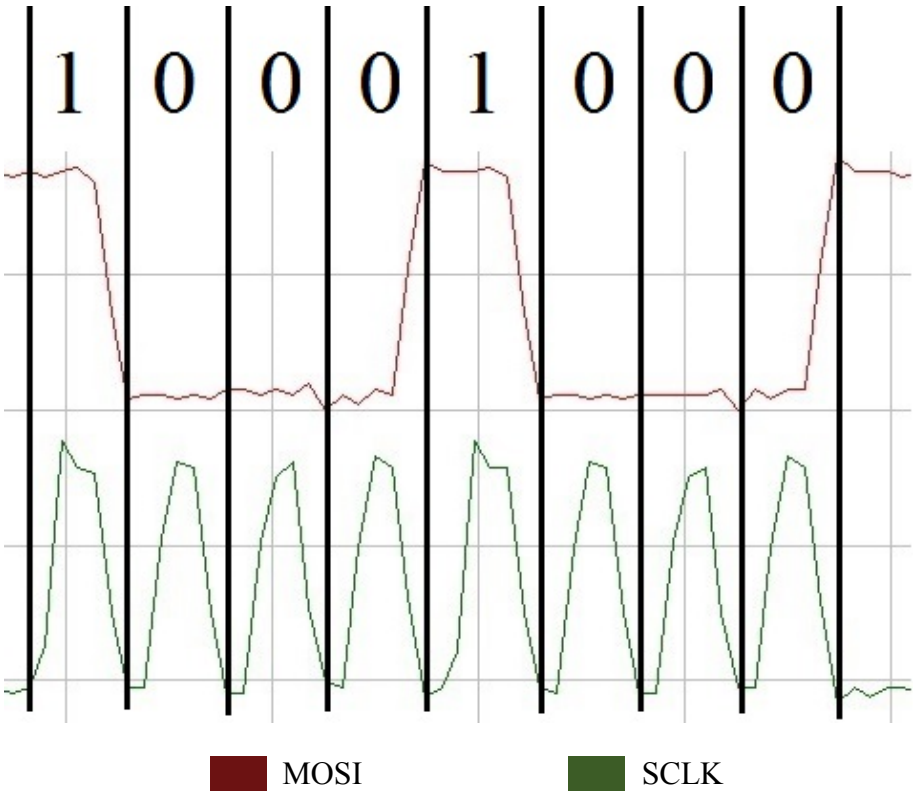


MOSI SCLK SS

Lo primero que se observa en la imagen superior es que solo hay transmisión de datos mientras la señal SS está a cero, y por lo tanto el esclavo está seleccionado para comunicarse con él.

La forma de onda verde es el reloj de transmisión de datos, así que cada vez que hay un flanco ascendente del esta señal, se recoge el dato de la señal MOSI durante ocho periodos para formar un byte.

A continuación se analiza el primer bloque de datos de esta transmisión (ocho primeros flancos del reloj):



Esta es la gráfica que se ha obtenido al ejecutar la instrucción "SPI.transfer (SS, 0x88, SPI_CONTINUE)", y como se ve en la imagen, al unir los ocho estados de la señal MOSI, se obtiene el número binario 10001000, que en formato hexadecimal corresponde al número 88, que es el que se ha indicado en la instrucción de transmisión de datos (0x indica que el número a continuación está en formato hexadecimal).

Si se desean realizar varias transmisiones consecutivas, hay que incluir "SPI_CONTINUE" después del número que se desea enviar al esclavo, de forma que no se deshabilita el esclavo hasta que se envíe un dato sin la instrucción de continuar con el envío.

El siguiente envío de información que aparece en la primera imagen corresponde a la instrucción "SPI.transfer (CS, 0x03)", de forma que se envía el número binario 00000011, que en hexadecimal es 3.

En el caso de que sea el esclavo el que envíe la información al maestro, las gráficas serian iguales cambiando el MOSI por el MISO, ya que el flujo de información ha cambiado de sentido.

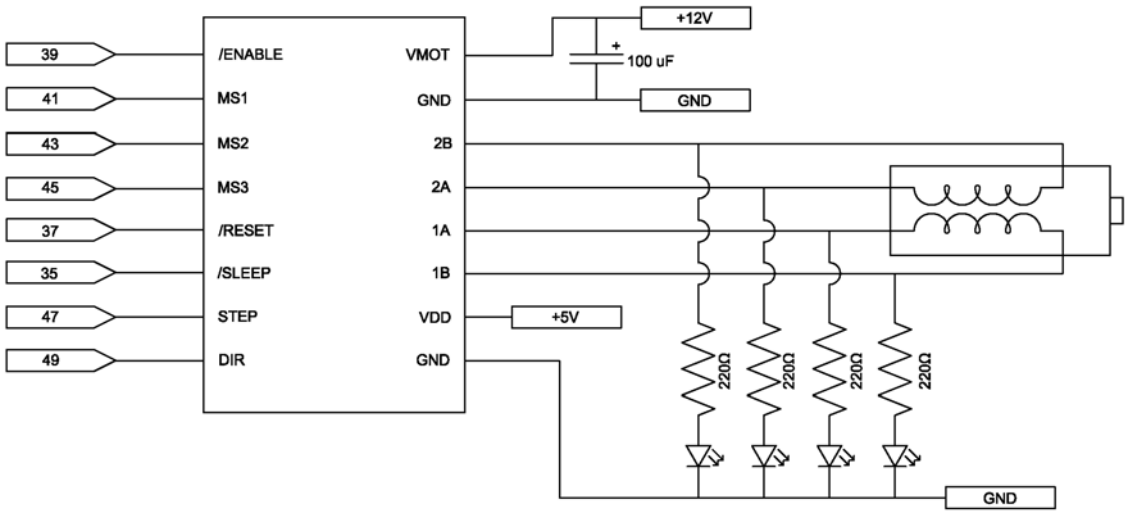
Por último, también podemos observar que en el momento en el que se acaba la transmisión de datos, el valor de la línea SS se pone a uno, deshabilitando así el dispositivo esclavo. En el caso de que en la primera instrucción de transferencia de datos no se hubiese incluido "SPI_CONTINUE", habría un cambio de estado en la señal SS de cero a uno al acabar la primera transmisión y de uno a cero al iniciar la segunda. Esta funcionalidad de dejar habilitado el esclavo para varias transmisiones consecutivas solo está disponible en el Arduino due, ya que este tiene alguna funcionalidad más en lo que refiere al SPI que el resto de placas Arduino.

6.- Conexión de los componentes:

En este apartado se va a abordar el conexionado entre los distintos elementos que conforman este proyecto y que ya se han descrito en apartados anteriores, así como los cálculos necesarios para tomar la decisión del valor a utilizar de algunos componentes tales como resistencias o condensadores.

En este punto también se van a describir los conectores que se han utilizado para conectar los elementos externos a la placa de circuito impreso, como por ejemplo el teclado o el display LCD, ya que dichos componentes no pueden conectarse directamente a la PCB.

6.1.- Controlador del motor paso a paso:



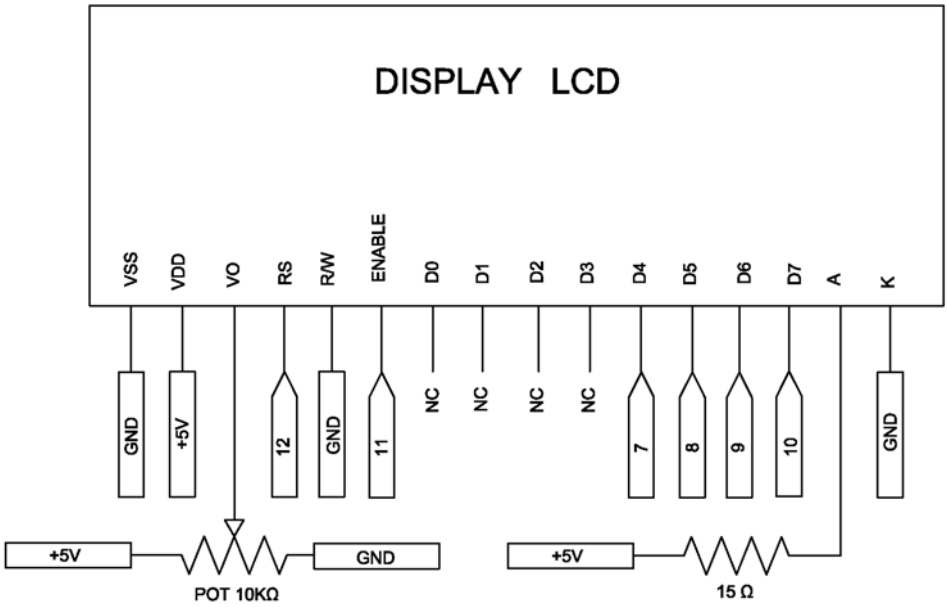
Conexión del controlador del motor

Como puede verse en la imagen, todas las entradas de control del controlador (los pines de la parte izquierda) se conectan a los pines de salida impares de la placa Arduino desde la 37 hasta la 49.

Los pines de la parte derecha del controlador corresponden a alimentaciones y conexión con el motor. El pin VDD y el GND situado justo debajo se corresponden con la alimentación del circuito integrado A4988 de Allegro, que se alimenta a 5 voltios de corriente continua. Los otros dos pines de alimentación son VMOT Y el GND de debajo de él, Estos terminales son la alimentación del motor paso a paso, que el caso del utilizado en este proyecto debe alimentarse a 12 voltios. El condensador de 100uF colocado entre VMOT y GND es un condensador de filtrado que el fabricante recomienda colocar para un correcto funcionamiento del equipo.

Por último, las salidas 1A, 1B, 2A y 2B se conectan directamente a las bobinas del motor paso a paso.

6.2.- Display LCD:



Conexión del display de cristal liquido

En la imagen superior puede verse como se ha realizado la conexión del LCD.

La alimentación de la pantalla se realiza a 5 voltios entre los pines VSS y VDD situados en la parte izquierda de la imagen. A continuación se encuentra el pin VO, que se conecta a un potenciómetro de 10KΩ (indicado por el fabricante) conectado entre 5 voltios y masa para poder regular el contraste del LCD.

La salida 12 de la placa Arduino se conecta a la entrada RS, que es la que selecciona el registro del LCD que se va a utilizar, ya que el display tiene dos registros de 8 bits, uno de instrucciones (0) y otro de datos (1). Al seleccionar el de instrucciones, se indica al LCD que lo que hay en el bus de datos (pines desde 7 a 14) es una instrucción, y al seleccionar el registro de datos, se indica al LCD que lo que hay en el bus de datos es un carácter cuyo destino es la DDRAM o la CGRAM que son las dos zonas de memoria del LCD.

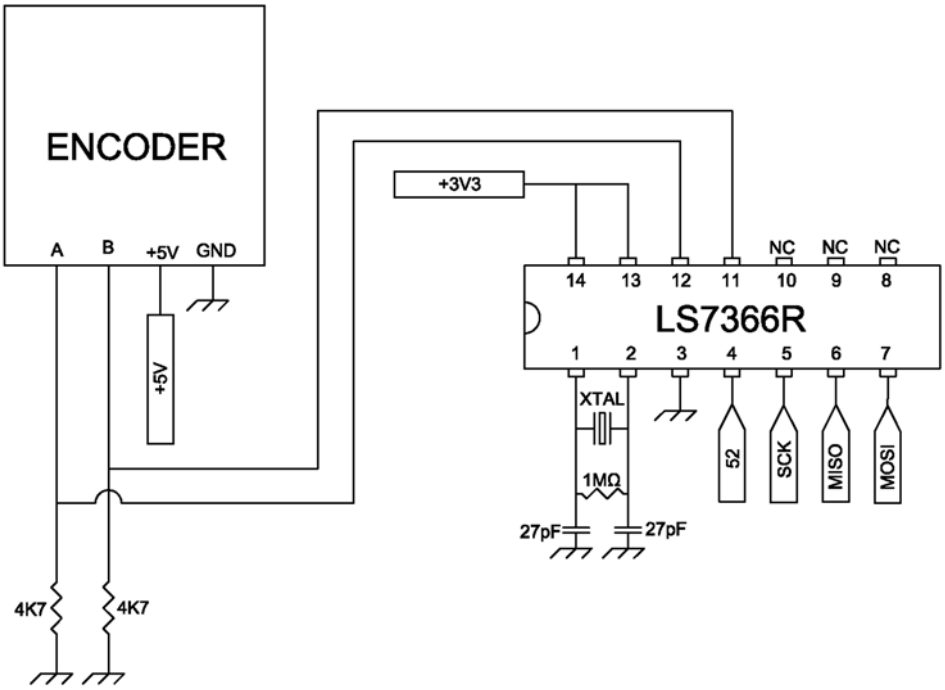
Como en este caso siempre se va a escribir en el display y no se va a leer en ningún momento, el pin RW se conecta a masa con el fin de ahorrar una salida del Arduino y simplificar así la gestión de la pantalla. Mientras que la habilitación del LCD (pin ENABLE) se ha conectado a la salida numero11 del Arduino.

Los siguientes ocho terminales (D0 a D7) son los pines del bus de datos. Para la gestión del LCD en este proyecto, solamente se va a utilizar la mitad del bus (D4 a D7, por lo que estos terminales se conectan a las salidas 7, 8, 9 y 10 de la placa Arduino.

Por último, los terminales A y K son el ánodo y el cátodo de los diodos led encargados de la retroiluminación del display. El cátodo se conecta directamente a masa, pero el ánodo hay que conectarlos a 5 voltios a través de una resistencia de 15 ohmios que limite la corriente que va a circular por los diodos led.

$$I_{LED} = \frac{5V}{15\Omega} = 0.3333\ A \simeq 333\ mA$$

6.3.- Encoder y LS7366R:



Conexión del encoder y el integrado LS7366R

Como se puede ver en la imagen, del encoder salen únicamente cuatro hilos, señales A y B, y la alimentación. Las señales A y B se conectan a los pines 12 y 13 del integrado respectivamente, pero también se deben conectar a masa a través de sendas resistencia para asegurar el cero en las zonas del ciclo que sea necesario.

Aunque el circuito integrado LS7366R se puede alimentar a 5 voltios, en este caso se va a alimentar a 3,3 voltios entre los pines 14 (3,3 voltios) y 3 (GND), ya que las salidas son al nivel de alimentación, y la placa de Arduino no puede recibir más de 3,3 voltios a través de las entradas. El pin número 13 (habilitación de contaje) también se conecta a 3,3 voltios porque el integrado va a estar siempre habilitado, ya que como no hay más elementos conectados al bus SPI no es necesario deshabilitarlo en ningún momento.

Los pines 1 (fcko) y 2 (fcki) hay que conectarlos a un oscilador externo que genere un reloj para filtrar las señales A y B del encoder. Este oscilador puede ser generado por un reloj externo, o como en este caso por un cristal de cuarzo. El montaje necesario para hacer un oscilador mediante un cristal de cuarzo incluye una resistencia de un valor muy elevado (1MΩ) en paralelo con el cristal y dos condensadores pequeños (27pF) entre masa y los terminales del cristal de cuarzo. Este montaje viene indicado en la hoja de características del fabricante, en la que también se indica como calcular el valor de los condensadores.

$$C = 2 * (C_L - C_P) - 10pF$$

$$C_L = Capacidad\ de\ carga\ del\ cristal\ (indicada\ por\ el\ fabricante) = 20\ pF$$

$$C_P = Capacidad\ parasita\ de\ la\ PCB\ vista\ desde\ el\ cristal \simeq 1\ pF$$

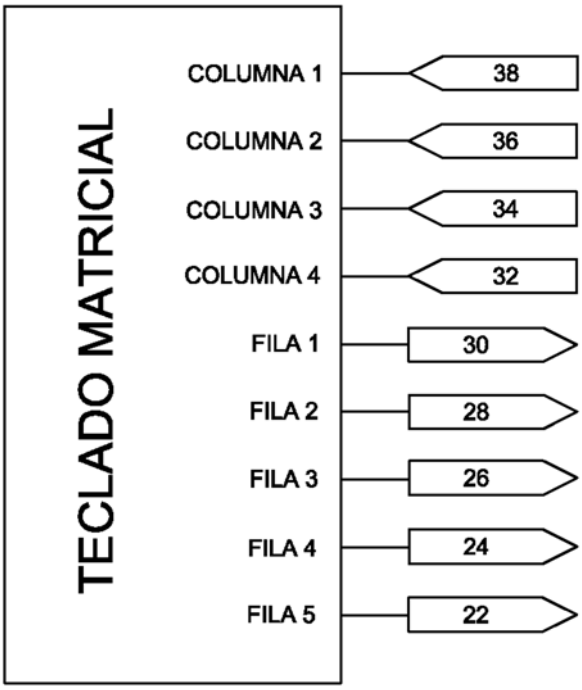
$$C = 2 * (20\,pF - 1\,pF) - 10\,pF = 28\,pF$$

$$Tomamos\ el\ valor\ comercial\ mas\ proximo\ \rightarrow C = 27\,pF$$

El pin 4 (/SS) del LS7366R es el de selección de esclavo (Slave Select). Como es una entrada negada, se selecciona con un nivel bajo (0) y se deselecciona con un nivel alto (3,3V). Este pin se conecta a la salida 52 de la placa Arduino, y su gestión corre a cargo de la librería SPI (hay que indicar en que pin se ha conectado).

Los pines 5, 6 y 7 se corresponden con las tres señales necesarias para la comunicación SPI (SCK, MISO y MOSI), que obviamente se conectan a los pines con el mismo nombre del Arduino.

6.4.- Teclado matricial:

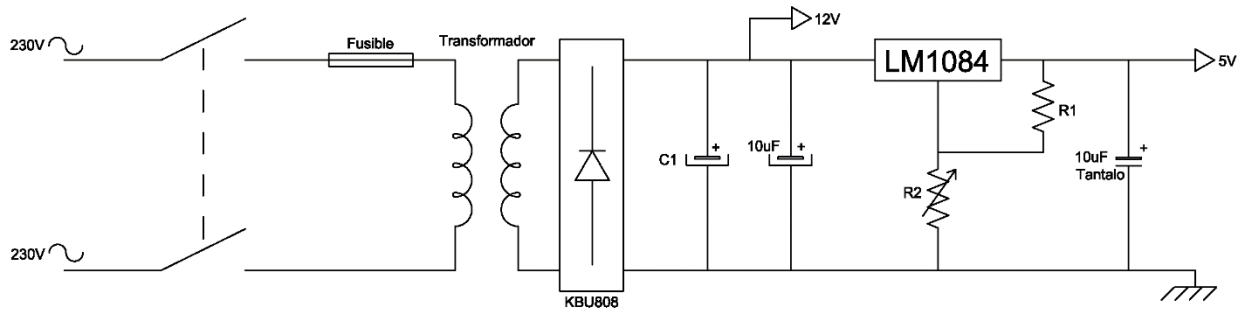


Conexión del teclado matricial

Como ya se ha explicado en apartados anteriores, un teclado matricial está formado por una matriz en las que las filas son salidas (entradas del Arduino) y las columnas son entradas /salidas del Arduino), por lo que en el caso de este proyecto, se necesitan 9 hilos, 5 entradas de la placa Arduino y 4 salidas.

Para el teclado no es necesario definir como entradas o salidas cada pin, ya que lo hace automáticamente la librería que lo gestiona al decirle en que pines se conecta cada fila o columna.

6.5.- Fuente de alimentación:



Conexión de la fuente de alimentación

En el esquema se puede ver una típica fuente de alimentación con dos salidas, una de 12 voltios y otra de 5 voltios. A la entrada se ha colocado un interruptor que se utilizara como marcha general del equipo, y después un fusible para proteger la fuente.

Después del fusible se coloca un transformador reductor, cuya tensión del primario será de 230V y la del secundario se calcula mediante la siguiente expresión:

$$V_{\sec min} = 12V$$

$$V_{sec} > \frac{V_{\sec min}}{\sqrt{2}} = \frac{12}{\sqrt{2}} = 8.48V$$

$$Tomamos\ V_{sec} = 9V$$

El transformador reduce la tensión, y el puente rectificador de cuatro diodos que se conecta al secundario del transformador hace la tensión unipolar pero aun discontinua. Para convertir esta tensión en más o menos continua, se utiliza un condensador de filtrado C1, cuyo valor se calcula a continuación:

$$I_{0\,MAX} = 0.311\ A\ (dato\ del\ fabricante\ del\ trnsformador)$$

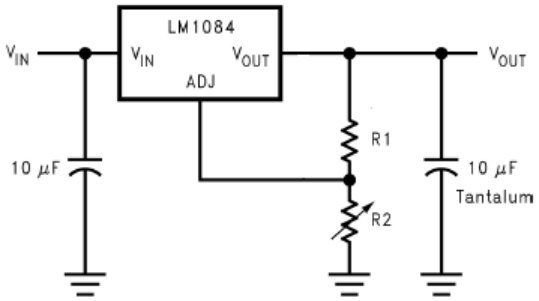
$$R_{eq} = \frac{V_{C\,MAX}}{I_{O\,MAX}} = \frac{12\ V}{0.311\ A} = 38.58\ \Omega$$

$$Tomamos\ \Delta V_{O\,MAX} = 1V$$

$$\Delta V_O > \frac{V_C * T}{2 * R_{eq} * C_1} \rightarrow C_1 > \frac{V_C}{\Delta V_O * 2 * f * R_{eq}} = \frac{12}{1 * 2 * 50 * 38.58} = 3110\ uF$$

$$C_1 > 3110\ uF \rightarrow Tomamos\ C_1 = 4700\ uF$$

A continuación se coloca el regulador de tensión variable LM1084 junto con los dos condensadores de desacoplo de 10 uF, siendo el de la entrada electrolítico de aluminio, y él se salida de tántalo. La decisión de usar un condensador de tántalo se debe a que estos condensadores son mucho más precisos que los de aluminio, presentan muy poca caída de tensión y presentan una muy baja impedancia a altas frecuencias. El valor de estos condensadores viene marcado por el fabricante en la hoja de características del elemento.



Captura del datasheet del LM1084

Para realizar el ajuste de la tensión de salida, hay que elegir el valor que se debe dar al potenciómetro R2. Para esto hay que aplicar la siguiente expresión:

$$V_O = 1.25 * \left(1 + \frac{R_2}{R_1}\right)$$

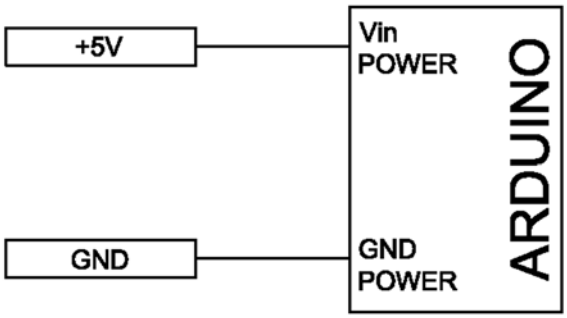
$$Si\ se\ fija\ R_1 = 100\Omega\ \rightarrow\ R_2 = (\frac{5}{1.25} - 1) * R_1$$

$$R_2 = 300\Omega$$

Para esta aplicación se podría utilizar una resistencia fija en vez de un potenciómetro, pero el potenciómetro permite realizar un ajuste mucho más preciso, ya que aunque con la expresión anterior se hace un cálculo bastante preciso, no es perfecto, y el potenciómetro permite compensar el pequeño error que se comete con la expresión matemática.

Una vez ajustada la resistencia para la tensión de salida, se obtienen 5 voltios a la salida del regulador, que servirán para alimentar a todo el equipo.

6.6.- Alimentación de la placa Arduino:

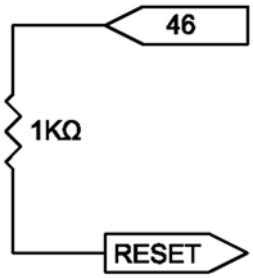


Conexión de la alimentación de la placa Arduino

La placa de Arduino integra un regulador de tensión que permite alimentarlo con tensiones de hasta 12 voltios, ya sea a través del conector redondo de 2,1 mm de diámetro o mediante los pines destinados a tal fin, denominados Vin y GND marcados como POWER.

En el caso de este proyecto se alimenta la placa de Arduino a 5 voltios. Esta tensión proviene de la fuente de alimentación descrita en el apartado anterior.

6.7.- Reset:



Conexión del pin de reset

La placa de Arduino tiene un pin destinado a realizar un reset del equipo (al igual que con el pulsador RESET). Este reset se produce con un flanco descendente.

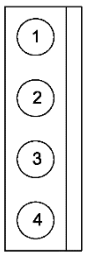
Este pin se ha conectado a la salida 46 de la propia placa de Arduino a través de una resistencia de 1KΩ para limitar la corriente de entrada y que no se destruya el microprocesador.

$$I = \frac{3.3V}{1K\Omega} = 3.3\ mA$$

6.8.- Conectores:

A continuación se detallan los conectores utilizados y se describe como se ha cableado cada uno de ellos, indicando que se ha conectado a cada pin.

6.8.1.- Conector del motor:



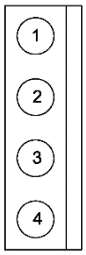
Pin 1: 1B (cable rojo)

Pin 2: 1A (cable azul)

Pin 3: 2A (cable blanco)

Pin 4: 2B (cable amarillo)

6.8.2.- Conector LED 1 motor:



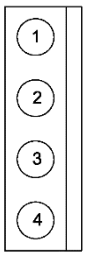
Pin 1: LED 1ª

Pin 2: LED 1B

Pin 3: GND

Pin 4: GND

6.8.3.- Conector LED 2 motor:



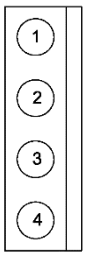
Pin 1: LED 2B

Pin 2: LED 2A

Pin 3: GND

Pin 4: GND

6.8.4.- Conector encoder:



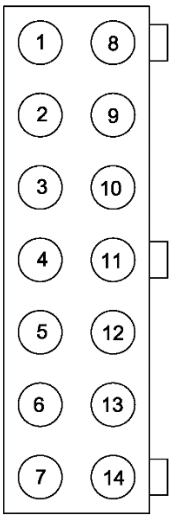
Pin 1: Señal A

Pin 2: Señal B

Pin 3: +5V

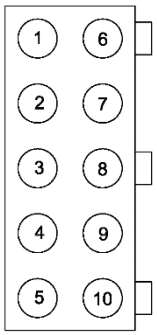
Pin 4: GND

6.8.5.- Conector LCD:



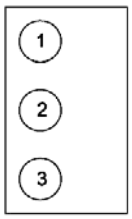
Pin 1: GND
Pin 2: GND
Pin 3: GND
Pin 4: Potenciómetro 10KΩ
Pin 5: +5V
Pin 6: RS
Pin 7: ENABLE
Pin 8: D4
Pin 9: D5
Pin 10: D6
Pin 11: D7
Pin 12: No conectado
Pin 13: No conectado
Pin 14: Resistencia 15Ω conectada a +5V

6.8.6.- Conector teclado:



Pin 1: Fila 1
Pin 2: Fila 2
Pin 3: Fila 3
Pin 4: Fila 4
Pin 5: Fila 5
Pin 6: Columna 2
Pin 7: Columna 1
Pin 8: Columna 3
Pin 9: Columna 4
Pin 10: No conectado

6.8.7.- Conector alimentación:



Pin 1: GND

Pin 2: 230V AC

Pin 3: 230V AC

7.- Programa:

7.1.- Librerías utilizadas:

En este apartado se citaran todas las librerías utilizadas y se describirá su funcionamiento así como sus principales instrucciones. Algunas de estas librerías vienen instaladas por defecto en el software de desarrollo de Arduino, mientras que otras hay que descargarlas de internet e instalarlas.

En el caso de tener que instalar una librería, simplemente hay que abrir el software de Arduino, abrir el menú programa y seleccionar importar librería. Por último se debe hacer click en añadir librería y seleccionar el archivo comprimido que se ha descargado de internet.

Para hacer uso de una librería en un programa de arduino, hay que declararla al inicio del programa mediante el símbolo almohadilla (#) seguido del nombre de la librería.

7.1.1.- Keypad:

La librería keypad permite el manejo de un teclado matricial de cualquier tamaño mediante su conexión a una placa Arduino. La versión utilizada en este proyecto es la 3.1 (versión más reciente), que como principal novedad respecto a las versiones anteriores soporta la presión simultánea de varias teclas por defecto.

Esta librería fue creada para promover la abstracción de hardware, lo que hace que la programación mediante Arduino sea mucho más simple que si no se utilizase esta librería, ya que el código resultante al utilizar la librería es mucho más corto y legible gracias a que se ocultan muchas instrucciones como la definición o lectura de los pines.

Esta librería permite conectar el teclado directamente a la placa de Arduino, ya que como se utilizan las resistencias de pullup internas, no hace falta conectar ningún elemento externo como resistencias o diodos para asegurar el cero en las entradas del Arduino.

Para utilizar esta librería lo primero que hay que hacer es definir el teclado, indicando el número de filas y columnas, el nombre de cada pulsador y los pines a los que se han conectado cada fila y columna. A continuación se muestra un ejemplo en el que se configura un teclado de 4x3, en el que se conectan las filas en los pines 5, 4, 3 y 2 y las columnas en los pines 8, 7 y 6.

```
const byte rows = 4;                    // Cuatro filas.
const byte cols = 3;                    // Tres columnas.
char keys[rows][cols] = {


// Tabla que indica el nombre de cada pulsador.
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'#','0','*'}
};
```

```
byte rowPins[rows] = {5, 4, 3, 2}; // Pines de conexión de las filas.
byte colPins[cols] = {8, 7, 6};                                    //
Pines de conexión de las columnas.
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, rows, cols );
// Instrucción para generar el teclado
```

Las funciones más importantes de esta librería son las siguientes:

- **void begin(makeKeymap(userKeymap))**: Esta instrucción inicializa el teclado interno para que sea igual al configurado por el usuario.

- **char waitForKey()**: Esta función espera hasta que se pulsa una tecla, lo que significa que el programa no hace nada hasta que no se pulse una tecla del teclado. Lo único que funcionaria mientras no se pulse ninguna tecla serían las interrupciones.

- **char getKey()**: Devuelve la tecla pulsada en el caso de que se pulse alguna. Al contrario que la instrucción anterior, esta no bloquea el programa a la espera de que se pulse una tecla.

- **KeyState getState()**: Devuelve el estado de cualquiera de las teclas. Los cuatro estados posibles son IDLE, PRESIONADA, SOLTADA o MANTENIDA.

- **boolean keyStateChanged()**: Permite saber cuándo una tecla ha cambiado de estado.

- **setHoldTime(unsigned int time)**: Configura el tiempo en milisegundos que se debe mantener la tecla pulsada hasta que el sistema la de por pulsada. Esta función puede usarse como una seguridad para impedir que se pulsen teclas por error. Esta función solo debe incluirse una vez en la zona de setup del programa.

- **setDebounceTime(unsigned int time)**: Mediante esta instrucción se establece un tiempo en milisegundos durante el cual no se atiende a la pulsación de una tecla tras haber pulsado otra. Esto se utiliza para evitar los rebotes o dobles pulsaciones involuntarias. Esta función solo debe incluirse una vez en la zona de setup del programa.

- **addEventListener(keypadEvent)**: Se lanza un evento en el momento en el que se usa el teclado. Esta función solo debe incluirse una vez en la zona de setup del programa.

Esta librería no se encuentra incluida en el software de desarrollo de Arduino, por lo que hay que descargarla desde http://playground.arduino.cc/uploads/Code/keypad.zip e instalarla siguiendo los pasos que se han indicado anteriormente.

7.1.2.- SPI:

La librería SPI permite el control del flujo de información entre dos o más dispositivos conectados entre sí a través del bus SPI. Esta librería se encuentra incorporada en el programa de desarrollo de Arduino, por lo que para usarla simplemente hay que indicarlo en la cabecera del código.

Anteriormente se ha descrito el funcionamiento del protocolo de comunicaciones SPI, y a continuación se describe tanto el funcionamiento como las instrucciones más comunes de esta librería para su correcto funcionamiento en una placa de Arduino due, ya que como se ha dicho anteriormente, las instrucciones de la librería varían entre esta placa y las demás:

- **SPI.begin(SS):** Esta instrucción inicializa el bus SPI configurando los pines SCK, MOSI y SS (indicado entre paréntesis en la instrucción) como salidas y el pin MISO como entrada. También se fuerza el estado de SCK y MOSI a bajo y el de SS a alto. Hay que destacar que una vez se ha configurado el pin SS mediante esta instrucción, no se puede usar como un pin de entrada/salida normal hasta que se deshabilite el bus SPI.

- **SPI.end(SS):** Mediante esta instrucción se deshabilita el bus SPI dejando la configuración de los pines como se encontraban anteriormente, por lo que ya se podrá volver a usar el pin SS como una entrada/salida normal, pero habrá que configurarla en el modo deseado.

- **SPI.setBitOrder(SS, orden):** Esta instrucción establece el oren de los bits en las transmisiones de envío o recepción. Esto se hace escribiendo LSBFIRST o MSBFIRST como parámetro “orden” de la instrucción. LSBFIRST significa least-significant bit first, es decir que se envía primero el bit menos significativo, y MSBFIRST significa most-significant bit first, que traducido al español indica que primero se envía el bit más significativo. Si se indica el pin SS al que se se quiere aplicar, solo se configura uno de los dispositivos, permitiendo así que cada dispositivo tenga una configuración distinta.

- **SPI.setClockDivider(SS, divisor):** Configura el divisor del reloj SPI respecto al reloj del sistema. En Arduino due el valor por defecto del divisor es 21, lo que fija la frecuencia del reloj a 4 MHz como en otras placas de Arduino, pero el valor del divisor puede tomar cualquier valor entre 1 y 255. Si se indica el pin SS, esta configuración se establece solo para el dispositivo conectado al pin indicado, lo que permite tener distinta configuración para cada dispositivo esclavo conectado al equipo maestro.

- **SPI.setDataMode(SS, modo):** Con esta instrucción se establece el modo de operación de la comunicación SPI dependiendo de la fase y polaridad del reloj. Los cuatro posibles modos de operación ya se han definido en el apartado 5.2. Si no se usa esta instrucción al inicializar la comunicación, se usa el modo 0, que el utilizado por defecto.

- **SPI.transfer(SS, valor, modo de transmisión):** Transfiere un byte por el bus SPI, esta instrucción se usa tanto para el envío como para la recepción de datos. Si se indica el pin SS, este pin cambia a estado bajo antes de que se produzca la transmisión y se fuerza a estado alto al terminarla. El parámetro "modo de transmisión" no es obligatorio pero si recomendable, e indica si la transmisión termina o no al acabar de ejecutarse la instrucción. Los posibles valores para el "modo de transmisión" son SPI_CONTINUE o SPI_LAST. Si se selecciona el primero, se mantiene el pin SS en estado bajo permitiendo el envío de otro byte, y se selecciona el segundo modo, el pin SS vuelve a estado alto después de la transmisión. Si no se indica el "modo de transmisión" deseado, por defecto se configura como SPI_LAST.

A continuación se muestra un fragmento de código en el que se usan estas instrucciones con los correspondientes comentarios para identificar la función de cada instrucción:

```
SPI.begin(4);                          // Inicializa el bus para el pin 4 como SS.
SPI.setBitOrder(4, LSBFIRST);          // Configura la transmisión del pin 4 como el bit
                                          menos significativo primero.
SPI.setClockDivider(4, 21);            // Configura el divisor del reloj del pin 4 a 21.
SPI.setDataMode(4, SPI_MODE1);         // Configura el modo de transmisión con el
                                          dispositivo conectado al pin 4 como modo 1.
SPI.transfer(4, 0xF0, SPI_CONTINUE);// Se envía un 0xF0 al elemento del pin 4 y se
                                          indica que sigue la comunicación.
SPI.transfer(4, 0×00, SPI_LAST);       // Se envía un 0x00 al elemento del pin 4 y se
                                          indica que se finaliza la comunicación.
SPI.end(4);                            // Se deshabilita el bus SPI.
```

7.1.3.- LiquidCrystal:

Esta librería permite a una placa Arduino controlar un display de cristal líquido (LCD) basado en el controlador Hitachi HD44780 o uno compatible, ya que es el controlador que se encuentra en la gran mayoría de los LCD. Esta librería puede funcionar en modo de 4 u 8 bits, es decir, usar 4 u 8 líneas de transmisión de datos).

Esta librería tiene un gran número de instrucciones que se numeran y describen a continuación, aunque en este proyecto solamente se utilizan las más básicas:

- **LiquidCrystal lcd (rs, rw, enable, d4, d5, d6, d7):** Define las conexiones del display con la placa Arduino. Los parámetros de esta instrucción son el número de pin al que se han conectado el rs, el rw, el enable y los pines de comunicación del LCD y el nombre que se asigna al display, en este caso lcd. Para configurarlo como 4 u 8 bits, simplemente hay que indicar 4 u 8 pines de transmisión de datos.

- ***lcd*.begin(cols, filas):** Inicializa la interfaz de la pantalla LCD, e indica el número de columnas (cols) y filas de la misma. Esta instrucción debe ser llamada antes de empezar a utilizar el display.

- ***lcd*.clear():** Borra todo el contenido de la pantalla LCD y sitúa el cursor en la esquina superior izquierda del display.

- **lcd.home():** Sitúa el cursor en la esquina superior izquierda del display, punto en el que se seguirá escribiendo al enviar la instrucción de representar un carácter.

- **lcd.setCursor(col, fila):** Con esta instrucción se sitúa el cursor en la posición indicada mediante los parámetros col (columna) y fila.

- **lcd.write(dato):** Se escriben los caracteres indicados mediante el parámetro "dato" en la pantalla LCD.

- **lcd.print(dato, BASE):** Imprime en código ASCII el "dato" indicado en el display. El parámetro BASE (que es opcional) indica la base en la que se escriben los números en la pantalla, este parámetro puede tomar los valores BIN para binario, DEC para decimal, OCT para octal y HEX para hexadecimal.

- **lcd.cursor():** Se muestra el cursor de forma fija en la posición en la que se encuentra. El cursor es una línea horizontal que aparecerá en la posición en la que se escribiría un nuevo carácter.

- **lcd.noCursor():** Esconde el cursor del display.

- **lcd.blink():** Muestra el cursor de forma intermitente.

- **lcd.noBlink():** Esconde el cursor que parpadea.

- **lcd.noDisplay():** Apaga la pantalla sin perder la información que se está mostrando en ella.

- **lcd.display():** Enciende la pantalla después de haber utilizado la instrucción lcd.noDisplay().

- **lcd.scrollDisplayLeft():** Desplaza el contenido del display (texto y cursor) un espacio hacia la izquierda.

- **lcd.scrollDisplayRight():** Desplaza el contenido del display (texto y cursor) un espacio hacia la derecha.

- **lcd.autoscroll():** Mediante esta instrucción se activa el desplazamiento horizontal automático del LCD, de forma que al escribir texto, los caracteres se mueven automáticamente de forma que no se sobrescriben los caracteres.

- **lcd.noAutoscroll():** Se desactiva el desplazamiento horizontal automático del LCD.

- **lcd.leftToRight():** Configura la dirección de escritura de texto de izquierda a derecha. Esta es la dirección de texto por defecto.

- **lcd.rightToLeft():** Configura la dirección de escritura de texto de derecha a izquierda.

- **lcd.createChar(num, dato):** Esta instrucción crea un carácter personalizado para su uso en el LCD. Se pueden crear hasta 8 caracteres, cuyo nombre se identifica mediante un número del 1 al 7 correspondiente al parámetro "num". Para definir el símbolo, hay que crear una tabla con 8 bytes en los que se indica el estado de cada punto del bloque de pixeles de cada carácter. A continuación se muestra un ejemplo de cómo crear un símbolo.

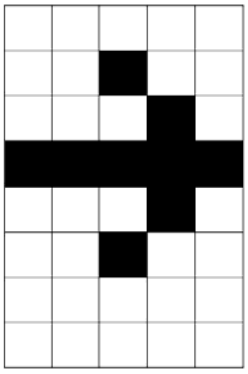
```
#include <LiquidCrystal.h>                // Se incluye la librería LiquidCrystal.
LiquidCrystal lcd(12, 14, 11, 7, 8, 9, 10);   // Conexiones del LCD.

Byte flecha_derecha[8] = {                // Carácter flecha_derecha.

   B00000,
   B00100,
   B00010,
   B11111,
   B00010,
   B00100,
   B00000,
   B00000
   };

void setup() {
lcd.createChar(0, flecha_derecha);        // Se crea el carácter flecha_derecha.
lcd.begin(20, 4);                         // Tamaño del LCD (16x4).
lcd.write(0);                             // Se escribe el carácter flecha_derecha.
}

void loop() {}
```



Carácter flecha_derecha

7.1.4.- DueTimer:

Esta librería creada específicamente para el Arduino due mejora el sistema de interrupciones nativo del código ANSI C que se utiliza para programar las placas de Arduino, ya que permite realizar interrupciones cada cierto tiempo o retrasarlas un tiempo desde que se produce un evento, acciones que no permite realizar Arduino de forma nativa (sin librerías).

Se puede definir una interrupción como la detección de un evento, y la realización de una acción en consecuencia, con la particularidad de que esto ocurre en cualquier momento, de forma asíncrona y con independencia de las líneas de código que se estén ejecutando en ese momento. Al acabar la acción a realizar en la interrupción, se vuelve al punto del código en el que se estaba en el momento en el que se produjo la interrupción.

La librería DueTimer permite llamar a una interrupción además de al detectar un cambio de estado en una entrada, cada cierto tiempo sin necesidad de que se haya producido ningún evento.

Hay que tener mucho cuidado con las interrupciones, ya que aunque en un principio puedan parecer una forma fácil de programar, si se abusa de ellas se pueden dar situaciones no deseadas, como una llamada a una interrupción mientras se está ejecutando una acción que no debe interrumpirse o durante un delay. Por esto hay que tener muy claro si es necesario usar una interrupción para llamar a una función o se puede hacer de otra manera.

En las placas de Arduino solo se pueden conectar los dispositivos que van a generar una interrupción (pulsador o sensor) a unos pines determinados, excepto en la placa de Arduino due, en la que se puede usar cualquier pin siempre que se indique el nombre del mismo.

La librería DueTimer permite la creación de hasta nueve temporizadores que ejecutan distintas interrupciones. A continuación se numeran y detallan las instrucciones que se pueden utilizar gracias a la librería DueTimer:

- **Timer.getAvailable():** Se toma el primer temporizador que este libre.

- **Timer2.attachInterrupt(función):** Se asocia la función "función" al temporizador indicado (en este caso el 2).

- **Timer2.detachInterrupt(función):** Se desasocia la función "función" al temporizador indicado (en este caso el 2).

- **Timer2. start(long microseconds = -1):** Se inicia el temporizador indicado (en este caso el 2) con un parámetro de tiempo opcional.

- **Timer2.start(tiempo):** Se inicia el temporizador indicado (en este caso el 2) cada "tiempo" microsegundos.

- **Timer2.setFrequency(frecuencia):** Se fija la frecuencia en hercios del temporizador deseado (en este caso el 2) a la indicada mediante el parámetro "frecuencia".

- **Timer2.setPeriod(periodo):** Se fija el periodo en microsegundos del temporizador deseado (en este caso el 2) al indicado mediante el parámetro "periodo".

Si se hace uso de la librería Servo.h para controlar servomotores, no se podrán utilizar los temporizadores 0, 1, 2, 3, 4 y 5, ya que los utiliza esta librería.

A continuación se incluye un pequeño programa que enciende y apaga un led cada medio segundo mediante interrupciones:

```
#include <DueTimer.h>                    // Se incluye la librería DueTimer.

int Led = 13;                            //LED conectado al pin 13.

bool ledOn = false;                      //LED apagado de inicio.
void parpadeo(){                         // Definición de parpadeo.
   ledOn = !ledOn;                       // Cambia el LED de estado.
   digitalWrite(Led, ledOn); // LED on   , LED off, LED on, …
}

void setup(){
   pinMode(Led, OUTPUT);                 // El pin del LED es una salida.

   Timer3.attachInterrupt(parpadeo);     // Se asocia parpadeo al temporizador 3.
   Timer3.start(500000);                 // Llamada al temporizador 3 cada 500ms.
}

void loop(){ }
```

7.2.- Variables utilizadas:

En el ámbito de la programación, una variable está formada por un espacio de memoria en la que se almacena un valor y un nombre que identifica a dicho valor. El nombre utilizado para referirse a este espacio de memoria es totalmente independiente del valor (numérico o alfabético) almacenado en este espacio, de forma que el nombre de la variable puede ser "*numero_de_alumnos*" (tipo alfabético) y tener un valor asociado "*26*" (tipo numérico).

Dependiendo del ámbito de influencia de cada variable se distinguen dos tipos:

- Variable global: Este tipo de variables están disponibles en todas las partes del programa, por lo que se puede acceder a su valor o modificarlo en cualquier momento y desde cualquier lugar del programa.

- Variable local: Este tipo de variables solo están disponibles en el procedimiento, función o subrutina en el que han sido declaradas, por lo que no es posible ni acceder a ellas ni modificarlas desde un procedimiento, función o subrutina que no sean en las que se han declarado.

Si se atiende al tipo de dato que se va a almacenar, se pueden distinguir distintos tipos de variables:

- Int: Números enteros con signo comprendidos entre los valores -32 768 y 32 767. Estos números se pueden expresar en decimal, hexadecimal (0x) o binario (B), pero hay que indicarlo escribiendo el identificador correspondiente antes del valor.

- Unsigned int: Números enteros sin signo comprendidos entre los valores 0 y 65 535. Estos números se pueden expresar en decimal, hexadecimal (0x) o binario (B), pero hay que indicarlo escribiendo el identificador correspondiente antes del valor.

- Long: Números enteros largos con signo comprendidos entre los valores -2 147 483 648 y 2 147 483 647. Estos números se pueden expresar en decimal, hexadecimal (0x) o binario (B), pero hay que indicarlo escribiendo el identificador correspondiente antes del valor.

- Unsigned long: Números enteros largos sin signo comprendidos entre los valores 0 y 4 294 967 295. Estos números se pueden expresar en decimal, hexadecimal (0x) o binario (B), pero hay que indicarlo escribiendo el identificador correspondiente antes del valor.

- Byte: Número sin signo de 8 bits comprendido entre los valores 0 y 255. Puede expresarse como un valor numérico o como un conjunto de 8 bits.

- Float: Números decimales con signo comprendidos entre los valores $-3.4*10^{38}$ y $3.4*10^{38}$.

- Double: Es un numero de 8 bytes que puede expresarse en decimal, hexadecimal (0x) o binario (B), pero hay que indicarlo escribiendo el identificador correspondiente antes del valor.

- Char: Caracteres. Si es un único carácter se usan comillas simples (‘A’) y si es una cadena de caracteres se usan comillas dobles (“ABC”).
  Aquí hay que diferenciar entre char y char*, ya que char es un carácter normal (por ejemplo “a”) y char* es un puntero que apunta a un carácter indicado por el usuario (por ejemplo en char* string = “HOLA”, strig[0] es igual a H y string[3] es igual a L).

- Bool: Es una variable de tipo booleano, por lo tanto solo puede tomar dos valores, 1 (true) y 0 (false). Cada una de estas variables ocupa un byte completo de memoria.

- Const: Indica que el valor es constante. Si se compara con el nombre “variable” puede resultar contradictorio, pero si se atiende a la definición se comprende el sentido de que una variable sea constante. Cualquier variable de las anteriormente mencionadas puede ser se tipo constante.

7.2.1.- Variables generales:

Nombre	Tipo	Descripción
frec	Entero Global	Indica el valor de la frecuencia introducido por el usuario mediante el teclado.
grados_girados	Entero Global	Indica los grados que ha girado realmente el motor. Este dato lo da el encoder acoplado al eje del motor.
grados_deseados	Entero Global	Indica los grados que el usuario desea que gire el motor introduciendo este dato mediante el teclado.
num_pul_act	Entero Global	Indica el número de pasos (o pulsos) que ha dado el motor paso a paso.
num_pul_des	Entero Global	Indica el número de pasos que el usuario desea que gire el motor introduciendo este dato mediante el teclado.
estado_CLK	Booleano Global	Indica el estado en el que se encuentra el reloj que marca los pasos del motor paso a paso

7.2.2.- Variables relacionadas con el LCD:

Nombre	Tipo	Descripción
grupo_msg	Entero constante Global	Indica el número de grupos de mensajes existente, como hay 3 pantallas en las que mostrar mensajes, esta variable vale 3.
mensaje	Entero constante Global	Indica el número de mensajes que hay en cada grupo, aunque en realidad toma el valor del grupo con más mensajes y se dejan mensajes vacíos en los grupos que sobran huecos. Su valor es 6.
pantalla	Entero Global	Indica el número de pantalla que se está mostrando en cada momento. Inicialmente esta variable toma el valor cero, ya que es la primera pantalla a mostrar.
opciones	Carácter Global	Es una matriz formada por (grupo_msg) filas y (mensaje) columnas en la que está el nombre de las distintas opciones a elegir.
introduccion_datos	Carácter Global	Es una matriz de una sola fila donde están los títulos de las pantallas con introducción de datos numéricos mediante el teclado.

pos_flecha	Entero Global	Indica la fila en la que se escribe la flecha que aparece en el LCD. Inicialmente esta variable toma el valor -1 ya que no está dentro del LCD.
flecha_derecha	Byte Global	Es la variable que define el símbolo "flecha derecha" mediante los pixeles que deben estar encendidos o apagados en el LCD.
símbolo_grados	Byte Global	Es la variable que define el símbolo "símbolo grados" mediante los pixeles que deben estar encendidos o apagados en el LCD.
nombre_tipo_paso	Carácter Global	Indica el nombre del tipo de paso que se ha seleccionado para poder escribirlo en el LCD. Para esto no se puede usar la variable opciones porque sus nombres son demasiado largos y no caben en el LCD.
nombre direccion	Carácter Global	Indica el nombre del sentido de giro que se ha seleccionado para poder escribirlo en el LCD. Para esto no se puede usar la variable opciones porque sus nombres son demasiado largos y no caben en el LCD.

7.2.3.- Variables relacionadas con el teclado:

Nombre	Tipo	Descripción
ROWS	Entero constante Global	Indica el número de filas que tiene el teclado matricial. En el caso de este proyecto vale 5.
COLS	Entero constante Global	Indica el número de columnas que tiene el teclado matricial. En el caso de este proyecto vale 4.
keys	Carácter Global	Es una matriz que contiene el carácter asociado a cada tecla del teclado matricial.
tecla	Carácter Global	Variable en la que se almacena el carácter de la tecla que ha sido pulsada. Inicialmente vale 0 porque no hay ninguna tecla pulsada.
valor_tec	Entero Global	Variable que indica el valor numérico final introducido mediante los números del teclado.
valor_1, valor_2 y valor_3	Entero Global	Variables que almacenan el número pulsado en el teclado para las centenas, decenas y unidades para luego agruparlos en la variable valor_tec.
tecla_pulsada	Entero Global	Variable que almacena el valor (no el carácter) asociado a las teclas numéricas del teclado.
num_puls	Entero Global	Variable que indica el número de pulsaciones que ha habido en el teclado para saber dónde almacenar el valor de la tecla pulsada.

7.2.4.- Variables relacionadas con el integrado LS7366R:

Nombre	Tipo	Descripción
cuentaencoder	Largo con signo Local	En esta variable se almacena el valor del contaje de número de pulsos el encoder girados.
valor_cont	Largo Global	Es el valor que devuelve la subrutina de lectura del encoder (lee_encoder) e indica en número de pulsos leídos por el encoder.
cont_1, cont_2, cont_3 y cont_4	Entero sin signo Local	Variables en las que se almacena el contaje parcial del encoder antes de unirlos en la variable valor_cont.

7.2.5.- Variables relacionadas con el controlador del motor:

Nombre	Tipo	Descripción
estado_enable	Entero Global	Indica el estado del pin enable del controlador del motor paso a paso. 0 (LOW): Habilitado. 1 (HIGH): Deshabilitado.
sentido_giro	Entero Global	Indica el estado del pin dir (sentido de giro) del controlador del motor paso a paso. 0 (LOW): Sentido antihorario. 1 (HIGH): Sentido horario.
estado_reset	Entero Global	Indica el estado del pin reset del controlador del motor paso a paso. 0 (LOW): Paro. 1 (HIGH): Marcha.
modo_trabajo	Entero Global	Indica el modo de trabajo seleccionado por el usuario. Inicialmente vale -1 porque no hay ningún modo de trabajo seleccionado. 1: Lazo cerrado. 2: Lazo abierto. 3: Paso único.
tipo_paso	Entero Global	Indica el tipo de paso seleccionado por el usuario. Inicialmente vale -1 porque no hay ningún tipo de paso seleccionado. 1: Paso completo. 2: Medio paso. 3: Un cuarto de paso. 4: Un octavo de paso. 5: Un dieciseisavo de paso.

7.3.- Entradas y salidas utilizadas:

A continuación se detallan las entradas y salidas del Arduino utilizadas en el proyecto y su función.

SALIDAS		
Pin Arduino	**Pin dispositivo**	**Dispositivo**
7	D4	Display LCD
8	D5	Display LCD
9	D6	Display LCD
10	D7	Display LCD
11	ENABLE	Display LCD
12	R/S	Display LCD
32	Columna 1	Teclado
34	Columna 2	Teclado
36	Columna 3	Teclado
37	/RESET	Controlador del motor paso a paso
38	Columna 4	Teclado
39	/ENABLE	Controlador del motor paso a paso
41	MS1	Controlador del motor paso a paso
43	MS2	Controlador del motor paso a paso
45	MS3	Controlador del motor paso a paso
46	RESET	Arduino
47	STEP	Controlador del motor paso a paso
49	DIR	Controlador del motor paso a paso
52	/SS	Circuito integrado LS7366R
SCK	SCK	Circuito integrado LS7366R
MOSI	MISO	Circuito integrado LS7366R

ENTRADAS		
Pin Arduino	**Pin dispositivo**	**Dispositivo**
22	Fila 5	Teclado
24	Fila 4	Teclado
26	Fila 3	Teclado
28	Fila 2	Teclado
30	Fila 1	Teclado
MISO	MOSI	Circuito integrado LS7366R

POTENCIA		
Pin Arduino	**Pin dispositivo**	**Dispositivo**
Vin	+5V	Fuente de alimentación
GND	GND	Masa de todos los dispositivos
3V3	VDD	Circuito integrado LS7366R

7.4.- Subrutinas:

Una subrutina es un segmento de código separado del bloque principal y que puede ser llamado en cualquier momento, y se usa para ejecutar una acción que se repite varias veces durante la ejecución del programa principal, evitando así tener que escribir el código correspondiente a la subrutina varias veces.

Al llamar a una subrutina ser llamada dentro de un programa, hace que el código principal se detenga y se dirija a ejecutar el código de la subrutina, y al terminar se sigue ejecutando el código principal desde donde se había detenido al llamar a la subrutina.

El programa que se ha escrito para este proyecto, está dividido en varias subrutinas y un programa principal desde el que se llama a las distintas subrutinas dependiendo de la tecla que se pulse en el teclado así como del estado de otras variables.

De esta forma se consigue que el programa sea mucho más ordenado, fluido y rápido, ya que solo hay que leer las condiciones que hacen que una subrutina se ejecute o no, pero de no ser necesario ejecutarla, no se lee la subrutina, ahorrando así tiempo de ejecución y evitando que se pierdan "cosas" que ocurren mientras se está leyendo otra parte del programa.

Estas subrutinas se definen entre el programa principal y la zona de setup, para ello hay que indicar el tipo de subrutina que es, el nombre que se le quiere dar y las variables que va a utilizar la subrutina, estas variables se declaran automáticamente al declarar la subrutina, y por lo tanto son variables locales que solo se pueden utilizar en esta subrutina.A continuación se muestra un ejemplo de definición de subrutina.

```
void suma (num_1, num_2) {          // Se define el nombre de la subrutina
  resultado = num_1 + num_2         // Programa de la subrutina
  }                                 // Fin de subrutina
```

Una vez que se ha definido una subrutina, puede llamarse desde cualquier parte del programa (zona de setup, zona de declaración de variables, programa principal o incluso otra subrutina). Para realizar la llamada a una subrutina simplemente hay que escribir el nombre de la misma y el valor de las variable indicadas en la declaración de la subrutina. A continuación se indica cómo se realiza el llamamiento a la subrutina anteriormente declarada.

```
suma (5, 2);
```

Los valores introducidos entre paréntesis al llamar a la subrutina "suma" hacen referencia al valor que se desea asignar a las variables indicadas entre paréntesis en la declaración de la subrutina, por lo que num_1 toma el valor 5 y num_2 toma el valor 2, y por tanto, la variable "resultado" pasa a valer 7. Si se quiere poder utilizar la variable resultado fuera de la subrutina "suma", la variable ha de ser definida como global fuera de la subrutina.

A continuación se procede a detallar las subrutinas que se han utilizado en el programa de este proyecto.

7.4.1.- Giro del motor:

Esta es la subrutina encargada de hacer el motor paso a paso, y su funcionamiento se basa en poner a uno y a cero la salida del Arduino conectada al pin STEP del controlador del motor, por lo que cada vez que se ejecuta la subrutina, el motor da un paso.

El momento en el que se para el motor se decide según el modo de trabajo que se haya seleccionado. Si se selecciona lazo abierto, el motor parará cuando se alcance el número de pasos deseado, y si se selecciona lazo cerrado, el motor parará cuando se alcance el número de grados deseados.

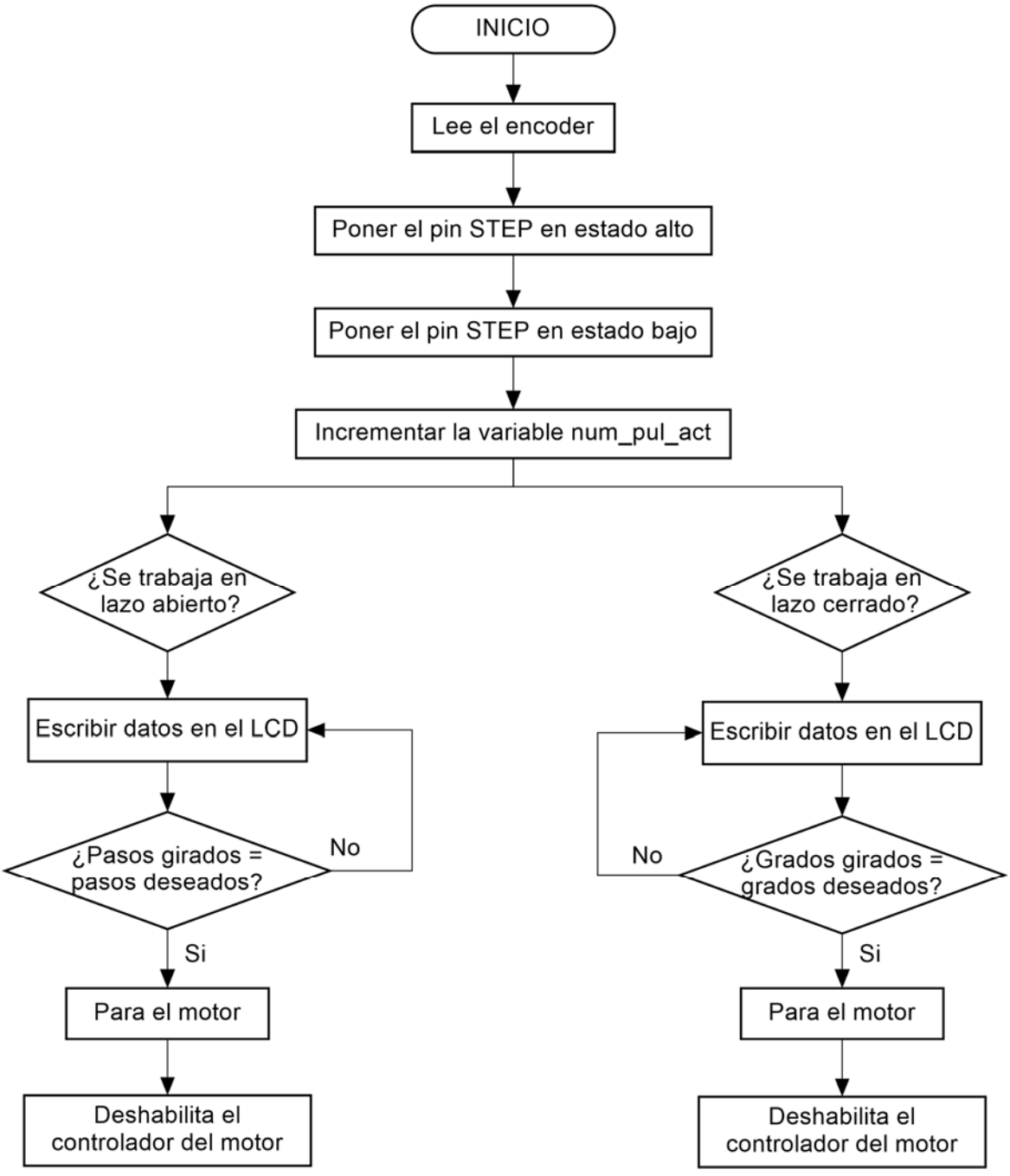


Diagrama de flujo de la subrutina giro_motor

7.4.2.- Inicialización del integrado LS7366R:

Mediante esta subrutina se inicializa el circuito integrado LS7366R encargado de realizar el contaje del encoder. Para esto hay que comunicarse con el mediante el bus SPI explicado anteriormente (apartado 5) y enviar los bytes necesarios para configurar los distintos parámetros existentes con los valores necesarios.

En el caso de este proyecto, se ha enviado al registro MDR0 un 3 (en binario 00000011), lo que se traduce en las siguientes selecciones:

- B7 = 0: Divisor del reloj igual a 1.
- B6 = 0: Index asíncrono.
- B5 y B4 = 00: Index deshabilitado.
- B3 y B2 = 00: Modo de contaje continuo.
- B1 y B0 = 11: Cuatro contajes por ciclo (señales A y B con flanco ascendente y descendente).

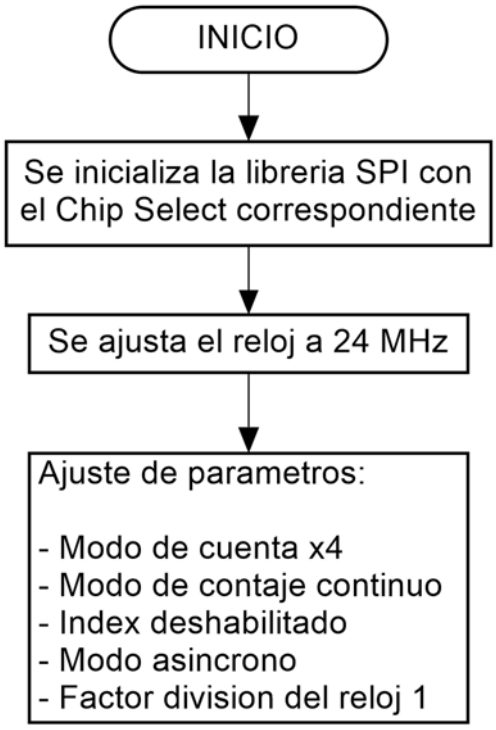


Diagrama de flujo de la subrutina inicializa

7.4.3.- Lectura del encoder:

Mediante esta subrutina se realiza la lectura del encoder. Este proceso conlleva leer el registro de contaje del circuito integrado LS7366R en cuatro veces, ya que el valor del contaje es muy grande y se divide en cuatro bytes. Posteriormente hay que unir los cuatro valores para formar uno único que se almacenara en la variable valor_cont.

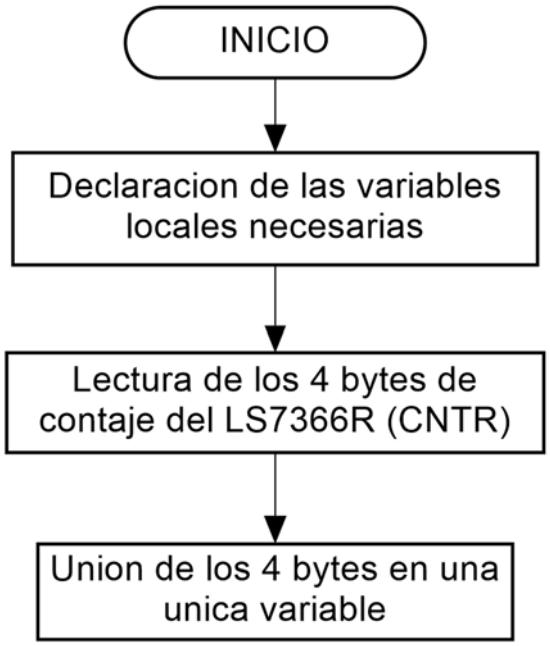


Diagrama de flujo de la subrutina lee_encoder

Para unir los 4 bytes(cont_1, cont_2, cont_3 y cont_4) con la lectura del contaje del encoder hay que usar una variable de tipo long, en este caso usaremos una llamada "valor_cont", e ir moviendo hacia la izquierda su valor para añadir datos por la derecha. A continuación se representa esta acción gráficamente para aclarar cómo funciona:

valor_cont = 00000000 00000000 00000000 00000000
cont_1 = 00011101
cont_2 = 11000101
cont_3 = 00111001
cont_4 = 11111000

valor_cont =	00000000	00000000	00000000	00000000 ←	cont_1 = 00011101
valor_cont =	00000000	00000000	00000000 ←	00011101	
valor_cont =	00000000	00000000	00011101	00000000 ←	cont_2 = 11000101
valor_cont =	00000000	00000000 ←	00011101 ←	11000101	
valor_cont =	00000000	00011101	11000101	00000000 ←	cont_3 = 00111001
valor_cont =	00000000 ←	00011101 ←	11000101 ←	00111001	
valor_cont =	00011101	11000101	00111001	00000000 ←	cont_4 = 11111000
valor_cont =	00011101	11000101	00111001	11111000	

7.4.4.- Borrado de la cuenta del encoder:

Mediante esta subrutina se borra el registro del contaje del encoder para poder así empezar un nuevo ensayo.

Para poder realizar el borrado del contaje es necesario usar un registro intermedio llamado DTR, que puede ser de 8, 16, 24 o 32 bits (1, 2, 3 o 4 bytes), en el que hay que cargar 4 bytes, todos con valor cero. Una vez el registro DTR vale cero, hay que transferir este valor al registro de contaje CNTR.

Para asegurarse de que entre ambas transmisiones no se pierde información, hay que crear un pequeño retardo de 100 microsegundos entre transmisión y transmisión.

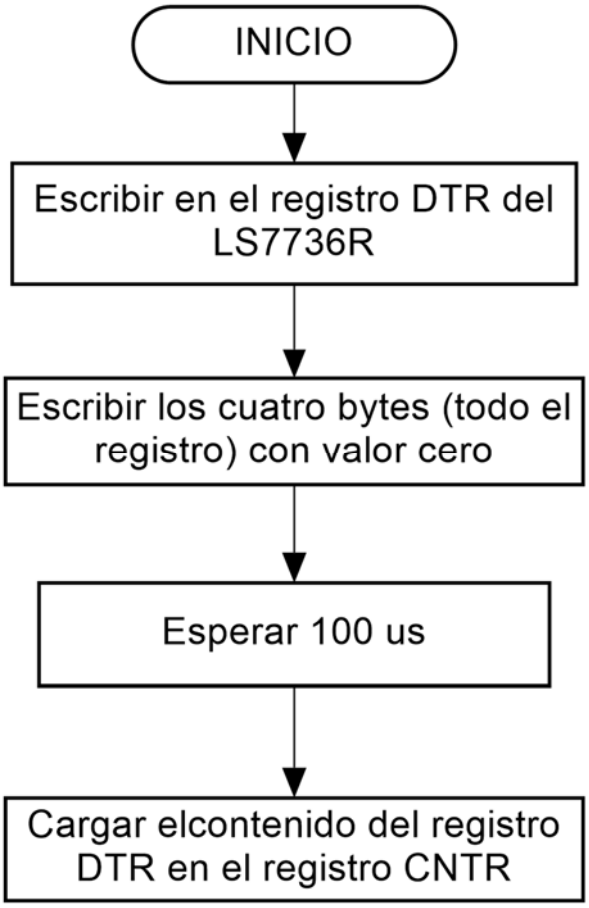


Diagrama de flujo de la subrutina borracuenta

7.4.5.- Movimiento de mensaje y flecha hacia abajo:

Esta subrutina es la encargada de desplazar tanto los mensajes como la flecha que indica la opción seleccionada hacia abajo en el LCD.

Diagrama de flujo de la subrutina pantalla_abajo

7.4.6.- Movimiento de mensaje y flecha hacia arriba:

Esta subrutina es la encargada de desplazar tanto los mensajes como la flecha que indica la opción seleccionada hacia arriba en el LCD.

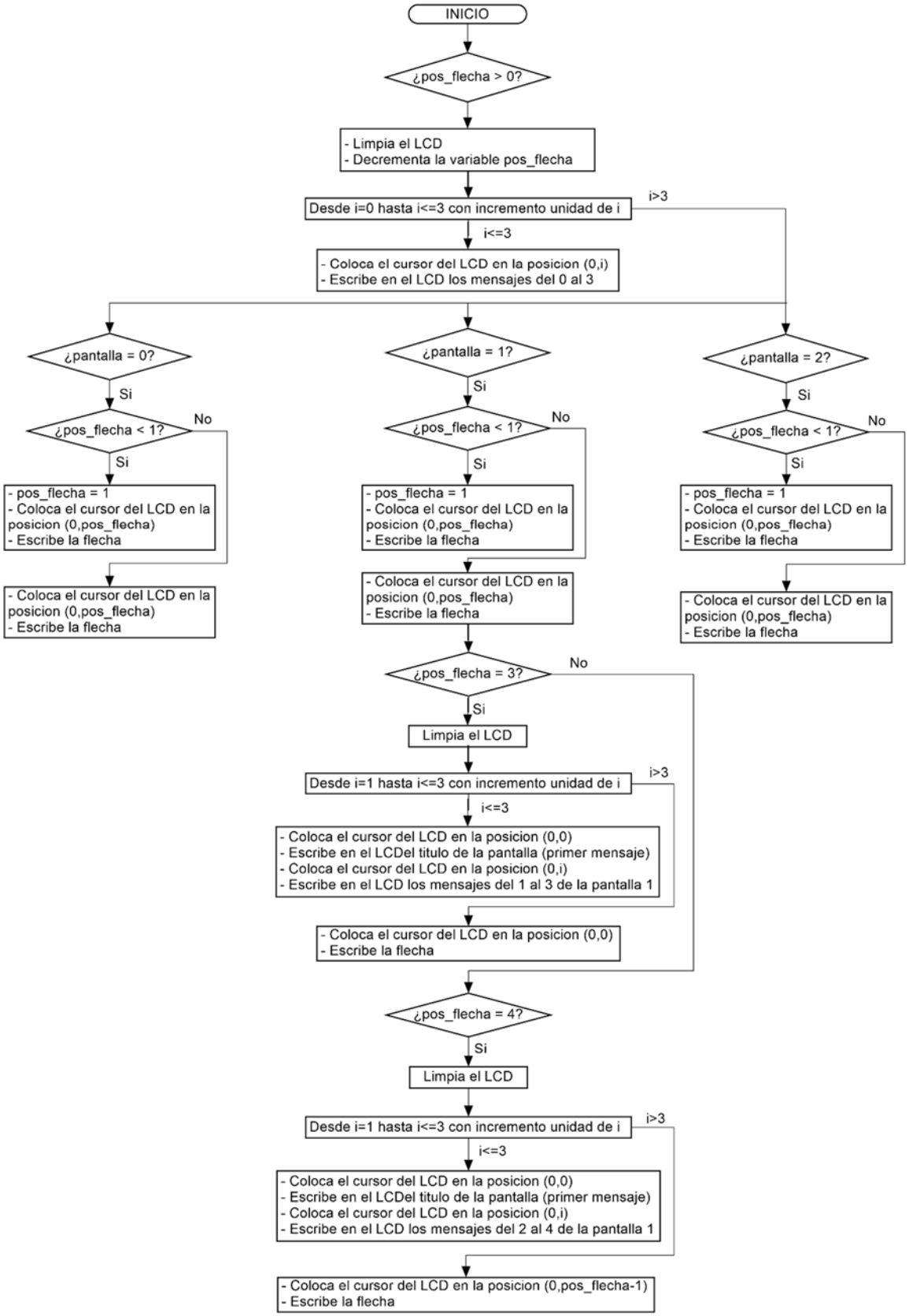


Diagrama de flujo de la subrutina pantalla_arriba

7.4.7.- Configuración del tipo de paso:

Esta subrutina es la encargada de configurar los pines de salida MS1, MS2 y MS3 con el objetivo de fijar el tipo de paso que se desea que gire el motor.

Como ya se ha explicado anteriormente, para configurar el tipo de paso que se desea, basta con combinar el estado de los tres pines del controlador destinados para tal función según la siguiente tabla.

MS1	MS2	MS3	Tipo de paso
0	0	0	Paso completo
1	0	0	Medio paso
0	1	0	Cuarto de paso
1	1	0	Octavo de paso
1	1	1	Dieciseisavo de paso

Para el correcto funcionamiento de esta subrutina hay que indicar el tipo de paso que se desea en la variable “tipo_paso”, dándole los siguientes valores.

Tipo de paso	Valor de la variable tipo_paso
Paso completo	1
Medio paso	2
1/4 de paso	3
1/8 de paso	4
1/16 de paso	5

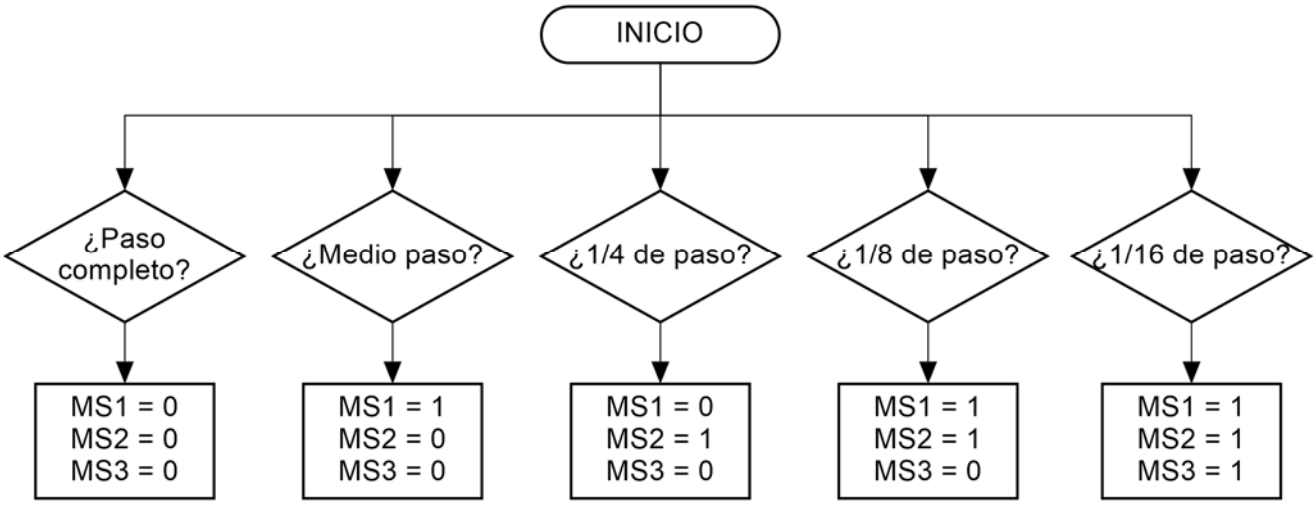


Diagrama de flujo de la subrutina config_tipo_paso

7.4.8.- Lectura números introducidos por teclado:

Ante la necesidad de introducir números de varias cifras mediante el teclado para introducir la frecuencia, el número de pasos o los grados a girar, se ha diseñado esta subrutina capaz de leer números de hasta tres cifras introducidos mediante el teclado y escribirlo en el LCD para poder así visualizar el valor introducido y corregirlo si fuese necesario.

Para ello se utilizan tres variables en las que se almacenan las pulsaciones de cada tecla (valor_1, valor_2 y valor_3), que luego se unen en una única variable (valor_tec) multiplicando cada valor por el múltiplo de diez adecuado con el fin de darle el peso adecuado a cada número.

Para saber que factor de multiplicación hay que aplicar a cada número introducido, se usa la variable "num_puls", que cuenta el número de pulsaciones que se han realizado, pudiendo saber así si se han introducido las unidades, las decenas o las centenas.

Como el teclado devuelve un carácter ASCII, es necesario convertirlo a decimal para poder usarlo como un valor a introducir en una variable. Para hacer esta conversión, simplemente hay que restar 48 al carácter ASCII que ha entregado el teclado.

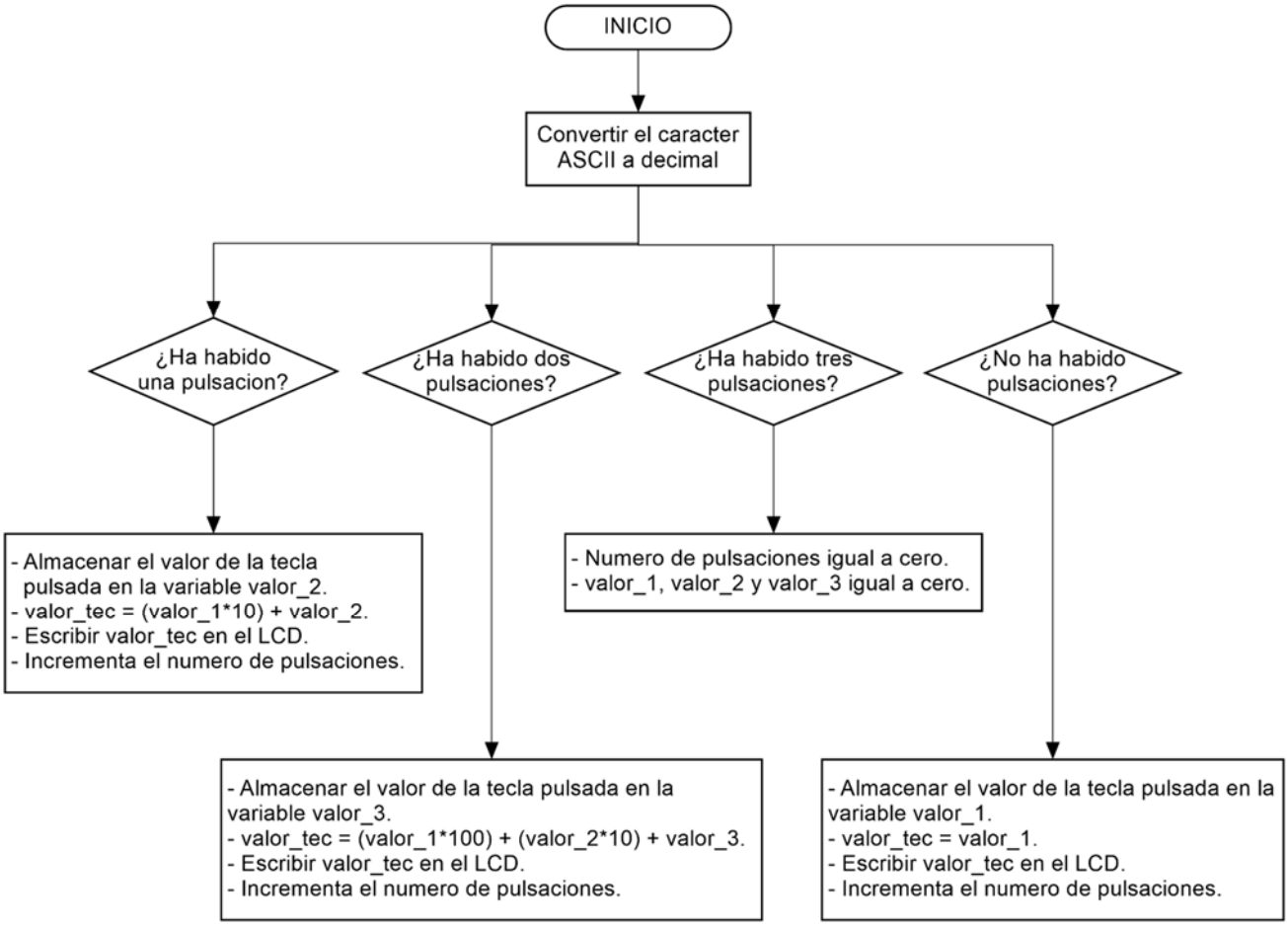


Diagrama de flujo de la subrutina lee_num_tec

7.4.9.- Paso único:

Esta subrutina es la encargada de gestionar la situación en la que se desea dar un paso único, de forma que con cada pulsación de la tecla enter se llama a esta subrutina y se genera un único pulso de reloj (entrada STEP del controlador del motor) para que el motor gire un único paso.

La variable que almacena el número de pasos que ha girado el motor comienza con el valor -1 porque aún no se ha dado ningún paso, de forma que lo primero que se hace es incrementarla para que valga cero y en este proceso no se da ningún paso. Después de esto, ya sí que se gira el motor a la vez que se incrementa la variable que almacena el número de pasos que ha girado el motor.

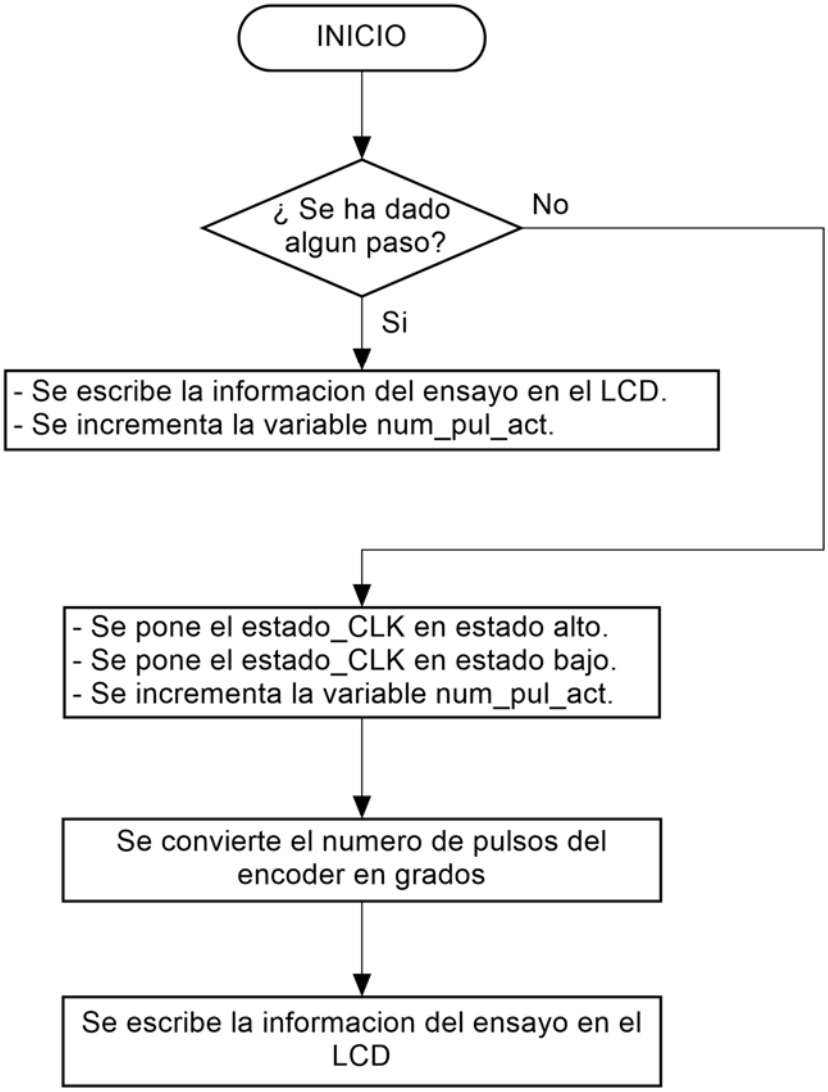


Diagrama de flujo de la subrutina paso_unico

7.4.10.- Atrás:

Mediante esta subrutina se produce el movimiento hacia atrás entre pantallas, de forma que si se quiere corregir una selección ya realizado, se pulsa la tecla atrás y se va retrocediendo entre las pantallas.

En el caso de que se pulse en alguna de las pantallas que requieren la introducción de un dato numérico, se borrara el dato y con una segunda pulsación se retrocede a la pantalla anterior. Al retroceder una pantalla hay que inicializar las variables que intervienen en la selección de las distintas opciones de cada pantalla para evitar errores en la selección de estas opciones.

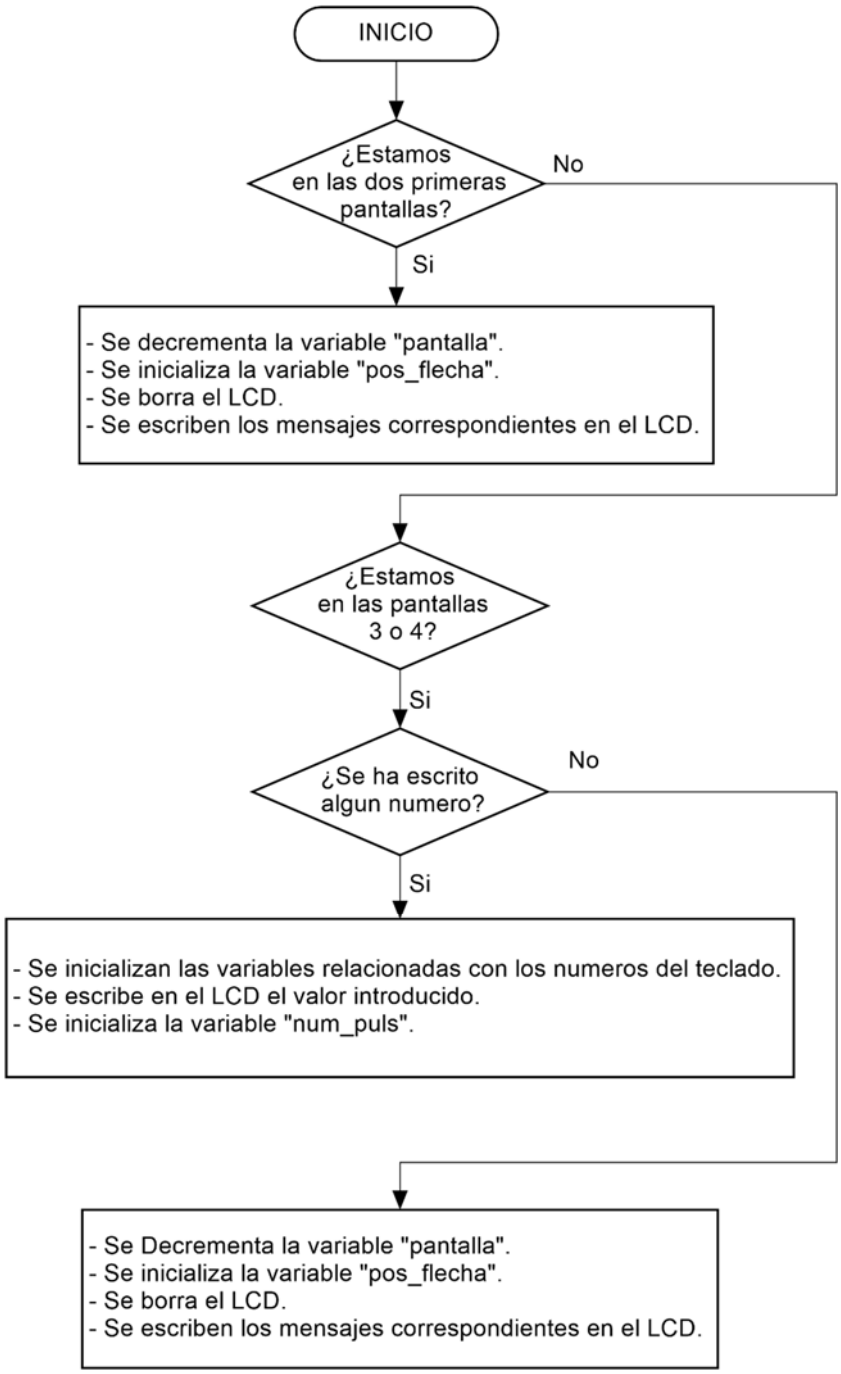


Diagrama de flujo de la subrutina atras

7.4.11.- Nuevo ensayo:

Esta es una subrutina muy simple cuya única finalidad es la de poder realizar un nuevo ensayo una vez se ha terminado el anterior sin tener que hacer un reset completo del sistema, para ello se deben borrar todas las variables que se han ido actualizando al realizar las distintas selecciones en la configuración del ensayo e inicializar las salidas de la placa Arduino conectadas al controlador del motor paso a paso.

Mediante esta subrutina, también se para el reloj que marca los pasos del motor para poder interrumpir un ensayo sin terminar e iniciar uno nuevo, se limpia el LCD para poder escribir los mensajes de la primera pantalla (selección de modo de operación) sin que se superpongan a los mensajes de la pantalla que se estaba mostrando en el momento de pulsar la tecla de nuevo ensayo.

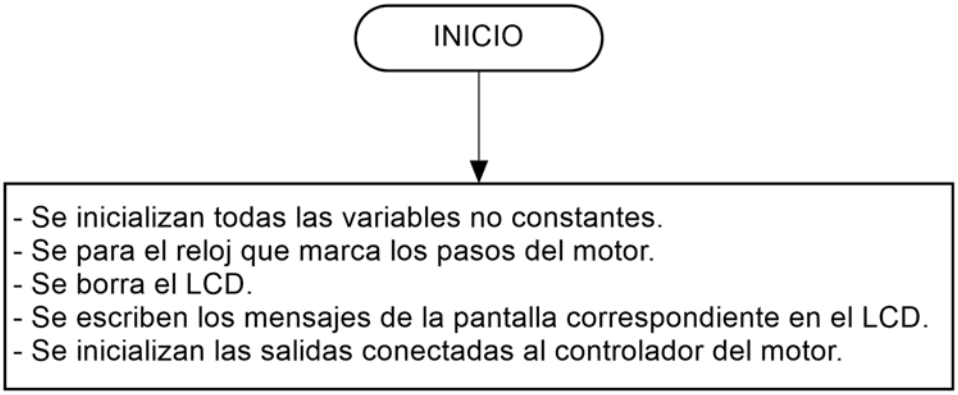
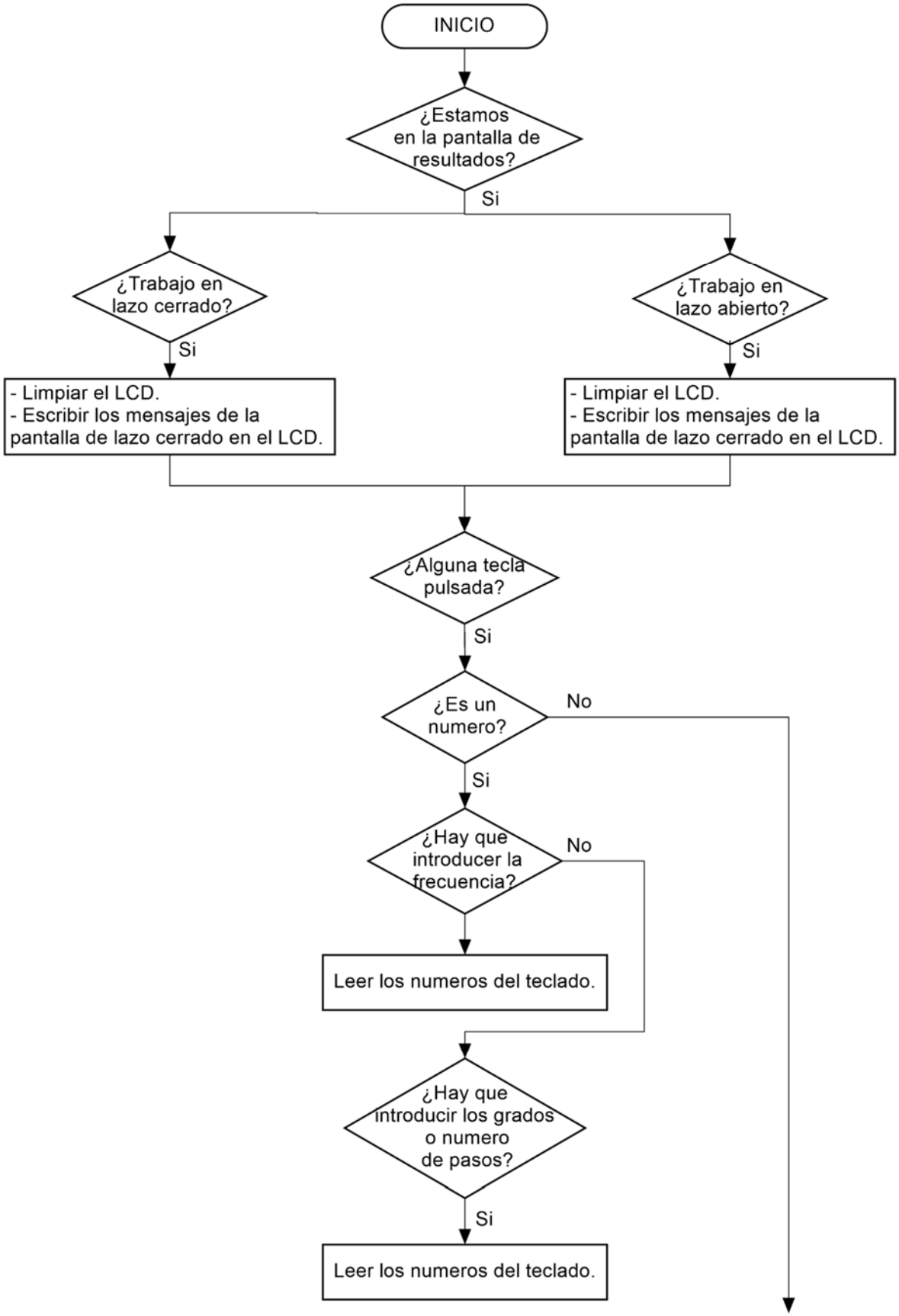


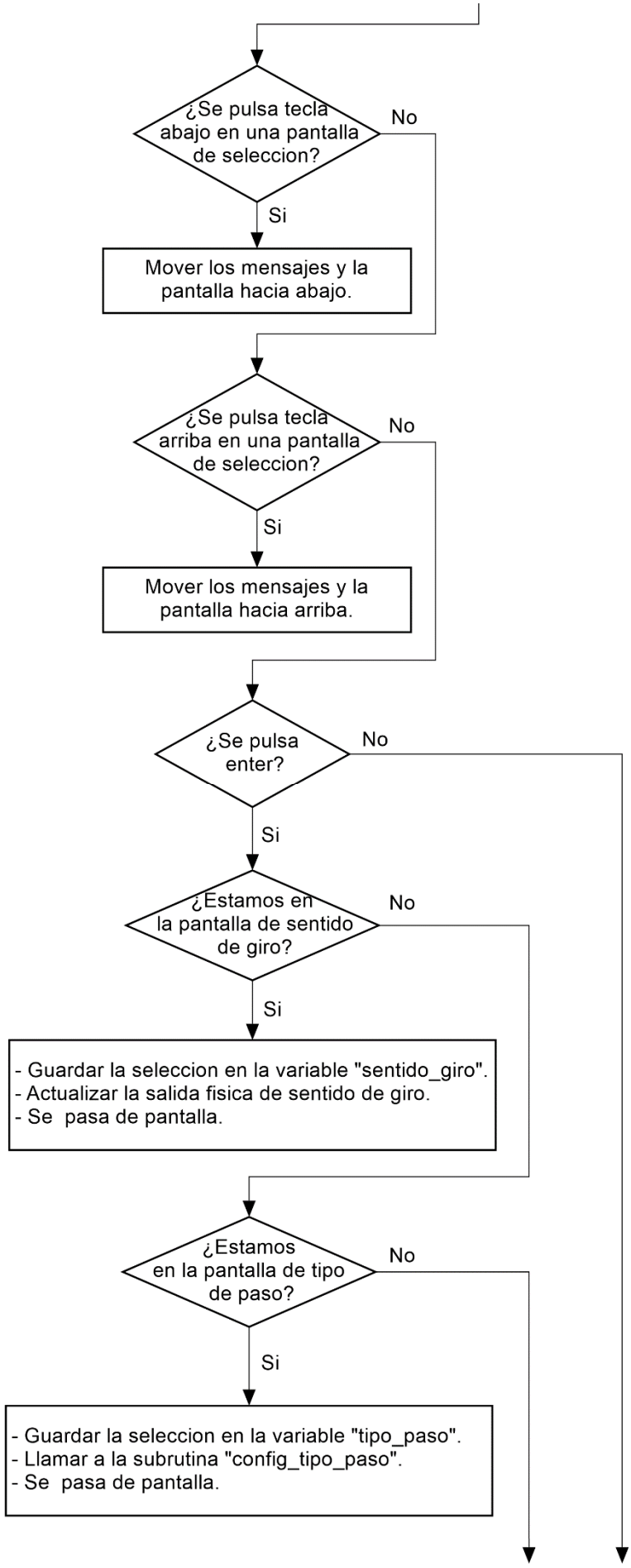
Diagrama de flujo de la subrutina nuevo_ensayo

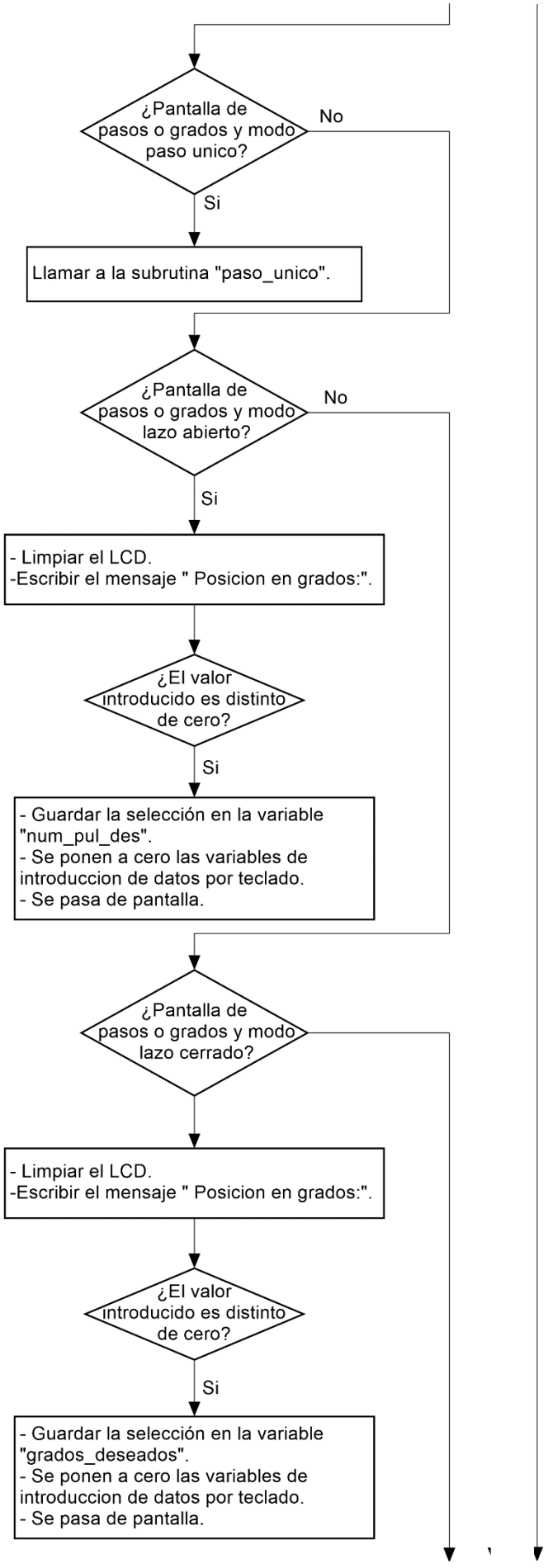
7.5.- Programa principal:

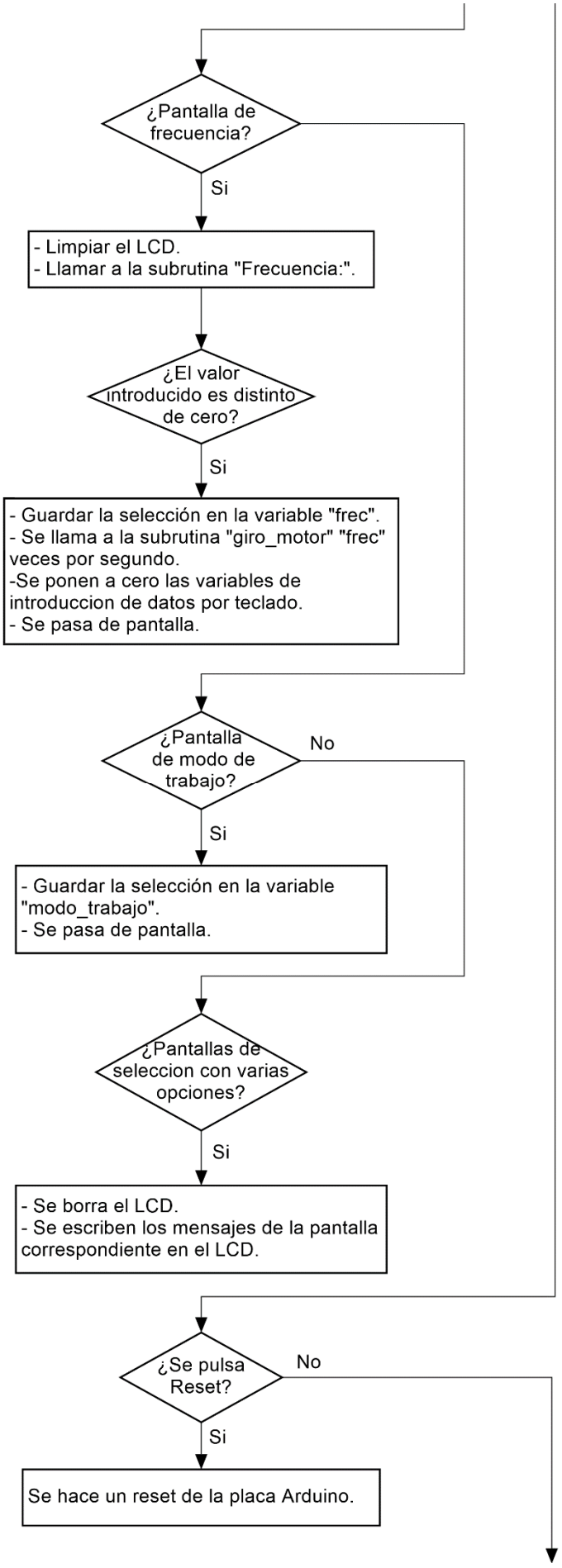
A continuación se presenta el diagrama de flujo correspondiente al programa principal de este proyecto.

En esta parte del programa, se va llamando a las distintas subrutinas que se han explicado anteriormente en función del botón que se haya pulsado en el teclado, a excepción del resumen de los datos del ensayo y de los resultados del mismo, ya que para que esta información sea visible, no es necesario pulsar ninguna tecla, sino que con estar en la pantalla adecuada (resumen y resultados del ensayo) es suficiente.


INICIO
¿Estamos en la pantalla de resultados?
Si
¿Trabajo en lazo cerrado?
Si
- Limpiar el LCD.
- Escribir los mensajes de la pantalla de lazo cerrado en el LCD.
¿Trabajo en lazo abierto?
Si
- Limpiar el LCD.
- Escribir los mensajes de la pantalla de lazo cerrado en el LCD.
¿Alguna tecla pulsada?
Si
¿Es un numero?
No
Si
¿Hay que introducer la frecuencia?
No
Leer los numeros del teclado.
¿Hay que introducir los grados o numero de pasos?
Si
Leer los numeros del teclado.


¿Se pulsa tecla abajo en una pantalla de seleccion?
No
Si
Mover los mensajes y la pantalla hacia abajo.
¿Se pulsa tecla arriba en una pantalla de seleccion?
No
Si
Mover los mensajes y la pantalla hacia arriba.
¿Se pulsa enter?
No
Si
¿Estamos en la pantalla de sentido de giro?
No
Si
- Guardar la seleccion en la variable "sentido_giro".
- Actualizar la salida fisica de sentido de giro.
- Se pasa de pantalla.
¿Estamos en la pantalla de tipo de paso?
No
Si
- Guardar la seleccion en la variable "tipo_paso".
- Llamar a la subrutina "config_tipo_paso".
- Se pasa de pantalla.


¿Pantalla de pasos o grados y modo paso unico?
No
Si
Llamar a la subrutina "paso_unico".
¿Pantalla de pasos o grados y modo lazo abierto?
No
Si
- Limpiar el LCD.
-Escribir el mensaje " Posicion en grados:".
¿El valor introducido es distinto de cero?
Si
- Guardar la selección en la variable "num_pul_des".
- Se ponen a cero las variables de introduccion de datos por teclado.
- Se pasa de pantalla.
¿Pantalla de pasos o grados y modo lazo cerrado?
- Limpiar el LCD.
-Escribir el mensaje " Posicion en grados:".
¿El valor introducido es distinto de cero?
Si
- Guardar la selección en la variable "grados_deseados".
- Se ponen a cero las variables de introduccion de datos por teclado.
- Se pasa de pantalla.


¿Pantalla de frecuencia?
Si
- Limpiar el LCD.
- Llamar a la subrutina "Frecuencia:".
¿El valor introducido es distinto de cero?
Si
- Guardar la selección en la variable "frec".
- Se llama a la subrutina "giro_motor" "frec" veces por segundo.
-Se ponen a cero las variables de introduccion de datos por teclado.
- Se pasa de pantalla.
¿Pantalla de modo de trabajo?
No
Si
- Guardar la selección en la variable "modo_trabajo".
- Se pasa de pantalla.
¿Pantallas de seleccion con varias opciones?
Si
- Se borra el LCD.
- Se escriben los mensajes de la pantalla correspondiente en el LCD.
¿Se pulsa Reset?
No
Si
Se hace un reset de la placa Arduino.

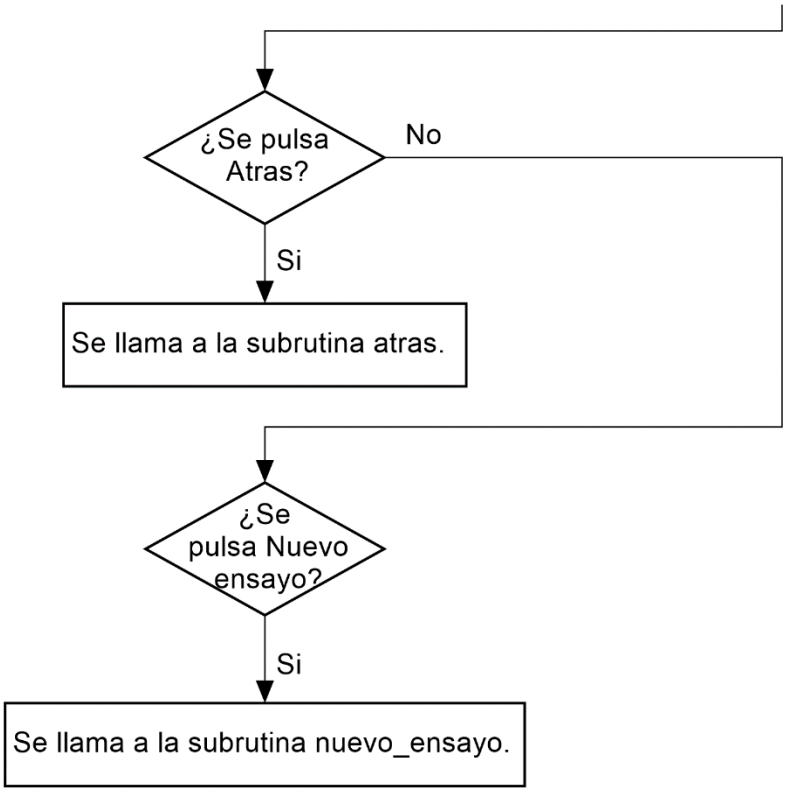


Diagrama de flujo del programa principal.

7.6.- Código de programa:

En este apartado se va incluir el código utilizado para hacer funcionar correctamente el proyecto, incluyendo los comentarios necesarios para la correcta comprensión de todas las líneas de código escritas.

```
//*********************************************************************
//******************************** LIBRERIAS ****************************
//*********************************************************************

#include <LiquidCrystal.h>          // Librería del LCD
#include <Keypad.h>                 // Librería del teclado
#include <SPI.h>                    // Librería de comunicación SPI
#include <DueTimer.h>               // Librería de temporizadores para interrupción


//*********************************************************************
//************************** NOMBRE DE LOS PINES **********************
//*********************************************************************

const int reset_arduino = 46;       // Nombre del pin de reset del Arduino
```

```
//**************** PINES DEL CONTROLADOR DEL MOTOR  ******************

const int CLK = 47;                 // Nombre del pin de salida del reloj del controlador del motor
const int MS1 = 41;                 // Nombre del pin de la salida MS1
const int MS2 = 43;                 // Nombre del pin de la salida MS2
const int MS3 = 45;                 // Nombre del pin de la salida MS3
const int dir = 49;                 // Nombre del pin de la salida dir (indica la dirección de giro  del
                                       motor     0->antihorario      1->horario)
const int sleep = 35;               // Nombre del pin de la salida sleep
const int reset = 37;               // Nombre del pin de la salida reset ( 0-> motor parado           1->
                                       motor en marcha)
const int enable = 39;              // Nombre del pin de la salida enable (1-> deshabilitado          0->
                                       habilitado)


//*********************** PINES DEL LS7366R  ***************************

const int CS = 52;                  //CS(ChipSet) o enable del integrado


//*****************************************************************
//***************************** VARIABLES  ****************************
//*****************************************************************

//******************** VARIABLES GENERALES **************************

int modo_trabajo = 0;         // 1->LAZO CERRADO    2->LAZO ABIERTO    3->PASO UNICO
int tipo_paso = 2;            // 1->PASO COMPLETO    2->MEDIO PASO    3->1/4 DE PASO          4-
                                 >1/8 DE PASO    5->1/16 DE PASO
int sentido_giro = 1;         //  1->SENTIDO ANTIHORARIO    2->SENTIDO HORARIO (inicialmente no
                                 tiene ningún valor)
int frec = 10;                      // Frecuencia a la que se desea que gire el motor
int grados_girados = 0;             // Grados que ha girado el motor
int grados_deseados = 0;            // Grados que se desea que gire el motor
int num_pul_act = -1;               // Numero de pasos que ha girado el motor
int num_pul_des = 0;                // Numero de pasos que se desea que gire el motor
bool estado_CLK = false;            // Estado del reloj que marca los pasos del motor
char* nombre_tipo_paso [5] = {"Completo", "Medio", "1/4", "1/8", "1/16"};  // Textos a mostrar
                                       en las pantallas de resumen del ensayo en el caso del tipo
                                       de paso
char* nombre_direccion [2] = {"Antihora", "Horario"};   // Textos a mostrar en las pantallas de
                                       resumen del ensayo en el caso del sentido de giro
```

```
//***************** VARIABLES RELACIONADAS CON EL LCD *****************

const int grupo_msg = 3;              // Se definen 3 grupos de mensajes, uno por pantalla (filas de
                                         la matriz)
const int mensaje = 6;                // Se definen 5 mensajes por pantalla aunque algunos estarán
                                         en blanco (columnas de la matriz)
int pantalla = 0;                     // Indica la pantalla en la que nos encontramos
char* opciones[grupo_msg][mensaje] = { {"Modo de operacion:", " Lazo cerrado", " Lazo abierto",
" Paso unico", " ", " "},  {"Tipo de paso:", " Paso completo", " Medio paso", " 1/4 de paso", " 1/8 de
paso", " 1/16 de paso"},  {"Sentido de giro:", " Antihorario", " Horario", " ", " ", " "}   };
                                      // Matriz de la lista de mensajes
char* introduccion_datos [3] = {"Posicion en grados:", "Numero de pasos:", "Frecuencia en Hz:"};
                                      // Lista de titulos para las pantallas de introducción de  datos
char* resumen_lazo_abierto [4] = {"Tipo paso: ", "Direccion: ", "Numero pasos: ", "Posicion :"};
char* resumen_lazo_cerrado [3] = {"Tipo paso: ", "Direccion: ", "Posicion :"};
int pos_flecha = -1;                   // pos_flecha toma valores entre 0 y 3, y empieza en -1 para
                                         que inicialmente no aparezca
byte flecha_derecha[8] = {B00000, B00100, B00010, B11111, B00010, B00100, B00000, B00000};
                                    // Definición del carácter flecha derecha como el carácter número 1
byte simbolo_grados[8] = {B00100, B01010, B00100, B00000, B00000, B00000, B00000, B00000};
                                   // Definición del carácter símbolo grados como el carácter número 2


//*************** VARIABLES RELACIONADAS CON EL TECLADO ***************

const int ROWS = 5;                   // Numero de filas del teclado
const int COLS = 4;                   // Numero de columnas del teclado
char keys[ROWS][COLS] = {             // Caracter que se devuelve al pulsar cada tecla
  {'1', '2', '3', 'R'},
  {'4', '5', '6'},
  {'7', '8', '9', 'N'},
  {'0', 'A', 'B'},
  {'D', 'E', 'F'}
};
char tecla = '0';                  // Variable en la que se almacena la tecla del teclado pulsada. Se
                                      inicia con Z para que no haya ninguna tecla pulsada
int valor_tec = NULL;              // Variable en la que se almacena el valor numérico introducido
                                      por teclado
int valor_1 = NULL;                // Variable en la que se almacena el valor numérico de la primera
                                      tecla pulsada
int valor_2 = NULL;                // Variable en la que se almacena el valor numérico de la segunda
                                      tecla pulsada
int valor_3 = NULL;                // Variable en la que se almacena el valor numérico de la tercera
                                       tecla pulsada
int tecla_pulsada = NULL;          // Variable que indica la tecla que se ha pulsado
int num_puls = NULL;               // Variable que indica el número de pulsaciones que se han
                                      realizado
```

```
//******** VARIABLES RELACIONADAS CON LA LECTURA DEL ENCODER *********

signed long cuentaencoder = 0;      // Variable donde se almacena el contaje del encoder (debe ser
                                       de tipo long por la gran cantidad de datos a almacenar)
long valor_cont;                    // Valor que devuelve la subrutina de lectura del encoder


//******* VARIABLES RELACIONADAS CON EL CONTROLADOR DEL MOTOR ******

int estado_enable = LOW;            // Indica el estado del pin enable del controlador del motor
int estado_reset = HIGH;            // Indica el estado del pin reset del controlador del motor


//*****************************************************************
//********************* DEFINICION DEL TECLADO ********************
//*****************************************************************

byte rowPins[ROWS] = {30, 28, 26, 24, 22};       // Conexión de las filas del teclado
byte colPins[COLS] = {38, 36, 34, 32};           // Conexión de las columnas del teclado
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );   // Se crea un
                                                    objeto de la librería keypad para usar sus funciones


//*****************************************************************
//********************** DEFINICION DEL LCD ***********************
//*****************************************************************

LiquidCrystal lcd(12, 11, 7, 8, 9, 10);     // Conexiones del LCD (RS, ENABLE, D4, D5, D6, D7)



//*****************************************************************
//****************************** SETUP ****************************
//*****************************************************************

void setup() {

  inicializa();                     // Se inicializa el integrado de contaje del encoder
  borracuenta();                    // Se borra la cuenta del encoder

  Serial.begin(9600);
```

```
//*************************** SALIDAS ****************************

pinMode(reset, OUTPUT);              // Definición de reset como salida
pinMode(sleep, OUTPUT);              // Definición de sleep como salida
pinMode(enable, OUTPUT);             // Definición de enable como salida
pinMode(MS1, OUTPUT);                // Definición de MS1 como salida
pinMode(MS2, OUTPUT);                // Definición de MS2 como salida
pinMode(MS3, OUTPUT);                // Definición de MS3 como salida
pinMode(CLK, OUTPUT);                // Definición de CLK como salida
pinMode(dir, OUTPUT);                // Definición de dir como salida
pinMode(reset_arduino, OUTPUT);      // Definición de reset_arduino como salida


//******************** INICIALIZACION DEL LCD ************************

lcd.createChar(1, flecha_derecha);       // Declaración del carácter flecha derecha como el
                                           símbolo número 1
lcd.createChar(2, simbolo_grados);       // Declaración del carácter símbolo grados como el
                                           símbolo número 2

lcd.begin(20, 4);                        // Tamaño del LCD (20x4)
lcd.clear();                             // Se borra el LCD por si hay algo escrito

lcd.setCursor(0, 0);                     // Se escribe la pantalla de presentación
lcd.write("********************");
lcd.setCursor(0, 1);
lcd.write("*ENTRENADOR MOTORES*");
lcd.setCursor(0, 2);
lcd.write("*   PASO A PASO    *");
lcd.setCursor(0, 3);
lcd.write("********************");

delay(5000);                             // Se mantiene la pantalla de presentación durante 5
                                           segundos (5000 ms)

lcd.clear();                             //Se borra el LCD
for (int i = 0; i <= 3; i++)             // Se escriben los mensajes de la pantalla 0 en el LCD
{
  lcd.setCursor(0, i);
  lcd.write(opciones[pantalla][i]);
}

//************** INICIALIZACION DE ENTRADAS Y SALIDAS ****************

digitalWrite(reset_arduino, HIGH);   // Se inicializa el pin reset a estado alto (el reset se hace con
                                        un cero)
digitalWrite(sleep, LOW);            // Se inicializa el pin sleep a estado bajo (pausa)
```

```
  digitalWrite(MS1, LOW);               // Se inicializa el controlador sin ningún tipo de paso
                                           seleccionado (MS1=0, MS2=0 y MS3=1 no es nada)
  digitalWrite(MS2, LOW);
  digitalWrite(MS3, LOW);
  digitalWrite(dir, sentido_giro);      // El pin de sentido de giro se inicializa con el valor de la
                                           variable sentido_giro
  digitalWrite(enable, estado_enable);// El pin de enable se inicializa con el valor de la variable
                                           estado_enable
  digitalWrite(reset, estado_reset);    // El pin de reset se inicializa con el valor de la variable
                                           estado_reset

}


//*****************************************************************
//***************************** SUBRUTINAS ************************
//*****************************************************************

// ***************************** GIRO DEL MOTOR *********************


void giro_motor() {                     // Subrutina de giro del motor

  lee_encoder;                          // Se lee el encoder

  digitalWrite(sleep, HIGH);            // Se "despierta" el controlador del motor
  estado_CLK = HIGH;                    // Se pone el pin del reloj en estado alto
  digitalWrite(CLK, estado_CLK);
  estado_CLK = LOW;                     // Se pone el pin del reloj en estado bajo
  digitalWrite(CLK, estado_CLK);
  num_pul_act++;                        // Se incrementa la variable que contabiliza el número de
                                           pasos

  cuentaencoder = lee_encoder(1);       // Se guarda el valor del contaje del encoder en la variable
                                           cuentaencoder
  grados_girados = (cuentaencoder * 360 / 16384);     // Se convierte el número de pulsos del
                                                         encoder en grados

  if (modo_trabajo == 1)                // Si se trabaja en lazo cerrado
  {
    lcd.setCursor(11, 0);               // Se escriben los datos del ensayo en el LCD
    lcd.write(nombre_tipo_paso[tipo_paso - 1]);
    lcd.setCursor(11, 1);
    lcd.write(nombre_direccion[sentido_giro]);
    lcd.setCursor(11, 2);
    lcd.print(grados_girados);
    lcd.write(2);
    lcd.setCursor(0, 3);
```

```
  lcd.write("               ");              // Borra símbolos extraños que aparecen en la última fila


  if (abs(grados_girados) >= grados_deseados)    // Giro por número de grados (Lazo cerrado). Se
                                                    compara con el valor absoluto porque al girar
                                                    en sentido antihorario se cuenta en negativo
  {
   Timer6.stop();                          // Se detiene el contador que llama a la subrutina de giro
                                              del motor
   digitalWrite(sleep, LOW);               // Se "duerme" el controlador del motor
  }
 }

 if (modo_trabajo == 2)                    // Si se trabaja en lazo abierto
 {

  lcd.setCursor(11, 0);                    // Se escriben los datos del ensayo en el LCD
  lcd.write(nombre_tipo_paso[tipo_paso - 1]);
  lcd.setCursor(11, 1);
  lcd.write(nombre_direccion[sentido_giro]);
  lcd.setCursor(14, 2);
  lcd.print(num_pul_act);
  lcd.setCursor(11, 3);
  lcd.print(grados_girados);
  lcd.write(2);

  if (num_pul_act > num_pul_des - 1)   // Giro por número de pulsos (Lazo abierto)
  {
   Timer6.stop();                          // Se detiene el contador que llama a la subrutina de giro
                                              del motor
   digitalWrite(sleep, LOW);               // Se "duerme" el controlador del motor
  }
 }
}

//****************** INICIALIZACION DEL CONTADOR *********************

void inicializa() {                        // Subrutina de inicialización del contador del encoder

 SPI.begin(CS);                            // Se inicializa la librería SPI con el CS en el pin 4

 SPI.setClockDivider(CS, 21);              // Se ajusta la división para el reloj en la pata 4 (84Mhz/21 =
                                              4Mhz)

 SPI.transfer(CS, 0x88, SPI_CONTINUE);   // (En binario 10001000) Se escribir en MDR0 y se
                                              continua con la transmision de datos
 SPI.transfer(CS, 0x03);                   // (En binario 00000011) Modo de cuenta x4
                                           // Modo de contaje continuo
```

```
                                        // Index deshabilitado y asíncrono
                                        // Factor división del reloj 1

}


//********************** LECTURA DEL ENCODER ***********************

long lee_encoder(int encoder) {         // Subrutina de lectura del encoder


  unsigned int cont_1, cont_2, cont_3, cont_4;       // Declara las variables para leer el SPI

  SPI.transfer(CS, 0x60, SPI_CONTINUE);               // (En binario 01100000) Lee CNTR para pedir
la cuenta
  cont_1 = SPI.transfer(CS, 0x00, SPI_CONTINUE);   // Se lee el byte más alto
  cont_2 = SPI.transfer(CS, 0x00, SPI_CONTINUE);
  cont_3 = SPI.transfer(CS, 0x00, SPI_CONTINUE);
  cont_4 = SPI.transfer(CS, 0x00);                    // Se lee el byte más bajo

  valor_cont = (cont_1 << 8) + cont_2;                 // Calcula la cuenta del encoder
  valor_cont = (valor_cont << 8) + cont_3;
  valor_cont = (valor_cont << 8) + cont_4;

  return valor_cont;
}


//****************** BORRADO CONTAJE ENCODER **********************

void borracuenta() {                    // Subrutina de borrado de la cuenta del encoder

  SPI.transfer(CS, 0x98, SPI_CONTINUE);          //(En binario 10011000) Escribir en DTR
  SPI.transfer(CS, 0x00, SPI_CONTINUE);          // Byte de mayor orden
  SPI.transfer(CS, 0x00, SPI_CONTINUE);
  SPI.transfer(CS, 0x00, SPI_CONTINUE);
  SPI.transfer(CS, 0x00);                         // Byte de menor orden

  delayMicroseconds(100);                         // Retraso entre transmisiones del bus SPI

  SPI.transfer(CS, 0xE0);                         // (En binario 11100000) Cargar DTR en CNTR

}
```

```
//******** MOVIMIENTO DE LOS MENSAJES Y LA FLECHA HACIA ABAJO ********

void pantalla_abajo() {          // Subrutina de movimiento de mensajes y flecha hacia abajo

 pos_flecha++;                   // Se incrementa la variable pos_flecha en una unidad
 lcd.clear();                    // Se limpia el LCD
 for (int i = 0; i <= 3; i++)    // Se escriben los mensajes del 0 al 3
 {
  lcd.setCursor(0, i);
  lcd.write(opciones[pantalla][i]);
 }


 if (pantalla == 0)          // ********* Comportamiento en la pantalla 0 ********
 {
  if (pos_flecha == 0)                  // Si pos_flecha = 0 (la línea del título)
  {
   pos_flecha++;                        // Se incrementa en uno la variable pos_flecha para
                                           colocarla en la línea 1
   lcd.setCursor(0, pos_flecha);        // Se coloca el cursor en la posición deseada para escribir la
                                           flecha (0,pos_flecha)
   lcd.write(1);                        // Se escribe la flecha en la línea pos_flecha
  }

  else if (pos_flecha <= 3)             // La variable pos_flecha es menor o igual que 3 (estamos
                                           en la fila 1, 2 o 3 (no podemos estar en la 0 por la
                                           condición anterior))
  {
   lcd.setCursor(0, pos_flecha);        // Se coloca el cursor en la posición deseada para escribir la
                                           flecha (0,pos_flecha)
   lcd.write(1);                        // Se escribe la flecha en la línea pos_flecha
  }

  else                                  // Si pos_flecha es mayor que 3 (pos_flecha supera el
                                           número de mensajes)
  {
   pos_flecha = 3;                      // Se hace pos_flecha = 3 para acotar el valor de la variable
                                           (3 como máximo)
   lcd.setCursor(0, 3);                 // Coloca el cursor en la posición 0,3
   lcd.write(1);                        // Se escribe la flecha en la línea 3
  }
 }

 if (pantalla == 1)          // ********** Comportamiento en la pantalla 1 *******
 {
  if (pos_flecha == 0)                  // Si pos_flecha = 0 (la línea del título)
  {
```

```
 pos_flecha++;                    // Se incrementa en uno la variable pos_flecha para
                                     colocarla en la línea 1
 lcd.setCursor(0, pos_flecha);    // Se coloca el cursor en la posición deseada para escribir la
                                     flecha (0,pos_flecha)
 lcd.write(1);                    // Se escribe la flecha en la línea pos_flecha
}

else if ( pos_flecha < 3)         // Si la variable pos_flecha es menor que 3 (aún no se ha
                                     llegado a la última posición)
{
 lcd.setCursor(0, pos_flecha);    // Se coloca el cursor en la posición deseada para escribir la
                                     flecha (0,pos_flecha)
 lcd.write(1);                    // Se escribe la flecha en la línea pos_flecha
}

else                              // Si pos_flecha es mayor o igual que 3
{
 lcd.setCursor(0, 3);             // Se coloca el cursor en la posición 0,3
 lcd.write(1);                    // Se escribe la flecha

 if ( pos_flecha == 4)            // La variable pos_flecha es mayor o igual que 4 (se ha
                                     superado el número de mansajes que cabe en la
                                     pantalla)
 {
  lcd.clear();                    // Se limpia el LCD
  for (int i = 1; i <= 3; i++)    // Se escriben los mensajes del 2 al 4
  {
   lcd.setCursor(0, 0);
   lcd.write(opciones[pantalla][0]);
   lcd.setCursor(0, i);
   lcd.write(opciones[pantalla][i + 1]);
  }

  lcd.setCursor(0, 3);            // Se coloca el cursor en 0,3
  lcd.write(1);                   // Se escribe la flecha
 }

 else if ( pos_flecha >= 5)       // La variable pos_flecha es mayor o igual que 5 (se ha
                                     superado el número de mansajes que cabe en la
                                     pantalla)
 {
  lcd.clear();                    // Se limpia el LCD
  for (int i = 1; i <= 3; i++)    // Se escriben los mensajes del 3 al 5
  {
   lcd.setCursor(0, 0);
   lcd.write(opciones[pantalla][0]);
   lcd.setCursor(0, i);
   lcd.write(opciones[pantalla][i + 2]);
```

```
      }

      lcd.setCursor(0, 3);              // Se coloca el cursor en 0,3
      lcd.write(1);                     // Se escribe la flecha
      pos_flecha = 5;                   // pos_flecha se pone a 5 para acotar el valor de la variable
                                           (4 como máximo)
    }
   }
  }

 if (pantalla == 2)          // ********** Comportamiento en la pantalla 2 *******
 {
   if (pos_flecha == 0)             // Si pos_flecha es 0 (la línea del título)
   {
     pos_flecha++;                  // Se incrementa en uno la variable pos_flecha para
                                       colocarla en la línea 1
     lcd.setCursor(0, pos_flecha);  // Se coloca el cursor en la posición deseada para escribir la
                                       flecha (0,pos_flecha)
     lcd.write(1);                  // Se escribe la flecha
   }
   else
   {
     pos_flecha = 2;                // Se pone pos_flecha a 2 para que no supero esta posición;
     lcd.setCursor(0, pos_flecha);  // Se coloca el cursor en la posición deseada para escribir la
                                       flecha (0,pos_flecha)
     lcd.write(1);                  // Se escribe la flecha
   }
  }
}


//******** MOVIMIENTO DE LOS MENSAJES Y LA FLECHA HACIA ARRIBA *******

void pantalla_arriba() {          // Subrutina de movimiento de mensajes y flecha hacia abajo

  if (pos_flecha > 0)             // Solo se ejecuta esta parte si pos_flecha es mayor que 0 (se evita
                                     que inicialmente aparezca la flecha al pulsar la flecha arriba)

 {
   pos_flecha--;                 // Se decrementa la variable pos_flecha en una unidad
   lcd.clear();                  // Se limpia el LCD
   for (int i = 0; i <= 3; i++)  // Se escriben los mensajes del 0 al 3
   {
     lcd.setCursor(0, i);
     lcd.write(opciones[pantalla][i]);
   }
```

```
if (pantalla == 0)          // ********* Comportamiento en la pantalla 0 ********
{
 if (pos_flecha < 1)                // Si pos_flecha es menor que uno (nos salimos de la zona de
                                       mensajes por arriba)
 {
  pos_flecha = 1;                   // pos_flecha se hace igual a 1 para acotar el valor de la
                                       variable pos_flecha
  lcd.setCursor(0, pos_flecha);     // Se sitúa el cursor en la columna cero fila pos_flecha
  lcd.write(1);                     // Se escribe la flecha
 }

 else                               // Si estamos en la línea 1, 2 o 3
 {
  lcd.setCursor(0, pos_flecha);     // Se sitúa el cursor en la columna cero fila pos_flecha
  lcd.write(1);                     // Se escribe la flecha
 }
}


if (pantalla == 1)          // ********* Comportamiento en la pantalla 1 ********
{
 if (pos_flecha < 1)                // Si pos_flecha es menor que uno (nos salimos de la zona de
                                       mensajes por arriba)
 {
  pos_flecha = 1;                   // pos_flecha se hace igual a 1 para acotar el valor de la
                                       variable pos_flecha
  lcd.setCursor(0, pos_flecha);     // Se sitúa el cursor en la columna cero fila pos_flecha
  lcd.write(1);                     // Se escribe la flecha
 }

 else                               // La variable pos_flecha no es menor que 1 (es mayor o igual
                                       que 1 (aún no se han movido los mensajes))
 {
  lcd.setCursor(0, pos_flecha);     // Se coloca el cursor en la posición deseada para escribir la
                                       flecha
  lcd.write(1);                     // Se escribe la flecha en la linea pos_flecha
 }

 if ( pos_flecha == 3)              // La variable pos_flecha es 3
 {
  lcd.clear();
  for (int i = 1; i <= 3; i++)      // Se escriben los mensajes del 1 al 3
  {
   lcd.setCursor(0, 0);
   lcd.write (opciones[pantalla][0]);
   lcd.setCursor(0, i);
   lcd.write(opciones[pantalla][i]);
```

```
      }

      lcd.setCursor(0, pos_flecha);   // Se coloca el cursor en 0,0
      lcd.write(1);                   // Se escribe la flecha
     }

     else if ( pos_flecha == 4)       // La variable pos_flecha es = 4
     {
      lcd.clear();
      for (int i = 1; i <= 3; i++)    // Se escriben los mensajes del 2 al 4
      {
       lcd.setCursor(0, 0);
       lcd.write (opciones[pantalla][0]);
       lcd.setCursor(0, i);
       lcd.write(opciones[pantalla][i + 1]);
      }

      lcd.setCursor(0, pos_flecha - 1);    // Se coloca el cursor en 0,0
      lcd.write(1);                   // Se escribe la flecha
     }
    }


    if (pantalla == 2)          // ********* Comportamiento en la pantalla 2 ********
    {
     if (pos_flecha < 1 )           // Si pos_flecha es menor que uno (nos salimos de la zona de
                                        mensajes por arriba)
     {
      pos_flecha = 1;               // pos_flecha se pone a 1 para acotar su valor (1 como mínimo)
      lcd.setCursor(0, pos_flecha); // Se sitúa el cursor en 0,pos_flecha
      lcd.write(1);                 // Se escribe la flecha
     }
     else                           // Si pos_flecha no es menor que 1 (solo puede valer 1)
     {
      lcd.setCursor(0, pos_flecha); // Se sitúa el cursor en la columna cero fila pos_flecha
      lcd.write(1);                 // Se escribe la flecha
     }
    }
   }
}
```

```
//****************** CONFIGURACION TIPO DE PASO **********************

void config_tipo_paso()
{
 if (tipo_paso == 1)              // Configuración para paso completo
 {
  digitalWrite(MS1, LOW);         // MS1 = 0
  digitalWrite(MS2, LOW);         // MS2 = 0
  digitalWrite(MS3, LOW);         // MS3 = 0
 }

 else if (tipo_paso == 2)         // Configuración para medio paso
 {
  digitalWrite(MS1, HIGH);        // MS1 = 1
  digitalWrite(MS2, LOW);         // MS2 = 0
  digitalWrite(MS3, LOW);         // MS3 = 0
 }

 else if (tipo_paso == 3)         // Configuración para 1/4 de paso
 {
  digitalWrite(MS1, LOW);         // MS1 = 0
  digitalWrite(MS2, HIGH);        // MS2 = 1
  digitalWrite(MS3, LOW);         // MS3 = 0
 }

 else if (tipo_paso == 4)         // Configuración para 1/8 de paso
 {
  digitalWrite(MS1, HIGH);        // MS1 = 1
  digitalWrite(MS2, HIGH);        // MS2 = 1
  digitalWrite(MS3, LOW);         // MS3 = 0
 }

 else if (tipo_paso == 5)         // Configuración para 1/16 de paso
 {
  digitalWrite(MS1, HIGH);        // MS1 = 1
  digitalWrite(MS2, HIGH);        // MS2 = 1
  digitalWrite(MS3, HIGH);        // MS3 = 1
 }
}
```

```
//****************** LECTURA NUMEROS DEL TECLADO ********************

void lee_num_tec()                  // Subrutina de lectura de los números del teclado
{

 tecla_pulsada = tecla - 48;        // Se resta 48 para pasar de ASCII a decimal

 if (num_puls == 2)                 // Si ha habido dos pulsaciones
 {
  valor_3 = tecla_pulsada;          // Se almacena el valor de la tecla pulsada en la variable
valor_3
  valor_tec = (valor_1 * 100) + (valor_2 * 10) + valor_3;    // Se aplican las operaciones necesarias
                                                               para que el valor final sea correcto
  lcd.setCursor(0, 1);              // Se escribe en el LCD el valor introducido
  lcd.print(valor_tec);
  num_puls++;                       // Se incrementa el número de pulsaciones
 }

 if (num_puls == 1)                 // Si ha habido una pulsación
 {
  valor_2 = tecla_pulsada;          // Se almacena el valor de la tecla pulsada en la variable
valor_2
  valor_tec = (valor_1 * 10) + valor_2;      // Se aplican las operaciones necesarias para que el
                                               valor final sea correcto
  lcd.setCursor(0, 1);              // Se escribe en el LCD el valor introducido
  lcd.print(valor_tec);
  num_puls++;                       // Se incrementa el número de pulsaciones
 }

 if (num_puls == 0)                 // Si no ha habido pulsaciones
 {
  valor_1 = tecla_pulsada;          // Se almacena el valor de la tecla pulsada en la variable
                                       valor_1
  valor_tec = valor_1;              // El valor final introducido es igual a valor_1
  lcd.setCursor(0, 1);              // Se escribe en el LCD el valor introducido
  lcd.print(valor_tec);
  num_puls++;                       // Se incrementa el número de pulsaciones
 }

 if (num_puls == 3)                 // Si ha habido tres pulsaciones
 {
  num_puls = 0;                     // Se pone a cero el número de pulsaciones
  valor_1 = valor_2 = valor_3 = 0;  // Se ponen a cero las variables valor_1, valor_2 y valor_3
 }
}
```

```
//************************ PASO UNICO ****************************

void paso_unico()                    // Subrutina de paso unico
{

 if (num_pul_act == -1)              // Si aún no se ha dado ningún paso
 {
  num_pul_act++;                     // Se incrementa la variable num_pul_act (número de pulsos
                                        actual) en una unidad
  lcd.clear();
  lcd.setCursor(0, 0);               // Se escribe la información del ensayo en el LCD
  lcd.write("Numero de pasos: 0");
  lcd.print(num_pul_act, DEC);
  lcd.setCursor(0, 1);
  lcd.write("Grados girados: 0");
  lcd.print(grados_girados, DEC);
 }

 else
 {
  digitalWrite(sleep, HIGH);
  estado_CLK = HIGH;                 // Se pone el estado_CLK en estado alto (primera parte del
                                        paso)
  digitalWrite(CLK, estado_CLK);     // Se envía el valor de estado_CLK al pin CLK
  estado_CLK = LOW;                  // Se pone el estado_CLK en estado bajo (segunda parte del
                                        paso)
  digitalWrite(CLK, estado_CLK);     // Se envía el valor de estado_CLK al pin CLK
  num_pul_act++;                     // Se incrementa la variable num_pul_act (número de pulsos
                                        actual) en una unidad

  cuentaencoder = lee_encoder(1);
  grados_girados = (cuentaencoder * 360 / 16384);       // Se convierte el número de pulsos del
                                                           encoder en grados

  lcd.clear();                       // Se escribe la información del ensayo en el LCD
  lcd.setCursor(0, 0);
  lcd.write("Numero de pasos: ");
  lcd.print(num_pul_act, DEC);
  lcd.setCursor(0, 1);
  lcd.write("Grados girados: ");
  lcd.print(abs(grados_girados), DEC);
  //digitalWrite(sleep,HIGH);
 }
}
```

```
//****************************** ATRAS ******************************

void atras()
{

 if (pantalla == 1 || pantalla == 2)          // Estamos en las dos primeras pantallas
 {
  pantalla--;                          // Se decrementa la variable pantalla en una unidad
  pos_flecha = -1;                     // Se inicializa la variable pos_flecha
  lcd.clear();                         // Se borra el LCD
  for (int i = 0; i <= 3; i++)         // Se escriben los mensajes de la pantalla correspondiente en
                                           el LCD
  {
   lcd.setCursor(0, i);
   lcd.write(opciones[pantalla][i]);
  }
 }

 else if (pantalla == 3 || pantalla == 4)       // Estamos en las pantallas de introducción de datos
                                                    numéricos
 {
  if (valor_1 != 0)                    // Se ha escrito algún numero
  {
   valor_1 = valor_2 = valor_3 = valor_tec = 0;   // Se inicializan las variables relacionadas con los
                                                      números del teclado
   lcd.setCursor(0, 1);                // Se escribe en el LCD el valor introducido
   lcd.print(valor_tec);
   num_puls = 0;                       // Se inicializa la variable num_puls
  }

  else                                 // No se ha escrito ningún numero
  {
   pantalla--;                         // Se decrementa la variable pantalla en una unidad
   pos_flecha = -1;                    // Se inicializa la variable pos_flecha
   lcd.clear();                        // Se borra el LCD
   lcd.write(introduccion_datos[modo_trabajo - 1]);     // Se escriben los mensajes de la pantalla
                                                            correspondiente en el LCD
  }
 }
}
```

```
//************************ NUEVO ENSAYO ****************************

void nuevo_ensayo()
{

 int modo_trabajo = 0;          // 1->LAZO CERRADO   2->LAZO ABIERTO   3->PASO UNICO
 int tipo_paso = 2;             // 1->PASO COMPLETO   2->MEDIO PASO   3->1/4 DE PASO   4-
                                   >1/8 DE PASO   5->1/16 DE PASO
 int sentido_giro = 1;          // 1->SENTIDO ANTIHORARIO   2->SENTIDO HORARIO
                                   (inicialmente no tiene ningún valor)
 int frec = 10;                 // Frecuencia a la que se desea que gire el motor
 int grados_girados = 0;        // Grados que ha girado el motor
 int grados_deseados = 0;       // Grados que se desea que gire el motor
 int num_pul_act = -1;          // Numero de pasos que ha girado el motor
 int num_pul_des = 0;           // Numero de pasos que se desea que gire el motor
 bool estado_CLK = false;       // Estado del reloj que marca los pasos del motor
 int pantalla = 0;              // Indica la pantalla en la que nos encontramos
 int pos_flecha = -1;           // pos_flecha toma valores entre 0 y 3, y empieza en -1 para que
                                   inicialmente no aparezca
 char tecla = '0';              // Variable en la que se almacena la tecla del teclado pulsada. Se
                                   inicia con Z para que no haya ninguna tecla pulsada
 int valor_tec = NULL;          // Variable en la que se almacena el valor numérico introducido
                                   por teclado
 int valor_1 = NULL;            // Variable en la que se almacena el valor numérico de la primera
                                   tecla pulsada
 int valor_2 = NULL;            // Variable en la que se almacena el valor numérico de la segunda
                                   tecla pulsada
 int valor_3 = NULL;            // Variable en la que se almacena el valor numérico de la tercera
                                   tecla pulsada
 int tecla_pulsada = NULL;      // Variable que indica la tecla que se ha pulsado
 int num_puls = NULL;           // Variable que indica el número de pulsaciones que se han
                                   realizado
 signed long cuentaencoder = 0;     // Variable donde se almacena el contaje del encoder (debe
                                       ser de tipo long por la gran cantidad de datos a almacenar)
 int estado_enable = LOW;       // Indica el estado del pin enable del controlador del motor
 int estado_reset = HIGH;       // Indica el estado del pin reset del controlador del motor

 Timer6.stop();                 // Se detiene el contador que llama a la subrutina de giro del
                                   motor

 lcd.clear();                   // Se borra el LCD
 for (int i = 0; i <= 3; i++)   // Se escriben los mensajes de la pantalla correspondiente en el
                                   LCD
 {
  lcd.setCursor(0, i);
  lcd.write(opciones[pantalla][i]);
 }
```

```
  digitalWrite(MS1, LOW);             // Se inicializa el controlador sin ningún tipo de paso
                                         seleccionado (MS1=0, MS2=0 y MS3=1 no es nada)
  digitalWrite(MS2, LOW);
  digitalWrite(MS3, LOW);
  digitalWrite(dir, sentido_giro);    // El pin de sentido de giro se inicializa con el valor de la
                                         variable sentido_giro
  digitalWrite(enable, estado_enable);  // El pin de enable se inicializa con el valor de la variable
                                         estado_enable
  digitalWrite(reset, estado_reset);    // El pin de reset se inicializa con el valor de la variable
                                         estado_reset
}


//*****************************************************************
//******************** PROGRAMA PRINCIPAL ************************
//*****************************************************************

void loop() {

  // Esta parte se tiene ejecutar aunque no se pulsa ninguna tecla

  if (pantalla == 5)                  // Si se está en la pantalla 5 (Muestra los parámetros del test)
  {
   if (modo_trabajo == 1)             // Si se trabaja en lazo cerrado
   {
    lcd.clear();                      // Se borra el LCD
    for (int i = 0; i <= 3; i++)      // Se escriben los mensajes de la pantalla correspondiente en el
                                         LCD
    {
     lcd.setCursor(0, i);
     lcd.write(resumen_lazo_cerrado[i]);
     pantalla++;                      // Se incrementa la variable pantalla en una unidad
    }
   }

   if (modo_trabajo == 2)             // Si se trabaja en lazo abierto
   {
    lcd.clear();                      // Se borra el LCD
    for (int i = 0; i <= 3; i++)      // Se escriben los mensajes de la pantalla correspondiente en el
                                         LCD
    {
     lcd.setCursor(0, i);
     lcd.write(resumen_lazo_abierto[i]);
     pantalla++;                      // Se incrementa la variable pantalla en una unidad
    }
   }
  }
```

```
tecla = keypad.getKey();          // Cuando se pulsa una tecla, se guarda en la variable tecla
if (tecla != NO_KEY) {            // Si se pulsa una tecla cualquiera

 if (tecla == '1' || tecla == '2' || tecla == '3' || tecla == '4' || tecla == '5' || tecla == '6' || tecla ==
'7' || tecla == '8' || tecla == '9' || tecla == '0')  //******* SE PULSA UN NUMERO *******
 {
  if (pantalla == 4)              // Si se está en la pantalla 4 (introducción de la frecuencia)
  {
   lee_num_tec();                 // Se leen los números del teclado
  }

  else if (pantalla == 3 & modo_trabajo != 3)    // Si se está en la pantalla 3 (introducción grados
                                                    o número de pasos) en lazo abierto o cerrado
  {
   lee_num_tec();                 // Se lee el teclado
  }
 }

 if (tecla == 'F')                //********* SE PULSA FLECHA ABAJO **********
 {
  if (pantalla == 0 || pantalla == 1 || pantalla == 2)      // Los mensajes solo se mueven en las
                                                               tres primeras pantallas
  {
   pantalla_abajo();              // Se llama a la subrutina pantalla_abajo
  }
 }

 if (tecla == 'B')                //********* SE PULSA FLECHA ARRIBA *********
 {
  if (pantalla == 0 || pantalla == 1 || pantalla == 2)     // Los mensajes solo se mueven en las tres
                                                              primeras pantallas
  {
   pantalla_arriba();             // Se llama a la subrutina pantalla_arriba
  }
 }

 if (tecla == 'D')                //************** SE PULSA ENTER ************
 {

  if (pantalla == 2)              // Si se está en la pantalla 2 (sentido de giro)
  {
   sentido_giro = pos_flecha - 1;          // El valor de pos_flecha se almacena en la variable
                                              sentido giro
   digitalWrite(dir, sentido_giro);        // El pin de sentido de giro se inicializa con el valor de la
                                              variable sentido_giro
   pantalla++;                    // Se incrementa la variable pantalla en una unidad para pasar de
                                     pantalla
```

```
  pos_flecha = -1;              // Se pone pos_flecha a -1 para iniciar la pantalla sin que aparezca
                                   la flecha
 }

 if (pantalla == 1)             // Si se está en la pantalla 1 (Tipo de paso)
 {
  tipo_paso = pos_flecha;       // El valor de pos_flecha se almacena en la variable tipo_paso
  config_tipo_paso();           // Se configura el tipo de paso seleccionado
  pantalla++;                   // Se incrementa la variable pantalla en una unidad para pasar de
                                   pantalla
  pos_flecha = -1;              // Se pone pos_flecha a -1 para iniciar la pantalla sin que aparezca
                                   la flecha
 }

 if (pantalla == 3 & modo_trabajo == 3)       // Si estamos en la pantalla 3 y en modo paso único
 {
  paso_unico();                 // Se llama a la subrutina paso_unico
 }

 if (pantalla == 3 & modo_trabajo == 2)       // Se está en la pantalla 3 (introducción grados o
                                                 número de pasos) y en lazo abierto
 {
  lcd.clear();                  // Se limpia el LCD
  lcd.setCursor(0, 0);          // Se coloca el cursor en 0,0
  lcd.write(introduccion_datos[1]);          // Se escribe el mensaje 1 de la lista
                                                introduccion_datos (Posición en grados:)
  lcd.setCursor(0, 1);          // Se coloca el cursor en 0,1 para introducir el dato

  if (valor_tec != 0)           // Si el valor introducido por teclado no es cero
  {
   num_pul_des = valor_tec;          // Se guarda el valor introducido en la variable
                                        num_pul_des
   valor_1 = valor_2 = valor_3 = 0;  // Se ponen a cero las variables de introducción de datos
                                        por teclado
   valor_tec = 0;               // Se pone a cero la variable valor_tec
   num_puls = 0;                // Se pone a cero la variable num_puls
   pantalla++;                  // Se incrementa la variable pantalla en una unidad
  }
 }

 if (pantalla == 3 & modo_trabajo == 1)       // Se está en la pantalla 3 (introducción grados o
                                                 número de pasos) y en lazo cerrado
 {


lcd.clear();                    // Se limpia el LCD
    lcd.setCursor(0, 0);        // Se coloca el cursor en 0,0
```

```
  lcd.write(introduccion_datos[0]);      // Se escribe el mensaje 0 de la lista introduccion_datos
                                            (Numero de pasos:)
  lcd.setCursor(0, 1);                  // Se coloca el cursor en 0,1 para introducir el dato

  if (valor_tec != 0)                   // Si el valor introducido por teclado no es cero
  {
   grados_deseados = valor_tec;             // Se guarda el valor introducido en la variable
                                               grados_deseados
   valor_1 = valor_2 = valor_3 = 0;         // Se ponen a cero las variables de introducción de
                                               datos por teclado
   valor_tec = 0;                   // Se pone a cero la variable valor_tec
   num_puls = 0;                    // Se pone a cero la variable num_puls
   pantalla++;                      // Se incrementa la variable pantalla en una unidad
  }
 }

 if (pantalla == 4)                 // Si se está en la pantalla 4 (introducción de la frecuencia)
 {

  lcd.clear();                      // Se limpia el LCD
  lcd.setCursor(0, 0);              // Se coloca el cursor en 0,0
  lcd.write(introduccion_datos[2]);         // Se escribe el mensaje 2 de la lista
                                               introduccion_datos (Frecuencia:)
  lcd.setCursor(0, 1);              // Se coloca el cursor en 0,1 para introducir el dato
  if (valor_tec != 0)               // Si el valor introducido por teclado no es cero
  {
   frec = valor_tec;                // Se guarda el valor introducido en la variable frec
   Timer6.attachInterrupt(giro_motor).setFrequency(frec).start();   // Se llama a la subrutina
                                                                       giro_motor frec veces por segundo
   valor_1 = valor_2 = valor_3 = 0;         // Se ponen a cero las variables de introducción de
                                               datos por teclado
   valor_tec = 0;                   // Se pone a cero la variable valor_tec
   num_puls = 0;                    // Se pone a cero la variable num_puls
   pantalla++;                      // Se incrementa la variable pantalla en una unidad
  }
 }

 if (pantalla == 0)                 // Si se está en la pantalla 0 (selección del modo de trabajo)
 {
  modo_trabajo = pos_flecha;    // El valor de pos_flecha se almacena en la variable
                                   modo_trabajo
  pantalla++;                   // Se incrementa la variable pantalla en una unidad para pasar
                                   de pantalla
  pos_flecha = -1;              // Se pone pos_flecha a -1 para iniciar la pantalla sin que
                                   aparezca la flecha
 }
 if (pantalla <= 2)             // Si pantalla en menor o igual que 2 (estamos en las pantallas
                                   de selección con varias opciones)
```

```
    {
      lcd.clear();                    // Se borra el LCD
      for (int i = 0; i <= 3; i++)    // Se escriben los mensajes de la pantalla correspondiente en
                                          el LCD
      {
        lcd.setCursor(0, i);
        lcd.write(opciones[pantalla][i]);
      }
    }
  }

  if (tecla == 'R')                   // *********** SE PULSA RESET ************
  {
    digitalWrite(reset_arduino, LOW);     // Se pone el pin reset (conectado a la entrada de reset)
                                              a cero para hacer un reset
  }

  if (tecla == 'E')                   // ************ SE PULSA ATRAS ***********
  {
    atras();                          // Se llama a la subrutina atrás
  }

  if (tecla == 'N')                   // ******** SE PULSA NUEVO ENSAYO *******
  {
    nuevo_ensayo();                   // Se llama a la subrutina nuevo_ensayo
    pantalla = 0;                     // Se pone a cero la variable pantalla
  }
 }
}
```

8.- Modo de empleo:

En este apartado se va a explicar cómo manejar el entrenador de motores paso a paso que se ha diseñado en este proyecto.

La interacción del equipo con el usuario se realiza mediante un display LCD y un teclado. Mediante el teclado se van seleccionando las distintas opciones que se muestran en la pantalla o introduciendo los datos numéricos necesarios para configurar el ensayo deseado. Por último, una vez que el ensayo está en marcha, se muestra una pantalla con la información del ensayo que está teniendo lugar.

En el teclado del equipo podemos encontrar un teclado numérico, un enter, una flecha de arriba y otra de abajo, una flecha de retroceso, una tecla de nuevo ensayo y otra de reset del equipo. El teclado tiene el siguiente aspecto.

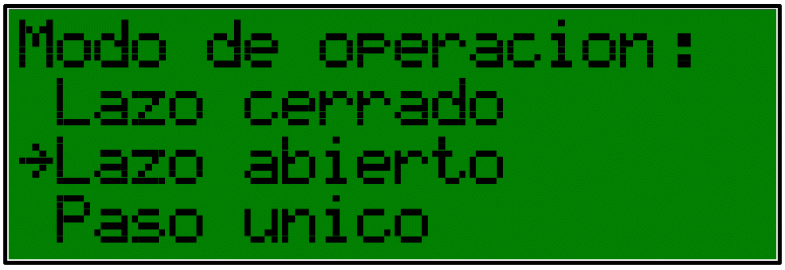
Teclado.

Al encender el equipo, lo primero que aparece en el display es una pantalla de presentación como la que se puede ver en la imagen.

Pantalla inicial.

Pasados cinco segundos, aparecerá otra pantalla en la que se selecciona el modo de trabajo deseado para el ensayo, en la que las opciones son las siguientes:

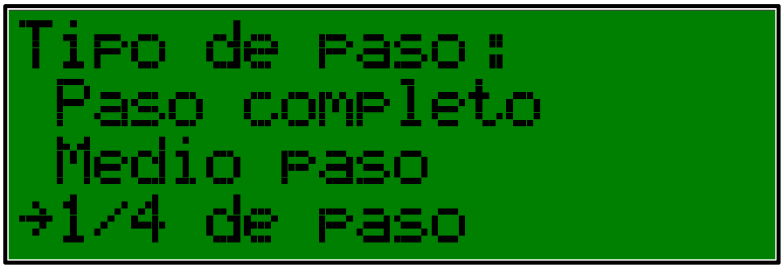
- Lazo cerrado: Se seleccionan los grados que se desea que gire el motor.
- Lazo abierto: Se selecciona el número de pasos que se desea que gire el motor.
- Paso único: Con cada pulsación de la tecla enter, el motor gira un paso.

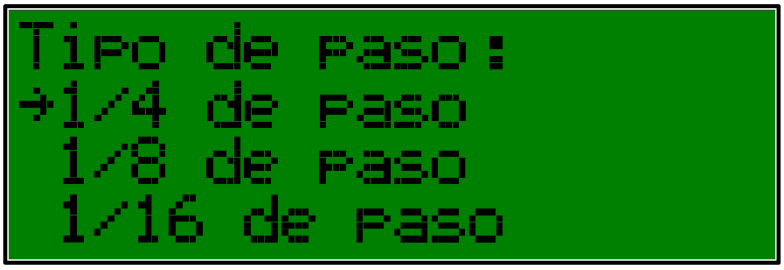


Pantalla de selección de modo de trabajo.

Al pulsar las teclas de flecha arriba o abajo, la flecha que aparece en el margen izquierdo de la pantalla se desplazara en la dirección indicada, y al pulsar enter se selecciona la opción marcada.

Una vez seleccionada una opción, la pantalla cambiará y pasara a mostrar la pantalla de selección de tipo de paso, donde podremos elegir entre los distintos tipos de paso que es capaz de dar el entrenador, siendo las opciones paso completo, medio paso, un cuarto de paso, un octavo de paso y un dieciseisavo de paso.

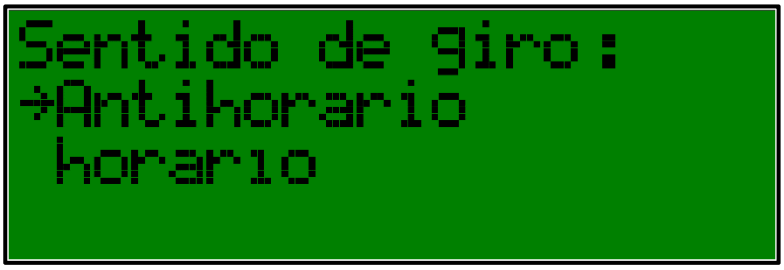




Pantalla de selección de tipo de paso.

En este caso, como en el anterior, nos desplazaremos entre las distintas opciones existentes mediante las flechas, y se valida la opción seleccionada mediante la tecla enter.

La siguiente pantalla que se mostrará será la de selección de sentido de giro, pudiendo seleccionar sentido horario o anti horario.



Pantalla de selección de sentido de giro.

Llegados a este punto, las siguientes pantallas en mostrarse dependerán del modo de trabajo que se haya seleccionado. A continuación se indican cuáles son esas pantallas, así como su contenido y funcionamiento.

8.1.- Lazo cerrado:

Como se ha indicado anteriormente, en este modo de trabajo se introducen los grados que se desea que gire el motor, y como no se detiene el motor hasta que se ha alcanzado la posición deseada, aunque se pierdan pasos, la posición pedida se va a alcanzar siempre.

Pantalla de introducción de los grados a girar.

El primer parámetro que se pide al usuario es la posición en grados que se desea alcanzar. Este dato se introduce mediante el teclado numérico y se valida con la tecla enter.

A continuación se solicita al usuario la frecuencia de giro del motor. Este proceso es igual que el caso anterior (introducción de grados a girar).

Pantalla de introducción de la frecuencia de giro del motor.

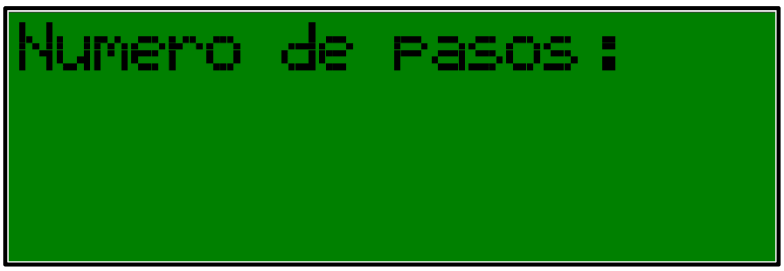
Al pulsar la tecla enter, el ensayo se iniciara y se mostrara una pantalla con un resumen de las opciones seleccionadas y de los datos introducidos, así como los resultados del ensayo. Esta pantalla puede verse en la siguiente imagen.

Pantalla de resumen del ensayo en lazo cerrado.

8.2.- Lazo abierto:

En este modo de trabajo se selecciona el número de pasos que se desea que gire el motor. En este modo se puede ver como a frecuencias elevadas, no coinciden los grados girados con los que teóricamente debe girar el motor. Esto se debe a que a frecuencias elevadas, el motor pierde pasos y por lo tanto gira menos de lo que debería.

Para este caso, el primer parámetro que debe introducir el usuario es los números de paso que se desea que gire el motor. Este proceso es igual que la introducción de la frecuencia o de los grados que se quiere que gire el motor.

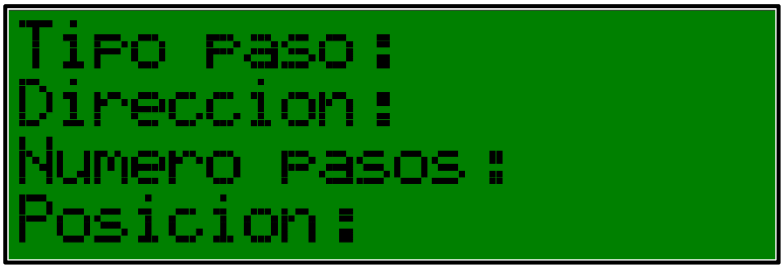


Pantalla de introducción del número de pasos a girar.

A continuación se solicita al usuario la frecuencia de giro del motor. Este proceso es igual que el caso anterior (introducción del número de pasos a girar).

Pantalla de introducción de la frecuencia de giro del motor.

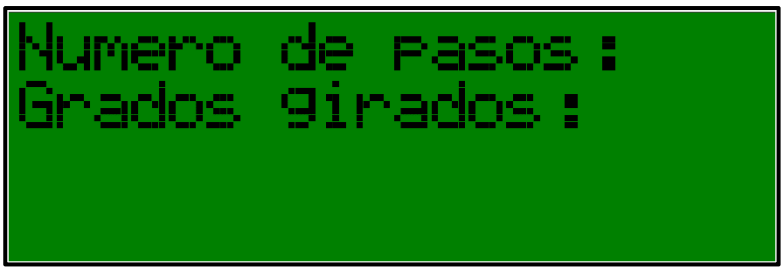
Al igual que en el caso anterior, se inicia el ensayo al pulsar la tecla enter, y en la pantalla se mostrara un resumen de las opciones elegidas y de los resultados del ensayo. Esta pantalla puede verse en la siguiente imagen.



Pantalla de resumen del ensayo en lazo abierto.

8.3.- Paso único:

En este modo de trabajo, cada vez que se pulsa la tecla enter el motor gira un paso. Mientras se está en este modo, siempre está visible en el LCD una pantalla en la que se indican los pasos que se han dado y los grados que ha girado el motor. Esta pantalla puede verse en la siguiente imagen.



Pantalla de resumen del ensayo en paso único.

9.- Problemas encontrados y soluciones adoptadas:

9.1.- Raspberry Pi:

Una Raspberry Pi es un ordenador integrado en una solo placa y de tamaño y coste muy reducido. Este ordenador incluye una CPU de ARM a 700 MHz, y usa un sistema operativo basado en Linux (los más usados son Raspbian y Fedora). Este diseño no incluye disco duro, ya que usa una tarjeta de memoria SD para almacenar toda la información y el sistema operativo, tampoco incluye fuente de alimentación ni carcasa.

En un principio se pensó basar el funcionamiento del equipo en una Raspberry Pi tipo B en vez de en Arduino, ya que se pensó que la posibilidad de conectar un monitor directamente a la Raspberry Pi facilitaría la interacción entre el usuario y el equipo. Pero una vez que se inició el desarrollo aparecieron varios problemas que hicieron que se decidiese cambiar la Raspberry Pi por Arduino.

Los problemas aparecidos fueron los siguientes:

- El número de entradas y salidas resultaba insuficiente, ya que en un principio este dispositivo solo cuenta con dieciocho pines de entrada/salida, ampliables a veintidós si se suelda un conector.
- El manejo de las entradas/salidas no es tan eficiente como en Arduino, ya que la Raspberry Pi no está pensada para aplicaciones de tanta precisión como la generación de un reloj.
- La Raspberry Pi se programa en Python. Este lenguaje es interpretado, lo que hace quesea más lento que un lenguaje compilado, como el ANSI C que usa Arduino. Otro inconveniente de este tipo de lenguaje es que no se puede calcular el tiempo que se tarda en ejecutar una instrucción, haciendo ejecutar generar instrucciones periódicas con precisión.

9.2.- Generación de los pulsos de control del motor:

En un principio se pensó en generar la secuencia de control del motor paso a paso mediante cuatro salidas de la placa Arduino para hacer que los transistores de los puentes en H pasen a conducir o dejen de hacerlo.

Esta idea se acabó desechando por dos razones:

- La cantidad de instrucciones necesarias para generar esta secuencia y la necesidad de leer el encoder al mismo tiempo, haría que se pudiesen perder pasos fácilmente a frecuencias no muy elevadas.
- Mediante este sistema solo se podía conseguir una precisión de medio paso, lo que limitaba bastante las posibilidades del motor paso a paso.

Debido a estos problemas, se ha decidido incluir hardware externo que se encargue de realizar esta función, y aunque se necesiten más de cuatro salidas, los beneficios obtenidos son claramente superiores a los perjuicios. Como ya se ha dicho anteriormente, este hardware externo es el circuito integrado A4988 de Allegro.

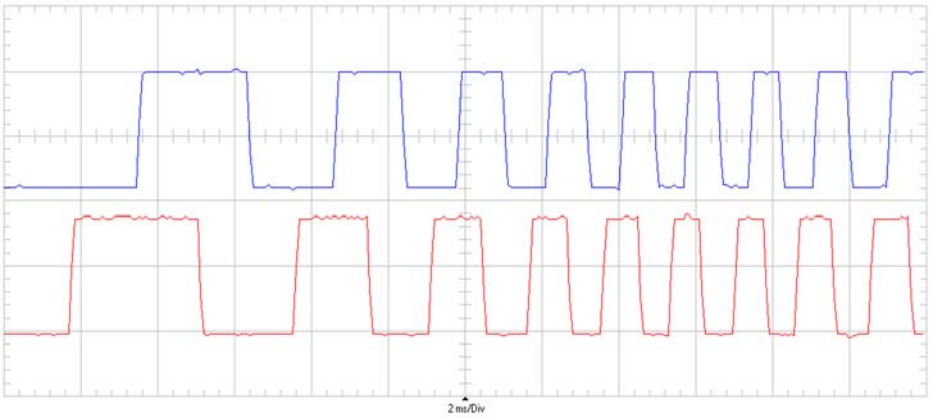
Otro de los beneficios de incluir este circuito impreso es que como ya se ha dicho anteriormente, no es necesario incluir los dos puentes en H necesarios para el correcto funcionamiento del equipo, ya que están incluidos en el controlador del motor.

9.3.- Encoder:

El primer encoder que se escogió contaba tan solo con 200 marcas, que teniendo en cuenta la codificación de la cuadratura, se convertían en un total de 800 marcas por vuelta. Observando el número de pulsos del motor trabajando en micro pulsos, la cantidad de marcas del encoder se hace totalmente insuficiente.

Otro problema con el encoder es la tensión de alimentación, ya que este trabaja a 5V y el Arduino due lo hace a 3.3V, por lo que es necesario incluir una fuente de alimentación de 5V en el equipo.

En el momento de conectar el encoder a la PCB final, apareció un problema y es que no se leían los pulsos del encoder, por lo que se empezó a buscar el fallo para ponerle solución. Lo primero que se hizo para buscar el problema fue conectar el encoder al osciloscopio para comprobar la integridad de las señales A y B que debe proporcionar, obteniéndose los resultados que se pueden ver en la siguiente imagen.

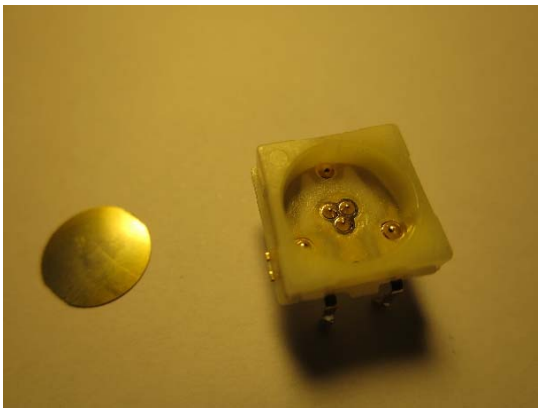


Lectura del encoder.

Como se puede ver en la imagen, las señales A y B tiene la forma adecuada (cuadrada) y están desfasadas entre si 90º. Una comprobado el correcto funcionamiento del encoder, se sigue buscando el origen del problema, para lo que sin tensión se mide la continuidad de las pistas de circuito impreso mediante el polímetro en función continuidad, detectándose que la pista que une el pin SCK de la placa Arduino con el pin SCK del integrado LS7366R estaba cortada. Una vez detectado el problema, se ha arreglado soldando un hilo conductor que puentee la zona dañada de la pista. Con este fallo ya corregido, el encoder cuanta los pulsos correctamente.

Por último, aparecieron problemas a la hora de configurar el circuito integrada LS7366R encargado del contaje del encoder, ya que el manejo de la librería SPI es distinta en Arduino due que en el resto de modelos de Arduino. Esta diferenciación no se tuvo en cuenta inicialmente, dando lugar a un gran número de errores de compilación y funcionamiento. Esto problemas se resolvieron al usar la sintaxis correcta para la placa de Arduino utilizada en el proyecto.

9.4.- Teclado:



Pulsador del teclado desmontado

Como ya se ha dicho anteriormente, el teclado utilizado en este proyecto ha sido recuperado de un plotter antiguo. Este hecho hizo que la mayoría de las teclas no funcionasen correctamente.

Para solucionar este contratiempo, se desoldaron todos los pulsadores del teclado y se desmontaron con el objetivo de ver su funcionamiento y repararlos en caso de ser posible. El funcionamiento de estos pulsadores es muy sencillo, ya que al pulsarlos se une la parte interior de un circulo (un contacto) con la exterior del mismo (otro contacto) mediante una chapa circular y convexa llamada comúnmente rana. En la imagen de la derecha, se pueden ver la chapa redonda (rana) y la parte de plástico donde se coloca esta chapa y donde se encuentran los contactos del pulsador.

Una vez desmontados, se vio que el único problema existente era la suciedad que se encontraba en su interior, y que impedía la correcta conducción a través de la "rana", así que se procedió a limpiar ambas partes de los pulsadores y a soldarlos de nuevo al circuito impreso del teclado.

Una vez realizada esta tarea de mantenimiento, se comprobó de nuevo el funcionamiento del teclado, obteniendo esta vez un resultado positivo.

9.5.- LCD:

Inicialmente se utilizó un display LCD de 4x16. Esta pantalla tenía el inconveniente de que no funcionaba correctamente con la librería "lyquid cristal" incluida en el software de desarrollo de Arduino.

El problema era que las dos líneas inferiores comenzaban en la columna -4 en lugar de en la cero, por lo que había que tener especial cuidado de la línea en la que se iba a escribir para ajustar correctamente el inicio de la línea.

Finalmente esta pantalla se sustituyó por una de 4x20 en la que este problema no aparece, haciendo así mucho más fácil su uso.

9.6.- Reloj de pulsos:

Para que el driver encargado de controlar el motor paso a paso funcione correctamente, es necesario generar un reloj lo más preciso posible de la frecuencia a la que deseamos que se produzcan los pasos.

La primera idea fue poner una salida a uno, esperar el tiempo deseado mediante la instrucción "delay" y ponerla a cero de nuevo, repitiendo este proceso las veces que fueran necesarias. Pero esta instrucción tiene el inconveniente de que mientras se está ejecutando, el microcontrolador no realiza ninguna otra acción.

El siguiente paso fue generar ese retardo mediante comparaciones del valor de duración del pulso con un contador interno que se pone a cero cada vez que se produce un reset. De esta forma se consigue el mismo efecto que ante pero pudiendo ejecutar otras instrucciones a la vez. Este método tiene el problema de que si el programa es muy largo o entra en otro bucle, se pueden perder pulsos de reloj, haciendo que el funcionamiento no sea el deseado.

La solución definitiva a este problema pasa por generar el reloj mediante interrupciones. De esta forma, se generará el pulso de reloj sin importar el punto del programa en el que nos encontremos. Para generar estas interrupciones de una forma sencilla, se usa la librería DueTimer.

9.7.- PCB:

Este proyecto incluye el diseño de una placa de circuito impreso (PCB), pero se han encontrado limitaciones a la hora de realizar este diseño debido a que el proceso de fabricación al que se ha tenido acceso en los laboratorios de la universidad, impide realizar placas de circuito impreso de las dimensiones necesarias. Por este motivo se ha realizado la placa de circuito impreso con los medios de los que dispongo, consiguiendo un resultado bastante bueno.

Aun habiendo intentado situar todas las pistas en la cara superior de la PCB, no ha sido posible, por lo que ha sido necesario colocar tres pequeños puentes para unir pistas que ha sido imposible unir mediante una pista impresa por cruzarse con otras pistas.

En un principio se realizó un diseño de PCB de unas dimensiones algo elevadas (206 mm x 130 mm), por lo que se decidió realizar un segundo diseño en el que se han agrupado más los componentes y reubicados algunos de ellos para reducir el tamaño de la PCB hasta los 150 mm de ancho por 112 mm de alto, lo que implica una reducción de aproximadamente el 37% de superficie.

9.7.1.- Fabricación de la PCB:

Como se ha dicho anteriormente, para este proyecto se ha realizado una placa de circuito impreso con los medios de los que he podido disponer fuera de la universidad.

La fabricación de la PCB ha constado de dos partes, la de diseño, que se ha realizado mediante el software Altium Designer 15, y la fase de fabricación.

En la fase de diseño, fue necesario crear casi la mayoría de los componentes del proyecto, ya que a excepción de las resistencias, el resto de componentes no aparecían en las librerías incorporadas en el software de diseño. Posteriormente se han dibujado los esquemas que reflejan la interconexión entre todos los componentes, y por último se ha realizado la colocación de los componentes en la placa y su conexión automática mediante pistas de circuito impreso de los componentes. El software se configuró para trazar pistas solo por una cara de la PCB, pero como no fue posible, se realizaron tres puentes de forma manual.

El siguiente paso en la fabricación de la placa de circuito impreso fue imprimir las pistas de la placa en papel vegetal a escala 1:1, es importante que las pistas aparezcan lo más negras (opacas) posible, por motivos que veremos a continuación.

Para la fase de fabricación se han utilizado los siguientes materiales:

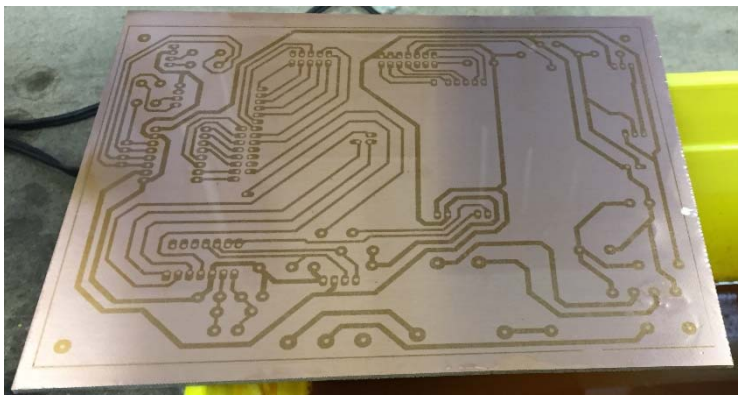
- Insolador.
- Hidróxido de sodio (sosa caustica).
- Cloruro Férrico.
- Agua.
- Placa de circuito impreso positiva virgen.
- Pistas de circuito impreso en papel vegetal.
- Dos cubetas para mezclar los componentes químicos.

El primer paso consiste en insolar la placa de circuito impreso positiva virgen. Para ello se coloca la PCB impresa en papel vegetal sobre el cristal del insolador, y encima del papel vegetal la placa de circuito impreso positiva virgen. Se cierra la tapa del insolador y se deja funcionar durante dos minutos. Este tiempo es bastante importante, ya que tanto un exceso como un defecto de insolación estropearían el proceso.

Para la realización dela PCB de este proyecto, se ha utilizado un insolador de fabricación casera, que simplemente consta de dos tubos de luz ultravioleta y un temporizador que desconecta el equipo una vez transcurrido el tiempo de insolación deseado.

Insolador

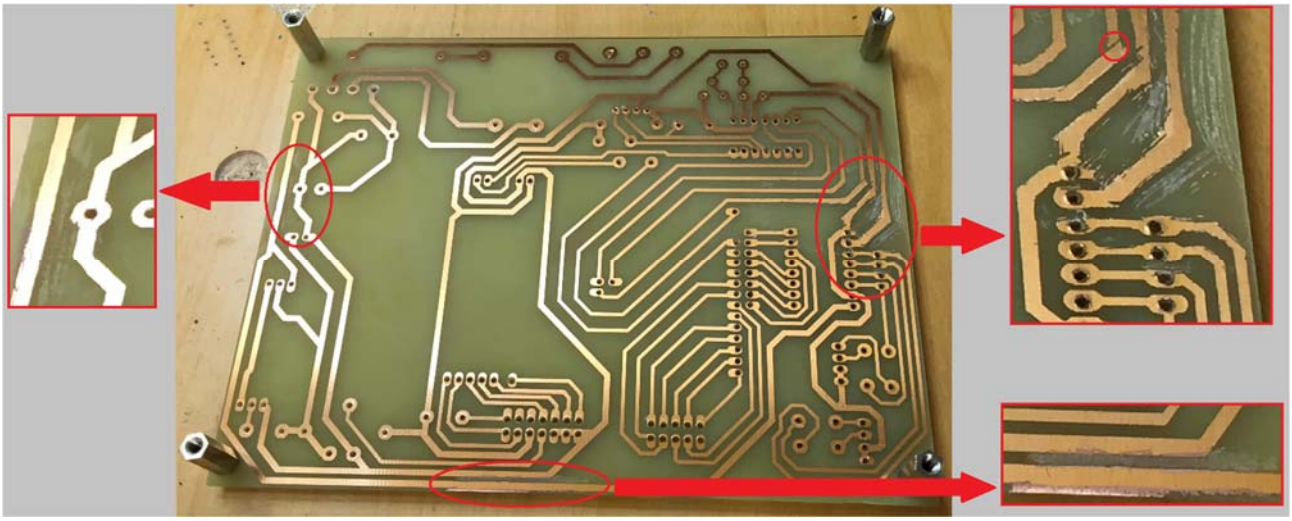
Una vez que se ha terminado con el proceso de insolación, se procede al revelado de la placa, Este proceso se parece bastante al proceso de revelado de los antiguos carretes fotográficos, con la excepción de que aunque es recomendable realizarlo con poca luz, no es necesario hacerlo solo con luz roja. Para realizar este proceso hay que diluir siete gramos de hidróxido de sodio, también conocido como sosa caustica, en un litro de agua. Una vez realizada la mezcla, se introduce la placa en la disolución y se mueve constantemente durante entre dos y tres minutos, hasta que se vean perfectamente definidas las pistas del circuito impreso. En la siguiente imagen puede verse como debe quedar la PCB tras el proceso de revelado.



PCB revelada.

El último proceso necesario para que la placa de circuito impreso sea funcional, hay que atacarla mediante un ácido, en el caso de este proyecto se ha utilizado cloruro férrico, que se vende en pequeñas bolas. Para poder atacar la placa, es necesario mezclar estas bolas con agua (preferiblemente caliente), y mezclarlo hasta que se disuelvan por completo. En este punto, hay que introducir la placa en la disolución y moverlo de forma constante hasta que desaparezca el cobre de las zonas en las que no hay pistas.

En la PCB que se ha fabricado, apareció el problema de que una pequeña zona de la placa no había recibido la insolación necesaria, por lo que a la hora de atacarla, en esta zona no desaparecía el cobre de entre las pistas, por lo que fue necesario rascar este cobre con una cuchilla para volver a atacar la placa posteriormente y obtener así el resultado funcional deseado, aunque estéticamente no se obtuvo un resultado perfecto.



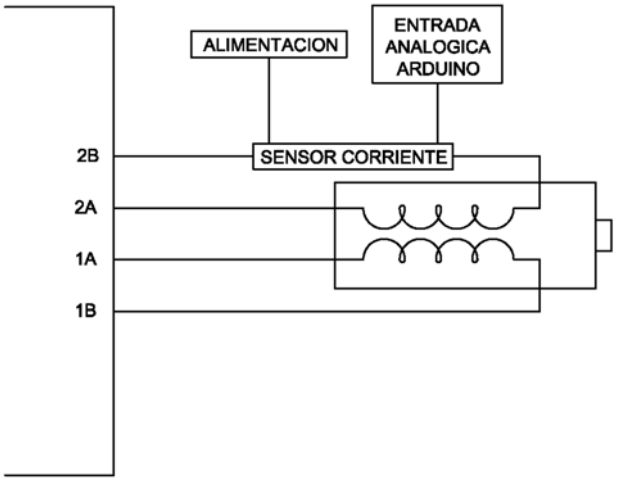
PCB terminada con detalle de las zonas problemáticas.

En la ampliación superior derecha se puede apreciar como hay una pista cortada, esto es consecuencia del rascado que hubo que hacer para eliminar el cobre sobrante durante en proceso de atacado. Este problema se ha resuelto soldando un hilo conductor que puentee el corte.

Por último, hay que taladrar los agujeros de inserción de los componentes, teniendo en cuenta que no todos los componentes tienen los pines del mismo tamaño, así en esta PCB se han realizado agujeros de 1 mm, de 1,2 mm y de 1,5 mm de diámetro.

9.8.- Sensor de corriente:

En un principio, se pensó en incluir en el proyecto un sensor de corriente para poder ver la forma de onda de la corriente por una de las bobinas del motor, esta representación gráfica se iba a visualizar mediante MatLab en un ordenador conectado al Arduino mediante puerto serie (USB). El sensor de corriente se intercalaría en uno de los cables que conectan el motor con el controlador del mismo, como puede verse en la siguiente imagen.



Conexión del sensor de corriente

Una vez probado el funcionamiento del sensor, se detecta que no se obtiene la forma de onda deseada, y una vez analizado el problema, se llega a la conclusión de que se debe a que las bobinas del motor presentan una fuerte reactancia inductiva, y por lo tanto se oponen a los cambios bruscos de corriente, dando así lugar una forma de onda que no aporta información relevante sobre el proceso de giro del motor.

Por este motivo, se ha decidido eliminar esta parte del proyecto. De todas formas, a continuación se indica el código necesario tanto de Arduino como de MatLab para leer el sensor de corriente y visualizar su lectura de forma gráfica.

Código Arduino:

```
const int sensor = A5;                  // Definición del pin A5 como sensor
int A =0;                               //Definición de la variable A (corriente)
                                        inicialmente con valor cero

void setup()
{
Serial.begin(9600);                     // Inicialización del puerto serie
}
void loop()
{
corriente = analogRead(sensor);         // Se almacena en la variable corriente la lectura
                                          del sensor
Serial.println(coriente);               // Enviar la variable corriente por el puerto serie
delay(100);                             // Espera 100 ms cada vez que envía un carácter
}
```

Código MatLab:

```
% GRAFICA EN TIEMPO REAL
% El puerto serie del arduino manda un valor cada 100 ms
% Matlab toma 50 lecturas (50*100=5000 ms, y tarda 5 segundos

% Dibujo de la gráfica con sus ejes

clc; clear all; close all;
title('FORMA DE ONDA DA LA CORRIENTE');
ylabel('Corriente (I)');
xlabel('tiempo (ms)');
grid on;
hold on;

% Inicialización de variables

A=0;
c=0;
z=zeros(1,1000);
t=zeros(1,1000);



% Bucle para poner el tiempo en tramos de 100 ms hasta 5000 ms

for i=1:50;
    c=c+100;
    t(i)=c;
end
% Configuración del puerto serie

delete(instrfind({'Port'},{'COM3'}));
x=serial('COM3');
x.BaudRate=9600;
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
fopen(x);

% Lectura del primer carácter que es alfafabético

y=fscanf(x,'%d')';

% Lee el reloj en el momento que empieza a leer el puerto

A=clock; A(6);
if A(6)~=0 disp('EMPIEZA');end

% Lee 50 veces el puerto enviando el arduino cada 100 ms

for i=1:50;
y=fscanf(x,'%d')';
z(i)=5*y/1023;
axis([0,5000,0,6])
plot(t(i:i),z(i:i),'r*');
end

% Lee el reloj en el momento que acaba de leer el puerto
```

```
B=clock; B(6);
if B(6)~=0 disp('FIN');end
disp(['Tiempo= ',num2str(B(6)-A(6)),' seg'])

% Cierra el puerto serie

fclose(x);
delete(x);
clear x;
```

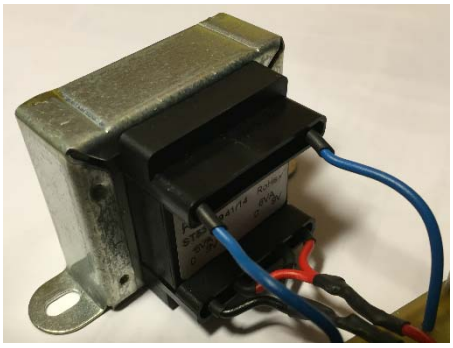
9.9.- Fuente de alimentación:

En un principio se colocó un transformador de 230/9V y 2,8VA, por lo que sin ningún elemento (teclado, LCD, motor o encoder) conectado al equipo, se obtenían aproximadamente 12V en la alimentación del motor, pero al ir conectando los distintos elementos del equipo, esta tensión caía hasta los 8,5V, lo que hacía que el motor no funcionase.

Este problema se debe a que la suma del consumo de los distintos elementos que componen el equipo supera la corriente que el transformador es capaz de proporcionar (300 mA), por lo que la tensión cae para mantener la corriente.

Se ha intentado usar un transformador de la misma potencia pero de más tensión de secundario para poder tener los 12V necesarios en la alimentación del motor, pero como al ser de la misma potencia y mayor tensión, la corriente es menor. El hecho de que la tensión sea la suficiente pero no así la corriente, hace que el motor no complete los pasos, girando de forma errática en ambas direcciones.

Para solucionar este problema, se ha sustituido el transformador anterior por otro de 230/9V y 12 VA (entre dos salidas), que es capaz de proporcionar hasta 1,3 amperios. La decisión de usar este transformador se ha tomado después de alimentar el equipo con una fuente de alimentación externa y medir el consumo del conjunto del sistema y obtener una medida de 700 mA. De esta forma se cubren sobradamente las necesidades de corriente del equipo.



Transformador 230/9V y 12VA

El transformador elegido es externo a la PCB, ya que no existen transformadores de PCB de 12VA del mismo tamaño que los de 2,8 VA, y por lo tanto el nuevo transformador no cabía en el hueco del antiguo. Como el nuevo tiene dos salidas de 6VA cada una, hay que conectarlas en paralelo para entre las dos sumen los 12VA necesarios.

A pesar de que el cálculo del condensador de filtrado variaría al cambiar el transformador, se ha comprobado de forma practica que la tensión de salida no varía. Por este motivo se ha decidido mantener el mismo condensador que se ha calculado anteriormente (4700 uF).

10.- Referencias:

10.1.- Bibliografía:

1. **Brian Evans**. Beginning Arduino Programming.
2. **Harold Timmis**. Practical Arduino Engineering.
3. **Maik Schmidt**. Arduino, a quick-start guide.
4. **Rafael Enríquez Herrador**. Guía de usuario de Arduino.
5. **Antonio Pardina Carrera**. Teoría sobre motores paso a paso.
6. **Vicente Fernández Escartín**. Apuntes de electrónica analógica.
7. **Departamento de ingeniería electrónica y comunicaciones – EUITIZ.** Apuntes de microprocesadores (Freescale 9S08).
8. **Francisco José Pérez Cebolla**. Apuntes de electrónica analógica.
9. **Manuel Torres y Miguel Angel Torres**. Diseño e ingeniería electrónica asistida con Protel DXP.

10.2.- Linkografía:

1. http://www.controleng.com/single-article/stepper-motion-evolution/a2bbc23f6619589d0d7834286273c9be.html. Evolución histórica de los motores paso a paso.
2. http://arduino.cc/es/Reference/LiquidCrystal. Manejo de la librería LyquidCristal.
3. http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus. Definición del bus SPI.
4. http://21stdigitalhome.blogspot.com.es/2013/02/arduino-due-hardware-spi.html. SPI para Arduino.
5. http://www.arduino.cc/en/Reference/SPI. Manejo y ejemplos de la librería SPI.
6. https://github.com/ivanseidel/DueTimer. Librería DueTimer con ejemplos de manejo.
7. http://geonobotwiki.free.fr/doku.php?id=robotics:electronics:shield_arduino_ls7366r. Manejo del integrado LS7366R con Arduino.
8. http://playground.arduino.cc/Main/KeypadTutorial. Manejo de la librería keypad.
9. https://www.pololu.com/product/1182. Descripción del controlador de motores paso a paso A4988.
10. http://es.rs-online.com/web/. Tienda de componentes electrónicos donde se han comprado gran parte de los componentes y se han conseguido muchos datasheets.
11. http://www.alldatasheet.es/. Página donde descargar datasheets.
12. https://www.pololu.com/product/1182. Página donde se ha comprado el controlador del motor paso a paso.

13. http://www.pjrc.com/teensy/td_libs_Encoder.html#polling. Funcionamiento y manejo de un encoder TTL.

14. http://www.crovisa.com/espanol/encap.htm. Página con características de transformadores para PCB.