UNIVERSIDAD POLITECNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería del Diseño

mim máster en ingeniería mecatrónica

Diseño e implementación de una máquina CNC con funcionalidad de taladradora

MEMORIA PRESENTADA POR:

Iván Piquer Mora

Máster Universitario en Ingeniería Mecatrónica

Universidad Politécnica de Valencia

DIRECTOR:

Juan José Serrano Martín

Julio 2016

ÍNDICE

1. Objeto

El objeto de este proyecto es la programación y puesta en marcha de una máquina CNC de tres ejes, dotada de la funcionalidad específica de taladrado.

A lo largo del mismo, se va a analizar cualquier aspecto importante y necesario para conseguir diseñar y controlar el sistema. Se tendrá en cuenta la elección de los materiales, centrando especial atención a los componentes y elementos correspondientes a la parte electrónica.

Con esto, se va a pretender llevar a cabo un sistema que en su conjunto, cumpla con los objetivos a la perfección y con buena precisión y resolución, tratando a su vez de que no suponga un coste elevado.

Por tanto, se analizarán en detalle los objetivos que debería poder cumplir la máquina una vez en funcionamiento.

Finalmente, se llevará a cabo un análisis completo de los elementos a utilizar, así como el coste desglosado y total del sistema.

2. Justificación

2.1. Académica

Este trabajo se lleva a cabo con el fin de completar el máster universitario y adquirir el título de “Máster Universitario en Ingeniería Mecatrónica”.

2.2. Técnico-económica

Los elementos mecánicos y electrónicos nombrados y utilizados para la realización del sistema, así como sus respectivos precios han sido tomados de catálogos de fabricantes, por lo que son reales.
Por tanto, el importe total que supondría la fabricación de la máquina en cuestión es el que se ha incluido en este trabajo, pudiéndose dar variaciones, aunque limitadas.

3. Introducción

Desde sus inicios, se ha tendido a llevar a cabo los trabajos de manera manual, ya fuera la realización de figuras, agujeros, pulidos, y la totalidad de actividades o procesos industriales, tanto de fabricación como de mecanizado.

Con el tiempo, ante la necesidad de implementar variedad de mecanizados diferentes, y de una elevada complejidad, se ha ido desarrollando la automatización de los procesos, suponiendo a su vez un incremento de la velocidad y por tanto cantidad de producción, un menor número de empleados, lo que reducía los costes salariales, y además, garantizaba una producción más regular y precisa, mediante lo cual se posibilita la realización de diferentes diseños. Además, conlleva la eliminación de ciertos peligros durante procesos que anteriormente debían ser procesados por personas.

Algunas de estos sistemas de automatización se han llevado a cabo en base al control numérico, lo que además en la actualidad está siendo muy utilizado, ya que supone un avance en todos los aspectos, dando precisión, regularidad y rapidez a todo tipo de procesos.

El CNC (Control Numérico por Computador), es un sistema que permite controlar la posición de la herramienta, así como su velocidad y aceleración, empleando una serie de órdenes o comandos estandarizados.

Además, hoy en día es un tipo de control muy conocido, ya que además de máquinas industriales, se están fabricando máquinas CNC de bajo coste que, llevando a cabo labores que no precisan de mucha potencia, la finalizan de manera correcta, y debido a su reducido precio, están muy extendidas.

Ahora bien, para llevar a cabo el uso de una máquina de este tipo, y sobre todo para poder diseñarla y realizar su puesta en marcha, conviene conocer más a fondo todos los aspectos relacionados con la misma.

3.1. Historia y evolución

El Control Numérico de Máquinas fue desarrollado con el fin de llevar a cabo diseños concretos pero cada vez más complejos que iban surgiendo en materia de fabricación y mecanizado. Esto es debido a que, dada la dificultad que iban suponiendo dichos diseños, se hacía más y más difícil realizarlo de manera manual con máquinas convencionales, y más aún crear modelos idénticos.

Pero hasta la actualidad, donde ya se controlan sin problemas diferentes sistemas por control numérico, se ha ido evolucionando lentamente desde la utilización única de máquinas convencionales. Algunas de las ideas, creaciones y mejoras que se han dado hasta ahora son:

- En 1942, Bendix Corporation lleva a cabo el cálculo de los puntos de la trayectoria para mecanizar una leva en 3D.
- En 1947, John Parsons desarrolla el sistema DIGITON para fabricar hélices de helicóptero, que consta de un mando automático.
- En 1953, tras desarrollar una fresadora de 3 ejes, se utiliza por primera vez el término CNC.
- En 1956, la U.S.A.F. pide 170 máquinas de Control Numérico.
- En 1960, en el M.I.T., se llevaron a cabo demostraciones de Control Adaptable, que supone una evolución del Control Numérico que permite además la autorregulación de las condiciones de trabajo de la máquina.
- En 1968, se producen los primeros ensayos de Control Numérico Directo (DNC).

3.2. Tipos de máquinas que emplean CNC

Mediante el sistema de control numérico, es posible llevar a cabo máquinas para multitud de aplicaciones que son muy utilizados en la industria actualmente. Algunas de las aplicaciones que se llevan a cabo son:

- Fresadora.
- Torno.
- Taladradora.
- Dobladora.
- Punzonadora.
- Impresora 3D.
- Cortadora por láser.

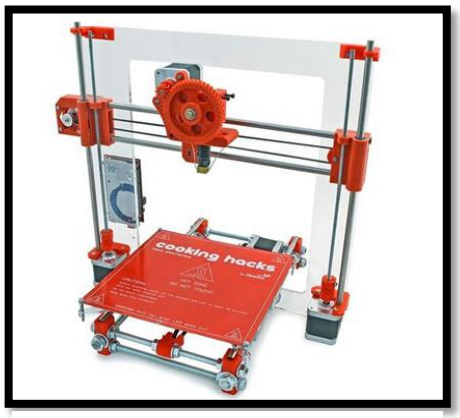


Ilustración 1: Impresora 3D, www.tecnologyc.com

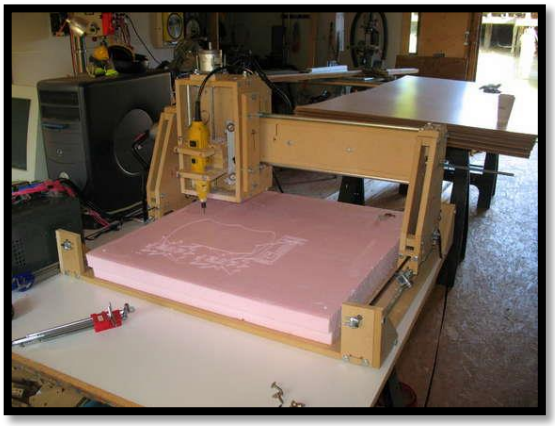


Ilustración 2: Máquina CNC casera, www.arquitecturaenred03.com

Además de las aplicaciones para las que ya se emplea este sistema de control, puede ir ampliándose su utilización en algunas otras, siempre y cuando vayan a facilitar la tarea de alguna manera, ya sea por mejorar el rendimiento y la calidad o bien por evitar riesgos de seguridad de operarios.

3.3. Ventajas

La utilización de sistemas CNC, tiene una gran cantidad de ventajas o mejoras con respecto al uso de otro tipo de sistemas o máquinas.

Algunas de las ventajas que aporta su uso son:

- Aumento en la exactitud y repetibilidad, lo que conlleva a un incremento en la calidad final de los productos fabricados.
- Debido al movimiento y posicionamiento automáticos, permite dotar a los productos de una perfecta uniformidad.
- Un único operario es capaz de utilizar varias máquinas de este tipo de manera simultánea.
- Mayor flexibilidad, ya que permite la realización de geometrías y procesados manualmente inviables, y mantener guardado el programa empleado para utilizarlo posteriormente en caso de necesario.
- Mayor velocidad de producción.
- Incremento de la seguridad por realización de tareas peligrosas por medio de máquinas.
- Posibilidad de llevar a cabo simulaciones previas del proceso a realizar, de modo que es posible evitar errores e incluso visualizar el aspecto del resultado final.

3.4. Inconvenientes

Del mismo modo, este tipo de sistemas, cuenta con ciertas desventajas, aunque más reducidas que las ventajas. Algunas de ellas son:

- Coste elevado tanto de materiales como de mantenimiento.
- Precisa de una programación eficiente, así como de una puesta a punto correcta.
- Se precisa de gran cantidad de trabajo para llevar a cabo la amortización.

3.5. Comparativa con máquinas convencionales

De las máquinas convencionales, es decir, aquellas que precisan de un operario que ejecute tanto las órdenes como cada uno de los movimientos y procesos de la máquina, se ha evolucionado a máquinas que llevan a cabo su posicionamiento y control en función del trabajo y parámetros que se les configure, donde se incluyen las máquinas tipo CNC.

Ahora bien, las diferencias más importantes existentes entre las máquinas convencionales y las actuales son:

- Un operario solo es capaz de manejar una única máquina convencional, mientras que puede controlar varias máquinas CNC.
- En las máquinas convencionales se precisa de una mayor preparación y conocimientos, mientras que en las máquinas CNC, dada su autonomía, precisa de una menor preparación.
- Las máquinas convencionales, requieren de un elevado tiempo de fabricación, mientras que las máquinas CNC permiten una producción más elevada.
- Dado que en las máquinas convencionales el operario lleva a cabo cada movimiento y proceso, es más probable realizar errores en algún momento, mientras que en las CNC, se parametriza inicialmente, incluso con posibilidad de simulación previa, de modo que automáticamente lleva a cabo los mismos procesos, lo que reduce los posibles errores humanos.

3.6. Implementación de máquinas CNC

3.6.1. Elementos necesarios

Para llevar a cabo una máquina CNC, se precisa de una serie de elementos básicos, los de mayor importancia son los que se detallan a continuación.

3.6.1.1. Motores paso a paso

Los motores paso a paso son un tipo de motor eléctrico de corriente continua, sin escobillas. La característica principal y que los diferencia de otros motores es que es posible llevar a cabo movimientos precisos, ya que no giran libremente como ocurre con el resto de motores de continua, sino que realizan giros discretos, mediante pequeños pasos que pueden ser configurados y por tanto controlados. Es por ello que, además, permiten controlar la velocidad de giro del mismo, variando el tiempo transcurrido entre pasos.

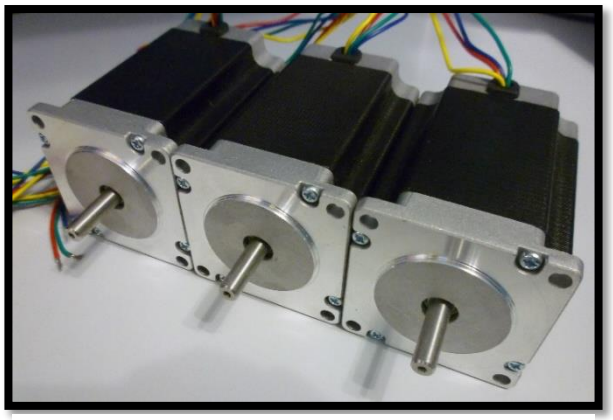


Ilustración 3: Ejemplo motores paso a paso, www.ahipcnc.com

3.6.1.1.1. Parámetros característicos

- **Par de mantenimiento (holding torque):** Par resistente ejercido por el motor detenido en una posición estable y con alimentación. Pueden darse variaciones dependiendo de la posición del eje del motor respecto a las bobinas.

-Par de retención (detent torque): Par máximo que ofrece el motor cuando no se encuentra alimentado.

-**Par pull-out:** Relación del par que es capaz de entregar el motor a máxima velocidad sin pérdida de pasos.

-**Par pull-in:** Relación del par que es capaz de entregar el motor con la velocidad sin pérdida de pasos durante el arranque y la parada.

-**Ángulo de paso (step angle):** Ángulo de giro que se produce en el eje del motor al cambiar de una posición estable a la siguiente.

-**Número de pasos por vuelta:** Cantidad de pasos necesarios para efectuar una vuelta completa del eje del motor.
Siendo:

$$\boldsymbol{Núm.pasos} = \frac{\mathbf{360}}{\boldsymbol{Ángulo_{paso}}}$$

-**Frecuencia de paso máximo (máximum pull-in/pull-out):**
Número máximo de pasos por segundo que puede realizar el motor en un funcionamiento correcto del mismo.

-**Precisión de paso (step accuracy):** Error de la posición actual del rotor con respecto a la posición teórica en la cual debería encontrarse, estando el motor trabajando sin carga o con ella constante. Se trata de un valor constante, y que no se incrementa con la cantidad de pasos que se realicen.

3.6.1.1.2. Tipos

Dentro de los motores paso a paso, se pueden distinguir varios tipos en función de su estructura interna:

- **Imanes permanentes:**

El estator se compone de unos núcleos sobre los cuales se encuentran arrolladas bobinas. Sus extremos constituyen los polos del estator.
El rotor está formado por polos magnéticos S-N.

- **Reluctancia variable:**

El estator está formado por entre 3 y 5 bobinas arrolladas a unos núcleos formando sus extremos los polos del estator, y generando un campo magnético.
El rotor está constituido por hierro dulce laminado que contiene varios dientes.

- **Híbridos:**

Unos núcleos, normalmente dos que cuentan con bobinas arrolladas. Sus extremos forman los polos del estator, pudiendo contar con más de 4.
El rotor se constituye por dos ruedas dentadas de hierro dulce, y están separadas por un imán. De este modo, una rueda tiene polarización N y la otra S.

Además, dentro de los motores paso a paso del tipo híbridos y de imanes permanentes, se diferencian según la formación del estator en:

- **Unipolares:**

Se componen por 2 semidevanados y cuentan con 1 corriente por devanado.

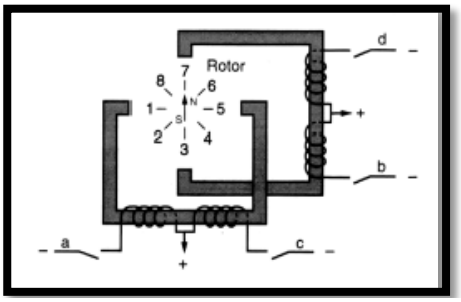


Ilustración 4: Esquema interno motor paso a paso unipolar, www.nmbtc.com

- **Bipolares:**

Se componen por 1 devanado y cuentan con 2 corrientes por devanado.

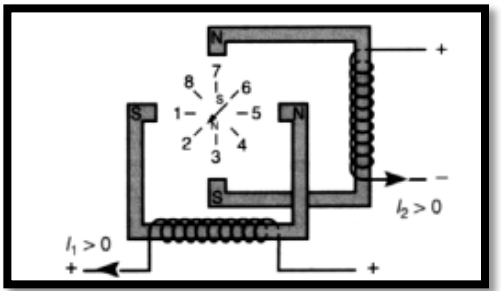


Ilustración 5: Esquema interno motor paso a paso bipolar, www.nmbtc.com

3.6.1.1.3. Motor PaP híbrido bipolar

El funcionamiento de este tipo de motores es muy similar al de imanes permanentes. La diferencia se encuentra en que la cantidad de polos puede variarse de manera sencilla modificando el número de dientes de las ruedas. Este factor hace que sea más fácil llevar a cabo su construcción.

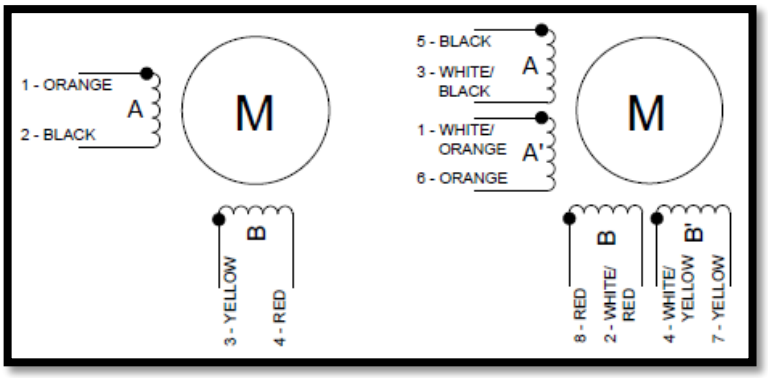


Ilustración 6: Esquema motor paso a paso híbrido bipolar, www.diymakers.es

Cuentan con una alta resolución, además de un elevado par motor al igual que frecuencia de trabajo. En cuanto al movimiento de los mismos, el sentido de giro depende tanto del sentido de las corrientes por las bobinas como de la secuencia de alimentación de las mismas.

3.6.1.2. Drivers

Se trata de dispositivos electrónicos diseñados para controlar motores paso a paso unipolares y bipolares, y que suponen la etapa de potencia del sistema.

Se reciben señales generadas desde el microcontrolador, que se corresponden con el tipo de control (full step, half step, microstep), señal de avance, sentido de giro y permiso, todo ello a la tensión y corriente de salida del propio microcontrolador, siendo éste el que empleando la alimentación de una fuente externa, y con ayuda de puentes H, comanda al motor a la tensión y corriente nominales del motor, y según las configuraciones y órdenes del microcontrolador.

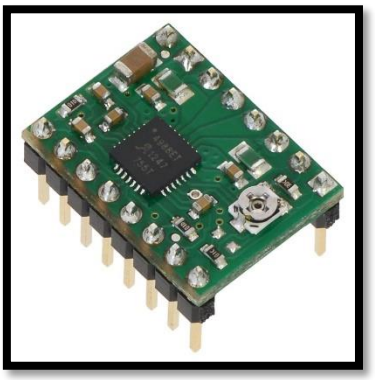


Ilustración 8: Driver A4988, www. imprimalia3D.com

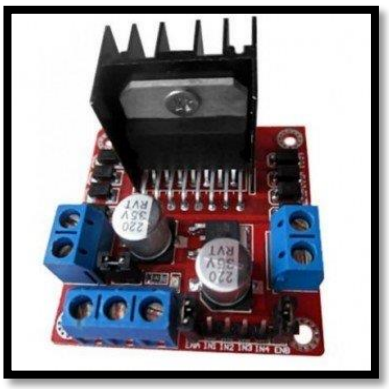


Ilustración 7: Controlador L298N, www.interarduino.com

3.6.1.3. PLC o microcontrolador

Se trata de un circuito integrado programable, que tras introducirle un programa previamente diseñado y verificado, es capaz de ejecutar las órdenes programadas e introducidas en su memoria.
Además, los microcontroladores generalmente cuentan con una serie de bloques funcionales que son:

- Memoria.
- Periféricos.

Además cuentan con diferentes posibilidades de comunicación como SPI, I2C, Ethernet, Puerto serie… y todo ello empleando un consumo reducido y son capaces de llevar a cabo una gran cantidad de tareas de una manera rápida y eficaz.

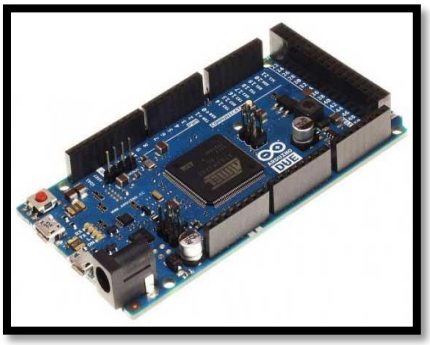


Ilustración 10: Arduino Due, www.bricogeek.com

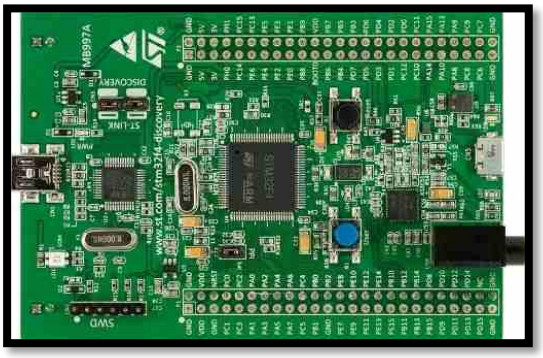


Ilustración 9: ARM Cortex, www.todopic.com

3.6.1.4. Finales de carrera

Se trata de elementos que abren y cierran sus contactos cuando algo entra en contacto con la parte funcional de los mismos. Además se pueden encontrar de diferentes tipos, como pueden ser los ópticos, en los cuales no se necesita un contacto físico para que se accionen, o los finales de carrera mecánicos, que precisan de un contacto directo para que entre en funcionamiento.

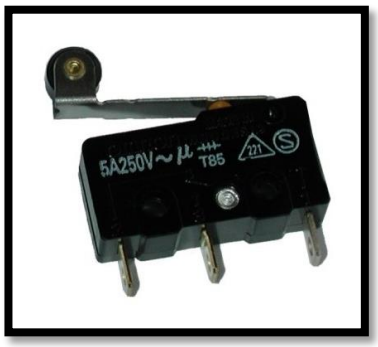


Ilustración 11: Final de carrera mecánico, www.todopic.com

Ilustración 12: Final de carrera óptico, www.electronilab.com

3.6.2.Movimiento y control de motores

El movimiento de los motores paso a paso se lleva a cabo mediante variaciones de posiciones estables y discretas. El cambio de una posición estable a otra se denomina paso, y se consigue mediante la excitación de las bobinas del estator en una secuencia determinada.
En cuanto la excitación de las bobinas, se puede realizar siguiendo tres patrones:

- **Full step:** Se trata de un modo de excitación mediante el cual se hace circular intensidad por al menos un devanado con lo que se desplaza el rotor al paso siguiente.

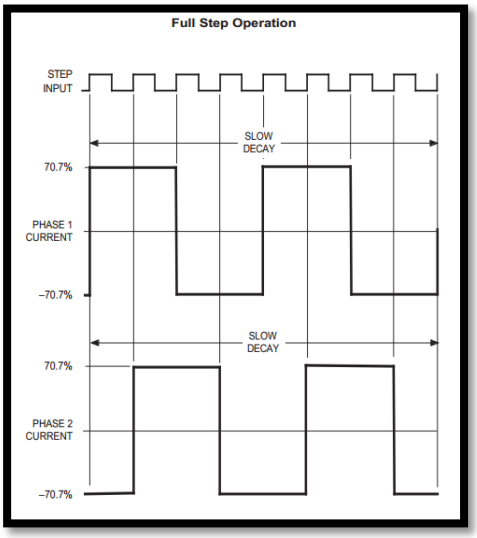


Ilustración 13: Control Full – Step, www.poscope.com

- **Half step:** Se trata de un modo mediante el cual se alcanzan posiciones estables intermedios entre pasos, lo que implica la posibilidad de disponer del doble de pasos por revolución.

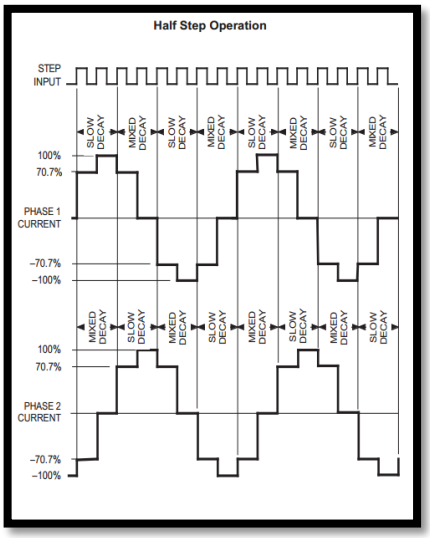


Ilustración 14: Control Half – Step, www.poscope.com

Pero al emplear los modos de excitación anteriormente nombrados, se cuenta con una baja resolución, así como con transiciones bruscas que disminuyen el rendimiento de trabajo del motor.

- **Microstepping:** Se trata de una alternativa que se basa en la alimentación de las bobinas con una corriente con forma de onda senoidal mediante la aplicación de una tensión PWM.
Mediante el empleo de este método de excitación de las bobinas de los motores, se obtiene una mayor cantidad de pasos por revolución, dotando al sistema de una mayor resolución, disponiendo a su vez, de movimientos con mayor suavidad.

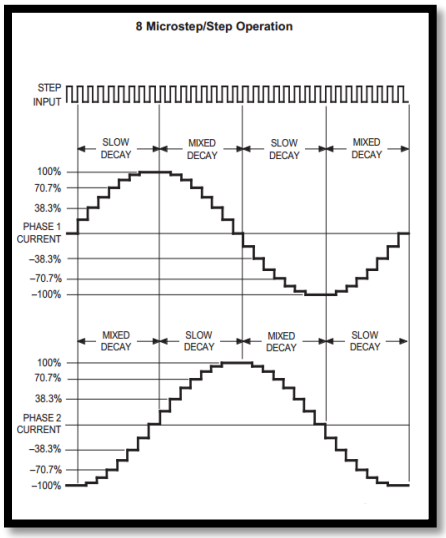


Ilustración 15: Control Microstepping, www.poscope.com

Por tanto, para el control de los motores, se debe excitar las bobinas del estator siguiendo la secuencia correspondiente en función del tipo de control deseado, pudiendo controlar, conociendo el número de pasos realizados, la posición del eje del motor con respecto a un punto de referencia, así como la velocidad del motor en todo momento.

3.6.3. Comandos Gcode

G-code constituye un tipo de lenguaje de programación muy utilizado en el control numérico (CNC). Se trata de comandos o instrucciones que se envían a la máquina, indicando posición de destino, velocidad, así como tipo de movimiento, la cual las comprende y las lleva a cabo.

Comando	Ejemplo	Descripción	TX	RX
G0	G0 X10	Movimiento lineal Rápido	Si	Si
G1,G01	G1 X10 Y15 Z0 [F100]	Movimiento lineal Controlado (Avance: 100)	Si	Si
G2,G02	G02 X60 Y30 I30 J-10 F02	Movimiento curvo (sentido horario) Controlado	Si	Si
G3,G03	G03 X60 Y30 I10 J20	Movimiento curvo (antihorario) Controlado	Si	Si
G4,G04	G4 P200	Pausa con retardo (Retardo: 200ms)	Si	Si
G20	G20	Definir Unidades en Pulgadas	Si	Si
G21	G21	Definir Unidades en milímetros	Si	Si
G28	G28	Ir a Origen	Si	Si
G30	G30 X10 Y20 Z30	Ir a Origen a través de un punto	Si	Si
G90	G90	Definir Coordenadas absolutas	Si	Si
G91	G91	Definir Coordenadas relativas	Si	Si
G92	G92	Definir punto actual como origen	Si	Si
M0	M0	Paro (Pausa programada)	Si	No
M3,M03	M3	Marcha del cabezal	Si	Si
M5,M05	M5	Paro del cabezal	Si	Si

Ilustración 16: Tabla de comandos G-code, www.txapuzas.com

3.6.4. Transmisiones

Para llevar a cabo el movimiento de elementos en la máquina CNC empleando los motores, se precisa de una serie de elementos que posibiliten el enlace del motor a los elementos e incluso para modificar el plano del movimiento. Este tipo de elementos se conocen como elementos de transmisión, siendo los más conocidos y utilizados los siguientes:

- **Correas dentadas**, las cuales uniendo dos o más ruedas dentadas, transmiten el movimiento del eje del motor a elementos haciendo que se muevan perpendicular al motor, pudiendo llevarse a cabo reducciones para así aumentar el par del motor en los movimientos.

Ilustración 17: Transmisión por correa, www.dreamstime.com

- **Poleas:** Se trata de un mecanismo mediante el cual se transmite un esfuerzo, facilitando además el movimiento.

Ilustración 18: Transmisión por polea, www.3despana.com

- **Engranajes:** Se trata de un mecanismo para la transmisión de potencia, muy similar al de las correas, pero por medio de dos ruedas dentadas, denominadas corona y piñón, a la mayor y a la menor, respectivamente. Además, el empleo de este sistema asegura que no va a perderse parte del movimiento, a diferencia que con las correas que puede darse el caso de que se produzca deslizamiento.

Ilustración 19: Transmisión por engranajes, www.energia9.es

3.7. Objetivos

A lo largo de este trabajo se procederá a realizar el análisis, diseño y programación de una máquina CNC con funcionalidad de taladradora, siendo los objetivos principales los siguientes:

- **Elección de elementos electrónicos necesarios.**
  - Microcontrolador.
  - Motores.
  - Finales de carrera.
  - Fuente de alimentación.
  - Cableado.

- **Programación del microcontrolador.**
- **Puesta en marcha y verificación del funcionamiento.**

En el caso del presente proyecto, se realizará la puesta en marcha de una máquina CNC con la funcionalidad de taladradora, la cual tras recibir cada línea completa, procederá a llevar a cabo la orden o el ciclo de taladrado especificado, y en el número de veces ordenado.

Tras alcanzar el final del último agujero, comenzará a subir hasta que alcance la altura del eje Z especificada, avisará de que ha finalizado la orden y procederá a analizar la siguiente orden en cuanto la reciba.

4. Materiales seleccionados

4.1. Sistema mecánico

- **Estructura**

  La estructura principal y de sujeción de la máquina CNC se encuentra compuesta por barras metálicas, de acero inoxidable, de forma rectangular, y que ubicadas en las zonas exteriores proporcionan un buen soporte y robustez.

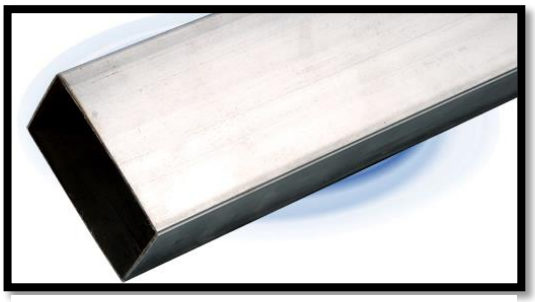


Ilustración 20: Barra de acero inoxidable, www.sunnysteel.com

Además, las estructuras llevan los orificios necesarios donde van sujetas las barras y sistemas de guiado del sistema.

- **Varilla lisa calibrada**

  Para llevar a cabo el desplazamiento de las distintas partes, se dispone de varillas lisas calibradas, de acero inoxidable, de un diámetro de 22 mm.

Ilustración 21: Varilla lisa calibrada de acero inoxidable, www.diymania.es

Para una mayor precisión, y evitar vibraciones, flexiones y deformaciones, el sistema cuenta con dos varillas por eje de movimiento, cuyos extremos van introducidos de manera precisa en los orificios de las barras de acero inoxidable correspondientes a la estructura principal.

- **Rodamiento lineal**

Para permitir el desplazamiento de las partes móviles se han empleado rodamientos lineales, que correctamente engrasados y evitando la suciedad, limitaran el rozamiento durante los movimientos.

Ilustración 22: Rodamiento lineal

Dado que las varillas sobre las cuales deslizarán los rodamientos tienen un diámetro de 22 mm, los rodamientos dispondrán del mismo diámetro interno para evitar holguras.

- **Husillo de bolas**

Como sistema de transmisión del movimiento rotatorio del motor a desplazamiento lineal del sistema, en el eje Z (bajar/subir herramienta) se emplea un husillo de bolas.

Ilustración 23: Husillo de bolas, www.banggood.com

Los husillos empleados se componen de una varilla roscada que cuenta con un diámetro de 16 mm y 22 mm.

- **Correa dentada**

Para la transmisión del movimiento en los ejes X e Y se emplea correa de transmisión dentada, como la que se puede ver en la siguiente imagen.

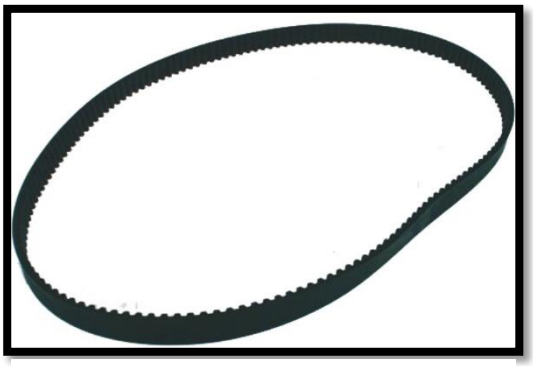


Ilustración 24: Correa dentada, www.repuestosautos.el

- **Polea dentada**

Para poder completar la transmisión por correas, dicha correa va enlazada con el eje del motor mediante una polea dentada, así como por el otro extremo, cerrando así el sistema de movimiento.

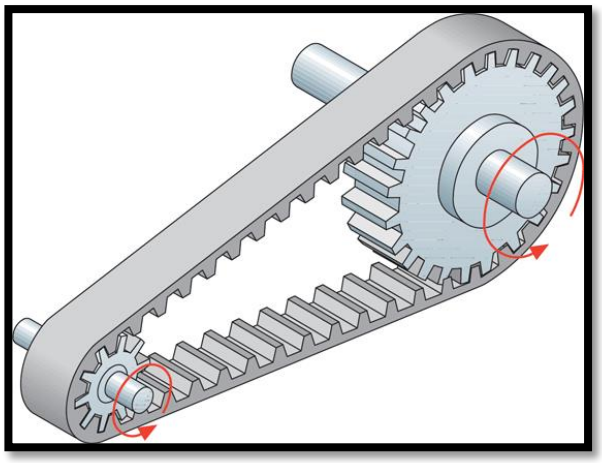


Ilustración 25: Polea dentada, www.skat.ihmc.us

Como se puede observar en la imagen anterior, la polea del eje del motor es de diámetro menor que el de la polea de salida, lo que dota al movimiento de un par más elevado.

- **Elementos de sujeción**

  Por último, todos los elementos principales y necesarios anteriormente nombrados, así como los elementos electrónicos (motores, drivers, finales de carrera…) deben estar sujetos correctamente, para lo cual se ha empleado tornillos de acero inoxidable de diferentes métricas según la finalidad y el esfuerzo precisado, quedando todo el sistema perfectamente anclado.

Ilustración 26: Tornillos, www.fresno.pntic.mec.es

4.2. Parte electrónica

- **Fuente de alimentación:**

Para la alimentación del sistema se precisa de una fuente de alimentación. Para la selección de la misma, en primer lugar se necesita conocer cuál es la potencia máxima del conjunto de elementos que componen el sistema, y que puedan funcionar en un mismo instante de tiempo durante su utilización.

Para llevar a cabo las pruebas del correcto funcionamiento del software implementado, se emplea una fuente de alimentación con 4 salidas de 16 V cada una de ellas.

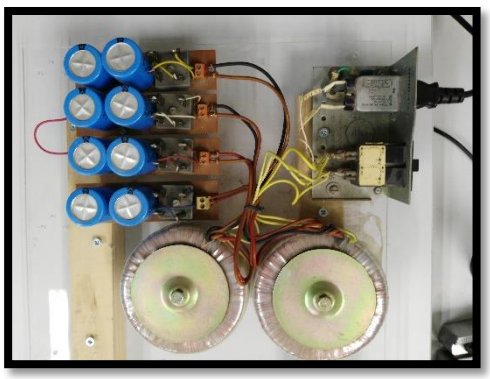


Ilustración 27: Fuente de alimentación

Se trata de una fuente de alimentación con la que se pueden realizar pruebas de manera sencilla y efectiva.

Es posible que al llevar a cabo el sistema completo, se precise de una reducción de tamaño y peso, así como de una fuente que proporcione una corriente mayor. En este caso se procede, una vez conocidas las potencias de los elementos del sistema, a la utilización de una fuente de alimentación que se ajuste a las necesidades del sistema.

Para este caso, la fuente de alimentación debe ser de 45 V de tensión, y capaz de suministrar una corriente de 20 A, de modo que el sistema pueda funcionar sin limitaciones.

- **Motores:**

Dado que se trata de un sistema dotado de movimiento, se emplean motores. Pero para ello, es necesario determinar el tipo de motor que se pretende utilizar, teniendo en cuenta las características y el precio, así como las limitaciones que supone el uso de cada tipo de motor.

En este caso, se ha decidido emplear motores paso a paso. Las razones de uso de este tipo de motores es debido a que en principio no se precisa del uso de encoders para determinar la posición, y es posible trabajar por tanto en bucle abierto y conocer a su vez la posición y velocidad del mismo de una manera sencilla.

Además, este tipo de motores, gracias a los cuales se puede conseguir una precisión más que suficiente en gran parte de aplicaciones, no tienen un coste elevado, lo que supone también un factor importante para su elección.

Por último, y considerando la información vista en internet, este tipo de motores es el más utilizado para este tipo de aplicaciones, estando muy arraigado, especialmente en aplicaciones en las que se pretenda incrementar lo menos posible el precio.

Los motores empleados son paso a paso bipolares, NEMA 23, más concretamente los ‘MAE HY200-2232-190C8’.

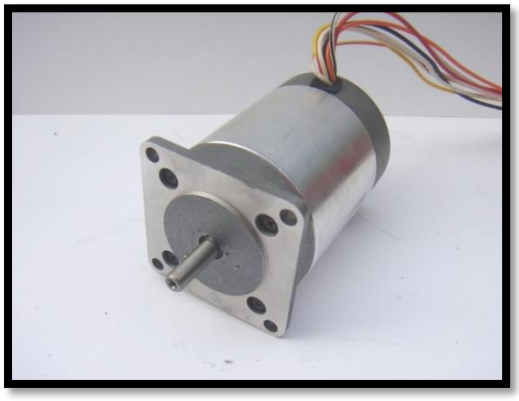


Ilustración 28: Motor paso a paso MAE HY200

Cuyas características más importantes son:

- Tensión de alimentación → 48 V
- Corriente por fase → 2.7 A
- Resistencia de fase → 1.8 Ω
- Inductancia de fase → 3.3 mH
- Par de mantenimiento → 113 N·cm
- Par sin alimentación → 8.5 N·cm
- Inercia del rotor → 200 g·cm²
- Ángulo de paso → 1.8 º
- Precisión ángulo de paso → 5 %

Estos motores moverán cada uno de los ejes de manera independiente (X, Y, Z), empleando un motor por eje, así como uno más para el extrusor, en caso de que la máquina se vaya a emplear como impresora 3D.

- **Drivers para los motores:**

Dado que el control de los motores se va a llevar a cabo mediante un microcontrolador cuya salida alcanza un valor de 5 V aproximadamente y una corriente muy reducida, la tensión de salida del mismo no es suficiente para el funcionamiento de los motores, y la corriente que puede aportar tampoco lo es.

Además, se precisa de la utilización de drivers, para así controlar los motores desde el microcontrolador, pero permitiendo que reciban la alimentación desde una fuente externa, con valores de tensión y corriente suficientes para poder hacer funcionar el motor o motores de manera correcta, incluso simultáneamente.

Así pues, cada motor dispondrá de su driver, que es el 'NUCLEO IHM01A1'. El motivo de la elección del driver es debido a que están especialmente diseñados para su conexionado con el microcontrolador que se va a emplear, lo que evita tener que llevar a cabo tanto el diseño de circuitos, como la posterior fabricación de las pcb con dichos circuitos impresos, que lleven a cabo la unión entre el micro y los drivers.

Ilustración 29: Driver NUCLEO IHM01A1

Las especificaciones técnicas de interés de este tipo de driver son:

- Rango de tensión → 8 – 45 V
- Corriente de fase máxima → 3 A r.m.s.
- Resolución máxima → 1/16 micropasos
- Indicación de alimentación y de fallo
- Control de corriente
- Protección etapa de potencia

Empleando un solo microcontrolador, se pueden utilizar 3 drivers, conectados uno encima del otro, y así controlar sin problemas tres motores paso a paso, lo que permite el control de los 3 ejes. En caso de la inclusión o utilización de otro motor más, como por ejemplo para un extrusor en una impresora 3D, se precisaría de controlarlo mediante un segundo microcontrolador, o bien utilizando otros tipo de drivers que permitieran el uso de más de tres.

Al margen de esta posible necesidad futura, dado que actualmente esta máquina va a ser empleado como método de aprendizaje, en el control de 3 ejes radica la parte importante a tratar.

• **Microcontrolador:**

Como elemento principal y de mayor importancia en el sistema, se encuentra el microcontrolador. Éste es el cual se debe programar y que llevará a cabo tanto la lectura de sensores como la actuación sobre los motores y sobre cualquier otro actuador del cual se dote al sistema.

El microcontrolador utilizado se corresponde con un ARM Cortex, más concretamente se trata del 'NUCLEO-L053R8' de ST Microelectronics.

Ilustración 31: Microcontrolador STM32L053R8, www.st.com

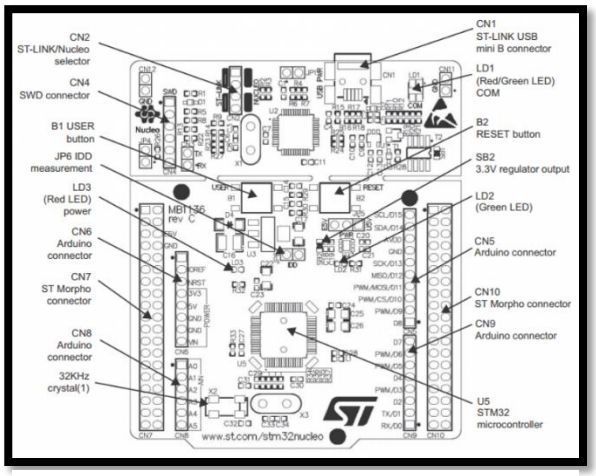


Ilustración 30: Partes microcontrolador STM32L053R8, www. st.com

Cuyas características y especificaciones más importantes son:

- Dos posibles módulos de extensión:
  - Arduino Uno R3.
  - STM Morpho.
- Alimentación:
  - USB → 3.3 V
  - VBUS → 5 V
  - Fuente externa → 7 – 12 V
- Núcleo:
  - ARM 32-bit Cortex-M0
  - Frecuencia de 75 MHz
  - 125 DMIPS
- Memoria:
  - 512 Kbytes de memoria Flash
  - 128 Kbytes de SRAM

- **Final de carrera:**

Como método de delimitación en el movimiento de los ejes, se emplean finales de carrera, para así evitar colisiones o problemas en caso de pretender avanzar en un eje más que el límite de su recorrido.

Ilustración 32: Final de carrera mecánico

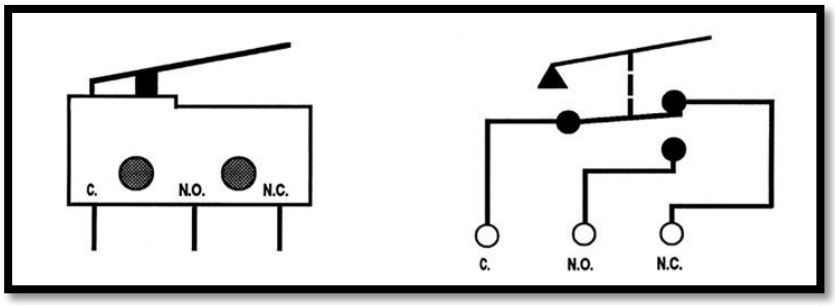


Ilustración 33: Esquema interno final de carrera mecánico, www.pourla.com

Además, los finales de carrera también supondrán el punto de partida del sistema, es decir, constituirán el punto (0, 0, 0) del sistema de coordenadas (Homing), a partir del cual, cada paso que se mueva el motor en cada eje, tendrá que quedar reflejado, ya que gracias a su lectura se conocerá la posición actual del motor, siempre respecto al punto (0, 0, 0) del sistema de coordenadas.

Por otra parte, y dado que este tipo de componentes produce imprecisiones o incluso fallos en determinadas ocasiones durante la transición de un estado a otro (accionado o no accionado), se ha añadido un circuito adicional. Se trata de un circuito anti rebote, el cual mantiene un nivel lógico estable en todo momento, evitando así las posibles oscilaciones que pudieran producirse.

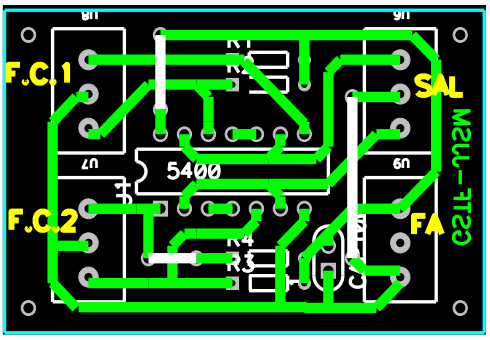


Ilustración 34: Diseño circuito anti rebote

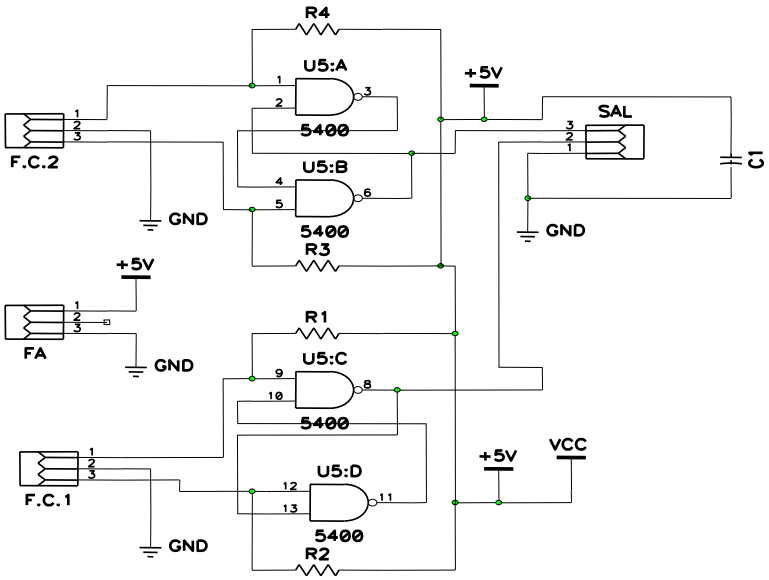


Ilustración 35: Conexionado circuito anti rebote

4.3. Software

Para la programación del microcontrolador, se precisa de un software de programación específico o compatible con el microcontrolador.

En este caso, se emplea el "KEIL µVision", por diversas razones. Como principal razón, se trata de un software libre, aunque con ciertas limitaciones en cuanto al tamaño del programa, pero sin que llegue a afectar para lo que se va a emplear.

Ilustración 36: Imagen etiqueta del software de programación "Keil µVision 5"

Además, se trata de un software que he utilizado para programar en alguna asignatura a lo largo de los estudios, y más concretamente, en la de "Sistemas Embebidos" en el "Máster en Ingeniería Mecatrónica".

Por último, se sabe que se trata de un software con bastantes posibilidades, lo que puede suponer una ventaja para su uso.

Por estas razones, este software supone la elección realizada para este trabajo.

5. Desarrollo y construcción mecánica

La parte mecánica correspondiente a la máquina CNC es una parte de igual importancia que la parte electrónica. Si la parte mecánica no está diseñada o construida de manera correcta, ya sea por un fallo en el diseño, por empleo de materiales que no son idóneos para dicha aplicación, o por no estar correctamente fijados, puede ocurrir que el sistema no se comporte como se esperaba, teniendo una menor estabilidad, así como produciéndose pérdida de precisión.

Para evitar problemas, en los que se incluyen los anteriormente nombrados, las piezas utilizadas son metálicas, fijadas entre sí por medio de tornillos, y las correas empleadas deberán estar bien tensas, ya que de esta manera se asegura un par mayor sin posibilidad de errores en el número de pasos debidos a fallos de construcción.

Al margen de ciertas pautas importantes a la hora de construir el sistema mecánico, algunas de las cuales se han nombrado, y de los materiales empleados, los cuales se han detallado en el apartado de materiales de este trabajo, no se va a entrar en más detalle. Esto es debido a que ya se dispone de la parte mecánica construida, y el trabajo se centra en la parte electrónica, así como en la programación del software para el microcontrolador, y en llevar a cabo la puesta en marcha del sistema de tres ejes completos.

6. Desarrollo y programación del microcontrolador

6.1. Punto de partida

En la realización de la programación del microcontrolador se ha tratado de simplificar el proceso en la medida de lo posible.

Se ha empleado el “STM32Cube”, creado por STMicroelectronics con la finalidad de reducir el esfuerzo, tiempo y coste de desarrollo a la hora de llevar a cabo un programa. Más concretamente, en este caso se emplea el “STM32CubeL0”.

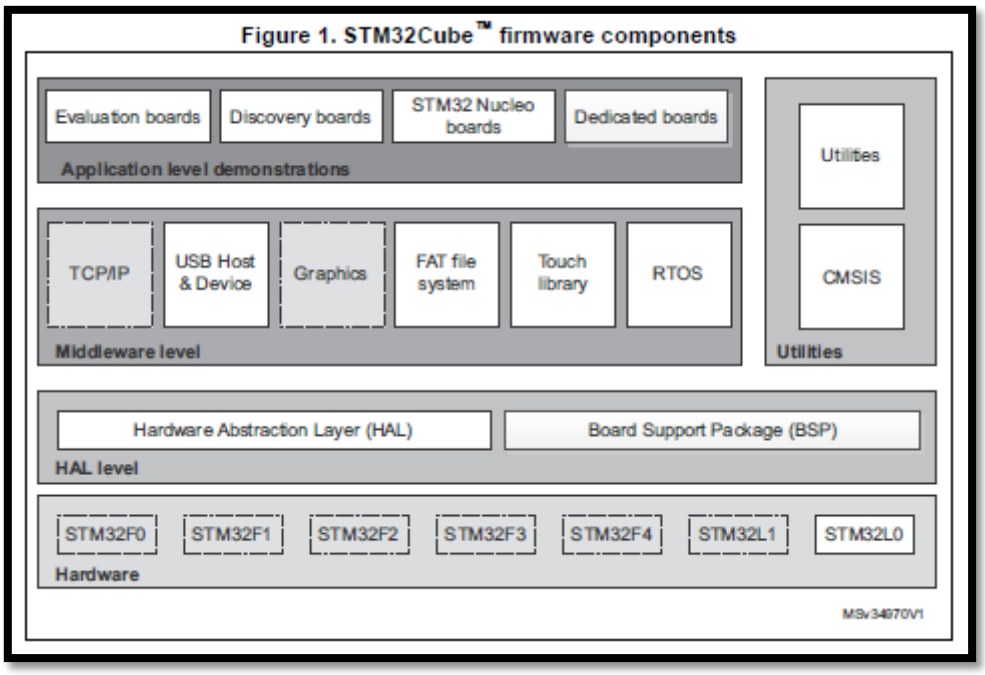


Ilustración 37: Estructuración STM32Cube, www.st.com

Su firmware se divide en tres niveles:

1. **Nivel 0:** Se divide en tres subniveles:

   - **"Board Support Package" (BSP):** Ofrece APIs relacionadas con el hardware de los componentes (LCD, Micro SD…).

   - **Hardware Abstraction Layer (HAL):** Proporciona los drivers de bajo nivel y los métodos de interfaz del hardware para interactuar con los niveles más altos (aplicaciones, librerías…).

   - **Basic peripheral usage examples:** Contiene ejemplos de operaciones básicas de los periféricos del STM32L0.

2. **Nivel 1:** Se divide en dos subniveles:

   - **Middleware components:** Conjunto de librerías que cubre los siguientes puntos:

     - USB Device Library.
     - FreeRTOS.
     - FAT File system.
     - STM32 Touch Sensing Library.

- **Examples based on the Middleware components:** Cada componente viene acompañado de uno o varios ejemplos.

3. **Nivel 2:** Está compuesto por un único nivel, que cuenta con tiempo real global y demostraciones gráficas.

En la siguiente imagen, se puede observar cómo se distribuyen los diferentes niveles anteriormente nombrados:

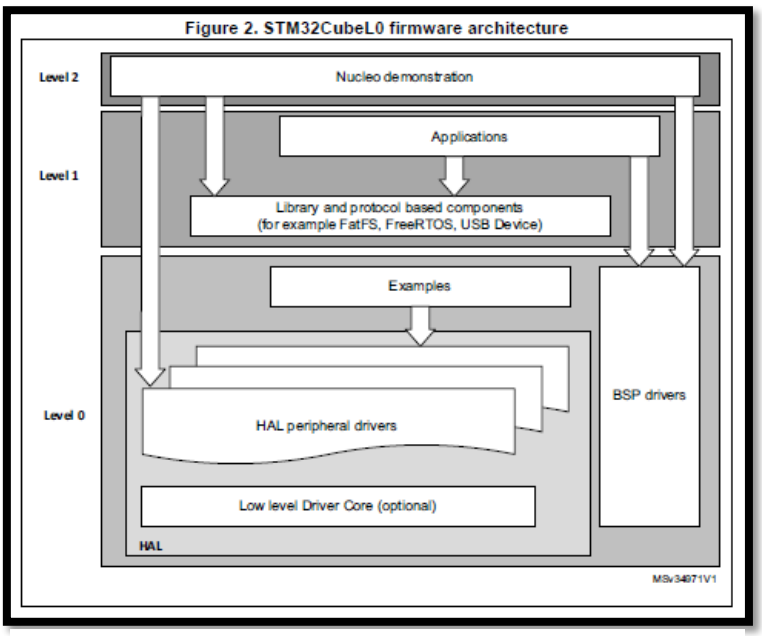


Ilustración 38: Arquitectura STM32Cube, www.st.com

De este modo, se simplifica en gran medida la configuración de las entradas y salidas que se desea emplear, ya que se lleva a cabo de una manera intuitiva, rápida, reduciendo así la complejidad de dicha tarea.

En la siguiente imagen se muestra, dentro del software de ayuda, los pines disponibles del microcontrolador, y que al seleccionarlos se dispone de todas las posibles opciones de configuración.

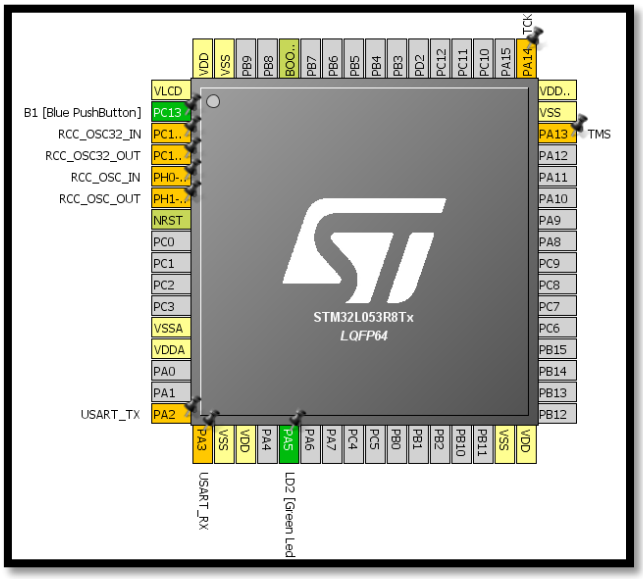
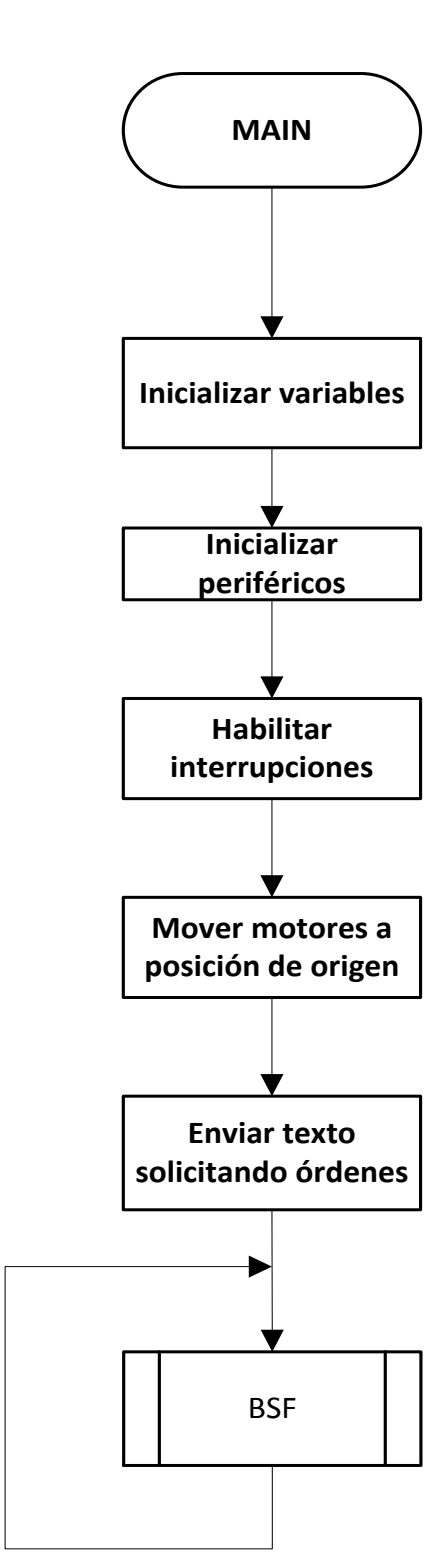
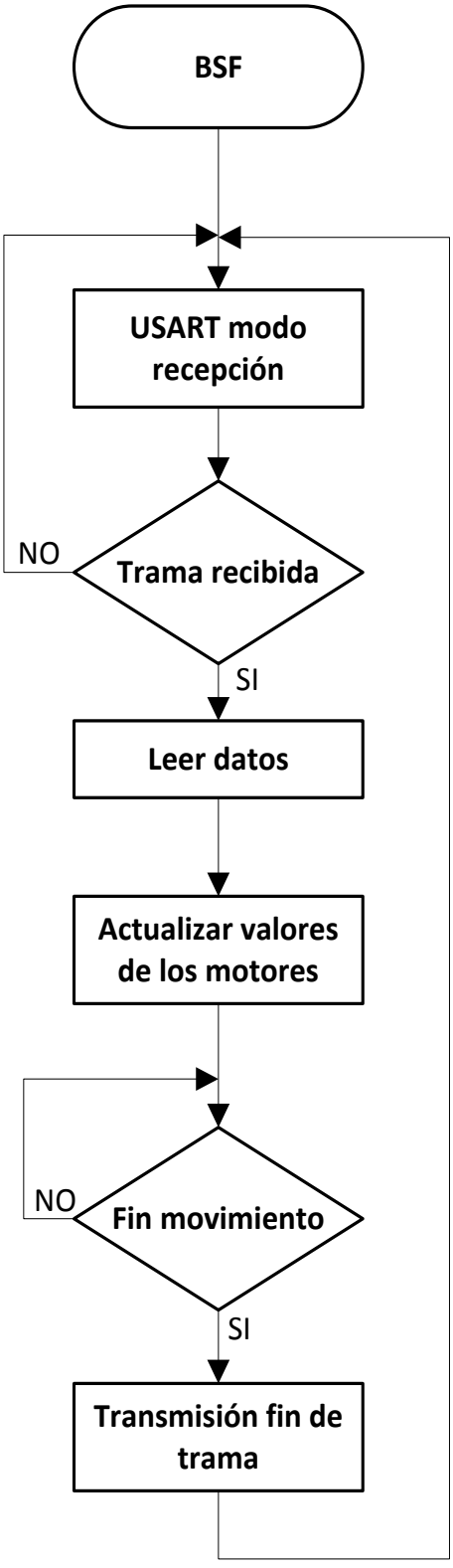
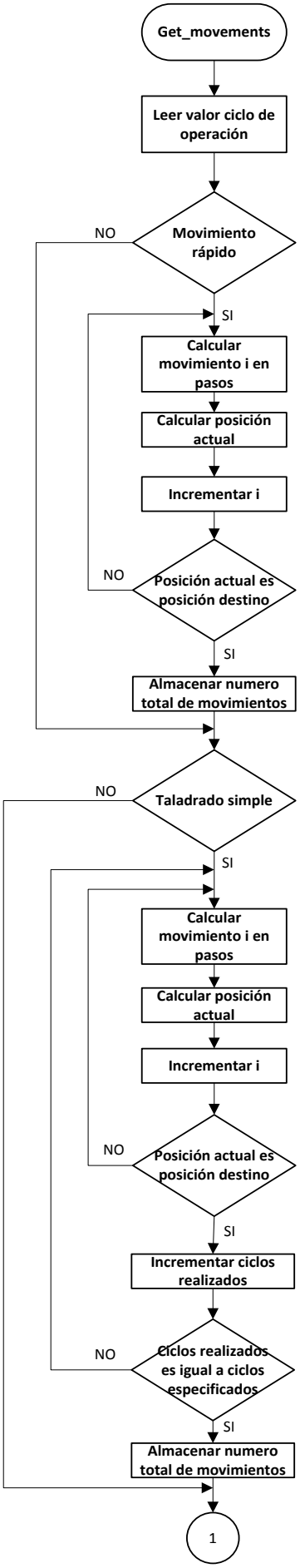


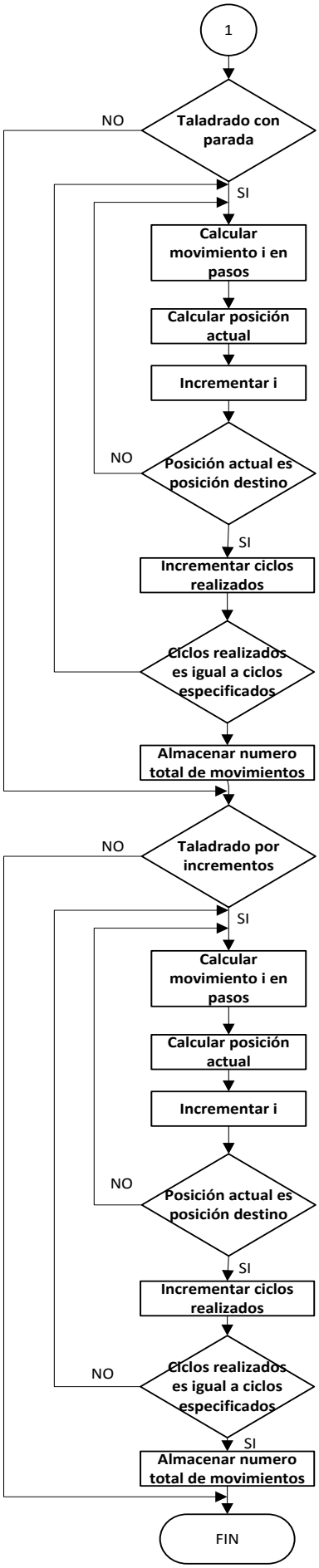
Ilustración 39: Configuración STM32CubeL0

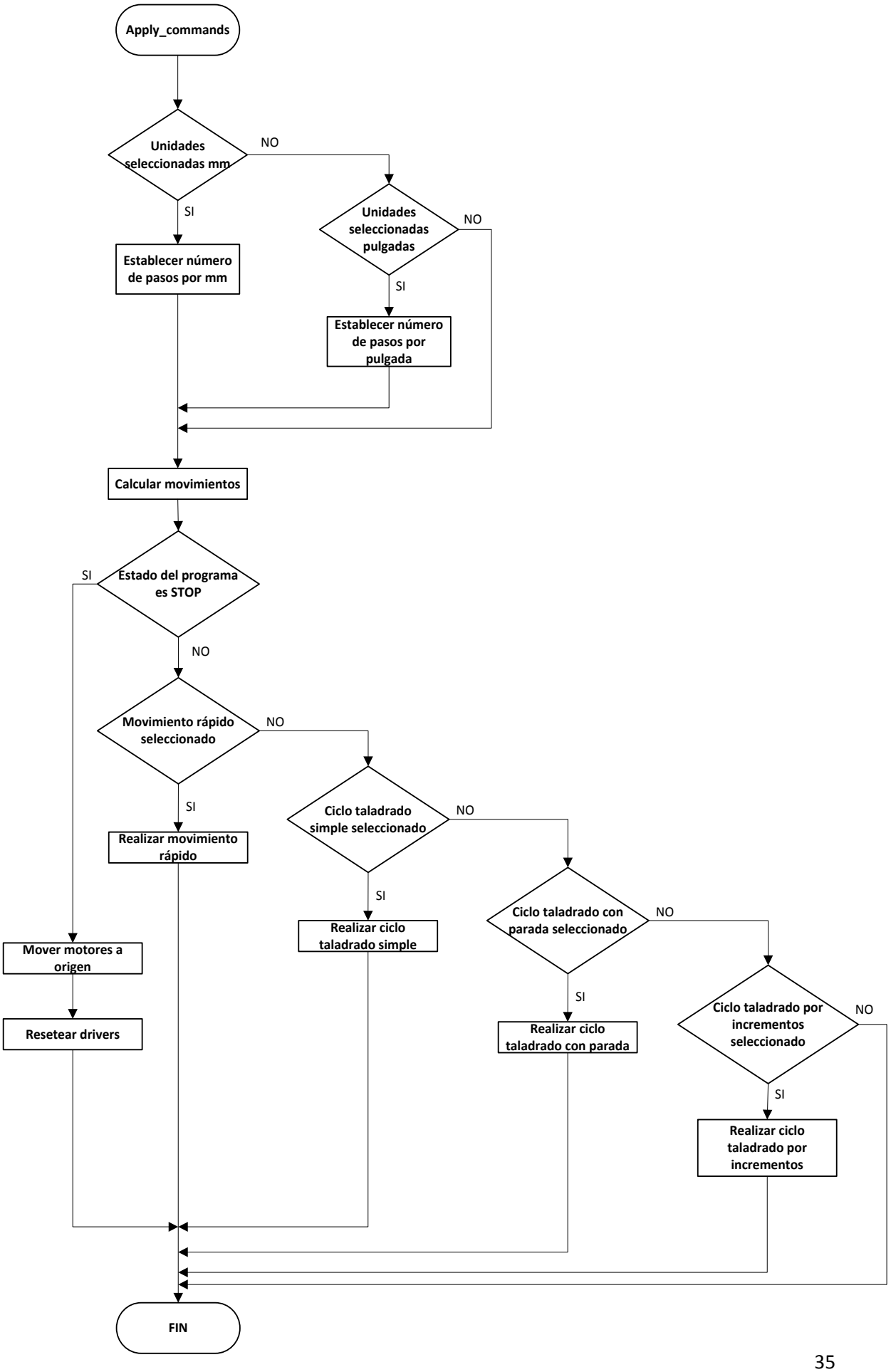
6.2. Flujogramas






Get_movements
Leer valor ciclo de operación
Movimiento rápido
NO
SI
Calcular movimiento i en pasos
Calcular posición actual
Incrementar i
Posición actual es posición destino
NO
SI
Almacenar numero total de movimientos
Taladrado simple
NO
SI
Calcular movimiento i en pasos
Calcular posición actual
Incrementar i
Posición actual es posición destino
NO
SI
Incrementar ciclos realizados
Ciclos realizados es igual a ciclos especificados
NO
SI
Almacenar numero total de movimientos
1


1
Taladrado con parada
NO
SI
Calcular movimiento i en pasos
Calcular posición actual
Incrementar i
Posición actual es posición destino
NO
SI
Incrementar ciclos realizados
Ciclos realizados es igual a ciclos especificados
Almacenar numero total de movimientos
Taladrado por incrementos
NO
SI
Calcular movimiento i en pasos
Calcular posición actual
Incrementar i
Posición actual es posición destino
NO
SI
Incrementar ciclos realizados
Ciclos realizados es igual a ciclos especificados
NO
SI
Almacenar numero total de movimientos
FIN


Apply_commands
Unidades seleccionadas mm
NO
SI
Unidades seleccionadas pulgadas
NO
SI
Establecer número de pasos por mm
Establecer número de pasos por pulgada
Calcular movimientos
Estado del programa es STOP
SI
NO
Movimiento rápido seleccionado
NO
SI
Realizar movimiento rápido
Ciclo taladrado simple seleccionado
NO
SI
Realizar ciclo taladrado simple
Ciclo taladrado con parada seleccionado
NO
SI
Realizar ciclo taladrado con parada
Ciclo taladrado por incrementos seleccionado
NO
SI
Realizar ciclo taladrado por incrementos
Mover motores a origen
Resetear drivers
FIN

6.3. Configuración de E/S digitales

Para llevar a cabo el sistema, así como su control, se precisa del empleo de entradas y salidas digitales, así como generar señales PWM.
Las entradas y salidas digitales utilizadas son:

- 3 entradas digitales correspondientes a los finales de carrera para limitar los movimientos máximo y mínimo en los tres ejes X, Y, Z.

FUNCIÓN	PIN
X mín	PA13
Y mín	PA14
Z mín	PA15

- 3 salidas digitales para controlar el sentido de giro de cada uno de los motores.

FUNCIÓN	PIN
Sentido giro motor eje X	PA8
Sentido giro motor eje Y	PB5
Sentido giro motor eje Z	PB4

- 2 salidas digitales, una para resetear el driver y otra empleada como "flag".

FUNCIÓN	PIN
Reset	PA9
Flag	PA10

6.4. Temporizadores y generación de señales PWM

Para llevar a cabo el control de los motores paso a paso, es preciso el empleo de señales PWM, concretamente de tres señales cada una de las cuales ordenará al driver el movimiento del motor en el número de pasos como de pulsos se envíen. En cuanto a las señales PWM, han de estar enlazadas y controladas por un timer, ya que de este modo se puede variar la frecuencia de envío de pulsos y así comandar tanto la posición como la velocidad de los motores.

Así pues, cada señal PWM empleada, lleva consigo la utilización de un timer, habiendo escogido como se puede ver en la tabla siguiente:

FUNCIÓN	PIN
PWM1 – TIM1	PC7
PWM2 – TIM2	PB3
PWM3 – TIM0	PB10

6.5. Comunicación USART

Para el envío de órdenes a la máquina, más concretamente de comandos Gcode, para que se ejecuten y los motores puedan llevar a cabo la tarea, previamente han de ser recibidas por el microcontrolador.

Este punto supone uno de los principales, y se realiza por comunicación por el puerto serie, es decir, empleando la comunicación USART (Universal Asynchronous Receiver-Transmitter) entre el pc y el microcontrolador, aunque en este caso, como las órdenes no han de llegar en instantes determinados y preestablecido, la señal de reloj no va a ser necesaria, y la comunicación y envío de comandos se llevará a cabo de manera asíncrona.

Este sistema de comunicación precisa del uso de dos señales:

- **Transmisión (TX):** Se llevará a cabo la salida de datos.
- **Recepción (RX):** Tendrá lugar la entrada de datos.

Los pines empleados para tal fin, así como la velocidad de transmisión de datos son:

FUNCIÓN	PIN
TX puerto serie	PA2
RX puerto serie	PA3
Velocidad envío puerto serie:	19200 baudios

6.6. Comunicación SPI

La comunicación entre el microcontrolador y los drivers, los cuales que conectan superpuestos, se realiza por medio del bus SPI.

Empleando una señal de reloj, envío de datos, recepción de datos, así como una conexión correspondiente a la selección del chip.

Es pues, un bus de comunicaciones sincrónico en el cual a cada pulso de la señal de reloj se envía un bit de datos al esclavo seleccionado mediante el pin correspondiente a la selección del dispositivo deseado (chip select).

El siguiente esquema muestra a rasgos generales el funcionamiento de este bus de comunicaciones, que como en este proyecto, se compone de un maestro (microcontrolador) y tres esclavos (drivers para los motores):

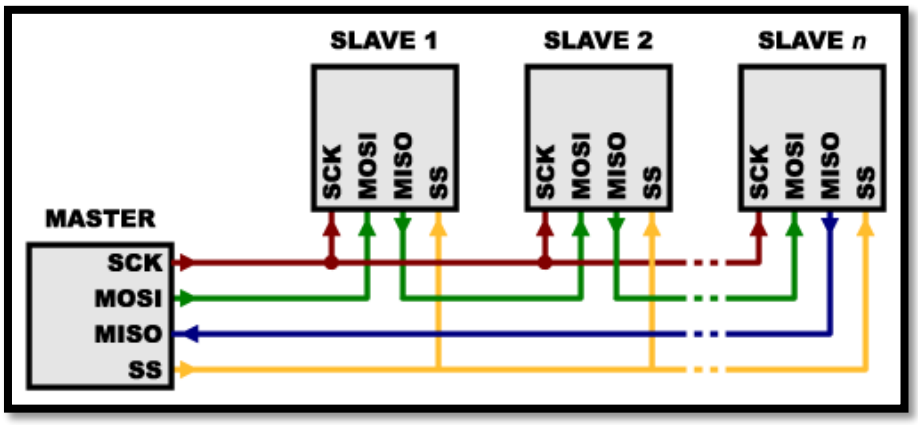


Ilustración 40: Esquema conexionado bus de comunicación SPI, www.prometec.net

Que como se puede observar, cuenta con 4 hilos:

- **SCLK (Clock):** Señal de reloj del busLos pulsos generados por este pin marcan la sincronización. A cada pulso del reloj, se envía o se lee un bit de datos.
- **MOSI (Master Out Slave In):** Salida de datos del maestro y entrada al esclavo.
- **MISO (Master In Slave Out):** Salida de datos del esclavo y entrada al maestro.
- **SS/CS:** "Chip select", selecciona un esclavo para que se active.

De este modo, para la comunicación entre dispositivos electrónicos es un bus de comunicaciones que emplea los mismos hilos, tanto para enviar como para recibir datos, convirtiéndolo en un sistema sencillo y eficaz para sistemas como el abordado en el presente proyecto.

En cuanto a los pines que se emplean para este fin, son los siguientes:

FUNCIÓN	PIN
SPI chip select	PB6
SPI SCLK	PA5
SPI MISO	PA6
SPI MOSI	PA7

6.7. Conjunto de pines empleados

Una vez establecidos y ubicados todos los pines que se van a emplear del microcontrolador, se pueden agrupar todos en una única tabla para tener una visualización general:

PUERTO	PIN	UTILIDAD
PA	13	**X_min (DI)**
PA	14	**Y_min (DI)**
PA	15	**Z_min (DI)**
PC	7	**Control motor X (PWM1)**
PB	3	**Control motor Y (PWM2)**
PB	10	**Control motor Z (PWM3)**
PA	8	**Dirección motor X (DO)**
PB	5	**Dirección motor Y (DO)**
PB	4	**Dirección motor Z (DO)**
PA	10	**Flag (EXTI_FALLING)**
PA	9	**Reset (DO)**
PB	6	**SPI CS (DO)**
PA	5	**SPI SCK (AF-SPI1)**
PA	6	**SPI MISO (AF-SPI1)**
PA	7	**SPI MOSI (AF-SPI1)**
PA	3	**UART RX (AF-USART2)**
PA	2	**UART TX (AF-USART2)**

6.8. Conjunto de comandos Gcode necesarios

Como se ha podido ver anteriormente, en el apartado de introducción a comandos Gcode, la cantidad de comandos es muy amplia, ya que abarca, además de variedad de movimientos, las diferentes aplicaciones para las cuales puede ser utilizada (fresadora, taladradora, impresora 3D…). Pero en este caso, dado que la máquina CNC está pensada en un principio para únicamente realizar tareas de taladrado, no se configurará la totalidad de comandos, sino solo los necesarios para dicha tarea.

Así pues, los comandos implementados, y que por tanto, serán los que comprenderá la máquina de control numérico en cuestión, así como el efecto de cada uno de ellos son:

- **Modo distancia:**
  - **G90:** Modo de distancia absoluta, es decir, las posiciones que se envíen deben ser respecto al punto 0 (homming).
  - **G91:** Mode de distancia incremental, según el cual las coordenadas representan el incremento con respecto a la posición actual, ya sea 0 u otra cualquiera.

- **Unidades:**
  - **G20:** Pulgadas como unidad de trabajo seleccionada.
  - **G21:** Milímetros como unidad de trabajo seleccionada.

- **Estado del sistema:**
  - **Por defecto:** Se encontrará en estado “run” (marcha).
  - **M2:** Indica la finalización del programa completo (fin).

- **Tipo de ciclo:**

  - **G0 → MOVIMIENTO RÁPIDO:**

```
G0 axes
```

    Este ciclo ordena la realización de un movimiento rápido según el valor especificado en cada uno de los ejes.

  - **G81 → CICLO DE TALADRADO:**

```
G81 (X- Y- Z-) or (U- V- W-) R- L-
```

    Este ciclo consiste en un simple proceso de taladrado, cuyo funcionamiento se lleva a cabo en los siguientes pasos:

    1. Movimiento rápido en el plano XY desde la posición actual hasta la posición especificada donde se pretende realizar el taladrado.
    2. Movimiento del eje Z a la velocidad de taladrado (F) desde la posición actual hasta la especificada en el comando Z.
    3. Retorno a velocidad rápida a la posición Z inicial (Z - homming).

  - **G82 → CICLO DE TALADRADO CON ESPERA:**

```
G82 (X- Y- Z-) or (U- V- W-) R- L- P-
```

    Este ciclo consiste en el mismo proceso de taladrado que el G81, pero con la variante de que se realiza una parada programada al final del orificio. El funcionamiento se lleva a cabo en los siguientes pasos:

    1. Movimiento rápido en el plano XY desde la posición actual hasta la posición especificada donde se pretende realizar el taladrado.
    2. Movimiento del eje Z a la velocidad de taladrado (F) desde la posición actual hasta la especificada en el comando Z.
    3. Realizar una pausa de “P” segundos.
    4. Retorno a velocidad rápida a la posición Z inicial (Z - homming).

- **G83 → CICLO DE TALADRADO CON PICOTEO:**

```
G83 (X- Y- Z-) or (U- V- W-) R- L- Q-
```

Este ciclo consiste en un proceso de taladrado en diferentes pasos, es decir, según el cual se va incrementando la profundidad del orificio en "Q" mm hasta alcanzar el valor de Z final. Tras cada incremento, la herramienta retrocede en el eje Z, hasta la posición R inicial. Tras alcanzar el valor de Z final, la herramienta se desplaza en el eje Z hasta la posición inicial. El funcionamiento se lleva a cabo en los siguientes pasos:

1. Movimiento rápido en el plano XY desde la posición actual hasta la posición especificada donde se pretende realizar el taladrado.
2. Movimiento del eje Z a la velocidad de taladrado (F) desde la posición actual hasta actual + Q, con límite el valor de Z final.
3. Movimiento rápido de vuelta en el eje Z a la posición R.
4. Movimiento rápido hasta la profundidad actual del orificio.
5. Repetición de los pasos 2, 3 y 4 hasta alcanzar la posición Z final.
6. Retorno a velocidad rápida a la posición Z inicial (Z - homming).

6.9. Lectura de comandos Gcode recibidos

Una vez con todas las conexiones y comunicaciones del sistema establecidas, se precisa de comenzar con la programación en lenguaje C del microcontrolador, donde se han implementado las funciones correspondientes y necesarias para la realización de las tareas propias.

En primer lugar, se ha realizado el envío de comandos Gcode desde el pc al microcontrolador (RX) y a la inversa, es decir, desde el microcontrolador al pc (TX).

Para ello se ha empleado un programa de software libre llamado "termite", el cual simplemente envía y recibe comandos o tramas por un determinado canal del puerto serie, de modo que se ha podido verificar que el microcontrolador y el pc se comunicaban sin problemas.

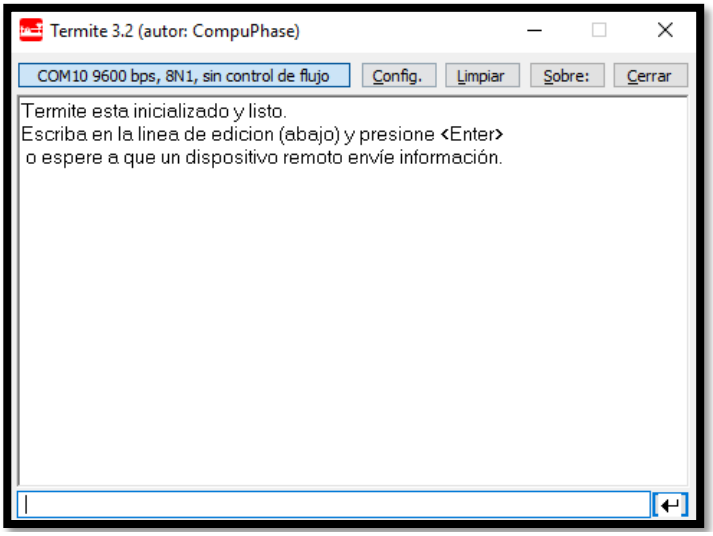


Ilustración 41: Captura de pantalla programa terminal "Termite"

Pero para llevarlo a cabo, se ha precisado de un convertidor USB-TTL (YP-01) para así convertir el conector USB del pc en un puerto serie.

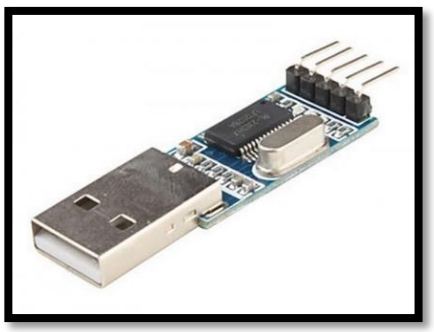


Ilustración 42: Convertidor USB - TTL (YP-01)

Ahora bien, una vez enviando y recibiendo datos por el puerto serie, se precisa que el microcontrolador sea capaz de, tras recibir una trama de lenguaje Gcode, desglosar los valores, siempre teniendo en cuenta a que comando pertenece cada uno de ellos.
Para tal fin, se ha creado la siguiente función:

"**void Read_command_Value(char Command)"**

Esta función comienza a buscar el valor que acompaña al comando especificado (G, X, Y, Z, F, P, Q ó R), y tras localizalo, lo almacena en una estructura que contiene las variables de dichos comandos creadas.

Se emplea una segunda función:

"void Read_from_buffer ()"

Lleva implícita la llamada a la función anteriormente nombrada, de modo que realiza la búsqueda de los valores de todos los comandos contenidos en la cadena de datos recibida.

Posteriormente, los valores contenidos en esa estructura serán los que se empleen para llevar a cabo las actualizaciones en el movimiento de los motores.

6.10. Movimiento de los motores

En primer lugar, al iniciarse el sistema, todos los motores deberán desplazarse a la posición de origen (0, 0, 0), la cual será alcanzada cuando los motores entren en contacto con los finales de carrera correspondientes al movimiento marcha atrás. A partir de ahí, ya se podrá conocer la posición de cada uno de los ejes, así como controlar su velocidad.
Pero para ello, se debe ordenar el movimiento de los motores, y que se realiza de la siguiente manera:
El microcontrolador comunica con el driver de cada uno de los motores mediante el bus SPI, como se ha explicado anteriormente. De este modo, especificará mediante una salida digital el sentido de giro del mismo (backward/forward), al igual que se le especificará el tipo de control escogido (full step/half step/microstepping…). Por último, será la señal PWM introducida la que ordenará por cada pulso que se genere que el motor se desplace un paso en la dirección y tipo de paso especificados.

Además, en función de la velocidad con que se envíen sucesivos pulsos, variará la velocidad de giro del motor.
Para el movimiento de los motores, se ha configurado las señales PWM así como sus respectivos temporizadores, y se emplea la siguiente función:

"BSP_MotorControl_Move(Axis, Direction, Steps)"

Dicha función ordena el movimiento del motor especificado en el número de pasos que se le indique.

6.11. Conversión giro del motor a desplazamiento lineal

El movimiento de los motores se traslada a los elementos móviles del sistema mediante elementos de transmisión (correas, poleas, husillos…).
Como la base de la máquina CNC es ordenar a los motores que muevan la herramienta a un punto concreto expresado en milímetros, se necesita conocer la cantidad de pasos que se necesita ordenar a los motores de cada eje para que la herramienta se desplace 1 mm respecto a la posición anterior.
Para conocerlo, se ha analizado en detalle los elementos de transmisión de los que estaba compuesto el sistema, y posteriormente, se ha analizado la relación de transmisión, así como los valores de movimiento del motor que se representan en la hoja de características del mismo. De este modo, cuando se reciba el valor de los Gcode en milímetros o pulgadas, este factor dará a conocer el número de pasos que debe dar el motor para alcanzar la posición.

- Correa dentada, con poleas dentadas de diferente medida:

Ilustración 43: Correa y poleas dentadas

Para los ejes “X” y “Z” del sistema la relación de transmisión es la misma, con lo que el mismo cálculo será válido. El eje Y cuenta con otro diámetro de polea además de otro husillo.

Dichas relaciones vienen dadas por los siguientes elementos o transmisiones:

1. Polea entrada:

- **Ejes X, Y, Z:**

$$Z_e = 15\ dientes$$

$$N_e \rightarrow Velocidad\ eje$$

2. Polea salida:

- **Ejes X, Z:**

$$Z_s = 25\ dientes$$

- **Eje Y:**

$$Z_s = 30\ dientes$$

$$N_s \rightarrow Velocidad\ eje$$

Con lo que se puede conocer la relación entre la entrada y la salida de estos sistemaS de transmisión, y que son:

$$Z_e \cdot N_e = Z_s \cdot N_s$$

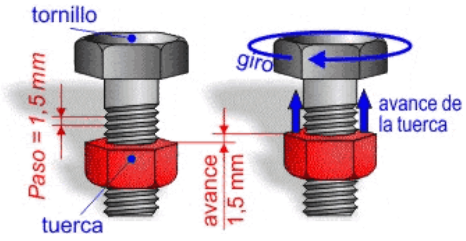
- **Ejes X, Z:**

$$\frac{N_s}{N_e} = \frac{Z_e}{Z_s} = \frac{15}{25} = 0.6$$

- **Eje Y:**

$$\frac{N_s}{N_e} = \frac{Z_e}{Z_s} = \frac{15}{30} = 0.5$$

- Husillo, conectado al eje de la polea secundaria. Este husillo tiene un paso de rosca de 7 mm para los ejes “X” y “Z” y de 10 mm para el eje “Y” , con lo que:



$$\boldsymbol{Avance\ (ejes\ X, Z) = 7\,\frac{mm}{revolución}}$$

$$\boldsymbol{Avance\ (eje\ Y) = 10\,\frac{mm}{revolución}}$$

Finalmente, se puede conocer la relación de transmisión final (correa + husillo), mediante las dos relaciones anteriormente calculadas.

$$\frac{Pasos}{revolución} = 16 \cdot 200 = 3200$$

$$\boldsymbol{Ejes\ "X"\ y\ "Z"} \rightarrow \frac{Revoluciones}{mm} = \frac{1}{7} = 0.143$$

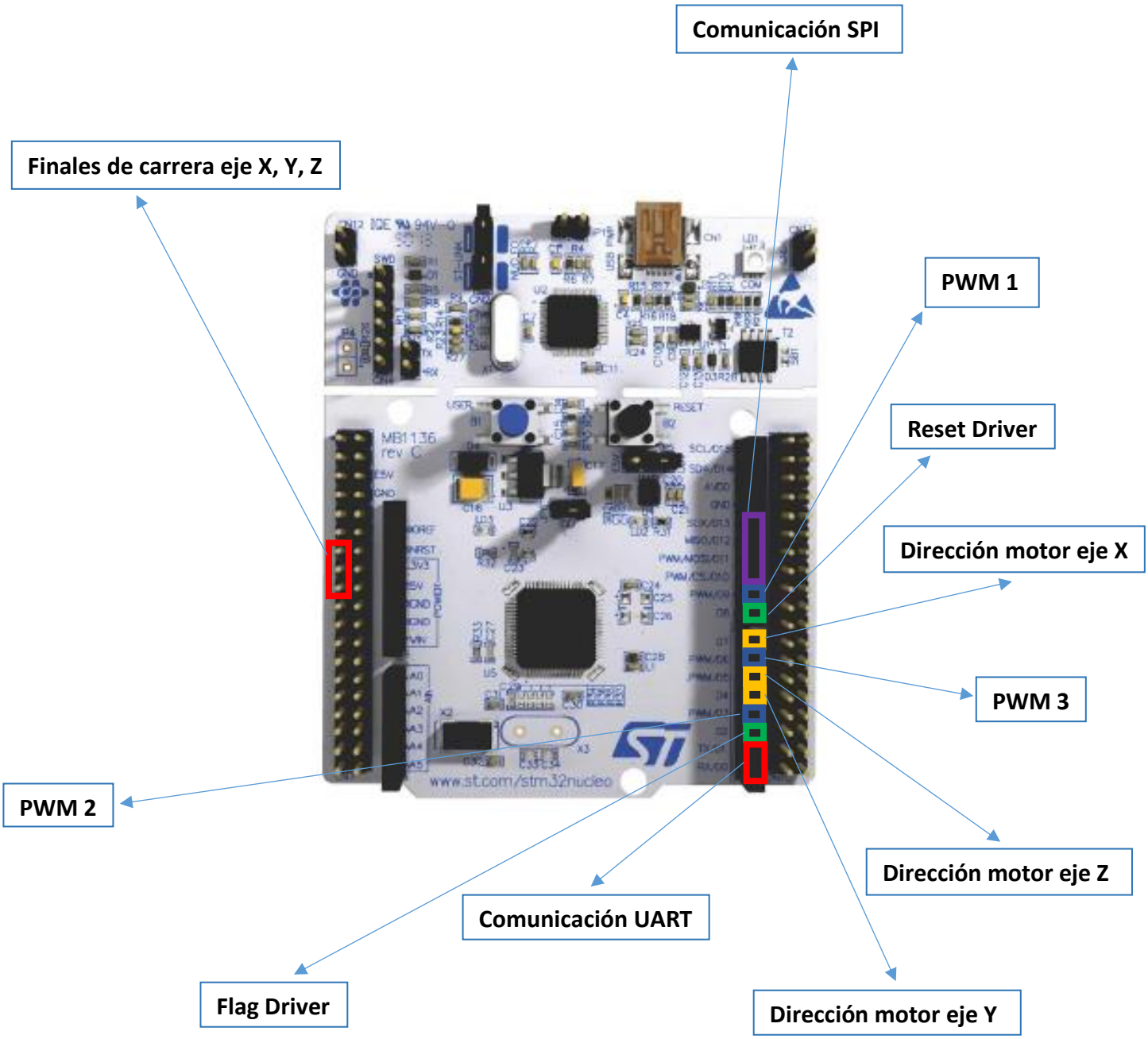
$$\boldsymbol{Eje\ "Y"} \rightarrow \frac{Revoluciones}{mm} = \frac{1}{10} = 0.1$$

$$Relación\ ejes\ "X"\ y\ "Z" = 1.66 \cdot (0.143 \cdot 3200) = 1.66 \cdot 457.6 = \boldsymbol{760\ \frac{pasos}{mm}}$$

$$Relación\ eje\ "Y" = 2 \cdot (0.1 \cdot 3200) = 2 \cdot 320 = \boldsymbol{640\ \frac{pasos}{mm}}$$

Por tanto, con los motores correspondientes a los ejes “X” y “Z”, se necesita dar 760 pasos, y con el motor del eje “Z” 640 pasos para recorrer linealmente 1 mm.

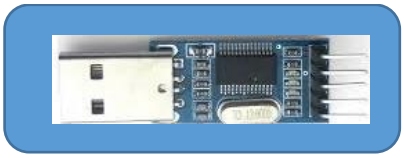
7. Diseño y conexionado de elementos electrónicos



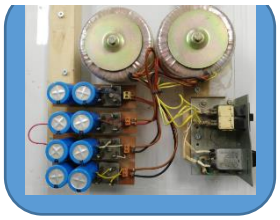
UNIVERSIDAD POLITECNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería del Diseño

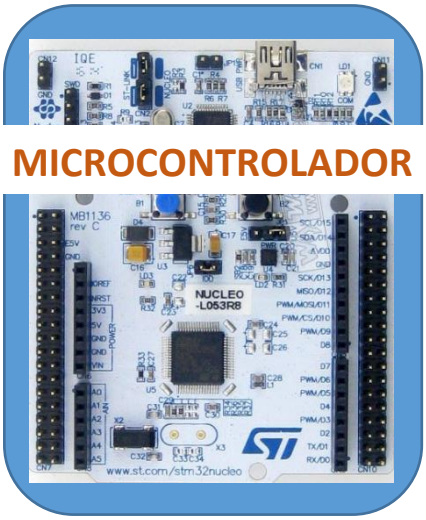
CONVERTIDOR USB - SERIE

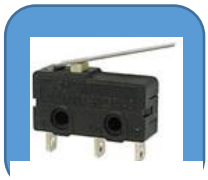


F. ALIMENTACIÓN



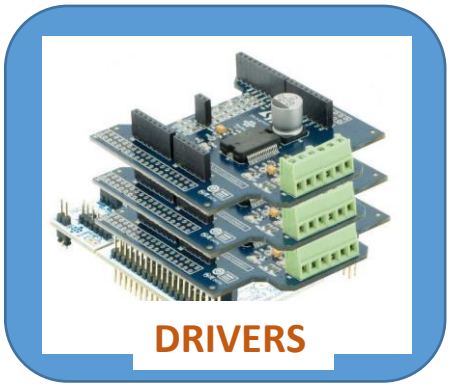
ANTIREBOTE





FC EJE X

ANTIREBOTE



DRIVERS

ANTIREBOTE

FC EJE Y

FC EJE Z

MOTOR EJE X

MOTOR EJE Y

MOTOR EJE Z

8. Puesta en marcha del sistema

Finalmente, con la programación correcta y verificada paso a paso, así como con el conexionado correcto de todos los elementos, se ha realizado la puesta en marcha de la máquina CNC con el fin de comprobar que la máquina lleva a cabo los movimientos.

Tras sucesivas pruebas a diferentes velocidades, así como aceleraciones y deceleraciones, se ha podido observar que cuenta con una carga bastante importante que debe vencer cada uno de los ejes. Este factor ha llevado a que, para tratar de intentar que trabaje a una velocidad relativamente rápida, se generaban ruidos en los momentos en los cuales trataba de incrementar su velocidad hasta la máxima del movimiento.

Para descartar la posibilidad de algún otro problema, el mismo software y electrónica ha sido colocado para el control de una máquina CNC casera, concretamente una impresora 3D. Al ponerla en funcionamiento se pudo ver que, en este caso, dado que las cargas a vencer eran bastante menores que para el caso de la máquina del laboratorio, se movía a altas velocidades sin problemas de aceleración ni de ruido.

Dado que el trabajo se centra en la puesta en marcha de la máquina CNC del laboratorio, ha sido de la cual se ha pretendido grabar un pequeño vídeo, que aunque no se aprecie demasiado, es posible comprobar que el software desarrollado funciona correctamente, y que la máquina interpreta la información y órdenes que se le envían.

9. Posibles mejoras

El sistema realizado se corresponde, como se ha nombrado a lo largo del documento, con un sistema de coste reducido, pero que como cualquier sistema o máquina, es posible mejorar tanto la parte mecánica, como la parte electrónica, así como incluso llevar a cabo modificaciones en el software que faciliten la labor de utilizarla o para mejorar aspectos o comportamientos no deseados que puedan observarse durante el funcionamiento normal de la máquina.

Algunas de estas mejoras podrían ser:

- **Parte mecánica:**

Dado que la estructura y el resto de los elementos que componen la mayor parte de la máquina son de acero inoxidable, es muy resistente, por lo que no es necesario modificarlo.

En cambio, con el fin de evitar problemas futuros, podría plantearse la idea de sustituir las varillas lisas calibradas por otras de mayor diámetro, aunque dándole un uso adecuado no debería haber problemas de flexión con las actuales.

De igual manera, podría tratarse de reducir posibles rozamientos, que puedan empeorar el movimiento.

- **Parte electrónica:**

En la parte electrónica, sí que es posible llevar a cabo una serie de modificaciones, algunas de las cuales podrían ser con el fin de mejorar, mientras que otras únicamente conseguirían incrementar el coste final de la máquina.

Algunas de estas modificaciones podrían ser:
- Utilizar motores paso a paso de mayor potencia, en caso de ser necesario.
- Realimentar el sistema, para llevar a cabo el control de los motores en bucle cerrado mediante el empleo de encoders, para limitar la posible pérdida de pasos.
- Sustituir los motores paso a paso actuales por servomotores, lo que dotaría al sistema de una mayor precisión.

- **Programación:**

En cuanto a la programación, es posible llevar a cabo mejoras de una manera gradual, es decir, aunque el funcionamiento global de la máquina sea el correcto, conforme se vaya usando es posible corregir ciertos detalles que, o bien no gusten, o bien no deban producirse, además de ampliar las funcionalidades de la misma en caso de ser necesario.

Por tanto, no es posible determinar estas mejoras de programación de manera específica ya que dependen totalmente de los requerimientos y objetivos de cada persona.

10. Conclusión

A la vista de la diversidad de elementos necesarios para llevar a cabo una máquina CNC, así como de ciertos conocimientos tanto mecánicos como electrónicos y de programación, se puede calificar totalmente como un sistema mecatrónico.

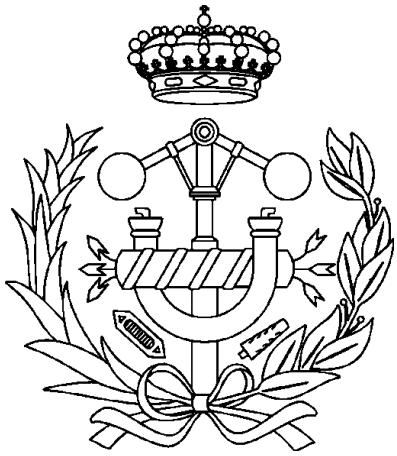
Así pues, y como se ha podido demostrar, se han ido abordando todos los aspectos de interés en relación al tema del proyecto, adquiriendo conocimientos de los cuales no se disponía, así como aprovechando los que ya se tenían gracias a los estudios finalizados.

En cuanto al proyecto una vez terminado, ha sido de gran ayuda para una dedicación y aprendizaje interesantes en este tramo final del "Máster en Ingeniería Mecatrónica", ya que se ha tratado con elementos electrónicos, así como mecánicos a grandes rasgos, habiendo estudiado su importancia y funcionamiento, así como los precios que tienen actualmente en el mercado.

Además, se trata de un proyecto en el cual se han alcanzado los objetivos fijados al inicio del mismo y de manera correcta y satisfactoria.

11.Bibliografía

- Apuntes de poliformat de la asignatura de “Sistemas embebidos” del “Máster en Ingeniería Mecatrónica”.
- Apuntes de poliformat de la asignatura de “Diseño Electrónico Avanzado” del “Máster en Ingeniería Mecatrónica”.
- https://developer.mbed.org/components/X-NUCLEO-IHM01A1-Stepper-Motor-Driver/
- http://www.st.com
- https://developer.mbed.org/
- http://www.miklor.com/COM/UV_Drivers.php
- https://es.wikipedia.org/wiki/Serial_Peripheral_Interface
- http://www.prometec.net/bus-spi/
- https://es.wikipedia.org/wiki/Universal_Asynchronous_Receiver-Transmitter
- http://www.mioio.es/comunicacion-serie-uart/
- http://picfernalia.blogspot.com.es/2012/06/comunicaciones-puerto-serie-uart.html
- http://hotboards.org/index.php/es/blog/22-spanish/blog/stm32cube/94-transmision-serial-uart
- http://www.keil.com/
- Hoja de datos de los diferentes componentes electrónicos.



Anexo I:

Programación microcontrolador

DEFINICIONES

- ## DEFINICIONES GENERALES

```
#define NUM_AXIS 3          // Total number of axis
#define RX_BUFFER_SIZE 80   // Reception buffer size

#define X_AXIS 0    // X-axis identity
#define Y_AXIS 1    // Y-axis identity
#define Z_AXIS 2    // Z-axis identity

#define X_MIN_POSITION 0    // X-axis min. position in mm
#define Y_MIN_POSITION 0    // Y-axis min. position in mm
#define Z_MIN_POSITION 0    // Z-axis min. position in mm

#define X_MAX_POSITION 400  // X-axis max. position in mm
#define Y_MAX_POSITION 400  // X-axis max. position in mm
#define Z_MAX_POSITION 400  // X-axis max. position in mm

// Distance mode
#define ABSOLUTE_MODE 0          // G90 (default)
#define INCREMENTAL_MODE 1       // G91

// Units used
#define MM 0                     // G21 (default)
#define INCHES 1                 // G20

// Program state
#define RUN 0                    // default
#define STOP 1                   // M2

// Speed values
#define DRILLING_SPEED 8000
#define FAST_SPEED 10000

// Operation cycles
#define NONE_OR_STOP_CYCLE 0       // G80 (default)
#define DRILLING 1                 // G81 Simple drilling
#define DRILLING_DWELL 2           // G82 Drilling with dwell
#define DRILLING_PECK 3            // G83 Drilling with peck
#define FAST_MOVEMENT 4            // G0 Fast movement to a position
```

- **VARIABLES DEFINIDAS**

```
extern int Complete_line;

int i, init, end, j, number;

uint8_t Initial_TxBuffer[] = "Send Gcode order \n"; // Message send to
order more commands
uint8_t TxBuffer_OK[] = "OK \n";    // Message send when a Gcode command
has been received completely
uint8_t RxBuffer[RX_BUFFER_SIZE];   // Array where the received data is
kept
char Conversion[7];
float axis_steps_per_mm[NUM_AXIS]={760.0, 640.0, 760.0};   // Steps/mm
conversion
float axis_steps_per_inch[NUM_AXIS]={19304.0, 16256.0, 19304.0}; //
Steps/inch conversion
float Conversion_selected[NUM_AXIS];        // Array with each axis
conversion in selected units
float Actual_position[NUM_AXIS] = {0.0, 0.0, 0.0};      // Actual position
(X, Y, Z)

float Movement[50];     // Array for storing all the necessary movements
for a Gcode order
int Total_movements;    // Total number of movements for each Gcode order
int Direction[50];      // Array for storing the direction of each specific
movement
float Dwell_time;       // Time in miliseconds to wait at the bottom of the
hole

float Value;    // Variable to store the array to float converted value

__IO ITStatus UartReady = SET;
UART_HandleTypeDef UartHandle;

typedef struct      // Structure to store all the received configuration
parameters
{
    uint8_t Distance_Mode;      // Absolute or relative
    uint8_t Units;              // Inches or mm
    uint8_t Program_State;      // Run, pause or stop
    uint8_t Operation_Cycle;    // Simple drilling, drilling with dwell,
peck drilling or none.

}Working_Config;
```

```
typedef struct      // Structure to store all the parameter values
(position, speed, number of cycles...)
{
    float X_position;   // X position
    float Y_position;   // Y position
    float Z_position;   // Z position
    float R;            // retract Z position
    float P;            // Dwell time at the bottom of the hole
    float Q;            // Delta depth to increase each time
    float F;            // Cutting feedrate
    float L;            // Number of repetitions

}Working_Values;


Working_Config Command_Value;
Working_Values Values;
```

CONFIGURACIONES

- ## COMUNICACIÓN USART

```
UartHandle.Instance = USART2;
UartHandle.Init.BaudRate = 19200;
UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;


void HAL_UART_MspInit( UART_HandleTypeDef *huart )
{
    GPIO_InitTypeDef GPIO_InitStruct;
    __GPIOA_CLK_ENABLE();
    __USART2_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_2 | GPIO_PIN_3;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF4_USART2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    HAL_NVIC_SetPriority(USART2_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(USART2_IRQn);
}
```

- **SPI**

```
void HAL_SPI_MspInit(SPI_HandleTypeDef *hspi)
{
  GPIO_InitTypeDef  GPIO_InitStruct;

  if(hspi->Instance == SPIx)
  {
    SPIx_SCK_GPIO_CLK_ENABLE();
    SPIx_MISO_GPIO_CLK_ENABLE();
    SPIx_MOSI_GPIO_CLK_ENABLE();
    SPIx_CLK_ENABLE();
    GPIO_InitStruct.Pin       = SPIx_SCK_PIN;
    GPIO_InitStruct.Mode      = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull      = GPIO_NOPULL;
    GPIO_InitStruct.Speed     = GPIO_SPEED_MEDIUM;
    GPIO_InitStruct.Alternate = SPIx_SCK_AF;

    HAL_GPIO_Init(SPIx_SCK_GPIO_PORT, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = SPIx_MISO_PIN;
    GPIO_InitStruct.Alternate = SPIx_MISO_AF;

    HAL_GPIO_Init(SPIx_MISO_GPIO_PORT, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = SPIx_MOSI_PIN;
    GPIO_InitStruct.Alternate = SPIx_MOSI_AF;

    HAL_GPIO_Init(SPIx_MOSI_GPIO_PORT, &GPIO_InitStruct);
  }
                                      }
```

```
void HAL_SPI_MspDeInit(SPI_HandleTypeDef *hspi)
{
  if(hspi->Instance == SPIx)
  {
    SPIx_FORCE_RESET();
    SPIx_RELEASE_RESET();

    HAL_GPIO_DeInit(SPIx_SCK_GPIO_PORT, SPIx_SCK_PIN);
    HAL_GPIO_DeInit(SPIx_MISO_GPIO_PORT, SPIx_MISO_PIN);
    HAL_GPIO_DeInit(SPIx_MOSI_GPIO_PORT, SPIx_MOSI_PIN);
  }
                                      }
```

- **PROGRAMA PRINCIPAL**

```
int main(){
    /*------------- Default Values -----------------*/
    Values.L = 1;
    Command_Value.Units = MM;
    Command_Value.Distance_Mode = ABSOLUTE_MODE;
    Command_Value.Program_State = RUN;
    /*----------------------------------------------*/
    int32_t pos;
    uint16_t mySpeed;

      HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();
        //Endstops_GpioInit();
    /* STM32xx HAL library initialization */

    UartHandle.Instance = USART2;
    UartHandle.Init.BaudRate     = 19200;
    UartHandle.Init.WordLength   = UART_WORDLENGTH_8B;
    UartHandle.Init.StopBits     = UART_STOPBITS_1;
    UartHandle.Init.Parity       = UART_PARITY_NONE;
    UartHandle.Init.HwFlowCtl    = UART_HWCONTROL_NONE;
    UartHandle.Init.Mode         = UART_MODE_TX_RX;
    UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;

    HAL_UART_Init(&UartHandle);

    //----- Init of the Motor control library
  /* Start the L6474 library to use 3 devices */
  /* The L6474 registers are set with the predefined values */
  /* from file l6474_target_config.h*/
  BSP_MotorControl_Init(BSP_MOTOR_CONTROL_BOARD_ID_L6474, 3);

  /* Attach the function MyFlagInterruptHandler (defined below) to the flag
interrupt */
  BSP_MotorControl_AttachFlagInterrupt(MyFlagInterruptHandler);

  /* Attach the function Error_Handler (defined below) to the error
Handler*/
  BSP_MotorControl_AttachErrorHandler(Error_Handler);
  BSP_MotorControl_SelectStepMode(0, STEP_MODE_1_16);
  BSP_MotorControl_SelectStepMode(1, STEP_MODE_1_16);
  BSP_MotorControl_SelectStepMode(2, STEP_MODE_1_16);

  /* Move all the axis backwards to hit the min. endstop (home position) */
        BSP_MotorControl_Run(X_AXIS, BACKWARD);
        while((HAL_GPIO_ReadPin(X_MIN_ENDSTOP_PORT, X_MIN_ENDSTOP_PIN)) !=
GPIO_PIN_RESET){
        }
        BSP_MotorControl_HardStop(X_AXIS);
```

```
        BSP_MotorControl_Run(Y_AXIS, BACKWARD);
        while((HAL_GPIO_ReadPin(Y_MIN_ENDSTOP_PORT, Y_MIN_ENDSTOP_PIN)) !=
GPIO_PIN_RESET){
        }
        BSP_MotorControl_HardStop(Y_AXIS);
        BSP_MotorControl_Run(Z_AXIS, BACKWARD);
        while((HAL_GPIO_ReadPin(Z_MIN_ENDSTOP_PORT, Z_MIN_ENDSTOP_PIN)) !=
GPIO_PIN_RESET){
        }
        BSP_MotorControl_HardStop(Z_AXIS);

        for(i=0; i <= NUM_AXIS - 1; i++){
        BSP_MotorControl_SetHome(i);        // When endstop hit, set this
position as home position
      }

    HAL_UART_Transmit_IT(&UartHandle, Initial_TxBuffer,
sizeof(Initial_TxBuffer));  // Send message asking for Commands
    HAL_Delay (500);        // Half second delay to be sure everything sent

    while(1){

        HAL_UART_Receive_IT(&UartHandle, RxBuffer, RX_BUFFER_SIZE);     //
Receive Gcode commands

        //if (HAL_UART_Receive_IT(&UartHandle, RxBuffer, RX_BUFFER_SIZE) ==
HAL_OK){    // If a complete buffer has been received
        if (Complete_line == 1){        // If a complete order line has
been received

            Read_from_buffer();  // Store received data for reading each
specific value
            Apply_commands();    // Apply the specifications and orders
received

            HAL_UART_Transmit_IT(&UartHandle, TxBuffer_OK,
sizeof(TxBuffer_OK));  // Send message to verify the string has been
received correctly
            Complete_line = 0;      // Reset complete line flag
        }

    }
}
```

- **FUNCIONES PRINCIPALES**

```
void Read_command_Value(char Command){

            int i;
            uint8_t number;
      for(i = 0; i < RX_BUFFER_SIZE; i++){
      if(RxBuffer[i] == Command){
            i++;
            init = i;
            j = 0;


            while((RxBuffer[i] != ' ') && (RxBuffer[i] != '\n')
            && (RxBuffer[i] != '\r')){
                  end = i;
                  i++;
            }
                  for(number = init; number < end+1; number++){
                  Conversion[j] = RxBuffer[number];
                        j++;
                  }
                  Value = atof(Conversion);
                  switch (Command){
                        case 'G':
                              switch (Value){
                                    case 20:
                                          Command_Value.Units
                                                = INCHES;
                                    break;
                                    case 21:
                                          Command_Value.Units
                                                = MM;
                                    break;
                                    case 90:
                                          Command_Value.Distance_Mode
                                                = ABSOLUTE_MODE;
                                    break;
                                    case 91:
                                          Command_Value.Distance_Mode
                                                = INCREMENTAL_MODE;
                                    break;
                                    case 80:
                                          Command_Value.Program_State
                                                = NONE_OR_STOP_CYCLE;
                                    break;
                                    case 81:
                                          Command_Value.Operation_Cycle
                                                = DRILLING;
                                    break;
                                    case 82:
                                          Command_Value.Operation_Cycle
                                                = DRILLING_DWELL;
                                    break;
                                    case 83:
                                          Command_Value.Operation_Cycle
```

```
                                        = DRILLING_PECK;
                        break;
                        default:
                        break;
                }
                break;
        case 'M':
                switch (Value){
                        case 2:
                                Command_Value.Program_State
                                        = STOP;
                        break;
                        default:
                        break;
                }
        case 'X':
                Values.X_position
                        = Value;
        break;
        case 'Y':
                Values.Y_position
                        = Value;
        break;
        case 'Z':
                Values.Z_position
                        = Value;
        break;
        case 'R':
                Values.R
                        = Value;
        break;
        case 'P':
                Values.P
                        = Value;
        break;
        case 'Q':
                Values.Q
                        = Value;
        break;

        case 'F':
                Values.F
                        = Value;
        break;

        case 'L':
                Values.L
                        = Value;
        break;
        default:
        break;
}
        }

        }
}
```

```
void Read_from_Buffer (void){

      Read_command_Value('G');
      Read_command_Value('M');

      Read_command_Value('X');
      Read_command_Value('Y');
      Read_command_Value('Z');

      Read_command_Value('R');
      Read_command_Value('P');
      Read_command_Value('F');
      Read_command_Value('Q');
      Read_command_Value('L');
}
```

```
void Get_movements(void){    // Function to calculate the necessary
movements and the order
    uint8_t i = 0;
    uint8_t j = 0;
    uint8_t k = 0;
    float Actual_hole_depth = 0.0;
    float Start_position = 0.0;
    switch(Command_Value.Operation_Cycle){
        case FAST_MOVEMENT:
                Movement[0] = Values.Z_position - (Actual_position[Z_AXIS]
                              * (1 - Command_Value.Distance_Mode));
                Actual_position[Z_AXIS] = Values.Z_position +
                                          (Actual_position[Z_AXIS] *
                                          Command_Value.Distance_Mode);
                Movement[1] = Values.X_position - (Actual_position[X_AXIS]
                              * (1 - Command_Value.Distance_Mode));
                Actual_position[X_AXIS] = Values.X_position +
                                          (Actual_position[X_AXIS] *
                                          Command_Value.Distance_Mode);
                Movement[2] = Values.Y_position - (Actual_position[Y_AXIS]
                              * (1 - Command_Value.Distance_Mode));
                Actual_position[Y_AXIS] = Values.Y_position +
                                          (Actual_position[Y_AXIS] *
                                          Command_Value.Distance_Mode);
        break;
        case DRILLING:
            Movement[i] = Values.R - (Actual_position[Z_AXIS] * (1 -
                          Command_Value.Distance_Mode));
            Actual_position[Z_AXIS] = Values.R + (Actual_position[Z_AXIS] *
                                      Command_Value.Distance_Mode);
            i++;
            for(j = 0; j <= (Values.L - 1); j++){
                Movement[i] = Values.X_position - (Actual_position[X_AXIS]
                              * (1 - Command_Value.Distance_Mode));
                Actual_position[X_AXIS] = Values.X_position +
                                          (Actual_position[X_AXIS] *
                                          Command_Value.Distance_Mode);
                i++;
                Movement[i] = Values.Y_position - (Actual_position[Y_AXIS]
                              * (1 - Command_Value.Distance_Mode));
```

```
            Actual_position[Y_AXIS] = Values.Y_position +
                                      (Actual_position[Y_AXIS] *
                                      Command_Value.Distance_Mode);
            i++;
            Movement[i] = Values.Z_position - (Actual_position[Z_AXIS]
                          * (1 - Command_Value.Distance_Mode));
            Actual_position[Z_AXIS] = Values.Z_position +
                                      (Actual_position[Z_AXIS] *
                                      Command_Value.Distance_Mode);
            i++;
            Movement[i] = - Movement[i-1];
            Actual_position[Z_AXIS] = Values.R +
                                      (Actual_position[Z_AXIS] *
                                      Command_Value.Distance_Mode);
            i++;
        }
        Total_movements = i;
    break;

    case DRILLING_DWELL:
        Movement[i] = Values.R - (Actual_position[Z_AXIS] * (1 -
                      Command_Value.Distance_Mode));
        Actual_position[Z_AXIS] = Values.R + (Actual_position[Z_AXIS] *
                                  Command_Value.Distance_Mode);
        i++;
        for(j = 0; j <= (Values.L - 1); j++){
            Movement[i] = Values.X_position - (Actual_position[X_AXIS]
                          * (1 - Command_Value.Distance_Mode));
            Actual_position[X_AXIS] = Values.X_position +
                                      (Actual_position[X_AXIS] *
                                      Command_Value.Distance_Mode);
            i++;
            Movement[i] = Values.Y_position - (Actual_position[Y_AXIS]
                          * (1 - Command_Value.Distance_Mode));
            Actual_position[Y_AXIS] = Values.Y_position +
                                      (Actual_position[Y_AXIS] *
                                      Command_Value.Distance_Mode);
            i++;
            Movement[i] = Values.Z_position - (Actual_position[Z_AXIS]
                          * (1 - Command_Value.Distance_Mode));
            Actual_position[Z_AXIS] = Values.Z_position +
                                      (Actual_position[Z_AXIS] *
                                      Command_Value.Distance_Mode);
            i++;
            Dwell_time = (Values.P) * 1000;
            Movement[i] = - Movement[i-1];
            Actual_position[Z_AXIS] = Values.R +
                                      (Actual_position[Z_AXIS] *
                                      Command_Value.Distance_Mode);
            i++;
        }
        Total_movements = i;
    break;

    case DRILLING_PECK:
        Movement[i] = Values.R - (Actual_position[Z_AXIS] * (1 -
                      Command_Value.Distance_Mode));
        Actual_position[Z_AXIS] = Values.R + (Actual_position[Z_AXIS] *
                                  Command_Value.Distance_Mode);
        Start_position = Actual_position[Z_AXIS];
        i++;
```

```
            for(j = 0; j <= (Values.L - 1); j++){
                Movement[i] = Values.X_position - (Actual_position[X_AXIS]
                              * (1 - Command_Value.Distance_Mode));
                Actual_position[X_AXIS] = Values.X_position +
                                          (Actual_position[X_AXIS] *
                                          Command_Value.Distance_Mode);
                i++;
                Movement[i] = Values.Y_position - (Actual_position[Y_AXIS]
                              * (1 - Command_Value.Distance_Mode));
                Actual_position[Y_AXIS] = Values.Y_position +
                                          (Actual_position[Y_AXIS] *
                                          Command_Value.Distance_Mode);
                i++;
                Actual_hole_depth = 0;
                for(k = 0; k <= (((Start_position - (Start_position +
                   Values.Z_position))/Values.Q) - 1); k++){
                    Movement[i] = - Actual_hole_depth;
                    Actual_position[Z_AXIS] = Actual_position[Z_AXIS] -
                                              Actual_hole_depth;
                    i++;
                    Movement[i] = - Values.Q;
                     Actual_position[Z_AXIS] = Actual_position[Z_AXIS] -
                                               Values.Q;
                    Actual_hole_depth = Start_position -
                                        Actual_position[Z_AXIS];
                    i++;
                    Movement[i] = Actual_hole_depth;
                    Actual_position[Z_AXIS] = Start_position;
                    i++;
                }
            }
            Total_movements = i;
        break;
    }
}




void Apply_commands(){

      int i;

    switch (Command_Value.Program_State){
            case STOP:
                for (i = 0; i < NUM_AXIS; i++){
                   BSP_MotorControl_GoHome(i);
                   BSP_MotorControl_WaitWhileActive(i);
                }
              BSP_MotorControl_ResetAllDevices();
            break;
            case RUN:
              BSP_MotorControl_ReleaseReset();
            break;
            default:
            break;
    }
```

```
      switch(Command_Value.Units){
            case MM:
                  for(i = 0; i < NUM_AXIS; i++){
                        Conversion_selected[i] = axis_steps_per_mm[i];
                  }
            break;
            case INCHES:
                  for(i = 0; i < NUM_AXIS; i++){
                     Conversion_selected[i] = axis_steps_per_inch[i];
                  }
            break;
      }


      Get_movements();    // Apply all the configurations and operation
cycle to get the list of movements




      switch(Command_Value.Operation_Cycle){
            case FAST_MOVEMENT:
                 Fast_coordinates_movement();
            break;

            case DRILLING:
                  Simple_drilling_operation();
            break;
            case DRILLING_DWELL:
                  Driling_with_dwell_operation();
            break;
            case DRILLING_PECK:
                  Drilling_with_peck_operation();
            break;
      }

}




void Fast_coordinates_movement(void){

    for(i = 0; i <= NUM_AXIS + 1; i++){
        if (Movement[i] <= 0){
            Direction[i] = BACKWARD;
            Movement[i] = - Movement[i];
        }
        else{
            Direction[i] = FORWARD;
        }
    }
    BSP_MotorControl_SetMaxSpeed(Z_AXIS, FAST_SPEED);
```

```
    BSP_MotorControl_Move(Z_AXIS, Direction[0], Movement[0] *
                          Conversion_selected[Z_AXIS]);
    BSP_MotorControl_WaitWhileActive(Z_AXIS);
    BSP_MotorControl_Move(X_AXIS, Direction[1], Movement[1] *
                          Conversion_selected[X_AXIS]);
    BSP_MotorControl_Move(Y_AXIS, Direction[2], Movement[2] *
                          Conversion_selected[Y_AXIS]);
    BSP_MotorControl_WaitWhileActive(X_AXIS);
    BSP_MotorControl_WaitWhileActive(Y_AXIS);
}
```

```
void Simple_drilling_operation(void){   // Function which specify the
movements for a simple drilling operation
    uint8_t j = 0;
    int i;
    uint8_t cycle = 0;

    for(i = 0; i <= (Total_movements+1); i++){
        if (Movement[i] <= 0){
            Direction[i] = BACKWARD;
            Movement[i] = - Movement[i];
        }
        else{
            Direction[i] = FORWARD;
        }
    }
    BSP_MotorControl_SetMaxSpeed(Z_AXIS, FAST_SPEED);
    BSP_MotorControl_Move(Z_AXIS, Direction[0], Movement[0] *
                          Conversion_selected[Z_AXIS]);
    BSP_MotorControl_WaitWhileActive(Z_AXIS);
    for(i = 0; i <= (Values.L - 1); i++){
        BSP_MotorControl_Move(X_AXIS, Direction[(i*4)+1], Movement[(i*4)+1]
                    * Conversion_selected[X_AXIS]);
        BSP_MotorControl_Move(Y_AXIS, Direction[(i*4)+2], Movement[(i*4)+2]
                    * Conversion_selected[Y_AXIS]);
        BSP_MotorControl_WaitWhileActive(X_AXIS);
        BSP_MotorControl_WaitWhileActive(Y_AXIS);
        cycle = 0;
        for(j = (3+(4*i)); j <= (4+(4*i)); j++){
            if (cycle == 0){
                BSP_MotorControl_SetMaxSpeed(Z_AXIS, DRILLING_SPEED);
            }
            else{
                BSP_MotorControl_SetMaxSpeed(Z_AXIS, FAST_SPEED);
            }
            cycle = 1;
            BSP_MotorControl_Move(Z_AXIS, Direction[j], Movement[j] *
                                  Conversion_selected[Z_AXIS]);
            BSP_MotorControl_WaitWhileActive(Z_AXIS);
        }
    }

}
```

```
void Drilling_with_dwell_operation(void){   // Function which specify the
movements for a drilling with dwell operation
    uint8_t j = 0;
    int i;
    uint8_t cycle = 0;

    for(i = 0; i <= (Total_movements+1); i++){
        if (Movement[i] <= 0){
            Direction[i] = BACKWARD;
            Movement[i] = - Movement[i];
        }



        else{
            Direction[i] = FORWARD;
        }
    }
    BSP_MotorControl_Move(Z_AXIS, Direction[0], Movement[0] *
                          Conversion_selected[Z_AXIS]);
    BSP_MotorControl_WaitWhileActive(Z_AXIS);
    for(i = 0; i <= (Values.L - 1); i++){
        BSP_MotorControl_Move(X_AXIS, Direction[(i*4)+1], Movement[(i*4)+1]
                    * Conversion_selected[X_AXIS]);
        BSP_MotorControl_Move(Y_AXIS, Direction[(i*4)+2], Movement[(i*4)+2]
                    * Conversion_selected[Y_AXIS]);
        BSP_MotorControl_WaitWhileActive(X_AXIS);
        BSP_MotorControl_WaitWhileActive(Y_AXIS);
        cycle = 0;
        for(j = (3+(4*i)); j <= (4+(4*i)); j++){
            if (cycle == 0){
                BSP_MotorControl_SetMaxSpeed(Z_AXIS, DRILLING_SPEED);
            }
            else{
                BSP_MotorControl_SetMaxSpeed(Z_AXIS, FAST_SPEED);
            }
            cycle = 1;
            BSP_MotorControl_Move(Z_AXIS, Direction[j], Movement[j] *
                                  Conversion_selected[Z_AXIS]);
            BSP_MotorControl_WaitWhileActive(Z_AXIS);
            if(j == (3+(4*i))){
                HAL_Delay (Dwell_time);
            }
        }
    }
}
```

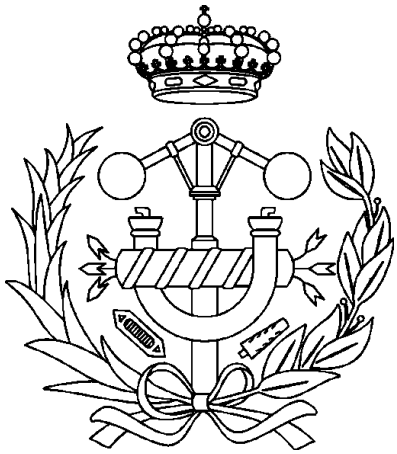
```
void Drilling_with_peck_operation(void){    // Function which specify the
movements for a peck drilling operation
    uint8_t j = 0;
    int num = 0;
    int i;
    uint8_t cycle = 0;

    int Movements_per_cycle = (Total_movements/Values.L);

    for(i = 0; i <= (Total_movements+1); i++){
        if (Movement[i] <= 0){
            Direction[i] = BACKWARD;
            Movement[i] = - Movement[i];
        }
        else{
            Direction[i] = FORWARD;
        }
    }
    BSP_MotorControl_SetMaxSpeed(Z_AXIS, FAST_SPEED);
    BSP_MotorControl_Move(Z_AXIS, Direction[num], Movement[num] *
                          Conversion_selected[Z_AXIS]);
    num++;
    BSP_MotorControl_WaitWhileActive(Z_AXIS);



    for(i = 0; i <= (Values.L - 1); i++){
        BSP_MotorControl_Move(X_AXIS, Direction[num], Movement[num] *
                              Conversion_selected[X_AXIS]);
        num++;
        BSP_MotorControl_Move(Y_AXIS, Direction[num], Movement[num] *
                              Conversion_selected[Y_AXIS]);
        num++;
        BSP_MotorControl_WaitWhileActive(X_AXIS);
        BSP_MotorControl_WaitWhileActive(Y_AXIS);
        cycle = 0;
        for(j = 0; j <= ((((Total_movements-1)/Values.L)-2)-1); j++){
            if (cycle == 0){
                BSP_MotorControl_SetMaxSpeed(Z_AXIS, DRILLING_SPEED);
            }
            else{
                BSP_MotorControl_SetMaxSpeed(Z_AXIS, FAST_SPEED);
            }
            cycle++;
            if (cycle == 3){
                cycle = 0;
            }
            BSP_MotorControl_Move(Z_AXIS, Direction[num], Movement[num] *
                        Conversion_selected[Z_AXIS]);
            num++;
            BSP_MotorControl_WaitWhileActive(Z_AXIS);
        }
    }
}
```



Anexo II:

Documentación Técnica

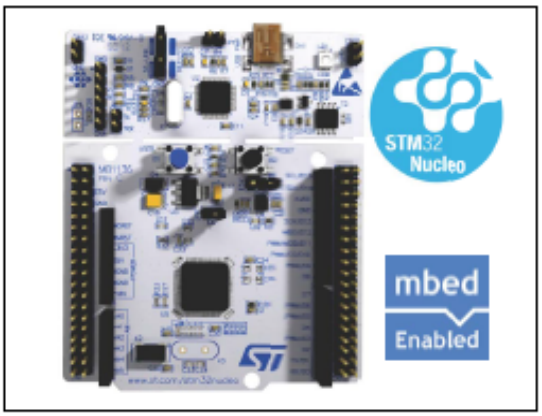
NUCLEO-XXXXRX

STM32 Nucleo-64 boards

Data brief

Features

- STM32 microcontroller with LQFP64 package
- Two types of extension resources
  - Arduino Uno Revision 3 connectivity
  - STMicroelectronics Morpho extension pin headers for full access to all STM32 I/Os
- mbed-enabled (http://mbed.org)
- On-board ST-LINK/V2-1 debugger/programmer with SWD connector
  - selection-mode switch to use the kit as a standalone ST-LINK/V2-1
- Flexible board power supply
  - USB VBUS or external source (3.3 V, 5 V, 7 - 12 V)
  - Power management access point
- Three LEDs
  - USB communication (LD1), user LED (LD2), power LED (LD3)
- Two push buttons: USER and RESET
- USB re-enumeration capability: three different interfaces supported on USB
  - Virtual Com port
  - Mass storage
  - Debug port
- Supported by wide choice of Integrated Development Environments (IDEs) including IAR™, Keil®, GCC-based IDEs



1. Picture not contractual

Description

The STM32 Nucleo board provides an affordable and flexible way for users to try out new ideas and build prototypes with any STM32 microcontroller line, choosing from the various combinations of performance, power consumption and features. The Arduino™ connectivity support and ST Morpho headers make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields. The STM32 Nucleo board does not require any separate probe as it integrates the ST-LINK/V2-1 debugger and programmer. The STM32 Nucleo board comes with the STM32 comprehensive software HAL library together with various packaged software examples, as well as direct access to mbed online resources.

Table 1. Device summary

Reference	Part number
NUCLEO-XXXXRX	NUCLEO-F030R8, NUCLEO-F070RB, NUCLEO-F072RB, NUCLEO-F091RC, NUCLEO-F103RB, NUCLEO-F302R8, NUCLEO-F303RE, NUCLEO-F334R8, NUCLEO-F401RE, NUCLEO-F410RB, NUCLEO-F411RE, NUCLEO-F446RE, NUCLEO-L053R8, NUCLEO-L073RZ, NUCLEO-L152RE, NUCLEO-L476RG

life.augmented

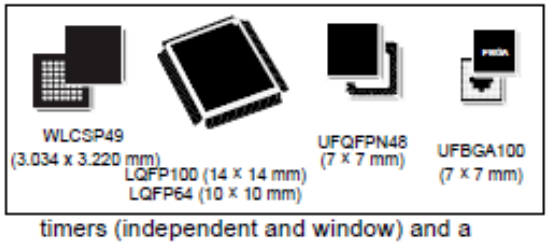
STM32F411xC STM32F411xE

ARM® Cortex®-M4 32b MCU+FPU, 125 DMIPS, 512KB Flash, 128KB RAM, USB OTG FS, 11 TIMs, 1 ADC, 13 comm. interfaces

Datasheet - production data

Features

- Dynamic Efficiency Line with BAM (Batch Acquisition Mode)
- Core: ARM® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 100 MHz, memory protection unit, 125 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
  - up to 512 Kbytes of Flash memory
  - 128 Kbytes of SRAM
- Clock, reset and supply management
  - 1.7 V to 3.6 V application supply and I/Os
  - POR, PDR, PVD and BOR
  - 4-to-26 MHz crystal oscillator
  - Internal 16 MHz factory-trimmed RC
  - 32 kHz oscillator for RTC with calibration
  - Internal 32 kHz RC with calibration
- Power consumption
  - Run: 100 µA/MHz (peripheral off)
  - Stop (Flash in Stop mode, fast wakeup time): 42 µA Typ @ 25C; 65 µA max @25 °C
  - Stop (Flash in Deep power down mode, fast wakeup time): down to 10 µA @ 25 °C; 30 µA max @25 °C
  - Standby: 2.4 µA @25 °C / 1.7 V without RTC; 12 µA @85 °C @1.7 V
  - V_{BAT} supply for RTC: 1 µA @25 °C
- 1×12-bit, 2.4 MSPS A/D converter: up to 16 channels
- General-purpose DMA: 16-stream DMA controllers with FIFOs and burst support
- Up to 11 timers: up to six 16-bit, two 32-bit timers up to 100 MHz, each with up to four IC/OC/PWM or pulse counter and quadrature (incremental) encoder input, two watchdog



timers (independent and window) and a SysTick timer

- Debug mode
  - Serial wire debug (SWD) & JTAG interfaces
  - Cortex®-M4 Embedded Trace Macrocell™
- Up to 81 I/O ports with interrupt capability
  - Up to 78 fast I/Os up to 100 MHz
  - Up to 77 5 V-tolerant I/Os
- Up to 13 communication interfaces
  - Up to 3 x I^2C interfaces (SMBus/PMBus)
  - Up to 3 USARTs (2 x 12.5 Mbit/s, 1 x 6.25 Mbit/s), ISO 7816 interface, LIN, IrDA, modem control)
  - Up to 5 SPI/I2Ss (up to 50 Mbit/s, SPI or I2S audio protocol), SPI2 and SPI3 with muxed full-duplex I^2S to achieve audio class accuracy via internal audio PLL or external clock
  - SDIO interface (SD/MMC/eMMC)
  - Advanced connectivity: USB 2.0 full-speed device/host/OTG controller with on-chip PHY
- CRC calculation unit
- 96-bit unique ID
- RTC: subsecond accuracy, hardware calendar
- All packages (WLCSP49, LQFP64/100, UFQFPN48, UFBGA100) are ECOPACK®2

Table 1. Device summary

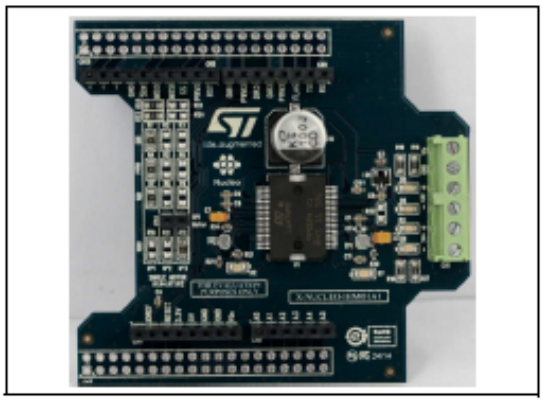
Reference	Part number
STM32F411xC	STM32F411CC, STM32F411RC, STM32F411VC
STM32F411xE	STM32F411CE, STM32F411RE, STM32F411VE

life.augmented

X-NUCLEO-IHM01A1

Stepper motor driver expansion board based on L6474 for STM32 Nucleo

Data brief



Features

- Voltage range from 8 V to 45 V
- Phase current up to 3 $A_{r.m.s.}$
- Power OK and fault LEDs
- Advanced current control
- Fully protected power stage
- Up to 1/16 microstepping resolution
- Compatible with Arduino UNO R3 connector
- Compatible with STM32 Nucleo boards
- Suitable for multi-motor solutions
- RoHS compliant

Description

The X-NUCLEO-IHM01A1 is a stepper motor driver expansion board based on the L6474.

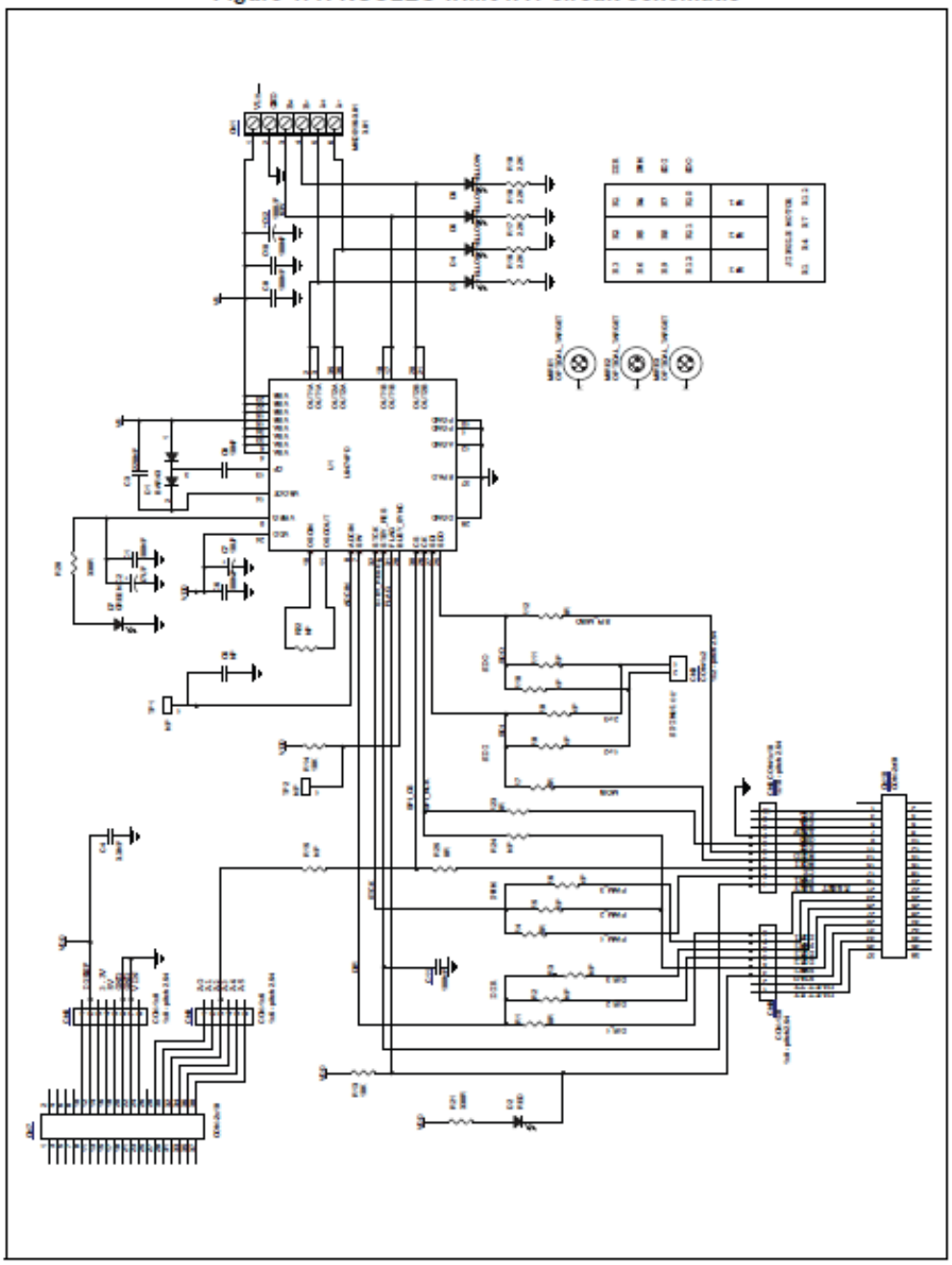
It provides an affordable and easy-to-use solution for driving a stepper motor in your STM32 Nucleo project.

The advanced current control of the L6474 and a complete set of protection features offer high levels of both performance and robustness.

The X-NUCLEO-IHM01A1 is compatible with the Arduino UNO R3 connector, and supports the addition of other boards which can be stacked to drive up to three stepper motors with a single STM32 Nucleo board.

1 Schematic diagram

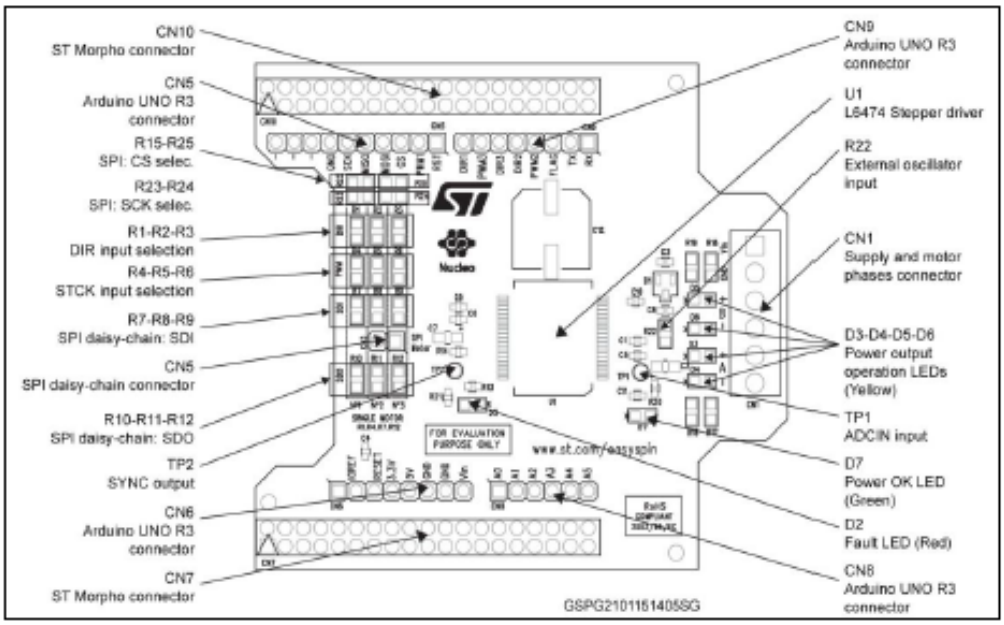
Figure 1. X-NUCLEO-IHM01A1 circuit schematic



2 Hardware description and configuration

The *Figure 2: "Jumpers and connectors position"* shows the position of the connectors and the configuration jumpers of the board.

Figure 2: Jumpers and connectors position



The following tables provide the connector details for the Arduino UNO R3 and ST Morpho, respectively.

Table 1: Arduino UNO R3 connector table

Connector	Pin(1)	Signal	Remarks
CN5	1	L6474 STBY\\RESET	
	2	PWM1	See *Section 2.1: "Selecting the chip select and clock lines of the SPI"*
	3	SPI CS	See *Section 2.2: "Multi motor configuration"*
	4	SPI MOSI	See *Section 2.1: "Selecting the chip select and clock lines of the SPI"*
	5	SPI MISO	
	6	SPI SCK	See *Section 2.2: "Multi motor configuration"*
	7	Ground	
CN9	3	L6474 FLAG output	
	4	PWM2\\SPI SCK	See *Section 2.2: "Multi motor configuration"* and *Section 2.1: "Selecting the chip select and clock lines of the SPI"*
	5	DIR2	See *Section 2.2: "Multi motor configuration"*
	6	DIR3	
	7	PWM3	

Connector	Pin(1)	Signal	Remarks
	8	DIR1	
CN6	2	VDD	
	6	Ground	
	7	Ground	
CN8	3	SPI CS	See *Section 2.1: "Selecting the chip select and clock lines of the SPI"*

Notes:

(1)All the non-listed pins are not connected.

Table 2: ST Morpho connector table

Connector	Pin(1)	Signal	Remarks
CN10	9	Ground	
	11	SPI SCK	See *Section 2.1: "Selecting the chip select and clock lines of the SPI"*
	13	SPI MISO	See *Section 2.2: "Multi motor configuration"*
	15	SPI MOSI	
	17	SPI CS	See *Section 2.1: "Selecting the chip select and clock lines of the SPI"*
	19	PWM1	See *Section 2.2: "Multi motor configuration"*
	21	L6474 STBY\\RESET	
	23	DIR1	See *Section 2.2: "Multi motor configuration"*
	25	PWM3	
	27	DIR3	
	29	DIR2	
	31	PWM2	See *Section 2.1: "Selecting the chip select and clock lines of the SPI"* and *Section 2.2: "Multi motor configuration"*
	33	L6474 FLAG output	
CN7	12	VDD	
	20	Ground	
	22	Ground	
	32	SPI CS	See *Section 2.1: "Selecting the chip select and clock lines of the SPI"*

Notes:

(1)All the non-listed pins are not connected.

2.1 Selecting the chip select and clock lines of the SPI

The chip select and the clock lines of the SPI interface can be selected through dedicated resistors as indicated in *Table 3: "Chip select line selection"* and *Table 4: "Chip select line selection"*.

Table 3: Chip select line selection

R15	R25	CS line
Not mounted	0R	CN5 pin 3, CN10 pin 17 (default)
0R	Not mounted	CN8 pin 3, CN7 pin 32

Table 4: Chip select line selection

R23	R24	CS line
0R	Not mounted	CN5 pin 6, CN10 pin 9 (default)
Not mounted	0R	CN9 pin 4, CN10 pin 31

When the alternative clock line is selected (CN9 pin 4, CN10 pin 31) the PWM2 signal is no longer available for multi-motor configurations (see *Section 2.2: "Multi motor configuration"*).

2.2 Multi motor configuration

The expansion boards can be stacked on a single STM32 Nucleo board in order to drive up to three stepper motors (one for each motor).

The configuration can be changed by mounting the necessary resistors from R1 to R12 as listed in the *Table 5*. The other resistors are not mounted.

By default, the stepper driver board is configured for a single motor setup, so board configurations for multi-motor setups must be changed before stacking the boards on the STM32 Nucleo.

Table 5: Multi-motor setup table

Number of motors	Board	STCK\\DIR	Mounted resistors (0R)
1	-	PWM1\\DIR1	R1, R4, R7, R12
2	1 (bottom)	PWM1\\DIR1	R1, R4, R7, R10
	2 (top)	PWM2\\DIR2	R2, R5, R8, R12
3	1 (bottom)	PWM1\\DIR1	R1, R4, R7, R10
	2	PWM2\\DIR2	R2, R5, R8, R11
	3 (top)	PWM3\\DIR3	R3, R6, R9, R12

If the alternative SPI clock line is selected (see *Section 2.1: "Selecting the chip select and clock lines of the SPI"*) the PWM2 step clock is no longer available and the multi-motor setup is limited to two motors maximum.

The *Table 6: "Multi-motor setup with alternative SPI clock line"* shows the proper configuration in this case.

PITTMAN® MAE®
PennEngineering®
Motion Technologies

BULLETIN STP

MAE® Stepper Motors

Stepper Motors

PennEngineering Motion Technologies offers a wide range of MAE brand stepper motor solutions. The HY series hybrid stepper motors feature low rotor inertia for maximum possible acceleration. The HN series hybrid stepper motors offer a calculated balance between low rotor inertia and high torque. The HS series hybrid stepper motors are optimized for superior torque characteristics. Additionally, both the HN and HS series feature low detent torque to holding torque ratios to provide smooth operation as well as the fine positioning capability required for microstep operation.

Motors may be customized with value added features including, but not limited to: gearboxes, encoders, shaft details, leadwire-connector assemblies, and more.

All specifications shown are typical at 20° C unless otherwise noted.

Shaft extensions
All motors can be supplied with single or double ended shaft.

Rotation
The motor rotation can run clockwise or counterclockwise, depending on the commutation.

Operating temperature
Ambient operating temperature: -20°C to +40°C.

Number of leads
Refer to specifications of individual models for standard lead wire configuration. Motors can be supplied with 4, 6, or 8 leads upon request; however, rated current and torque may be reduced.

Angular accuracy
Standard angular accuracy is ±5%. Angular accuracy is defined as the deviation from a theoretical position, in percentage of one step, after any number of steps.

Holding torque
The typical values of holding torque of the different models are indicated in the data charts. Holding torque is measured with two phases each supplied at the rated current.

Specifications and approvals
Motors are manufactured according to EN 60034-1: 1995-02. Motors with drive voltage higher than or equal to 120 V are suitable to be fitted on machines equipped with additional insulation or when the motor itself has the grounding through its clamping screws.

Due to thermal considerations, stepper motors cannot always be operated continuously in dynamic conditions at the level of their static rated phase current.

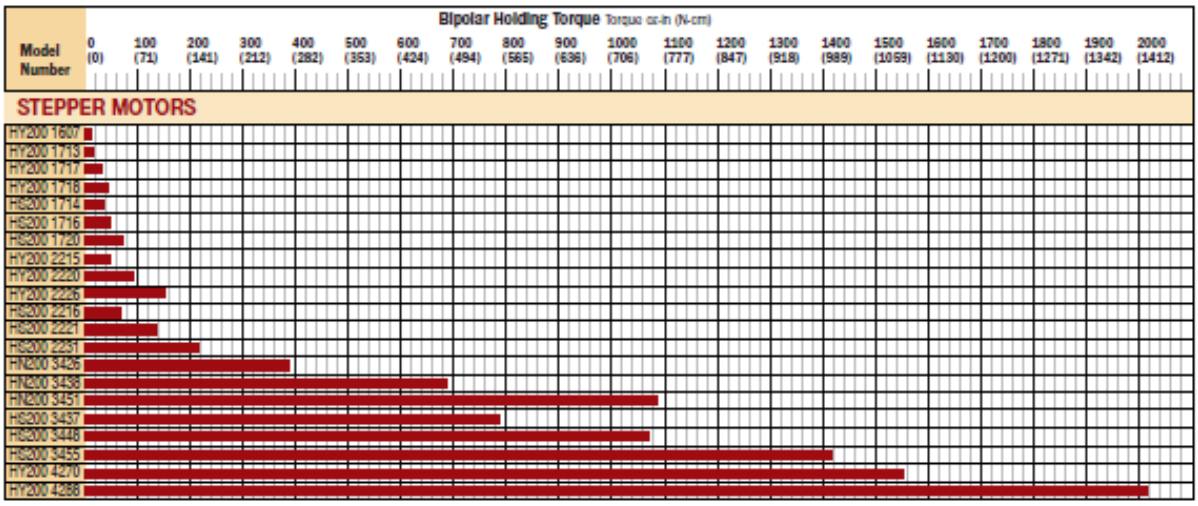
Stepper Motors

- **Accurate open loop control for high performance positioning applications**
- **Excellent low speed torque**
- **Simple, rugged construction for high reliability and long service life**
- **Smooth, quiet operation**
- **Standard NEMA frame sizes**
- **Precision honed stators and ground rotors for tight air gap and maximum performance**
- **CE approved**

MOTIONExpress™
Get same day shipment of sample motors for models listed in this bulletin.

PennEngineering Motion Technologies offers a complete line of PITTMAN® and MAE® brand brush, brushless, and stepper motors which can be customized to meet your exact requirements.

MOTOR SELECTION GUIDE



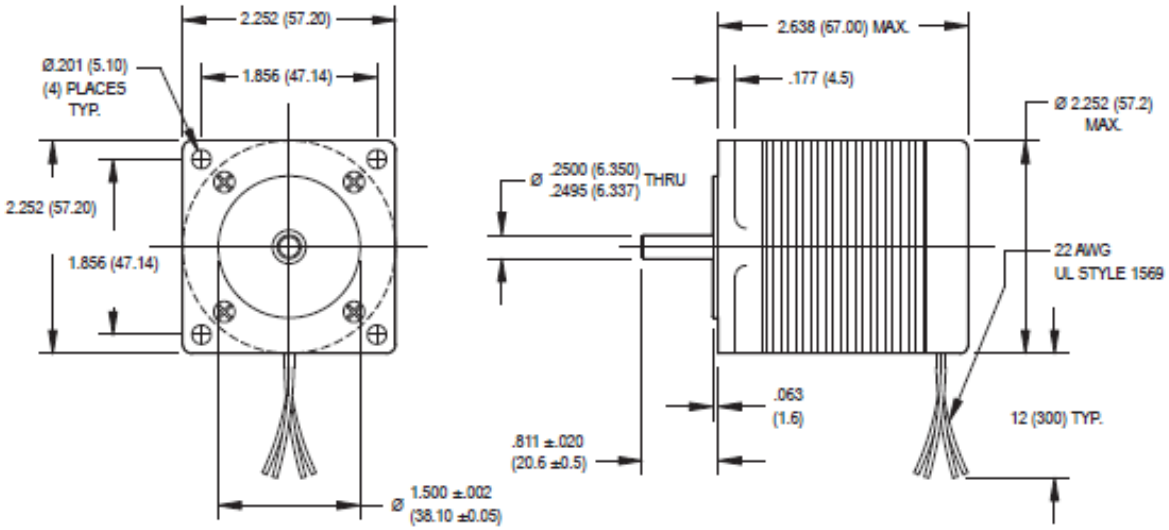
CONNECTION-DEPENDENT RATINGS FOR 8 LEAD MOTORS

Stepper motors supplied with 8 leads provide maximum flexibility and allow the user to decide what connection method is most suitable for their application. Some of the motor phase characteristics are dependent on the connection method chosen for the windings.

The values for current, resistance, and inductance shown in the data tables for 8 lead motors assume a unipolar connection and measure from the center tap to the end of one winding. To determine the phase characteristics for other connection methods, multiply the given unipolar ratings by the conversion factors listed in the chart below that correspond to the chosen connection method.

	Unipolar Connection	Bipolar Series Connnection	Bipolar Parallel Connection
Rated Phase Current	1	0.7	1.4
Phase Resistance	1	2	0.5
Phase Inductance	1	4	1

SIZE 23 STEPPER MOTOR DATA

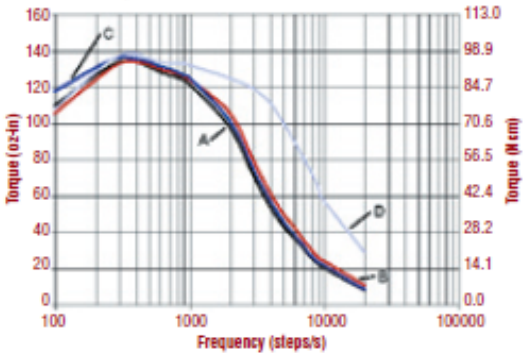


Specification	Units	HY 200 2226			
		0160 ✔	0190 ✔	0250	0470
Rated Phase Current	A	1.60	1.90	2.50	4.70
Phase Resistance	Ω	2.6	1.8	1.1	0.33
Phase Inductance	mH	4.7	3.3	4.0	0.5
Holding Torque Unipolar	oz-in	123	126	—	123
	Ncm	87	89	—	87
Holding Torque Bipolar	oz-in	154	160	161	154
	Ncm	109	113	114	109
Detent Torque	oz-in	12.0	12.0	12.0	12.0
	Ncm	8.5	8.5	8.5	8.5
Rotor Inertia	oz-in-s^{2}x10^{-4}	28	28	28	28
	g-cm^2	200	200	200	200
Motor Weight (Mass)	lb	1.5	1.5	1.5	1.5
	kg	0.70	0.70	0.70	0.70
Maximum Voltage	V	75	75	75	75
Std. No. of Leads	—	8	8	4	8

✔ Available through the MotionExpress program.

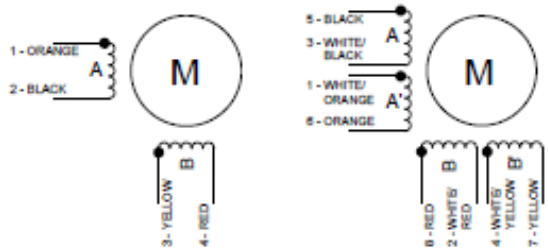
Performance

Motor Pull-Out Torque



A - 0160, Bipolar chopper, Parallel, 48V, 2.3A/Phase
B - 0190, Bipolar chopper, Parallel, 48V, 2.7A/Phase
C - 0250, Bipolar chopper, Parallel, 48V, 2.5A/Phase
D - 0470, Bipolar chopper, Parallel, 48V, 6.7A/Phase

Standard Leadwire Configuration



Standard Features

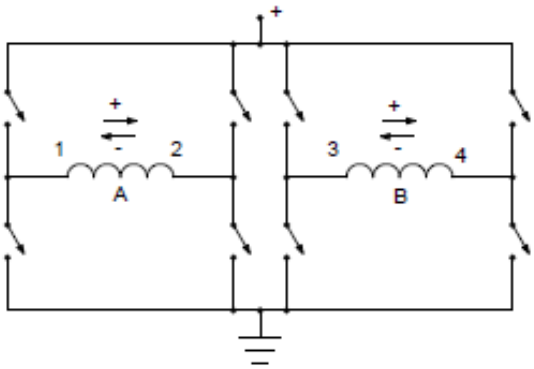
- Step angle: 1.8°
- Step angle accuracy: 5%
- Insulation class: B (130°C)
- NEMA 23 mounting configuration
- Neodymium magnets
- Additional windings and customization options available
- CE approved

Complementary Products (See Bulletin CO)

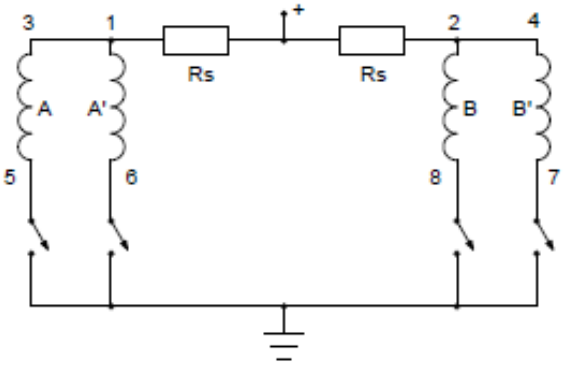
- Gearboxes
- Encoders

CONNECTION DIAGRAMS

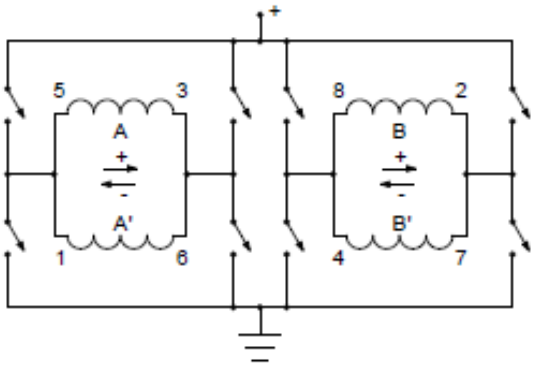
BIPOLAR



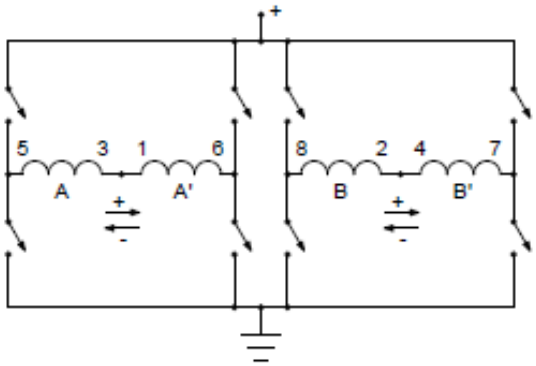
UNIPOLAR



BIPOLAR (PARALLEL)



BIPOLAR (SERIES)



STEP SEQUENCES

FULL STEP OPERATION

One Phase On							
Unipolar					*Bipolar*		
	A	A'	B	B'		A	B
1	+	0	0	0	1	0	+
2	0	0	+	0	2	–	0
3	0	+	0	0	3	0	–
4	0	0	0	+	4	+	0
1	+	0	0	0	1	0	+

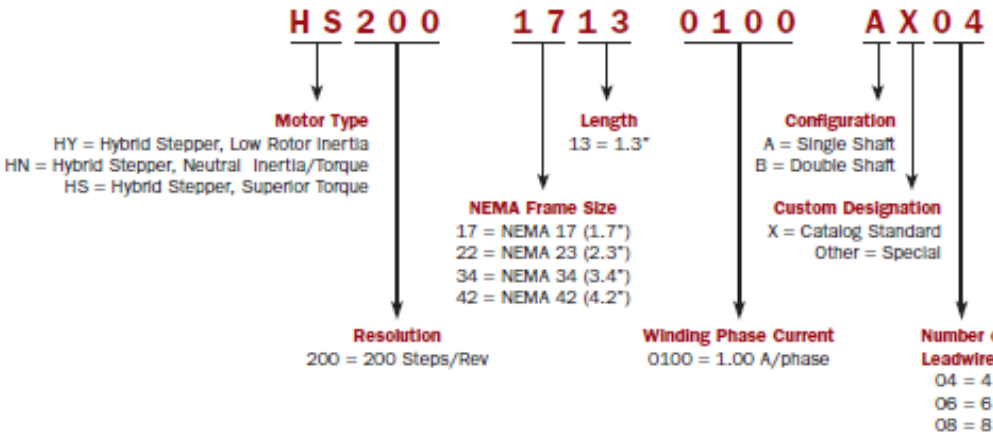
Two Phases On							
Unipolar					*Bipolar*		
	A	A'	B	B'		A	B
1	+	0	0	+	1	+	–
2	+	0	+	0	2	+	+
3	0	+	+	0	3	–	+
4	0	+	0	+	4	–	–
1	+	0	0	+	1	+	–

HALF STEP OPERATION

Unipolar					*Bipolar*		
	A	A'	B	B'		A	B
1	+	0	0	+	1	+	+
2	+	0	0	0	2	0	+
3	+	0	+	0	3	–	+
4	0	0	+	0	4	–	0
5	0	+	+	0	5	–	–
6	0	+	0	0	6	0	–
7	0	+	0	+	7	+	–
8	0	0	0	+	8	+	0
1	+	0	0	+	1	+	+

NOTE: Following the above steps in sequential order results in clockwise rotation of the shaft when viewed from the mounting end. Reversing the sequence results in counter-clockwise rotation.

CATALOG PART NUMBER DESCRIPTION



NOTE: The factory may assign a part number that differs from the catalog designation for purposes of uniqueness and brevity.

Miniature Snap Switch

Single and Double Pole

D4 Series

Features

- Choice of standard or light operating force
- Long life coil spring mechanism
- RoHS compliant
- Cadmium free
- Various aux actuators
- Various terminal types
- Agency approved extended life versions available

Standard Parts[1]

D413-R1AA-G2	D429-R1TD-G2	D433-R1AA-G2	D443-R1LD-G2	D449-R1LL-G2	D453-R1RA-G2
D413-R1LA-G2	D429-R1AA-G2	D433-R1LD-G2	D443-R1MD-G2	D449-R1MD-G2	D453-R1RD-G2
D413-R1LD-G2	D429-R1LD-G2	D433-R1RA-G2	D443-R1RA-G2	D449-R1ML-G2	D459-R1AA-G2
D413-R1RA-G2	D429-R1LL-G2	D433-R1RD-G2	D443-R1RD-G2	D449-R1RA-G2	D459-R1LD-G2
D419-R1AA-G2	D429-R1MD-G2	D439-R1LL-G2	D449-R1AA-G2	D449-R1RD-G2	D459-R1LL-G2
D423-R1RA-G2	D429-R1ML-G2	D439-V1ML-G2	D449-R1LD-G2	D453-B8AA-G2	D459-R1ML-G2
D423-R1RD-G2		D443-R1AA-G2		D453-R1AA-G2	

[1]The part number configuration matrix below provides details to the part numbers above.
For configurable part numbers that are not listed above, not listed in your region, or for custom part numbers, contact the factory or your distributor.
For information on crossing discontinued U.S. only part numbers, see D4 cross-reference chart at www.cherryswitches.com.

Part Number Breakdown

D413-R1AA-G2 (most part numbers end in –G2; see D4 cross-reference chart for exceptions)

Series: D4

Contact Rating[2]

1	0.1 A
2	3 A
3	5 A
4	10 A
5	15 A
8	21 A

Configuration[3]

85 °C			125 °C		150 °C		
1	SPST – NO	Standard Op. Force	G	SPST – NO	S	SPST – NO	Light Op. Force
2	SPST – NC		H	SPST – NC	T	SPST – NC	
3	SPDT		M	SPDT	U	SPDT	
7	SPST – NO	Light Op. Force	N	SPST – NO			
8	SPST – NC		P	SPST – NC			
9	SPDT		R	SPDT			

[2]UL AC current rating; see table on reverse for complete details
[3]D48 only available with light operating force

Terminal Type mm (inches)

A1	Welding	R1	4.8 x 0.5 (0.187 x 0.020) QC, straight**
B8	Short solder	R3	4.8 x 0.5 (0.187 x 0.020) QC, dog leg**
P4	1.2 x 0.5 (0.047 x 0.020) PCB, underside*	S1	Solder with temp stop
P5	1.2 x 0.5 (0.047 x 0.020) PCB, rear*	V1	6.3 x 0.8 (0.250 x 0.032) QC, straight
PA	1.2 x 0.5 (0.047 x 0.020) PCB, housing side*	V3	6.3 x 0.8 (0.250 x 0.032) QC, dog leg
PB	1.2 x 0.5 (0.047 x 0.020) PCB, cover side*	W9	Screw*
Q1	4.8 x 0.8 (0.187 x 0.032) QC, straight*	X5	1.5 x 0.8 (0.059 x 0.032) QC, 2.5 mm grid
Q3	4.8 x 0.8 (0.187 x 0.032) QC, dog leg*	Y5	6.3 x 0.8 (0.250 x 0.032) QC, 5 mm grid

*not available for D48 **not available for D45 or D48

Actuator Type mm (inches)

		Actuation length
AA	Button	N/A
JA	SS lever, standard ratio	21.2 (0.835)
JD	SS lever, standard ratio	35.6 (1.4)
JL	SS lever, standard ratio	69.9 (2.75)
KA	SS lever, high ratio	25.7 (1.01)
KD	SS lever, high ratio	40.1 (1.58)
KL	SS lever, high ratio	74.4 (2.93)
LA	Lever, standard ratio	21.2 (0.835)
LD	Lever, standard ratio	35.6 (1.4)
LL	Lever, standard ratio	69.9 (2.75)

		Actuation length
MA	Lever, high ratio	25.7 (1.01)
MD	Lever, standard ratio	40.1 (1.58)
ML	Lever, high ratio	74.4 (2.93)
RA	Roller, standard ratio	20.6 (0.811)
RD	Roller, standard ratio	34.1 (1.34)
SA	Roller, simulated, rear mount	20.6 (0.811)
TA	Roller, high ratio	25.1 (0.988)
TD	Roller, high ratio	38.6 (1.52)
UA	Roller, simulated, front mount	25.1 (0.988)

For actuator details, see D4 actuator reference sheet at www.cherryswitches.com

Electrical Specifications – Contact Ratings

Contact	EN61058	UL1054
1	0.1 (0.5) A, 250 VAC; 0.5 A, 30 VDC; 1 (1) A, 250 VAC	0.1 A, 125/250 VAC; 0.5 A, 30 VDC; 1 A, 125 VAC
2	3 (1) A, 250 VAC	3 A, 125/250 VAC; 1/10 HP, 250 VAC
3	6 (2) A, 250 VAC	5 A, 125/250 VAC; ¼ HP, 250 VAC
4	10 (3) A, 250 VAC	10 A, 125/250 VAC; 6 A, 30 VDC; ½ HP, 125/250 VAC
5	16 (4) A, 250 VAC	15 A, 125/250 VAC; ½ HP, 125-250 VAC
8	21 (8) A, 250 VAC	21 A, 250 VAC; 1 HP, 125 VAC; 2 HP, 250 VAC

UL file number E314201

Environmental and Mechanical Specifications

Operating Temperature		-40 °C to 85, 125 or 150 °C (-40 °F to 185, 257 or 302 °F)
Flammability Rating		UL 94V-0
Operating life – electrical life at rated load, 85 °C		
Contact	EN61058	UL1054[4]
1	50,000	6,000
2	50,000	6,000
3	50,000	6,000
4	50,000	6,000
5	50,000	6,000
7	10,000	6,000

[4]Extended life available

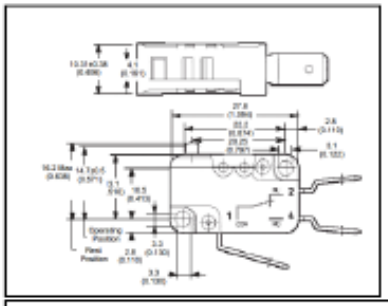
Material Specifications

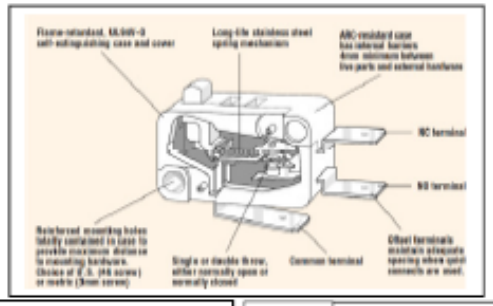
Case	Thermoplastic Polyester PET
Cover	Thermoplastic Polyester PET
Actuating Button	Thermoplastic Acetyl (POM) 85 °C, Thermoplastic Polyester (PET) 125 or 150 °C
Contacts	Gold Cross Point (D41), Silver (D42), Silver Alloy (D43-D48)
Terminals	Brass
Auxiliary Actuator	Stainless Steel or Cold-Rolled Steel (Nickel Plated)
Moving Blade	Silver Plated Brass

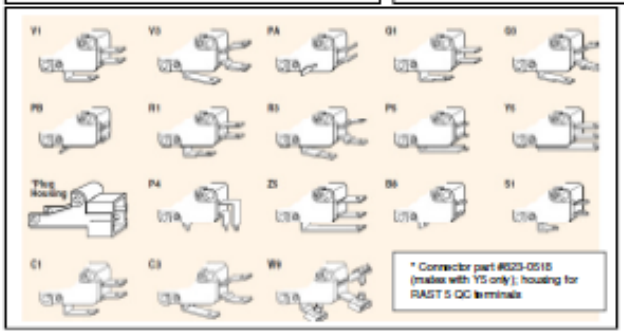
Environmental Specifications

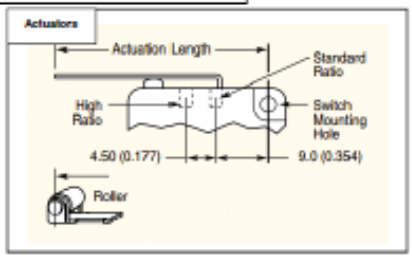
Ambient Temperature	-40 °C to 85 °C/120 °C (-40 °F to 185 °F/248 °F)
Operating Force	70 – 280 cN (model dependent, without actuator)
Total Travel	1.6 mm
Ingress Protection	IP50

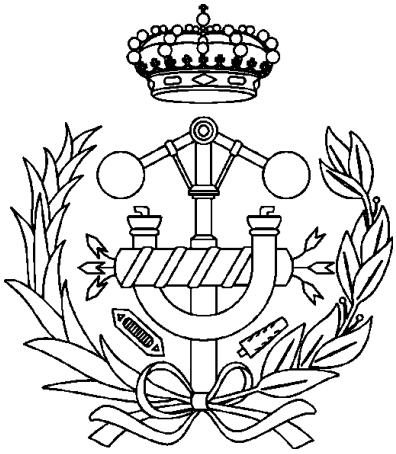
Dimensions mm (inches) and terminal configurations











Anexo III:

Imágenes sistema mecánico

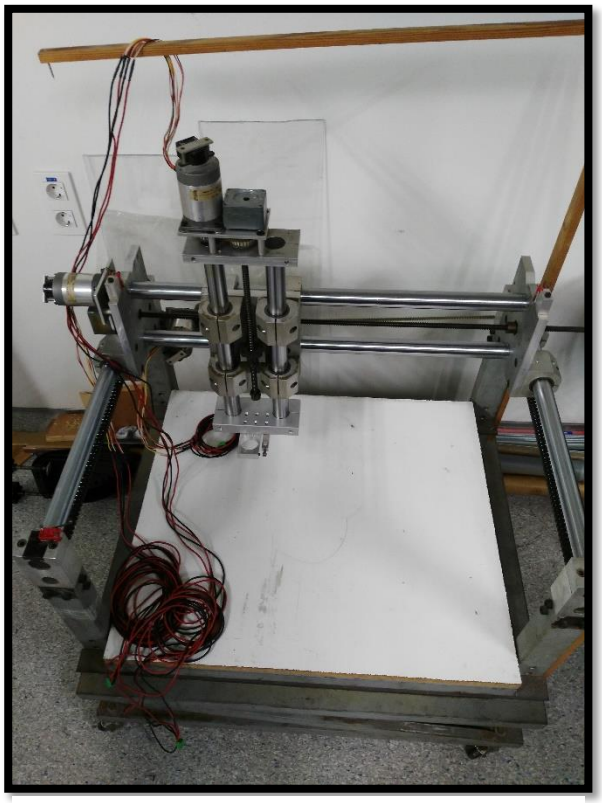


Ilustración 45: Estructura

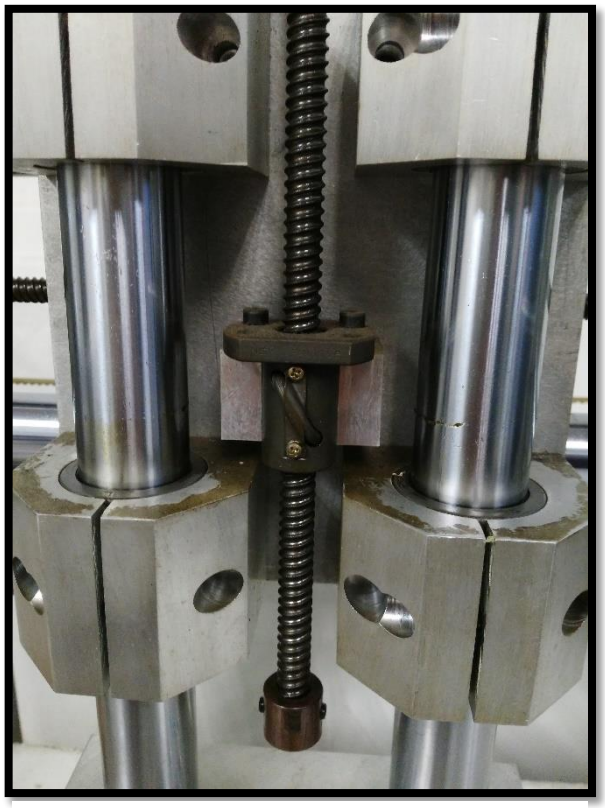


Ilustración 44: Husillo de bolas

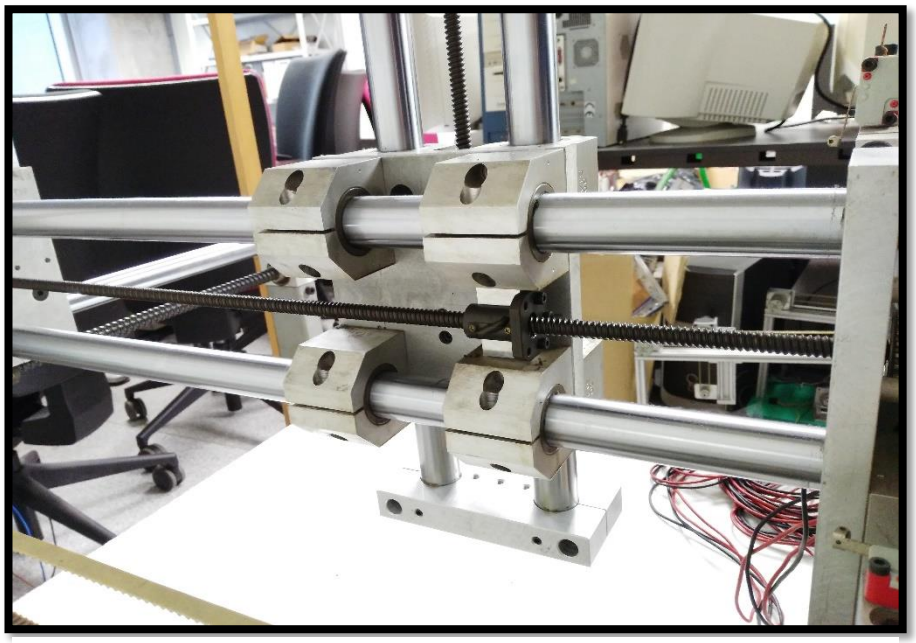


Ilustración 46: Barra lisa y rodamientos lineales del eje X

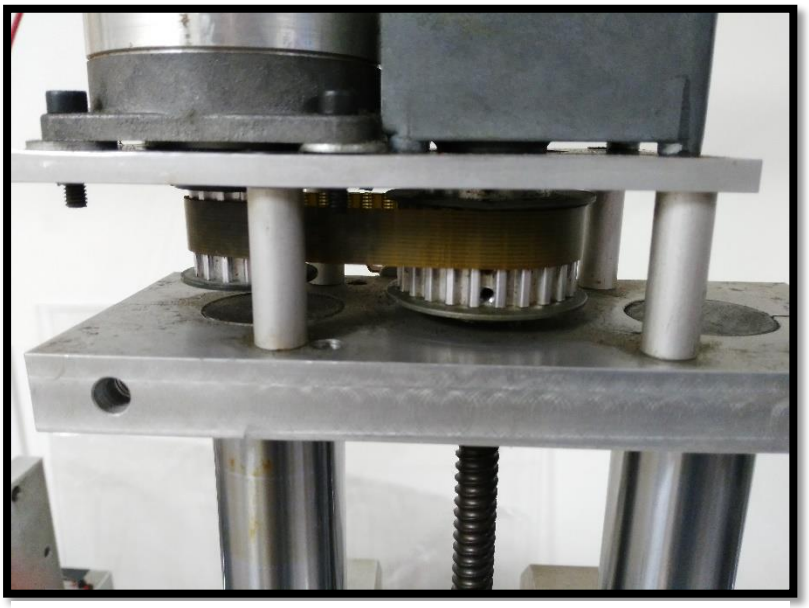


Ilustración 47: Transmisión por correa eje Z

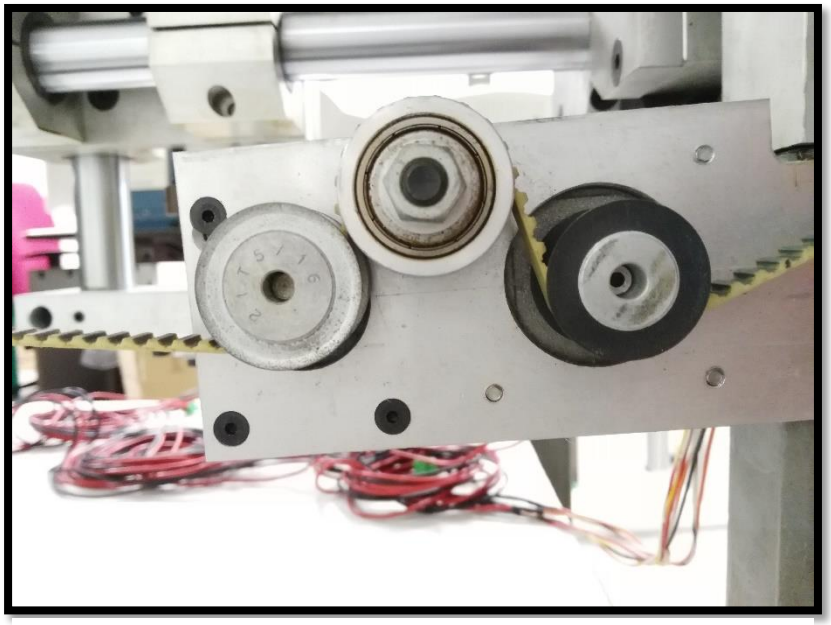


Ilustración 48: Sistema de poleas

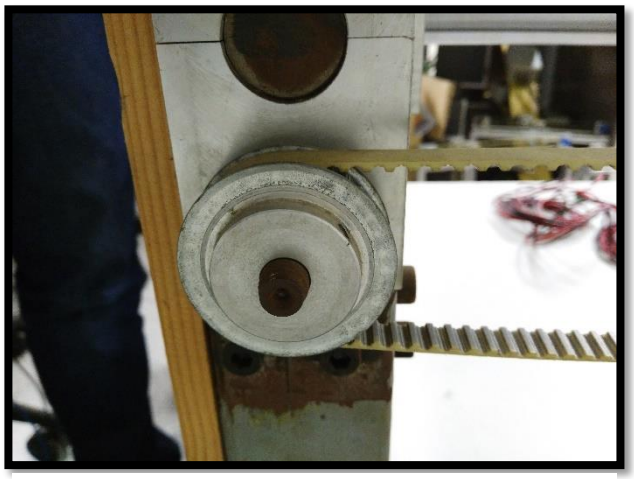


Ilustración 50: Polea y correa

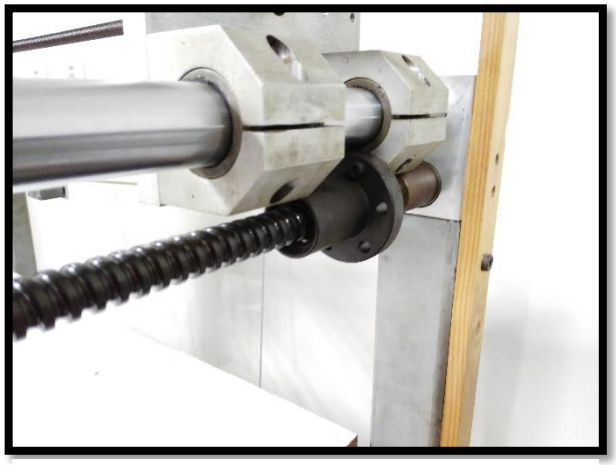


Ilustración 49: Husillo y sistema de guiado eje X

Anexo IV:

Imágenes parte electrónica

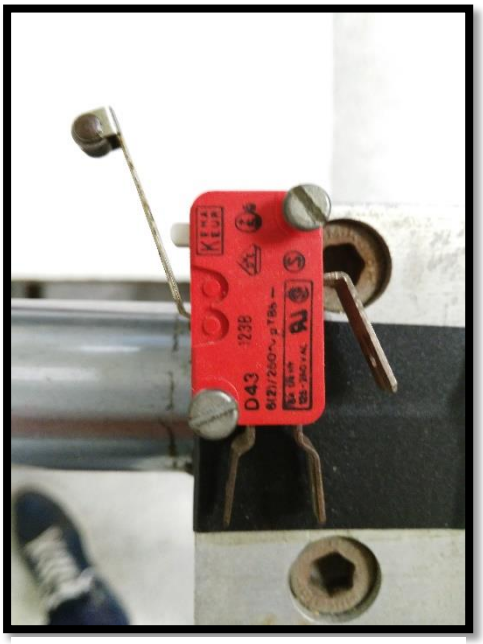


Ilustración 51: Final de carrera

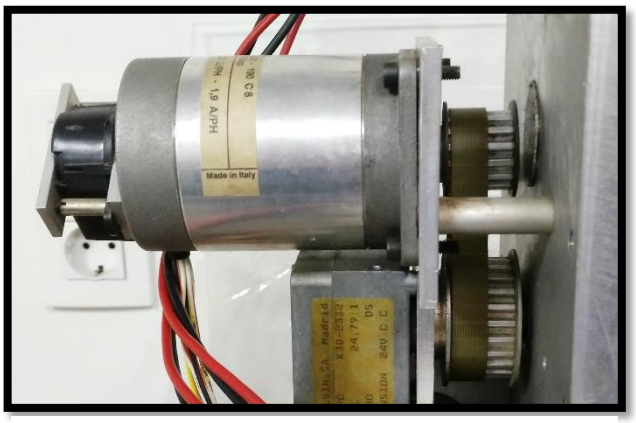


Ilustración 52: Motor eje X

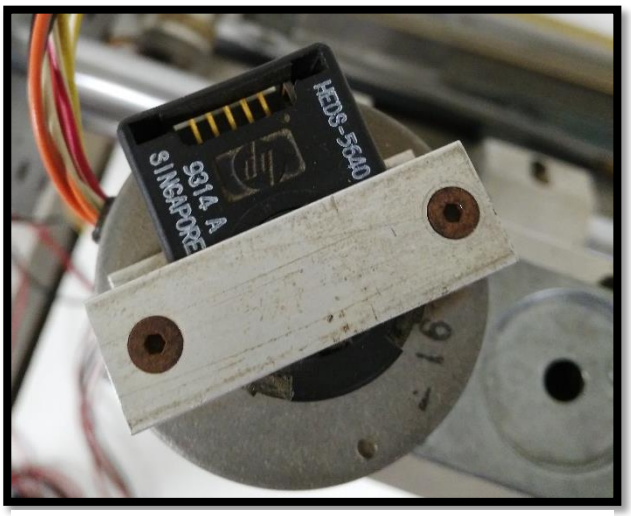


Ilustración 54: Encoder óptico eje Z

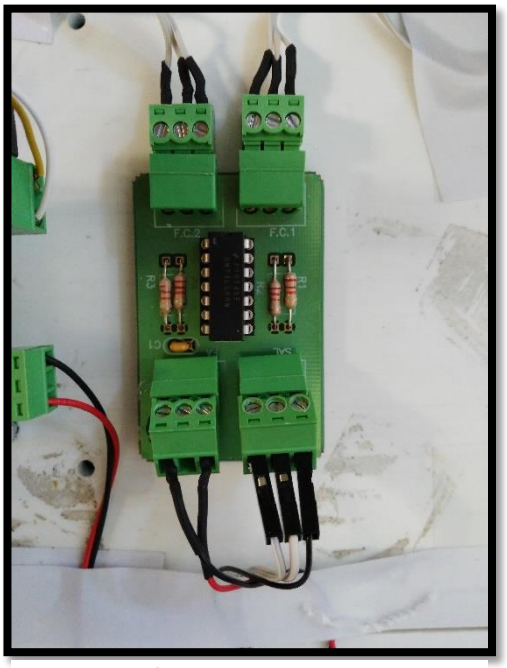


Ilustración 53: Circuito antirebote

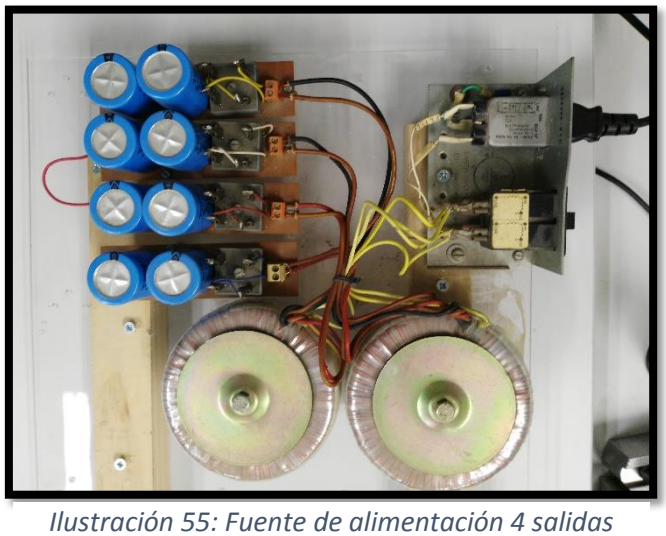


Ilustración 55: Fuente de alimentación 4 salidas

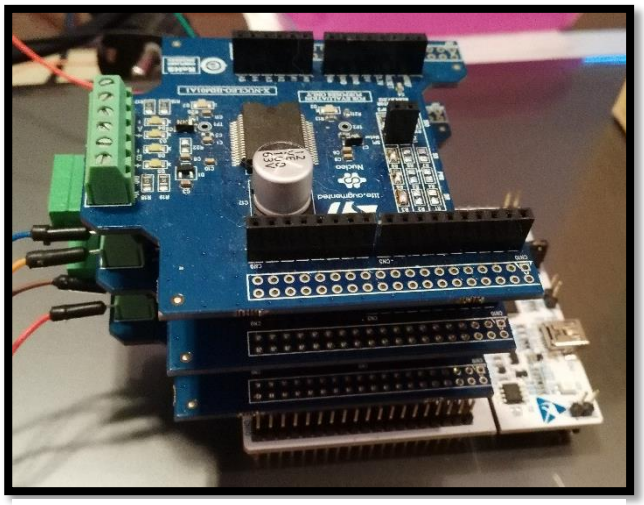


Ilustración 57: Vista de perfil del microcontrolador y drivers conectados

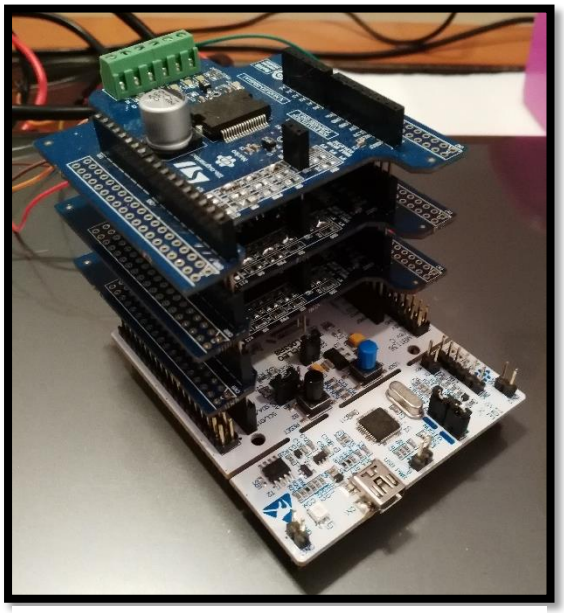


Ilustración 56: Vista alzado del microcontrolador y los drivers conectados

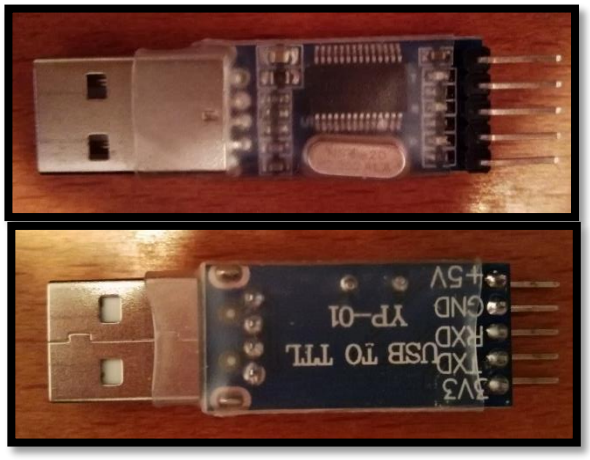


Ilustración 59: Convertidor USB - serie

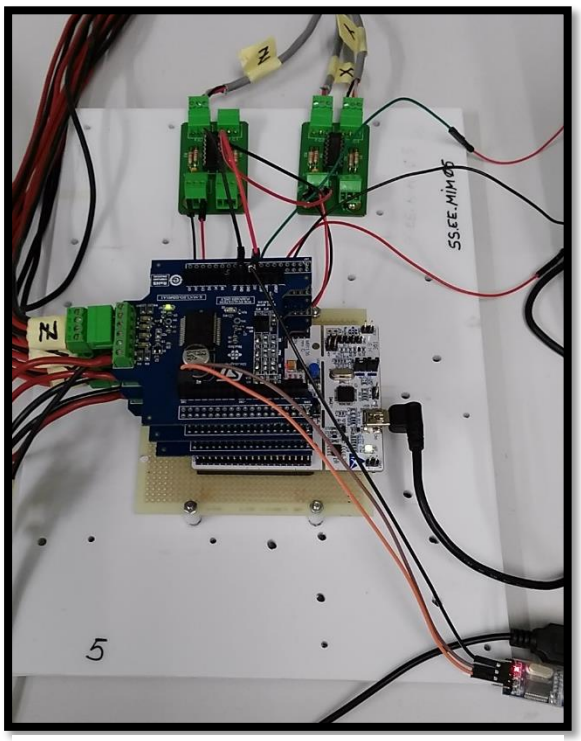


Ilustración 58: Conexionado de elementos

Anexo V:

Imágenes sistema completo

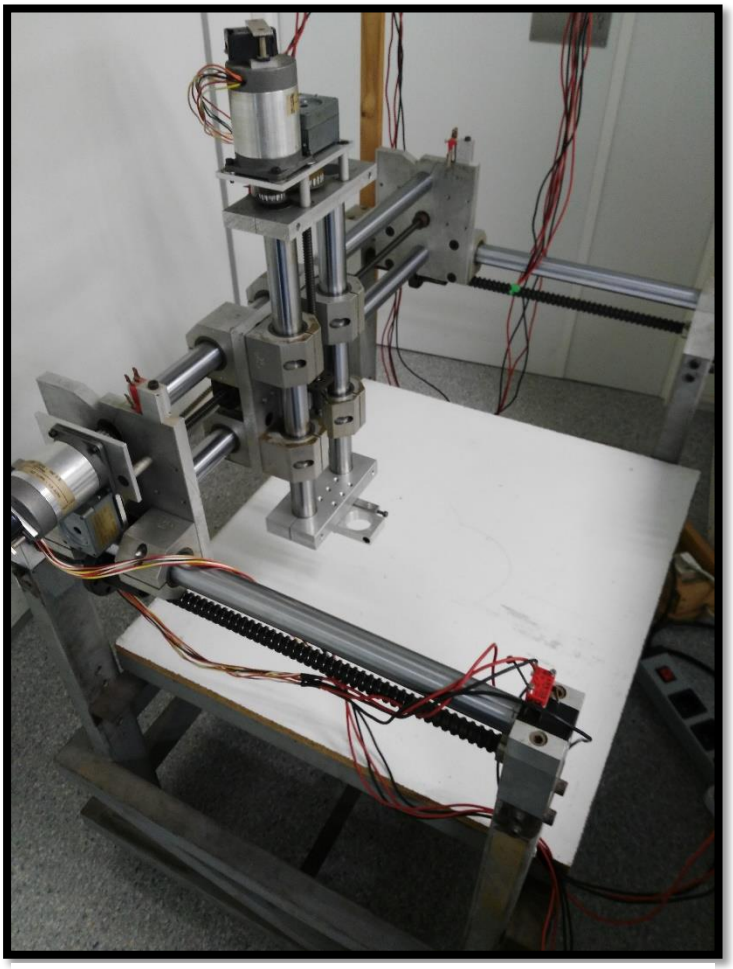
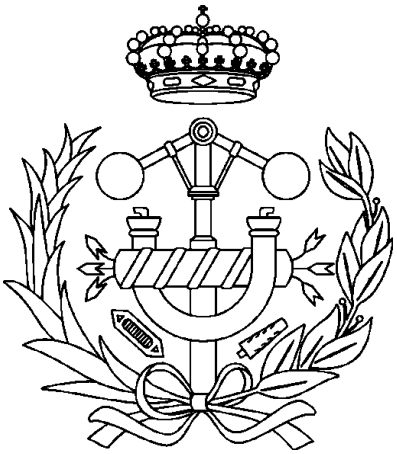


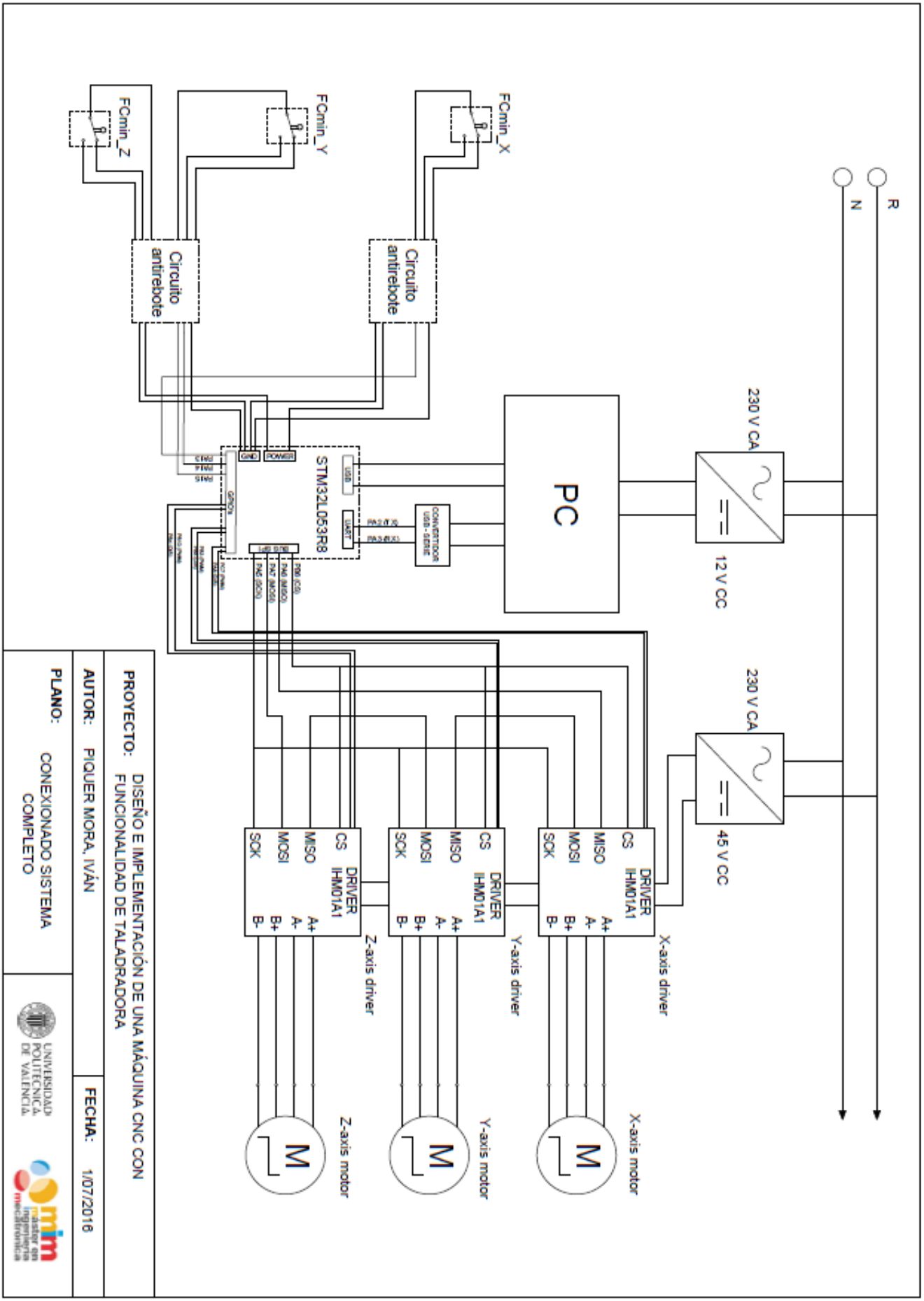
Ilustración 60: Vista general del sistema

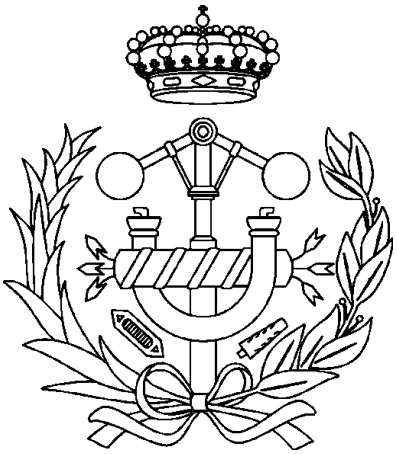
Ilustración 61: Vista frontal del sistema completo



Anexo VI:

Planos


R
N
230 V CA
12 V CC
230 V CA
45 V CC
PC
FCmin_X
FCmin_Y
FCmin_Z
Circuito antirebote
Circuito antirebote
Circuito antirebote
STM32L053R8
GND
POWER
USB
UART
CONVERTIDOR USB - SERIE
DRIVER IHM01A1
CS
MOSI
MISO
SCK
A+
A-
B+
B-
X-axis driver
Y-axis driver
Z-axis driver
X-axis motor
Y-axis motor
Z-axis motor
M
PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UNA MÁQUINA CNC CON FUNCIONALIDAD DE TALADRADORA
AUTOR: PIQUER MORA, IVÁN
FECHA: 1/07/2016
PLANO: CONEXIONADO SISTEMA COMPLETO
UNIVERSIDAD POLITECNICA DE VALENCIA
máster en ingeniería mecatrónica



Anexo VII:

Presupuesto

Objeto

El presente documento tiene por objeto plasmar y reflejar la totalidad de los elementos que intervienen en la fabricación de la máquina CNC.

Por tanto, recoge tanto los materiales electrónicos como mecánicos. En él se van a mostrar los precios de cada uno de ellos con el fin de tener la certeza de la procedencia de cada uno de los costes aplicados a la misma y tener conocimiento del gasto que se va a generar.

Se debe tener en cuenta que los precios no incluyen el IVA, ya que se tiene en cuenta sobre la cuantía final.

Es por tanto que este documento, junto con la memoria, el código y los planos, supone la base más importante y necesaria para el desarrollo y realización de la máquina CNC que se ha desarrollado.

Nº producto	*Elemento*	*Precio*	*Cantidad*	*Importe total*
	PARTE ELECTRÓNICA			
1	Microcontrolador STM32L053R8	10,50 €	1 ud	10,50 €
2	Motor paso a paso MAE HY200-2232-190C8	150 €	3 uds	450 €
3	Driver motor paso a paso	11,44 €	3 uds	34,32 €
4	Convertidor serie-TTL	5,30 €	1 ud	5,30 €
5	Final de carrera	1,33 €	3 uds	3,99 €
6	Fuente de alimentación 48 V – 20 A	93,50 €	1 ud	93,50 €
7	Cable	0,55 €/m	8 uds	4,40 €
8	Máquina herramienta para taladrar /fresar	25,00 €	1 ud	25,00 €
	PARTE MECÁNICA			
9	Estructura	40 €	1 ud	40 €
10	Barra lisa calibrada	35 €/m	6 uds	210 €
11	Husillo + tuerca de bolas	62 €	4 ud	248 €
12	Polea dentada 20 dientes	7,30 €	3 uds	21,90 €
13	Polea dentada 40 dientes	13,90 €	3 uds	41,70 €
14	Correa dentada 48 dientes	15,34 €	3 uds	46,02 €
15	Rodamiento lineal	17,30 €	12 uds	207,60 €
16	**Base imponible**	**-**	**-**	**1442,23 €**
17	**Importe IVA**	**21 %**	**-**	**302,87 €**
18	**Importe total**	**-**	**-**	**1745,10 €**