UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Caracterización del sistema electrónico de una CNC e implementación del programa Arduino para el control de la máquina

Trabajo Fin de Grado

Grado en Ingeniería Mecánica

AUTOR/A: Dolz Molina, Alejandro

Tutor/a: Masot Peris, Rafael

CURSO ACADÉMICO: 2023/2024

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería del Diseño

CARACTERIZACIÓN DEL SISTEMA ELECTRÓNICO DE UNA CNC E IMPLEMENTACIÓN DEL PROGRAMA ARDUINO PARA EL CONTROL DE LA MÁQUINA.

Trabajo Fin de Grado

Grado en Ingeniería Mecánica

Autor: Alejandro Dolz Molina

Tutor: Rafael Masot Peris

Curso académico 2023/24

Resumen

Las máquinas herramienta de control numérico, o por sus siglas CNC, tienen en la actualidad gran importancia en la fabricación a nivel industrial, en maquinaria como fresadoras, tornos o cortadoras láser. En este sentido este proyecto pretende obtener un programa capaz de hacer que una máquina CNC sea capaz de interpretar código G para dibujar una imagen enviada por el usuario. Para ello se han estudiado primero todos los componentes, tanto eléctricos, electrónicos y mecánicos que componen una de estas máquinas, para después crear, mediante Arduino y Processing un programa que puedan tomar la imagen que envía el usuario y crear una rutina para dibujar dicha imagen sobre un lienzo. Como resultado se han obtenido varios dibujos que demuestran las capacidades tanto del programa como del montaje físico. Con todo esto podemos afirmar que hemos desarrollado una herramienta perfecta para que los alumnos se familiaricen tanto con la programación como con máquinas CNC.

Palabras clave

Control Numérico por Computador; CNC; Arduino; Processing; Motores paso a paso; Automatización.

Abstract

Numerical control machine tools or CNC machines are currently of immense importance in industrial manufacturing, in machinery such as milling machines, lathes or laser cutters. In this sense, this project aims to obtain a program capable of making a CNC machine able to interpret G code to draw an image sent by the user. To do this we have first studied all the components, both electrical, electronic and mechanical that make up one of these machines, and then create, using Arduino and Processing, a program capable of take any image sent by the user to then create a routine to draw the image on a canvas. As a result, we have obtained several drawings that demonstrate the capabilities of both the program and the physical assembly. With all this we can say that we have developed a perfect tool for students to become familiar with both programming and CNC machines.

Key words

Computer Numerical Control; CNC; Arduino; Processing; Stepper Motors; Automation.

Índice general:

Índice de Figuras:

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería del Diseño

CARACTERIZACIÓN DEL SISTEMA ELECTRÓNICO DE UNA CNC E IMPLEMENTACIÓN DEL PROGRAMA ARDUINO PARA EL CONTROL DE LA MÁQUINA.

Documento 1º: Memoria

Autor: Alejandro Dolz Molina

Tutor: Rafael Masot Peris

Curso académico 2023/24

Índice de la memoria:

1. INTRODUCCIÓN

1.1. Contextualización

Desde su invención entre las décadas de 1940 y 1950 las máquinas CNC (control numérico por computadora) han sido uno de los avances más significativos en el campo de la fabricación del siglo XX y a lo largo de las últimas décadas han experimentado un gran desarrollo. Actualmente gozan de una gran extensión en la industria gracias a su precisión, su eficiencia y a la flexibilidad de su estructura. Sus usos comprenden desde fresadoras y tornos, pasando por máquinas de corte hidráulico, hasta impresoras 3D. Su sistema se basa en el control del movimiento de la herramienta en varios ejes, ya sean 2, 3 o hasta 5 ejes.

La creación de las CNC se le atribuye al ingeniero John T. Parsons quien quería implementar mejores métodos para mecanizar las hélices de un helicóptero, ya que cada vez se necesitaban más puntos para definir dichas geometrías, pero los tiempos de fabricación eran más y más largos con cada punto que se añadía. Para solucionar esto con calculó dichas coordenadas con un multiplicador IBM 602A y volcó la información en tarjetas perforadas, patentando la primera máquina por control numérico por computador de la historia en el año 1958.

Con esta base Parsons continuó trabajando para mejorar más el sistema y para ello se alió con investigadores del Massachusetts Institute of Technology (MIT) quienes mejoraron el prototipo enormemente. Ahora, no solo se movía de un punto A a un punto B, sino que lo hacía de manera suave y precisa. También se mejoró la manera en la que se organizaba la información, pasando de una tarjeta perforada a tres tiras perforadas, una para cada eje de la máquina. Este modelo se presentó en 1952 cuando fue comprado por la Fuerza Aérea de los Estados Unidos.

Durante las siguientes décadas las máquinas de CNC tuvieron una amplia difusión en todo tipo de industrias. También ha mejorado en gran medida la manera de almacenar los movimientos, pasando de las tiras perforadas al *floppy disc* y actualmente se recurren a programas informáticos para ello.

Finalmente se puede decir con seguridad que el fututo de las CNC es prometedor, con la implementación de otras tecnologías emergentes como la inteligencia artificial o el *machine learning.* Estas innovaciones auguran que la automatización y el control de procesos llegará a nuevos niveles permitiendo métodos para la optimización de tareas o la capacidad de autoajuste en tiempo real.

1.2. Motivación

La principal motivación para hacer este proyecto viene por mi personal interés por la electrónica y la automatización, y es que, la ingeniería mecánica y la electrónica son ciencias hermanas que actualmente trabajan conjuntamente en gran cantidad de proyectos de ingeniería. Un ejemplo perfecto de esta simbiosis es la máquina de control numérico, donde es igual de importante la construcción de la máquina como la creación de un buen sistema electrónico y de control que sea capaz de aprovecharlo al máximo.

Es por esto por lo que este proyecto se centra tanto en la caracterización de los elementos mecánicos que componen una máquina CNC como en el estudio de constituyentes electrónicos que la componen.

2. OBJETIVOS

El objetivo del proyecto es en primer lugar comprender el funcionamiento básico de la fresadora, los motores paso a paso, los drivers y la placa base. En segundo lugar y a partir de estos conocimientos de deberá desarrollar una serie de programas que sean capaces de controlar la máquina de manera directa por el usuario y de forma automática siguiendo una rutina, para finalmente obtener un dibujo hecho enteramente por la máquina.

De esta manera, el usuario podrá o bien escoger si quiere indicarle de manera manual qué movimientos tiene que hacer la herramienta, o bien si quiere aportar una imagen para que el programa la interprete y, posteriormente y de manera automática, la máquina sea capaz de dibujar dicha imagen.

3. COMPONENTES

3.1. Fresadora CNC 1610

Como base para este proyecto se ha escogido una fresadora CNC 1610, ya que se trata de una herramienta muy compacta y versátil, contando con tres ejes, y con la capacidad de adaptarse para mecanizar materiales diversos como plásticos, madera y metales blandos. Cuneta incluso con una herramienta para grabado o cortado por láser. Pero para este proyecto hemos decidido no usar ninguna de estas, en su lugar lo sustituiremos por un soporte especializado que sostenga un bolígrafo o lápiz para así dibujar sobre un papel aquello que se le envíe.

Sus dimensiones aproximadas son de 260x140x220 mm lo que la hace perfecta para un uso de sobremesa. Sus ejes giran propulsados por motores paso a paso lo que sumado a los husillos de avance de gran calidad que monta hace que el conjunto sea sumamente preciso en sus movimientos.

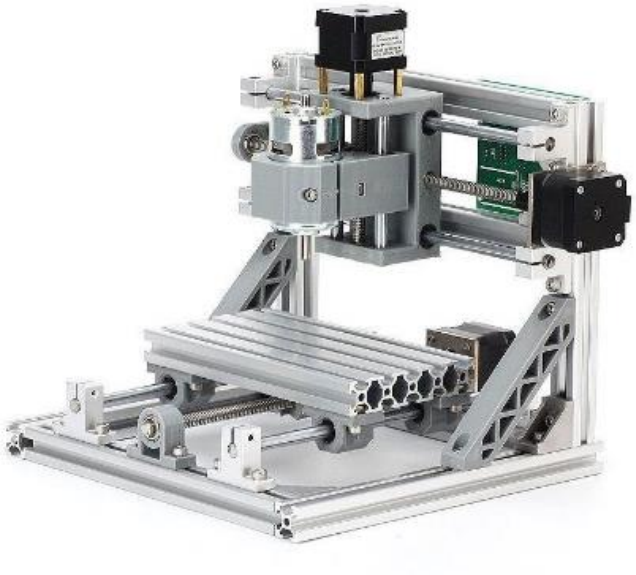


Figura 1: Fresadora CNC 1610

3.2. Motores NEMA17

Los motores que controlarán los ejes de la máquina son los NEMA 17. Esta denominación es un estándar en la industria, que viene de las pulgadas de alto y ancho de la cara frontal.

Los NEMA son un tipo de motores de corriente continua sin escobillas que rotan variando el campo magnético de las bobinas en su interior. En concreto se trata motor bipolar que, significa que energizan toda la bobina para cambiar el campo magnético de esta, a diferencia de los motores unipolares que tienen un cable central que energiza la mitad del devanado. Por un lado, esto hace que sean capaces de desarrollar un par mayor, pero, por otro lado, provoca que controlar la dirección de la corriente sea más difícil, normalmente usando para esta tarea controladores con puentes en H integrados.

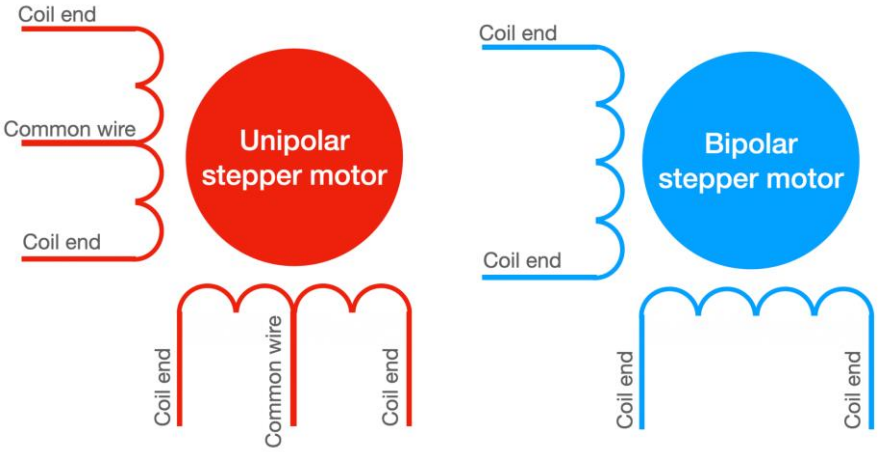


Figura 2: Esquema bobinas motores paso a paso

Dentro de los motores NEMA de 17 pulgadas hay tres variantes según el tamaño, 17S, 17M y 17L. En nuestro caso hemos usado el intermedio NEMA 17M. Algunas de las características que lo definen son: un par de 25 Nmm, y un peso de 0,44 kg, lo que lo hace perfecto para este tipo de aplicaciones. Su tensión de alimentación es entre 14 y 48 VDC y su corriente por fase es de 1.8 A. Su ángulo de paso es de 1.8 °, o lo que es lo mismo, 200 pasos por revolución, esto sin contar las fracciones de pasos que es capaz de realizar gracias a los controladores.

Figura 3: Motor Nema 17

3.3. *Driver* A4988

Como ya hemos mencionado antes, es muy común usar *drivers* para controlar la polarización de las bobinas en los motores paso a paso bipolares. En nuestro caso hemos optado por el controlador A4988 por los siguientes motivos: la gran cantidad de información que existe sobre este controlador debido a su extenso uso y por su bajo precio. Además, nos ofrece la capacidad de controlar la corriente que recibe el motor e incorpora un sistema de protección mediante el cual se apaga si la corriente o la temperatura exceden niveles normales. Finalmente, y aunque no lo vayamos a utilizar, también podemos con este mismo controlador hacer las siguientes fracciones de pasos: 1/2, 1/4, 1/8, y 1/16.

Figura 4: Driver A4988

Es capaz de proporcionar una corriente de salida de hasta 2 A si se le incorpora un disipador. Su alimentación principal admite valores en el rango de 8-35 V, aunque nosotros lo alimentaremos con 15 V o 24 V por motivos de seguridad. Para finalizar su tensión lógica se puede situar entre 3,3-5,5 V, y nosotros lo alimentaremos con 5 V ya que es además la tensión que requieren otros componentes como el Arduino NANO.

Cuenta con un total de 16 pines, empezando por arriba a la izquierda en la Figura 5 tenemos el pin del *ENABLE*, que desactiva todos los outputs cuando se le suministra un 1 lógico. Los pines MS1, MS2 y MS3 controlan la resolución de los pasos. Los pines de *RESET* y *SLEEP* deben recibir ambos un 1 lógico para el correcto funcionamiento del controlador. La entrada de *STEP* hace que, cuando se pasa de un estado bajo a uno alto el motor da un paso, y el pin de *DIRECTION* controla la dirección en la que gira el motor. Siguiendo por el flanco derecho tenemos *VMOT* y *GND* que suministrarán la tensión. Los pines *2A*, *2B*, *1A* y *1B* irán conectados al motor. Es importante que se conecten los pines con un mismo número a una misma bobina, si no es así el motor no funcionará. El orden en que se conecten no es tan importante y solo variará la dirección de giro. Para acabar *VDD* y *GND* definen la tensión lógica que usará el chip.

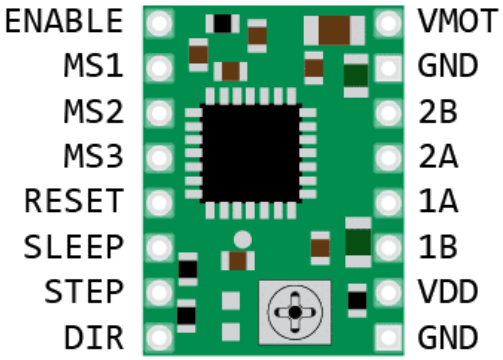


Figura 5: Pinout del driver A4988

3.3.1. Ajuste del voltaje de referencia

Este controlador incorpora también un pequeño potenciómetro, (como se puede apreciar a la derecha del chip en la *Figura 4*) con el que podemos regular el voltaje que reciben los motores y evitar así que estos sufran daños. Pero esto lo tenemos que hacer con mucho cuidado ya que si nos quedamos cortos y limitamos en exceso la corriente los motores no serán capaces de mover los tornillos guía de la CNC.

Para asegurar un correcto funcionamiento primero incrementaremos poco a poco la tensión hasta que veamos que los motores se mueven con suavidad y sin tirones, y posteriormente, para comprobar que estamos trabajando en un rango seguro mediremos con el multímetro y calcularemos a con dicha medida cuál es la corriente máxima que puede recibir el motor, asegurándonos de que es inferior a la corriente máxima establecida por el fabricante.

De esta manera obtuvimos que la tensión de referencia, que en nuestro caso se sitúa en 0,737 V, que tras aplicar la ecuación que aparece en la ficha técnica del *driver*:

$$I_{TripMAX} = V_{Ref}/(8 \times Rs) \quad (1)$$

Siendo $I_{TripMAX}$ (I) máxima corriente que recibe el motor y *Rs* (Ω) la resistencia de detección de corriente, en nuestro caso 0,1 Ω. Sustituyendo en la ecuación (1) obtenemos:

$$I_{TripMAX} = 0{,}737/(8 \times 0{,}1) = 0{,}921\ A \quad (2)$$

Con lo que podemos concluir que los motores no sufrirán ningún daño.

3.4. Arduino NANO

El microcontrolador Arduino NANO es la base del proyecto y será el encargado de procesar la información que reciba del ordenador, y generar los pulsos que hagan que los motores se muevan. El Arduino NANO está basado en microchip Atmega328 de 8 bits cuenta con 32 KB de memoria Flash y 2KB de memoria SRAM. Esta se puede alimentar o bien con 5 V a través del pin designado o mediante un cable USB Mini-B. Cuenta además con un total de 14 pines de entradas/salidas digitales, 8 pines de entrada analógica y 6 pines PWM (modulación de ancho de pulso).

El Arduino NANO es perfecto para proyectos donde el consumo de energía deba de ser bajo, como sería el caso de aquellos que deben de ser alimentados por batería, aunque también es muy útil en ocasiones en las cuales no se requiere de mucha potencia de computación o cuando el espacio de trabajo es muy reducido.

En nuestro caso nos hemos decidido por este microcontrolador por dos motivos. En primer lugar, por su tamaño reducido (tan solo mide 18 mm x 45 mm) lo que le permite ser implementado con facilidad en circuitos integrados como el nuestro. En segundo lugar, por la gran comunidad de usuarios que existe detrás, que hace que el aprendizaje del programa y la implementación en proyectos sea de lo más sencilla, gracias a los numerosos recursos como librerías y ejemplos desarrollados por otros usuarios, que están disponibles en los foros y páginas oficiales.

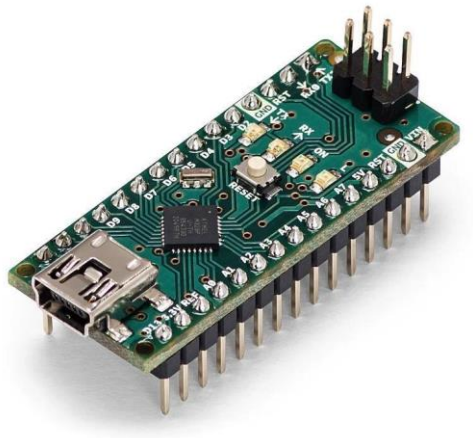


Figura 6: Arduino NANO

3.5. Placa de circuito

3.5.1. Características principales

La placa de circuito con la que hemos trabajado a lo largo de este proyecto ha sido el resultado del TFG de otro alumno, "Control por computador de una máquina fresadora CNC y desarrollo de una aplicación móvil para controlarla", de Juan Criado Vivó (2023). En dicho TFG el autor explica con detalle cuáles son las especificaciones de la placa, el proceso que se siguió para crearla, desde los primeros diseños que se llevaron a cabo en el *software* Proteus, pasando por los tratamientos químicos que se le dieron y el soldado de todos los componentes necesarios para su funcionamiento. Aunque el objetivo del presente TFG no sea el diseño de esta placa si es interesante detenerse a estudiarla.

La placa cuenta con dos opciones para la alimentación, o bien 15V a través de una fuente de alimentación o 24V con un transformador de ordenador. Posee también un regulador de tensión que es el encargado de trasformar esta tensión de la entrada a los 5V para la señal lógica.

Pasando a los componentes externos tenemos por un lado el Arduino NANO que se encuentra en el lado derecho y los espacios para los drivers A4988 en la parte superior, que se conectan a los motores a través de 4 cables que entran de manera directa a los espacios habilitados justo por encima.

Para finalizar tenemos el espacio que se reservó en un inicio para conectar los finales de carrera y que, por desgracia no se llegó a aprovechar y un interruptor, que actúa sobre la alimentación principal con un led que nos indica si el circuito está siendo alimentado o no.

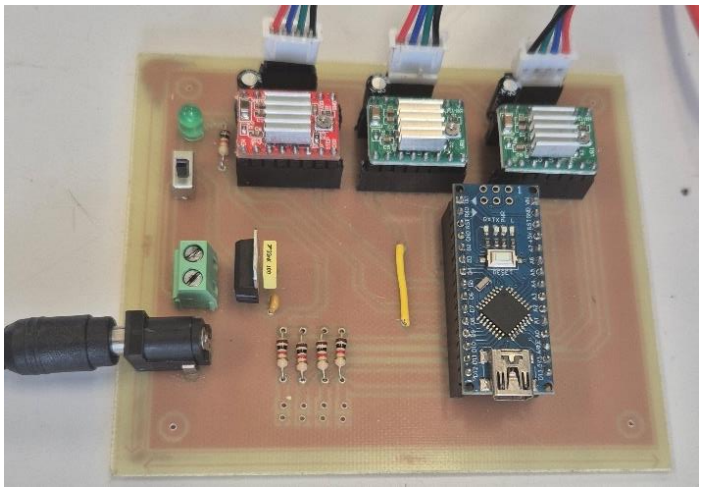


Figura 7: Fotografía de la placa de circuito

3.5.2. Problemas de la placa

La placa que desarrolló este compañero es sin duda de muy buena calidad y tiene detalles como el LED o el interruptor que son francamente útiles, pero sin embargo tiene algunos otros pormenores que todavía pueden mejorar.

Para empezar el diseño propio de la placa nos impide hacer uso de una de las características más convenientes del *driver* que hemos elegido, y esta es la capacidad de hacer fracciones de pasos. Es detalle de lo más útil a la hora de usar uno de estos drivers ya que aumenta drásticamente el nivel de precisión al que pueden trabajar estos motores. Solo usando una salida digital del Arduino por motor, es decir tres salidas digitales, seríamos capaces de hacer cuartos de paso, que es un salto muy grande de precisión a coste 0, y si usáramos dos podríamos hacer medios, cuartos y octavos de pasos.

Para continuar podría ser interesante alimentar al Arduino con los 5V que obtenemos del regulador de tensión así, si le implementamos un programa para hacer una rutina, no tendría que estar conectado todo el tiempo a la alimentación del ordenador para funcionar.

Finalmente, en el caso que haya un problema, el comportamiento más normal es el de cortar la alimentación del sistema usando el interruptor que tiene integrada la placa, pero esto no es del todo seguro para los chips del circuito, pudiendo estos dañarse si le cortamos la alimentación principal. Es por esto por lo que puede ser una buena idea incluir otro interruptor que tan solo corte la corriente de 5V, siendo esta una opción más segura para cuando haya algún problema en el funcionamiento.

3.6. Soporte de la herramienta

Originalmente la máquina viene preparada para usar como herramienta o bien una pequeña fresadora o bien un haz láser, pero para este proyecto se decidió cambiar estas herramientas por algo más seguro, y a su vez capaz de dibujar sobre un papel o un lienzo. Es por esto por lo que se debe adaptar un soporte que sea capaz de agarrar o bien un lápiz, un bolígrafo o un rotulador.

3.6.1. Elección de herramienta

Una vez que se sabía que se iba a cambiar a un rotulador o similar debíamos decidir por qué herramienta nos íbamos a decantar, ya que un lápiz no tiene la misma sección que un bolígrafo ni que un rotulador y por lo tanto el soporte sería para cada caso distinto.

Primero se descartó la opción del lápiz, ya que al ser programas en los que se iba a dibujar mucho, la punta iba a ir desapareciendo hasta llegar a un punto en el que ni siquiera fuese capaz de pintar.

Más tarde se probó con un bolígrafo BIC, que en un inicio era un avance considerable con respecto al lápiz. No se desgastaba la punta, era más rígido y no dejaba restos de grafito. Pero tenía un problema, y es que se necesitaba bastante presión para que consiguiera dibujar y en ciertos momentos rasgó el papel sobre el que se estaba trabajando, teniendo que empezar de nuevo el dibujo.

Es por esto por lo que finalmente nos decantamos por un bolígrafo calibrado de 0.5 mm. Las principales ventajas de estos bolígrafos son que no se le gasta la punta y que no necesita apenas tocar el papel para dibujar, asimismo, al igual que el BIC, es bastante rígido. Sin embargo, tiene un problema, y es que los movimientos tienen que ser rápidos, si no, la tinta manchará el papel y podría dañarlo.

3.6.2. Diseño del soporte

Para diseñar el soporte teníamos que hacer que encajase en el sitio que había dejado la herramienta anterior (el haz láser), en adición a esto debía de agarrar con fuerza el rotulador para que no se deslizase a la hora de dibujar. Con respecto al material se eligió PLA impreso en 3D ya que es de muy fácil acceso y al no estar sometido a grandes esfuerzos cumplirá nuestras expectativas.

Conociendo esto se llegó al resultado que se puede ver en la *Figura 8: Diseño del soporte*. Con unas colas de milano que aseguran que no se va a desprender y un orificio por el que se deslizará el bolígrafo, agarrándolo con fuerza. Cabe señalar que para que no haya problemas de ajuste se han dejado 0,3 mm de holgura entre las caras de las colas de milano.

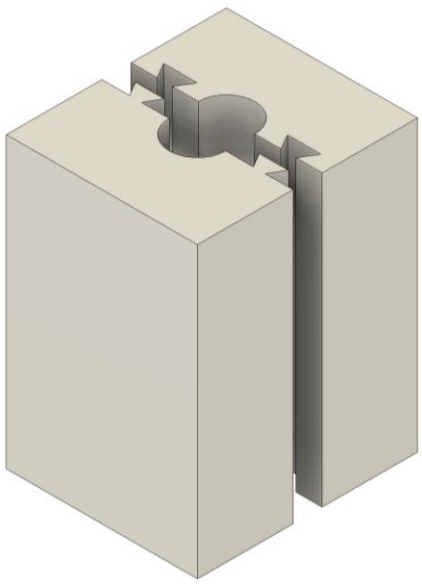


Figura 8: Diseño del soporte

4. METODOLOGÍA

4.1. Selección de los programas

Desde un principio sabíamos que para este proyecto íbamos a usar la aplicación de Arduino para ejercer el control sobre los componentes electrónicos que integran la CNC. Pero esto tiene un inconveniente muy importante y es que la comunicación directa entre el usuario final y Arduino es imposible siendo casi siempre necesario otros programas intermedios que sean capaces de realizar dicha comunicación.

Es por esto mismo que debíamos de elegir un segundo lenguaje de programación para llevar a cabo esta comunicación intermedia entre el usuario y el microchip, y para esto surgieron tres principales opciones: App Inventor, Matlab y Processing.

En un inicio se pensó en utilizar el programa desarrollado por el MIT, App Inventor, que, mediante programación con bloques, nos permite desarrollar aplicaciones móviles de bastante complejidad de una manera fácil y accesible. Sin embargo, este tipo de programación planteó varios problemas. El primero de ellos es que la potencia con la que dispone un teléfono móvil es mucho menor que la de cualquier ordenador de hoy en día y, considerando la relativa facilidad de acceso hoy en día a un PC se estimó que usar como soporte un teléfono no era conveniente. De manera adicional, el acceso y la manipulación de archivos dentro de un móvil es bastante más compleja que la de un ordenador y puesto que durante la ejecución del programa se necesitarán crear y leer archivos hacía todavía más complicado usar esta aplicación.

Las otras dos opciones son muy similares. Ambas se ejecutan desde el ordenador, con las dos se pueden crear programas muy complejos y las dos pueden crear una interfaz de usuario para que su uso sea sencillo. Sin embargo, hay un detalle clave que las diferencia, y este es el lenguaje en el que se basa cada una de estas dos aplicaciones. Tanto Processing como Arduino están basados en lenguajes de programación orientada a objetos o POO (de hecho, las interfaces de usuario de los dos programas guardan muchas similitudes). Por esta razón la comunicación entre estos dos lenguajes en relativamente sencilla, siendo esta el motivo por el cual se decidió elegir Processing como lenguaje para que el usuario se comunique con la máquina.

4.2. Arduino

Como ya hemos comentado anteriormente, la aplicación de Arduino es una plataforma de hardware libre para la programación de sus propios microcontroladores. El lenguaje de programación que usa es Wiring, muy similar a C o C++. Además, el IDE (Integrated Development Environment) proporciona una interfaz amigable para el usuario, facilitando la gestión de librerías, datos, estructuras, etc.

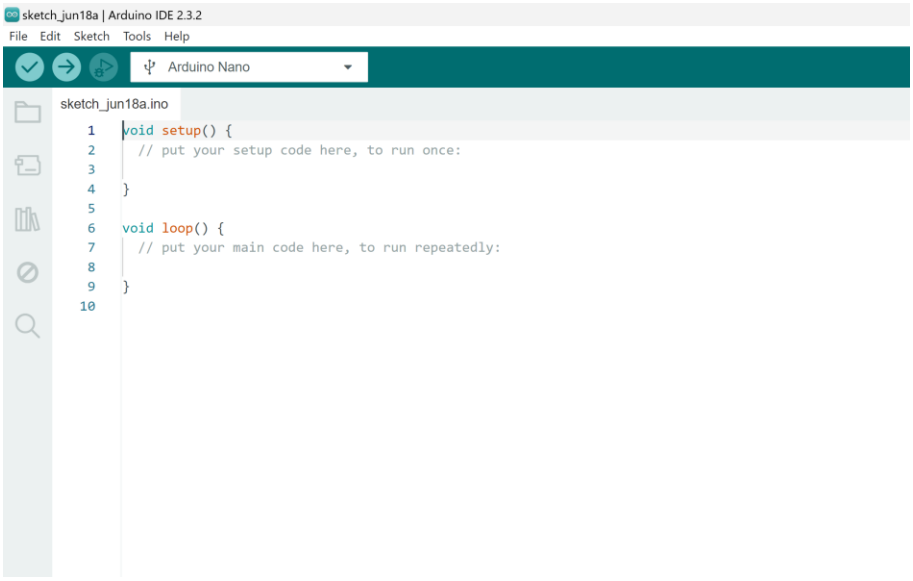


Figura 9: Interfaz de usuario Arduino

Los programas desarrollados en Arduino tienen fundamentalmente dos estructuras. La primera es el *setup* que se ejecuta una sola vez al iniciar el código y en segundo lugar el *loop*, que se ejecuta de manera recurrente tantas veces como el usuario quiera.

4.2.1. Librerías

Para hacer más sencilla la aplicación de códigos los usuarios pueden crear librerías de funciones específicas para distintos usos. Hay librerías de todo tipo, desde las que incluyen más funciones matemáticas, opciones para controlar sensores de todo tipo o, las que nos interesan en nuestro caso, librerías que nos facilitan la operación de los motores paso a paso.

En este proyecto se han usado dos librerías “AccelStepper”, creada por Mike McCauley (2010-2021) y “MultiStepper” del mismo autor. La primera de ella está enfocada a hacer más flexible la utilización de motores paso a paso ofreciendo también mucha más precisión de la que se puede conseguir usando los recursos base que nos ofrece Arduino y la segunda nos permite operar con hasta 10 motores simultáneamente. En conjunto estas librerías nos permitirán controlar los 3 motores de una manera infinitamente más simple.

4.3. Processing

El sistema de programación Processing es una plataforma flexible diseñada para la creación de proyectos interactivos y el diseño gráfico. Originado en el MIT, es un entorno basado en el lenguaje de programación Java. Gracias a su sintaxis simplificada y su enfoque visual permiten a los usuarios desarrollar aplicaciones complejas.

Otra de los grandes atractivos de Processing es la variedad de bibliotecas que han ido desarrollando los usuarios, quienes también proporcionan ejemplos y ayuda a los nuevos usuarios. También cabe destacar que debido a que es un lenguaje orientado al diseño gráfico existen gran cantidad de funciones que nos van a permitir procesar imágenes, que es precisamente lo que estamos tratando de hacer.

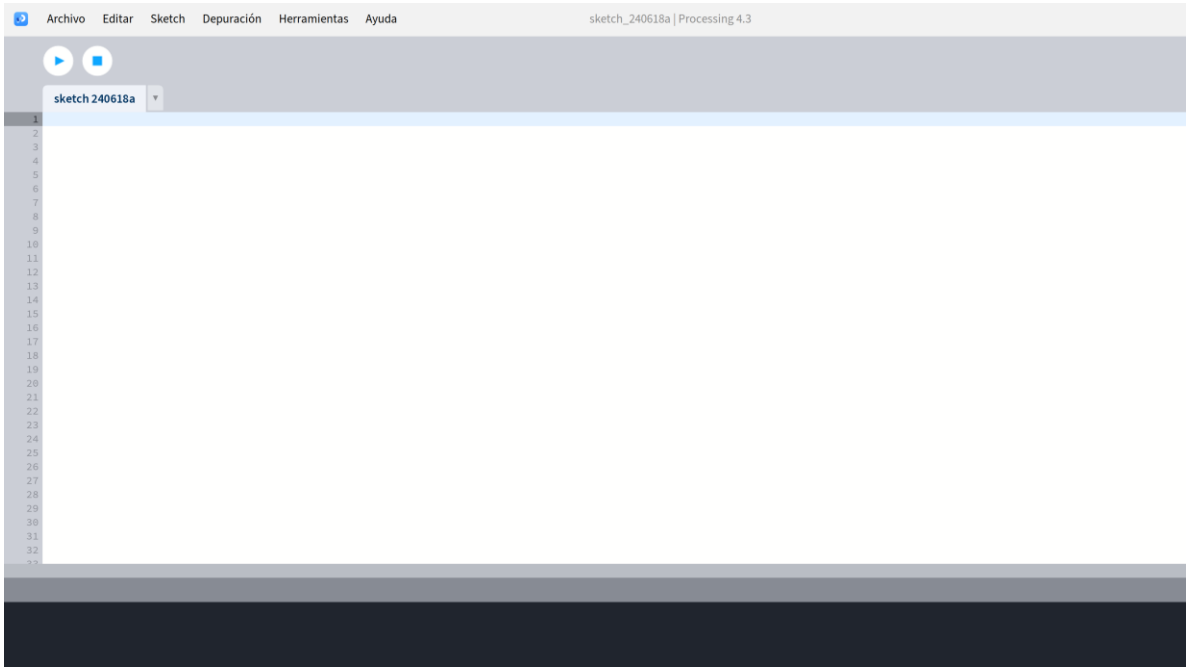


Figura 10: Interfaz de usuario Processing

4.3.1. Librerías

Al igual que en Arduino hay ciertas tareas que serían imposibles hacerlas sin el uso de librerías, es por eso por lo que en los programas que se han desarrollado se han incluido las siguientes librerías.

En total se han usado tres librerías "controlP5", de Andreas Schlegel, con la que se facilita la creación de interfaces gráficas, "java.io.File", desarrollada por Sun Microsystems, que es una biblioteca estándar de Java que tiene como objetivo proporcionar herramientas para la manipulación de archivos y el flujo de datos. Por último "processing.serial", que fue creada por el equipo principal de la plataforma Processing y que proporciona una forma fácil de comunicarse con dispositivos a través de puertos serie, en nuestro caso la usamos para comunicarnos con Arduino.

5. DESCRIPCIÓN DE LA SOLUCIÓN

5.1. Programa de control directo

Como primer paso para controlar la máquina de control numérico debemos de idear un programa que sea capaz de mover la herramienta en los 3 ejes. Esto por dos motivos, primeramente, ya que debido a que no tenemos ningún sensor de posición instalado en el sistema la mejor alternativa para controlar la posición inicial es que lo haga directamente el usuario. En segundo lugar, si en algún momento el usuario final requiere situar la herramienta en algún punto en concreto la mejor forma para hacerlo será a través de este programa.

Para comprender mejor la estructura de los programas y su funcionamiento primero se mostrará el código de Processing, ya que es quien genera la información y posteriormente veremos qué hace Arduino para interpretar dicha información. Para más detalles de estos o de los siguientes códigos véase el anexo correspondiente.

5.1.1. Control directo con Processing

Este programa lo único que hará es generar dos botones por dirección, uno para hacer un paso grande en un sentido del eje y otro para hacer un paso pequeño, esto multiplicado por 3 ejes y 2 sentidos por eje tenemos un total de 12 botones, y, por tanto, 12 señales distintas que se deben de enviar al Arduino. Además, se ha añadido un botón extra que hace que el punto en el que está se convierta en el origen de coordenadas, esto para localizar el punto desde es que comenzará a dibujar en el siguiente programa.

```
    cp5.addButton("masX")
    .setLabel("+X")
    .setPosition(centerX, centerY-btnSize-120)
    .setSize(btnSize,btnSize)
    .getCaptionLabel()
    .setFont(fuente);
    }

    void masX(){
  myPort.write("W");
}
```

Figura 11: Detalle del programa

Como podemos apreciar en la *Figura 11*, cada botón generará una letra distinta que enviará al Arduino. Finalmente, y tras añadir todos los botones la interfaz que verá el usuario quedará de la siguiente manera:

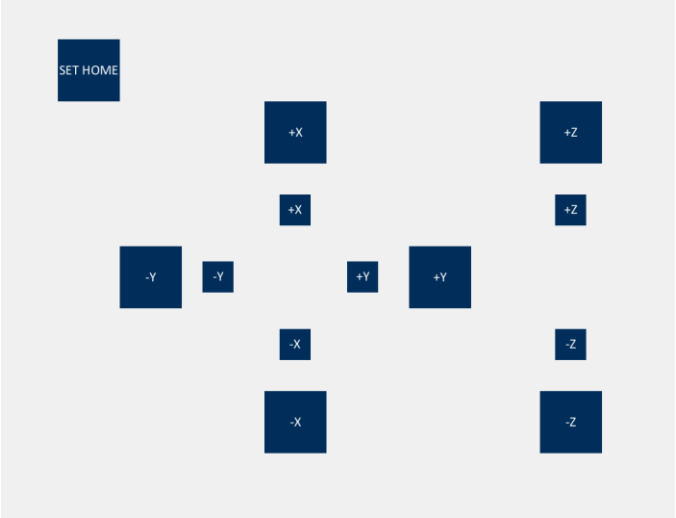


Figura 12: Interfaz de usuario del programa de control directo

5.1.2. Control directo con Arduino

Para el programa de Arduino tendremos que hacer que reconozca cada uno de los 13 códigos que se le pueden enviar y que asocie cada uno de estos a una acción. Para ello se ha usado una estructura de *switch-case* de la siguiente manera:

```
void loop() {
  if(Serial.available()){
   line = Serial.read();
    switch(line){
    case 'W':
      stepperX.move(step);
      break;
```

Figura 13: Detalle del programa

De tal manera que, para el caso que enseñamos antes donde se enviaba una letra "W", aquí se interpreta y se le manda avanzar un número de pasos concretos en dirección positiva. Si quisiéramos hacer un paso pequeño se divide entre 4 el número de pasos que se dan y si queremos avanzar en el otro sentido del eje simplemente se le cambia el signo a la orden. Para el botón de "*Set Home*" se le indicará que localice dicha posición como el punto 0 en los tres ejes, o lo que es lo mismo, que marque ese punto como origen de coordenadas.

5.2. Programa de control automático

5.2.1. Control automático con Processing

Este programa deberá hacer una serie de pasos para finalmente conseguir que el Arduino, y por extensión, los motores, reciban la información de dónde deben de posicionarse en cada momento. El primero de ellos será el de dejar al usuario elegir una foto. Una vez la ubicación de dicha imagen haya sido determinada el programa convertirá todos los píxeles a blanco o negro según su luminosidad. El umbral (o como se llama en el programa *threshold*) que usa Processing para determinar si un píxel debe de ser blanco o negro lo seleccionará el usuario mediante una deslizadera.

Figura 14: Interfaz tras ejecutar el programa

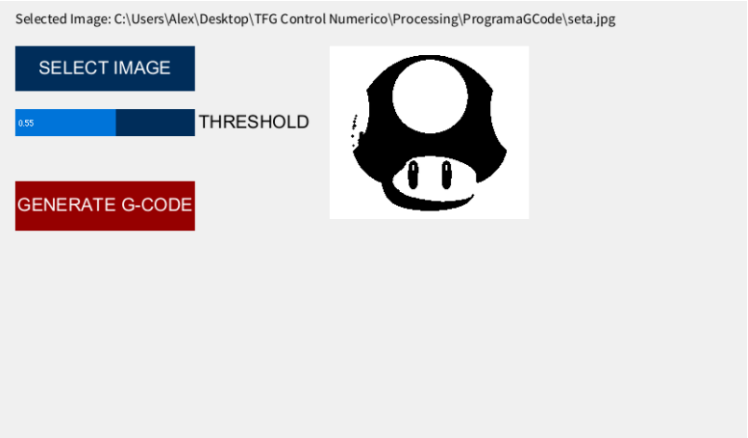


Figura 15: Interfaz una vez seleccionada la fotografía

Como se puede observar en la *Figura 15: Interfaz una vez seleccionada la fotografía,* se ha elegido un umbral del 55%, si el usuario lo aumentase a, digamos, el 70% habría más píxeles negros y menos blancos. Este paso hay que hacerlo con precaución puesto que es necesario que los píxeles definan una imagen lo más nítida posible. En la *Figura 16* se ha elegido un límite demasiado bajo y como resultado el dibujo no será representativo de la imagen escogida. Asimismo, se indica en la parte superior el directorio donde se encuentra ubicado el archivo de imagen que se ha elegido.

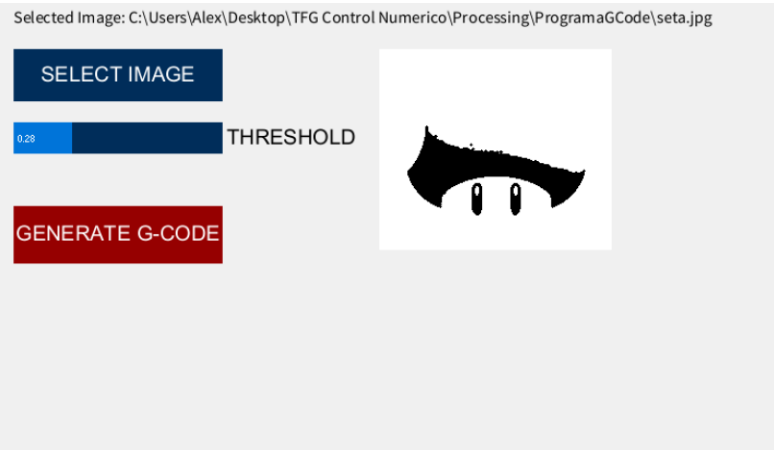


Figura 16: Threshold demasiado bajo

El siguiente paso es sencillo para el usuario, solo tendrá que pulsar el botón rojo, "*GENERATE G-CODE*". Internamente Processing repasará todos los píxeles de la imagen que aparece en pantalla y creará un archivo de texto en la que escribirán todos los comandos que posteriormente deberá seguir la máquina. Si un pixel es negro bajará el bolígrafo y avanzará al siguiente píxel negro con marcando todo el camino, si por el contrario el píxel es blanco, se subirá el bolígrafo y avanzará hasta el próximo píxel negro para posteriormente, volver a bajarlo y seguir dibujando. El documento de texto con todas las instrucciones queda guardado en la misma carpeta donde esté el programa, con el nombre de *output.txt.*

```
G1 X77 Y53
G1 X76 Y56
G1 X75 Y58
G1 X74 Y59
G1 X73 Y60
G1 Z5.0
G1 X30 Y64
G1 Z0.0
G1 X29 Y63
G1 X28 Y62
```

Figura 17: Sección del código generado

Para esto se tomó la decisión de que los píxeles se analizarían de manera circular concéntrica, porque en la mayoría de los casos, resultaba en trazos que amoldaban mejor a la foto original. Sumado a esto se puede decir que el resultado es más original y le da cierta personalidad al dibujo.

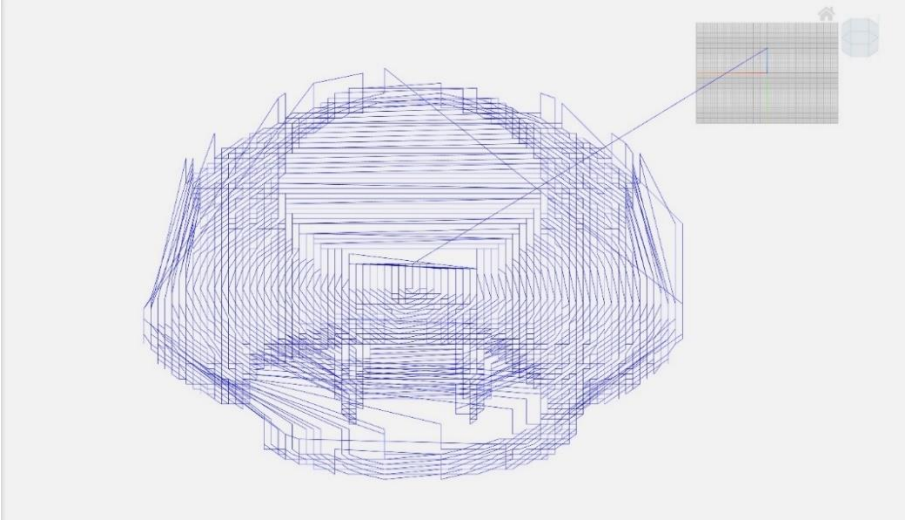


Figura 18: Vista previa de la rutina generada

Fuente: NC viewer

En la *Figura 18*, se puede apreciar una previsualización de cuáles serán los movimientos que debe de hacer la máquina, las líneas bajas son aquellas que serán plasmadas en el papel, el resto de las líneas son tan solo recorrido que hace la máquina de un punto de dibujo hasta el siguiente.

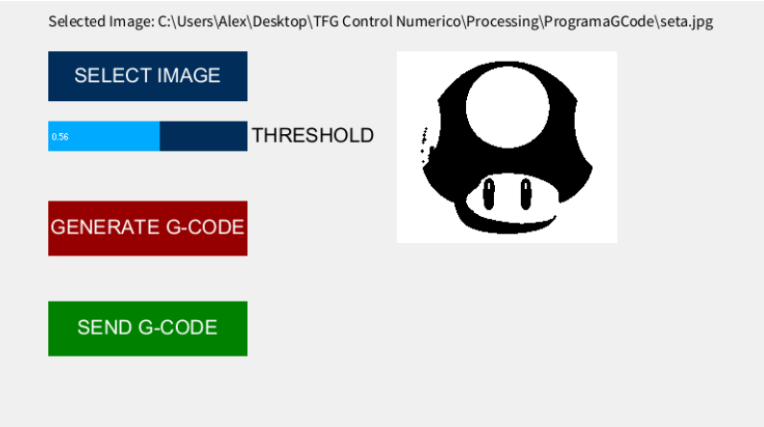


Figura 19: Programa listo para enviar

Para finalizar con este programa, y una vez creado el documento con la rutina, aparecerá por pantalla un tercer botón, de color verde, cuya función será la de iniciar la puesta en contacto con el Arduino para que se pueda enviar el código que se acaba de generar. Para ello se cargarán todas las líneas de código que se hayan creado en el paso anterior y se irán enviando poco a poco a la máquina. Cuando esta haya acabado una orden, se enviará la siguiente. Cuando llegue al final se volverá al punto X0 Y0 con el bolígrafo levantado y el programa finalizará.

Para que el usuario sepa que porcentaje del programa se ha completado se imprimirá por pantalla tanto la orden que le ha llegado a Arduino como el porcentaje de líneas que ya han sido enviadas y ejecutadas.

Para más detalle del programa consultar el Anexo 7.6 *Programa control automático Processing*.

5.2.2. Control automático con Arduino

El programa de Arduino no actúa hasta que no se ha pulsado el último botón del programa en Processing. Una vez están en contacto Arduino comienza a recibir instrucciones. Cuando llega una de estas instrucciones el primer paso que realiza es el de leer el código que identifica que tipo de función es. Puede ser o bien G1, si se trata de una función de movimiento, o bien M02 si es la función que indica el final del programa. Una vez que conoce de que tipo de función se trata se parte el código en trozos. Lo que sigue a la letra X se guarda en un *string*, lo que sigue a la Y en otro y en Z se procede de la misma manera. Posteriormente estos *strings* se pasan a decimal y se le aplica el cambio de mm a pasos (consultar el anexo *7.2* Cálculo de los pasos por vuelta*)* y finalmente estos valores se les remiten a los motores para que estos se muevan a las coordenadas indicadas.

Cada vez que se completa una línea de código el programa envía una letra a Processing que marca que ya ha terminado una función y está listo para recibir la siguiente.

5.2.3. Comunicación entre programas

Uno de los grandes retos de este proyecto es conseguir que los dos procesadores (el ordenador y Arduino) se comuniquen de manera efectiva, y es que, a diferencia del programa de control directo, en este se debe de establecer una comunicación bidireccional en la que uno espere a que el otro esté preparado para recibir para que el otro empiece a enviar.

Para conseguir esto se procede de la siguiente manera:

Al iniciar el programa de Arduino este enviará la letra A una vez por segundo, una vez que Processing reciba un mensaje con la letra A, sabrá que ha establecido contacto, pero para que Arduino se ponga a la espera de mensajes Processing enviará de vuelta otra letra A. Tras esto Processing enviará una línea de código, pero no enviará la siguiente hasta que Arduino no haya devuelto una letra G, marcando que ya ha completado la tarea y está listo para iniciar la siguiente.

```
void SendGCode(){
  String[] Outstring = loadStrings(outputFile);
  while(myPort != null && myPort.active()){
   if (i<=Outstring.length){
    Instring = myPort.readStringUntil('\n');
      if (Instring != null) {
        Instring = trim(Instring);
        println(Instring);
        if (firstContact == false) {
          if (Instring.equals("A")) {
          myPort.clear();
          firstContact = true;
          myPort.write("A");
          println("contact");
          }
        } else{
            if(Instring.equals("G")){
            myPort.write(Outstring[i]);
            println(i*100/Outstring.length + "%");
            i++;
          }
        }
      }
    }
  }
}
```

Figura 20: Bucle de envío de información Processing.

```
void establishContact() {
  while (Serial.available() <= 0) {
  Serial.println("A");   // Envía el mensaje de conexión
  delay(1000);
  }
}
```

Figura 21: Establecimiento de contacto Arduino

```
void loop() {
  if (Serial.available() > 0) {
    command = Serial.readStringUntil('\n');
    command.trim();
    Serial.println("Arduino recieved: " + command);
    processCommand(command);
    steppersControl.moveTo(gotopos);
    steppersControl.runSpeedToPosition();
    Serial.println("G");
  }
}
```

Figura 22: Bucle de recepción Arduino

6. RESULTADOS

6.1. Fase previa al dibujo

Antes de comenzar con la impresión tenemos que asegurar una superficie plana y rígida sobre el soporte móvil de la máquina para poder pintar sobre ella. Para ello se utilizó una tabla de conglomerado , sujeta al soporte móvil con dos tornillos. Encima de la esta tabla se colocó un papel usando estos tornillos y unas pinzas para asegurar que el papel estuviera bien tensado.

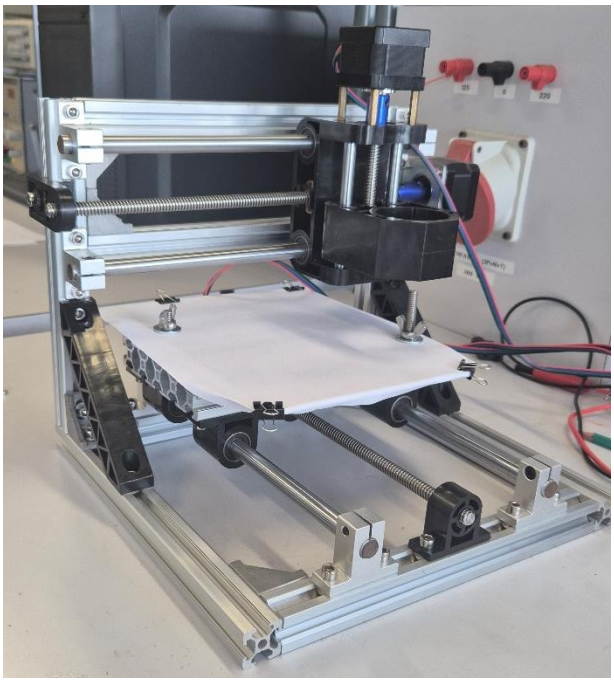


Figura 23: Montaje del lienzo

Por otro lado, para delimitar los márgenes de dibujo de la máquina se marcó un rectángulo, usando el programa de control directo que antes vimos, y como resultado se obtuvo un área útil de 105 mm de ancho por 90 mm de alto, aunque finalmente solo se usará un espacio de 100x85 mm por seguridad.

6.2. Dibujos obtenidos

Para poner a prueba el programa y el montaje primero se envió una foto del personaje de dibujos animados “Pluto”, ya que es un dibujo con colores sólidos y el programa pudo entender bien el delineado de la imagen.

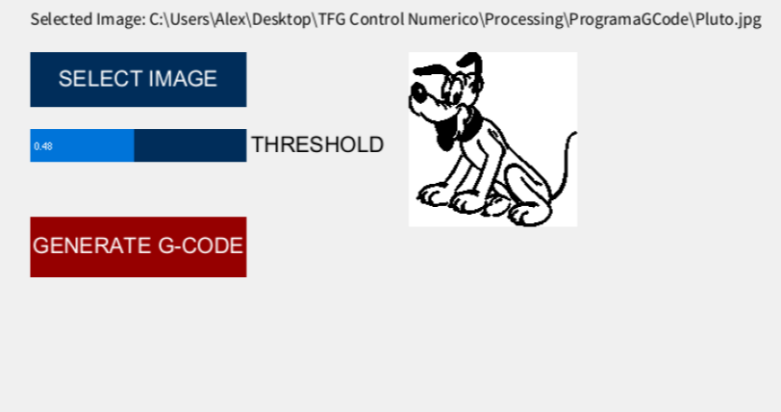


Figura 24: Vista previa de la imagen

Si bien se pudo dibujar algo, el problema de esta imagen fue que carecía del detalle suficiente, y las líneas que marcó la máquina estaban algo separadas y no se entendían. Como resultado se obtuvo una figura, que, aunque parecida, no era del todo reconocible. Aunque es cierto que al ser un dibujo que prácticamente todo es delineado es más difícil de llevar a cabo.

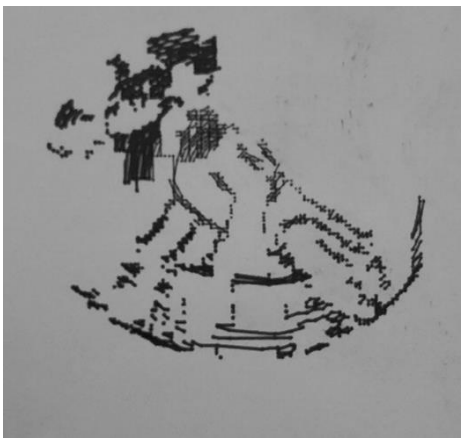


Figura 25: Resultado de la primera impresión

Tras este resultado, se decidió usar una foto con más resolución, y con colores con mayor contraste, de tal manera que hubiese partes completamente oscuras en lugar de solo trazos delimitando los bordes. Para ello se escogió la siguiente imagen.

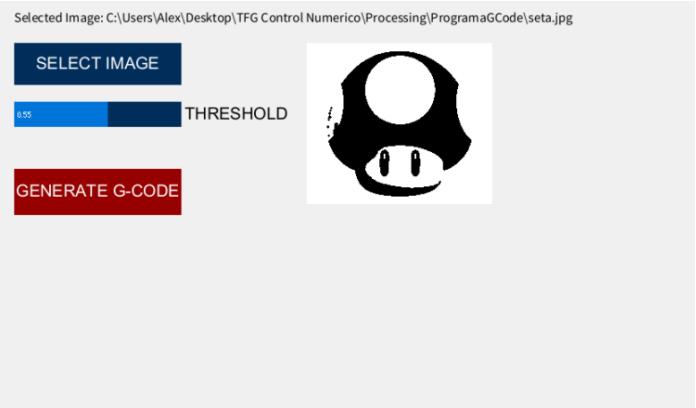


Figura 26: Vista previa de la segunda impresión

El resultado esta vez fue mucho más satisfactorio siendo este más reconocible y similar a la fotografía original.

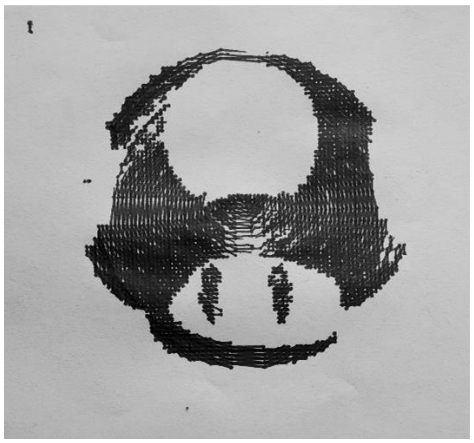


Figura 27: Resultado de la segunda impresión

6.3. Conclusiones y líneas futuras

Podemos afirmar que la máquina es capaz de dibujar aquello que se le envía y por lo tanto se han cumplido los objetivos que se fijaron en un inicio. Sin embargo, todavía se pueden mejorar muchos aspectos del montaje y del modo en el que se analiza e imprime la imagen.

Para comenzar, se debería corregir la planitud de la superficie sobre la que se dibuja ya que, como se puede apreciar en la *Figura 27*, hay partes donde el bolígrafo no llega a tocar el papel. Esto se podría realizar tomando medidas con un reloj comparador, nivelando lo máximo posible todos los puntos. Luego para asegurarse que la calibración es correcta se le puede pasar un programa que vaya tocando el papel en diferentes puntos. Si hubiese un lugar donde no dibujase se debería elevar ese lado. Este problema es, de hecho, muy común en máquinas parecidas como las impresoras 3D, la diferencia es que estas tienen sistemas que le facilitan al usuario esta nivelación.

Para continuar, actualmente solo se pueden dibujar fotografías con alto contraste, como personajes de dibujos animados y otras ilustraciones, ya que son las que mejor se convierten a blanco y negro. Se podría modificar el programa para que, en lugar de que analice si el píxel es blanco o negro, guarde su nivel de luminosidad, y, a la hora de dibujar, que apriete más o menos el bolígrafo en función de este valor. Como

inconveniente de esto es que hay que hacer una muy buena calibración de la planitud de la superficie y no habría mucha variedad de grises, ya que tampoco se puede apretar mucho con este tipo de marcadores, ya que la punta podría dañarse.

Para finalizar, la manera en la que recorre la imagen al dibujar es muy ineficiente, teniendo que hacer muchos recorridos en "vacío". Esto se podría arreglar haciendo que el programa evite los píxeles blancos, solo desplazándose a través de píxeles negros que todavía no hayan sido trazados. Cuando esto no se pueda cumplir se empezarán a pasar por encima de píxeles blancos. Esto puede hacer que se reduzcan en gran medida el número total de movimientos y, por tanto, el tiempo que tarda la máquina en dibujar.

7. BIBLIOGRAFÍA

1. *A4988 Datasheet*. (n.d.). Retrieved June 27, 2024, from http://pdf1.alldatasheet.com/datasheet-pdf/view/338780/ALLEGRO/A4988.html
2. *CNC Machines Complete Guide - Evolution and Industry 4.0*. (2018, August 13). Engineering Technology Group - ETG. https://engtechgroup.com/cnc-machines-evolution/
3. Criado, J. (2023). *Control por computador de una maquina fresadora cnc y desarrollo de una aplicación móvil para controlarla.* [TFG]. UPV.
4. E. Weisberg, D. (2010, July 7). *Computer-Aided Design's Strong Roots at MIT*. https://web.archive.org/web/20100707074750/http://www.cadhistory.net/chapters/03_MIT_CAD_Roots_1945_1965.pdf
5. McCauley, M. (n.d.). *AccelStepper: AccelStepper library for Arduino*. Retrieved June 27, 2024, from http://www.airspayce.com/mikem/arduino/AccelStepper/index.html
6. *NEMA17 Datasheet*. (n.d.). Retrieved June 27, 2024, from http://pdf1.alldatasheet.com/datasheet-pdf/view/1572377/ETC/NEMA17.html
7. Schlegel, A. (n.d.). *controlP5*. Retrieved June 27, 2024, from https://sojamo.de/libraries/controlP5/

7. ANEXOS

7.1. Objetivos de desarrollo sostenible y Agenda 2030

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la Pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**		X		
ODS 5. **Igualdad de Género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico**.		X		
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo sostenibles.**	X			
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia, e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Figura 28: Tabla ODS

Las máquinas CNC tienen, sin lugar a duda, un impacto positivo en el desarrollo de varios ODS, en especial los que tienen relación con la innovación industrial y eficiencia de recursos. La integración de estas tecnologías en la manufactura moderna no solo mejora la productividad y calidad, sino que incluso promueve prácticas más responsables y sostenibles en línea con los objetivos globales de desarrollo.

Con respecto al *ODS 4 Educación de calidad*, este es un proyecto con el que se pretende aplicar de manera amigable el funcionamiento de una máquina de control numérico convencional, de tal manera que puede ser usado como recurso práctico en las aulas. Se puede usar incluso como proyecto para los alumnos en la programación con Arduino.

En relación con los *ODS 8 Trabajo decente y crecimiento económico* y el *ODS 9 Industria, innovación e infraestructura* se puede decir que, como cualquier proyecto de innovación el impacto sobre la industria será positivo creando nuevos métodos, que repercutirán positivamente creando tanto nuevo tejido industrial como puestos de trabajo cualificados y dignos.

En relación con el ODS 12 Producción y consumo sostenibles podemos afirmar que gracias a las CNC se reducen considerablemente los desperdicios, ya que se reducen casi al mínimo cualquier error en la producción. De esta manera se hace también un uso más eficiente de los recursos y además en muchos casos los desperdicios de una CNC como son las virutas de material se pueden reciclar, reduciendo así el impacto ambiental.

7.2. Cálculo de los pasos por vuelta

Al establecer la comunicación con los motores se les debe de indicar el número de pasos que deseas avancen, pero nosotros al analizar la imagen que queremos imprimir obtenemos coordenadas los ejes X, Y y Z en milímetros, por lo tanto, para que los motores nos entiendan debemos establecer una relación entre milímetros y pasos del motor.

Lo primero que debemos saber es cuántos pasos hay en una vuelta de motor, y como antes hemos comentado, este valor es de 200 pasos. En segundo lugar, sabemos que siempre que nos movemos lo hacemos en pasos enteros por lo tanto esta cifra de 200 pasos por revolución en este caso se mantiene invariable.

En tercer lugar, y para acabar de caracterizar esta relación, nos tenemos que fijar en los tornillos guía que transforman el movimiento circular de los motores en movimientos lineales en los 3 ejes. Estos tornillos están definidos por dos valores, el número de hilos y el paso, que se relacionan de la siguiente manera:

$$D = N \times \mathrm{P} \tag{3}$$

Siendo *P* el paso, *N* el número de comienzos o hilos y *D* la distancia recorrida por vuelta

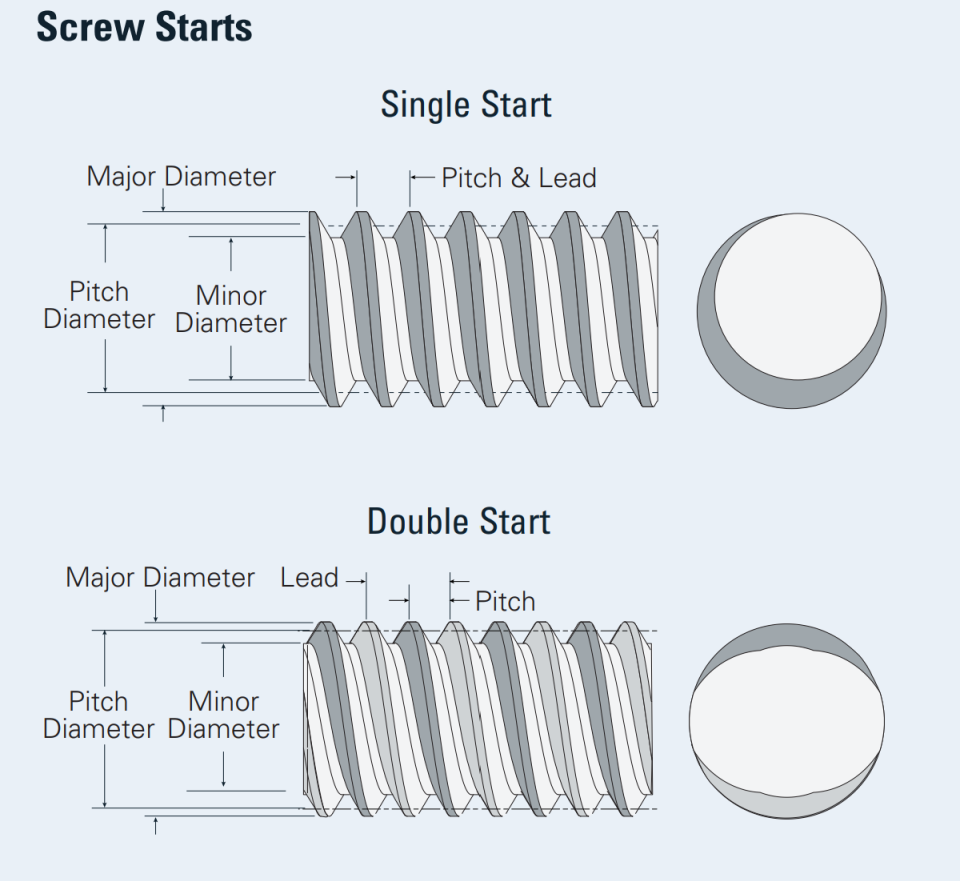


Figura 29: Ejemplo de un comienzo y dos comienzos

Para determinar cuántos hilos tiene nuestro tornillo nos debemos de fijar en su punta. En nuestro caso y como se puede observar en la *Figura 30* el tornillo tiene 2 comienzos, ya que se pueden observar dos marcas en su extremo, además la punta tiene forma ovalada, al igual que el segundo tornillo de la *Figura 29*.

Figura 30: Detalle comienzo del tornillo

Para obtener el paso de nuestro tornillo se hizo de la siguiente manera: se midieron la distancia entre dos crestas no contiguas y posteriormente, se contaron el número de valles que separaban dichas crestas, tal y como se muestra en la siguiente imagen.

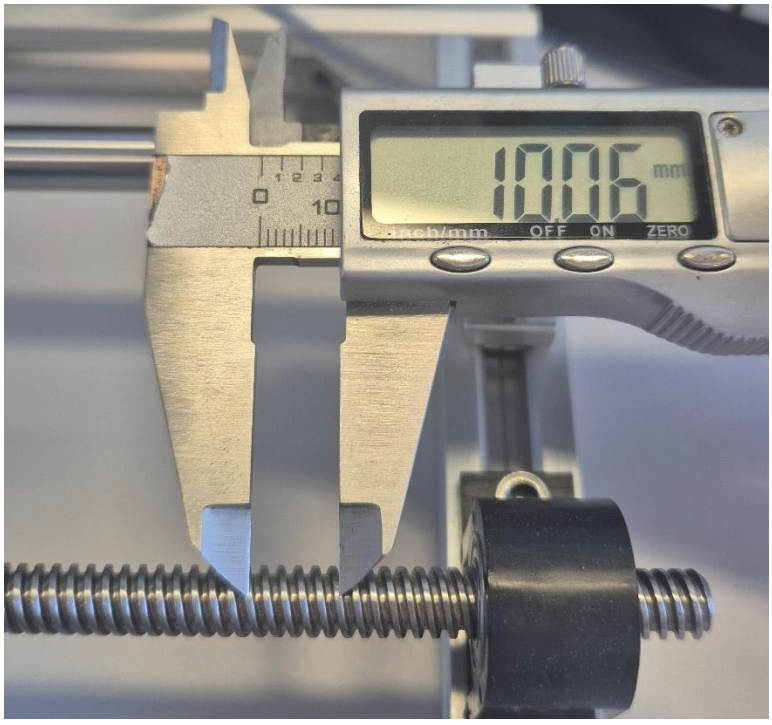


Figura 31: Medida del paso

Como podemos ver en la *Figura 31: Medida del paso*, el número de valles entre las dos crestas que medimos es de 5. Conociendo esto y aplicando la fórmula obtenemos:

$$P = \frac{10{,}06}{5} \approx 2\ mm \tag{4}$$

Para finalizar y conociendo los valores de *N* y *P*:

$$D = 2 * 2 = 4mm/rev \tag{5}$$

Como hemos expuesto al principio la relación que nos interesa es los pasos que se deben de realizar por milímetro, para ello simplemente operamos

$$\frac{pasos}{mm} = \frac{200}{4} = 50\ pasos/mm \tag{6}$$

Para finalizar este apartado en la siguiente figura se muestra la función de Arduino a la que se llama siempre que es necesario pasar de mm a pasos.

```
int mm2steps(float mm){
  return(round(mm*50));
}
```

Figura 32: Función mm a pasos

7.3. Programa de prueba de motores

```
const int STEP=2;
const int DIR=A2;
int StepsperRev=200;
int i=0;
void setup() {
  pinMode(STEP,OUTPUT);
  pinMode(DIR,OUTPUT);
}

void loop() {
  digitalWrite(DIR,HIGH);
  for(i=0;i<StepsperRev;i++){
    digitalWrite(STEP,HIGH);
    delay(2);
    digitalWrite(STEP,LOW);
    delay(2);
  }
  delay(2000);
  digitalWrite(DIR,LOW);
  for(i=0;i<StepsperRev;i++){
    digitalWrite(STEP,HIGH);
    delay(2);
    digitalWrite(STEP,LOW);
    delay(2);
  }
  delay(2000);
}
```

7.4. Programa control directo Processing

```

  import controlP5.*;
  import processing.serial.*;

  ControlP5 cp5;
  Serial myPort;
  PFont fuente;

void setup(){
      size(1200, 800);
      cp5 = new ControlP5(this);
      fuente = createFont("Calibri", 20);
      String portName = Serial.list()[1];
      myPort = new Serial(this, portName, 115200);

      int btnSize = 90;
```

```
17.       int centerX = width / 2;
18.       int centerY = height / 2;
19.
20.         cp5.addButton("masX")
21.         .setLabel("+X")
22.         .setPosition(centerX, centerY-btnSize-120)
23.         .setSize(btnSize,btnSize)
24.         .getCaptionLabel()
25.         .setFont(fuente);
26.
27.          cp5.addButton("smallmasX")
28.         .setLabel("+x")
29.         .setPosition(centerX+btnSize/4, centerY-btnSize/2-30)
30.         .setSize(btnSize/2,btnSize/2)
31.         .getCaptionLabel()
32.         .setFont(fuente);
33.
34.       cp5.addButton("menosX")
35.       .setLabel("-X")
36.       .setPosition(centerX, centerY+btnSize+120)
37.       .setSize(btnSize,btnSize)
38.       .getCaptionLabel()
39.       .setFont(fuente);
40.
41.        cp5.addButton("smallmenosX")
42.       .setLabel("-x")
43.       .setPosition(centerX+btnSize/4, centerY+btnSize+30)
44.       .setSize(btnSize/2,btnSize/2)
45.       .getCaptionLabel()
46.       .setFont(fuente);
47.
48.         cp5.addButton("masY")
49.         .setLabel("+Y")
50.         .setPosition(centerX+btnSize+120, centerY)
51.         .setSize(btnSize,btnSize)
52.         .getCaptionLabel()
53.         .setFont(fuente);
54.
55.          cp5.addButton("smallmasY")
56.         .setLabel("+y")
57.         .setPosition(centerX+btnSize+30, centerY+btnSize/4)
58.         .setSize(btnSize/2,btnSize/2)
59.         .getCaptionLabel()
60.         .setFont(fuente);
61.
62.
63.       cp5.addButton("menosY")
64.       .setLabel("-Y")
65.       .setPosition(centerX-btnSize-120, centerY)
66.       .setSize(btnSize,btnSize)
67.       .getCaptionLabel()
68.       .setFont(fuente);
69.
70.        cp5.addButton("smallmenosY")
71.       .setLabel("-y")
72.       .setPosition(centerX-btnSize, centerY+btnSize/4)
```

```
73.       .setSize(btnSize/2,btnSize/2)
74.       .getCaptionLabel()
75.       .setFont(fuente);
76.
77.         cp5.addButton("masZ")
78.         .setLabel("+Z")
79.         .setPosition(centerX+400, centerY-btnSize-120)
80.         .setSize(btnSize,btnSize)
81.         .getCaptionLabel()
82.         .setFont(fuente);
83.
84.         cp5.addButton("smallmasZ")
85.         .setLabel("+z")
86.         .setPosition(centerX+btnSize/4+400, centerY-btnSize/2-30)
87.         .setSize(btnSize/2,btnSize/2)
88.         .getCaptionLabel()
89.         .setFont(fuente);
90.
91.       cp5.addButton("menosZ")
92.       .setLabel("-Z")
93.       .setPosition(centerX+400, centerY+btnSize+120)
94.       .setSize(btnSize,btnSize)
95.       .getCaptionLabel()
96.       .setFont(fuente);
97.
98.       cp5.addButton("smallmenosZ")
99.       .setLabel("-z")
100.      .setPosition(centerX+btnSize/4+400, centerY+btnSize+30)
101.      .setSize(btnSize/2,btnSize/2)
102.      .getCaptionLabel()
103.      .setFont(fuente);
104.
105.      cp5.addButton("SetHome")
106.      .setLabel("Set Home")
107.      .setPosition(centerX-300, centerY-300)
108.      .setSize(btnSize,btnSize)
109.      .getCaptionLabel()
110.      .setFont(fuente);
111.    }
112.
113.    void draw() {
114.      background(240);
115.    }
116.
117.    void masX(){
118.      myPort.write("W");
119.    }
120.    void smallmasX(){
121.      myPort.write("I");
122.    }
123.    void menosX(){
124.      myPort.write("S");
125.    }
126.    void smallmenosX(){
127.      myPort.write("K");
128.    }
```

```
129.    void masY(){
130.      myPort.write("D");
131.    }
132.    void smallmasY(){
133.      myPort.write("L");
134.    }
135.    void menosY(){
136.      myPort.write("A");
137.    }
138.    void smallmenosY(){
139.      myPort.write("J");
140.    }
141.    void masZ(){
142.      myPort.write("T");
143.    }
144.    void smallmasZ(){
145.      myPort.write("Y");
146.    }
147.    void menosZ(){
148.      myPort.write("G");
149.    }
150.    void smallmenosZ(){
151.      myPort.write("H");
152.    }
153.    void SetHome(){
154.      myPort.write("C");
155.    }
```

7.5. Programa control directo Arduino

```
1. #include <AccelStepper.h>
2.
3. #define STEPX 3
4. #define DIRX A0
5.
6. #define STEPY 2
7. #define DIRY A2
8.
9. #define STEPZ 11
10.#define DIRZ A1
11.
12.AccelStepper stepperX(1, STEPX, DIRX);
13.AccelStepper stepperY(1, STEPY, DIRY);
14.AccelStepper stepperZ(1, STEPZ, DIRZ);
15.
16.char line;
17.
18.int step=100;
19.
20.void setup() {
```

```
21.  Serial.begin(115200);
22.  stepperX.setMaxSpeed(1000.0);
23.  stepperX.setAcceleration(250.0);
24.
25.  stepperY.setMaxSpeed(1000.0);
26.  stepperY.setAcceleration(250.0);
27.
28.  stepperZ.setMaxSpeed(1000.0);
29.  stepperZ.setAcceleration(250.0);
30.
31.}
32.
33.void loop() {
34.  if(Serial.available()){
35.   line = Serial.read();
36.    switch(line){
37.    case 'W':
38.      stepperX.move(step);
39.      break;
40.    case 'I':
41.      stepperX.move(step/4);
42.      break;
43.    case 'S':
44.      stepperX.move(-step);
45.      break;
46.    case 'K':
47.      stepperX.move(-step/4);
48.      break;
49.    case 'D':
50.      stepperY.move(step);
51.      break;
52.    case 'L':
53.      stepperY.move(step/4);
54.      break;
55.    case 'A':
56.      stepperY.move(-step);
57.      break;
58.    case 'J':
59.      stepperY.move(-step/4);
60.      break;
61.    case 'T':
62.      stepperZ.move(step);
63.      break;
64.    case 'Y':
```

```
65.       stepperZ.move(step/4);
66.       break;
67.     case 'G':
68.       stepperZ.move(-step);
69.       break;
70.     case 'H':
71.       stepperZ.move(-step/4);
72.       break;
73.     case 'C':
74.       stepperX.setCurrentPosition(0);
75.       stepperY.setCurrentPosition(0);
76.       stepperZ.setCurrentPosition(0);
77.       break;
78.     }
79.   }
80.
81.   do{
82.     stepperX.run();
83.     stepperY.run();
84.     stepperZ.run();
85.   }while(stepperX.distanceToGo()!=0||stepperY.distanceToGo()!=0||stepperZ.dis
   tanceToGo()!=0);
86. }
```

7.6. Programa control automático Processing

```
1.
2. import controlP5.*;
3. import java.io.File;
4. import processing.serial.*;
5.
6. ControlP5 cp5;
7. PFont fuente;
8. PImage img;
9. PImage processedImg;
10.     int scale = 2; // Factor de escala para reducir la resolución de la
   imagen
11.     String outputFile = "output.txt";
12.     String imgPath = "";
13.     float thresholdValue = 0.5;
14.     boolean firstContact = false;
15.     String Instring;
16.     int i;
17.
18.     Serial myPort;
19.
20.     void setup() {
21.       //myPort = new Serial(this, Serial.list()[1], 115200); // Abrimos el
   COM5
```

```
22.       cp5 = new ControlP5(this);
23.       size(1000, 600);
24.       fuente = createFont("Arial", 20);
25.
26.       cp5.addButton("selectImage")
27.          .setLabel("Select Image")
28.          .setPosition(50, 50)
29.          .setSize(200, 50)
30.          .getCaptionLabel()
31.          .setFont(fuente);
32.
33.       cp5.addSlider("thresholdValue")
34.          .setLabel("Threshold")
35.          .setPosition(50, 120)
36.          .setSize(200, 30)
37.          .setRange(0, 1)
38.          .setValue(0.5)
39.          .setColorLabel(color(0))
40.          .getCaptionLabel()
41.          .setFont(fuente);
42.
43.
44.       cp5.addButton("createGCode")
45.          .setLabel("Generate G-Code")
46.          .setPosition(50, 200)
47.          .setSize(200, 55)
48.          .setColorBackground(color(150, 0, 0))
49.          .hide()
50.          .getCaptionLabel()
51.          .setFont(fuente);
52.
53.       cp5.addButton("SendGCode")
54.          .setLabel("Send G-Code")
55.          .setPosition(50, 300)
56.          .setSize(200, 55)
57.          .setColorBackground(color(0, 130, 0))
58.          .hide()
59.          .getCaptionLabel()
60.          .setFont(fuente);
61.
62.       textSize(16);
63.     }
64.
65.     void draw() {
66.       background(240);
67.
68.       if (processedImg != null) {
69.         image(processedImg, 400, 50, processedImg.width,
   processedImg.height);
70.       }
71.
72.       if (!imgPath.equals("")) {
73.         fill(0);
74.         text("Selected Image: " + imgPath, 50, 25);
75.       }
76.     }
```

```
77.
78.     void selectImage() {
79.       selectInput("Select an image to process:", "fileSelected");
80.     }
81.
82.     void fileSelected(File selection) {
83.       if (selection == null) {
84.         println("Window was closed or the user hit cancel.");
85.       } else {
86.         imgPath = selection.getAbsolutePath();
87.         img = loadImage(imgPath);
88.         img.resize(img.width / scale, img.height / scale);
89.         processImage();
90.         cp5.getController("createGCode").show();
91.       }
92.     }
93.
94.     void processImage() {
95.       if (img != null) {
96.         processedImg = img.copy();
97.         processedImg.filter(GRAY);
98.         processedImg.filter(THRESHOLD, thresholdValue);
99.         processedImg.filter(ERODE);
100.      }
101.    }
102.
103.    void controlEvent(ControlEvent theEvent) {
104.      if (theEvent.isFrom("thresholdValue")) {
105.        thresholdValue = theEvent.getValue();
106.        processImage();
107.      }
108.    }
109.
110.    void createGCode() {
111.      if (processedImg == null) {
112.        println("No image loaded.");
113.        return;
114.      }
115.
116.      StringBuilder gcode = new StringBuilder();
117.      gcode.append("G1 Z5.0 \n");
118.
119.      int centerX = processedImg.width / 2;
120.      int centerY = processedImg.height / 2;
121.      boolean penDown = false;
122.      int prevX = centerX;
123.      int prevY = centerY;
124.
125.      for (float r = 0; r < min(centerX, centerY); r += 1) {
126.        for (float angle = 0; angle < TWO_PI; angle += 0.01) {
127.          int x = round(centerX + r * cos(angle));
128.          int y = round(centerY + r * sin(angle));
129.
130.          if (x >= 0 && x < processedImg.width && y >= 0 && y <
    processedImg.height) {
131.            int pixel = processedImg.get(x, y);
```

```
132.            x=(round(map(x,0,processedImg.width,0, 100)));
133.            y=(round(map(y,0,processedImg.height,0, 85 )));
134.            if (x != prevX && y != prevY) {
135.              if (brightness(pixel) == 0) { // Si el píxel es negro, está
   en el borde
136.                gcode.append("G1 X" + x + " Y" + y + "\n");
137.                if (!penDown) {
138.                  gcode.append("G1 Z0.0 \n");
139.                  penDown = true;
140.                }
141.              } else {
142.                if (penDown) {
143.                  gcode.append("G1 Z5.0 \n");
144.                  penDown = false;
145.                }
146.              }
147.              prevX = x;
148.              prevY = y;
149.            }
150.          }
151.        }
152.      }
153.
154.      gcode.append("G1 Z5.0 F500 \n");
155.      gcode.append("G1 X0 Y0\n");
156.      gcode.append("M02\n");
157.      saveStrings(outputFile, gcode.toString().split("\n"));
158.      println("G-code saved to " + outputFile);
159.      cp5.getController("SendGCode").show();
160.    }
161.
162.    void SendGCode(){
163.      String[] Outstring = loadStrings(outputFile);
164.      while(myPort != null && myPort.active()){
165.       if (i<=Outstring.length){
166.        Instring = myPort.readStringUntil(&apos;\n&apos;);
167.          if (Instring != null) {
168.            Instring = trim(Instring);
169.            println(Instring);
170.            if (firstContact == false) {
171.              if (Instring.equals("A")) {
172.              myPort.clear();
173.              firstContact = true;
174.              myPort.write("A");
175.              println("contact");
176.              }
177.            } else{
178.                if(Instring.equals("G")){
179.                myPort.write(Outstring[i]);
180.                println(i*100/Outstring.length + "%");
181.                i++;
182.              }
183.            }
184.          }
185.        }
186.      }
```

```
187.    }
```

7.7. Programa control automático Arduino

```
8. #include <AccelStepper.h>
9. #include <MultiStepper.h>
10.
11.#define STEPX 3
12.#define DIRX A0
13.#define STEPY 2
14.#define DIRY A2
15.#define STEPZ 11
16.#define DIRZ A1
17.
18.AccelStepper stepperX(1, STEPX, DIRX);
19.AccelStepper stepperY(1, STEPY, DIRY);
20.AccelStepper stepperZ(1, STEPZ, DIRZ);
21.
22.MultiStepper steppersControl;
23.long gotopos[3]= {0, 0, 0};
24.String command;
25.float x = NAN, y = NAN, z = NAN;
26.
27.void setup() {
28.  Serial.begin(115200); //Iniciamos la comunicación
29.  stepperX.setMaxSpeed(650.0);
30.  stepperX.setAcceleration(100.0);
31.
32.  stepperY.setMaxSpeed(650.0);
33.  stepperY.setAcceleration(100.0);
34.
35.  stepperZ.setMaxSpeed(650.0);
36.  stepperZ.setAcceleration(100.0); //Definimos tanto la velocidad máxima como
   la aceleración de los motores
37.
38.  steppersControl.addStepper(stepperX);
39.  steppersControl.addStepper(stepperY);
40.  steppersControl.addStepper(stepperZ); //Añadimos al conjunto de control los
   tres motores
41.
42.  stepperX.setCurrentPosition(0);
43.  stepperY.setCurrentPosition(0);
44.  stepperZ.setCurrentPosition(0);
45.
```

```
46.   establishContact();
47.}
48.
49.void loop() {
50.   if (Serial.available() > 0) {
51.     command = Serial.readStringUntil('\n');
52.     command.trim();
53.     Serial.println("Arduino recieved: " + command);
54.     processCommand(command);
55.     steppersControl.moveTo(gotopos);
56.     steppersControl.runSpeedToPosition();
57.     Serial.println("G");
58.   }
59.}
60.
61.void processCommand(String command) {
62.   command.trim(); // Elimina los espacios en blanco iniciales y finales
63.   if (command.startsWith("G1")) {
64.     handleG1Command(command);
65.   } else if (command.startsWith("M02")) {
66.     handleM02Command();
67.   }
68.}
69.
70.void handleG1Command(String command){
71.
72.   // Busca las posiciones de los parámetros X, Y, Z
73.   float indexX = command.indexOf('X');
74.   float indexY = command.indexOf('Y');
75.   float indexZ = command.indexOf('Z');
76.
77.   if (indexX != -1) {
78.     x = command.substring(indexX + 1).toFloat();
79.   }
80.   if (indexY != -1) {
81.     y = command.substring(indexY + 1).toFloat();
82.   }
83.   if (indexZ != -1) {
84.     z = command.substring(indexZ + 1).toFloat();
85.   }
86.
87.   //Si el parámetro que ha leído es un número (not a NAN)
88.   if (!isnan(x)) {
89.     gotopos[0]=mm2steps(x);
```

```
90.   }
91.   if (!isnan(y)) {
92.     gotopos[1]=mm2steps(y);
93.   }
94.   if (!isnan(z)) {
95.     gotopos[2]=mm2steps(z);
96.   }
97.}
98.
99.void handleM02Command() {
100.     Serial.println("FINISHED");
101.   }
102.
103.   int mm2steps(float mm){
104.     return(round(mm*50));
105.   }
106.
107.   void establishContact() {
108.     while (Serial.available() <= 0) {
109.     Serial.println("A");   // Envía el mensaje de conexión
110.     delay(1000);
111.     }
112.   }
```

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería del Diseño

CARACTERIZACIÓN DEL SISTEMA ELECTRÓNICO DE UNA CNC E IMPLEMENTACIÓN DEL PROGRAMA ARDUINO PARA EL CONTROL DE LA MÁQUINA.

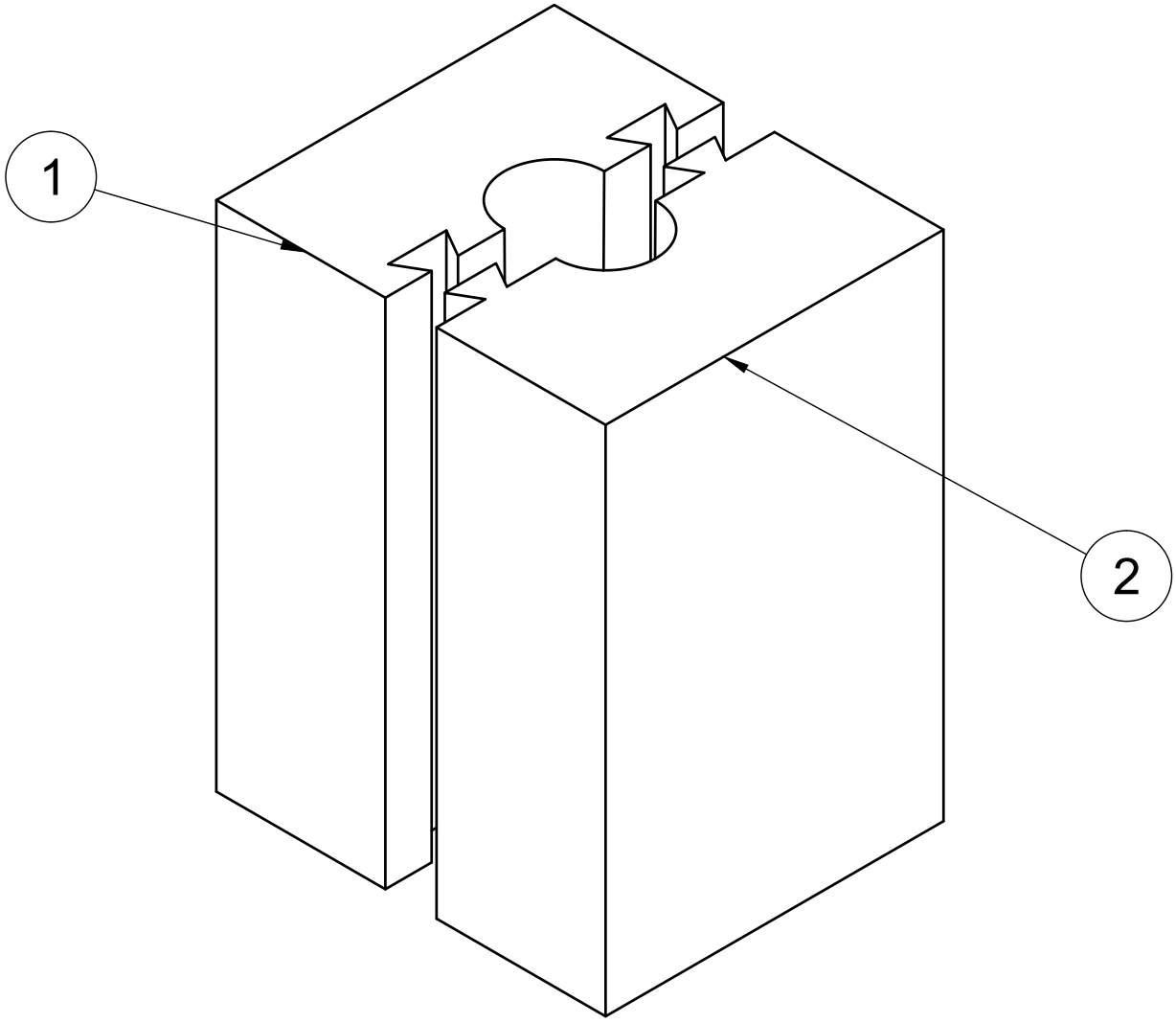
Documento 2º: Planos

Autor: Alejandro Dolz Molina

Tutor: Rafael Masot Peris

Curso académico 2023/24

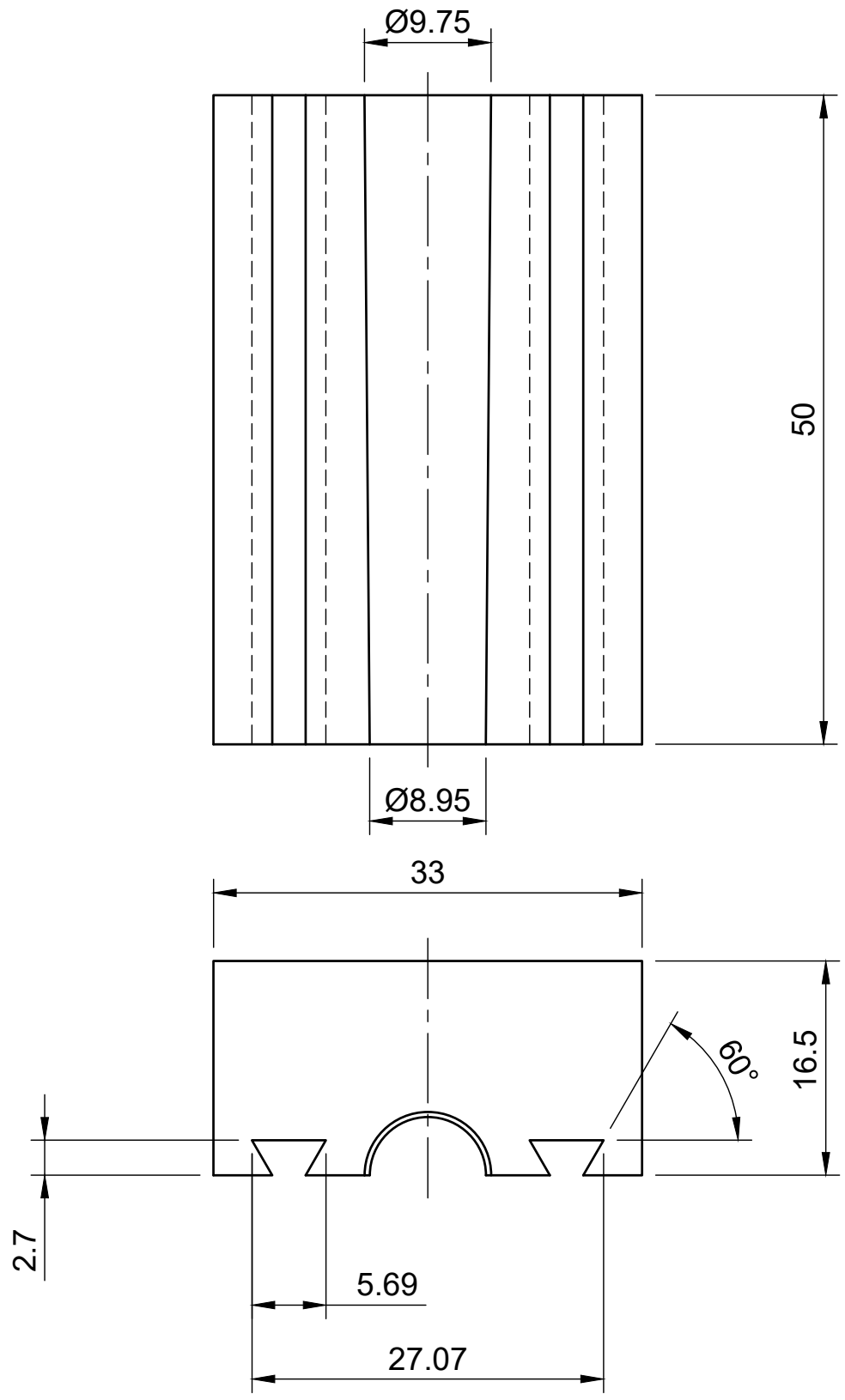
Índice de planos:



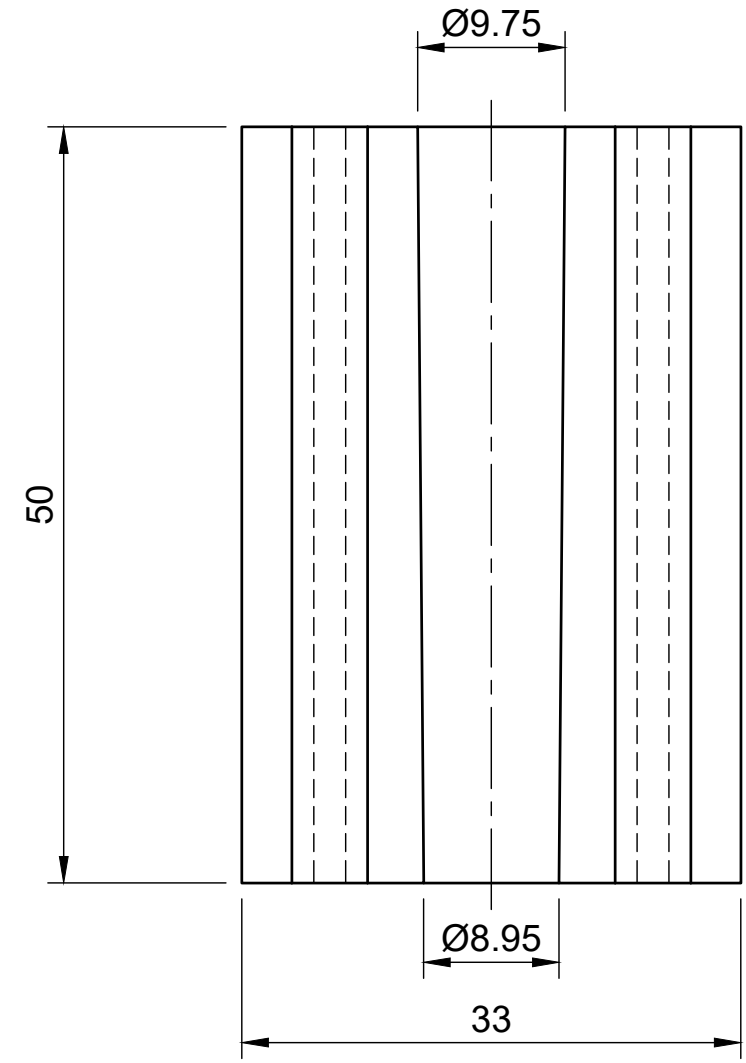
2	1	Macho	1.2
1	1	Hembra	1.1
Elemento	Ctd	Pieza	Referencia

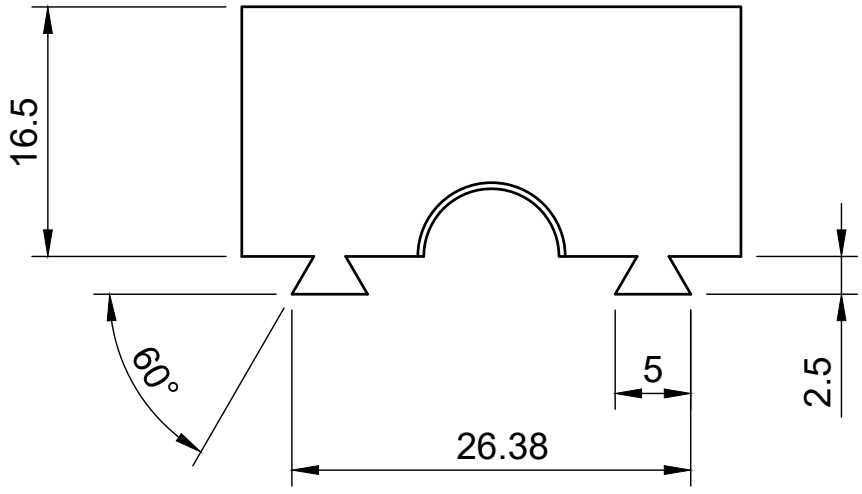
Lista de piezas

Proyecto: Implementación CNC		Fecha: 06/2024 Escala: 2:1
Autor: Alejandro Dolz Molina	Plano: Conjunto portabolígrafos	Nº plano: 1

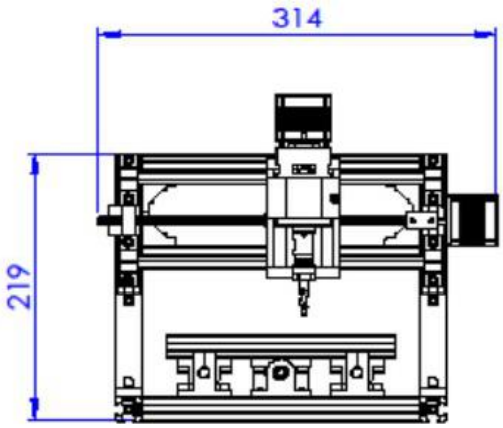


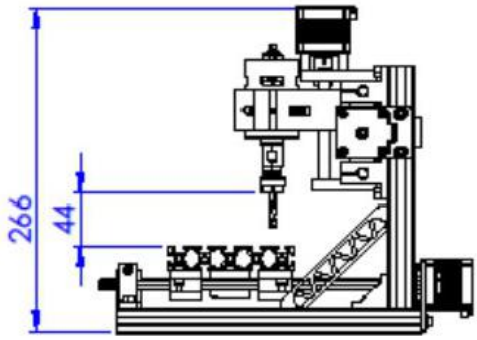
Proyecto: Implementación CNC		Fecha: 06/2024 Escala: 2:1
Autor: Alejandro Dolz Molina	Plano: Hembra	Nº plano: 1.1

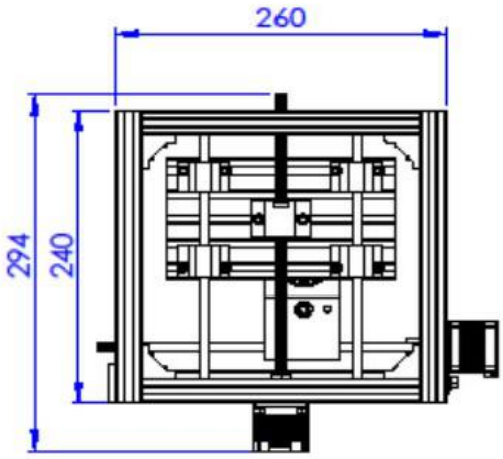


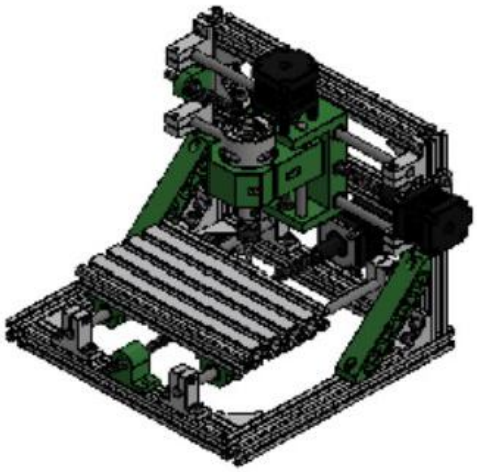


Proyecto: Implementación CNC		Fecha: 06/2024 Escala: 2:1
Autor: Alejandro Dolz Molina	Plano: Macho	Nº plano: 1.2

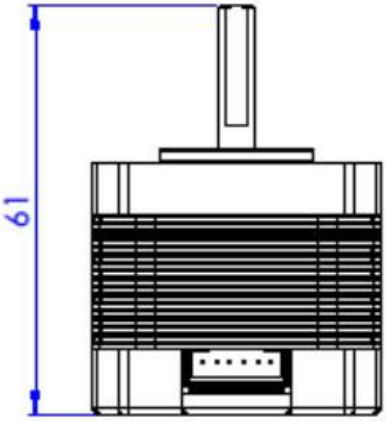


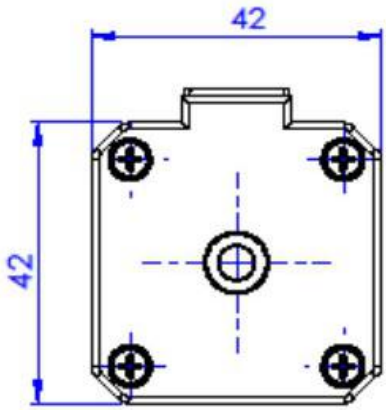


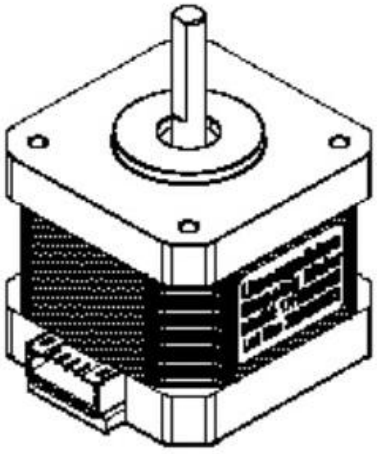




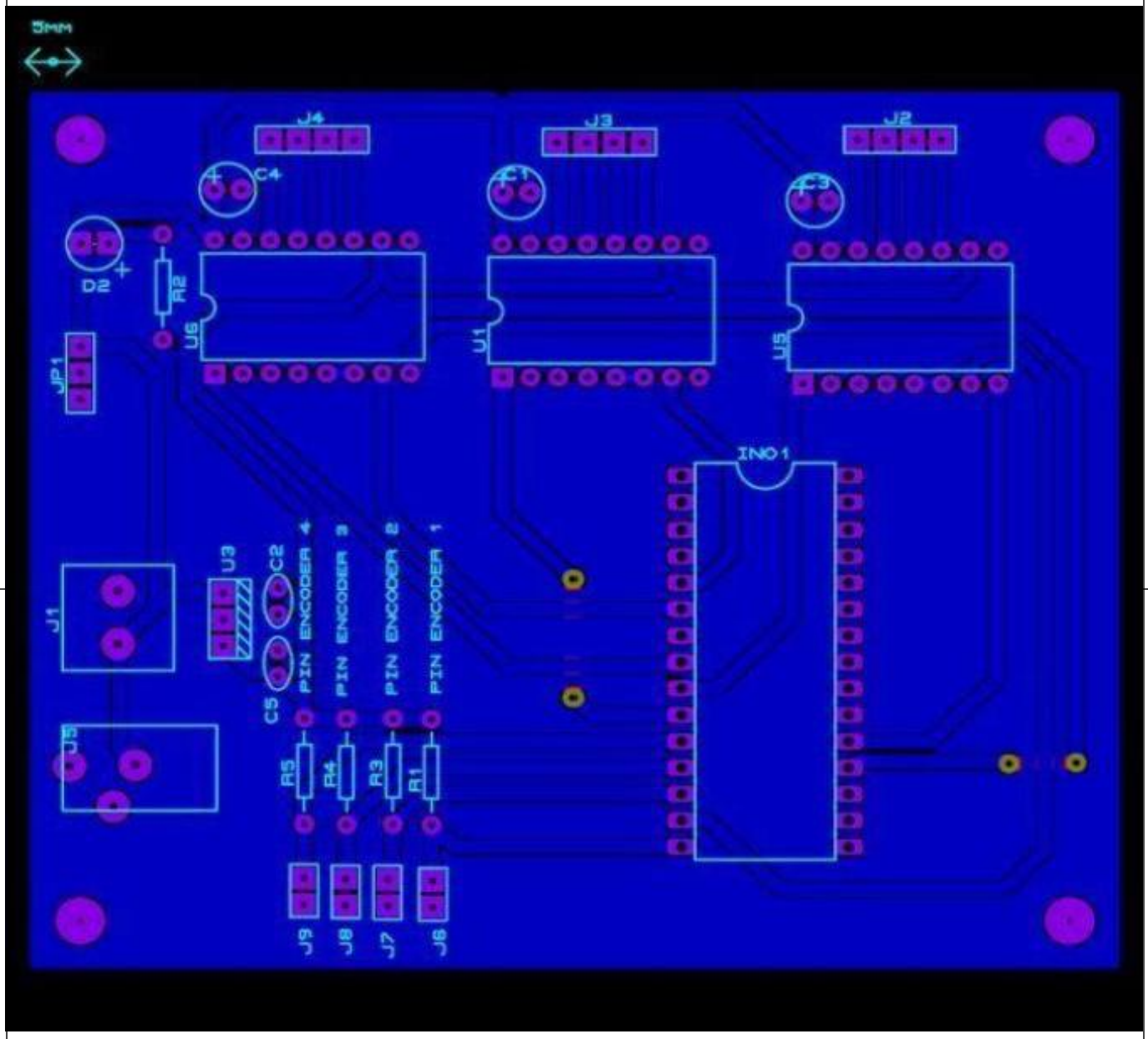
Proyecto: Implementación CNC		Fecha: 06/2024 Escala: 1:5
Autor: Alejandro Dolz Molina	Plano: Máquina CNC 1610	Nº plano: 2



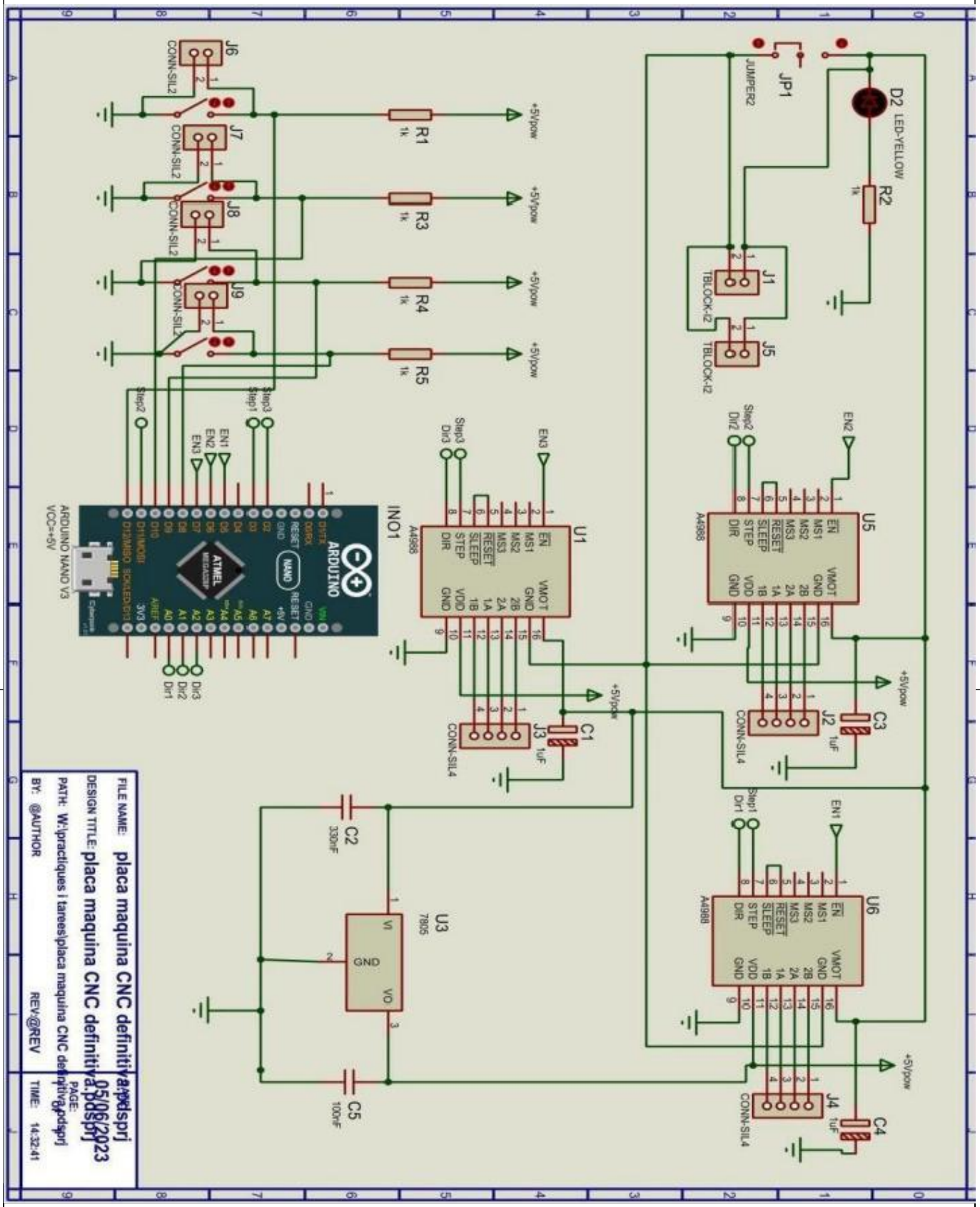




Proyecto: Implementación CNC		Fecha: 06/2024 Escala: 1:1
Autor: Alejandro Dolz Molina	Plano: Motor Nema 17M	Nº plano: 3



Proyecto: Implementación CNC		Fecha: 06/2024 Escala: --
Autor: Alejandro Dolz Molina	Plano: Esquema eléctrico PCB I	Nº plano: 4



Proyecto: Implementación CNC		Fecha: 06/2024 Escala: --
Autor: Alejandro Dolz Molina	Plano: Esquema eléctrico PCB II	Nº plano: 5

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería del Diseño

CARACTERIZACIÓN DEL SISTEMA ELECTRÓNICO DE UNA CNC E IMPLEMENTACIÓN DEL PROGRAMA ARDUINO PARA EL CONTROL DE LA MÁQUINA.

Documento 3º: Pliego de condiciones

Autor: Alejandro Dolz Molina

Tutor: Rafael Masot Peris

Curso académico 2023/24

Índice del pliego de condiciones:

1. OBJETIVO

El presente documento tiene por objetivo la determinación de las condiciones técnicas necesarias para el montaje, la instalación y la correcta puesta en marcha y funcionamiento de la máquina CNC. Este contiene la normativa de aplicación, la descripción detallada de los componentes necesarios, las comprobaciones pertinentes para confirmar que el montaje es correcto y las pruebas que se le deben de realizar a cada uno de los componentes, asegurando con todo esto el correcto funcionamiento de la máquina durante su uso.

2. NORMATIVA

Directiva de Máquinas 2006/42/CE:	Que se dirige a los fabricantes y distribuidores de máquinas y establece las tareas necesarias para que las nuevas máquinas cumplan con los requisitos de salud y seguridad.
Directiva 2014/30/UE:	Sobre la armonización de las legislaciones de los Estados miembros en materia de compatibilidad electromagnética.
UNE-EN ISO 16090-1:2023	Seguridad de las máquinas herramienta. Centros de mecanizado, centros de fresado, máquinas transfer. Parte 1: Requisitos de seguridad.
Norma UNE-EN 60204-1:2019:	Seguridad de las máquinas. Equipo eléctrico de las máquinas. Parte 1: Requisitos generales.

3. MATERIALES

Como fresadora se ha de utilizar la máquina CNC 1610, en el caso que los motores paso a paso no sean suministrados por el vendedor o que no sean NEMA 17M, se deberán colocar o intercambiar por el mencionado motor NEMA 17M. Si es necesario cambiar cualquier componente de la máquina usar siempre recambios originales.

Con respecto a la placa de circuito se deberá de usar obligatoriamente una placa de características idénticas a la detallada en el plano número 5.

Para el soporte del bolígrafo se deberá de utilizar PLC o cualquier otro plástico inyectable por impresora 3D.

4. NORMAS DE EJECUCIÓN Y SEGURIDAD

Para asegurar la seguridad durante la utilización de la máquina se deberá de cumplir en todo momento la normativa de seguridad expuesta en el apartado de este documento Normativa.

El rango de tensión que se le suministre a la placa deberá de ser de entre 15 y 24 voltios. Exceder estos valores puede ser peligroso y es posible que los componentes electrónicos y mecánicos de la fresadora pueden quedar dañados o inservibles.

Hay que destacar que, en caso de usar el sistema con una tensión superior a 15 V será obligatorio el uso de disipadores para los *chips* del *driver* A4988, esto para evitar que se puedan sobrecalentar y quedar dañados.

Durante la duración del programa no se deberán tocar ni los tonillos guía de la máquina ni ningún otro componente móvil, para evitar cualquier posible accidente.

5. PRUEBAS DE SERVICIO

En primer lugar, se deberá conectar un solo motor a la placa, y con una tensión de 24 V se deberá comprobar que el voltaje de diferencia es de **0,737 V** tal y como se indica en el apartado 3.3.1 de la memoria, *Ajuste del voltaje de referencia*, esto para asegurar que no se sobrepasarán los niveles de corriente admitidos por los motores. Para ello se colocará el borne positivo del multímetro conectado con la parte metálica interna del potenciómetro del controlador y el borne negativo en contacto con el pin de *GRD* de la placa.

En segundo lugar, se deberá comprobar el correcto funcionamiento de cada uno de los drivers usando el programa especificado en el anexo *7.3. Programa de prueba de motores*. Durante la ejecución de este programa se deberá comprobar que los motores giran adecuadamente tanto en sentido horario como en sentido antihorario y que ni los motores, ni ninguno de los componentes, se calienta en exceso. Se comprobará también que los chips reciben 5 V de tensión en los pines de la entrada lógica y la tensión que se le este suministrado (siempre entre 14 y 24 V) en los pines de alimentación.

Si en alguno de los componentes se detecta un funcionamiento erróneo o alguna anomalía deberá de ser sustituido de inmediato y, posteriormente, desechado en un punto limpio para productos electrónicos.

Se deberá comprobar también que el LED de la placa de circuito funciona adecuadamente, iluminándose cuando se está alimentando el circuito.

Estos procesos descritos en el apartado de deberán verificar tantas veces como sea necesario en caso de duda.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Escuela Técnica Superior de Ingeniería del Diseño

CARACTERIZACIÓN DEL SISTEMA ELECTRÓNICO DE UNA CNC E IMPLEMENTACIÓN DEL PROGRAMA ARDUINO PARA EL CONTROL DE LA MÁQUINA.

Documento 4º: Presupuesto

Autor: Alejandro Dolz Molina

Tutor: Rafael Masot Peris

Curso académico 2023/24

Índice del presupuesto:

1. RESUMEN

RESUMEN DEL PRESUPUETSO

Descripción	Materiales	Mano de obra	Gastos de equipamiento	Total
Fresadora	126,42 €	24,00 €	-	150,42 €
Soporte bolígrafo	0,45 €	85,00 €	-	85,45 €
Circuitos	68,23 €	700,00 €	76,82 €	845,06 €
Programación	-	4.250,00 €	-	4.250,00 €

Descripción	Total
Amortizaciones	45,97 €

	Materiales	Mano de obra	Gastos de equipamiento	PEM
Total	**195,10 €**	**5.059,00 €**	**76,82 €**	**5.376,90 €**

Uds.	Concepto	Cantidad	Importe
%	Gastos de Estructura	12%	645,23 €
%	Beneficio Industrial	7%	376,38 €

		Total SIN IVA	6.398,51 €

%	IVA	21%	1.343,69 €

		P.V.P.	**7.742,19 €**

Figura 33:TablarResumen presupuesto

2. FRESADORA

FRESADORA				
Materiales				
Uds	Descripción	Cantidad	Precio (€)	Total
uds	Kit montaje CNC 1610	1	126,42 €	126,42 €
			Subtotal materiales	**126,42 €**
Mano de obra				
Uds	Descripción	Tiempo	Precio (€)	Total
h	Tiempo total de montaje	2	12,00 €	24,00 €
			Subtotal mano de obra	**24,00 €**
			Coste total	**150,42 €**

Figura 34: Tabla presupuesto fresadora

3. SOPORTE BOLÍGRAFO

Soporte bolígrafo				
Materiales				
Uds	Descripción	Cantidad	Precio (€)	Total
g	Filamento PLC	43	0,0104 €	0,45 €
			Subtotal materiales	**0,45 €**
Mano de obra				
Uds	Descripción	Tiempo	Precio (€)	Total
h	Tiempo de diseño	5	17,00 €	85,00 €
			Subtotal mano de obra	**85,00 €**
			Coste total	**85,45 €**

Figura 35: Tabla presupuesto soporte bolígrafo

4. CIRCUITOS

Circuitos				
Materiales				
Uds	Descripción	Cantidad	Precio (€)	Total
uds	Placa PCB revestida de cobre	1	1,14 €	1,14 €
uds	Resistencia	5	0,10 €	0,50 €
uds	Condensador	5	0,20 €	1,00 €
uds	Driver A4988	3	0,66 €	1,98 €
uds	Arduino NANO	1	21,60 €	21,60 €
uds	Regulador tensión	1	0,61 €	0,61 €
uds	Jumper conexión	3	0,78 €	2,34 €
uds	Regleta conexión	2	0,61 €	1,22 €
uds	Interruptor	1	0,35 €	0,35 €
uds	LED	1	0,20 €	0,20 €
g	Estaño	10	0,23 €	2,26 €
l	Ácido clorhídrico 37%	2	10,45 €	20,90 €
kg	Ácido férrico	2	4,50 €	9,00 €
kg	Sosa cáustica	1	5,13 €	5,13 €
			Subtotal materiales	**68,23 €**
Mano de obra				
Uds	Denominación	Tiempo	Precio (€)	Total
h	Diseño	20	17,00 €	340,00 €
h	Fabricación placa	20	12,00 €	240,00 €
h	Soldado placa	10	12,00 €	120,00 €
			Subtotal mano de obra	**700,00 €**
			Coste fabricación	**768,23 €**
Gastos de equipamiento				
Uds	Descripción		Cantidad	Total
Costes directos	Soldador, lupas, eqipos de revelado de la placa.		10%	76,82 €
			Subtotal equipamiento	**76,82 €**
			Coste total	**845,06 €**

Figura 36: Tabla presupuesto circuitos

5. PROGRAMACIÓN

Programación				
Mano de obra				
Uds	Denominación	Tiempo	Precio (€)	Total
h	Tiempo de programación	250	17,00 €	4.250,00 €
			Subtotal mano de obra	**4.250,00 €**

Figura 37: Tabla presupuesto programación

6. AMORTIZACIONES

AMORTIZACIONES					
Descripción	Coste Inicial	Vida útil (h)	Horas empleadas	€/h	Total
Impresora 3D PRUSA	1200	14000	3	0,0857	0,26 €
Ordenador Gigabyte	1500	10000	300	0,1500	45,00 €
Fuente de alimentación	100	7000	50	0,0143	0,71 €
			TOTAL		**45,97 €**

Figura 38: Tabla amortizaciones