UNIVERSIDAD TECNOLÓGICA ISRAEL

TRABAJO DE TITULACIÓN EN OPCIÓN AL GRADO DE:

“INGENIERÍA EN ELECTRÓNICA DIGITAL Y TELECOMUNICACIONES”

TEMA: Implementación de una tarjeta de control para una mesa mediante Control Numérico Computarizado.

AUTOR: ABRAHAM ALEJANDRO TURRIAGO ESCOBAR

TUTOR: Mg. René Ernesto Cortijo Leyva

AÑO: 2018

UNIVERSIDAD TECNOLÓGICA ISRAEL

APROBACIÓN DEL TUTOR

En mi calidad de tutor del trabajo de titulación certifico:

Que el trabajo de titulación **"Implementación de una tarjeta de control para una mesa mediante Control Numérico Computarizado.**..", presentado por el Sr. Abraham Alejandro Turriago Escobar, estudiante de la carrera de Electrónica Digital y Telecomunicaciones, reúne los requisitos y méritos suficientes para ser sometido a la evaluación del Tribunal de Grado, que se designe, para su correspondiente estudio y calificación.

Quito D.M. enero del 2018

TUTOR

……..

Ing. Rene Ernesto Cortijo Leyva, Mg

Agradecimiento:

Agradezco a esa persona superior que siempre ha guiado mis pasos, a la Universidad Israel por darme la oportunidad de terminar lo que un día dejé pendiente, a los profes por toda la paciencia, en especial al Ing. Mauricio Alminati y al Mg. René Cortijo por apoyarme en este proyecto y finalmente a mi gran amigo Johnny Mena por compartir su sabiduría e ingenio conmigo.

Dedicatoria

Quiero dedicar este trabajo a mis padres, porque ellos son mi inspiración de superación, a la persona más importante en mi vida porque siempre me ha apoyado en todo, mi novia y compañera de vida Xime y a todos quienes en algún momento hicieron posible la realización de este sueño.

Resumen.

El presente trabajo plantea realizar la implementación de una tarjeta de control para una mesa mediante Control Numérico Computarizado con precisión a velocidades de producción, en respuesta a la problemática de las pequeñas y medianas empresas locales por la inaccesibilidad a las máquinas CNC, debido a su alto precio, impidiendo mejorar la velocidad de producción de este sector metal-mecánico.

Esta mesa está basada en un sistema que controla los movimientos de la herramienta con la que se realiza los cortes basándose en los ejes de coordenadas X, Y, Z, a través de un código de programación que se ejecuta desde un computador.

Es así, que el funcionamiento de la mesa para convertir un modelo virtual en un corte de una pieza consiste en seis etapas. Primero se inicia con la elaboración de un diseño asistido por computadora CAD. Luego, se generan los códigos G, los cuales se ingresan en el software de control a la PC que controla al microcontrolador Arduino Uno. A través del GRBL, el Arduino Uno controla el driver A4988 en cargado de hacer girar los motores para realizar el proceso de corte.

Luego de la elaboración de la tarjeta de control para la mesa CNC se concluyó que la mesa puede ser construida bajo especificaciones propias del cliente, para mejorar la calidad del trabajo terminado. En este caso el corte más adecuado para la necesidad del usuario, fue utilizar una alimentación del plasma a 220 V, 18 A, con presión de 95psi. Además, a través del envió del código G se consiguió comprobar el correcto funcionamiento de los motores paso a paso, permitiendo realizar el corte de la mesa CNC. El app inventor dos nos permitió realizar una interfaz inalámbrica de control

Palabras clave: CNC, Arduino uno, APP Inventor 2, Shield CNC, Plasma

Abstract

The present Project proposes the implementación of a control card for a CNC machine , which works with precision and speed , this is in response of the lack of these type of machines in the small and médium industries, due to its high cost, avoiding the improvement of this metal mechanical sector.

This table is based on a system that controls the movements of the tool with which the cuts are made based on the axes of X, Y, Z, through a programming code that is executed from a computer.

Thus, the operation of the table to convert a virtual model into a piece through cut consists of six stages. First, it begins with the development of a computer-aided CAD design. Then, the G codes are created, which are entered in the control software to the PC that controls the Arduino Uno microcontroller. Through the GRBL, the Arduino Uno controls the A4988 driver in charge of turning the motors to carry out the process of cut.

After the development of the control card for the CNC table it was concluded that the table can be built under the client's own specifications, improving the quality of the final result. In this case, the most suitable cut for the user's need was to use a 220 V, 18 A plasma power supply, with a pressure of 95 psi. In addition, through the G code, it was possible to check the proper operation of the step by step motors , allowing the cutting of the CNC table.The inventor two app did not allow a wireless control interface

Keywords:

CNC, Arduino uno, APP Inventor 2, Shield CNC, Plasma

Índice.

Índice de figuras.

Índice de Tablas.

Introducción

Hace aproximadamente 3.400 años se desarrollaron las primeras herramientas rudimentarias de corte, para la elaboración de piezas que ayudarían al hombre a construir cosas. Desde ese entonces hasta ahora, la industria metalmecánica se ha ido desarrollando aceleradamente, es así que tecnológicamente se puede decir que las primeras herramientas diseñadas fueron taladros y tornos sencillos.

Este desarrollo de herramientas y maquinarias no ha parado, uno de los mejores inventos ha sido las maquinas CNC, las que a través de una computadora pueden controlar la velocidad de sus motores, recibiendo órdenes para generar movimientos como círculos, diagonales y figuras de alta complejidad que con máquinas manuales serían difíciles de realizar. El término CNC significa control numérico computarizado y hoy en día es un método que permite controlar con gran precisión la operación de una máquina, así como de varias a través de instrucciones a base de códigos, para crear piezas y componentes con un mínimo de error humano.

En cuanto a estos avances de la industria metal mecánica en nuestro país se puede decir, que en el mercado local las plantas de producción a gran escala han podido importar máquinas CNC, lo que no sucede en las empresas que son pequeñas y medianas, ya que éstas no cuentan con el presupuesto suficiente para adquirir este tipo de herramientas, muy costosas en nuestro país.

De esta forma, se evidencia una problemática al aceptar que las empresas pequeñas locales carecen de máquinas CNC que les ayuden a mejorar la velocidad de producción con alta precisión a precios accesibles.

Antecedentes de la situación objeto de estudio

Las opciones de importaciones de estas máquinas CNC están destinadas a las industrias de gran escala, no a pequeños y medianos productores, situación que se visualiza en el precio de adquisición de esta herramienta.

Planteamiento del problema

En el medio ecuatoriano esta herramienta está a un precio inaccesible para los profesionales que necesitan de este tipo de tecnología para complementar su trabajo en el ámbito de la industria. De igual forma, no es posible encontrar una herramienta con precisión a velocidades de producción adecuadas como el router CNC propuesta en este trabajo.

Formulación del problema

Es posible realizar la implementación de una tarjeta de control para una mesa mediante Control Numérico Computarizado con precisión a velocidades de producción, de tal forma que sea indispensable en las industrias metal mecánica, que se encuentra en el mercado a un precio inaccesible para los profesionales de la industria mencionada.

Justificación

La implementación de esta tesis, aumentará la productividad por ende los ingresos económicos serán más altos y se podrá realizar y desarrollar innovaciones de nuevos productos en el mercado.

Con respecto a los trabajadores que cortan manualmente con el plasma, no les quita o desplaza de su área laboral, sino más bien les permite capacitarse y ser más técnicos al usar esta máquina herramienta, además disminuye el riesgo laboral al estar lejos de la antorcha del plasma, evitando exposición a la radiación del arco eléctrico, así como también a las quemaduras por alta temperatura del arco y proyección de partículas calientes a alta velocidad (chispas).

Además, al contar con la mesa se minimiza el remanente de polvos, limallas, que desprende el corte por plasma quedan en el ambiente y exponiendo a contaminación de vías aéreas de los trabajadores.

Costo bajo comparado con el existente en el mercado. La mesa puede ser construida bajo especificaciones propias del cliente, para mejorar la calidad del trabajo terminado, incrementando la productividad y el desarrollo de nuevos productos e innovaciones.

Objetivo General

Desarrollar una tarjeta de control para una mesa mediante Control Numérico Computarizado

Objetivo Específico

Establecer el método de corte más adecuado de acorde a la necesidad del usuario.

Diseñar la tarjeta de control automático para realizar los movimientos requeridos mediante Arduino.

Realizar el programa que permita controlar los movimientos requeridos de la herramienta de corte, utilizando algoritmo de control numérico computarizado.

Diseñar la interfaz de comunicación entre la placa de control y el sistema central computarizado por un medio inalámbrico.

Verificar el correcto funcionamiento a través de una serie de pruebas del sistema implementado.

CAPÍTULO I. FUNDAMENTACIÓN TEÓRICA

1.1 Marco Teórico

Automatización

Actualmente, dentro de los avances tecnológicos está la automatización, que se define como un sistema donde se trasfieren tareas de producción, normalmente realizadas por trabajadores contratados, a un conjunto de elementos tecnológicos. Las máquinas de Control Numérico Computarizado (CNC) son elementos importantes de la automatización, para la creación de piezas.

Programación

El objetivo de la programación es desarrollar programas que demuestren un comportamiento esperado. Con el fin de obtener este comportamiento deseado se debe escribir códigos, para lo que se requiere tener conocimientos en diferentes áreas. Por ejemplo, el código G en CNC detalla el tipo de movimiento u operación en el bloque que incluye el código G.

Diseño Electrónico

Al diseño electrónico se lo define como el arte de inventar, cambiar o solucionar un problema, en el ámbito de la electrónica. El diseño electrónico abarca varios campos como el hardware, firware, y software.

Plasma

El plasma es un gas ionizado eléctricamente, con el cual se puede realizar el corte del metal. Debido a las propiedades conductivas de los metales es conocido técnicamente como PAW (Plasma Arc Welding). En el caso de la suelda por plasma, lo necesario es la energía para la ionización, la que se genera por el arco eléctrico que crea entre un electrodo de tungsteno y el metal base a soldar. Durante este proceso, se utiliza un gas como soporte de arco, éste

normalmente es argón en estado puro o helio con porciones pequeñas de hidrógeno que pasa ha estado plasmático con el orificio que se encuentra en la boquilla, en donde se estrangula el arco.

Bluetooth

Es un sistema para Redes Inalámbricas de Área Personal (WPAN), que admite transmitir datos medios y voz a través de enlaces de radio frecuencia a 2.4 GHZ.

Permite comunicación entre equipos móviles y no utiliza cables ni conectores extras.

Además, se pueden crear redes inalámbricas pequeñas, las que hacen más fácil la sincronización de datos entre equipos, se los usa en teléfonos celulares, computadoras, cámaras digitales, parlantes de audio.

1.2 Descripción Mesa de Corte CNC para Plasma.

Es una mesa altamente automatizada en la cual se ejecutan diferentes programas de control numérico, realizados por el software CAD/CAM, partiendo de un diseño. Según Pintag y Hernández (2014) en ella se podrá realizar diversas operaciones a nivel industrial con la participación mínima de empleados humanos.

Principios de Funcionamiento.

Esta mesa está basada en un sistema que controla los movimientos de la herramienta con la que se realiza los cortes basándose en los ejes de coordenadas X, Y, Z, a través de un código de programación que se ejecuta desde un computador. En la mesa CNC se controlan estos ejes de desplazamiento por medio de motores paso a paso.

Programación y Códigos en CNC

Con la finalidad de alcanzar un estándar en el código G, se designa letras que ordenan parámetros, así como también actividades y oficios específicos.

Maldonado (2015) explica que el código G designa la mayor parte de parámetros tales como mecanizado o los sistemas de medidas, por ejemplo, si se desea arrancar la operación de la máquina se utiliza, G70 en el caso de pulgadas, pero para milímetros se utiliza G71.

Estos códigos de programación se ven ejemplificados en la tabla 1, con sus respectivas letras correspondientes a las funciones principales para CNC

Tabla 1.

Letras para códigos de programación en CNC

N	Número de Secuencia
G	Funciones Preparatorias
X	Comando del Eje X
Y	Comando del Eje Y
Z	Comando del Eje Z
R	Radio desde el Centro Especificado
A	Ángulo contra los punteros del reloj desde el vector +X
I	Desplazamiento del Centro del Arco del Eje X
J	Desplazamiento del Centro del Arco del Eje Y
K	Desplazamiento del Centro del Arco del Eje Z

F	Tasa de Alimentación
S	Velocidad de Giro
T	Número de la Herramienta
M	Función Miscelánea

Fuente: Maldonado, 2015.

Funciones preparatorias

G es la letra destinada para las funciones preparatorias, con el fin de dar a conocer al control, los rasgos de las funciones de mecanizado, tal es el caso de la forma de la trayectoria, la programación sea esta absoluta o relativa, la corrección de la herramienta, ciclos automáticos, parada temporizada y otras características. A continuación de la letra G se asigna dos dígitos, los que permitirán programar una gran gama de funciones preparatorias.

G00: Este código es para un trayecto con velocidad máxima ó desplazamiento en rápido. G01: Este código ordena un movimiento de la herramienta en línea recta, con ejes gobernados.

G02: Este código es para interpolación lineal en sentido horario.

G03: Este código es para interpolación lineal en sentido anti horario.

G77: Este código es para para programar el torneado de un cilíndro pero con un único bloque y a través de un ciclo automático.

G33: Indica ciclo automático de roscado.

Funciones misceláneas

M es la letra con la que se programan funciones secundarias o complementarias. Por ejemplo, sirve para señalar a la máquina herramienta que debe realizar rotación de husillo con dirección derecha o izquierda o si no también para parada programada, cambio de útil, entre otras funciones. Al igual que se mencionó en líneas anteriores, con la letra M y dos dígitos también se puede programar un alto número de funciones distintas.

M00: Código para detener el husillo y refrigeración, provocando en el programa una parada incondicional.

M02: Código para finalizar el programa con el control, luego de haber realizado las funciones establecidas en el bloque.

M03: Código para programar que el husillo rote con dirección horaria.

M04: Código para programar que el husillo rote, pero a diferencia de la función anterior, ésta es con dirección anti horaria.

1.2.3 Elementos y dispositivos para el control

A los actuadores se los define como dispositivos que generan movimientos de los mecanismos, para cambiar energía eléctrica en energía mecánica en el sistema CNC.

Los factores de control son partes eléctricas que supervisan el funcionamiento adecuado de un sistema.

Motores a Paso

Los motores de paso son considerados para la elaboración de mecanismos en los que se requieren desplazamientos con una alta precisión, y se caracterizan por mover con cada pulso un único paso. Este desplazamiento varía desde 90° hasta movimientos imperceptibles de

1.8°, por lo tanto, en el caso de desear un desplazamiento de 90° se necesitarán 4 pasos y para 1.8° se necesitarán 200.

En la figura 1 se muestra la forma física de un motor paso a paso Nema

Figura 1. Stepper Motor Paso Nema 17

Fuente: Recuperado de http://www.pbclinear.com/Download/DataSheet/Stepper-Motor-Support-Document.pdf

Drivers.

El driver sirve para que los sistemas operativos reconozcan y permitan la comunicación con diferentes dispositivos, estos reciben una señal lógica de un computador y entregan una señal de potencia.

A4988.

Es un driver que ayuda a simplificar el manejo de motores paso a paso y tiene una intensidad máxima de 2A por bobinado, al igual que una tensión máxima de 8 a 35V. Este driver es altamente popular por su sencillez al momento de ser utilizado, ya que para su control únicamente se requiere una salida que indique el sentido de giro y otra que indique que se desea un avance de un paso del motor

Características (coldfire electrónica, s/f):

- Control sencillo de dirección y paso

- Resoluciones diferentes (5): paso completo, medio paso, un cuarto de paso, un octavo de paso y un dieciseisavo de paso.

- Control ajustable de corriente a través de un potenciómetro utilizando tensiones superiores logrando mayores tasas de paso.

- Control inteligente que selecciona automáticamente la corriente decay (fast decay y slow decay)

- Protección para sobrecalentamiento térmico para baja tensión y sobrepico de corriente.

En la figura 2 se muestra las conecciones del Driver A4988.

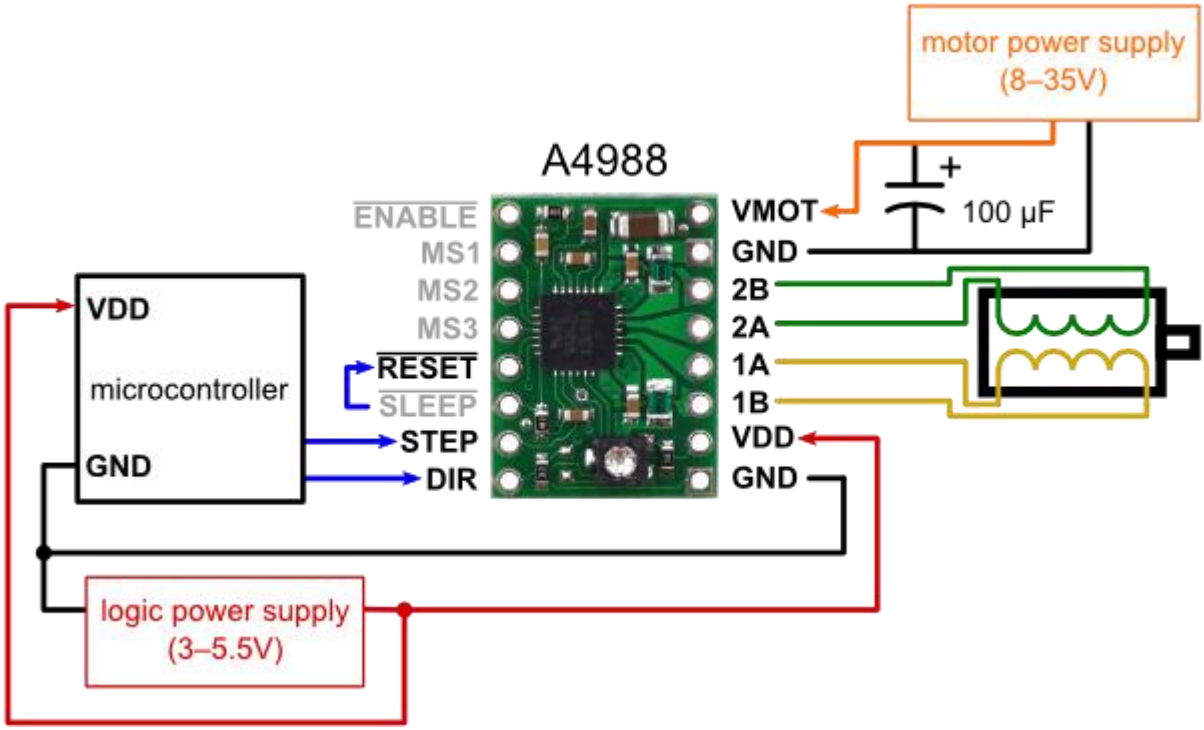


Figura 2. Driver A4988

Fuente: Recuperado de https://www.pololu.com/product/1182

Arduino Uno.

Es una tarjeta electrónica de software y hardware libre, es uno de los más utilizados para la elaboración de proyectos electrónicos y automatización, por su versatilidad fácil programación y bajo costo.

Figura 3 podemos observar el Arduino Uno y sus diferentes elementos

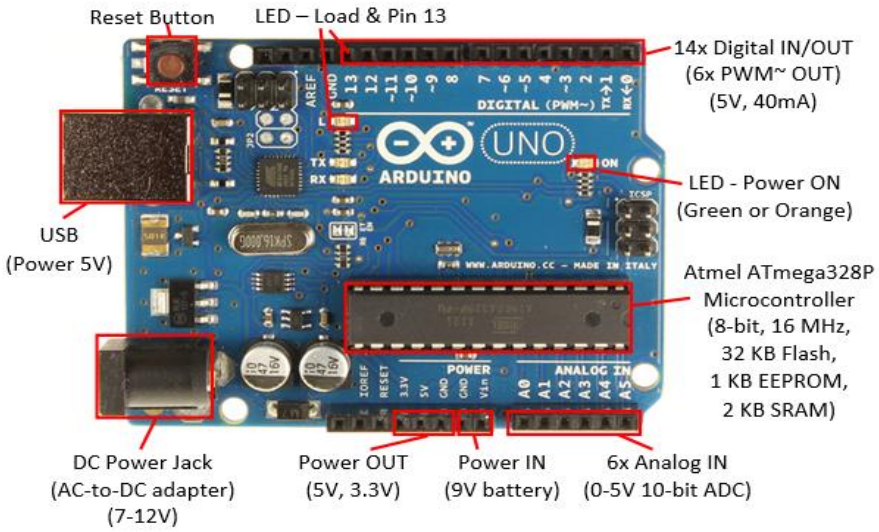


Figura 3. Arduino Uno

Fuente: Recuperado de https://www.ntu.edu.sg/home/ehchua/programming/arduino/Arduino.html

Controlador Arduino CNC Shield.

Es una placa que admite controlar fácilmente un máximo de 4 motores paso a paso. Este controlador es un circuito que se utiliza como complemento del Arduino y es compatible con el firmware de control GRLB, y además puede ser utilizado con cualquier modelo de Arduino, soporta 4 controladores A4988, que trabajan a 2 amperes y 35 voltios a la salida.

A continuación, en la figura 4 se muéstrala tarjeta Shield CNC que estamos utilizando.

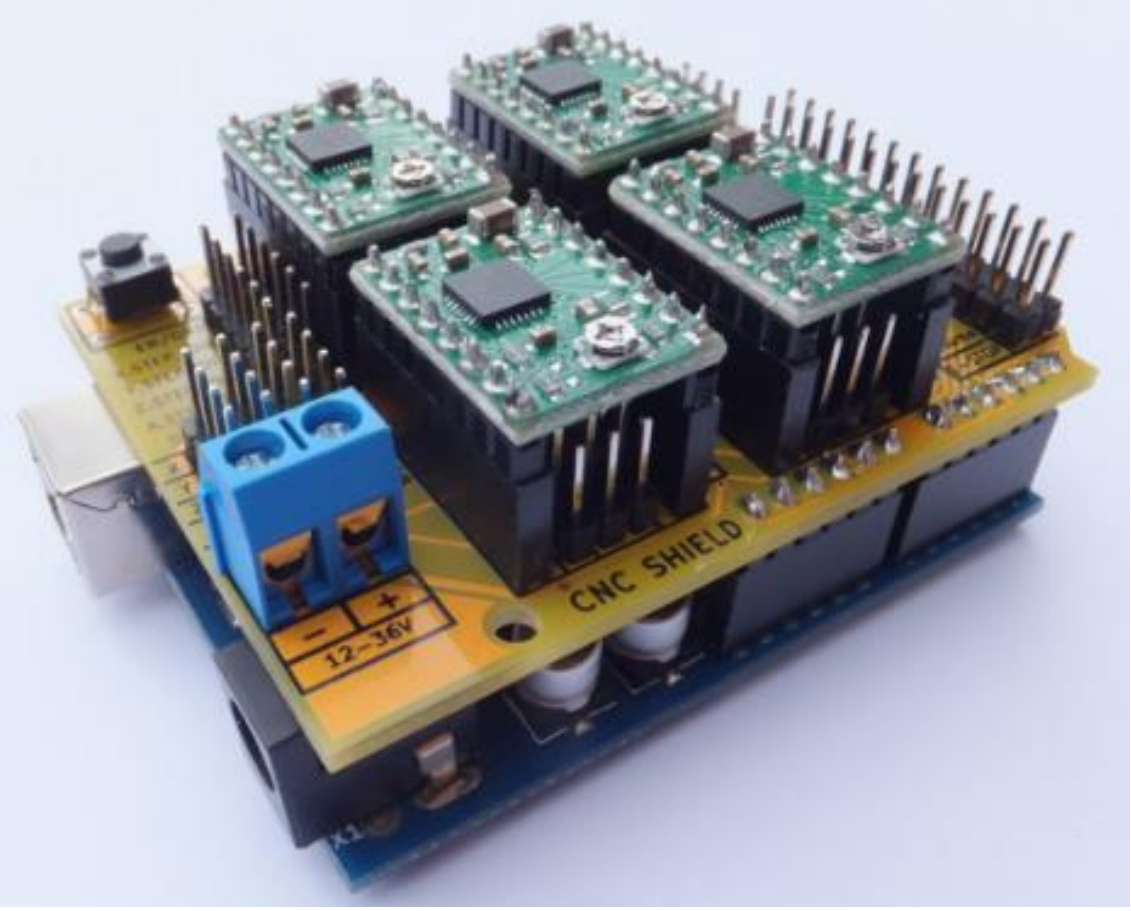


Figura 4. Controlador Arduino CNC Shield

Fuente: Cifuentes y Blandón (2015)

GRBL

Es un software creado para exportar G-code a máquinas CNC, de alta potencia y bajo costo diseñado para control de movimientos, se lo ejecuta desde el computador.

Tecnología Bluetooth.

La red Bluetooth permite la transmisión de voz y datos, entre dispositivos diferentes, a través de ondas de radio de baja potencia (2,45 GHz), con mayor exactitud va desde 2.402 GHz y 2.480 GHz. Con un acuerdo internacional, esta frecuencia fue anulada en el caso de uso de dispositivos industriales, científicos y médicos.

Bluetooth HC-06.

El tipo de Bluetooth que se va a utilizar en el proyecto es el módulo Bluetooth HC-06, montado en tarjeta base de 4 pines para una fácil para un simple manejo, interface serial, modo esclavo, Bluetooth v2.0 + EDR, 2.4 GHz, alcance 5 m a 10 m.

Especificaciones (Guangzhou HC Information Technology Co., Ltd, s/f)

- Especificación bluetooth v2.0 + EDR (Enhanced Data Rate)
- Modo esclavo (Solo puede operar en este modo)
- Puede configurarse mediante comandos AT (Deben escribirse en mayúscula)

Chip de radio: CSR BC417143

- Frecuencia: 2.4 GHz, banda ISM
- Modulación: GFSK (Gaussian Frequency Shift Keying)
- Antena de PCB incorporada
- Potencia de emisión: ≤ 6 dBm, Clase 2
- Alcance 5 m a 10 m
- Sensibilidad: ≤ -80 dBm a 0.1% BER
- Velocidad: Asincrónica: 2 Mbps (max.) /160 kbps, sincrónica: 1 Mbps/1 Mbps
- Seguridad: Autenticación y encriptación (Password por defecto: 1234)
- Perfiles: Puerto serial Bluetooth
- Módulo montado en tarjeta con regulador de voltaje y 4 pines suministrando acceso a VCC, GND, TXD, y RXD
- Consumo de corriente: 30 mA a 40 mA
- Voltaje de operación: 3.6 V a 6 V
- Dimensiones totales: 1.7 cm x 4 cm aprox.
- Temperatura de operación: -25 ºC a +75 ºC

Figura 5. Modulo Bluetooth HC-06

Fuente: Recuperado de http://www.electronicoscaldas.com/modulos-rf/482-modulo-bluetooth-hc-06.html

CAPÍTULO II. PROPUESTA

La propuesta ante el problema presentado en este documento, es la elaboración de una mesa de corte CNC, cumpliendo con especificaciones y reduciendo la dificultad que conlleva.

Algunos de los elementos que conforman la mesa de corte CNC son los controladores, los que son programados a base de estándares ISO 6983(Electonic Standarization Organización) y el estándar EIAR RS274 (International Industries Association).

Estos estándares de instrucciones de programación (código) permiten a las máquinas realizar operaciones específicas o particulares.

A continuación, un ejemplo para indicar que la máquina se mueva 10 mm por lado.

```
G90 G71          (cotas absolutas referidas al punto 0,0; Programación en mm)
G00 X0.0 Y0.0    (posicionamiento rápido lineal al punto 0,0 del plano XY)
G01 X10.0        (movimiento lineal de 10mm en la dirección X positiva)
G01 Y10.0        (movimiento lineal de 10mm en la dirección Y positiva)
G01 X0.0         (movimiento lineal de 10mm en la dirección X negativa)
G01 Y0.0         (movimiento lineal de 10mm en la dirección Y negativa)
```

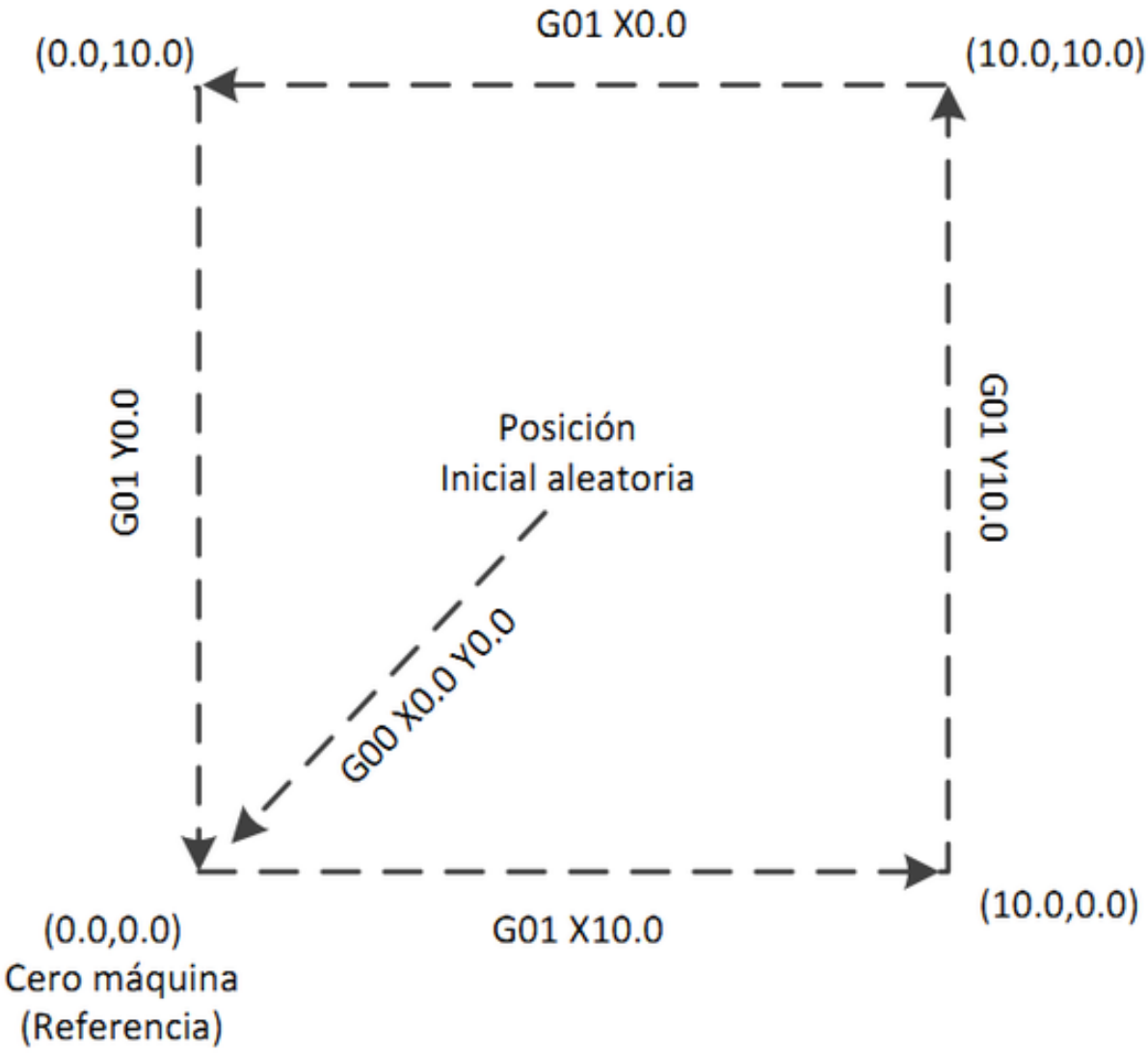


Figura 6. Puntos de referencia

Fuente: Recuperado de http://wiki.ead.pucv.cl/index.php/Archivo:EsquemaCodigoG.png

Un programa de máquina es el conjunto de órdenes siguiendo una secuencia lógica, por ejemplo, con las órdenes respectivas, la mesa puede plasmar una ranura o un dibujo.

Principio de Funcionamiento.

El principio de funcionamiento de una CNC se basa en conceptos geométricos. Utilizando sistemas de coordenadas para hacer posible los movimientos de corte de la máquina.

Sistema de Coordenadas Rectangulares.

El sistema de coordenadas rectangulares utiliza coordenadas XY, definiendo puntos en dos dimensiones, si se necesitara tres dimensiones estas serían XYZ. En la figura a continuación se observa el sistema de coordenadas rectangulares.

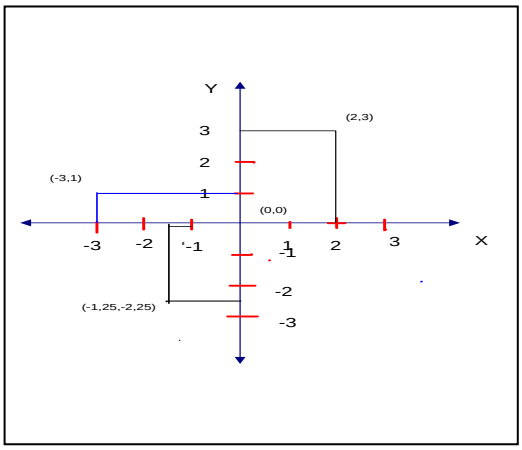


Figura 7. Sistemas de coordenadas

Ejes y Planos

Los ejes son referencias geométricas que se encuentra en el centro de un plano. Para la programación en CNC, es sumamente importante el plano ya que nos sirve como referencia para el posicionamiento.

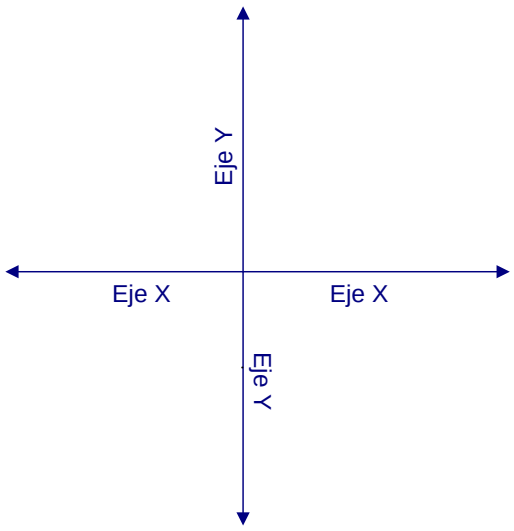


Figura 8. Ejes

Punto de Origen

El punto de origen es el punto donde se intersecan los ejes perpendiculares, donde el punto de origen corresponde a las coordenadas X0Y0, como se indica en la figura.

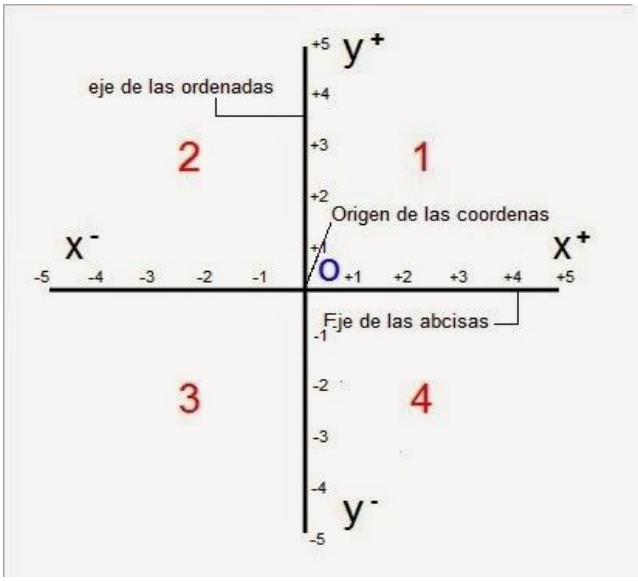


Figura 9. Punto de origen

En la programación CNC es muy importante el punto de origen, este es conocido como el punto cero.

Programa CNC

Sistema de automatización de máquinas a través de una lista de instrucciones ejecutadas por la máquina CNC. Las instrucciones también son conocidas como código CNC, las mismas que deben de tener todos los datos para lograr el maquinado.

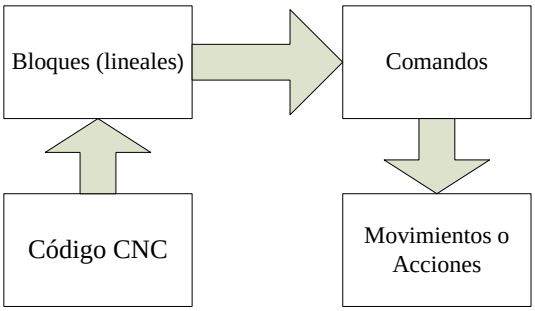


Figura 10. Esquema de un Programa CNC

Estructura de Bloque

Este nos permite comunicarnos con la CNC formando una estructura de bloque de instrucciones, cada carácter alfanumérico tiene un significado y representación propia como se muestra en la figura.

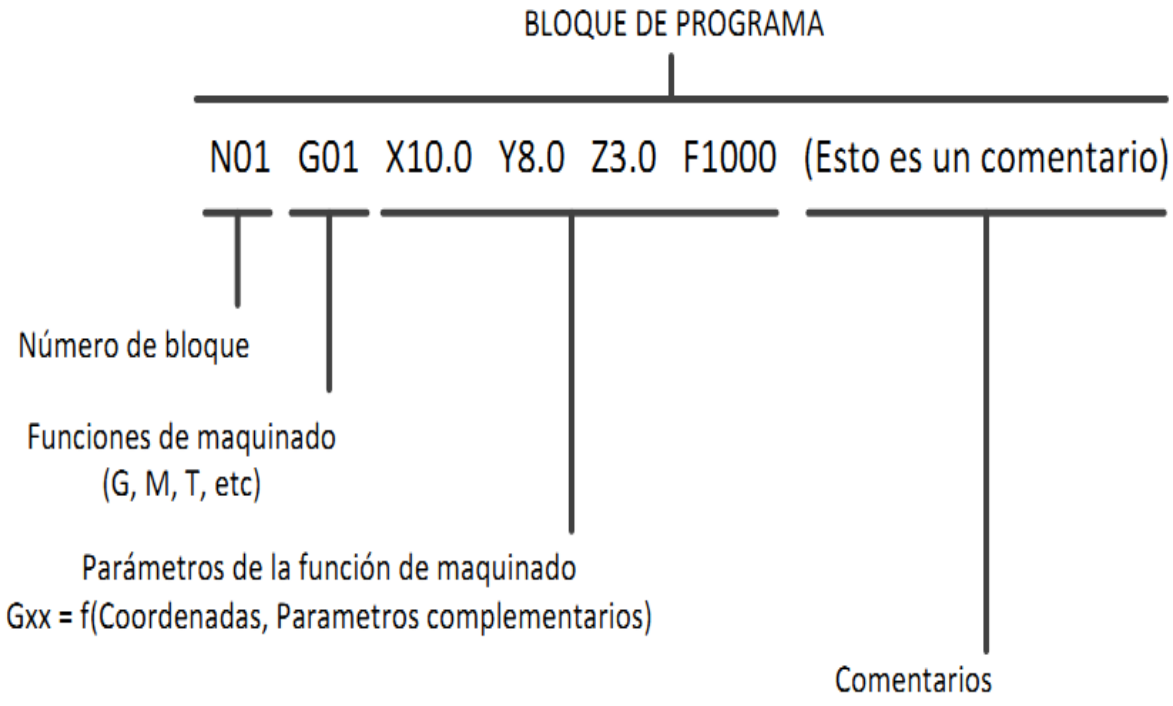


Figura 11. Estructura de bloques de un programa CNC

Fuente: Recuperado de http://wiki.ead.pucv.cl/index.php/Archivo:BloqueDePrograma.png

Funcionamiento de la Mesa CNC

Especificamos los conceptos principales para comprender el funcionamiento de la mesa de corte CNC, prototipo creado para el taller COMECPA. Los pasos para convertir un modelo virtual en un corte de una pieza consiste en seis etapas, las que son efectuadas individualmente o en etapas graduadas.

1. Diseño asistido por computadora (CAD).
2. Fabricación asistida por computadora (CAM).
3. Envío de código G a la mesa CNC.
4. Controlador CNC.
5. Driver de motores paso a paso (Stepper Driver).
6. Movimientos de ejes (Motores paso a paso).

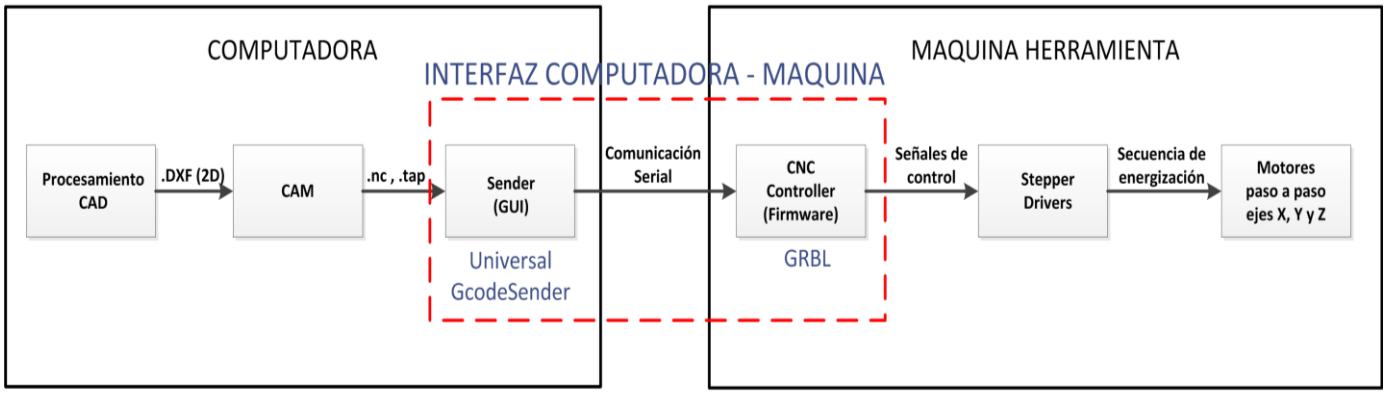


Figura 12. Estructura de funcionamiento de un CNC

Fuente: Recuperado de http://wiki.ead.pucv.cl/index.php/Archivo:DiagBloqueCADCAMCNC3.png

Sistemas CAD, CAM y código G

En el CAD (Computer Aided Design), en español diseño asistido por computadora, el segmento que se necesita maquinar, se lo realiza a través de una computadora, utilizando herramientas de dibujo.

Posteriormente el sistema CAM (Computer Aided Manufacturing), fabricación asistida por computadora, utiliza las especificaciones de diseño para convertirlas en especificaciones de producción. Con estas especificaciones de corte se crea de manera automática el programa de maquinado, el que se puede leer en la máquina con un dispositivo de almacenamiento o también se lo puede transmitir desde una computadora.

Actualmente, las máquinas CNC, gracias a los lenguajes conversacionales y los sistemas CAD/CAM, permiten generar piezas con eficiencia y eficacia, sin necesidad de tener una alta especialización.

El código G es un formato de texto que puede ser escrito a mano o a través de un script, por otro lado, las aplicaciones de CAM se utilizan normalmente para crear el código G. Las extensiones de archivo utilizadas en estos casos son Tap, nc y txt.

Sender

Este software tiene como objetivo enviar, desde la computadora, información del código G hacia la máquina, a través de un protocolo. Sin embargo, esta no es la única posibilidad, puesto que también se puede utilizar dispositivos de almacenamiento tales como tarjetas de memoria o pendrive. Los Software sender han ido evolucionando en el tiempo,en la actualidad ofrecen importantes características entre las cuales están:

Permiten ejecutar movimientos en la máquina de forma manual (código G desde teclado) o automática a través de la carga de un script (programa de mecanizado).

Permiten la visualización del objeto virtual y de la trayectoria que seguirá la herramienta en su recorrido.

Para la mesa CNC se ha escogido el software Universal GcodeSender, el que es una interfaz gráfica con todas las funciones necesarias para realizar la gestión de código G.

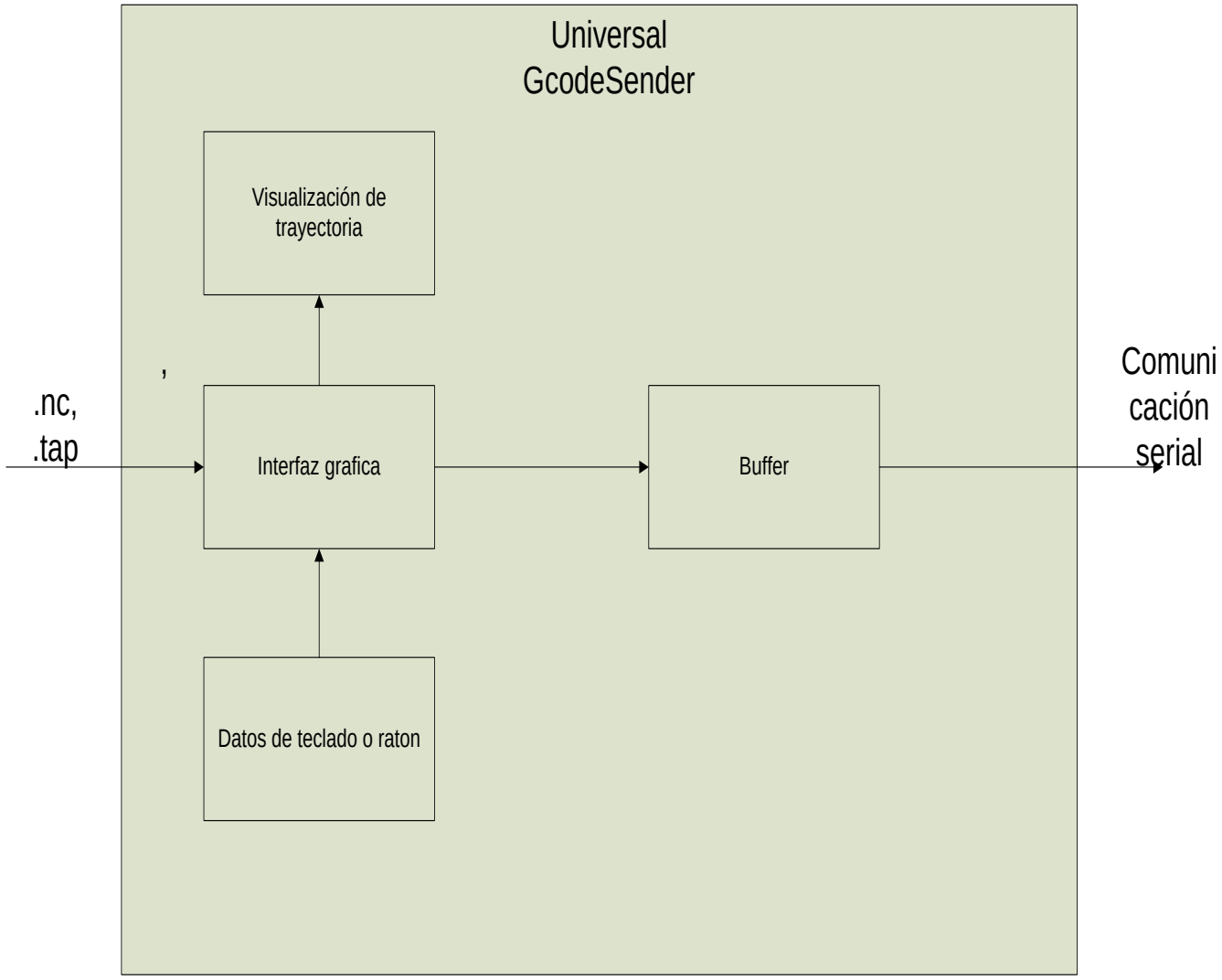


Figura 13. Diagrama de funcionamiento del UniversalGcode Sender

Esquema de funcionamiento.

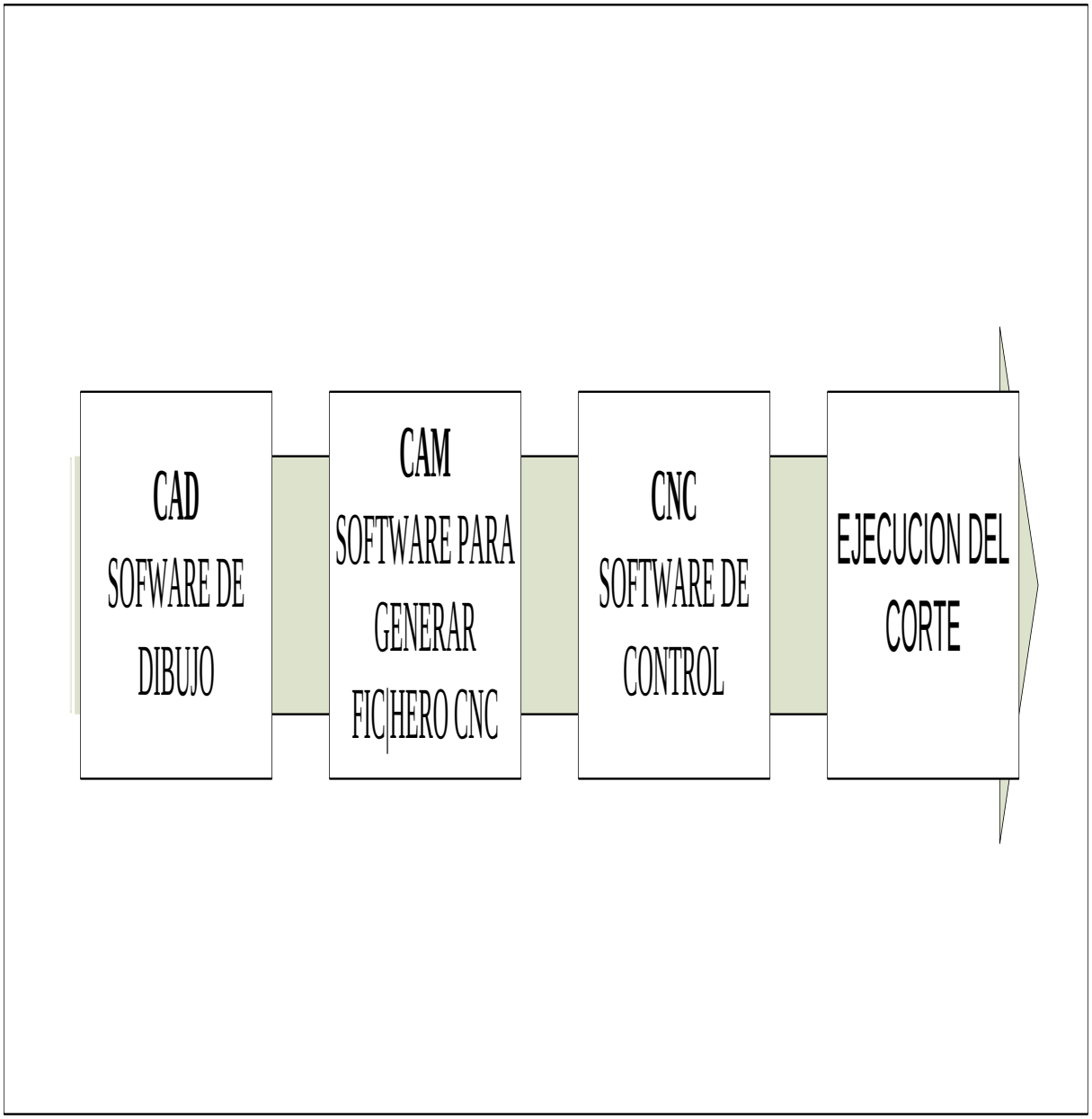


Figura 14. Esquema de funcionamiento

Programa para Control CNC shield.

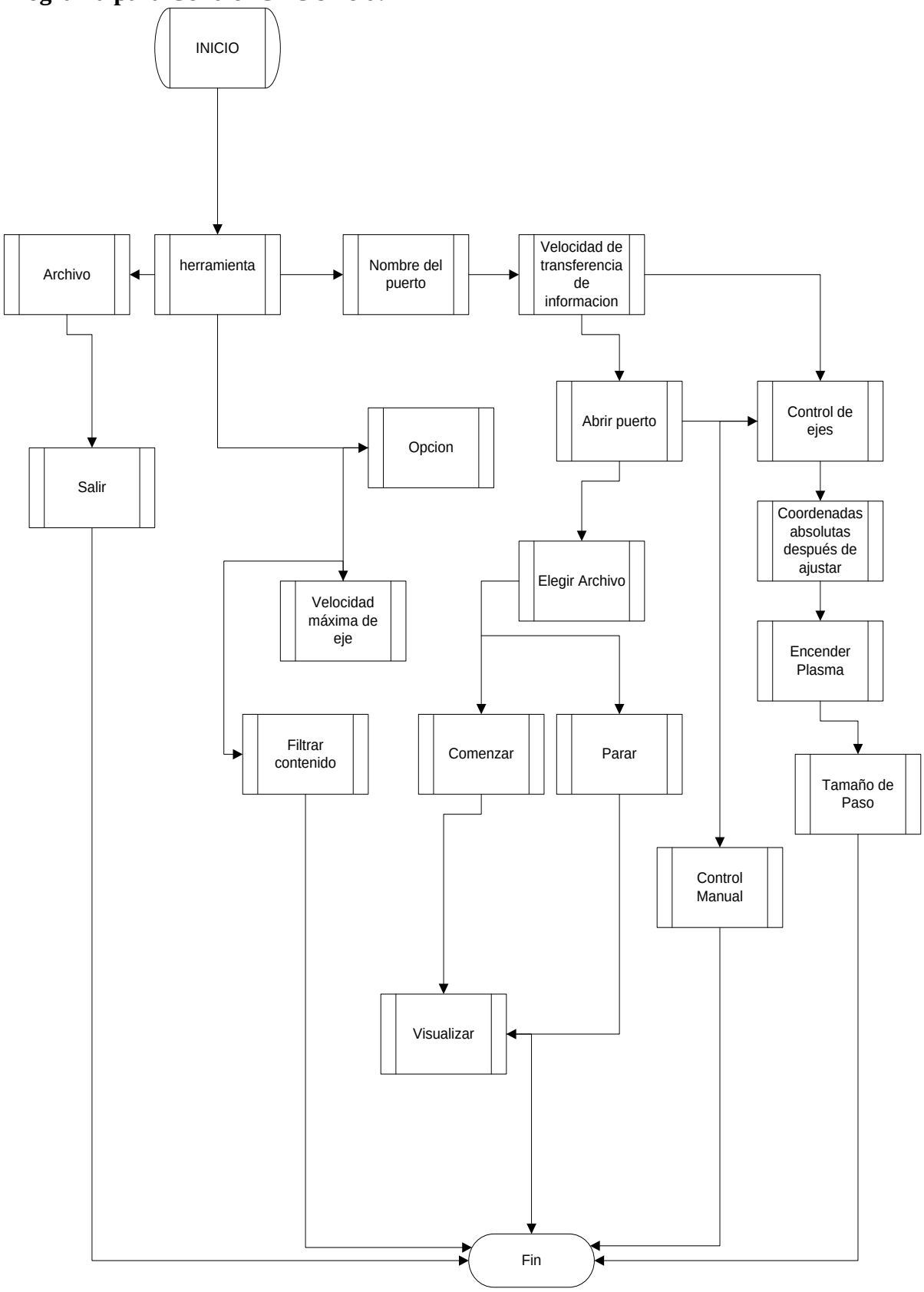


Figura 15. Diagrama de Flujo de Software GRLB

Programa aplicación APP Inventor para calibración.

Este programa permite la conexión del dispositivo con tecnología Android mediante Bloutooth, el mismo fue programado en el Arduino uno. Para hacer posible la conexión mediante la aplicación creada en el APP Inventor 2.

```
int estado=0;

void setup(){
Serial.begin(9600);
pinMode(3,OUTPUT);
pinMode(4,OUTPUT);
pinMode(5,OUTPUT);
pinMode(6,OUTPUT);
pinMode(7,OUTPUT);
pinMode(8,OUTPUT);
pinMode(9,OUTPUT);
pinMode(10,OUTPUT);

}

void loop(){
if(Serial.available()>0){
estado = Serial.read();
}
if (estado =='A')
{
digitalWrite(3,HIGH);
digitalWrite(4,LOW);
digitalWrite(5,LOW);
}
if (estado =='B')
{
digitalWrite(3,HIGH);
digitalWrite(4,LOW);
digitalWrite(5,LOW);

}
if(estado=='C')
{
digitalWrite(3,LOW);
digitalWrite(4,HIGH);
digitalWrite(5,LOW);
}
if(estado=='D')
```

```
{
digitalWrite(3,LOW);
digitalWrite(4,HIGH);
digitalWrite(5,LOW);
}

if (estado =='E')
{
digitalWrite(3,LOW);
digitalWrite(4,LOW);
digitalWrite(5,HIGH);
}
if (estado =='F')
{
digitalWrite(3,LOW);
digitalWrite(4,LOW);
digitalWrite(5,HIGH);
}

if(estado=='G')
{
digitalWrite(3,LOW);
digitalWrite(4,LOW);
digitalWrite(5,LOW);
digitalWrite(6,LOW);
}
if(estado=='H')
{
digitalWrite(6,HIGH);
}

if(estado=='P')
{
digitalWrite(7,HIGH);
digitalWrite(8,LOW);
digitalWrite(9,LOW);
digitalWrite(10,LOW);
}
if(estado=='R')
{
digitalWrite(7,LOW);
digitalWrite(8,HIGH);
digitalWrite(9,LOW);
digitalWrite(10,LOW);
}
if(estado=='J')
{
digitalWrite(7,LOW);
digitalWrite(8,LOW);
```

```
digitalWrite(9,HIGH);
digitalWrite(10,LOW);
}

if(estado=='S')
{
digitalWrite(7,LOW);
digitalWrite(8,LOW);
digitalWrite(9,LOW);
digitalWrite(10,HIGH);
}
}
```

Esquema de conexión de los elementos de control de la mesa CNC.

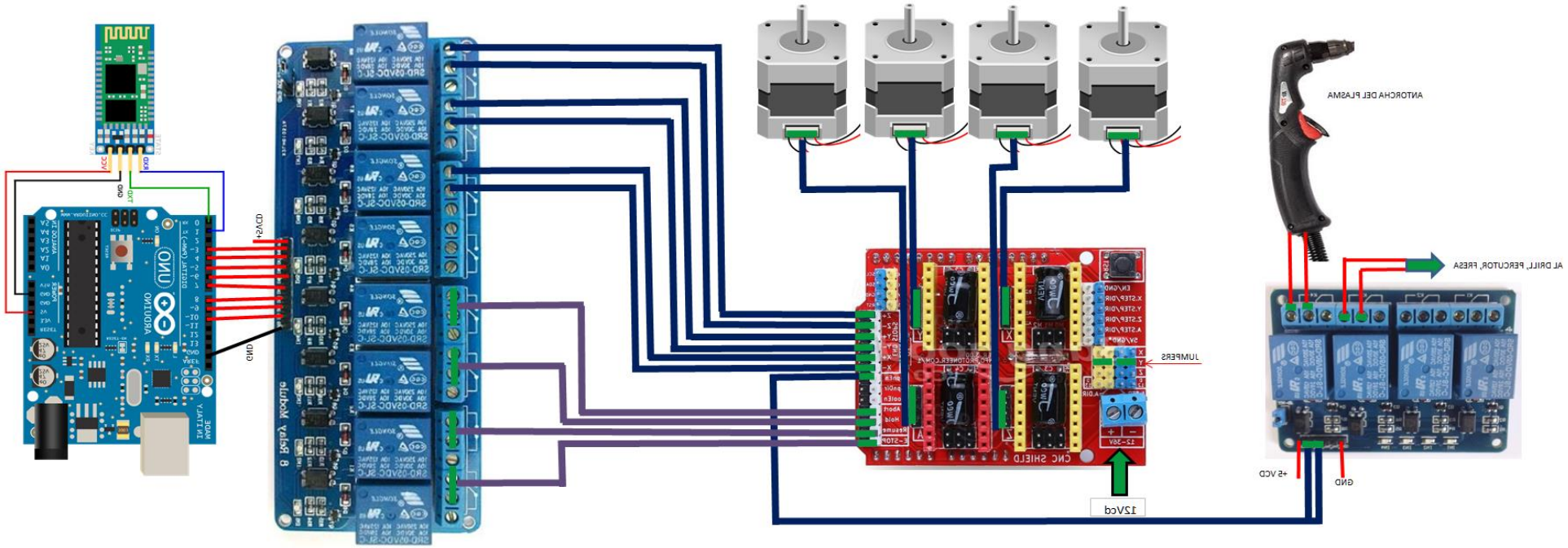


Figura 16. Esquema de conexión de los elementos de la CNC

CAPÍTULO III. IMPLEMENTACIÓN

3.1. Desarrollo.

3.1.1 Sistema de Control.

Control de los Actuadores

En esta mesa CNC, se optó por motores de paso Bipolares Nema 17, porque este modelo y tipo de motor cumple con el torque requerido por la máquina. Este motor es bipolar, tiene dos bobinas y cuatro terminales con los colores como se describe a continuación:

Cableado de Motores PAP.

Tabla 1. Cableado de Motores PAP

Señales y Colores de los Cables		
Fase A	Negro	
Fase/ A	Verde	
Fase B	Azul	
Fase/ B	Rojo	

La Tabla 2, permite reconocer cuál de los cables pertenecen a cada fase o bobina para poder evitar errores y controlar el motor al momento de la conexión.

En la Figura 17, se visualiza la conexión de los cables de acuerdo a los colores antes descritos. Para nuestro proyecto se utiliza el motor Bipolar de cuatro conductores, donde los cables negro y verde pertenecen a una fase (1A & 1B); rojo y azul pertenecen la otra fase. (2A & 2B).

El motor Nema 17, tiene doscientos pulsos por revolución, que posee un paso de 1.8°, que se puede modificar con el driver o controlador de motores.

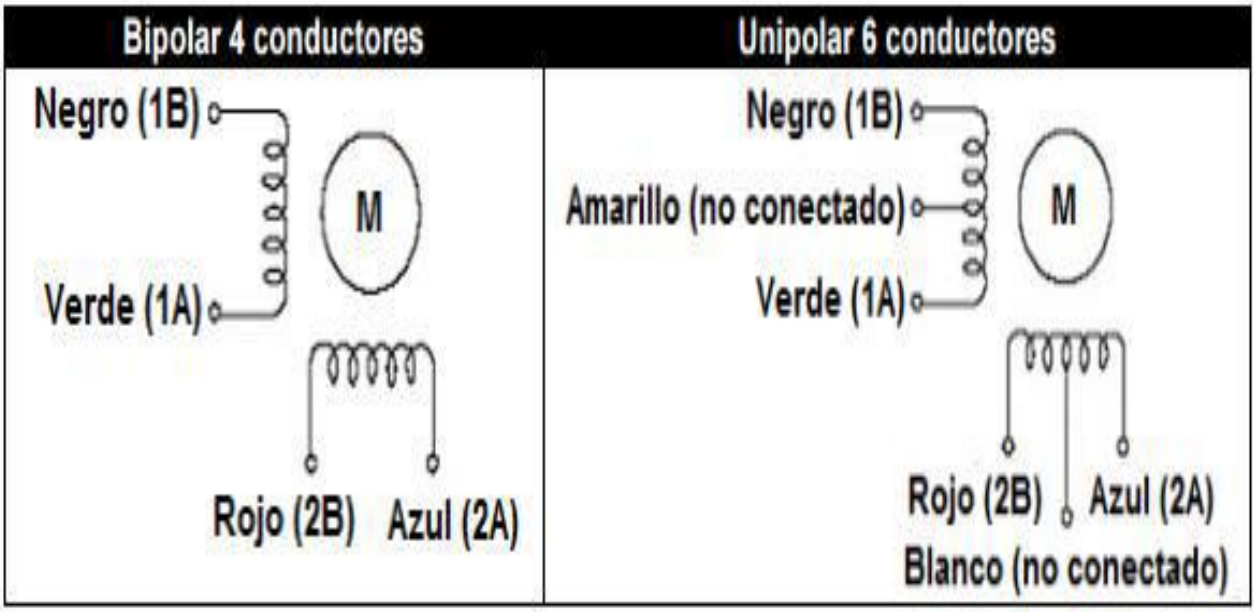


Figura 17. *Conexiones de motores PAP*

Fuente: *Motion Sheneider-Electric (2015)*

Al seleccionar el driver controlador del motor paso a paso, es primordial observar la corriente nominal del motor y su voltaje. En las características técnicas del motor Nema 17 que se encuentra en los manuales del fabricante podemos observar que la corriente nominal es de 1.5 Amperios y el voltaje de trabajo se lo deduce de la curva torque vs velocidad del motor.

I = 1.5 A [$Corriente\ nominal\ de\ trabajo$]

V = 24 V [$Voltaje\ de\ Operacion$]

Driver para controlar Motores Paso a Paso Nema 17.

Con la corriente nominal de trabajo y el voltaje del motor, seleccionamos el driver controlador Pololu A4988. El cual posee un voltaje máximo de 35 voltios, una corriente máxima de 2 amperios con disipador. Este driver posee un potenciómetro que permite regular la corriente de salida. Sus principales características se detallan a continuación:

Características driver A4988

Tabla 2. Características driver A4988

Tensión entrada motor	8-35V
Tensión entrada lógica	3.3-5V
Corriente de salida	1A (Hasta 2A con disipador
Temperatura Ambiente	20-85°C
Resolución Máxima	1/16 Res.

En la Figura 17, se puede observar las conexiones A4988

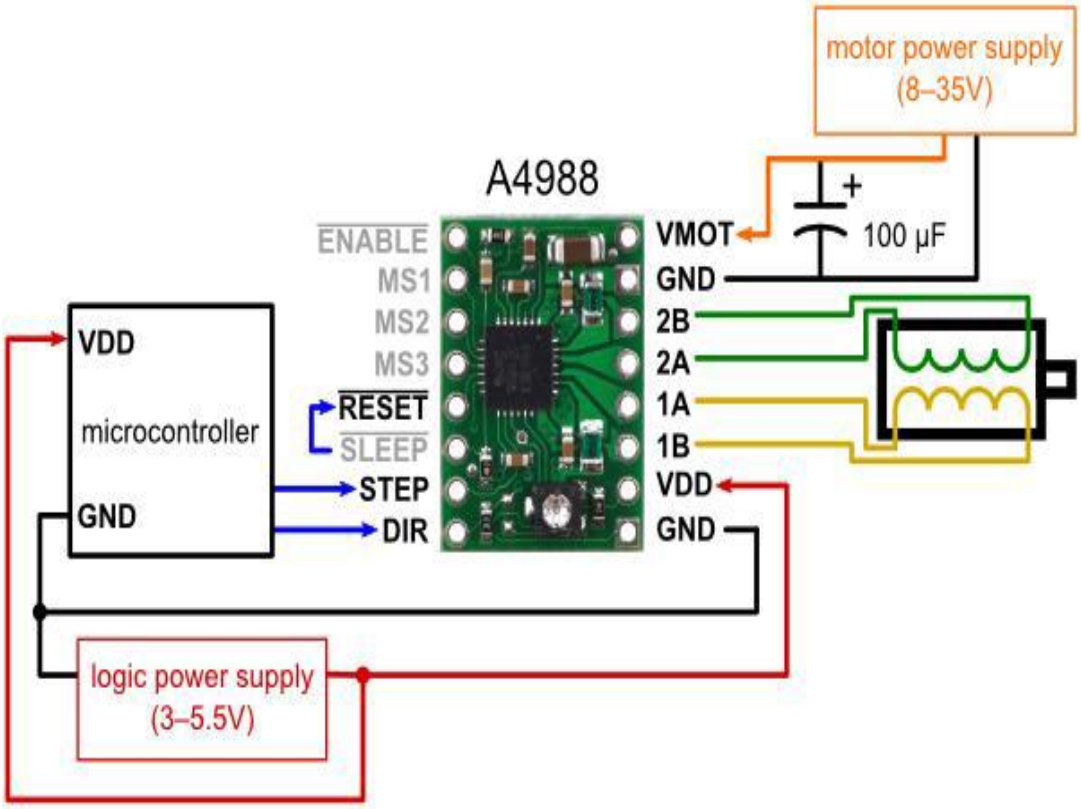


Figura 18. Esquema de conexión eléctrica del A4988

Fuente: Recuperado de https://www.pololu.com/product/1182

Las conexiones eléctricas para este driver son las siguientes:

Se conecta en los pines **VMOT y GND** la alimentación de voltaje para los motores paso a paso, estos pueden variar entre 8 a 35 Voltios dependiendo del voltaje de trabajo del motor.

El fabricante recomienda colocar un capacitor electrolítico de 100uf en paralelo entre la fuente de voltaje y los pines VMOT y GND para la alimentación del driver A4988 se conecta una tensión de 3 a 5.5 voltios, preferiblemente entre los pines **VDD Y GND**

Conexión del motor Paso a Paso Nema 17.

Para conectar el motor se usan los pines **2B, 2A, 1A, 1B**; estos se conectan en las cuatro terminales del motor paso a paso.

En los pines **(2A y 2B)** se conectan una fase o bobina y en los pines **(1A y 1B)** la otra fase de la bobina, configuración para motores paso a paso bipolares.

Entradas de control.

Los pines de control son STEP y **DIR,** donde el pin STEP permite mandar pulsos de control lógico o valores en alto y bajo para poder girar el motor. Cada pulso que se mande equivaldrá a un paso del motor.

El pin DIR permite el giro del motor

El pin **RESET** permite bloquear el pin STEP del driver cuando este está en

Estado alto.

El pin SLEEP permite dejar el driver en bajo consumo, cuando está en nivel alto.

Los pines MS1, MS2, MS3 permiten manejar los micros pasos, aumentando el número de pasos por revolución del motor, para obtener mayor precisión.

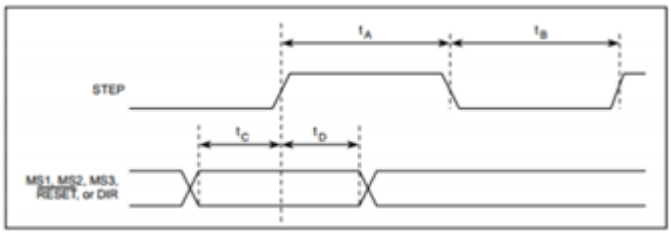


Figura 19. Pulsos de entrada del driver A4988

Fuente: Datasheet Pololu (2015)

En la figura, se puede ver las trayectorias de los pulsos de entradas de los pines STEP, RESET, DIR, MS1, MS2, MS3.

Regulador de intensidad de corriente del Pololu A4988.

El driver A4988 tiene un potenciómetro que permite calibrar la intensidad de corriente de los motores, esto de acuerdo a la corriente nominal que ejerce cada motor. Para esto se usa la siguiente fórmula de acuerdo al Dataheet del A4988

$I = Vref * 2.5$ ecuación # 1

El voltaje referencial (*Vref*) lo obtenemos con un multímetro como se muestra en la figura 19

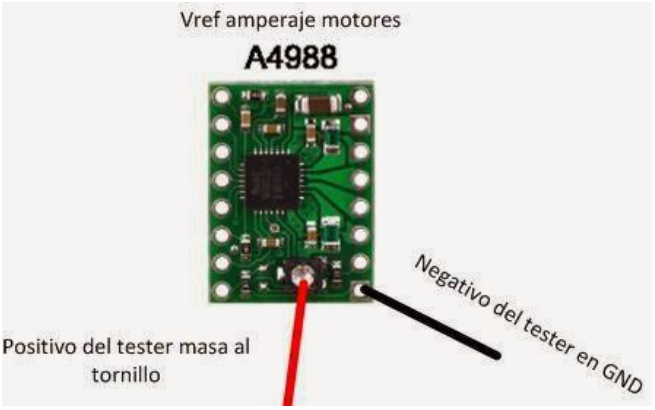


Figura 20. Calibración de la intensidad de corriente del pololu a4988

Fuente: Datasheet Pololu (2015)

El motor Nema 17 opera con corriente nominal de 1.5 amperios.

Con este valor de corriente se calcula el voltaje e referencia, utilizando la fórmula siguiente:

1.5 = $Vref * 2.5$ ecuación # 1

$Vref = 0.6\ V$

Controlador Principal Arduino Uno.

El controlador de la mesa de corte con plasma CNC es la placa Arduino uno, que controla el driver A4988.

En la tabla 4 se detallan las características técnicas del Arduino Uno

Tabla 3. Características Técnicas del Arduino Uno

Microcontrolador	ATmega328
tensión de servicio	5V
Voltaje de entrada	7-12V (recomendado)
Voltaje de entrada	6-20V (límites)
E / S digital	14 (de los cuales 6 proporcionan una salida PWM)
entradas analógicas	6
corriente continua por I / O Pin	40 mA
Pin de la corriente de 3,3 V CC	50 mA
memoria flash	32 KB de los cuales 0,5 KB utilizado por cargador de arranque
SRAM	2 KB
EEPROM	2 KB
velocidad de reloj	16 MHz

Este controlador envía la señal pulso alto o bajo a los pines DIR y STEP de drivr A4988.

Para la conexión de los driver A4988 se utiliza la tarjeta CNC-SHIELD, que permite acoplar los 4 drives A4988 evitando el uso de cables.

Además este se acopla directamente al Arduino uno.

Al conectar el driver A4988 al CNC-SHIELD, el pin macho ENABLE debe conectarce correcamente al pin embra ENABLE, como se indica en la figura 21 .

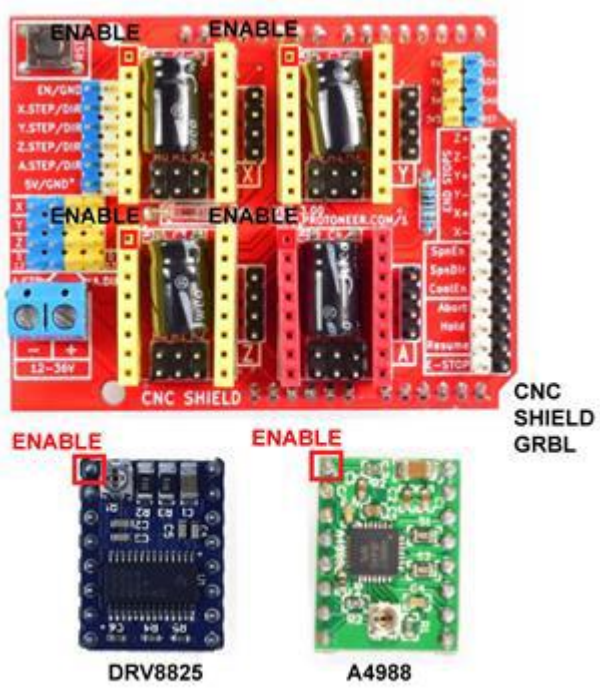


Figura 21. Ubicación correcta de los drivers Pololu A4988 en el CNC-SHIELD

Fuente: Kaiwatechnology (2013)

El CNC-SHIELD se conecta a una fuente de 12 voltios y corriente de 10 amperios , para la alimentación de placa Arduino , con los cuatro motores se octine una coriente de trabajo de 2 amperios.

Selección del software para el control de la mesa CNC.

El proceso de funcionamiento de la mesa de corte con plasma CNC , es la que semuestra en la figura 22.

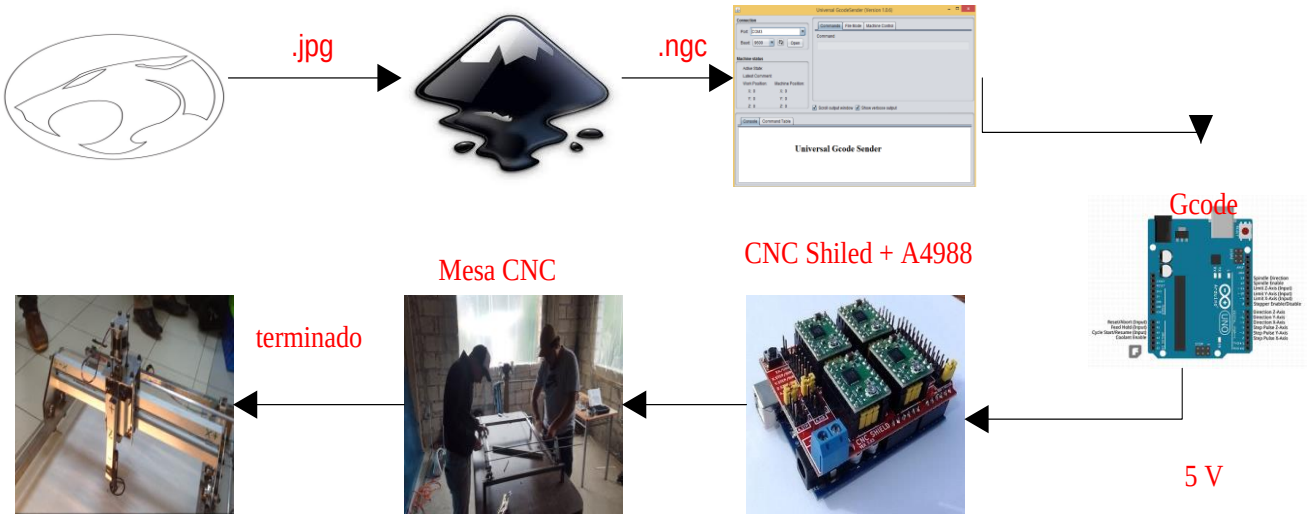


Figura 22. Proceso de funcionamiento de la mesa CNC

Las figuras muestran la secuencia de cómo obtener el corte en la mesa CNC, empezando por la elaboración de dibujo en CAD para poder generar los códigos G.

Estos códigos G son ingresados en el software de control a nuestra PC que controla al microcontrolador Arduino Uno, que además controla el driver A4988 en cargado de hacer girar los motores.

Firmware GRBL.

GRBL es un controlador de CNC de alto rendimiento. Utiliza código abierto para poder ejecutarse en el Arduino. Además, este tipo de firmware lo podemos ejecutar en todo tipo de ordenador.

En la figura 23 se muestra cómo funciona el firmware GRBL

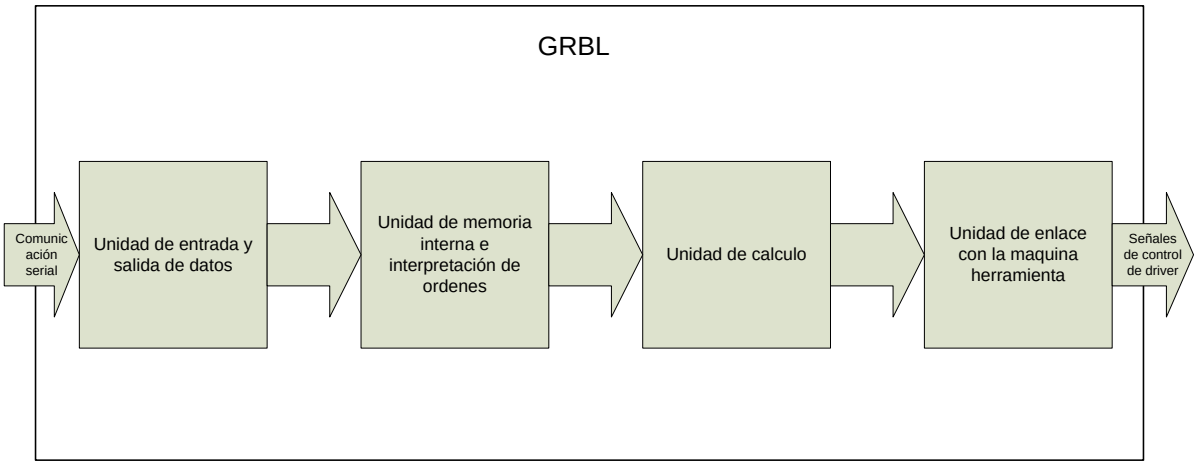


Figura 23. Esquema de funcionamiento de GRBL

Cargar firmware de GRBL en Arduino.

Antes de realizar cualesquiera tipos de conexión, se debe verificar que el Arduino uno este correctamente acoplado al CNC-SHIELD.

Antes de realizar las conexiones, se debe verificar que el Arduino Uno este correctamente acoplado al CNC-SHIELD

GRBL es un programa multiplataforma por lo que se puede ser utilizado tanto en Windows, Mac o Linux.

Para cargar el firmware al Arduino, se conecta el Arduino Uno a cualquier puerto del computador para determinar en qué puerto COM se asignó el Arduino Uno

Descargamos de la página de Arduino, el IDE de Arduino que sirve para programar el Arduino. Abriendo el IDE se descarga en el Arduino el Firmware GRBL.

El GRBL es el que ejecuta el código G, mandando señales al driver y por ende a los motores Paso a Paso Nema 17.

Interfaz Universal GcodeSender

Este software está diseñado para enviar el código a dispositivos de control numérico de 3 ejes, envía a la mesa cualquier código G.

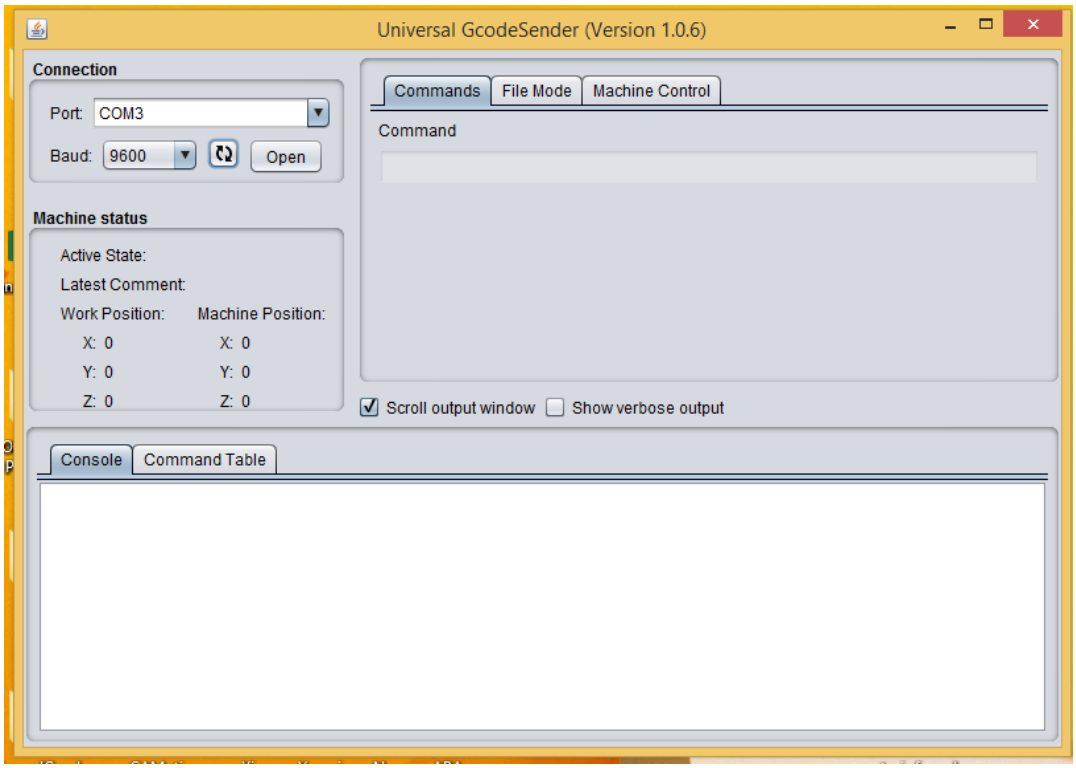


Figura 24. Interfaz de usuario

Con la ayuda de esta interfaz de usuario, podemos configurar y calibrar la mesa CNC, de acuerdo a los requerimientos necesarios.

Seleccionamos el puerto donde está conectado el Arduino,

El universal Gcode Sender, interpreta el G-code, pero para modificar se envía comandos especiales, el comando "$$", nos muestra una lista de 22 parámetros que se pueden configurar.

```
$0=1000 (x, step/mm)
$1=1000 (y, step/mm)
$2=1000 (z, step/mm)
$3=100 (step pulse, usec)
$4=240 (default feed, mm/min)
$5=250 (default seek, mm/min)
$6=192 (step port invert mask, int:00011100)
$7=50 (step idle delay, msec)
$8=5 (acceleration, mm/sec^2)
$9=0.050 (junction deviation, mm)
$10=0.100 (arc, mm/segment)
$11=25 (n-arc correction, int)
$12=3 (n-decimals, int)
$13=0 (report inches, bool)
$14=1 (auto start, bool)
$15=0 (invert step enable, bool)
$16=1 (hard limits, bool)
$17=0 (homing cycle, bool)
$18=0 (homing dir invert mask, int:00000000)
$19=240 (homing feed, mm/min)
$20=250.000 (homing seek, mm/min)
$21=100 (homing debounce, msec)
$22=1.000 (homing pull-off, mm)
```

Figura 25. Parámetros de configuración grbl

Los comandos $0, $1; $2 representa los pasos por milímetro de la máquina. CNC en los ejes X, Y, Z; Se puede calibrar estos valores manualmente o mediante un simple cálculo que se muestra en la ecuación 2:

$P = Pm/p$ Ecuación # 2

p = 2 mm/rev [$Paso\ del\ eje\ roscado\ X, Y, Z$]

Pm = 200 $Step/rev$ [$Paso\ por\ revolucion\ de\ los\ motores\ de\ paso$]

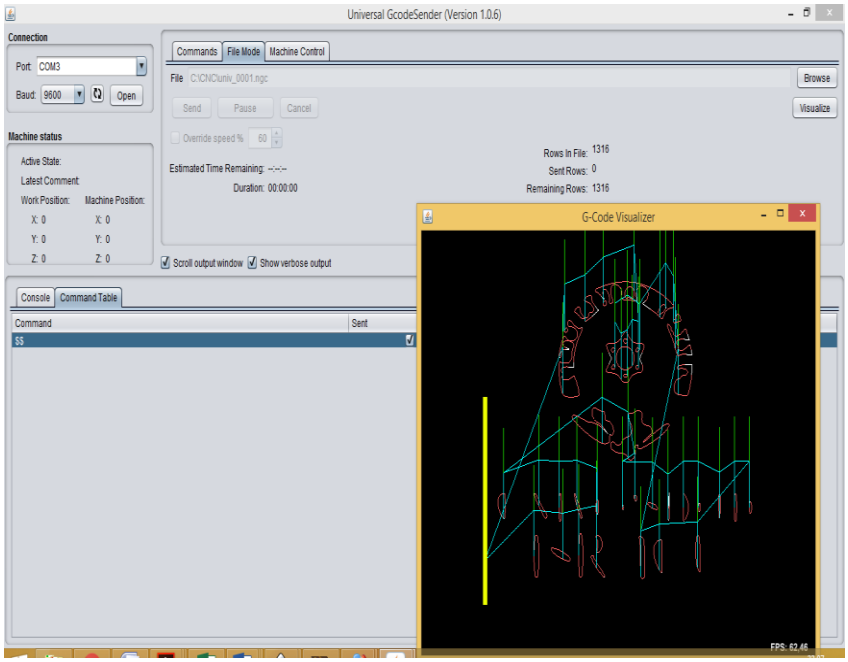


Figura 26. Imagen del universal gcodeSender

Para la instalacion del Universal Gcode Sender , es necesario la intalacion de Java en nuestra computador.

Creación de G-CODE: Programa INKSCAPE

Inkscape la herramienta de dibujo que utilizamos para gráficos vectoriales de libre acceso utiliza el formato SVG (Scalable Vectorial Graphics), también puede importar y exportar formatos como (JPG, PNG; etc.).

Para la creación de un código G, de una imagen para la mesa CNC, se debe vectorizar la imagen para obtener los desplazamientos del plasma dado en puntos o coordenadas.

A continuación, describimos los pasos básicos para obtener un código G.

Cargamos la imagen en el programa INKSCAPE, escogemos el tamaño de la imagen para graficar según los requerimientos.

Vectorizamos la imagen ubicando los puntos de referencia que tendrá la herramienta de corte para nuestra mesa utilizamos plasma. Asignamos las unidades de la altura del relieve del eje Z.

En la biblioteca de herramientas de INKSCAPE, en la casilla de extensiones, escogemos el tipo de herramienta de corte a usar.

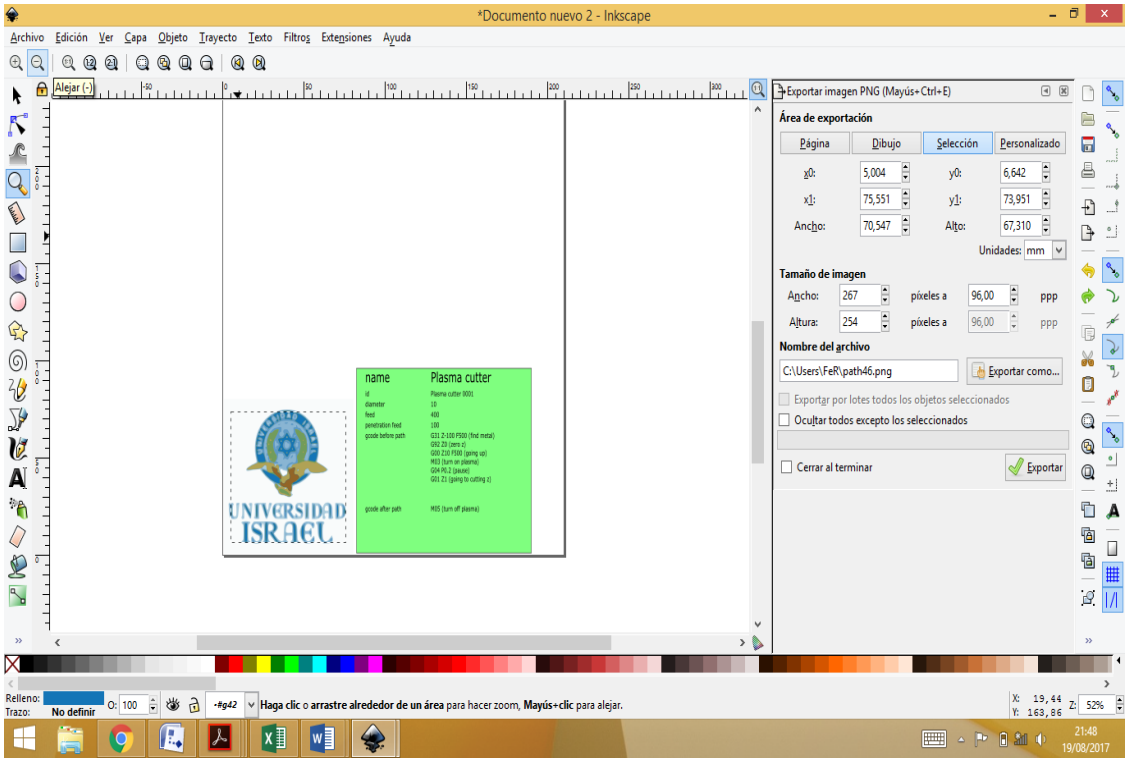


Figura 27. Programa INKSCAPE

Se selecciona la imagen con el cursor en trayecto, en la opción de extensiones se selecciona G-CODE, se traza el trayecto y automáticamente aparece una ventana donde se pondrá el nombre del archivo.

En la opción de trayectoria de G-CODE, se forma el recorrido de la imagen pudiéndose notar como es la trayectoria que realizaría la herramienta de corte al momento de realizar el corte en el material de trabajo.

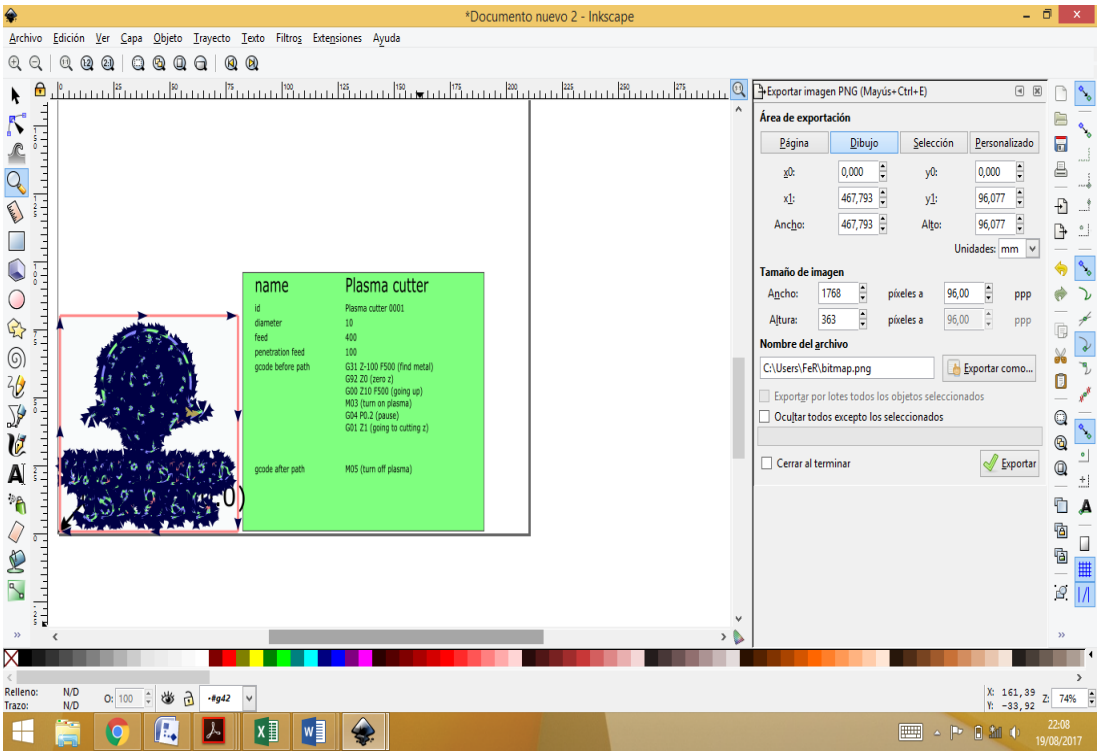


Figura 28. Imagen vectorizada en G-code

Con la imagen en G-CODE, se procede con la ejecución del programa Universal Gcode Sender, conectando el Arduino uno con las configuraciones necesarias, se selecciona el archivo y se lo carga. El sistema automáticamente mandará las órdenes y se ejecutará el comando en G-CODE, para realizar el corte de la imagen.

Simulador de corte CAMotincs.

Este programa nos permite realizar una verificación simulando el corte, antes de enviar al Universal Gcode Sender.se extrae el archivo que se vectorizó en INKSCAPE, se envía Start y este simula como se va realizar el corte.

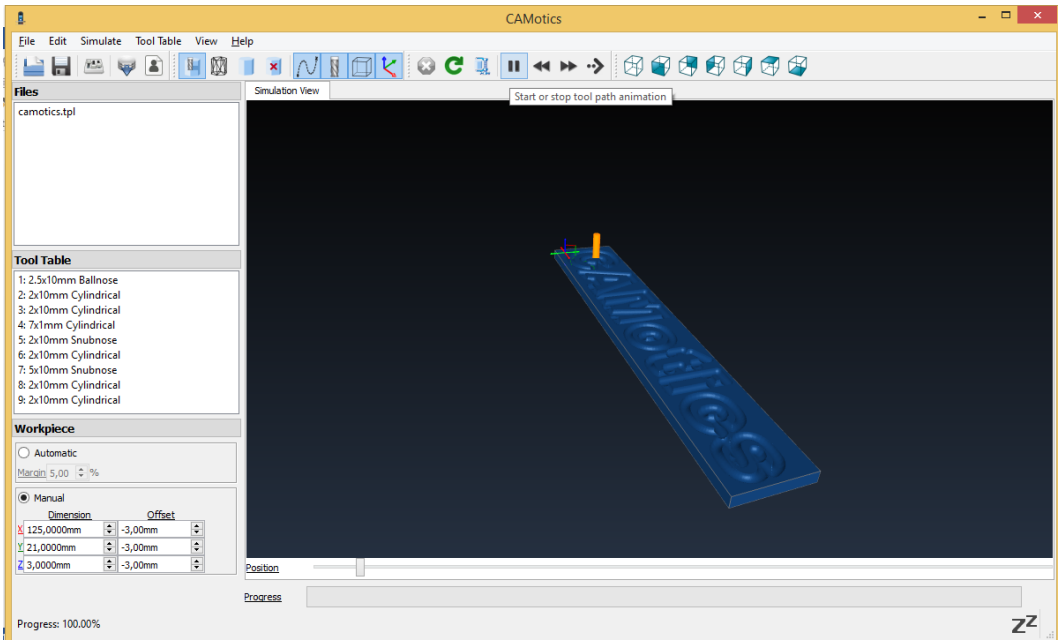


Figura 29. Imagen en camotics

App inventor 2

Es un software creado por la compañía Google labs, con la finalidad de generar aplicaciones en Android. Este software permite a los usuarios, de manera gratuita, enlazar bloques con herramientas básicas y de ésta manera crear aplicaciones en archivos APK.

Se desarrolló la siguiente App para la calibración de la CNC

Figura 30. Visualización de la aplicación

Activa la habilitación del botón del BLUETOOTH bluetooth (ListPicker1), 

que apenas se abre la aplicación, sin que se presione ningún botón (Before Picking), el dispositivo busca la lista de equipos con bluethooth conectados en el entorno (bluetooth client)  se genera una lista de direcciones con los nombres (AdressAndNames, para que luego se seleccione con cual dispositivo conectarse.

Al presionar el botón de Bluetooth de la BLUETOOTH aplicación llama al listado generado de clientes en el ítem anterior y permite la selección del dispositivo con el cual conectarse. (list picker selecction)

Al presionar el botón **CALIBRACIÓN**, 

la aplicación abre una nueva pantalla, en este caso la pantalla numero 2

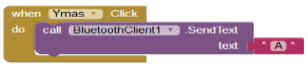
Al presionar el botón **OPERACION**, 

la aplicación abre una nueva pantalla, en este caso la pantalla numero 3

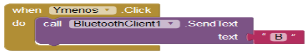
Figura 31 .Visualización de pantalla de Calibración

Al presionar el botón 

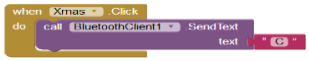
Y+ se abre el cliente del bluetooth habilitado y se le envía como dato tipo 

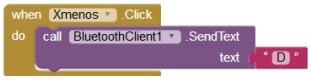
texto la letra **“A”** 

Al presionar el botón Y- se abre el cliente del bluetooth habilitado y se le envía como dato tipo texto la letra 

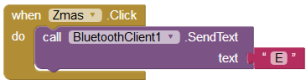
“B” 

Al presionar el botón X+ se abre el cliente del bluetooth habilitado y se le envía como dato tipo texto la letra 

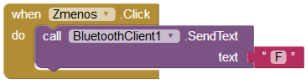
“C” 

Al presionar el botón X- se abre el cliente del bluetooth habilitado y se le envía como dato tipo texto la letra **“D”** 

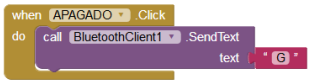
Al presionar el botón Z+ se abre el cliente del bluetooth habilitado y se le envía como dato tipo texto la letra 

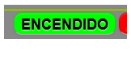
“E” 

Al presionar el botón Z+ se abre el cliente del bluetooth habilitado y se le envía como dato tipo texto la letra 

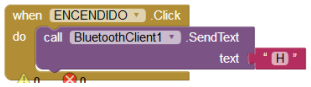
“F” 

Al presionar el botón 

APAGADO se abre el cliente del bluetooth habilitado y se le envía como dato tipo texto la letra **“G”** 

Al presionar el botón 

ENCENDIDO se abre el cliente del bluetooth habilitado y se le envía 

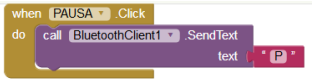
como dato tipo texto la letra **“H”** 

Al presionar el botón REGRESAR se llama a la pantalla 1 (pantalla principal)

Figura 32.Visualizacion de pantalla de parar

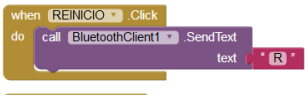
Al presionar el botón PAUSA 

se abre el cliente del bluetooth habilitado y se le envía como dato tipo texto 

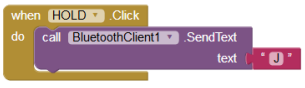
la letra **"P"** 

Al presionar el botón REINICIO se abre el cliente del bluetooth habilitado y se le envía como dato tipo texto 

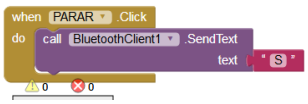
la letra **"R"**



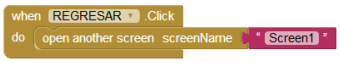
Al presionar el botón HOLD se abre el cliente del bluetooth habilitado y se le envía como dato tipo texto la letra 

"J" 

Al presionar el botón PARAR se abre el cliente del bluetooth habilitado y se le envía como dato tipo texto la letra **"S"** 



Al presionar el botón REGRESAR se llama a la pantalla 1 (pantalla principal)



3.2. Implementación

El proceso de implementación comenzó con la construcción del armado de la estructura de la mesa de corte en metal específicamente con tubo cuadrado de 25 mm, en el eje Y, y la parte del eje X construido con dos tubos cuadrados de 25mm de aluminio para alivianar el peso, el eje Z fue construido en placas de aluminio de 2mm de espesor, con ejes lineales de 8 mm de diámetro y rodamientos lineales.

3.3. Pruebas de funcionamiento

3.4. Análisis de resultados

A continuación, mostraremos los resultados obtenidos, en donde se detallan los parámetros de operación de la mesa para corte con plasma CNC. Las diferentes pruebas realizadas y la selección de los actuadores, de sus partes mecánicas y de control entre los que se incluye el software libre a utilizar. También un análisis económico donde se cotizan los precios de los materiales para la construcción de la mesa de corte con plasma CNC, al fin de establecer la rentabilidad de construir este tipo de máquina en el medio.

Resultados de los Parámetros de Operación:

Se definen los parámetros de operación de ciclos de trabajo de la mesa de corte de plasma CNC.

Selección de Actuadores

Para seleccionar los motores a utilizar en la mesa de corte con plasma CNC se escogió el eje más crítico, que recibía la mayor carga requiriendo un mayor torque. Por lo que se

escogió al Eje Y, debido a que tiene un mayor desplazamiento y porque puede desplazarse en forma total en los dos sentidos durante la operación.

Los actuadores que se seleccionaron para la mesa CNC, son los motores pasos a paso Nema 17 (Figura 33) que cuentan con una alta precisión de movimiento y cumple con los requerimientos de la mesa CNC

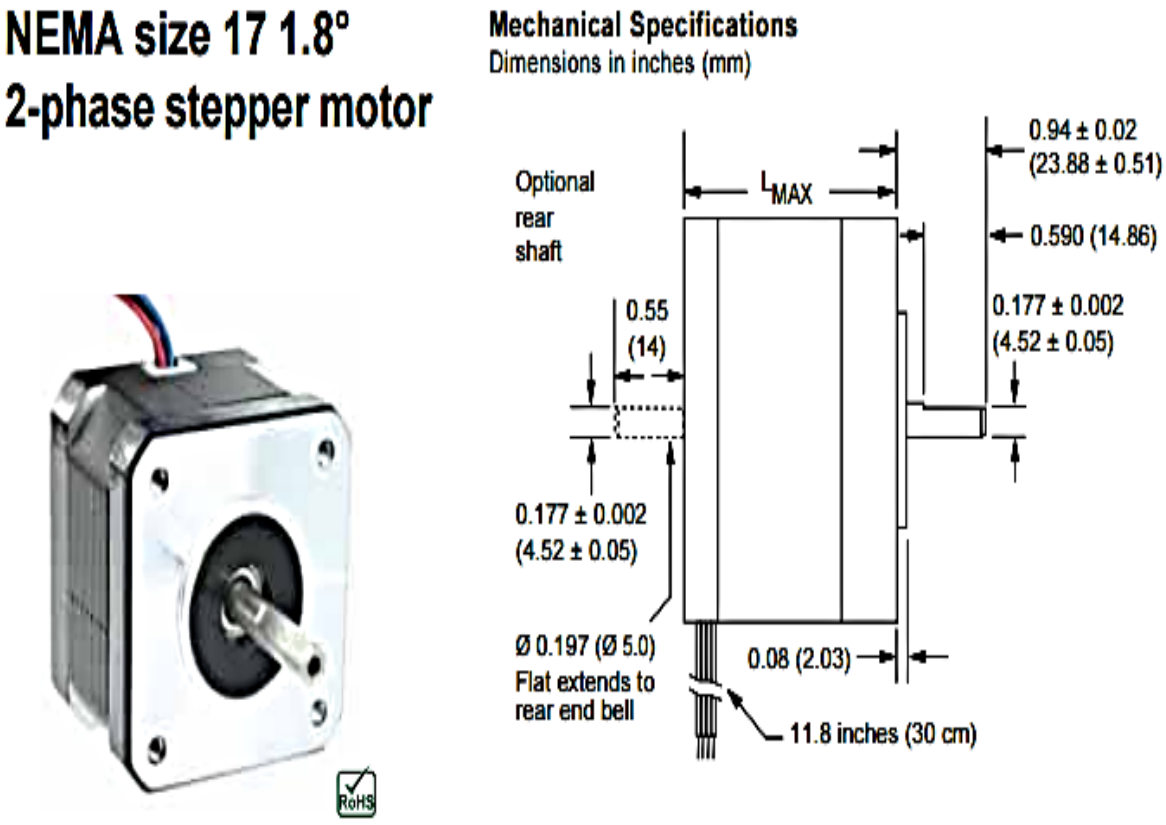


Figura 33. Motor PAP Nema 17

Fuente: Motion Schneider-Electric (2013)

3.4.1 Resultados del Sistema de Control

Control de actuadores:

Para poder controlar los actuadores se seleccionó la pala Arduino Uno, que utiliza un microprocesador ATMEL, junto a la placa SHIEL-CNC.

El SHIEL-CNC. Ayuda a evitar el uso de cables. En esta placa se acoplan los drivers A4988 los cuales se encargan de controlar los motores paso a paso. Considerando el

voltaje de aceptable de trabajo del A4988 que es de 8-35 voltios con una corriente máxima de 2 amperios.

A continuación, se detalla la secuencia para controlar los actuadores de la Máquina fresadora CNC.

En la siguiente figura se detalla la secuencia para el control de los actuadores en la mesa CNC.

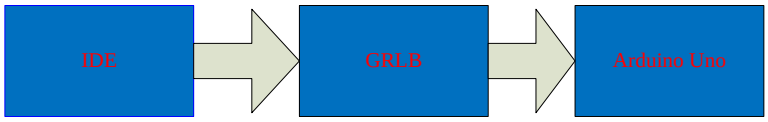


Figura 34. Secuencia para controlar actuadores en la mesa cnc

Firmware, Interfaz de usuario.

En este proyecto se utilizó el firmware GRBL ya que este es un código completo y acepta código G. mandando señales al driver y por ende a los motores, descargando de la página de Arduino el IDE de Arduino.

Para la interface gráfica de usuario se seleccionó el Universal Gcode Sender, que es compatible con Arduino e interpreta los comandos del código G.

Para la creación del código se utilizó el programa INKSCAPE, el mismo que es un programa de libre acceso, este vectoriza las imágenes de forma rápida, exacta y las trasforma en código G, dependiendo de la herramienta a utilizar.

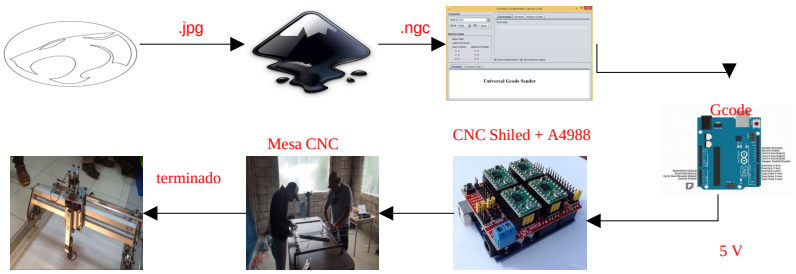


Figura 35. Secuencia de corte mesa CNC

PRUEBAS

Tabla 4. *Pruebas de calibración hardware*

PRUEBAS DE CALIBRACIÓN HARDWARE

DRIVER:	A4988
PRESENTACION:	Pines estándar para socket SHIEL CNC Arduino Uno
MÉTODO DE CALIBRACIÓN:	Manual con verificación del valor calculado.
AJUSTES:	Prueba/Error

VALOR (Voltios)	RESULTADO OPERACIÓN DEL MOTOR
0,30	Valor fuera de rango
0,45	Zumbido de motor sin moverse
0,60	Zumbido fuerte del motor, solo pulsa
0,75	baja el zumbido, gira normal, pierde pasos, bajo torque
0,90	operación normal, no hay perdida de pasos, bajo torque
0,96	operación normal, buen torque
1,00	operación normal buen torque, recalienta driver
1,05	pierde torque, aumenta recalentamiento
1,30	Valor fuera de rango

Tabla 5. Calibración de firmware

PRUEBAS DE CALIBRACIÓN FIRMWARE	
FIRMWARE:	GRLB Parámetros
PROGRAMA:	Universal G-Code Sender
MÉTODO DE CALIBRACIÓN:	Seteo de los valores dentro de los rangos
AJUSTES:	Prueba/Error

PARAMETROS: $0 ; $1 ; $2	
VALOR (STEP/MM)	**RESULTADO OPERACIÓN DEL MOTOR**
10	movimiento continuo, sin pérdida de pasos
30	movimiento continuo, sin pérdida de pasos
50	movimiento continuo, sin pérdida de pasos
142	movimiento continuo, sin pérdida de pasos
200	movimiento continuo, sin pérdida de pasos
400	movimiento continuo, sin pérdida de pasos
800	perdida ocasional de pasos
1500	perdida de pasos secuencial
10000	vibración y recalentamiento del motor

OBSERVACIONES: valores idóneos de operación; entre 10 y 800 pasos por milímetro. El valor calculado de 142 pasos por milímetro está dentro del rango de operación adecuada; este será el seleccionado para trabajar. Nota: para el desarrollo práctico de la calibración se utiliza 200 pasos por milímetro, ya que de acuerdo al motor existente, esto representa una vuelta de 360 grados cada milímetro lineal de desplazamiento, permitiendo una mejor visualización durante las pruebas de calibración. Cabe recalcar que para la operación el valor debe estar sitiado en 142pasos por milímetro, caso contrario la escala real cambia.

Análisis Económico.

Con el fin de realizar un análisis económico comparativo entre la manufactura de la mesa y la importación de la misma, se registraron cada uno de los costos de la elaboración del proyecto, en base a los materiales, accesorios y equipos seleccionados, además se analizó costos de mano de obra y ensamblaje.

Es importante recalcar que los precios utilizados como referentes para el análisis son fluctuantes, ya que varían con factores como el mercado o el proveedor.

A continuación se enlista los materiales utilizados en la fabricación de la mesa CNC:

Tabla 6. Costos de Materiales y Equipos para la mesa CNC

Costos de materiales y equipos para la mesa para corte con plasma CNC			
Descripción	**Cantidad**	**Precio Unitario**	**Precio total**
Arduino Uno	1	10.71	$ 10,71
Kit Cables 65 Jumper	1	1.34	$ 1,34
Pernos 8x40 acero	20	0.16	$ 3,20
Tuercas 8M acero	40	0.10	$ 4,00
pernos 12x40	18	0.45	$ 8,10
Tuercas 12mm acero	18	0,15	$ 2,70
Varillas roscadas ¼	5	1.1	$ 5,50
Varillas roscadas 5/16	3	1.45	$ 4,35
Macho Hembra 40	1	1.79	$ 1,79
Juego de cables largos	1	2.23	$ 2,23
HC-06 Bluethooth	1	8.04	$ 8,04
CNC SHIELD V3	1	8.93	$ 8,93
Driver paso a paso A4988	5	3.57	$ 17,86
Nema 17 Motor Pasos	1	14.29	$ 14,29
Arduino Uno Case	1	4.46	$ 4,46
CTO.ROD.PLANA 1/8	2	0.3	$ 0,30
CTO.ROD.PLANA 5/32	1	0,6	$ 0,60
CTO.ROD.PLANA 3/16	1	1	$ 1,00
CTO.ROD.PLANA GALV 5/16	1	2.5	$ 2,50
CTO.ROD.PRES. GALV 5/16	1	1.5	$ 1,50
CTO.ROD.PRES.3/16 NEGRA	1	0.6	$ 0,60
ROD,PLANA HIERRO 5/16	1	2	$ 2,00
TUBE CONTACT S.C.R O,35 X 1,125	4	0,82	$ 3,28
NOZZLE SLIP TYPE , 625	1	22,85	22,85
DEFLECTOR	1	14,94	14,94
RADNOR 16 OUNCE JAR NOZZLE GEL	1	5,73	5,73
PMX 30 CSA INTERNATIONAL HAND SYSTEM	1	1889,35	1.889,35
		SUBTOTAL	$ 2.042,15
		IVA	$ 245,06
		TOTAL	$ 2.287,21

Luego del análisis de costos, es necesario aclarar que cada uno de los materiales utilizados para la elaboración tienen un precio referencial sumamente elevado ya que fueron comprados al por menor, al momento de montar un proyecto de elaboración de varias mesas CNC, la compra de los materiales se realiza al por mayor, lo que reduce la tabla de costos y obteniendo un precio final de la mesa mucho más económico.

Rentabilidad de la mesa CNC.

Para establecer una rentabilidad del proyecto se compara con las existentes en el mercado, tienen precios elevadísimos debido a tasas e impuestos de importación. En cotizaciones simples, en páginas extranjeras comerciales se ofrece una mesa CNC por alrededor de $ 1.000, pero esto no incluye el envío hasta Ecuador, garantía, ni mantenimiento.

En el caso de este proyecto, con insumos de bajo costo, el valor de la mesa sería de 2, 87,21 dólares americanos incluido el plasma pero esta es una mesa multi-herramientas y puede usar diferentes herramientas de corte y ese valor incluiría garantía de funcionamiento y mantenimiento, ofertado por el elaborador del proyecto.

Como se puede observar, la importación de las mesas CNC, ya que no existe su elaboración en Ecuador, tienen un costo elevado, ya que incluye transporte e impuestos. Al momento de fabricar la mesa, como es el objetivo del proyecto, los costos se abaratan y se convierte en una perfecta oportunidad para las industrias pequeñas de adquirir una herramienta indispensable para el día a día de su negocio.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones.

Una vez finalizada la investigación, luego del análisis y discusión de los resultados y en función a los hallazgos más representativos, se establecen las conclusiones para el desarrollo de la tarjeta de control para una mesa mediante CNC en función a cada uno de los objetivos específicos que se establecieron en la investigación.

Con respecto al primer objetivo específico que consistió en establecer el método de corte más adecuado acorde a la necesidad del usuario, se halló que este consistió en utilizar una alimentación del plasma a 220 V, 18 A, con presión de 95psi. Así mismo con respecto a la posición perpendicular de la antorcha, esta debe estar ubicada a 0.8mm de separación entre el electrodo y placa acortar, con una velocidad de Arrastre de 400mm/seg, para planchas de acero negro de 1mm, 200mm/seg, para planchas de acero negro de 3mm, 50mm/seg, para planchas de acero negro de 6mm.

En función al objetivo dos, se utilizó la tarjeta electrónica del fabricante de Arduino logrando programar la tarjeta de control automático para realizar los movimientos sincronizados de los ejes X, Y, Z, instalando los GRLB de Arduino. Además, a través del envió del código G se consiguió comprobar el correcto funcionamiento de los motores paso a paso, permitiendo realizar el corte de la mesa CNC.

Así mismo, para el tercer objetivo el cual se refiere a la programación para realizar los movimientos requeridos de la herramienta, luego del análisis de diferentes programas, se determinó que los más apropiados eran el Inkscape, ya que éste permitió la creación del código G a través de una imagen, grafico o dibujo, para la conversión del código G en movimiento de los motores, se utilizó el GRBL, pues esta es una librería propia de Arduino creada exclusivamente para CNC, finalmente con la ayuda del UniversalGcode Sender o interfaz de usuario se logró producir el proceso de Corte en la CNC , cabe destacar que estos programas son fáciles de utilizar y por formar parte de la familia de

software libre no necesitan licencias para el funcionamiento y facilitaron el complimiento de este objetivo como fue programar la CNC.

El objetivo cuarto se refiere al diseño de la interfaz de comunicación inalámbrica, el cual se logró mediante el diseño de una aplicación inalámbrica utilizando app inventor 2, esta aplicación funciona en un dispositivo móvil con sistema operativo Android, conectándose por bluetooth. Se utilizó app inventor 2 por las facilidades que da, para la creación de aplicaciones para dispositivos con sistema Android, la que permite de manera gratuita enlazar bloques y herramientas básicas y crear los archivos APK para los dispositivos móviles con sistema Operativo Androi.

Luego de las experiencias adquiridas en la construcción del presente proyecto, se puedo verificar el correcto funcionamiento de la mesa CNC cumpliendo con el objetivo número cinco que es la verificación de correcto Funcionamiento de la mesa CNC.

Recomendaciones.

La mesa de corte CNC debe ser utilizada por personal calificado y capacitado, conociendo las características técnicas y advertencias de la mesa CNC.

Durante todo el proceso mecanizado se debe cumplir con todas las normas de seguridad para evitar, lesiones o acciones por desprendimiento de elementos durante el corte

No manipular las conexiones eléctricas electrónicas de los dispositivos de control de la mesa CNC, para evitar el mal funcionamiento de la misma o accidentes.

Mantener la tarjeta de control en un lugar libre de humedad y de elementos que puedan dañarla.

Realizar los mantenimientos preventivos cada año.

La difusión del proyecto en la universidad, empresas pequeñas que se dediquen al tallado.

REFERENCIAS BIBLIOGRÁFICAS.

Cifuentes Molano, M. F., & Jaramillo Blandon, J. S. (2015). *Diseño de un Sistema de Manufactura Automatico para Circuitos Impresos.* Pereira: Universidad Tecnológica de Pereira.

Guanoluisa, H. y Sanchez, H. (2013). *Diseño y Construccion de una Maquina Fresadora CNC de 3 grados de libertad.* Quito: Escuela Politecnica Nacional

Jiménez, R. (s/f) *Control numérico por computadora*. Recuperado de: http://materias.fi.uba.ar/7565/U4-control-numerico-por-computadora.pdf

Maldonado, F. (2015). *Diseño de una Maquina Fresadora CNC para Mecanizado de Prototipos de Barcos en Madera*. Guayaquil: *ESPOL*

Miguel, R., & Zamora, R. (2014). *Diseño y fabricación de una fresadora CNC de 3 ejes para el mecanizado de PCB con plataformas de desarrollo abiertas.* Cartagena: s/e

Peters, J. (2013). *Desarrollo Electrónica DIYLILCNC*. Recuperado de: www.wiki.ead.pucv.cl

.

ANEXOS

En la figura 1 se muestra visualmente la aplicación. Se continúa con la parte de programación.

Figura 34. Visualización de la aplicación app inventor 2

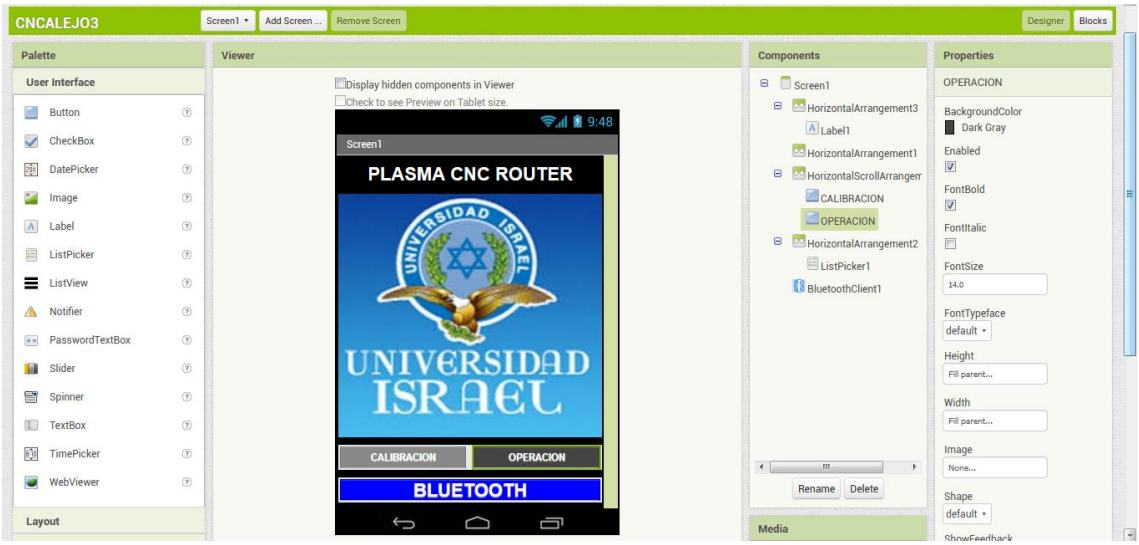


Figura 36. Visualización de la aplicación app inventor 2

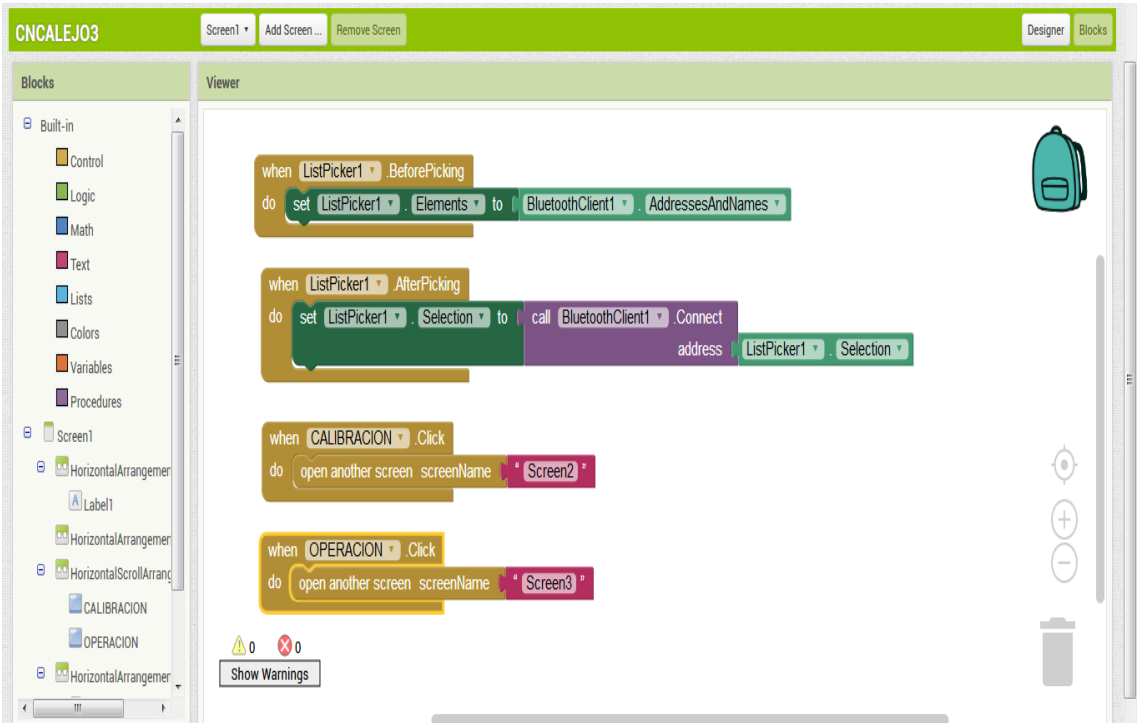


Figura 37. Programación en inventor 2

Figura 38. Visualización de cómo va a quedar la opción de calibración

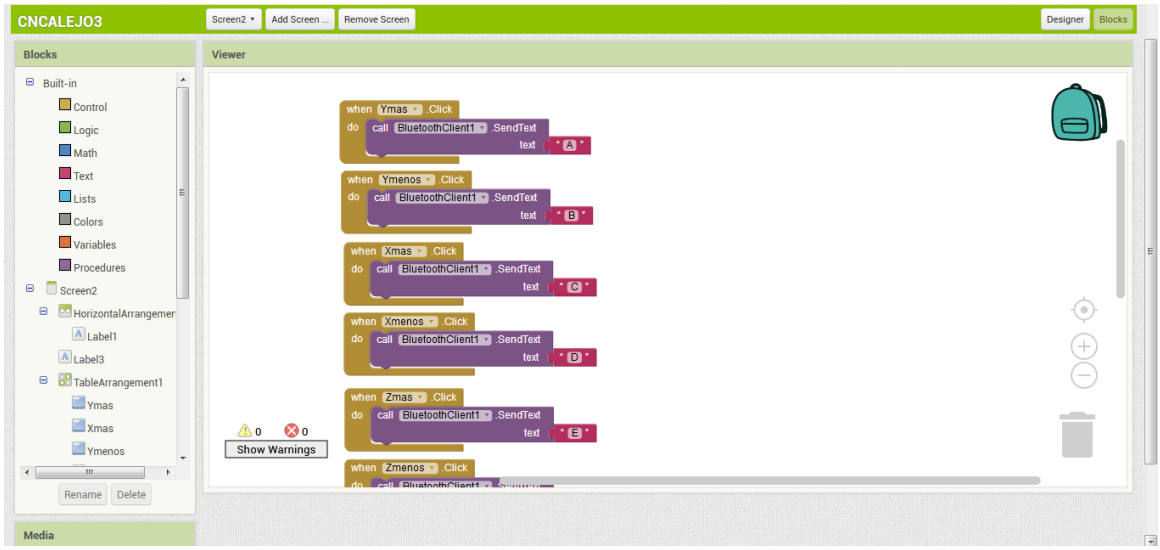
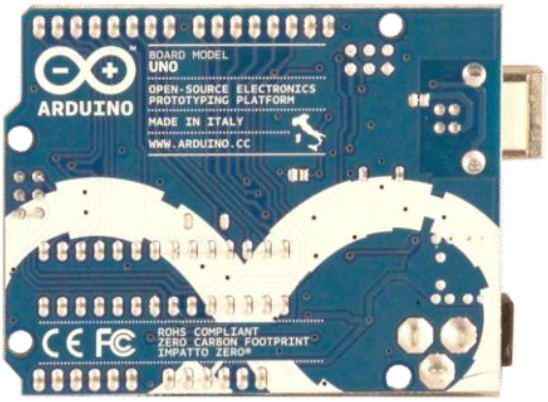


Figura 39. Programación del sistema de calibración

IMAGEN DE LA TARJETA ARDUINO UNO



RESUMEN DE CARACTERÍSTICAS

Microcontrolador	ATmega328
tensión de servicio	5V
Voltaje de entrada	7-12V (recomendado)
Voltaje de entrada	6-20V (límites)
E / S digital	14 (de los cuales 6 proporcionan una salida PWM)
entradas analógicas	6
corriente continua por I / O Pin	40 mA
Pin de la corriente de 3,3 V CC	50 mA
memoria flash	32 KB de los cuales 0,5 KB utilizado por cargador de arranque
SRAM	2 KB
EEPROM	2 KB
velocidad de reloj	16 MHz

DISTRIBUCION DE PINES

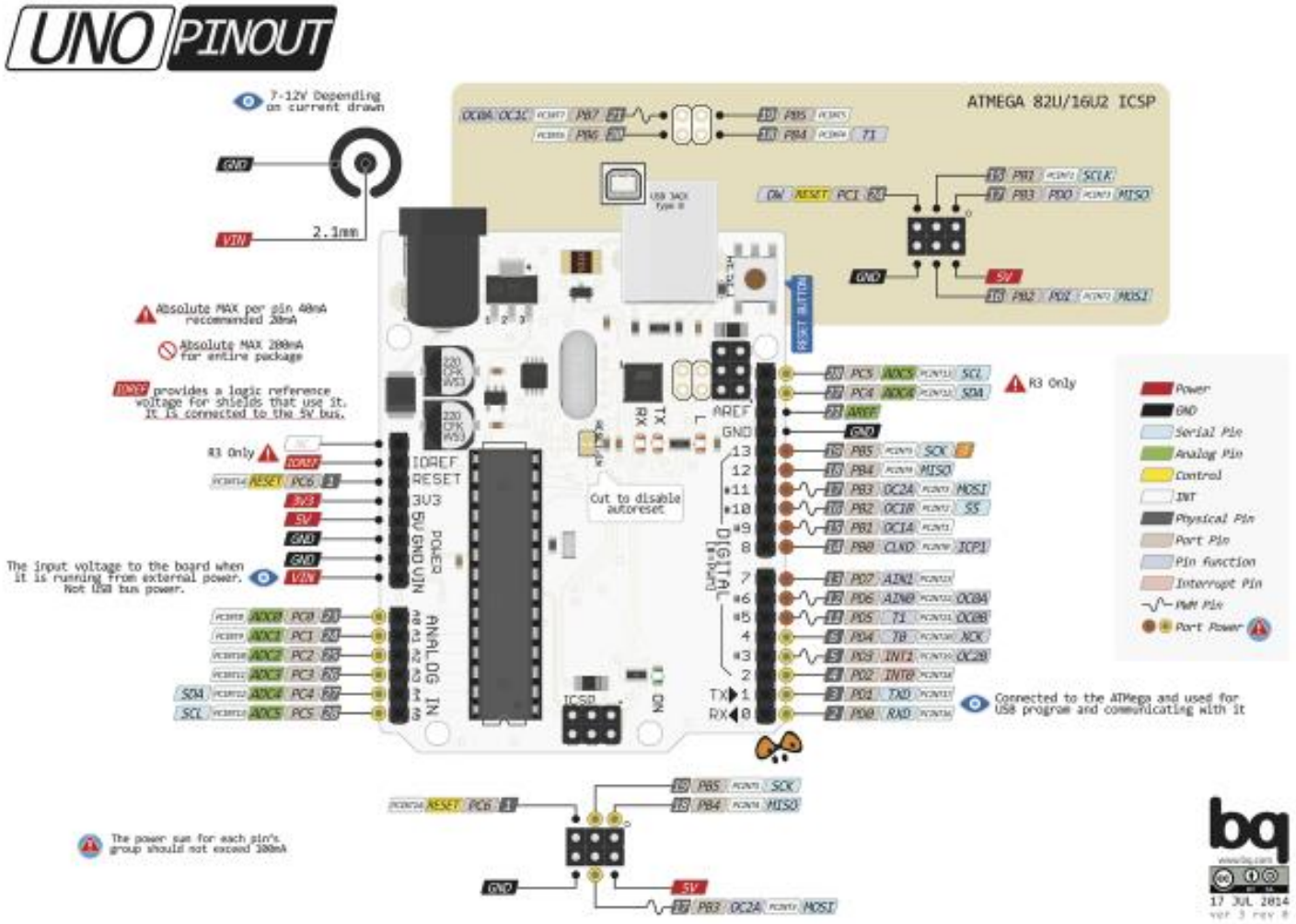
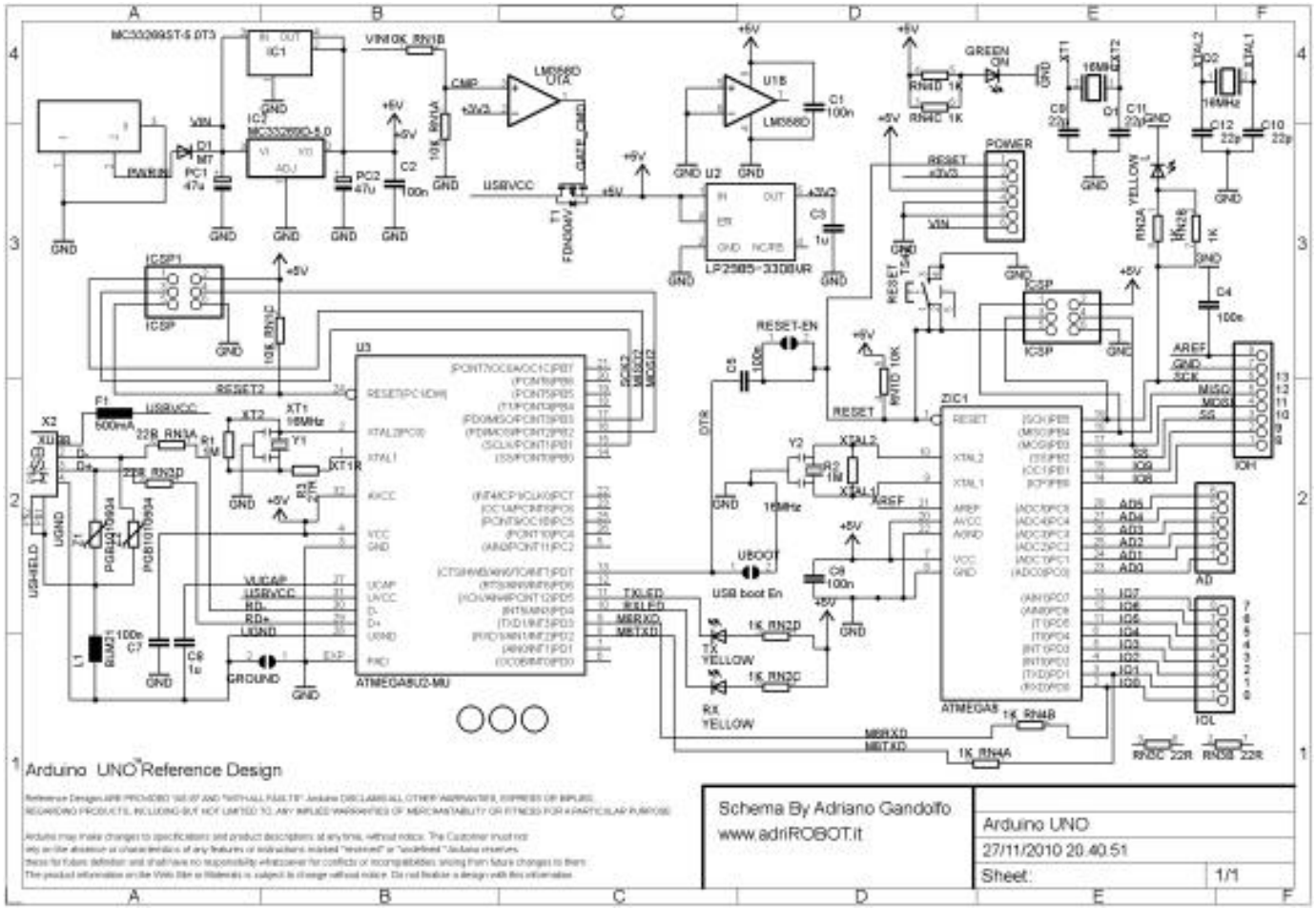
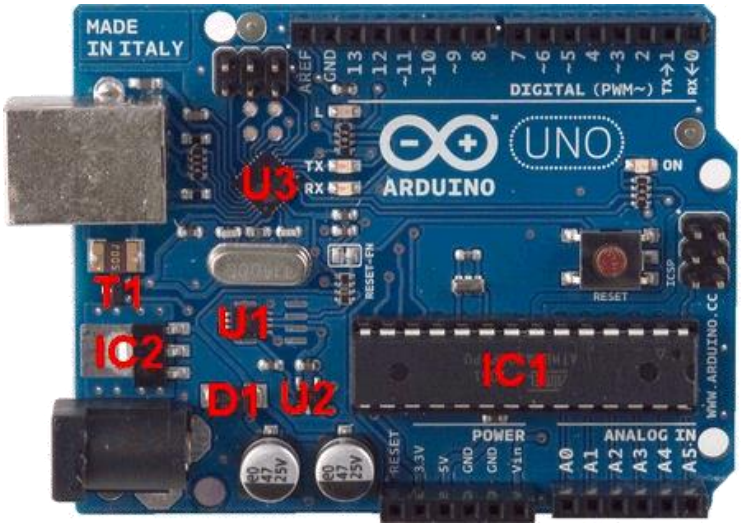


DIAGRAMA ELECTRICO DE LA TARJETA



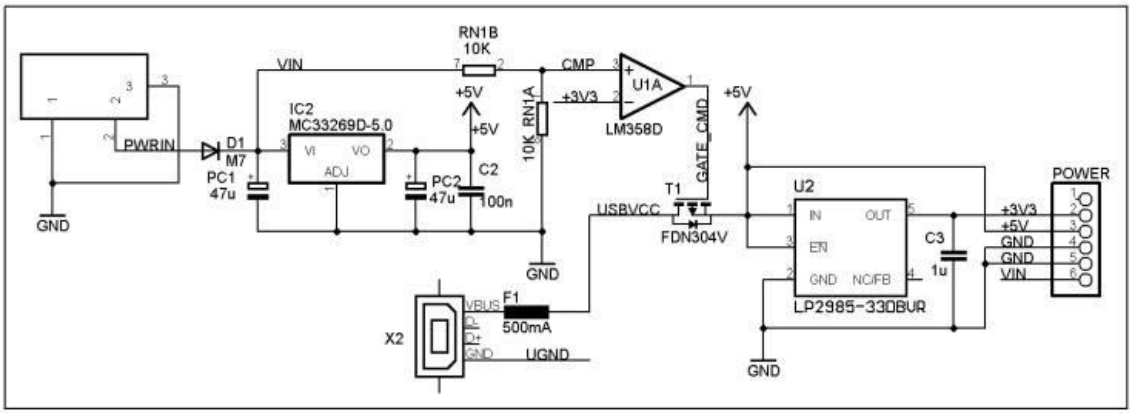
DESCRIPCIÓN Y UBICACIÓN DE LOS COMPONENTES

Sigla	valor
C1, C2, C4, C5, C6, C7	100nF
C3, C8	1uF
C9, C10, C11, C12	22pF
PC1, PC2	47µF
D1	diodo M7
F1	500mA Fusible
R1, R2	1MW
R3	27Ω
RN1	Red resistiva 10K
RN2, N4	Red resistiva 1k
RN3	Red resistiva 22Ω
REINICIAR	Botón de CS
RX, TX, L	LED amarillo
EN	LED verde
L1	WE-CBF ferrita BLM21
Z1-Z2	Varisitore PGB1010604
ZIC1	procesador **ATMEGA328**
U1	**LMV 358D**
U2	**LP2985-33DBVR**
U3	Procesador **ATmega8U2-MU**
T1	MOSFET **FDN304V**
IC1 / IC2	**MC33269ST-5.0T3**
UBoot	SJ jumper
X1	2.1mm DCJack
TIERRA	SJ
ICSP, ICSP1	PINHD-2X3
IOH, LIO	PINHD-1X8
POTENCIA	PINHD-1X6
AD	PINHD-1X6
Reset-ES	SJ jumper
X2	toma USB - PN61729
Y1, Y2	R 16MHz isuonatore
Q1, Q2	De cuarzo de 16 MHz
Z1, Z2	PGB1010604

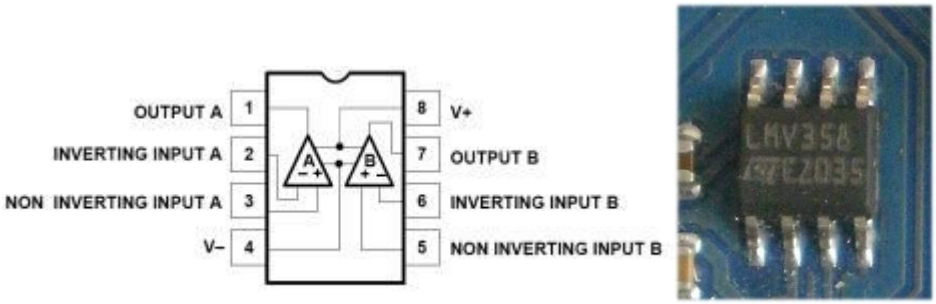


ALIMENTACION

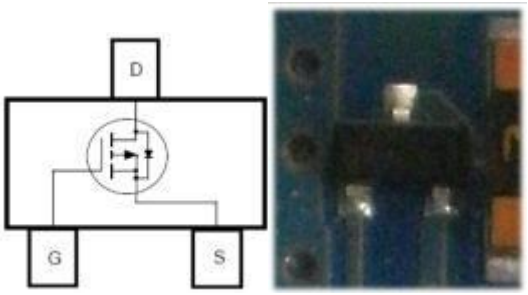
ESQUEMA DE LA ALIMENTACION



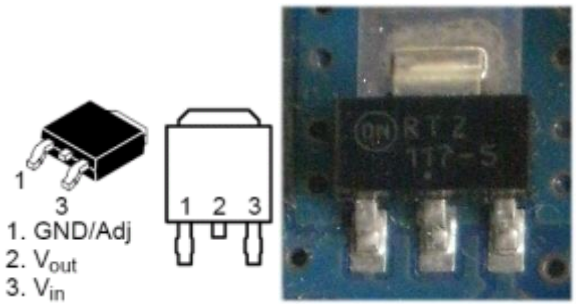
U1 Amplificadores Operacionales LM 358 de baja potencia Doble



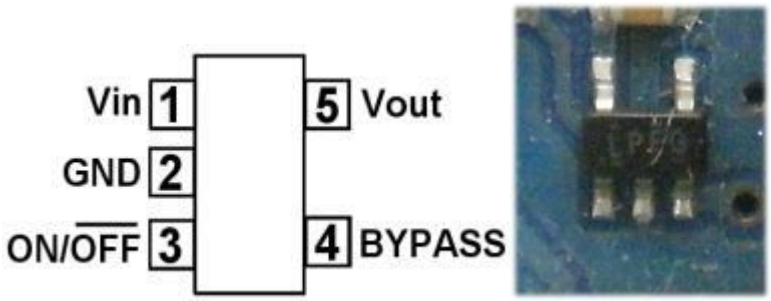
T1 - FDN304V P-Cha Transistor Modo nin Mejora de efecto de campo



MC33269D , MC33269ST-5.0 800 mA, regulador de baja caída de voltaje



U2 LP2985-33 - 150 mA de bajo ruido REGULADOR DE LA SALIDA DE BAJA CON PARADA

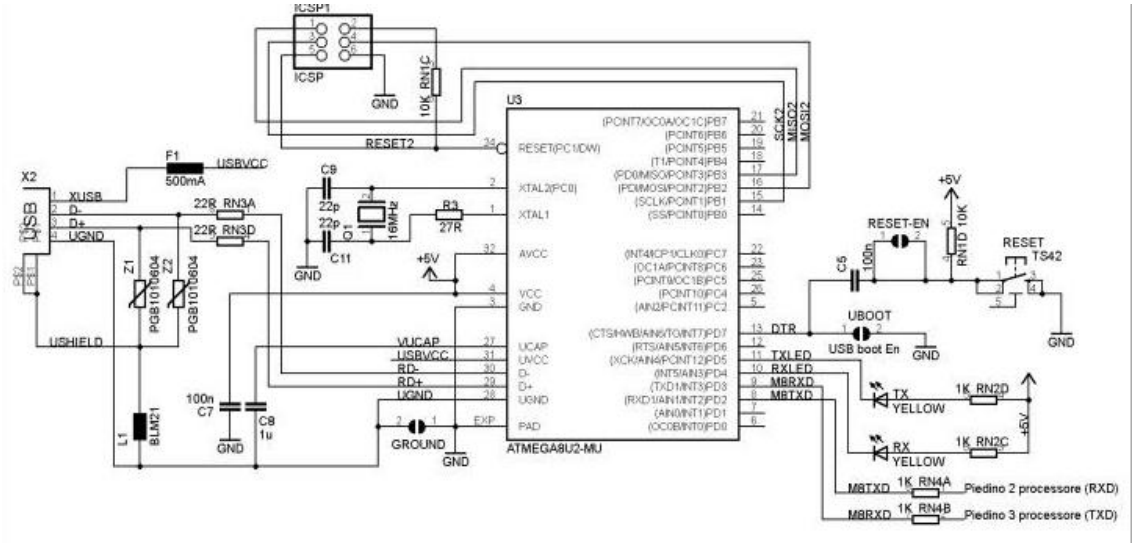


4007-SMD diodo rectificador 1ª

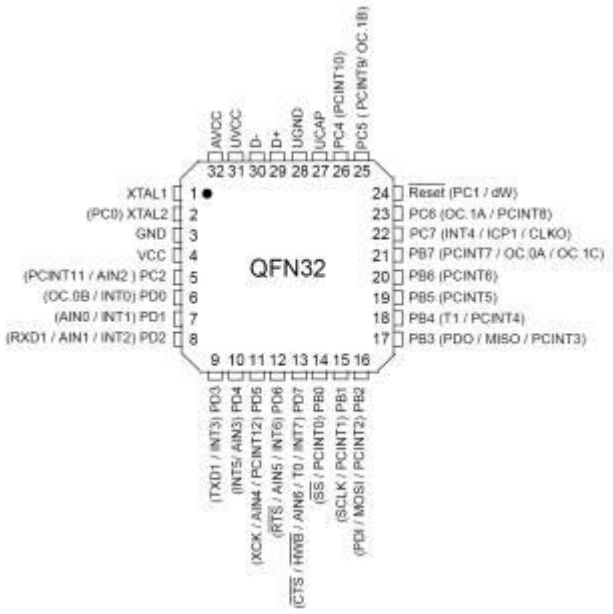
CONECTOR DE ALIMENTACION

INTERFAZ

ESQUEMA ELECTRICO DE LA SECCION DE INTERFAZ

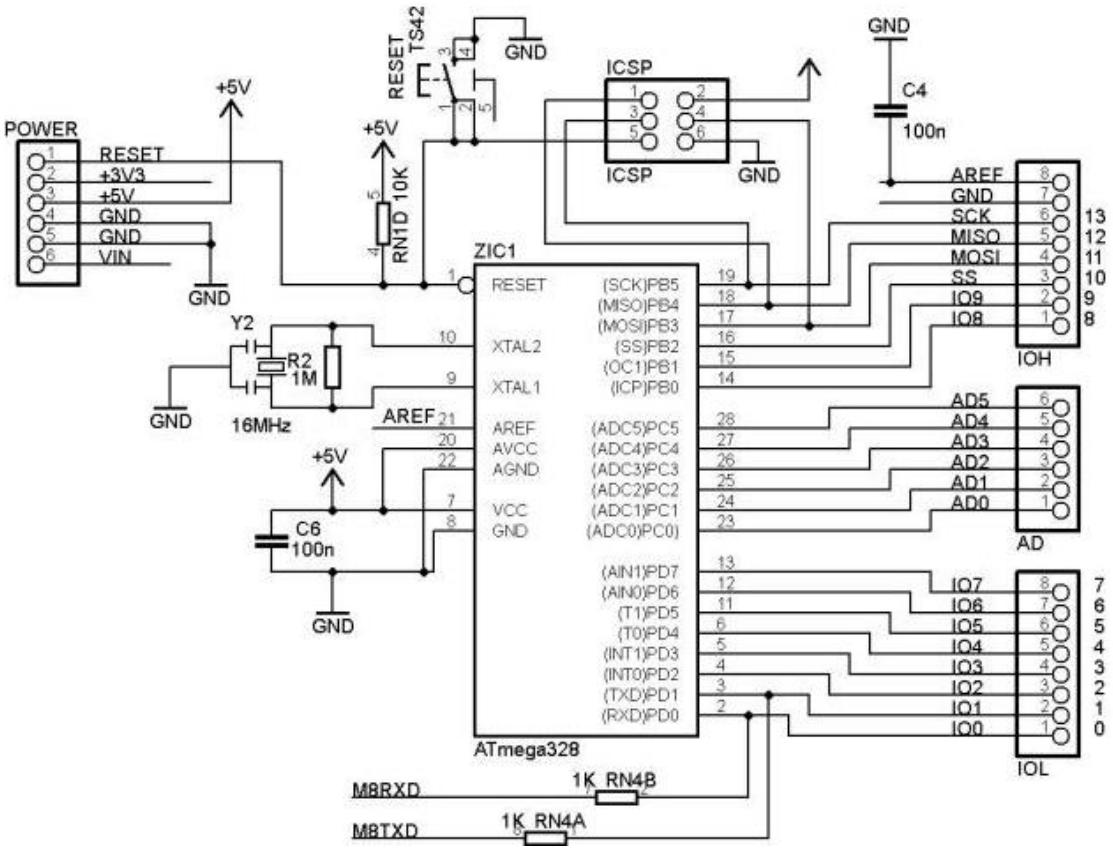


U3 - Procesador ATmega8U microcontrolador de 8 bits con el controlador USB



PROCESADOR

ESQUEMA ELÉCTRICO DEL PROCESADOR ATMEGA328



DISTRIBUCIÓN DE PINES DEL PROCESADOR ATMEGA328

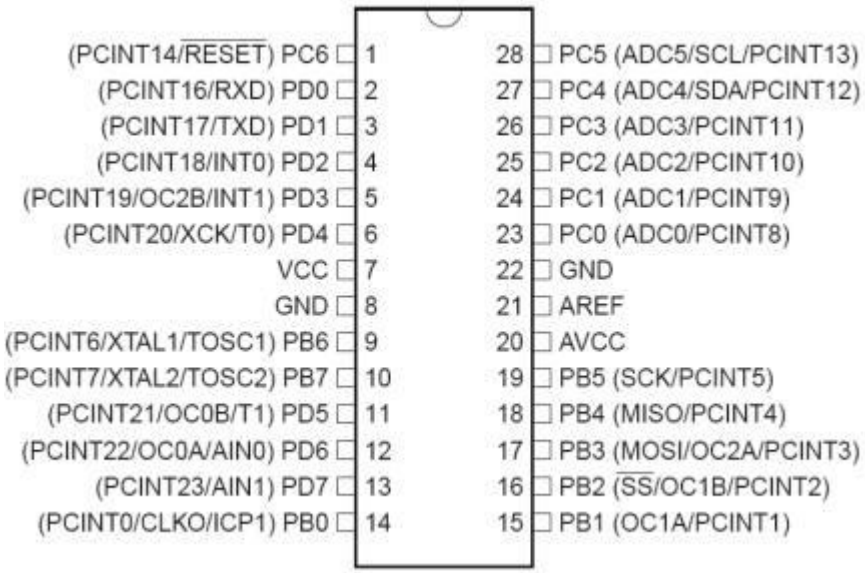
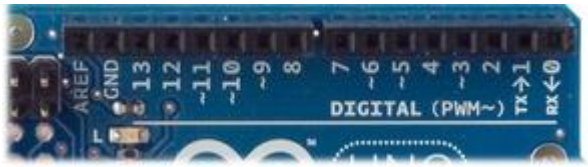


IMAGEN DEL PROCESADOR EN LA PLACA ARDUINO UNO

PIN CONECTOR DIGITAL



PIN CONECTOR ANALOGICO

Programa Arduino para app

```
int estado=0;

void setup(){
  Serial.begin(9600);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);

}

void loop(){
 if(Serial.available()>0){
 estado = Serial.read();
 }
if (estado =='A')
  {
   digitalWrite(3,HIGH);
```

```
   digitalWrite(4,LOW);
   digitalWrite(5,LOW);
    }
  if (estado =='B')
  {
   digitalWrite(3,HIGH);
   digitalWrite(4,LOW);
   digitalWrite(5,LOW);

  }
if(estado=='C')
  {
   digitalWrite(3,LOW);
   digitalWrite(4,HIGH);
   digitalWrite(5,LOW);
    }
if(estado=='D')
  {
   digitalWrite(3,LOW);
   digitalWrite(4,HIGH);
   digitalWrite(5,LOW);
    }

if (estado =='E')
  {
   digitalWrite(3,LOW);
   digitalWrite(4,LOW);
   digitalWrite(5,HIGH);
    }
  if (estado =='F')
  {
   digitalWrite(3,LOW);
   digitalWrite(4,LOW);
   digitalWrite(5,HIGH);
    }

if(estado=='G')
  {
   digitalWrite(3,LOW);
   digitalWrite(4,LOW);
   digitalWrite(5,LOW);
   digitalWrite(6,LOW);
  }
  if(estado=='H')
  {
   digitalWrite(6,HIGH);
  }

if(estado=='P')
  {
   digitalWrite(7,HIGH);
   digitalWrite(8,LOW);
   digitalWrite(9,LOW);
```

```
  digitalWrite(10,LOW);
 }
if(estado=='R')
 {
  digitalWrite(7,LOW);
  digitalWrite(8,HIGH);
  digitalWrite(9,LOW);
  digitalWrite(10,LOW);
 }
if(estado=='J')
 {
  digitalWrite(7,LOW);
  digitalWrite(8,LOW);
  digitalWrite(9,HIGH);
  digitalWrite(10,LOW);
 }

if(estado=='S')
 {
  digitalWrite(7,LOW);
  digitalWrite(8,LOW);
  digitalWrite(9,LOW);
  digitalWrite(10,HIGH);
 }

}
```

Líneas de código de las principales librerías para la ejecución del GRBL

```
/*
   config.h configuraciones del sistema


#ifndef config_h
#define config_h
#define DEFAULTS_GENERIC

#define BAUD_RATE 9600

#define STEPPING_DDR       DDRD
#define STEPPING_PORT      PORTD
#define X_STEP_BIT         2  // Uno Digital Pin 2
#define Y_STEP_BIT         3  // Uno Digital Pin 3
#define Z_STEP_BIT         4  // Uno Digital Pin 4
#define X_DIRECTION_BIT    5  // Uno Digital Pin 5
#define Y_DIRECTION_BIT    6  // Uno Digital Pin 6
#define Z_DIRECTION_BIT    7  // Uno Digital Pin 7
#define STEP_MASK ((1<<X_STEP_BIT)|(1<<Y_STEP_BIT)|(1<<Z_STEP_BIT)) //
All step bits
#define DIRECTION_MASK
((1<<X_DIRECTION_BIT)|(1<<Y_DIRECTION_BIT)|(1<<Z_DIRECTION_BIT))
#define STEPPING_MASK (STEP_MASK | DIRECTION_MASK) // All stepping-
related bits (step/direction)
```

```
#define STEPPERS_DISABLE_DDR    DDRB
#define STEPPERS_DISABLE_PORT   PORTB
#define STEPPERS_DISABLE_BIT    0  // Uno Digital Pin 8
#define STEPPERS_DISABLE_MASK (1<<STEPPERS_DISABLE_BIT)

#define LIMIT_DDR       DDRB
#define LIMIT_PIN       PINB
#define LIMIT_PORT      PORTB
#define X_LIMIT_BIT     1
#define Y_LIMIT_BIT     2
#define Z_LIMIT_BIT     3
#define LIMIT_INT       PCIE0
#define LIMIT_INT_vect  PCINT0_vect
#define LIMIT_PCMSK     PCMSK0
#define LIMIT_MASK ((1<<X_LIMIT_BIT)|(1<<Y_LIMIT_BIT)|(1<<Z_LIMIT_BIT))
#define SPINDLE_ENABLE_DDR   DDRB
#define SPINDLE_ENABLE_PORT  PORTB
#define SPINDLE_ENABLE_BIT   4
#define SPINDLE_DIRECTION_DDR   DDRB
#define SPINDLE_DIRECTION_PORT  PORTB
#define SPINDLE_DIRECTION_BIT   5
#define COOLANT_FLOOD_DDR   DDRC
#define COOLANT_FLOOD_PORT  PORTC
#define COOLANT_FLOOD_BIT   3  // Uno Analog Pin 3

#ifdef ENABLE_M7
  #define COOLANT_MIST_DDR   DDRC
  #define COOLANT_MIST_PORT  PORTC
  #define COOLANT_MIST_BIT   4 // Uno Analog Pin 4
#endif

#define PINOUT_DDR       DDRC
#define PINOUT_PIN       PINC
#define PINOUT_PORT      PORTC
#define PIN_RESET        0  // Uno Analog Pin 0
#define PIN_FEED_HOLD    1  // Uno Analog Pin 1
#define PIN_CYCLE_START  2  // Uno Analog Pin 2
#define PINOUT_INT       PCIE1  // Pin change interrupt enable pin
#define PINOUT_INT_vect  PCINT1_vect
#define PINOUT_PCMSK     PCMSK1 // Pin change interrupt register
#define PINOUT_MASK
((1<<PIN_RESET)|(1<<PIN_FEED_HOLD)|(1<<PIN_CYCLE_START))

#define CMD_STATUS_REPORT '?'
#define CMD_FEED_HOLD '!'
#define CMD_CYCLE_START '~'
#define CMD_RESET 0x18 // ctrl-x

#define ACCELERATION_TICKS_PER_SECOND 50L

#define MINIMUM_STEPS_PER_MINUTE 800 // (steps/min) - Integer value
only

#define DWELL_TIME_STEP 50 // Integer (1-255) (milliseconds)

#define HOMING_INIT_LOCK // Comment to disable
```

```
#define HOMING_RATE_ADJUST // Comment to disable

#define HOMING_SEARCH_CYCLE_0 (1<<Z_AXIS)
#define HOMING_SEARCH_CYCLE_1 ((1<<X_AXIS)|(1<<Y_AXIS))
#define HOMING_SEARCH_CYCLE_2                          // Uncomment and
add axes mask to enable
#define HOMING_LOCATE_CYCLE    ((1<<X_AXIS)|(1<<Y_AXIS)|(1<<Z_AXIS))
#define N_HOMING_LOCATE_CYCLE 2
#define N_STARTUP_LINE 2 // Integer (1-5)


#endif


/*
```

defaults.h – configuraciones por defecto

```
#ifndef defaults_h
#define defaults_h

#ifdef DEFAULTS_GENERIC
  #define DEFAULT_X_STEPS_PER_MM 250.0
  #define DEFAULT_Y_STEPS_PER_MM 250.0
  #define DEFAULT_Z_STEPS_PER_MM 250.0
  #define DEFAULT_STEP_PULSE_MICROSECONDS 10
  #define DEFAULT_MM_PER_ARC_SEGMENT 0.1
  #define DEFAULT_RAPID_FEEDRATE 500.0 // mm/min
  #define DEFAULT_FEEDRATE 250.0
  #define DEFAULT_ACCELERATION (10.0*60*60) // 10 mm/min^2
  #define DEFAULT_JUNCTION_DEVIATION 0.05 // mm
  #define DEFAULT_STEPPING_INVERT_MASK
((1<<Y_DIRECTION_BIT)|(1<<Z_DIRECTION_BIT))
  #define DEFAULT_REPORT_INCHES 0 // false
  #define DEFAULT_AUTO_START 1 // true
  #define DEFAULT_INVERT_ST_ENABLE 0 // false
  #define DEFAULT_HARD_LIMIT_ENABLE 0  // false
  #define DEFAULT_HOMING_ENABLE 0  // false
  #define DEFAULT_HOMING_DIR_MASK 0 // move positive dir
  #define DEFAULT_HOMING_RAPID_FEEDRATE 250.0 // mm/min
  #define DEFAULT_HOMING_FEEDRATE 25.0 // mm/min
  #define DEFAULT_HOMING_DEBOUNCE_DELAY 100 // msec (0-65k)
  #define DEFAULT_HOMING_PULLOFF 1.0 // mm
  #define DEFAULT_STEPPER_IDLE_LOCK_TIME 25 // msec (0-255)
  #define DEFAULT_DECIMAL_PLACES 3
  #define DEFAULT_N_ARC_CORRECTION 25
#endif

#ifdef DEFAULTS_SHERLINE_5400
KL23H256-21-8B 185 oz-in stepper motors,
power supply at 1.5A per winding.
  #define MICROSTEPS 2
  #define STEPS_PER_REV 200.0
  #define MM_PER_REV (0.050*MM_PER_INCH) // 0.050 inch/rev leadscrew
  #define DEFAULT_X_STEPS_PER_MM (STEPS_PER_REV*MICROSTEPS/MM_PER_REV)
  #define DEFAULT_Y_STEPS_PER_MM (STEPS_PER_REV*MICROSTEPS/MM_PER_REV)
  #define DEFAULT_Z_STEPS_PER_MM (STEPS_PER_REV*MICROSTEPS/MM_PER_REV)
```

```
  #define DEFAULT_STEP_PULSE_MICROSECONDS 10
  #define DEFAULT_MM_PER_ARC_SEGMENT 0.1
  #define DEFAULT_RAPID_FEEDRATE 635.0 // mm/min (25ipm)
  #define DEFAULT_FEEDRATE 254.0 // mm/min (10ipm)
  #define DEFAULT_ACCELERATION 50.0*60*60 // 50 mm/min^2
  #define DEFAULT_JUNCTION_DEVIATION 0.05 // mm
  #define DEFAULT_STEPPING_INVERT_MASK
((1<<Y_DIRECTION_BIT)|(1<<Z_DIRECTION_BIT))
  #define DEFAULT_REPORT_INCHES 1 // false
  #define DEFAULT_AUTO_START 1 // true
  #define DEFAULT_INVERT_ST_ENABLE 0 // false
  #define DEFAULT_HARD_LIMIT_ENABLE 0  // false
  #define DEFAULT_HOMING_ENABLE 0  // false
  #define DEFAULT_HOMING_DIR_MASK 0 // move positive dir
  #define DEFAULT_HOMING_RAPID_FEEDRATE 250.0 // mm/min
  #define DEFAULT_HOMING_FEEDRATE 25.0 // mm/min
  #define DEFAULT_HOMING_DEBOUNCE_DELAY 100 // msec (0-65k)
  #define DEFAULT_HOMING_PULLOFF 1.0 // mm
  #define DEFAULT_STEPPER_IDLE_LOCK_TIME 25 // msec (0-255)
  #define DEFAULT_DECIMAL_PLACES 3
  #define DEFAULT_N_ARC_CORRECTION 25
#endif

#ifdef DEFAULTS_SHAPEOKO
  // Description: Shapeoko CNC mill with three NEMA 17 stepper motors,
driven by Synthetos
  // grblShield with a 24V, 4.2A power supply.
  #define MICROSTEPS_XY 8
  #define STEP_REVS_XY 400
  #define MM_PER_REV_XY (0.08*18*MM_PER_INCH) // 0.08 in belt pitch, 18
pulley teeth
  #define MICROSTEPS_Z 2
  #define STEP_REVS_Z 400
  #define MM_PER_REV_Z 1.250 // 1.25 mm/rev leadscrew
  #define DEFAULT_X_STEPS_PER_MM
(MICROSTEPS_XY*STEP_REVS_XY/MM_PER_REV_XY)
  #define DEFAULT_Y_STEPS_PER_MM
(MICROSTEPS_XY*STEP_REVS_XY/MM_PER_REV_XY)
  #define DEFAULT_Z_STEPS_PER_MM
(MICROSTEPS_Z*STEP_REVS_Z/MM_PER_REV_Z)
  #define DEFAULT_STEP_PULSE_MICROSECONDS 10
  #define DEFAULT_MM_PER_ARC_SEGMENT 0.1
  #define DEFAULT_RAPID_FEEDRATE 1000.0 // mm/min
  #define DEFAULT_FEEDRATE 250.0
  #define DEFAULT_ACCELERATION (15.0*60*60) // 15 mm/min^2
  #define DEFAULT_JUNCTION_DEVIATION 0.05 // mm
  #define DEFAULT_STEPPING_INVERT_MASK
((1<<Y_DIRECTION_BIT)|(1<<Z_DIRECTION_BIT))
  #define DEFAULT_REPORT_INCHES 0 // false
  #define DEFAULT_AUTO_START 1 // true
  #define DEFAULT_INVERT_ST_ENABLE 0 // false
  #define DEFAULT_HARD_LIMIT_ENABLE 0  // false
  #define DEFAULT_HOMING_ENABLE 0  // false
  #define DEFAULT_HOMING_DIR_MASK 0 // move positive dir
  #define DEFAULT_HOMING_RAPID_FEEDRATE 250.0 // mm/min
  #define DEFAULT_HOMING_FEEDRATE 25.0 // mm/min
  #define DEFAULT_HOMING_DEBOUNCE_DELAY 100 // msec (0-65k)
```

```
  #define DEFAULT_HOMING_PULLOFF 1.0 // mm
  #define DEFAULT_STEPPER_IDLE_LOCK_TIME 255 // msec (0-255)
  #define DEFAULT_DECIMAL_PLACES 3
  #define DEFAULT_N_ARC_CORRECTION 25
#endif

#ifdef DEFAULTS_ZEN_TOOLWORKS_7x7
  // Description: Zen Toolworks 7x7 mill with three Shinano SST43D2121
65oz-in NEMA 17 stepper motors.
  // Leadscrew is different from some ZTW kits, where most are
1.25mm/rev rather than 8.0mm/rev here.
  // Driven by 30V, 6A power supply and TI DRV8811 stepper motor
drivers.
  #define MICROSTEPS 8
  #define STEPS_PER_REV 200.0
  #define MM_PER_REV 8.0 // 8 mm/rev leadscrew
  #define DEFAULT_X_STEPS_PER_MM (STEPS_PER_REV*MICROSTEPS/MM_PER_REV)
  #define DEFAULT_Y_STEPS_PER_MM (STEPS_PER_REV*MICROSTEPS/MM_PER_REV)
  #define DEFAULT_Z_STEPS_PER_MM (STEPS_PER_REV*MICROSTEPS/MM_PER_REV)
  #define DEFAULT_STEP_PULSE_MICROSECONDS 10
  #define DEFAULT_MM_PER_ARC_SEGMENT 0.1
  #define DEFAULT_RAPID_FEEDRATE 2500.0 // mm/min
  #define DEFAULT_FEEDRATE 1000.0 // mm/min
  #define DEFAULT_ACCELERATION 150.0*60*60 // 150 mm/min^2
  #define DEFAULT_JUNCTION_DEVIATION 0.05 // mm
  #define DEFAULT_STEPPING_INVERT_MASK (1<<Y_DIRECTION_BIT)
  #define DEFAULT_REPORT_INCHES 0 // false
  #define DEFAULT_AUTO_START 1 // true
  #define DEFAULT_INVERT_ST_ENABLE 0 // false
  #define DEFAULT_HARD_LIMIT_ENABLE 0  // false
  #define DEFAULT_HOMING_ENABLE 0  // false
  #define DEFAULT_HOMING_DIR_MASK 0 // move positive dir
  #define DEFAULT_HOMING_RAPID_FEEDRATE 500.0 // mm/min
  #define DEFAULT_HOMING_FEEDRATE 50.0 // mm/min
  #define DEFAULT_HOMING_DEBOUNCE_DELAY 100 // msec (0-65k)
  #define DEFAULT_HOMING_PULLOFF 1.0 // mm
  #define DEFAULT_STEPPER_IDLE_LOCK_TIME 25 // msec (0-255)
  #define DEFAULT_DECIMAL_PLACES 3
  #define DEFAULT_N_ARC_CORRECTION 25
#endif

#endif
```

eeprom configuraciones de la memoria

```
#include <avr/eeprom.h>

unsigned char eeprom_get_char( unsigned int addr )
{
      return eeprom_read_byte((unsigned char *) addr);
}

void eeprom_put_char( unsigned int addr, char new_value )
{
      eeprom_write_byte((unsigned char *) addr, new_value);
}
```

```
void memcpy_to_eeprom_with_checksum(unsigned int destination, char
*source, unsigned int size) {
  unsigned char checksum = 0;
  for(; size > 0; size--) {
    checksum = (checksum << 1) || (checksum >> 7);
    checksum += *source;
    eeprom_put_char(destination++, *(source++));
  }
  eeprom_put_char(destination, checksum);
}

int memcpy_from_eeprom_with_checksum(char *destination, unsigned int
source, unsigned int size) {
  unsigned char data, checksum = 0;
  for(; size > 0; size--) {
    data = eeprom_get_char(source++);
    checksum = (checksum << 1) || (checksum >> 7);
    checksum += data;
    *(destination++) = data;
  }
  return(checksum == eeprom_get_char(source));
}
```

Gcode //configuración del gcode

```
/*
  gcode.c - rs274/ngc parser.

#include "gcode.h"
#include <string.h>
#include "nuts_bolts.h"
#include <math.h>
#include "settings.h"
#include "motion_control.h"
#include "spindle_control.h"
#include "coolant_control.h"
#include "errno.h"
#include "protocol.h"
#include "report.h"

#define FAIL(status) gc.status_code = status;

static int next_statement(char *letter, float *float_ptr, char *line,
uint8_t *char_counter);

static void select_plane(uint8_t axis_0, uint8_t axis_1, uint8_t
axis_2)
{
  gc.plane_axis_0 = axis_0;
  gc.plane_axis_1 = axis_1;
  gc.plane_axis_2 = axis_2;
}

void gc_init()
```

```
{
  memset(&gc, 0, sizeof(gc));
  gc.feed_rate = settings.default_feed_rate; // Should be zero at
initialization.
  select_plane(X_AXIS, Y_AXIS, Z_AXIS);
  gc.absolute_mode = true;

  if (!(settings_read_coord_data(gc.coord_select,gc.coord_system))) {
    report_status_message(STATUS_SETTING_READ_FAIL);
  }
}

void gc_set_current_position(int32_t x, int32_t y, int32_t z)
{
  gc.position[X_AXIS] = x/settings.steps_per_mm[X_AXIS];
  gc.position[Y_AXIS] = y/settings.steps_per_mm[Y_AXIS];
  gc.position[Z_AXIS] = z/settings.steps_per_mm[Z_AXIS];
}

static float to_millimeters(float value)
{
  return(gc.inches_mode ? (value * MM_PER_INCH) : value);
}

uint8_t gc_execute_line(char *line)
{

  if (sys.state == STATE_ALARM) { return(STATUS_ALARM_LOCK); }

  uint8_t char_counter = 0;
  char letter;
  float value;
  int int_value;

  uint16_t modal_group_words = 0;  // Bitflag variable to track and
check modal group words in block
  uint8_t axis_words = 0;          // Bitflag to track which XYZ(ABC)
parameters exist in block

  float inverse_feed_rate = -1; // negative inverse_feed_rate means no
inverse_feed_rate specified
  uint8_t absolute_override = false; // true(1) = absolute motion for
this block only {G53}
  uint8_t non_modal_action = NON_MODAL_NONE; // Tracks the actions of
modal group 0 (non-modal)

  float target[3], offset[3];
  clear_vector(target); // XYZ(ABC) axes parameters.
  clear_vector(offset); // IJK Arc offsets are incremental. Value of
zero indicates no change.

  gc.status_code = STATUS_OK;

  uint8_t group_number = MODAL_GROUP_NONE;
  while(next_statement(&letter, &value, line, &char_counter)) {
    int_value = trunc(value);
    switch(letter) {
```

```
      case 'G':
        // Set modal group values
        switch(int_value) {
          case 4: case 10: case 28: case 30: case 53: case 92:
group_number = MODAL_GROUP_0; break;
          case 0: case 1: case 2: case 3: case 80: group_number =
MODAL_GROUP_1; break;
          case 17: case 18: case 19: group_number = MODAL_GROUP_2;
break;
          case 90: case 91: group_number = MODAL_GROUP_3; break;
          case 93: case 94: group_number = MODAL_GROUP_5; break;
          case 20: case 21: group_number = MODAL_GROUP_6; break;
          case 54: case 55: case 56: case 57: case 58: case 59:
group_number = MODAL_GROUP_12; break;
        }
        // Set 'G' commands
        switch(int_value) {
          case 0: gc.motion_mode = MOTION_MODE_SEEK; break;
          case 1: gc.motion_mode = MOTION_MODE_LINEAR; break;
          case 2: gc.motion_mode = MOTION_MODE_CW_ARC; break;
          case 3: gc.motion_mode = MOTION_MODE_CCW_ARC; break;
          case 4: non_modal_action = NON_MODAL_DWELL; break;
          case 10: non_modal_action = NON_MODAL_SET_COORDINATE_DATA;
break;
          case 17: select_plane(X_AXIS, Y_AXIS, Z_AXIS); break;
          case 18: select_plane(X_AXIS, Z_AXIS, Y_AXIS); break;
          case 19: select_plane(Y_AXIS, Z_AXIS, X_AXIS); break;
          case 20: gc.inches_mode = true; break;
          case 21: gc.inches_mode = false; break;
          case 28: case 30:
            int_value = trunc(10*value); // Multiply by 10 to pick up
Gxx.1
            switch(int_value) {
              case 280: non_modal_action = NON_MODAL_GO_HOME_0; break;
              case 281: non_modal_action = NON_MODAL_SET_HOME_0; break;
              case 300: non_modal_action = NON_MODAL_GO_HOME_1; break;
              case 301: non_modal_action = NON_MODAL_SET_HOME_1; break;
              default: FAIL(STATUS_UNSUPPORTED_STATEMENT);
            }
            break;
          case 53: absolute_override = true; break;
          case 54: case 55: case 56: case 57: case 58: case 59:
            gc.coord_select = int_value-54;
            break;
          case 80: gc.motion_mode = MOTION_MODE_CANCEL; break;
          case 90: gc.absolute_mode = true; break;
          case 91: gc.absolute_mode = false; break;
          case 92:
            int_value = trunc(10*value); // Multiply by 10 to pick up
G92.1
            switch(int_value) {
              case 920: non_modal_action =
NON_MODAL_SET_COORDINATE_OFFSET; break;
              case 921: non_modal_action =
NON_MODAL_RESET_COORDINATE_OFFSET; break;
              default: FAIL(STATUS_UNSUPPORTED_STATEMENT);
            }
```

```
            break;
          case 93: gc.inverse_feed_rate_mode = true; break;
          case 94: gc.inverse_feed_rate_mode = false; break;
          default: FAIL(STATUS_UNSUPPORTED_STATEMENT);
        }
        break;
      case 'M':
        switch(int_value) {
          case 0: case 1: case 2: case 30: group_number =
MODAL_GROUP_4; break;
          case 3: case 4: case 5: group_number = MODAL_GROUP_7; break;
        }
        switch(int_value) {
          case 0: gc.program_flow = PROGRAM_FLOW_PAUSED; break; //
Program pause
          case 1: break; // Optional stop not supported. Ignore.
          case 2: case 30: gc.program_flow = PROGRAM_FLOW_COMPLETED;
break; // Program end and reset
          case 3: gc.spindle_direction = 1; break;
          case 4: gc.spindle_direction = -1; break;
          case 5: gc.spindle_direction = 0; break;
          #ifdef ENABLE_M7
            case 7: gc.coolant_mode = COOLANT_MIST_ENABLE; break;
          #endif
          case 8: gc.coolant_mode = COOLANT_FLOOD_ENABLE; break;
          case 9: gc.coolant_mode = COOLANT_DISABLE; break;
          default: FAIL(STATUS_UNSUPPORTED_STATEMENT);
        }
        break;
    }
    if (group_number) {
      if ( bit_istrue(modal_group_words,bit(group_number)) ) {
        FAIL(STATUS_MODAL_GROUP_VIOLATION);
      } else {
        bit_true(modal_group_words,bit(group_number));
      }
      group_number = MODAL_GROUP_NONE; // Reset for next command.
    }
  }

  if (gc.status_code) { return(gc.status_code); }

  float p = 0, r = 0;
  uint8_t l = 0;
  char_counter = 0;
  while(next_statement(&letter, &value, line, &char_counter)) {
    switch(letter) {
      case 'G': case 'M': case 'N': break; // Ignore command statements
and line numbers
      case 'F':
        if (value <= 0) { FAIL(STATUS_INVALID_STATEMENT); } // Must be
greater than zero
        if (gc.inverse_feed_rate_mode) {
          inverse_feed_rate = to_millimeters(value); // seconds per
motion for this motion only
        } else {
```

```
          gc.feed_rate = to_millimeters(value); // millimeters per
minute
        }
        break;
      case 'I': case 'J': case 'K': offset[letter-'I'] =
to_millimeters(value); break;
      case 'L': l = trunc(value); break;
      case 'P': p = value; break;
      case 'R': r = to_millimeters(value); break;
      case 'S':
        if (value < 0) { FAIL(STATUS_INVALID_STATEMENT); } // Cannot be
negative
        // gc.spindle_speed = value;
        break;
      case 'T':
        if (value < 0) { FAIL(STATUS_INVALID_STATEMENT); } // Cannot be
negative
        gc.tool = trunc(value);
        break;
      case 'X': target[X_AXIS] = to_millimeters(value);
bit_true(axis_words,bit(X_AXIS)); break;
      case 'Y': target[Y_AXIS] = to_millimeters(value);
bit_true(axis_words,bit(Y_AXIS)); break;
      case 'Z': target[Z_AXIS] = to_millimeters(value);
bit_true(axis_words,bit(Z_AXIS)); break;
      default: FAIL(STATUS_UNSUPPORTED_STATEMENT);
    }
  }

  if (gc.status_code) { return(gc.status_code); }



  if (sys.state != STATE_CHECK_MODE) {

    spindle_run(gc.spindle_direction);

    coolant_run(gc.coolant_mode);
  }

  if ( bit_istrue(modal_group_words,bit(MODAL_GROUP_12)) ) { // Check
if called in block
    float coord_data[N_AXIS];
    if (!(settings_read_coord_data(gc.coord_select,coord_data))) {
return(STATUS_SETTING_READ_FAIL); }
    memcpy(gc.coord_system,coord_data,sizeof(coord_data));
  }

  switch (non_modal_action) {
    case NON_MODAL_DWELL:
      if (p < 0) { // Time cannot be negative.
        FAIL(STATUS_INVALID_STATEMENT);
      } else {
        // Ignore dwell in check gcode modes
        if (sys.state != STATE_CHECK_MODE) { mc_dwell(p); }
      }
      break;
```

```
    case NON_MODAL_SET_COORDINATE_DATA:
      int_value = trunc(p); // Convert p value to int.
      if ((l != 2 && l != 20) || (int_value < 1 || int_value >
N_COORDINATE_SYSTEM)) { // L2 and L20. P1=G54, P2=G55, ...
        FAIL(STATUS_UNSUPPORTED_STATEMENT);
      } else if (!axis_words && l==2) { // No axis words.
        FAIL(STATUS_INVALID_STATEMENT);
      } else {
        int_value--; // Adjust P index to EEPROM coordinate data
indexing.
        if (l == 20) {
          settings_write_coord_data(int_value,gc.position);
          // Update system coordinate system if currently active.
          if (gc.coord_select == int_value) {
memcpy(gc.coord_system,gc.position,sizeof(gc.position)); }
        } else {
          float coord_data[N_AXIS];
          if (!settings_read_coord_data(int_value,coord_data)) {
return(STATUS_SETTING_READ_FAIL); }
          // Update axes defined only in block. Always in machine
coordinates. Can change non-active system.
          uint8_t i;
          for (i=0; i<N_AXIS; i++) { // Axes indices are consistent, so
loop may be used.
            if ( bit_istrue(axis_words,bit(i)) ) { coord_data[i] =
target[i]; }
          }
          settings_write_coord_data(int_value,coord_data);
          // Update system coordinate system if currently active.
          if (gc.coord_select == int_value) {
memcpy(gc.coord_system,coord_data,sizeof(coord_data)); }
        }
      }
      axis_words = 0; // Axis words used. Lock out from motion modes by
clearing flags.
      break;
    case NON_MODAL_GO_HOME_0: case NON_MODAL_GO_HOME_1:
      // Move to intermediate position before going home. Obeys current
coordinate system and offsets
      // and absolute and incremental modes.
      if (axis_words) {
        // Apply absolute mode coordinate offsets or incremental mode
offsets.
        uint8_t i;
        for (i=0; i<N_AXIS; i++) { // Axes indices are consistent, so
loop may be used.
          if ( bit_istrue(axis_words,bit(i)) ) {
            if (gc.absolute_mode) {
              target[i] += gc.coord_system[i] + gc.coord_offset[i];
            } else {
              target[i] += gc.position[i];
            }
          } else {
            target[i] = gc.position[i];
          }
        }
```

```
        mc_line(target[X_AXIS], target[Y_AXIS], target[Z_AXIS],
settings.default_seek_rate, false);
      }
      // Retreive G28/30 go-home position data (in machine coordinates)
from EEPROM
      float coord_data[N_AXIS];
      home_select = SETTING_INDEX_G28;
      if (non_modal_action == NON_MODAL_GO_HOME_1) { home_select =
SETTING_INDEX_G30; }
      if (!settings_read_coord_data(home_select,coord_data)) {
return(STATUS_SETTING_READ_FAIL); }
      mc_line(coord_data[X_AXIS], coord_data[Y_AXIS],
coord_data[Z_AXIS], settings.default_seek_rate, false);
      memcpy(gc.position, coord_data, sizeof(coord_data)); //
gc.position[] = coord_data[];
      axis_words = 0; // Axis words used. Lock out from motion modes by
clearing flags.
      break;
    case NON_MODAL_SET_HOME_0: case NON_MODAL_SET_HOME_1:
      home_select = SETTING_INDEX_G28;
      if (non_modal_action == NON_MODAL_SET_HOME_1) { home_select =
SETTING_INDEX_G30; }
      settings_write_coord_data(home_select,gc.position);
      break;
    case NON_MODAL_SET_COORDINATE_OFFSET:
      if (!axis_words) { // No axis words
        FAIL(STATUS_INVALID_STATEMENT);
      } else {
        // Update axes defined only in block. Offsets current system to
defined value. Does not update when
        // active coordinate system is selected, but is still active
unless G92.1 disables it.
        uint8_t i;
        for (i=0; i<=2; i++) { // Axes indices are consistent, so loop
may be used.
          if (bit_istrue(axis_words,bit(i)) ) {
            gc.coord_offset[i] = gc.position[i]-gc.coord_system[i]-
target[i];
          }
        }
      }
      axis_words = 0; // Axis words used. Lock out from motion modes by
clearing flags.
      break;
    case NON_MODAL_RESET_COORDINATE_OFFSET:
      clear_vector(gc.coord_offset); // Disable G92 offsets by zeroing
offset vector.
      break;
  }

  // [G0,G1,G2,G3,G80]: Perform motion modes.
  // NOTE: Commands G10,G28,G30,G92 lock out and prevent axis words
from use in motion modes.
  // Enter motion modes only if there are axis words or a motion mode
command word in the block.
  if ( bit_istrue(modal_group_words,bit(MODAL_GROUP_1)) || axis_words )
{
```

```
    // G1,G2,G3 require F word in inverse time mode.
    if ( gc.inverse_feed_rate_mode ) {
      if (inverse_feed_rate < 0 && gc.motion_mode !=
MOTION_MODE_CANCEL) {
        FAIL(STATUS_INVALID_STATEMENT);
      }
    }
    // Absolute override G53 only valid with G0 and G1 active.
    if ( absolute_override && !(gc.motion_mode == MOTION_MODE_SEEK ||
gc.motion_mode == MOTION_MODE_LINEAR)) {
      FAIL(STATUS_INVALID_STATEMENT);
    }
    // Report any errors.
    if (gc.status_code) { return(gc.status_code); }

    // Convert all target position data to machine coordinates for
executing motion. Apply
    // absolute mode coordinate offsets or incremental mode offsets.
    // NOTE: Tool offsets may be appended to these conversions when/if
this feature is added.
    uint8_t i;
    for (i=0; i<=2; i++) { // Axes indices are consistent, so loop may
be used to save flash space.
      if ( bit_istrue(axis_words,bit(i)) ) {
        if (!absolute_override) { // Do not update target in absolute
override mode
          if (gc.absolute_mode) {
            target[i] += gc.coord_system[i] + gc.coord_offset[i]; //
Absolute mode
          } else {
            target[i] += gc.position[i]; // Incremental mode
          }
        }
      } else {
        target[i] = gc.position[i]; // No axis word in block. Keep same
axis position.
      }
    }

    switch (gc.motion_mode) {
      case MOTION_MODE_CANCEL:
        if (axis_words) { FAIL(STATUS_INVALID_STATEMENT); } // No axis
words allowed while active.
        break;
      case MOTION_MODE_SEEK:
        if (!axis_words) { FAIL(STATUS_INVALID_STATEMENT);}
        else { mc_line(target[X_AXIS], target[Y_AXIS], target[Z_AXIS],
settings.default_seek_rate, false); }
        break;
      case MOTION_MODE_LINEAR:
        // TODO: Inverse time requires F-word with each statement. Need
to do a check. Also need
        // to check for initial F-word upon startup. Maybe just set to
zero upon initialization
        // and after an inverse time move and then check for non-zero
feed rate each time. This
```

```
        // should be efficient and effective.
        if (!axis_words) { FAIL(STATUS_INVALID_STATEMENT);}
        else { mc_line(target[X_AXIS], target[Y_AXIS], target[Z_AXIS],
          (gc.inverse_feed_rate_mode) ? inverse_feed_rate :
gc.feed_rate, gc.inverse_feed_rate_mode); }
        break;
      case MOTION_MODE_CW_ARC: case MOTION_MODE_CCW_ARC:
        // Check if at least one of the axes of the selected plane has
been specified. If in center
        // format arc mode, also check for at least one of the IJK axes
of the selected plane was sent.
        if ( !( bit_false(axis_words,bit(gc.plane_axis_2)) ) ||
             ( !r && !offset[gc.plane_axis_0] &&
!offset[gc.plane_axis_1] ) ) {
          FAIL(STATUS_INVALID_STATEMENT);
        } else {
          if (r != 0) { // Arc Radius Mode
            /*
            float x = target[gc.plane_axis_0]-
gc.position[gc.plane_axis_0];
            float y = target[gc.plane_axis_1]-
gc.position[gc.plane_axis_1];

            clear_vector(offset);
            // First, use h_x2_div_d to compute 4*h^2 to check if it is
negative or r is smaller
            float h_x2_div_d = 4 * r*r - x*x - y*y;
            if (h_x2_div_d < 0) { FAIL(STATUS_ARC_RADIUS_ERROR);
return(gc.status_code); }
            // Finish computing h_x2_div_d.
            h_x2_div_d = -sqrt(h_x2_div_d)/hypot(x,y); // == -(h * 2 /
d)
            // Invert the sign of h_x2_div_d if the circle is counter
clockwise (see sketch below)
            if (gc.motion_mode == MOTION_MODE_CCW_ARC) { h_x2_div_d = -
h_x2_div_d; }

            if (r < 0) {
                h_x2_div_d = -h_x2_div_d;
                r = -r; // Finished with r. Set to positive for mc_arc
            }
            // Complete the operation by calculating the actual center
of the arc
            offset[gc.plane_axis_0] = 0.5*(x-(y*h_x2_div_d));
            offset[gc.plane_axis_1] = 0.5*(y+(x*h_x2_div_d));

          } else { // Arc Center Format Offset Mode
            r = hypot(offset[gc.plane_axis_0],
offset[gc.plane_axis_1]); // Compute arc radius for mc_arc
          }

          // Set clockwise/counter-clockwise sign for mc_arc
computations
          uint8_t isclockwise = false;
          if (gc.motion_mode == MOTION_MODE_CW_ARC) { isclockwise =
true; }
```

```
        }
        break;
    }

    memcpy(gc.position, target, sizeof(target));
  }

  if (gc.program_flow) {
    plan_synchronize(); // Finish all remaining buffered motions.
Program paused when complete.
    sys.auto_start = false; // Disable auto cycle start. Forces pause
until cycle start issued.

    if (gc.program_flow == PROGRAM_FLOW_COMPLETED) { mc_reset(); }
    else { gc.program_flow = PROGRAM_FLOW_RUNNING; }
  }

  return(gc.status_code);
}

static int next_statement(char *letter, float *float_ptr, char *line,
uint8_t *char_counter)
{
  if (line[*char_counter] == 0) {
    return(0); // No more statements
  }

  *letter = line[*char_counter];
  if((*letter < 'A') || (*letter > 'Z')) {
    FAIL(STATUS_EXPECTED_COMMAND_LETTER);
    return(0);
  }
  (*char_counter)++;
  if (!read_float(line, char_counter, float_ptr)) {
    FAIL(STATUS_BAD_NUMBER_FORMAT);
    return(0);
  };
  return(1);
}
```

Gcode main //ejecución del código g matriz

```
/*
  main.c - An embedded CNC Controller with rs274/ngc (g-code) support

#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include "config.h"
#include "planner.h"
#include "nuts_bolts.h"
#include "stepper.h"
#include "spindle_control.h"
#include "coolant_control.h"
#include "motion_control.h"
#include "gcode.h"
```

```
#include "protocol.h"
#include "limits.h"
#include "report.h"
#include "settings.h"
#include "serial.h"

system_t sys;

int startGrbl(void)
{
  serial_init(); // Setup serial baud rate and interrupts
  settings_init(); // Load grbl settings from EEPROM
  st_init(); // Setup stepper pins and interrupt timers
  sei(); // Enable interrupts

  memset(&sys, 0, sizeof(sys));  // Clear all system variables
  sys.abort = true;   // Set abort to complete initialization
  sys.state = STATE_INIT;  // Set alarm state to indicate unknown
initial position

  for(;;) {

    if (sys.abort) {
      serial_reset_read_buffer(); // Clear serial read buffer
      plan_init(); // Clear block buffer and planner variables
      gc_init(); // Set g-code parser to default state
      protocol_init(); // Clear incoming line data and execute startup
lines
      spindle_init();
      coolant_init();
      limits_init();
      st_reset(); // Clear stepper subsystem variables.

      // Sync cleared gcode and planner positions to current system
position, which is only
      sys.abort = false;
      sys.execute = 0;
      if (bit_istrue(settings.flags,BITFLAG_AUTO_START)) {
sys.auto_start = true; }

enabled to force homing cycle
commands, including the
internal commands. Only a homing alarm.
      #ifdef HOMING_INIT_LOCK
        if (sys.state == STATE_INIT &&
bit_istrue(settings.flags,BITFLAG_HOMING_ENABLE)) { sys.state =
STATE_ALARM; }
      #endif

an initial power up.
      if (sys.state == STATE_ALARM) {
        report_feedback_message(MESSAGE_ALARM_LOCK);
      } else {
        sys.state = STATE_IDLE;
        protocol_execute_startup();
      }
    }
```

```
    protocol_execute_runtime();
    protocol_process(); // ... process the serial protocol

  }
  return 0;   /* never reached */
}
```

Serial // configuración y programacion de la comunuicacion serial

```
/*
  serial.c -
#include <avr/interrupt.h>
#include "serial.h"
#include "config.h"
#include "motion_control.h"
#include "protocol.h"


uint8_t rx_buffer[RX_BUFFER_SIZE];
uint8_t rx_buffer_head = 0;
uint8_t rx_buffer_tail = 0;

uint8_t tx_buffer[TX_BUFFER_SIZE];
uint8_t tx_buffer_head = 0;
volatile uint8_t tx_buffer_tail = 0;

#ifdef ENABLE_XONXOFF
  volatile uint8_t flow_ctrl = XON_SENT; // Flow control state variable
  static uint8_t get_rx_buffer_count()
  {
    if (rx_buffer_head == rx_buffer_tail) { return(0); }
    if (rx_buffer_head < rx_buffer_tail) { return(rx_buffer_tail-
rx_buffer_head); }
    return (RX_BUFFER_SIZE - (rx_buffer_head-rx_buffer_tail));
  }
#endif


void serial_init()
{
  #if BAUD_RATE < 57600
    uint16_t UBRR0_value = ((F_CPU / (8L * BAUD_RATE)) - 1)/2 ;
    UCSR0A &= ~(1 << U2X0); // baud doubler off  - Only needed on Uno
XXX
  #else
    uint16_t UBRR0_value = ((F_CPU / (4L * BAUD_RATE)) - 1)/2;
    UCSR0A |= (1 << U2X0);  // baud doubler on for high baud rates,
i.e. 115200
  #endif
  UBRR0H = UBRR0_value >> 8;
  UBRR0L = UBRR0_value;

  UCSR0B |= 1<<RXEN0;
  UCSR0B |= 1<<TXEN0;
```

```
  UCSR0B |= 1<<RXCIE0;

}


void serial_write(uint8_t data) {
  uint8_t next_head = tx_buffer_head + 1;
  if (next_head == TX_BUFFER_SIZE) { next_head = 0; }

  while (next_head == tx_buffer_tail) {
    if (sys.execute & EXEC_RESET) { return; } // Only check for abort
to avoid an endless loop.
  }

  tx_buffer[tx_buffer_head] = data;
  tx_buffer_head = next_head;

  UCSR0B |=  (1 << UDRIE0);
}

#if defined(__AVR_ATmega644P__) || defined(__AVR_ATmega2560__)
ISR(USART0_UDRE_vect)
#else
ISR(USART_UDRE_vect)
#endif
{
  uint8_t tail = tx_buffer_tail;

  #ifdef ENABLE_XONXOFF
    if (flow_ctrl == SEND_XOFF) {
      UDR0 = XOFF_CHAR;
      flow_ctrl = XOFF_SENT;
    } else if (flow_ctrl == SEND_XON) {
      UDR0 = XON_CHAR;
      flow_ctrl = XON_SENT;
    } else
  #endif
  {
    UDR0 = tx_buffer[tail];

    tail++;
    if (tail == TX_BUFFER_SIZE) { tail = 0; }

    tx_buffer_tail = tail;
  }

  if (tail == tx_buffer_head) { UCSR0B &= ~(1 << UDRIE0); }
}



uint8_t serial_read()
{
  if (rx_buffer_head == rx_buffer_tail) {
    return SERIAL_NO_DATA;
  } else {
```

```
    uint8_t data = rx_buffer[rx_buffer_tail];
    rx_buffer_tail++;
    if (rx_buffer_tail == RX_BUFFER_SIZE) { rx_buffer_tail = 0; }

    #ifdef ENABLE_XONXOFF
      if ((get_rx_buffer_count() < RX_BUFFER_LOW) && flow_ctrl ==
XOFF_SENT) {
        flow_ctrl = SEND_XON;
        UCSR0B |=  (1 << UDRIE0); // Force TX
      }
    #endif

    return data;
  }
}


#if defined(__AVR_ATmega644P__) || defined(__AVR_ATmega2560__)
ISR(USART0_RX_vect)
#else
ISR(USART_RX_vect)
#endif
{
  uint8_t data = UDR0;
  uint8_t next_head;

  // Pick off runtime command characters directly from the serial
stream. These characters are
  // not passed into the buffer, but these set system state flag bits
for runtime execution.
  switch (data) {
    case CMD_STATUS_REPORT: sys.execute |= EXEC_STATUS_REPORT; break;
// Set as true
    case CMD_CYCLE_START:   sys.execute |= EXEC_CYCLE_START; break; //
Set as true
    case CMD_FEED_HOLD:     sys.execute |= EXEC_FEED_HOLD; break; //
Set as true
    case CMD_RESET:         mc_reset(); break; // Call motion control
reset routine.
    default: // Write character to buffer
      next_head = rx_buffer_head + 1;
      if (next_head == RX_BUFFER_SIZE) { next_head = 0; }

      // Write data to buffer unless it is full.
      if (next_head != rx_buffer_tail) {
        rx_buffer[rx_buffer_head] = data;
        rx_buffer_head = next_head;

        #ifdef ENABLE_XONXOFF
          if ((get_rx_buffer_count() >= RX_BUFFER_FULL) && flow_ctrl ==
XON_SENT) {
            flow_ctrl = SEND_XOFF;
            UCSR0B |=  (1 << UDRIE0); // Force TX
          }
        #endif

      }
```

```
  }
}

void serial_reset_read_buffer()
{
  rx_buffer_tail = rx_buffer_head;

  #ifdef ENABLE_XONXOFF
    flow_ctrl = XON_SENT;
  #endif
}
```

Steeper // configuracion y operacion de los motores a pasos

```
/*
  stepper.c - stepper motor driver: executes motion plans using stepper
motors
#include <avr/interrupt.h>
#include "stepper.h"
#include "config.h"
#include "settings.h"
#include "planner.h"

#define TICKS_PER_MICROSECOND (F_CPU/1000000)
#define CYCLES_PER_ACCELERATION_TICK
((TICKS_PER_MICROSECOND*1000000)/ACCELERATION_TICKS_PER_SECOND)

typedef struct {
  int32_t counter_x,        // Counter variables for the bresenham line
tracer
          counter_y,
          counter_z;
  uint32_t event_count;
  uint32_t step_events_completed;  // The number of step events left in
current motion

  uint32_t cycles_per_step_event;        // The number of machine
cycles between each step event
  uint32_t trapezoid_tick_cycle_counter; // The cycles since last
trapezoid_tick. Used to generate ticks at a steady
                                         // pace without
allocating a separate timer
  uint32_t trapezoid_adjusted_rate;      // The current rate of
step_events according to the trapezoid generator
  uint32_t min_safe_rate;  // Minimum safe rate for full deceleration
rate reduction step. Otherwise halves step_rate.
} stepper_t;

static stepper_t st;
static block_t *current_block;  // A pointer to the block currently
being traced

static uint8_t step_pulse_time; // Step pulse reset time after step
rise
```

```
static uint8_t out_bits;        // The next stepping-bits to be output
static volatile uint8_t busy;   // True when SIG_OUTPUT_COMPARE1A is
being serviced. Used to avoid retriggering that handler.

#if STEP_PULSE_DELAY > 0
  static uint8_t step_bits;  // Stores out_bits output to complete the
step pulse delay
#endif
void st_wake_up()
{
  if (bit_istrue(settings.flags,BITFLAG_INVERT_ST_ENABLE)) {
    STEPPERS_DISABLE_PORT |= (1<<STEPPERS_DISABLE_BIT);
  } else {
    STEPPERS_DISABLE_PORT &= ~(1<<STEPPERS_DISABLE_BIT);
  }
  if (sys.state == STATE_CYCLE) {
    out_bits = (0) ^ (settings.invert_mask);
    #ifdef STEP_PULSE_DELAY
computation from oscilloscope.
      step_pulse_time = -
(((settings.pulse_microseconds+STEP_PULSE_DELAY-
2)*TICKS_PER_MICROSECOND) >> 3);
      OCR2A = -(((settings.pulse_microseconds)*TICKS_PER_MICROSECOND)
>> 3);
    #else // Normal operation
oscilloscope. Uses two's complement.
      step_pulse_time = -(((settings.pulse_microseconds-
2)*TICKS_PER_MICROSECOND) >> 3);
    #endif
    TIMSK1 |= (1<<OCIE1A);
  }
}

// Stepper shutdown
void st_go_idle()
{
  TIMSK1 &= ~(1<<OCIE1A);
setting to not be kept enabled.
  if ((settings.stepper_idle_lock_time != 0xff) ||
bit_istrue(sys.execute,EXEC_ALARM)) {
    delay_ms(settings.stepper_idle_lock_time);
    if (bit_istrue(settings.flags,BITFLAG_INVERT_ST_ENABLE)) {
      STEPPERS_DISABLE_PORT &= ~(1<<STEPPERS_DISABLE_BIT);
    } else {
      STEPPERS_DISABLE_PORT |= (1<<STEPPERS_DISABLE_BIT);
    }
  }
}

    return(true);
  } else {
    return(false);
  }
}

ISR(TIMER1_COMPA_vect)
{
```

```
  if (busy) { return; } // The busy-flag is used to avoid reentering
this interrupt

the steppers
  STEPPING_PORT = (STEPPING_PORT & ~DIRECTION_MASK) | (out_bits &
DIRECTION_MASK);
  #ifdef STEP_PULSE_DELAY
    step_bits = (STEPPING_PORT & ~STEP_MASK) | out_bits; // Store
out_bits to prevent overwriting.
  #else  // Normal operation
    STEPPING_PORT = (STEPPING_PORT & ~STEP_MASK) | out_bits;
  #endif
of the main Timer1 prescaler.
  TCNT2 = step_pulse_time; // Reload timer counter
  TCCR2B = (1<<CS21); // Begin timer2. Full speed, 1/8 prescaler

  busy = true;
  sei();

buffer
  if (current_block == NULL) {
    // Anything in the buffer? If so, initialize next motion.
    current_block = plan_get_current_block();
    if (current_block != NULL) {
      if (sys.state == STATE_CYCLE) {
        // During feed hold, do not update rate and trap counter. Keep
decelerating.
        st.trapezoid_adjusted_rate = current_block->initial_rate;
        set_step_events_per_minute(st.trapezoid_adjusted_rate); //
Initialize cycles_per_step_event
        st.trapezoid_tick_cycle_counter =
CYCLES_PER_ACCELERATION_TICK/2; // Start halfway for midpoint rule.
      }
      st.min_safe_rate = current_block->rate_delta + (current_block-
>rate_delta >> 1); // 1.5 x rate_delta
      st.counter_x = -(current_block->step_event_count >> 1);
      st.counter_y = st.counter_x;
      st.counter_z = st.counter_x;
      st.event_count = current_block->step_event_count;
      st.step_events_completed = 0;
    } else {
      st_go_idle();
      bit_true(sys.execute,EXEC_CYCLE_STOP); // Flag main program for
cycle end
    }
  }

  if (current_block != NULL) {
algorithm
    out_bits = current_block->direction_bits;
    st.counter_x += current_block->steps_x;
    if (st.counter_x > 0) {
      out_bits |= (1<<X_STEP_BIT);
      st.counter_x -= st.event_count;
      if (out_bits & (1<<X_DIRECTION_BIT)) { sys.position[X_AXIS]--; }
      else { sys.position[X_AXIS]++; }
    }
```

```
    st.counter_y += current_block->steps_y;
    if (st.counter_y > 0) {
      out_bits |= (1<<Y_STEP_BIT);
      st.counter_y -= st.event_count;
      if (out_bits & (1<<Y_DIRECTION_BIT)) { sys.position[Y_AXIS]--; }
      else { sys.position[Y_AXIS]++; }
    }
    st.counter_z += current_block->steps_z;
    if (st.counter_z > 0) {
      out_bits |= (1<<Z_STEP_BIT);
      st.counter_z -= st.event_count;
      if (out_bits & (1<<Z_DIRECTION_BIT)) { sys.position[Z_AXIS]--; }
      else { sys.position[Z_AXIS]++; }
    }

    st.step_events_completed++; // Iterate step events
    if (st.step_events_completed < current_block->step_event_count) {
      if (sys.state == STATE_HOLD) {
        if ( iterate_trapezoid_cycle_counter() ) {
          if (st.trapezoid_adjusted_rate <= current_block->rate_delta)
{
            st_go_idle();
            bit_true(sys.execute,EXEC_CYCLE_STOP); // Flag main
          } else {
            st.trapezoid_adjusted_rate -= current_block->rate_delta;
            set_step_events_per_minute(st.trapezoid_adjusted_rate);
          }
        }

      } else {

        if (st.step_events_completed < current_block->accelerate_until)
{
increased.
          if ( iterate_trapezoid_cycle_counter() ) {
            st.trapezoid_adjusted_rate += current_block->rate_delta;
            if (st.trapezoid_adjusted_rate >= current_block-
>nominal_rate) {
              st.trapezoid_adjusted_rate = current_block->nominal_rate;
            }
            set_step_events_per_minute(st.trapezoid_adjusted_rate);
          }
        } else if (st.step_events_completed >= current_block-
>decelerate_after) {
          if (st.step_events_completed == current_block->
decelerate_after) {
            if (st.trapezoid_adjusted_rate == current_block-
>nominal_rate) {
              st.trapezoid_tick_cycle_counter =
CYCLES_PER_ACCELERATION_TICK/2; // Trapezoid profile
            } else {
              st.trapezoid_tick_cycle_counter =
CYCLES_PER_ACCELERATION_TICK-st.trapezoid_tick_cycle_counter; //
Triangle profile
            }
          } else {
            if ( iterate_trapezoid_cycle_counter() ) {
```

```
              if (st.trapezoid_adjusted_rate > st.min_safe_rate) {
                st.trapezoid_adjusted_rate -= current_block-
>rate_delta;
              } else {
                st.trapezoid_adjusted_rate >>= 1;
          }
        }
      }
    } else {
      // If current block is finished, reset pointer
      current_block = NULL;
      plan_discard_current_block();
    }
  }
  out_bits ^= settings.invert_mask;  // Apply step and direction invert
mask
  busy = false;
}

ISR(TIMER2_OVF_vect)
{
  STEPPING_PORT = (STEPPING_PORT & ~STEP_MASK) | (settings.invert_mask
& STEP_MASK);
  TCCR2B = 0; // Disable Timer2 to prevent re-entering this interrupt
when it's not needed.
}

#ifdef STEP_PULSE_DELAY
  ISR(TIMER2_COMPA_vect)
  {
    STEPPING_PORT = step_bits; // Begin step pulse.
  }
#endif

void st_reset()
{
  memset(&st, 0, sizeof(st));
  set_step_events_per_minute(MINIMUM_STEPS_PER_MINUTE);
  current_block = NULL;
  busy = false;
}

void st_init()
{
 STEPPING_DDR |= STEPPING_MASK;
  STEPPING_PORT = (STEPPING_PORT & ~STEPPING_MASK) |
settings.invert_mask;
  STEPPERS_DISABLE_DDR |= 1<<STEPPERS_DISABLE_BIT;

  TCCR1B &= ~(1<<WGM13);
  TCCR1B |=  (1<<WGM12);
  TCCR1A &= ~(1<<WGM11);
  TCCR1A &= ~(1<<WGM10);

  TCCR1A &= ~(3<<COM1A0);
  TCCR1A &= ~(3<<COM1B0);
```

```
  TCCR2A = 0; // Normal operation
  TCCR2B = 0; // Disable timer until needed.
  TIMSK2 |= (1<<TOIE2); // Enable Timer2 Overflow interrupt
  #ifdef STEP_PULSE_DELAY
    TIMSK2 |= (1<<OCIE2A); // Enable Timer2 Compare Match A interrupt
  #endif

  st_wake_up();
  st_go_idle();
}

{
  uint16_t ceiling;
  uint8_t prescaler;
  uint32_t actual_cycles;
  if (cycles <= 0xffffL) {
    ceiling = cycles;
    prescaler = 1; // prescaler: 0
    actual_cycles = ceiling;
  } else if (cycles <= 0x7ffffL) {
    ceiling = cycles >> 3;
    prescaler = 2; // prescaler: 8
    actual_cycles = ceiling * 8L;
  } else if (cycles <= 0x3fffffL) {
    ceiling =  cycles >> 6;
    prescaler = 3; // prescaler: 64
    actual_cycles = ceiling * 64L;
  } else if (cycles <= 0xffffffL) {
    ceiling =  (cycles >> 8);
    prescaler = 4; // prescaler: 256
    actual_cycles = ceiling * 256L;
  } else if (cycles <= 0x3ffffffL) {
    ceiling = (cycles >> 10);
    prescaler = 5; // prescaler: 1024
    actual_cycles = ceiling * 1024L;
  } else {
    ceiling = 0xffff;
    prescaler = 5;
    actual_cycles = 0xffff * 1024;
  }
  TCCR1B = (TCCR1B & ~(0x07<<CS10)) | (prescaler<<CS10);
  OCR1A = ceiling;
  return(actual_cycles);
}

static void set_step_events_per_minute(uint32_t steps_per_minute)
{
  if (steps_per_minute < MINIMUM_STEPS_PER_MINUTE) { steps_per_minute =
MINIMUM_STEPS_PER_MINUTE; }
  st.cycles_per_step_event =
config_step_timer((TICKS_PER_MICROSECOND*1000000*60)/steps_per_minute);
}

void st_cycle_start()
{
  if (sys.state == STATE_QUEUED) {
    sys.state = STATE_CYCLE;
```

```
    st_wake_up();
  }
}

// Execute a feed hold with deceleration, only during cycle. Called by
main program.
void st_feed_hold()
{
  if (sys.state == STATE_CYCLE) {
    sys.state = STATE_HOLD;
    sys.auto_start = false; // Disable planner auto start upon feed
hold.
  }
}


void st_cycle_reinitialize()
{
  if (current_block != NULL) {
    plan_cycle_reinitialize(current_block->step_event_count -
st.step_events_completed);
    st.trapezoid_adjusted_rate = 0; // Resumes from rest
    set_step_events_per_minute(st.trapezoid_adjusted_rate);
    st.trapezoid_tick_cycle_counter = CYCLES_PER_ACCELERATION_TICK/2;
// Start halfway for midpoint rule.
    st.step_events_completed = 0;
    sys.state = STATE_QUEUED;
  } else {
    sys.state = STATE_IDLE;
  }
}
```

Programa para Control CNC shield.

```
/*  FILE:    HC_CNC_Shield_Test


#define EN 8      /* Enable pin for all stepper outputs */

#define X_DIR 5   /* Direction pin for X axis */
#define X_STEP 2  /* Step pin for X axis */

#define Y_DIR 6   /* Direction pin for Y axis */
#define Y_STEP 3  /* Step pin for Y axis */

#define Z_DIR 7   /* Direction pin for Z axis */
#define Z_STEP 4  /* Step pin for Z axis */

#define A_DIR 13  /* Direction pin for Aux driver. Requires D13 and A-DIR pins to be
shorted */
#define A_STEP 12 /* Direction pin for Aux driver. Requires D12 and A-STEP pins to be
shorted */
```

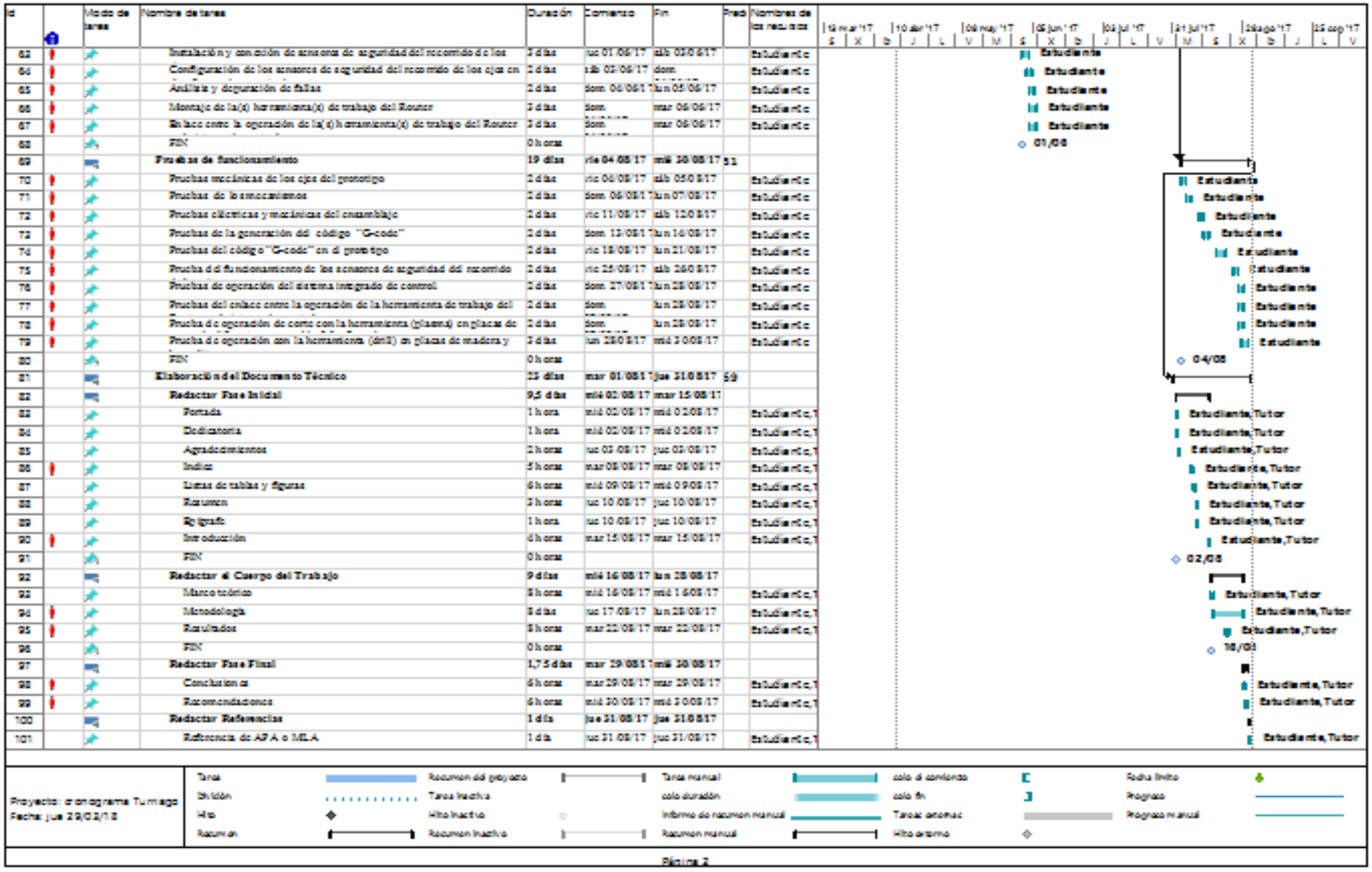
```
#define X_ENDSTOP 9   /* X axis endstop input pin */
#define Y_ENDSTOP 10  /* Y axis endstop input pin */
#define Z_ENDSTOP 11  /* Z axis endstop input pin */
#define ABORT A0  /* Abort input pin */
#define HOLD A1   /* Hold input pin */
#define RESUME A2 /* Resume input pin */

int Count = 0; /* Counter to count number of steps made */
boolean Direction = LOW; /* Rotational direction of stepper motors */

/* The setup routine runs once when you press reset: */
void setup()
{
  /* Initialize serial */
  Serial.begin(9600);

  /* Configure the steper drive pins as outputs */
  pinMode(EN, OUTPUT);
  pinMode(X_DIR, OUTPUT);
  pinMode(X_STEP, OUTPUT);
  pinMode(Y_DIR, OUTPUT);
  pinMode(Y_STEP, OUTPUT);
  pinMode(Z_DIR, OUTPUT);
  pinMode(Z_STEP, OUTPUT);
  pinMode(A_DIR, OUTPUT);
  pinMode(A_STEP, OUTPUT);

  /* Configure the control pins as inputs with pullups */
  pinMode(X_ENDSTOP, INPUT_PULLUP);
  pinMode(Y_ENDSTOP, INPUT_PULLUP);
  pinMode(Z_ENDSTOP, INPUT_PULLUP);

  pinMode(ABORT, INPUT_PULLUP);
  pinMode(HOLD, INPUT_PULLUP);
  pinMode(RESUME, INPUT_PULLUP);

  /* Enable the X, Y, Z & Aux stepper outputs */
  digitalWrite(EN, LOW); //Low to enable
}

// the loop routine runs over and over again forever:
void loop()
{
  /* Count one step */
  Count++;

  /* If we have reached 500 steps then change the stepper direction and reset the counter */
  if (Count >= 500)
  {
```

```
  Direction = !Direction;
  digitalWrite(X_DIR, Direction); // Low = CW
  digitalWrite(Y_DIR, Direction); // Low = CW
  digitalWrite(Z_DIR, Direction); // Low = CW
  digitalWrite(A_DIR, Direction); // Low = CW
  Count = 0;
}

/* Step the X, Y, Z, and Aux motors */
digitalWrite(X_STEP, HIGH);
delay(1);
digitalWrite(Y_STEP, HIGH);
delay(1);
digitalWrite(Z_STEP, HIGH);
delay(1);
digitalWrite(A_STEP, HIGH);
delay(1);

digitalWrite(X_STEP, LOW);
delay(1);
digitalWrite(Y_STEP, LOW);
delay(1);
digitalWrite(Z_STEP, LOW);
delay(1);
digitalWrite(A_STEP, LOW);
delay(1);

/* Check state of inputs */
if (!digitalRead(X_ENDSTOP)) Serial.println("X-axis end-stop triggered!");
if (!digitalRead(Y_ENDSTOP)) Serial.println("Y-axis end-stop triggered!");
if (!digitalRead(Z_ENDSTOP)) Serial.println("Z-axis end-stop triggered!");
if (!digitalRead(ABORT)) Serial.println("Abort input triggered!");
if (!digitalRead(HOLD)) Serial.println("Hold input triggered!");
if (!digitalRead(RESUME)) Serial.println("Resume input triggered!");
```

Id		Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
1			**PROYECTO FIC**	60 días	vie 09/06/17	jue 31/08/17		
2			**Definicion del Plan de Proyecto de Tesis**	18,5 días	mar 11/04/17	vie 05/05/17		
15			**Planificación del Diseño**	19,75 días	mar 09/05/17	lun 05/06/17	2	
16			**Recopilación de información**	13,13 días	mar 09/05/17	vie 26/05/17		
28			**Elegir materiales a usar**	3,63 días	vie 26/05/17	mié 31/05/17		
29			Arduino 1 y Arduino Mega	2 horas	vie 26/05/17	vie 26/05/17		Estudiante
30	!		Plasma PowerMax30 AIR	4 horas	sáb 27/05/17	sáb 27/05/17		Estudiante
31	!		Motor paso a paso Wantai Mini Stepper	4 horas	dom 28/05/17	dom 28/05/17		Estudiante
32	!		Diferente espesor del Aluminio usado en la Mesa	6 horas	lun 29/05/17	lun 29/05/17		Estudiante
33	!		Diferente espesor en los Angulos que sogortaran la Mesa	8 horas	lun 29/05/17	lun 29/05/17		Estudiante
34			Selección de sensores	5 horas	mié 31/05/17	mié 31/05/17		Estudiante
35			FIN	0 horas				
36			**Pruebas de Funcionamiento**	0,75 días	sáb 03/06/17	lun 05/06/17		
37	!		Verificar el funcionamiento del plasma al 100%	6 horas	sáb 03/06/17	sáb 03/06/17		Estudiante
38	!		Verificar el funcionamiento de los Arduinos	6 horas	dom 04/06/17	dom 04/06/17		Estudiante
39	!		Verificar el funcionamiento perfecto del torque y rpm de los motores	6 horas	lun 05/06/17	lun 05/06/17		Estudiante
40			FIN	0 horas				
41			**Diseño del Sistema**	18 días	mar 06/06/17	jue 29/06/17	15	
42	!		Diseño Mecánico del Prototipo	3 días	mar 06/06/17	jue 08/06/17		Estudiante
43	!		Diseño de la interfaz de operación de la multiherramienta	2 días	vie 09/06/17	dom 11/06/17		Estudiante
44			Diseño control de motores	2 días	lun 12/06/17	mar 13/06/17		Estudiante
45			Configuración de motores por eje	3 días	mié 14/06/17	vie 16/06/17		Estudiante
46	!		Configuración del controlador electrónico	2 días	sáb 17/06/17	dom 18/06/17		Estudiante
47	!		Configuración de sensores	3 días	dom 18/06/17	mar 20/06/17		Estudiante
48			Diseño del control de posición de la herramienta en el eje Z, considerando	3 días	jue 22/06/17	lun 26/06/17		Estudiante
49			Diseño interfaz Hombre Maquina	3 días	mar 27/06/17	jue 29/06/17		Estudiante
50			FIN	0 horas				
51			**Implementacion del Diseño**	43 días	jue 01/06/17	lun 31/07/17	41	
52			Ensamblaje del prototipo	2 días	vie 30/06/17	lun 03/07/17		Estudiante
53	!		Montaje de los mecanismos	2 días	vie 07/07/17	sáb 08/07/17		Estudiante
54	!		Montaje de los motores	2 días	dom 09/07/17	lun 10/07/17		Estudiante
55			Montaje y grudas del control del motor	1 día	vie 14/07/17	vie 14/07/17		Estudiante
56	!		Instalación del firmware del tipo CNC al controlador	2 días	sáb 15/07/17	lun 17/07/17		Estudiante
57	!		Instalación del software para manejo de motores	1 día	lun 17/07/17	lun 17/07/17		Estudiante
58			Comunicación entre el software de manejo de motores y el controlador	1 día	vie 21/07/17	vie 21/07/17		Estudiante
59	!		Implementación del la interfaz de operación de la multiherramienta	2 días	sáb 22/07/17	dom		Estudiante
60	!		Instalación de software de diseño de Router	2 días	dom 23/07/17	lun 24/07/17		Estudiante
61	!		Generación de código "G-code" para Router	2 días	vie 28/07/17	dom 30/07/17		Estudiante
62	!		Importar código "G-code" a software de manejo de motores	2 días	dom 30/07/17	lun 31/07/17		Estudiante

Proyecto: cronograma Tumiago
Fecha: jue 29/03/18

Tarea | Resumen del proyecto | Tarea manual | solo el comienzo | Fecha límite
División | Tarea inactiva | solo duración | solo fin | Progreso
Hito | Hito inactivo | Informe de resumen manual | Tareas externas | Progreso manual
Resumen | Resumen inactivo | Resumen manual | Hito externo



Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Pred	Nombres de los recursos
62		Instalación y conexión de sensores de seguridad del recorrido de los	2 días	vie 01/09/17	sáb 02/09/17		Estudiante
64		Configuración de los sensores de seguridad del recorrido de los ejes en	2 días	sáb 03/06/17	dom		Estudiante
65		Análisis y depuración de fallas	2 días	dom 04/06/17	lun 05/06/17		Estudiante
66		Montaje de la(s) herramienta(s) de trabajo del Router	3 días	dom	mar 06/06/17		Estudiante
67		Enlace entre la operación de la(s) herramienta(s) de trabajo del Router	2 días	dom	mar 06/06/17		Estudiante
68		FIN	0 horas				01/06
69		**Pruebas de funcionamiento**	**19 días**	**vie 04/08/17**	**mié 30/08/17**	51	
70		Pruebas mecánicas de los ejes del prototipo	2 días	vie 04/08/17	sáb 05/08/17		Estudiante
71		Pruebas de lo emocanismos	2 días	dom 06/08/17	lun 07/08/17		Estudiante
72		Pruebas eléctricas y mecánicas del ensamblaje	2 días	vie 11/08/17	sáb 12/08/17		Estudiante
73		Pruebas de la generación del código "G-code"	2 días	dom 13/08/17	lun 14/08/17		Estudiante
74		Pruebas del código "G-code" en el prototipo	2 días	vie 18/08/17	lun 21/08/17		Estudiante
75		Prueba del funcionamiento de los sensores de seguridad del recorrido	2 días	vie 25/08/17	sáb 26/08/17		Estudiante
76		Pruebas de operación del sistema integrado de control	2 días	dom 27/08/17	lun 28/08/17		Estudiante
77		Pruebas del enlace entre la operación de la herramienta de trabajo del	2 días	dom	lun 28/08/17		Estudiante
78		Prueba de operación de corte con la herramienta (plasma) en placas de	2 días	dom	lun 28/08/17		Estudiante
79		Prueba de operación con la herramienta (drill) en placas de madera y	3 días	lun 28/08/17	mié 30/08/17		Estudiante
80		FIN	0 horas				04/06
81		**Elaboración del Documento Técnico**	**25 días**	**mar 01/08/17**	**jue 31/08/17**	59	
82		**Redactar Fase Inicial**	**9,5 días**	**mié 02/08/17**	**mar 15/08/17**		
83		Portada	1 hora	mié 02/08/17	mié 02/08/17		Estudiante,Tutor
84		Dedicatoria	1 hora	mié 02/08/17	mié 02/08/17		Estudiante,Tutor
85		Agradecimientos	2 horas	jue 03/08/17	jue 03/08/17		Estudiante,Tutor
86		Indice	5 horas	mar 08/08/17	mar 08/08/17		Estudiante,Tutor
87		Listas de tablas y figuras	6 horas	mié 09/08/17	mié 09/08/17		Estudiante,Tutor
88		Resumen	2 horas	jue 10/08/17	jue 10/08/17		Estudiante,Tutor
89		Epígrafe	1 hora	jue 10/08/17	jue 10/08/17		Estudiante,Tutor
90		Introducción	4 horas	mar 15/08/17	mar 15/08/17		Estudiante,Tutor
91		FIN	0 horas				02/08
92		**Redactar el Cuerpo del Trabajo**	**9 días**	**mié 16/08/17**	**lun 28/08/17**		
93		Marco teórico	8 horas	mié 16/08/17	mié 16/08/17		Estudiante,Tutor
94		Metodología	8 días	jue 17/08/17	lun 28/08/17		Estudiante,Tutor
95		Resultados	8 horas	mar 22/08/17	mar 22/08/17		Estudiante,Tutor
96		FIN	0 horas				16/08
97		**Redactar Fase Final**	**1,75 días**	**mar 29/08/17**	**mié 30/08/17**		
98		Conclusiones	6 horas	mar 29/08/17	mar 29/08/17		Estudiante,Tutor
99		Recomendaciones	6 horas	mié 30/08/17	mié 30/08/17		Estudiante,Tutor
100		**Redactar Referencias**	**1 día**	**jue 31/08/17**	**jue 31/08/17**		
101		Referencia de APA o MLA	1 día	jue 31/08/17	jue 31/08/17		Estudiante,Tutor

Id		Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Pred.	Nombres de los recursos
102			FIN	0 horas				
103			Redactar Apéndices	22 días	mar 01/08/17	mié 30/08/17		
104			Listas de abreviaturas o siglas ordenadas alfabéticamente	3 horas	mar 01/08/17	mar 01/08/17		Estudiante,Tutor
105	!		Datos concretos sobre instrumentos de investigación	6 horas	mar 08/08/17	mar 08/08/17		Estudiante,Tutor
106	!		Dibujos, diagramas, fotos o cualquier otro material similar	7 horas	mar 15/08/17	mar 15/08/17		Estudiante,Tutor
107	!		Gráficas, tablas y cuadros estadísticos	2 días	mar 29/08/17	mié 30/08/17		Estudiante,Tutor
108			FIN	0 horas				
109								
110