Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de un controlador básico de impresión 3D

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Vicente Morales Muñoz

Tutor: Miguel Sánchez López

Curso 2016-2017

Dedicado a la memoria de mi abuela Teresa que tanto me ha apoyado a lo largo de la carrera pero que no podrá estar conmigo en el final de esta.

Resumen

En este proyecto se abordará el proceso de implementación de un controlador de impresión 3D intentado que este sea muy sencillo y fácil de comprender dado que hoy en día a pesar de que abundan los controladores de este tipo, pocos apuestan por la sencillez y esto es un problema para la gente que empieza con maquinas CNC. Este controlador principalmente constará de un intérprete de G-Code y de las funciones necesarias para realizar movimientos lineales y llevar un control de temperatura sencillo pero eficaz.

Palabras clave: Arduino, G-Code, Impresión 3D, CNC, Controlador.

Abstract

In this project, we will aboard the process of implementation of a 3D print driver in an attempt to make it very simple and easy to understand, given that today there are a lot of drivers of this type, few are betting on simplicity and this is a problem for people who start with CNC. This controller will basically consist in a G-code interpreter and the functions necessary to perform linear movements and carry out a simple but effective temperature control.

Keywords : Arduino, G-Code, 3D printing, CNC, Controller.

Índice de ilustraciones

Índice de contenidos

1 Introducción

El termino Impresión 3D hace referencia a una serie de procesos y tecnologías que sirven para la producción de productos a través de un proceso de adición de material en 3 dimensiones o ejes X, Y, Z. La impresión 3D es una tecnología muy presente en nuestro día a día y cada vez va en aumento llegando a ser considerada por unos cuantos como una nueva revolución industrial.

Esta tecnología es probablemente una de las más importantes en los últimos años. Ha permitido tanto avances en el mundo de la industria como en ámbitos aún más sensibles como el sanitario, esta tecnología ha facilitado la fabricación de prótesis que ayudan a mejorar la calidad de vida de una gran cantidad de personas y hoy en día se sigue investigando sobre esta vía con grandes resultados.

Ilustración 1: Prótesis de una mano realizada con una impresora 3D

¿Pero cómo funcionan las impresoras 3D?

El proceso de impresión consiste en ir imprimiendo un diseño previamente creado por ordenador capa a capa. Se deposita una capa de plástico utilizando un extrusor que lo que hace es calentar un filamento de plástico y lo va expulsando a la temperatura adecuada para depositarlo en la cama o base de la impresora, cuando se ha creado una capa se pasa a la capa superior, repitiendo el proceso hasta que la pieza está terminada.

Pero nos dejamos un detalle importante que es el cómo se pasa las ordenes necesarias que tiene que llevar a cabo la impresora para poder realizar el diseño que previamente hemos realizado por ordenador y como se controla que este proceso se está haciendo de la forma correcta. Este es el objetivo de los controladores, los controladores hacen de interfaz entre la maquina y el ordenador, reciben las ordenes del ordenador, las interpretan y ponen en marcha los actuadores necesarios para ejecutar la orden.

Además interpretan las lecturas que ofrecen los sensores de la máquina para conocer su estado y mostrarlo al usuario y/o detener la maquina en caso de un funcionamiento inadecuado. En definitiva, sin un controlador que permita la comunicación ordenador-maquina sería imposible hacer realidad cualquier diseño.

1.1 Motivación

Al estar en un momento en el que hay una gran comunidad en el ámbito de las impresoras 3D que desarrolla firmware propicia que haya una gran cantidad de controladores disponibles que con el tiempo han ido evolucionando. Esta evolución implica mas funcionalidades, más eficiencia o adaptabilidad, lo que conlleva que los firmwares actuales sean muy complicados para los usuarios noveles, muchas librerías, líneas de código, funciones extra... Pocos firmwares apuestan por ser accesibles, legibles y sencillos de entender. Nosotros hemos tomado como referencia varios de estos firmwares como base para nuestro proyecto.

1.2.1 Marlin

Marlin es un firmware de código abierto y de carácter comunitario para las impresoras RepRap o “auto replicables”. Es un firmware que toma como base entre otros el firmware para maquinaria CNC GRBL. Pasó a ser un proyecto abierto en 2011 con una versión que fue subida al portal GitHub. El firmware esta licenciado bajo GPLv3 y es totalmente libre.

Ilustración 2: Logotipo de Marlin

Desde sus orígenes Marlin ha sido desarrollado por aficionados y entusiastas de estas tecnologías con el objetivo de crear un firmware que fuera fiable y adaptable a una gran variedad de impresoras. Tal fue su éxito que varias compañías dedicas al sector han desarrollado su propio firmware como una variante del propio Marlin, empresas tales como Ultimate, Printrbot o Prusa. Otra de las claves del éxito de Marlin es su bajo coste, capaz de ejecutarse en micro-controladores de 8-bit. Su plataforma de referencia es el Arduino Mega2560 con RAMPS 1.4.

La característica a tener más en cuenta en Marlin es su gestión de las órdenes. Su lenguaje referencia es una variable del lenguaje G-Code implementado una gran variedad de comandos. En Marlin los comandos se procesan utilizando una cola, según se van añadiendo comandos se van almacenando en esa cola para que una vez finalizada la orden en ejecución pase a ejecutar la siguiente en orden de introducción.

1.2.2 GRBL

GRBL es un software de alto rendimiento y de código abierto para el control de cualquier tipo de maquinaria CNC, ya puedan ser impresoras 3D, grabadoras laser o fresadoras. Liberado en 2011 GRBL es considerado por la mayoría como un referente en el mundo de los firmwares de control dado que muchos de los actuales están desarrollados sobre el núcleo de GRBL al que han ido añadiendo funcionalidades propias para adaptarlos al tipo de CNC al que estarían destinados. Al igual que Marlin, GRBL está bajo la licencia GPLv3.

GRBL ha sido creado para poder trabajar con una amplia gama de aplicaciones de grabado, con una gran capacidad de adaptación capaz de funcionar tanto con equipos de altas prestaciones como con el más simple de los sistemas. Su plataforma suele ser Arduino.

Escrito en una versión optimizada de C es capaz de aprovechar al máximo las características de Arduino obteniendo unas tasas de sincronización excelentes. Implementa un control de pulso capaz de alcanzar una frecuencia de paso de motor de unos 30kHz.

Aunque quizás la característica más interesante de GRBL sea su gestión de la aceleración, utilizando una aceleración “hacia adelante”, esto quiere decir que es capaz

de predecir el movimiento futuro y adaptar la aceleración para realizarla de la manera más suave posible.

1.3 Objetivo

Se ha visto que a pesar de la gran cantidad de software hay actualmente, hay pocos que sean sencillos y fáciles de comprender. Por lo tanto el objetivo principal en este proyecto va a ser el desarrollar un firmware para el ámbito de la impresión 3D con las funcionalidades básicas que debe tener un software de este tipo, con una implementación lo más sencilla posible y de bajo coste que permita acercar esta tecnología a usuarios inexpertos. Principalmente deberá de atenerse a las siguientes premisas:

- Estar escrito con un código sencillo y entendible.
- Interpretar un juego de instrucciones básicas que permitan controlar las funciones principales de la maquina.
- Implementar un control de temperatura funcional tanto para la cama como para el extrusor.
- Servir como base para proyectos más amplios.
- Implementar un movimiento lineal en los ejes X, Y, Z además del extrusor que permita controlar la velocidad de este.

Además del objetivo principal, también sería interesante poder comprender mejor el funcionamiento de los controladores de maquinas CNC así como ganar soltura en la programación de Arduino y aprender a aprovechar el potencial que puede ofrecer esta tecnología para luego poder aplicarla a proyectos propios.

1.4 Estructura del trabajo

Este memoria está dividida principalmente en 7 capítulos que servirán para ofrecer una visión general de en qué consiste un controlador para impresoras 3D y en concreto una descripción de la estructura del firmware que se ha desarrollado a lo largo de este proyecto.

- En el primer capítulo pretende servir de introducción a la temática de la que trata esta memoria. Se tratará en qué consiste la impresión 3D y el porqué del impacto que está ganando durante estos últimos años. Más adelante se tratara el contexto en el que nos hayamos, el de los controladores, describiendo dos software de CNC conocidos y que han servido de referencia para realizar este trabajo. Finalmente se hablará de los objetivos que se esperan alcanzar.

- Para el segundo capítulo se describirá la plataforma en la que nos hemos basado para implementar nuestro controlador, que es la de Arduino con RAMPS 1.4, explicando también las razones que nos han llevado a optar por este hardware en concreto. Además de los pasos que se han realizado para preparar el entorno como paso previo a la implementación.

- Entrados en el tercer capítulo ya se empieza a hablar de la implementación del software. En concreto nos centraremos en el intérprete de comandos además de hablar del lenguaje utilizado en este proyecto, el G-Code y de los comandos de este lenguaje que se han implementado.

- En el cuarto capítulo trataremos como se han implementado los movimientos. En primera instancia se describirán los motores paso a paso, para poder entender cómo funcionan y como hemos implementado su control. Luego se llegará a la implementación del movimiento lineal. Aquí se describirá el algoritmo de Bresenham (Algoritmo que hemos utilizado para el movimiento lineal) de forma teórica además de la implementación del mismo adaptada a nuestro proyecto. El capitulo se cierra con la explicación de cómo se ha implementado el reinicio de los ejes o “homing”.

- En el quinto capítulo, que es el último sobre la implementación es en el que se trata el control de temperatura en el que explicaremos como hemos implementado la lectura de la temperatura haciendo uso de la interpolación

lineal, su tratamiento así como las rutinas que hemos introducido para garantizar un control adecuado.

- En el sexto capítulo se tratará de un capitulo de cierre en la cual se realizará una reflexión sobre el trabajo realizado y de los objetivos que se han ido cumpliendo. También se plantearán unas propuestas para ampliar este mismo proyecto que permitan dotarlo de más funciones y eficiencia.

- El séptimo capítulo es la bibliografía en la que se podrán ver las fuentes que se han consultado para la redacción de esta memoria.

2. Plataforma

En este apartado vamos a describir la plataforma en la que nos hemos basado para realizar la implementación de nuestro firmware y como hemos configurado el proyecto para adaptarlo.

2.1 Arduino

La plataforma principal que hemos elegido para llevar a cabo nuestra implementación ha sido Arduino con RAMPS 1.4.

Arduino es una plataforma de código abierto que se compone de hardware y software. Los arduinos pueden leer entradas de una variedad de sensores (Sensores de luz, sensores de temperatura, sensores de movimiento…). Además de poder generar salidas para los diversos actuadores que se pueden conectar a la placa (LEDs, motores, pantallas LCD…).

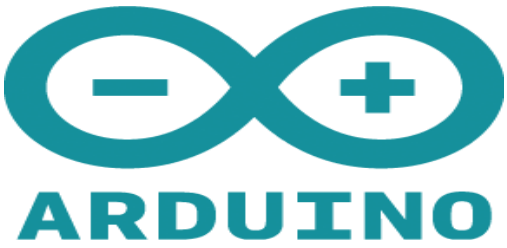

Ilustración 3: Logo de Arduino

Arduino ha supuesto un punto de partida para miles de proyectos que pueden ir desde un pequeño control casero hasta proyectos de investigación más complejos. A lo largo de los años ha ido desarrollando una amplia comunidad que colabora en el desarrollo de software de código abierto para esta plataforma. Surgido como una solución simple para prototipado ha ido evolucionando y hasta contar con una gran variedad de elementos que permiten trabajar en diversos ámbitos como en la IoT o la impresión 3D.

Hemos elegido Arduino por que cumple una serie de características que encajan con la propuesta de este proyecto de apostar por la sencillez y el bajo coste, como pueden ser:

- Bajo coste en comparación con otras plataformas de propósito parecido.

- Una gran compatibilidad: Permite que programas desarrollados en un sistema operativo concreto se puedan continuar desarrollando en cualquier otro.

- Programación sencilla: Arduino utiliza un lenguaje de programación y un IDE sencillos de utilizar y que mejoran la tarea de desarrollo. Su lenguaje es una versión de C++ con funciones propias.

- Una gran comunidad, por lo que hay una gran cantidad de proyectos en los que poder basarse además de foros y artículos que pueden servir de ayuda para aprender a manejarse con esta plataforma.

2.1.1 Arduino MEGA

De entre la gran variedad de placas de la plataforma de Arduino hemos elegido la Arduino Mega. Arduino Mega es un micro controlador basado en el procesador ATmega2560 de 8 bits.

Como características principales de este modelo encontramos:

- 54 pines digitales de entrada/salida.

- 16 entradas analógicas.

- 4 UART (Universal Asynchronous Receiver-Transmiter) es una interfaz de comunicación serie.

- 1 conexión USB.

- 1 conector de alimentación.

- 1 ICSP (In Circuit Serial Programming) para programar el BootLoader.

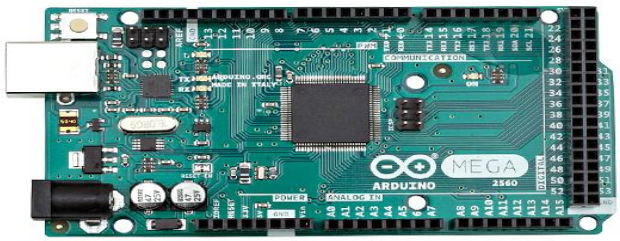

Ilustración 4: Placa Arduino MEGA

Los motivos que nos han llevado a elegir esta placa en particular como hardware base es la gran cantidad de entradas digitales y analógicas de las que dispone ya que nos permite controlar más componentes que otras placas en las que se pensó como puede ser la Arduino UNO. Además ofrece una gran compatibilidad con shields o placas de expansión como RAMPS.

2.1.2 RAMPS 1.4

Para poder controlar de una forma más sencilla los diferentes ejes y componentes de los que está compuesta una impresora 3D nos hemos decidido por una placa RAMPS.

RAMPS (RepRap Arduino Mega Pololu Shield) es un shield o placa adicional a Arduino MEGA que traduce las órdenes digitales de nuestro ordenador en órdenes por pasos, a través de los drivers para los motores paso a paso. Esta placa permite controlar de forma sencilla los motores que se corresponden con los ejes X, Y, Z y el extrusor, además de controlar las entradas de los sensores, los actuadores para los calentadores y los finales de carrera entre otras cosas. Se han desarrollado diferentes versiones de esta placa, actualmente la versión vigente es la 1.4 que es la que hemos utilizado. Son necesarios unos micro-controladores para poder controlar los motores, en este caso vamos a utilizar los micro controladores A4988, ya que son los más sencillos y baratos.

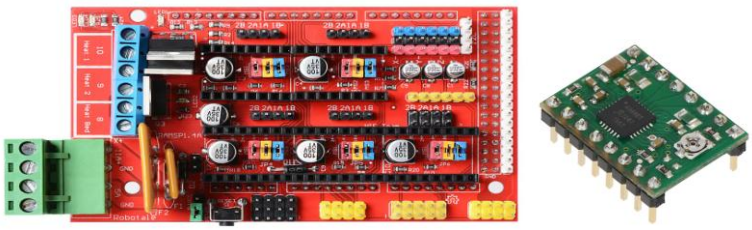

Ilustración 5: Placa RAMPS 1.4 y micro controlador A4988

2.2 Configuración

En este apartado vamos a hablar de cómo se han configurado las placas y el software de Arduino para permitir que el proyecto funcionara sobre esta plataforma.

2.2.1 IDE Arduino

Aunque el código para Arduino se puede escribir con cualquier editor de textos no es así para la compilación y el volcado de programa sobre la placa. Para ello vamos a utilizar el IDE de Arduino que facilita enormemente la tarea de programación y compilación.

El IDE de Arduino se puede descargar desde la web oficial de Arduino aunque las placas suelen ir con un CD en el que se incluye. Se instala como si de un programa cualquiera se tratase y una vez abierto esto es lo que veremos:

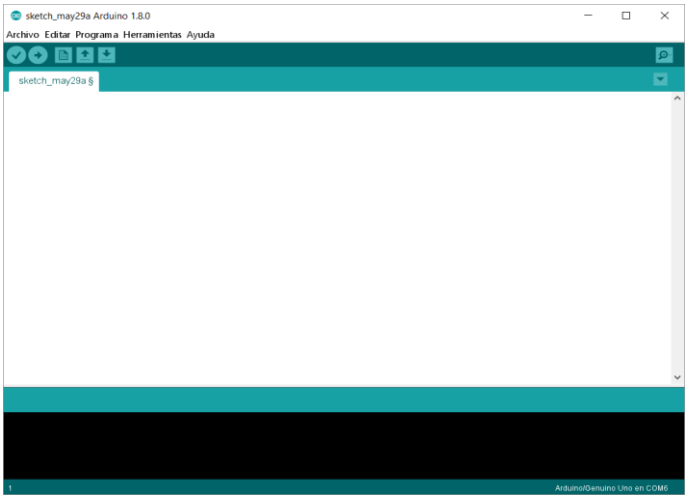

Ilustración 6: IDE Arduino

Ya que lo tenemos instalado nos faltaría indicarle el tipo de placa, el procesador que esta usa y el puerto en el que está conectada para poder subir ahí la placa.

Para indicar el tipo de placa vamos a Herramientas → Placa → "Modelo". En este caso nuestra placa es una Arduino Mega 2560, una vez seleccionada nos hará falta especificar que procesador monta la placa ya que es posible que tenga varios tipos de procesadores según el modelo. Para ello vamos a Herramientas → Procesador → "Procesador". La opción por defecto es la valida en nuestro caso.

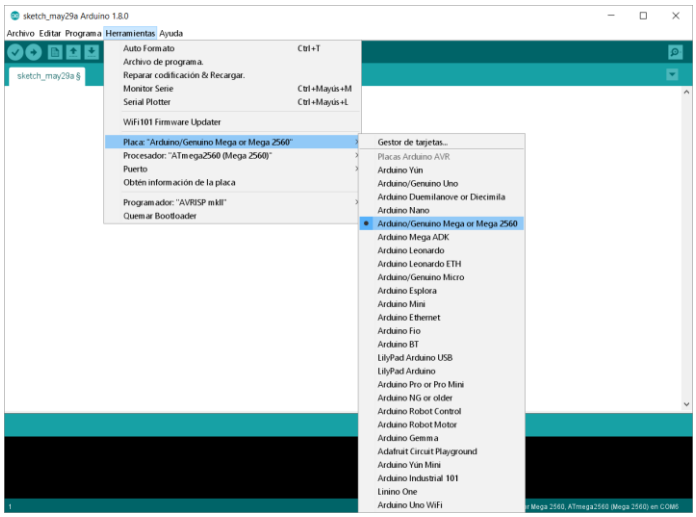

Ilustración 7: Elección de placa

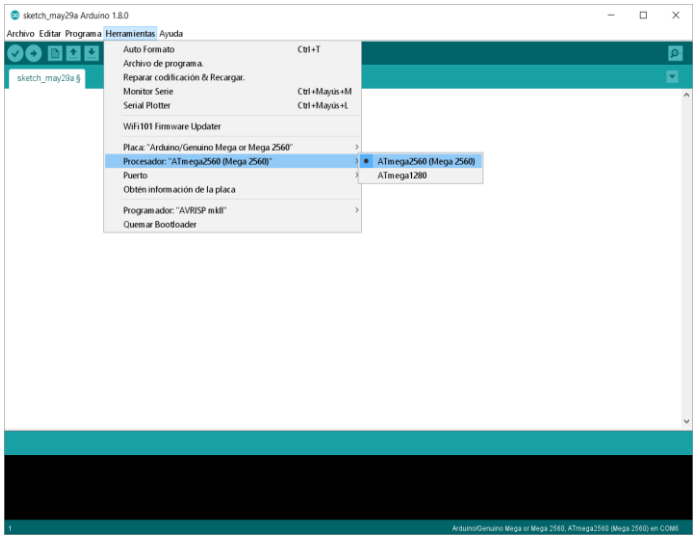

Ilustración 8: Elección de procesador

Una vez hecho todo este proceso ya tendríamos configurada nuestro Arduino con el IDE, solo nos falta elegir en que puerto está conectada la placa, para que se pueda subir el programa. En este caso lo haremos desde Herramientas → Puerto → "Puerto.

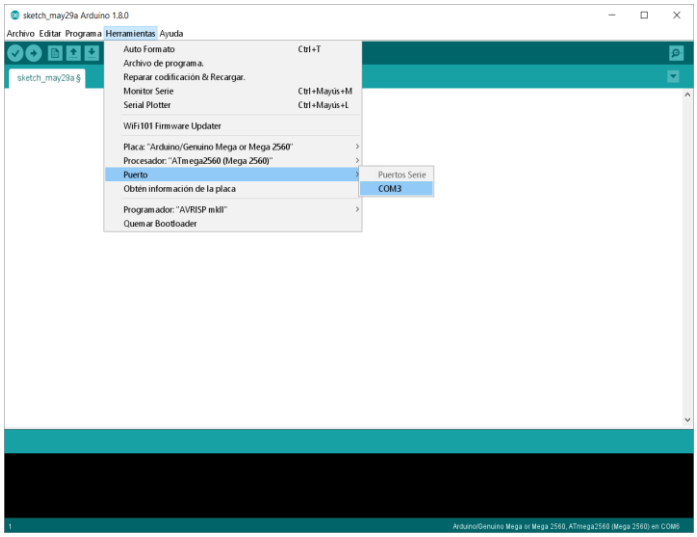

Ilustración 9: Elección del puerto

Con todo esto ya tendríamos nuestra placa lista para recibir los programas que le pasemos. Para compilar y/o subir programas a nuestra placa tenemos varias opciones como las teclas rápidas (CTRL + R y CTRL – U) desde el menú programa o desde los iconos de acción rápida que hay en la esquina superior izquierda de la pantalla del IDE.

2.2.2 Configuración de pines

Una vez ya tenemos listo nuestro entorno Arduino y tenemos definida que placas vamos a utilizar es el momento de incluir en el programa la asignación de pines para los actuadores y los sensores que vamos a controlar. Para ello vamos a ver la disposición de los elementos en la placa RAMPS que vamos a utilizar.

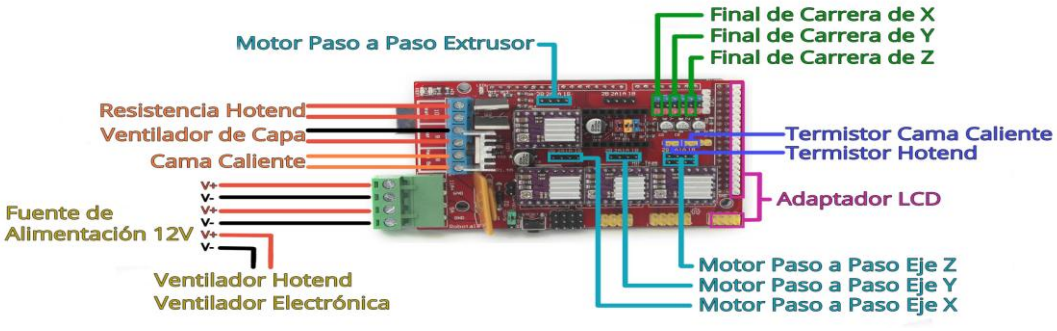

Ilustración 10: Conexiones RAMPS

Tal y como se ve en la imagen tenemos como actuadores cuatro motores (tres para los ejes y uno para el extrusor) y dos calentadores para el hot-end (extrusor) y la cama. Como sensores tenemos tres finales de carrera, uno por eje y dos termistores para el hot-end y para la cama.

Ahora que conocemos que elementos vamos a controlar y como van conectados a la RAMPS debemos de hallar la correspondencia de pines entre RAMPS y Arduino para asignarlos dentro del programa y que nuestra placa Arduino pueda actuar sobre los elementos de forma correcta. En este caso vamos a utilizar la asignación de pines que proporciona el fabricante de RAMPS para nuestro modelo el 1.4. Quedando de la siguiente forma:

Pines de paso para los motores: La excitación de estos pines provoca un paso en el motor. Están definidos como pines de salida digital.

```
#define PASOX 54   pinMode(PASOX, OUTPUT);
#define PASOY 60   pinMode(PASOY, OUTPUT);
#define PASOZ 46   pinMode(PASOZ, OUTPUT);
#define PASOE 26   pinMode(PASOE, OUTPUT);
```

Ilustración 11: Asignación y modo de los pines de paso

Pines de dirección de motores: Indica la dirección de giro del motor en sentido horario o anti-horario. Están asignados como pines de salida digital.

```
#define DIRX 55    pinMode(DIRX, OUTPUT);
#define DIRY 61    pinMode(DIRY, OUTPUT);
#define DIRZ 48    pinMode(DIRZ, OUTPUT);
#define DIRE 28    pinMode(DIRE, OUTPUT);
```

Ilustración 12: Asignación y modo de los pines de dirección

Pines de habilitación: Estos son los pines que permiten el movimiento de los motores, está activo a nivel bajo debido a la estructura del micro-controlador A4988 presente en RAMPS, por lo que un 0 es habilitado y un 1 es deshabilitado. Serian pines de salida digital.

```
#define ENABLEX 38    pinMode(ENABLEX, OUTPUT);
#define ENABLEY 56    pinMode(ENABLEY, OUTPUT);
#define ENABLEZ 62    pinMode(ENABLEZ, OUTPUT);
#define ENABLEE 24    pinMode(ENABLEE, OUTPUT);
```

Ilustración 13: Asignación y modo de los pines de habilitación

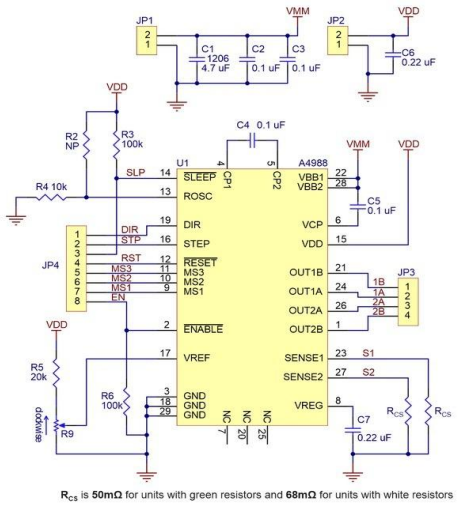

Ilustración 14: Diagrama esquemático del A4988

Pines de final de carrera: Estos son los pines que irán conectados a los finales de carrera, su activación indica que se ha llegado al final del eje, se definen como entradas digitales.

```
#define FINX 3     pinMode(FINX, INPUT);
#define FINY 14    pinMode(FINY, INPUT);
#define FINZ 18    pinMode(FINZ, INPUT);
```

Ilustración 15: Asignación y modo de los pines de final de carrera

Pines de los calentadores: Estos son los pines que activaron o desactivarán los calentadores del hot-end y de la cama. Están definidos como pines de salida digital.

```
#define CALENTADOREX 10    pinMode(CALENTADOREX, OUTPUT);
#define CALENTADORBD 8     pinMode(CALENTADORBD, OUTPUT);
```

Ilustración 16: Asignación y modo de los pines de los calentadores

Pines de los termistores: Pines que están asignados a los dos sensores de temperatura que utiliza nuestro sistema, en este caso son salidas de tipo analógico.

```
#define SENSOREX 13    pinMode(SENSOREX, INPUT);
#define SENSORBD 14    pinMode(SENSORBD, INPUT);
```

Ilustración 17: Asignación y modo de los pines de los termistores

Con todo esto ya tendríamos nuestra placa preparada para nuestro proyecto con las salidas y entradas asignadas ya solo faltaría utilizarlas correctamente durante la implementación del cuerpo principal del firmware.

3 Interprete de órdenes

En este apartado vamos a hablar de una de las funciones esenciales que debe tener un controlador 3D, la interpretación de órdenes. Esto es que el programa lea ya sea de la entrada serie o de un fichero instrucciones en lenguaje CNC con sus correspondientes parámetros (si los hubiera), comprobar que el comando leído se corresponde con uno implementado y ejecutarlo según corresponda.

Para que el programa pueda llevar a cabo esta operación es necesario definir e implementar el conjunto de códigos que van a estar disponibles para ejecución. Además de las rutinas necesarias para que se pueda leer la entrada, parsear la misma y decidir qué hacer acorde a la información recopilada.

3.1 G-Code

G-Code es el nombre de un lenguaje descriptor de operaciones para máquinas de control numérico o CNC. Estandarizado en la ISO 6983 y en la norma DIN 66025, el G-Code es uno de los lenguajes de programación para CNC más populares que existen ganando aun más notoriedad gracias a la aparición de las impresoras 3D.

Los comandos escritos en G-Code describen los movimientos y las operaciones necesarias que deben llevar a cabo las maquinas CNC. En lo referente a nuestro ámbito que es la impresión 3D G-Code permite controlar varios aspectos de la impresora, desde el movimiento de los ejes tanto de forma lineal como circular, hasta controlar las temperaturas del extrusor y de la base (cama) además de especificar a qué velocidad y en qué cantidad depositar el filamento con el material.

En cuanto al formato de los comandos en G-Code, por lo general estos comienzan con una letra que indica el tipo de función, que puede ser G, M (Funciones misceláneas o M-Code) o T (Seleccionar herramienta), seguido de un número que funciona a modo de descriptor de función. Además de poder ir acompañado de varios parámetros que se pueden corresponden cada uno con una letra del alfabeto y cada uno tiene su uso. En nuestro caso haremos uso de X, Y, Z, E para hacer referencia a los ejes y el extrusor, S para la temperatura y F para la velocidad o Feedrate.

Un programa en G-Code se trata como si fuera un fichero de texto, es decir, puede editarse de forma manual desde cualquier IDE o editor de texto, no es lo habitual dado que hoy en día se genera y visualiza el código desde aplicaciones de modelado 3D.

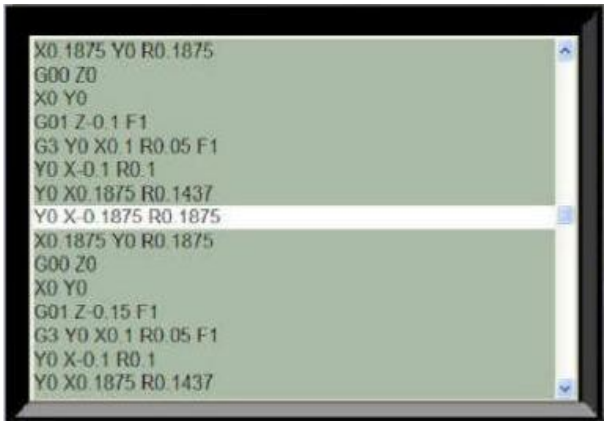

Ilustración 18: Programa en G-Code visualizado desde el software Mach3

3.2 Comandos implementados

Hay una gran cantidad de comandos en el lenguaje de G-Code, además de muchos otros, que pueden ser implementados en un controlador para dotarlo de diversas funcionalidades. Como el objetivo de este proyecto es buscar la sencillez hemos implementado solamente los comandos necesarios que permitan llevar a cabo las funciones básicas que debe tener una impresora 3D.

Los comandos G que hemos implementado son los siguientes:

- **G1 Xnn Ynn Znn Enn Fnn:** Este comando realiza el movimiento lineal y de forma coordinada de uno o varios ejes con los parámetros X, Y, Z y del extrusor con E además de permitir el control de la velocidad con la que queremos que se realice el movimiento (en mm/min) con el parámetro F. El movimiento se puede realizar con medidas absolutas o con medidas incrementales según esté configurado.

- **G28:** Este comando devuelve todos los ejes a la posición al cero maquina, con un movimiento rápido y en orden secuencial Z, Y, X para evitar interferencias con la pieza. También sería posible poder elegir cuál de los ejes queremos que vuelva a origen para una mayor funcionalidad, no obstante no se ha incluido en nuestro controlador ya que podría complicar el código y no es lo que buscamos.

- **G90:** Este comando sirve para indicarle al control que queremos definir las medidas de forma absoluta, esto es en relación al cero maquina. Aplicado a G1 con los parámetros X, Y, Z indicaremos la posición a la que queremos que vayan los ejes independientemente de donde estén ubicados.

- **G91:** Este comando sirve para indicarle al control que queremos definir las medidas de forma incremental, esto es en relación a la posición actual de los ejes. Esto con G1 quiere decir que con los parámetros X, Y, Z estamos indicando que queremos que los ejes se desplacen en esa cantidad respecto a su posición actual.

Estas serian las funciones misceláneas o códigos M implementados:

- **M0:** Parada controlada, utilizado para indicar fin de programa. En nuestro caso devuelve los ejes al principio, los deshabilita y cierra la comunicación con el puerto serie para terminar con la ejecución del programa.

- **M83:** Esta función establece el modo de medida a absoluto pero solo al extrusor, vendría a ser el equivalente a G90 pero para E.

- **M84:** Esta función establece el modo de medida del extrusor a incremental, esta función sería el equivalente a G91 para E.

- **M104 Snn:** Utilizada para establecer la temperatura consigna para el extrusor con el parámetro S. Esto es, la temperatura que se usará como referencia en el control de temperatura del extrusor y es a la que se ajustará.

- **M105:** Con este comando se obtienen las temperaturas que devuelven todos los sensores instalados en la maquina y se muestran por pantalla al usuario junto a las temperaturas consigna correspondientes y el estado de los calentadores que las controlan.

```
ok T:54.13  /50.00  B:56.30  /60.00  @:0  B@:1
```

Ilustración 19: Salida M105

- **M114:** Muestra por pantalla las coordenadas actuales de los ejes X, Y, Z, además de esta información se muestra la posición absoluta en pasos de motor. Este comando es más común utilizarlo en tareas de depuración para confirmar que un movimiento se ha ejecutado de forma correcta.

```
X:15.00  Y:20.00  Z:5.00  E:0.00    Cont:X:3750.00  Y:5000.00  Z:1250.00
ok
```

Ilustración 20: Salida M114

- **M119:** Muestra por pantalla el estado de los finales de carrera que haya instalados. Esto sirve a modo de depuración para saber si los finales de carrera funcionan a nivel alto o nivel bajo. También es útil para comprobar si el funcionamiento de estos es correcto o hay algún fallo.

```
Estado finales de carrera:
x_min: 0
y_min: 0
z_min: 1
ok
```

Ilustración 21: Salida M119

- **M140 Snn:** Sirve para ajustar la temperatura consigna para la base de la impresora a la temperatura indicada en el parámetro S.

Aparte de estos comandos G y M, hemos considerado adecuado el añadir un comando especial "?" que muestra por pantalla un mensaje de ayuda al usuario.

```
Firmware 3D - Version 4.01
Comandos:
G1 [X/Y/Z/E] [F] - Interpolacion lineal
G28 - Ejes a inicio
G90 - Modo absoluto (Ejes)
G91 - Modo relativo (Ejes)
M82 - Modo absoluto (Extrusor)
M83 - Modo relativo (Extrusor)
M0 - Fin de programa
M104 [S] - Establecer temperatura del extrusor
M105 - Mostrar temperaturas
M114 - Mostrar posicion
M119 - Mostrar el estado de los finales de carrera
M140 [S] - Establecer temperatura de la cama
? - Mostrar este mensaje de ayuda

Modo ejes: Absoluto
Modo extrusor: Absoluto
```

Ilustración 22: Mensaje de ayuda

3.3 Proceso de lectura e interpretación de órdenes

Una vez conocido el lenguaje que se ha utilizado y los comandos que han sido implementados, pasamos a explicar el proceso de lectura e interpretación de comandos.

La primera parte de este proceso está enfocado a la lectura de las órdenes. Esta parte se encuentra en el *loop()* y consiste en un bucle "while" que se ejecuta mientras la función de Arduino *SerialAvailable()* se cumpla, o lo que es lo mismo, se ejecuta mientras haya algo que leer en el puerto serie. Dentro de este bucle se van leyendo caracteres con la función *Serial.read()* y se almacena en el buffer previamente creado, todo esto hasta que se detecta un salto de línea "/n" o se excede el tamaño del buffer en cuyo caso se aborta la lectura. Si se ha leído hasta un salto de línea se llama a una función propia llama *ejecuta_comando()* que se correspondería con la segunda parte del proceso y finalmente cuando se ha ejecutado despeja el buffer como preparatorio para leer la siguiente orden.

```
while (Serial.available() > 0) {
  char c = Serial.read();
  if (ocupado < 64 - 1) buffer_comandos[ocupado] = c; ocupado++;
  if (c == '\n') {
    ejecuta_comando();
    ocupado = 0;
  }
}
```

Ilustración 23: Lectura de instrucciones

Esta segunda parte se sucede en la función propia *ejecuta_comando()*. La estructura principal de esta función es la de dos "switch", uno por si el comando es del tipo G, y otro por si es del tipo M.

Esta función trabaja con el buffer que contiene la instrucción previamente leída. Primero se determina qué tipo de orden es (G o M) y se lleva al "switch" correspondiente, esto se consigue con un "if" que utiliza la función *strch()* (Dada una cadena y un carácter en concreto devuelve verdadero o falso si ese carácter está presente en la cadena) como condición para averiguar si el carácter G se encuentra en el buffer, si es así se accede al "switch" de G, si no, se hace lo mismo para M. En caso de no haber nada que se ajuste a las instrucciones implementadas el programa muestra un mensaje indicando que no conoce la orden.

A partir de esta parte es conveniente hablar de una función auxiliar llamada *valor()* que recorre el buffer de comandos y hace uso de un puntero para ir recorriendo el mismo hasta que encuentra el carácter que se le pasa como argumento o llega al final. Si lo encuentra obtiene el valor que sigue a ese carácter con la función *atof()* y lo devuelve, en caso acabar de leer todo el buffer y no encontrar nada devuelve un valor de -9999 que indicaría que no se ha encontrado, hemos utilizado este número ya que no se corresponde con ningún valor valido que se pueda utilizar y vale para indicar falso.

```
float valor(char code) {
  ptr = buffer_comandos;
  while (ptr && *ptr && ptr < buffer_comandos + ocupado) {
    if (*ptr == code) {
      return atof(ptr + 1);
    }
    ptr++;
  }
  return -9999;
}
```

Ilustración 24: Función *valor*

Siguiendo con la ejecución, si ha entrado en uno de los "if" se almacena en una variable el resultado que devuelve la función antes descrita para el carácter que define el tipo de función. Esa variable es la que se utiliza en el "switch" como identificador de función, dependiendo de este valor se ejecutará una rama del "switch" que tendrá un comportamiento u otro según al comando al que se haga referencia, si no se entra por ninguna rama se muestra un mensaje al usuario diciendo que no se reconoce la orden.

```
void ejecuta_comando() {
  //Comprueba si el comando es G
  if (strchr(buffer_comandos, 'G')) {
    int gcode = valor('G');
    switch (gcode) {
      case 1: Serial.println("ok"); movimiento(valor('X'), valor('Y'), valor('Z'), valor('E'), valor('F')); break;
      case 28: ejes_a_inicio(); break;
      case 90: modo_absoluto_ej = true; Serial.println("ok"); break;
      case 91: modo_absoluto_ej = false; Serial.println("ok"); break;
      default: Serial.println("No se corresponde a ningun comando G definido"); Serial.println("ok"); break;
    }
  }
  //Comprueba si el comando es M
  if (strchr(buffer_comandos, 'M')) {
    int mcode = valor('M');
    switch (mcode) {
      case 0: parada(); break;
      case 82: modo_absoluto_ex = true; Serial.println("ok"); break;
      case 83: modo_absoluto_ex = false; Serial.println("ok"); break;
      case 104: establecer_temperatura_ex(valor('S')); Serial.println("ok"); break;
      case 105: mostrar_temperatura(); break;
      case 114: mostrar_pos(); break;
      case 119: mostrar_finales(); break;
      case 140: establecer_temperatura_bd(valor('S')); Serial.println("ok"); break;
      default: Serial.println("No se corresponde a ningun comando M definido"); Serial.println("ok"); break;
    }
  }
  if (strchr(buffer_comandos, '?')) ayuda();
}
```

Ilustración 25: Función *ejecuta_comando*

4 Movimiento

En este apartado vamos a hablar principalmente de cómo hemos implementado los comandos relativos al movimiento (G1 y G29). También explicaremos que son los motores paso a paso y como hemos implementado su control.

4.1 Movimiento de los motores paso a paso

Para empezar vamos a hablar de los elementos más importantes para poder realizar los movimientos de la máquina, los motores. Para las maquinas CNC se utilizan otro tipo de motores diferentes a los eléctricos convencionales, estos son los motores paso a paso que ofrecen una mayor precisión a la hora de realizar los movimientos, ahora veremos en qué consisten.

Los motores paso a paso o PaP son unos elementos electromecánicos que a diferencia de los motores de corriente continua permiten convertir los diferentes impulsos que reciben en movimientos angulares. Estos movimientos angulares también llamados pasos pueden variar desde pasos de 90º hasta micro pasos de 1’8º, es más común en el ámbito de nuestro proyecto utilizar estos últimos ya que nos permiten trabajar con una precisión mayor, cada motor puede tener un ángulo de paso diferente.

Estos motores están constituidos de dos partes, una fija que es el estator en el que se encuentran una serie de bobinas que harán posible los pasos y otra móvil llamada rotor que está basado en varios pares de imanes que van montados sobre el eje. El paso del motor se puede conseguir excitando las bobinas que están en el estator en una secuencia determinada, según la forma de proceder existen dos tipos de motores paso a paso, los unipolares y los bipolares.

Los unipolares suelen estar compuestos de 4 bobinas y reciben este nombre ya que la corriente que circula por las bobinas lo hace en un único sentido, físicamente se pueden reconocer por que están compuestos de 6 cables, dos para cada bobina, y otro para cada par de éstas, dependiendo de cómo se exciten podemos hablar de secuencias de paso, doble paso o medio paso.

Paso	Bobina A	Bobina B	Bobina C	Bobina D
1	+V	0	0	0
2	0	+V	0	0
3	0	0	+V	0
4	0	0	0	+V

Ilustración 26: Ejemplo secuencia de pasos de un motor unipolar

Y por otro lado están los motores unipolares, compuestos de 2 bobinas, que a diferencia de los motores unipolares la corriente que circula por las bobinas puede fluir en dos direcciones, necesitando un control bidireccional o bipolar. Estos motores son muchos más complicados de utilizar que en la secuencia de los motores unipolares hay que tener en consideración la polaridad.

Paso	Bobina A1	Bobina A2	Bobina B1	Bobina B2
1	+V	-V	+V	-V
2	+V	-V	-V	+V
3	-V	+V	-V	-V
4	-V	+V	+V	+V

Ilustración 27: Ejemplo secuencia de pasos de un motor bipolar

Todos esto es bastante complicado de controlar en un firmware, para facilitar el control están los micro controladores, en nuestro caso es el A4988. Con ellos solo tendremos que excitar el pin correspondiente al paso para realizarlo. Para ello tendremos que haber habilitado antes el motor activando el pin de habilitación, esto lo hemos realizado en *setup()* aunque se podría hacer en la primera ejecución de una orden de movimiento.

Dentro de nuestra firmware hemos implementado una función llamada *paso()* que pone a 1 y a 0 el pin de paso del motor que se le pase como argumento, así se consigue un paso, es una función muy útil para aumentar la legibilidad del código ya que sin ella tendríamos que poner a 0 y a 1 el pin de paso cada vez que se necesitara realizar un paso y esto puede complicar el código.

4.2 Movimiento lineal: Algoritmo de Bresenham

Una vez ya comprendemos mejor el funcionamiento de los motores paso a paso y como hemos implementado estos, falta saber cómo los vamos a coordinar para que se realice el movimiento que le pasemos en la orden G1.

Para ello vamos a hacer uso del algoritmo más conocido y utilizado para estos fines, el algoritmo de Bresenham. Este algoritmo fue diseñado en un principio para dibujar rectas en dispositivos gráficos de rastreo aunque se puede utilizar con buenos resultados con maquinas CNC para realizar movimientos rectilíneos. Este algoritmo genera líneas de rastreo utilizando cálculos incrementales con enteros. Este algoritmo se puede adaptar para realizar circunferencias aunque en nuestro caso solo usaremos el método lineal. Los ejes verticales muestran las posiciones de rastreo y los ejes horizontales identifican columnas de pixel. El pseudocódigo del algoritmo para dos ejes sería el siguiente:

```
Si 0<|m|<1
  *Se capturan los extremos de la línea y se almacena el extremo izquierdo en (x0,y0).
  *Se carga (x0,y0) en el bufer de estructura (se traza el primer punto)
  *Se calculan las constantes Δx,Δy, 2Δy y 2Δy-Δx y se obtiene el valor inicial para el
    parámetro de decisión p0=2Δy-Δx.
  Para j=0 mientras j<Δx
  *En cada xk a lo largo de la línea, que inicia en k=0 se efectúa la prueba siguiente:
      Si pk<0
          *Trazamos (xk+1,yk).
          *Asignamos pk+1= pk+2Δy.
      Si no
          *Trazamos (xk+1,yk+1).
          *Asignamos pk+1= pk+2Δy-2Δx.
  Fin Para
Si |m|>1
   *Recorremos la dirección en pasos unitarios y calculamos los valores sucesivos
     de x que se aproximen más a la trayectoria de la línea.
```

Ilustración 28: Pseudocódigo algoritmo de Bresenham

Para nuestro proyecto hemos adaptado este algoritmo para que trabaje con los ejes X, Y, Z y el extrusor además de adaptarlo al movimiento de los motores paso a paso. Todo esto lo hemos implementado en una función llamada *movimiento()*. El método recibe como argumentos el movimiento de cada eje (incluido extrusor) en milímetros y la velocidad a la queremos que se realice.

Este método está dividido en varias fases, en una primera fase se obtiene la distancia en pasos que va a recorrer cada eje. La distancia a recorrer depende del tipo de movimiento que tengamos por defecto o hayamos configurado, estos modos pueden ser el absoluto y el relativo. Si el modo es absoluto, lo que hacemos es calcular la diferencia entre la posición actual con la posición absoluta que se pasa como argumento, este resultado se multiplica por una variable que hemos calculado previamente y que depende de cada motor, que son los pasos necesarios para recorrer un milímetro, este variable se puede calcular fácilmente ateniéndose a las características de cada motor, lo más frecuente es que sean 200 pasos para un milímetro, con esa multiplicación

conseguimos pasar los milímetros a pasos. En el caso contrario que el modo de movimiento sea relativo, obviamos la parte de la resta y multiplicamos directamente por los pasos por milímetro. Para el extrusor el proceso es el mismo pero se hace separado por que el modo de funcionamiento es independiente del de los motores de los ejes. Con esto ya hemos obtenido las deltas en el algoritmo original.

```
//Variables
long dx, dy, dz, de;                          //Deltas
long dabsx, dabsy, dabsz, dabse, dmax;        //Deltas absolutas
long errx, erry, errz, erre;                  //Errores
//float acc, v;                               //Control de velocidad
boolean controlar = false;                    //Flag que indica si es necesario controlar temperatura

//Comprobamos los argumentos y calculamos las delta de los ejes
if (modo_absoluto_ej == true) {
  dx = x_obj > -1 ? (x_obj - posx) * PASOS_X_MM_X : 0;
  dy = y_obj > -1 ? (y_obj - posy) * PASOS_X_MM_Y : 0;
  dz = z_obj > -1 ? (z_obj - posz) * PASOS_X_MM_Z : 0;
} else {
  dx = x_obj != -9999 ? x_obj * PASOS_X_MM_X : 0;
  dy = y_obj != -9999 ? y_obj * PASOS_X_MM_Y : 0;
  dz = z_obj != -9999 ? z_obj * PASOS_X_MM_Z : 0;
}

//Obtenemos la delta del extrusor
if (modo_absoluto_ex == true) {
  de = e_obj > -1 ? (e_obj - pose) * PASOS_X_MM_E : 0;
} else {
  de = e_obj != -9999 ? e_obj * PASOS_X_MM_E : 0;
}
```

Ilustración 29: Declaración de variables y cálculo de deltas

Una vez hemos calculado las deltas toca decidir cuál de ellas es la mayor para elegir el eje dominante, para ello debemos calcular el valor absoluto de estas deltas, ya que para movimientos hacia el origen la delta será negativa y no vale para este cálculo. Una vez hemos calculado las deltas absolutas, las comparamos entre si y almacenamos la máxima que servirá en el bucle para hacer de ese eje el eje dominante.

```
//Obtenemos deltas absolutas
dabsx = abs(dx);
dabsy = abs(dy);
dabsz = abs(dz);
dabse = abs(de);

//Obtenemos la delta maxima
if(dabsx > dabsy) {dmax = dabsx; PASOS_X_MM = PASOS_X_MM_X;} else {dmax = dabsy; PASOS_X_MM = PASOS_X_MM_Y;}

if(dmax < dabsz) {dmax = dabsz; PASOS_X_MM = PASOS_X_MM_Z;}

if(dmax < dabse) {dmax = dabse; PASOS_X_MM = PASOS_X_MM_E;}
```

Ilustración 30: Obtención de la delta máxima

Ahora habría que preparar los motores, primero para que funcionen en un sentido o en el otro activando o desactivando el pin de dirección en función de la delta original. Luego para que funcionen a la velocidad que le hemos pasado.

```
//Aplicamos los sentidos
digitalWrite(DIRX, dx > 0 ? HIGH : LOW);
digitalWrite(DIRY, dy > 0 ? HIGH : LOW);
digitalWrite(DIRZ, dz > 0 ? HIGH : LOW);
digitalWrite(DIRE, de > 0 ? HIGH : LOW);

//Control de la velocidad
f_obj = f_obj > 0 && f_obj <= MAX_FEEDRATE ? f_obj : 2500 /*Por defecto*/;
```

Ilustración 31: Ajustes de dirección y velocidad

Ahora solo faltaría calcular los errores, esto es la mitad de la delta máxima. Ya entraríamos en el bucle que se ejecutará con un retraso que dependerá de la velocidad que le hayamos pasado por argumento y que hemos calculado previamente. En este bucle se van recorriendo todos los ejes hasta que el eje dominante ha cumplido todos los pasos o lo que es lo mismo. Por cada eje que se recorre se actualiza su error sumándole su delta absoluta seguido de un "if" que comprueba que el error es mayor que la delta máxima, si es así se realiza un paso y al error se le resta la delta máxima.

```
//Bucle principal
for (long i = 0; i < dmax; i++) {
  errx += dabsx;
  if (errx >= dmax) {
    errx -= dmax;
    paso('X');
  }
  erry += dabsy;
  if (erry >= dmax) {
    erry -= dmax;
    paso('Y');
  }
  errz += dabsz;
  if (errz >= dmax) {
    errz -= dmax;
    paso('Z');
  }
  erre += dabse;
  if (erre >= dmax) {
    erre -= dmax;
    paso('E');
  }

  actualmillismov = millis();
  if (actualmillismov - prevmillismov >= 500) {
    control_temperatura();
    prevmillismov = millis();
  }

  espera(60000 / (f_obj * PASOS_X_MM));
}
```

Ilustración 32: Bucle principal del algoritmo de Bresenham

Una vez salimos del bucle solo quedaría actualizar las posiciones de los ejes a las nuevas posiciones. Aunque no se hable en detalle en este apartado en este algoritmo hay incrustada una rutina de control de temperatura de la cual hablaremos en el apartado correspondiente.

4.3 Vuelta a origen de los ejes

Para hacer que los ejes vuelvan al origen hemos implementado la función *ejes_a_inicio()*. El funcionamiento de esta función es muy sencillo. Primero pone a 0 el pin de dirección de todos los motores para que cada vez que demos un paso lo demos en la dirección del origen (Si tenemos este pin invertido seria ponerlo a 1). Luego hay 3 bucles que son los encargados de devolver los ejes al cero maquina, dentro de los bucles se van realizando pasos de forma indefinida mientras no se active el final de carrera correspondiente (Puede ser un 1 o un 0 dependiendo de si está invertido o no). Estos bucles se ejecutan en el siguiente orden secuencial, X, Y, Z. Hemos elegido este orden ya que no interfiere en la pieza que estemos haciendo en caso de que haya una parada y los ejes tengan que volver a inicio mitad ejecución del programa.

```
void ejes_a_inicio() {

  //Ponemos la dirección hacia el principio
  digitalWrite(DIRX, LOW);
  digitalWrite(DIRY, LOW);
  digitalWrite(DIRZ, LOW);

  //Bucles que devuelven los motores al principio:
  //Eje Z
  while (digitalRead(FINZ) == LOW) {
    paso('Z');
  }
  posz = 0;

  //Eje Y
  while (digitalRead(FINY) == LOW) {
    paso('Y');
  }
  posy = 0;

  //Eje X
  while (digitalRead(FINX) == LOW) {
    paso('X');
  }
  posx = 0;

  mostrar_pos();
}
```

Ilustración 33: Vuelta a inicio de los ejes

5 Control de temperatura

En este apartado vamos a hablar del control de temperatura que hemos implementado en nuestro controlador, se trata de un controlador muy sencillo que lee la temperatura de los sensores, en este caso del sensor de la cama y del extrusor, procesa esa información y actúa en consecuencia.

5.1 Obtención de la temperatura

Para poder controlar la temperatura lo primero que necesitamos es saber cuál es. Para esto disponemos de dos termistores que leen la resistencia y según este valor la temperatura es una u otra dependiendo del tipo de termistor, principalmente están los NTC (A más resistencia, menos temperatura) o PTC (A más resistencia, más temperatura).

En nuestra implementación se ha tomado en consideración los transistores de tipo NTC, sin embargo como veremos más adelante si se desease utilizar transistores del tipo PTC las modificaciones serian prácticamente triviales.

Bien, entrando en cuestiones de implementación, como ya se ha visto en el segundo apartado hemos reservado los pines analógicos 9 y 10. Estos pines se leerán con la función de Arduino *analogRead()*. En Arduino esta función devuelve un valor comprendido entre 0 y 1024, este valor lo deberemos interpolar haciendo uso de la interpolación lineal para obtener su equivalencia a grados.

La interpolación lineal nos permite obtener valores independientes de una función dados dos valores extremos. Para ello se utiliza una tabla en la que se han tomado varios valores de X y de Y para utilizarlos con el polinomio de interpolación de Newton, dado un X podremos obtener la Y asociada o lo que sería en nuestro caso dado un valor del sensor podremos obtener el valor en grados asociado. Este no es un método del todo preciso, pero es sencillo de implementar y su precisión es la suficiente para nuestro proyecto a pesar de no ser del 100%.

$$f(x|x_1;x_2) = f(x_1) + \frac{f(x_2) - f(x_1)}{(x_2 - x_1)}(x - x_1)$$

Ilustración 34: Polinomio de interpolación de Newton

Para implementar todo esto primero deberemos de declarar la tabla de interpolación, nosotros hemos definido una sola tabla para los dos transistores ya que son del mismo tipo, la tabla la hemos definido como un array unidimensional con 62 valores o lo que es lo mismo 31 posibles temperaturas, aquí la declaración:

```
float tabla_temp[62] {
  23  , 300,
  27  , 290,
  31  , 280,
  35  , 270,
  41  , 260,
  48  , 250,
  56  , 240,
  66  , 230,
  78  , 220,
  92  , 210,
  109 , 200,
  131 , 190,
  156 , 180,
  187 , 170,
  224 , 160,
  268 , 150,
  320 , 140,
  379 , 130,
  445 , 120,
  516 , 110,
  591 , 100,
  665 , 90,
  737 , 80,
  801 , 70,
  857 , 60,
  903 , 50,
  939 , 40,
  966 , 30,
  985 , 20,
  999 , 10,
  1008 , 0
};
```

Ilustración 35: Declaración de la tabla de interpolación

Esta tabla viene definida por el tipo de transistor, que es NTC, pero si se quisiera hacer con uno PTC u otro NTC, simplemente habría que cambiar esta tabla por otra que se correspondan a valores de funcionamiento del transistor que tengamos instalado. El cambio de tabla no afecta para nada a la lectura ni a la interpolación.

Para realizar la lectura hemos definido una función que se llama *leer_temperatura(),* esta función lee las dos temperaturas que podemos medir. El funcionamiento es el siguiente, hay una primera parte en la que se obtiene el valor de uno de los sensores instalados y se almacena en una variable local que representaría X en la función, luego

se recorre el array en el que hemos definido la tabla de interpolación comprobando que el valor del sensor que hemos leído se encuentre entre dos valores "X" del array. Una vez se han encontrado los dos valores extremos se calcula el valor en grados utilizando la formula que hemos visto antes para realizar la interpolación, el valor de ese cálculo se almacena en una variable global destinada a esa temperatura. En el caso del otro sensor el procedimiento seria el mismo.

```
void leer_temperatura() {

  //Para obtener el valor de la temperatura utilizaremos el metodo de interpolación lineal:
  // Y = Ya + (X - Xa) * (Yb - Ya) / (Xb - Xa)

  //LEEMOS LA TEMPERATURA DEL EXTRUSOR
  float v_ex = analogRead(SENSOREX);

  for (int x = 0; x < 60; x = x + 2) {
    if (v_ex >= tabla_temp[x] && v_ex <= tabla_temp[x + 2]) {
      temp_ex = tabla_temp[x + 1] + (v_ex - tabla_temp[x]) * ((tabla_temp[x + 3] - tabla_temp[x + 1]) / (tabla_temp[x + 2] - tabla_temp[x]));
    }
  }


  //LEEMOS LA TEMPERATURA DE LA CAMA
  float v_bd = analogRead(SENSORBD);

  for (int x = 0; x < 60; x = x + 2) {
    if (v_bd >= tabla_temp[x] && v_bd <= tabla_temp[x + 2]) {
      temp_bd = tabla_temp[x + 1] + (v_bd - tabla_temp[x]) * ((tabla_temp[x + 3] - tabla_temp[x + 1]) / (tabla_temp[x + 2] - tabla_temp[x]));
    }
  }
}
```

Ilustración 36: Lectura de temperatura

Todo esto permite trabajar a la maquina con las temperaturas internamente, pero también es conveniente que el usuario sea capaz de ver a que temperatura esta la maquina, para ello se ha implementado la función *mostrar_temperatura()* que lee la temperatura y la muestra por la pantalla entre otros parámetros. Esta función es accesible mediante el comando M105.

5.2 Método de control

Una vez ya hemos leído la temperatura habría que implementar un método para que procese el valor que se ha leído previamente y compruebe si se ajusta a la temperatura deseada y en caso de no ser así, corregirla.

Para nuestro proyecto hemos elegido un método de corrección directa. Este método consiste en teniendo una temperatura que queremos conseguir y el valor de la

temperatura se aplique lo siguiente: Si la temperatura es mayor que la deseada, se reduce, si es menor, la aumentamos.

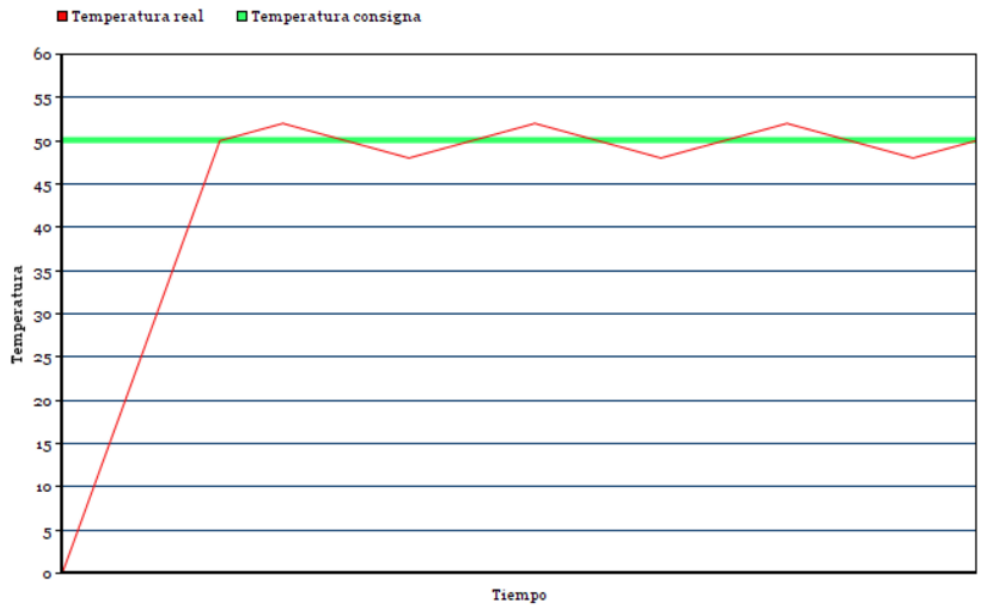

Ilustración 37: Gráfica sobre el comportamiento del control de temperatura implementado

No es el método más preciso para controlar la temperatura ya que existen mejores métodos como el PID que puede hacer una estimación de las temperaturas futuras y ajustarse mejor, obteniendo una curva de calibración más suave. Sin embargo, el método que hemos elegido, cumple perfectamente su cometido y es mucho más fácil de implementar y de entender que el PID.

Para poder implementar esto primeramente necesitamos dos variables que sirvan como temperaturas objetivo o consigna para los dos elementos que queremos controlar, se declararán como dos variables inicializadas a 0 por seguridad, aunque luego el usuario lo puede configurar para que se ajuste a sus necesidades, ya sea con los comandos M104 y M140 o modificando las mismas variables de inicio desde el código.

Nuestro algoritmo de control está dentro de una función llamada *control_temperatura()*, que lo que hará antes de todo será realizar una llamada a *leer_temperatura()* para obtener la temperatura actualizada.

Una vez ya se tienen las temperaturas actualizadas en las variables globales, se sigue una estructura sencilla para decidir la acción a tomar. Son dos"if" (Uno por elemento a controlar) en los que se comprobará que la temperatura que se ha leído antes sea mayor que la temperatura consigna, si es cierto, el pin del calentador correspondiente se pondrá a 0 para enfriar, en cambio si no es cierto, se pondrá a 1 para calentar.

```
void control_temperatura() {
  leer_temperatura();

  if (temp_ex < consigna_ex) {
    digitalWrite(CALENTADOREX, HIGH);
  } else {
    digitalWrite(CALENTADOREX, LOW);
  }

  if (temp_bd < consigna_bd) {
    digitalWrite(CALENTADORBD, HIGH);
  } else {
    digitalWrite(CALENTADORBD, LOW);
  }
}
```

Ilustración 38: Implementación del control de temperatura

5.3 Rutinas de control implementadas

Estando ya implementado todo lo necesario para permitir controlar la temperatura de la impresora es necesario utilizarlo. Para ello hemos implementado dos rutinas de control que permiten controlar la temperatura cada cierto tiempo, siendo este inferior al segundo.

La primera que hemos implementado está localizada en el bucle principal, esta rutina se dispara cada 500ms si no se ha realizado ninguna orden, está pensado para que la temperatura siempre este controlada aunque no estemos utilizando la maquina. Esta implementada como un "if" en el bucle principal que se comprobará cada 500ms, este "if" esta antes que el bucle de lectura de órdenes, ya que el control de la temperatura es una tarea más prioritaria. Con dos variables que controlan el tiempo, una el tiempo actual y otro el tiempo del último control. En ese "if" se restan el tiempo actual y el tiempo del último control y si es superior a 500ms realizara una llamada a *control_temperatura()* y actualizará las variables del tiempo.

```
actualmillis = millis();
if (actualmillis - prevmillis >= 500) {
  control_temperatura();
  prevmillis = millis();
}
```

Ilustración 39:Rutina de control de temperatura

La otra rutina está en la función G1 para el movimiento lineal, al final del bucle del algoritmo de Bresenham hay un if con el mismo funcionamiento que el que hemos visto antes aunque ahora al ser un movimiento hemos decidido que sería mejor hacerlo cada menos tiempo, en este caso 250ms. Con esto podremos tener controlada la temperatura

durante el movimiento, que para movimientos cortos es de poca transcendencia ya que no hay riesgo de sobrecalentamiento, no obstante en movimientos largos es muy importante tener controlada la temperatura para evitar daños.

6 Conclusiones

Una vez ya hemos finalizado la implementación de todo el código del firmware, hemos pasado a la realización de las pruebas. No todas estas pruebas han sido un éxito a la primera pero han servido para mejorar la idea inicial, para rectificar y comprender mejor el código que se estaba desarrollando. Al final de las pruebas se han obtenido unos resultados de funcionamiento dentro de lo esperado.

Lo que esperábamos era realizar utilizando la menor complejidad posible un controlador que fuera capaz de controlar el movimiento de varios ejes además de controlar la temperatura de la máquina. Todo esto lo hemos conseguido sin olvidar la sencillez a la hora de escribir el código, resultando un código de apenas 700 líneas esforzándonos además en la legibilidad de esas líneas, consiguiendo que el código sea fácilmente comprensible para aquella gente que está empezando en este mundillo, con varios comentarios que ayudan a comprender el programa.

En general hemos conseguido plasmar en este proyecto las ideas principales que teníamos en mente de cómo debería ser. Más allá de los objetivos técnicos este proyecto ha servido para ampliar los conocimientos que ya se tuviesen sobre las maquinas CNC que servirán como base para futuros proyectos más complejos, entre los que se podría incluir un proyecto de mejora para dotar este mismo firmware de mayor funcionalidad.

6.1 Posibles ampliaciones

El software que hemos desarrollado sirve como punto de partida para proyectos de mayor envergadura, ya que ofrece una gran cantidad de posibilidades de ampliación y mejora entra las que se encontrarían:

- Visualización del estado de la maquina y de la salida de las órdenes por un LCD.
- Control de temperatura por PID.
- Permitir realizar movimientos curvos.
- Control de aceleración.

- Una cola para ir almacenando las ordenes y aumentar la eficiencia en la interpretación de comandos.

- Ampliar el conjunto de comandos G y M que pueden ser ejecutados.

- Adaptar el firmware para que pueda trabajar no solo con impresoras 3D, sino que también lo pueda hacer con grabadoras láser o fresadoras.

- Carga de programas en G-Code desde tarjetas Micro-SD.

- Y mucho más…

7 Bibliografía

1. 3D Printing Industry. «The Free Beginner's Guide», http://3dprintingindustry.com/3d-printing-basics-free-beginners-guide/.

2. Anónimo. «Bresenham's line algorithm», https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm.

3. Arduino ORG. «What is Arduino», https://www.arduino.cc/en/Guide/Introduction.

4. Banzi, Massimo, y Shiloh, Michael. *Introducción a Arduino : edición 2016.* Madrid: Anaya Multimedia, 2015.

5. Geeetech. «Ramps1.4 - Geeetech Wiki», http://www.geeetech.com/wiki/index.php/Ramps1.4.

6. Grbl CNC Project. «GRBL wiki», https://github.com/grbl/grbl/wiki.

7. Lipson, Hod, y Kurman, Melba. *La revolución de la impresión 3D.* Madrid: Anaya Multimedia, 2014.

8. Marlinfw ORG. «Introduction | Marlin 3D Printer Firmware», http://marlinfw.org/docs/basics/introduction.html.

9. Palezzi, Ariel. «Motores paso a paso», http://www.neoteo.com/motores-paso-a-paso/.

10. Plaza, José Ángel. «Así funciona una impresora 3D», http://tlife.guru/profesional/asi-funciona-una-impresora-3d/.

11. RepRap. «G-Code», http://reprap.org/wiki/G-code.

12. Ventura, Victor. «¿Qué es G-Code?», https://polaridad.es/que-es-g-code/.