UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Sistema de visión para equipo pick-and-place

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Pardo Alcalá, Álvaro

Tutor: Sánchez López Miguel

Curso 2014-2015

Resumen

La problemática que se encuentran los diseñadores o aficionados a la electrónica cuando sueldan o fabrican una placa de circuito impreso es que con la vista y el pulso normal de una persona no es fácil poder manipular o comprobar las PCB. La idea principal del TFG que quiero realizar es la implementación de un sistema de visualización para facilitar la soldadura y la calidad de una placa de circuito impreso. Para realizar esta tarea el sistema esta basado en un CNC de tres ejes que permitirá el libre movimiento para situar la cámara en el lugar que se desee para poder visualizarlo en el PC, así se podrá comprobar por ejemplo si una soldadura esta bien realizada o directamente hacer la soldadura, con mas facilidad ya que tiene una imagen mucho mas nítida o incluso aumentada si se utiliza un microscopio en lugar de una simple cámara. Esta herramienta no es solo útil para esta tarea sino que puede ser útil para muchos otros trabajos de precisión.

Palabras clave: PCB, pick-and-place y CNC

Abstract

The problem of electronic designers when manufacture a printed circuit board is that with vision and normal pulse of a person is not easy to manipulate or check the PCB. The main idea of the project I want to do is to implement a system visualization to facilitate the welding of a PCB.

To perform this task, the system is based on three axes CNC allow free movement to place the camera in the desired location for you can display on your PC. Then can check for example if a weld, It is correct or directly solder a component. This tool is not only useful for this task but can be used for many other precision work.

Palabras clave: PCB, pick-and-place y CNC

Índice

Índice de figuras

Índice de cuadros

1. Introducción

El objetivo de este proyecto es crear un sistema de visualización, controlado por computador, para facilitar diferentes trabajos de precisión. La motivación inicial se baso en la problemática que tiene los diseñadores y aficionados a la electrónica, a la hora de soldar y comprobar los componentes de los circuitos impresos.

Para cumplir este objetivo me he basado en una CNC (control numérico por computador) de tres ejes. He creado los 3 elementos principales para conseguir un proyecto lo más completo posible, que englobe conocimientos de todo tipo. Los tres elementos que forman el proyecto serian hardware, electrónica y firmware para el microcontrolador elegido.

Los objetivos específicos de este proyecto:

- Desarrollo y construcción del sistema mecánico.
- Elección de componentes electrónicos tanto de control como de visualización.
- Diseño y fabricación del circuito de los elementos electrónicos.
- Desarrollo del firmware de control:
    - Capaz de gestionar comandos G-codes.
    - Control de varios motores paso a paso con un único microcontrolador.
    - Motores con perfil de aceleración para un funcionamiento más adecuado.
    - Control de un servo para el movimiento de la herramienta.

La electrónica esta formada por diferentes elementos. El principal seria el microcontrolador que es el encargado de la gestión del resto de componentes electrónicos, como motores paso a paso, servos y sensores de final de carrera. El microcontrolador elegido es un Arduino uno revisión 3, esta plataforma se basa en un ATmega328, del fabricante Atmel, en un apartado de este proyecto explicare cuales son las motivaciones de esta elección y del resto de la electrónica que compone el sistema, aparte describiré detalles técnicos y funcionamiento teórico. A parte del microprocesador, actuadores y sensores, también tenemos un microscopio usb capaz de transmitir las imágenes a un PC.

El firmware esta desarrollado por mi y es el encargado de que todo funcione, se graba en el Arduino y se comunica con el ordenar. Un firmware es el software más cercano al hardware, es el encargado de interactuar con la electrónica. En este proyecto desarrollare un interprete de G-codes. Los G-codes son los comandos se envían desde el ordenador, el microcontrolador los recoge los gestiona para saber que tiene que hacer y se encarga de actuar sobre los motores o de leer los sensores. El movimiento de los motores paso a paso es uno de los objetivos más relevantes del proyecto ya que es necesario acelerarlos y frenarlos para su correcto funcionamiento, esto estará explicado con mucho más detalle en un apartado de la memoria. Este tipo de firmware no es una idea original es similar a lo que utilizan las maquinas CNC o las populares impresoras 3d. Al igual que los comandos que se utilizan para comunicarse con las CNC, llamados G-codes, es un estándar que se desarrollo en

1960 tiene muchas ventajas porque a partir de diseños CAD/CAM puedes generar estos comandos de una forma especifica para tu maquina.

El hardware son los elementos necesarios para la construcción de la CNC, muchas de las piezas que forman esta maquina están impresas en 3d y basadas en una Prusa i3, que es una impresora 3d de código abierto y totalmente libre desarrollada por Josef Prusa. A parte de las piezas impresas en 3d consta de otros componentes, como pueden ser varillas lisas, varillas roscadas, poleas dentadas, rodamientos y otras partes que explicare con más detalle.

A pesar que este tipo de proyecto se podían desarrollar diversos software, me decidí por el desarrollo del firmware del microcontrolador porque es el elemento que más cercano al hardware y eso me ha permitido tener un relación directa con la electrónica y el funcionamiento de los diferentes componentes mecánicos del sistema, permitiéndome aprender cosas que no se han enseñado en la carrera y aplicar muchas otras que si que se han impartido. También ha influido la elección de este rumbo de desarrollo que mi especialización sea ingeniería de computadores, por lo tanto es el tema que más me interesa y me llama la atención.

2. Descripción de la Electrónica

La electrónica en este tipo de proyectos es una parte imprescindible ya que es la encargada de interactuar con el mundo real. Es importante tener los conocimientos sobre la electrónica previamente al desarrollo del firmware ya que están totalmente relacionados. A continuación voy a incluir un diagrama de bloques de como serian las comunicaciones de las diferentes partes del sistema.

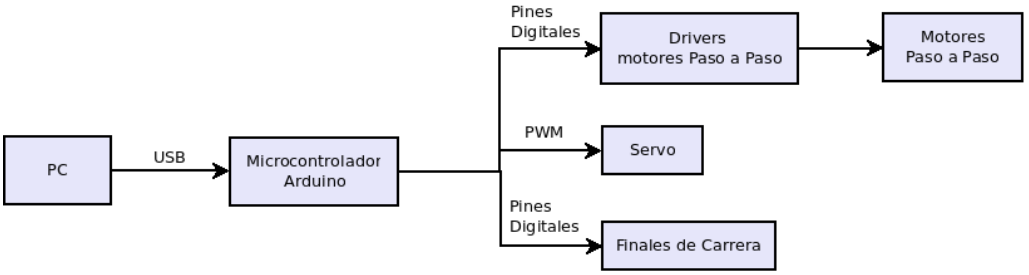

Figura 1: Comunicaciones

En este diagrama podemos observar varias cosas, una de ellas es como va fluir la información desde el PC, pasando por el microcontrolador hasta los actuadores y sensores. También podemos ver que tipo de conexiones se van a utilizar, como por ejemplo tipo de protocolos o tipo de conexión eléctrica.

2.1. Arduino

He dedicado un apartado solo al microcontrolador ya que es el parte más importante de la electrónica el encargado de ejecutar el firmware.

2.1.1. Definición

Un microcontrolador es un circuito integrado programable, capaz de ejecutar instrucciones grabadas en su memoria interna. Esta compuesto normalmente por tres unidades, todas ellas en el mismo integrado, que son: unidad central de procesamiento, memoria y periféricos de entrada salida.

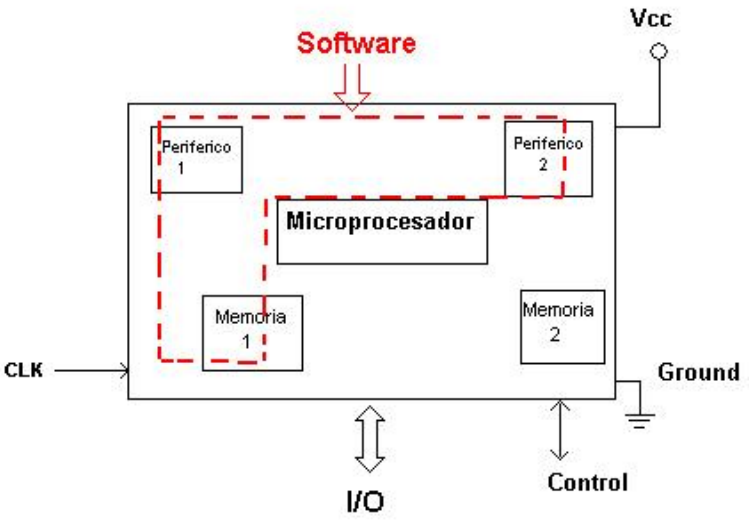

Figura 2: Microcontrolador

2.1.2. ¿Por qué Arduino?

Antes de explicar técnicamente las características del microcontrolador elegido, quiero justificar las motivaciones que me llevaron a elegir el Arduino Uno.

Arduino es una plataforma de hardware libre que facilita el uso de la electrónica gracias a las librerías que también son libres. Estas librerías te permiten acceder de forma sencilla a los pines de entrada salida del microcontrolador Atmega328. Otra de las grandes ventajas de Arduino es que dispone de un bootloader que te permite introducir tus propios firmwares a través de la interfaz de USB sin necesidad de utilizar ningún programador, esto es gracias a que esta placa dispone de un Atmega16U2 que esta programado como un convertidor de USB a serie. Se puede conseguir desde un precio bastante reducido y tiene una comunidad alrededor que te proporciona mucha información y te facilita el desarrollo. Estas son las principales razones por las que he elegido Arduino como plataforma de desarrollo y no otro tipo de microcontrolador, aparte de que cumplía las exigencias técnicas que son necesarias

para este tipo de proyectos, cosas como pines digitales y analógicos, comunicación serie, PWM, alimentación a través de la placa, interrupciones, tres timers etc.

2.1.3. Características técnicas Arduino Uno R3

Característica	**Descripción**
Microcontroladores	ATmega328
Tensión De Funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (límites)	6-20V
Pines Digitales I/O	14 (6 PWM)
Pines Analógicos de entrada	6
Corriente de DC por pines I/O	40 mA
Corriente de DC del pin 3.3V	50 mA
Memoria Flash	32 KB (ATmega328) de los cuales 0,5 KB utilizado por el bootloader
SDRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad De Reloj	16 MHz
Longitud	68,6 mm
Anchura	53,4 mm
Peso	25 g

Cuadro 1: Caractirísticas Arduino

2.2. Componentes

2.2.1. Motores Paso a Paso

Los motores paso a paso son motores de corriente continua especiales porque tiene un desplazamiento angular discreto. Estos motores se mueven mediantes los llamados pasos, que son el mínimo movimiento que puede hacer un motor de este tipo, estos pasos se generan mediante impulsos que se pueden variar dependiendo de sus entradas de control. La característica de los pasos nos permite saber en que posición, dirección y velocidad va el motor sin necesidad de tener una retroalimentación en el sistema, por ejemplo mediante un encoder, esto abarata costes ya que los motores de continua con encoders de calidad son costosos. Los pasos pueden variar desde 90 grados hasta valores menores de 0.01 grados y esto es lo que determina la precisión del motor.

Funcionamiento motores paso a paso

Este tipo de motores están formados por dos partes principales el estátor y el rotor, el estátor es la parte fija y el rotor la parte móvil. Normalmente el estátor esta formado por dos bobinas que generan un campo magnético y el rotor tiene otra bobina que forma otro campo magnético. El rotor se dirige a la posición más estable del campo magnético que se esta generando por el estátor.

La corriente que se aplica en el estátor vendrá gobernada por un electrónica (driver stepper), que sera la encargada dependiendo de la frecuencia y la polaridad con que se alimenten las bobinas del estátor de variar la velocidad y sentido del motor.

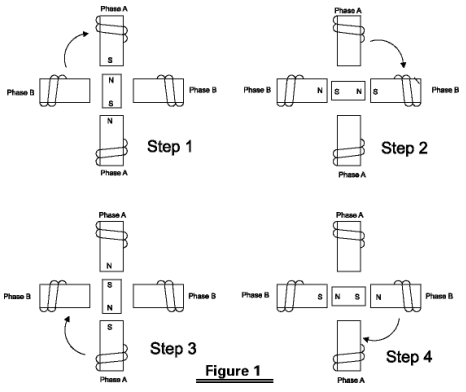

Figura 3: Motor Paso a Paso

Tipos de motores paso a paso según excitación del estátor

1. Motores unipolares:

   Su nombre indica que la corriente del estátor siempre tiene el mismo sentido a diferencia del motor bipolar. El estátor esta compuesto por dos bobinas pero uno de los terminales es común permitiendo un funcionamiento como si tuviera 4 bobinas. El numero de cables puede variar de 5 a 6 dependiendo del tipo de conexiones.

2. Motores bipolares:

   Al no tener terminal intermedio este tipo de motores tiene un control más complicado ya que la corriente que pasa por las bobinas tiene que tener sentido positivo y negativo. Este tipo de motores suelen utilizar una electrónica denominada puente en H que permite circular corriente en ambos sentidos. La ventaja respecto a los motores unipolares es que disponen de más par.

3. Motor universal:

   Este tipo de motor puede funcionar de las dos formas tanto unipolar o bipolar dependiendo de como se conecten sus terminales.

Modos de funcionamiento motores paso a paso

Según el tipo de excitación que se haga a la bobinas y el tipo de electrónica utilizada para controlar los motores paso a paso se puede aumentar el numero de pasos y el par del motor. Los modos de funcionamientos son:

1. Full-Step:

   Este modo de funcionamiento hace que el motor avance un paso por cada pulso aplicado sobre las bobinas del estátor. Tiene dos posibles variantes, una que únicamente se alimenta una bobina y el rotor se alinea enfrente de esa bobina. El otro modo que alimentan dos bobinas adyacentes y el rotor se alinea entre ese dos bobinas, esto permite aumentar el par del motor.

2. Half Step:

   Este modo de funcionamiento permite multiplicar por dos el número de pasos de un motor de este tipo. Esto se consigue por una secuencia de excitación que primero excita dos bobinas adyacentes de estátor y después en el siguiente

impulso solo se alimenta una de esas dos bobinas, esto permite que el motor avance medio paso por cada impulso.

3. Microstepping:

   Este método es una evolución del método anterior pero en lugar de alimentar las bobinas del estátor de un forma todo o nada, lo que se hace es aumentar las bobinas adyacentes con corrientes diferente así podemos conseguir que el rotor este en una posición diferente entre las bobinas, estará más cerca de la que tenga más corriente.

Motor paso a paso utilizado

El motor utilizado es un motor paso a paso bipolar 42BYGHW811 con el formato NEMA-17 que indica la dimensiones del mismo. Este modelo de motores es el habitual en la impresoras 3d y en la maquinas CNC (control numérico por computador) de dimensiones reducidas. Las características principales de este motor son las siguientes:

Característica	**Descripción**
Nº de cables	4
Ángulo de paso	1.8 grados
Torque	4800g-cm
Nº de pasos por revolución	200
Voltaje nominal	3.1V
Resistencia de la fase	1.25 ohm
Intensidad	2.5 A
Fase Inductiva	1.8 mH
Par de retención	4.8kg por cm
Dimensiones eje	diámetro 5mm x 22mm
Dimensiones	42 x 42 x 48 mm
Peso	340 g

Cuadro 2: Característicar motor paso a paso

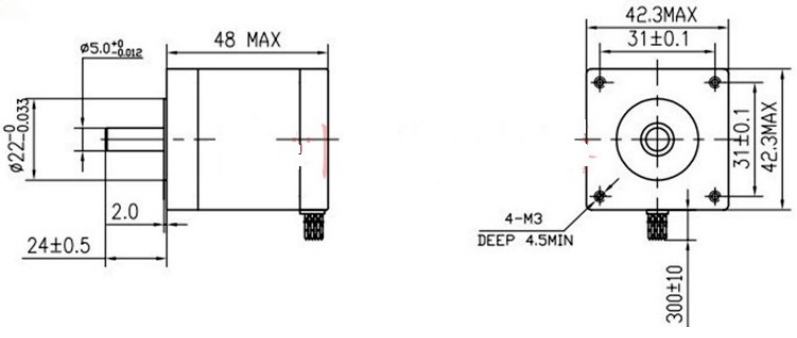

Figura 4: Motor paso a paso

2.2.2. Driver Stepper

El driver para los motores paso a paso es la electrónica necesaria para alimentar correctamente las bobinas del estátor. Este tipo de electrónica puede estar formada por diferentes componentes pero todos ellos con la misma finalidad controlar los impulsos, alimentar de una forma adecuada las bobinas y determinar el sentido. Los elementos principales que pueden formarla son:

1. Chip controlador de pasos.
2. Microcontrolador con uno o dos puentes en H.

Funcionamiento Driver Stepper

Como los microcontroladores que se utilizan para estos sistema no tiene la capacidad para alimentar estos motores paso a paso se añaden los drivers. Esta electrónica aísla la alimentación del microcontrolador principal de la alimentación de los motores, pudiendo darle a los motores un voltaje y intensidad mayor. Los drivers stepper también facilita los micropasos, incluso en diferentes rangos, ya que puede alimentar las bobinas del estátor con diferentes intensidades para que los campos magnéticos sean mayores o menores. Esto también hace que los motores sean mucho más suaves y a determinadas rpm que no vibren.

Normalmente estos drivers esta formado por un pequeño microcontrolador, que es el encargado de dar el impulso en el momento correcto y un punte en H que es el encargado de alimentar las bobinas, esto permite que las velocidades sean mayores que si solo utilizáramos un puente en H, como es habitual en los motores de continua que no son paso a paso. Estos chips tiene 3 pines importantes que son, el de STEP, DIR y GND como referencia. Estos pines se conectan el microcontrolador principal, con el pin de STEP le indicas el momento en el que quieres aplicar el impulso. Con DIR indicar las dirección de giro. Luego podemos encontrar los pines de alimentación del propio chip, los de alimentación para los motores y los pines para las bobinas del motor paso a paso.

Para explicar con un símil porque es tan importante dar el impulso en el momento correcto, es como cuando empujamos un columpio, cuando lo hacemos en el instante correcto la fuerza aplicada se manifestara de la forma más efectiva posible.

Driver stepper utilizado

El driver utilizado para estos motores es el Pololu con el chip A4988 de Alegro, es de 1A pero es capaz de administrar hasta 2A de corriente por bobina, para ello tiene que estar bien disipado el calor generado por el chip, yo utilizo unos pequeños radiadores.

Característica	Descripción
Voltaje mínimo de trabajo	8V
Voltaje máximo de trabajo	35V
Corriente por fase	1A
Maxima corriente por fase	2A
Voltaje lógico mínimo	3V
Voltaje lógico máximo	5.5V
Resolución de microstep	full, 1/2, 1/4, 1/8, y 1/16
Tamaño	0.6"x0.8"
Peso	1.3g

Cuadro 3: Caraterísticas driver stepper

Modo de conexión

En la siguiente figura podemos ver como se debe conectar este tipo de driver para su correcto funcionamiento.

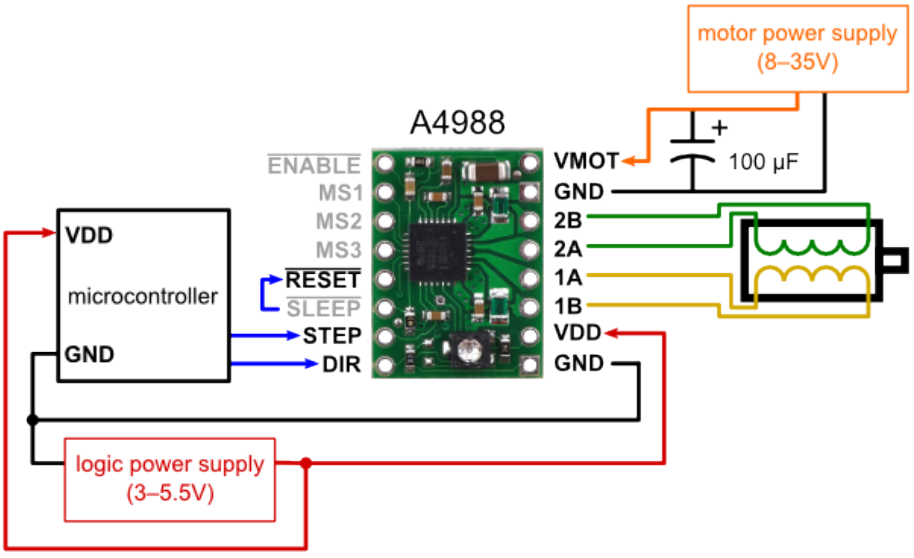

Figura 5: Pololu A4988

Alimentación

El controlador requiere una tensión de alimentación lógica entre 3V y 5,5 V que se conecta a los pines VDD y GND y una tensión de alimentación del motor de 8V a 35V para ser conectada a través de VMOT y GND. Estos suministros necesitan un condensadores de desacoplamiento cerca de la placa y que deben ser capaces de entregar la corrientes esperada.

Cuando en un integrado conectado mediante un conductor pide un pico de corriente, esta ve una caída en sus bornas. Para evitar esto se ponen condensadores de desacoplo, que actúan como un almacén de energía y estabilizan la tensión de

alimentación. En este proyecto yo he utilizado condensadores de desacoplo de 0.1 microF para la alimentación lógica y 100 microF para la alimentación de los motores.

Tamaño del paso y micropaso

Como hemos explicado anteriormente en el apartado de los motores paso a paso, se pueden utilizar los micropasos para aumentar la resolución. Esta placa dispone de una serie de pines que alimentados a 5V o 0V permiten configurar diferentes resoluciones, como se puede ver en la siguiente tabla.

MS1	**MS2**	**MS3**	**Resolución de micropaso**
Bajo	Bajo	Bajo	Paso completo (Full step)
Alto	Bajo	Bajo	Medio paso (Half step)
Bajo	Alto	Bajo	Un cuarto de paso (Quarter step)
Alto	Alto	Bajo	Un octavo de paso (Eighth step)
Alto	Alto	Alto	Un dieciseisavo de paso (Sexteenth step)

Cuadro 4: Configuración pasos

Para ello en mi diseño he incluido unos jumper que permitan fácilmente varias entre una configuración u otra.

Limitación de corriente

El A4988 soporta limitación activa de corriente, permitiendo mediante un potenciómetro ajustar el limite de corriente. Hay que tener en cuenta que el cambio de tensión lógica, a valores diferentes, cambiara el ajuste de limite de corriente porque el voltaje de referencia esta en función de la tensión lógica.

2.2.3. Servo

Este tipo de motor es capaz de situarse en cualquier posición dentro de un rango y mantenerse en esas posición. Esta compuesto por un motor de corriente continua una caja reductora y un circuito de control que normalmente es un potenciómetro y un amplificador de control, el margen de trabajo de estos motores siempre es inferior a 360 grados.

Funcionamiento de un servo

Para el funcionamiento de un servo motor lo que se hace es comparar mediante un amplificador de error, una señal de control que es la que indica la posición en la que quieres estar. Esta señal es un señal cuadra, el ancho de pulso de la señal es lo que indica el ángulo deseado, para variar este ancho de pulso se utiliza un señal PWM desde el microcontrolador. La otra señal que se comparara sera la señal del potenciómetro que indicara la posición actual de motor. Se calculara el error con el amplificador de error y se corregirá hasta que coincidan las dos señales.

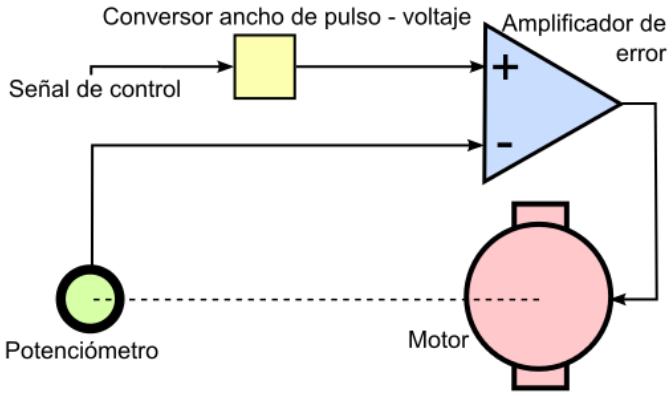

Figura 6: Control servo

Como variar la señal cuadrada de PWM que indica la posición, normalmente los servos esperan un pulso cada 2.5 ms. La duración de este pulso sirve para determinar el ángulo en que debe girar el servo. Por norma general, un pulso de 1.5ms equivale a 90º, 0.5 ms son 0º y 2.5ms son 180º.

2.2.4. Finales de carrera

Un final de carrera es un elemento electrónico, neumático o mecánicos que se sitúan al final de un recorrido para indicar un limite, puede ser por varias razones como por ejemplo por seguridad o como en este caso para indicar la posición 0, lo que se suele denominar homing, que es desde donde parte la maquina como punto de referencia inicial y previo al resto de movimientos.

Funcionamiento

Se suelen comportar como interruptores normalmente abiertos o normalmente cerrados, dependiendo de su estado original. Un final de carrera normalmente abierto tendrá un nivel bajo en la entrada del microcontrolador normalmente a no ser que se cierre y un final de carrera normalmente cerrado sera a la inversa.

Ventajas e Inconvenientes

Las ventajas más destacadas, son facilidad de instalación, coste reducido, robustez, inmune a electricidad estática y es insensible a estados transitorios.

Los inconvenientes, posibilidad de rebotes en el contacto si la velocidad es elevada y velocidad de detección.

2.2.5. Microscopio USB

Como herramienta en este proyecto he utilizado un microscopio USB que era la idea original de este proyecto, pero en realidad se podrían utilizar muchas otras herramientas desde fresas hasta rotuladores para utilizarlo como plotter. El microscopio utilizado es de una marca china, XCSOURCE, tiene las siguientes características:

Característica	**Descripción**
Sensor	CMOS
Regulador	DSP de alta velocidad
Lente	Micro-Scope lente
Distancia de enfoque	enfoque manual desde 0 mm a 200 mm
Snap Shot	Software y Hardware
Captura de vídeo Resolución	1600x1200 (2M píxeles) 1280x960 (1.3M píxeles), 800x600, 640x480
Resolución de captura de la imagen	1600x1200 (2M píxeles) 1280x960 (1.3M píxeles), 800x600, 640x480
Velocidad de cuadros	Max 30f / s Bajo brillo de 600 lux
Formato de vídeo	AVI
Snap Shot Formato	JEPG
Fuente de luz	8 LED de luz blanca (con el regulador en el cable USB)
Cociente de la ampliación	20X 800X (Manual)
Fuente de alimentación	Puerto USB (5V DC)
Software	AMCAP (Driver)
Funcionamiento del sistema	Windows 7 / Vista / XP

Cuadro 5: Características Micoscopio USB

Posibles aplicaciones

Un microscopio USB tiene aplicaciones en muchos sectores tales como, medicina, electrónica, forense, automoción etc. Gracias al incluirlo en una CNC (control numérico por computador) se podría automatizar el proceso, de diferentes formas, por ejemplo marcando varios sitios donde queremos revisar con mayor resolución un objeto, desde una placa de circuito impreso hasta un trozo de metal recién pintado que queremos saber si tiene imperfecciones. Esas comprobaciones la podría hacer un humano, visualmente y juzgar el mismo si es correcto o hay que rectificar, pero también se podría automatizar porque el microscopio USB tiene la ventaja de que puede capturar vídeo y imagen. Gracias a eso se podría comparar las imágenes mediante software con una referencia correcta por ejemplo y así encontrar posibles errores, o si queremos comprobar un color saber si el color utilizado en todo la pieza es igual o hay imperfecciones de tonalidad después de un proceso de pintado.

2.2.6. Fuente de alimentación

La fuente de alimentación utilizada es una fuente de PC ATX que se ha modificado para este proyecto. Solo se utiliza para alimentar los motores con 12V. Estas fuentes de alimentación tiene el siguiente cableado.

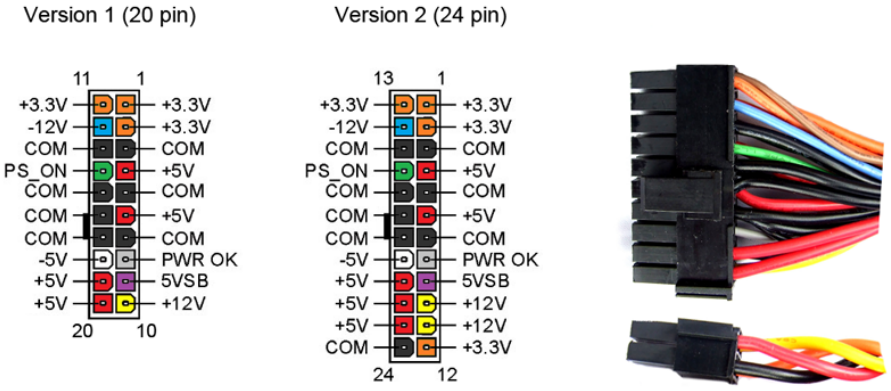

Figura 7: Fuente de alimentación

Como necesitamos 12V seleccionaremos el cable amarillo y negro que es la masa y las soldamos a un conector para que sea más sencillo conectarlo a la PCB que he diseñado. Para que este tipo de fuentes se encienda hay que hacer un puente entre el cable verde (PS_ON) y GND (negro), en lugar de hacer un puente he puesto un interruptor que me permite apagar la fuente de alimentación rápidamente. Este tipo de sistemas se utilizan en la industria en caso de emergencia y se denominan setas de emergencia, conceptualmente es similar a lo que yo he hecho con el interruptor.

2.3. Diseño de PCB

El diseño de la electrónica lo he desarrollado en un programa que se denomina Kicad, en el diseño se pueden ver todos los elementos explicados anteriormente y como se conectan entre ellos, al igual que elementos de la electrónica añadidos para su correcto funcionamiento, como puede ser jumpers para la configuración de la escala de micropasos o los condensadores de desacoplo.

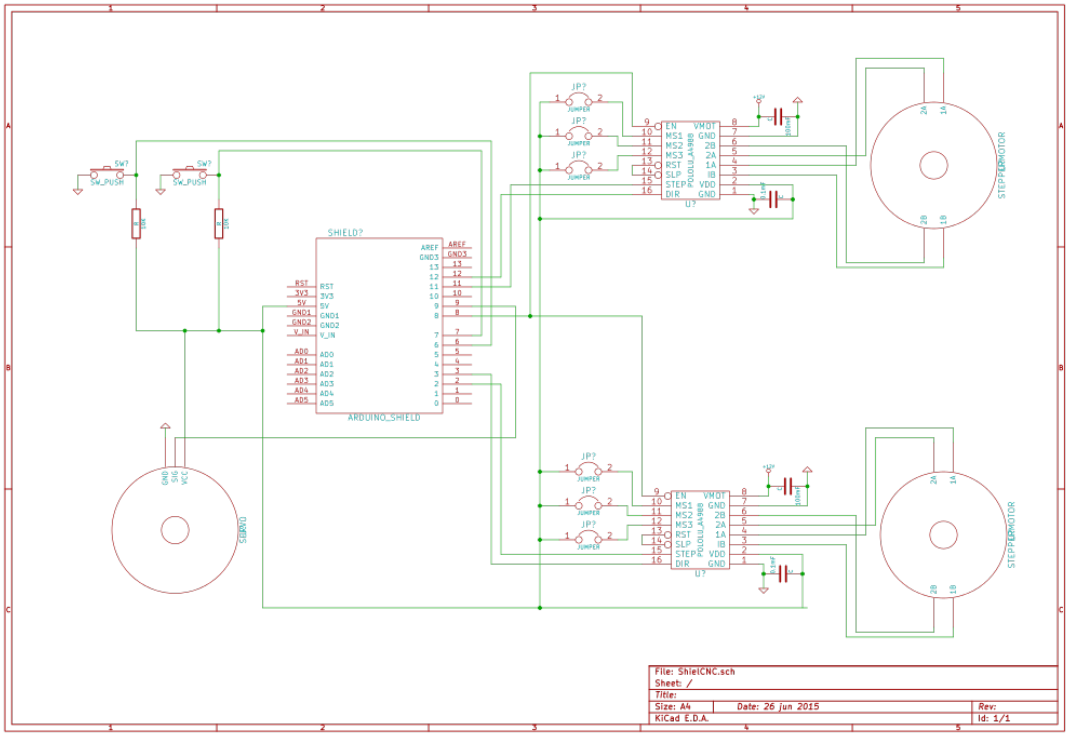

Figura 8: Esquema Electrónica

2.3.1. Shield

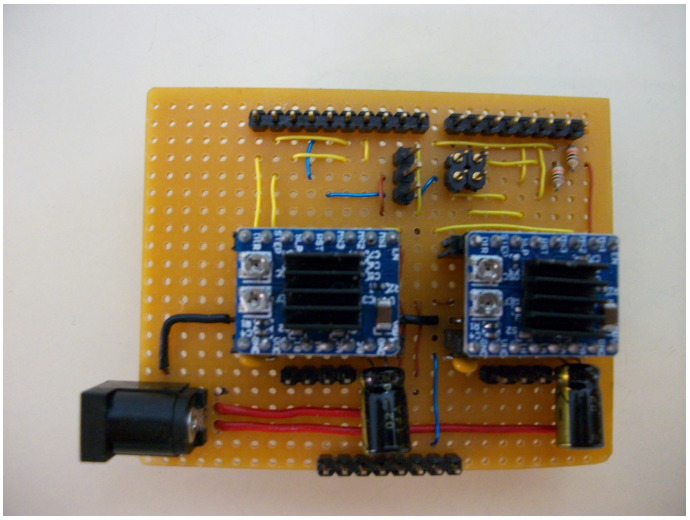

Figura 9: Shield 1

Para la construcción y conexión de los diferentes componentes electrónicos que componen el sistema, he diseñado lo que en el mundo de la electrónica y sobretodo en el mundo de Arduino se conoce como una shield. Un shield que se traduce como escudo, es un placa de circuito impreso que se conecta en la parte superior de una placa Arduino, comunicando el Arduino con los diferentes elementos.

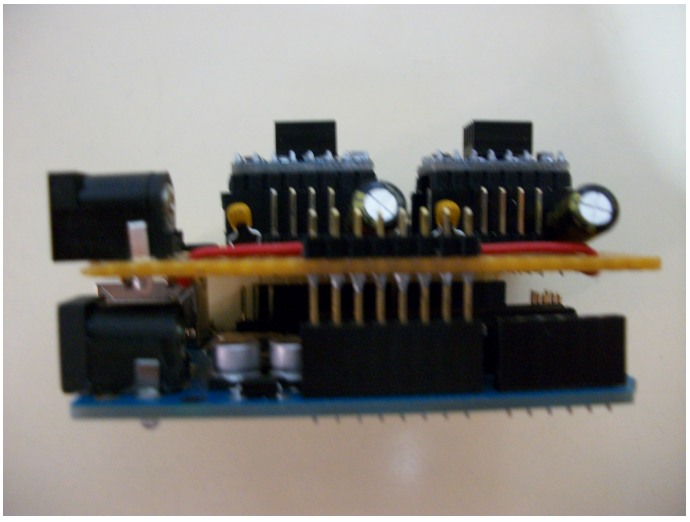

Figura 10: Shield 2

3. Descripción del Firmware

Como ya hemos indicado en la introducción, el firmware que voy a desarrollar es un interprete de G-codes, que es el encargado de recibir estos comandos a través del puerto serie, procesar y identificar, cuando ya sabemos que comando es, tenemos que actuar en consecuencia, por ejemplo moviendo los motores con un movimiento lineal o curvo.

3.1. Herramienta de desarrollo y lenguaje

Como el microcontrolador utilizado es Arduino, su empresa desarrollo un IDE apropiado para esa plataforma, bajo mi punto de vista este IDE no me resulta cómodo de utilizar, ya que el editor de código no tiene todas las funciones que estamos acostumbrados como desarrolladores. Por lo tanto busque una alternativa en la cual me sintiera más cómodo desarrollando.

La alternativa que encontré fue UECIDE (Universal Embedded Computing IDE). Lo primero que llama la atención es la pantalla principal, ya que no se limita a ser un mero editor, sinos que te proporciona información adicional como se puede ver en la siguiente imagen.

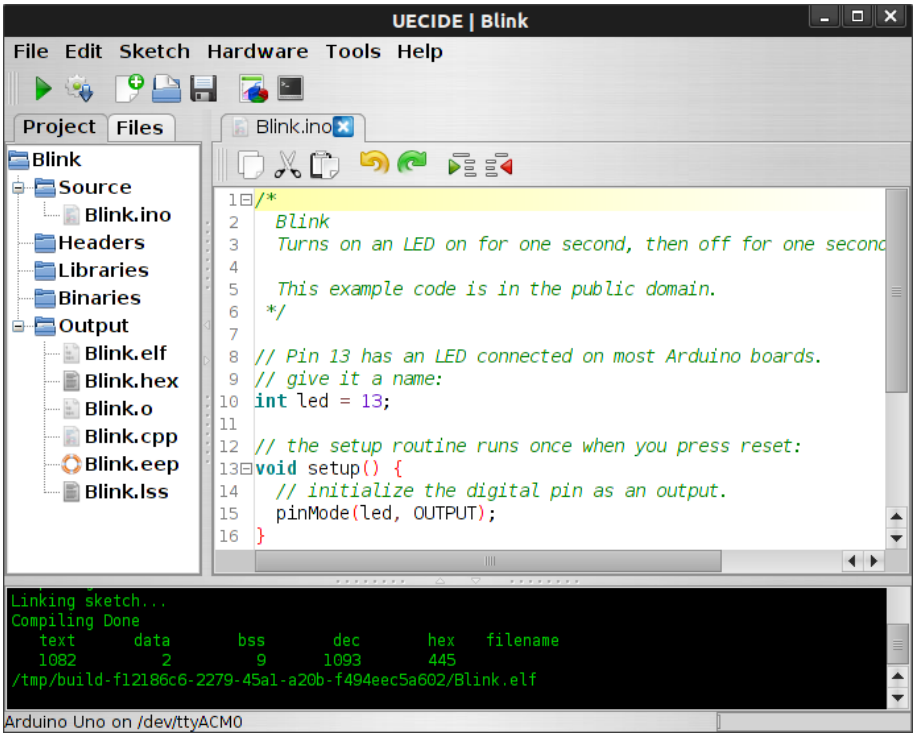

Figura 11: Ventana UECIDE

Tenemos un árbol con los diferentes componentes que pueden formar una aplicación, gracias a eso podemos navegar sin problemas entre los diferentes Sketch de Arduino. También podemos acceder a la bibliotecas que hemos incluido en nuestro proyecto y así consultar los diferentes métodos. Por ultimo tenemos una carpeta

donde podemos ver los diferentes archivos de salida de los que disponemos y que se generan después de la compilación.

Arduino se puede programar en diferentes lenguajes, pero la propia plataforma tiene un lenguaje propio que esta basado en processing y es muy similar a C. Este lenguaje se basa principalmente en dos estructuras el setup y el loop. El setup es la encargada de inicializar lo que se va utilizar en el firmware, se tiene que iniciar desde los pines y en que modo se van a utilizar, entrada o salida, hasta las comunicaciones y por ejemplo la velocidad en Baudios que se va a utilizar. La estructura setup solo se va a ejecutar una única vez al principio. Por otra parte la estructura loop hace lo que su nombre indica ejecutarse de forma consecutiva, este tipo de funcionamiento es el habitual en microcontroladores y ejecuciones en tiempo real, porque permite comprobar dependiendo de los valores obtenidos de las entradas y actuar sobre las salidas.

3.2. G-codes admitidos por el firmware

Aunque hablamos de los G-codes como comando se pensaron como lenguaje de programación para que en la industria se pudieran manipular diferentes tipos de maquinas.

Los comando que admite este firmware, cual es su función y su estructura son los siguientes:

Comando	**Ejemplo**	**Descripción**
G0	G0 X10	Movimiento lineal a la máxima velocidad
G1	G1 X10 Y15 Z0 [F100]	movimiento lineales con la velocidad indicada
G2	G02 X60 Y30 I30 J10 F02	Movimiento curvo (sentido horario)
G3	G03 X60 Y30 I10 J20	Movimiento curvo (sentido antihorario)
G4	G4 P200	Pausa con retardo (Retardo: 200ms)
G28	G28	Ir a origen
G90	G90	Definir coordenadas absolutas
G91	G91	Definir coordenadas relativas
G92	G92	Definir punto actual como origen
M0	M0	Paro programa
M3	M3	Encender motores
M5	M5	Parar motores

Cuadro 6: G-codes firmware

Aunque estos son los G-codes utilizados por mi, hay otros interpretes que utilizan muchos más y con funciones muy diversas como por ejemplo, controlar un láser, extruir plástico o activar y desactivar una fresadora.

3.3. Teoria Utilizada

Para el desarrollo de este firmware hay varios conceptos teóricos, que son importantes para el desarrollo de los algoritmos de aceleración y deceleración de los motores paso a paso, Bresenham, movimientos curvos y PWM. A continuación dedicare un apartado a cada uno de estos conceptos teóricos.

3.3.1. Aceleración y deceleración

Cuando en el apartado de motores paso a paso hemos hablado de que estos motores van por pasos y que estos pasos se tiene que dar en el momento idóneo para conseguir que el motor funcione correctamente. Si eso se perfecciona se puede conseguir acelerar el motor, para poder conseguir movimientos lo más uniformes posibles y velocidades mucho más altas. De la misma forma que aceleramos el motor tenemos que frenarlo.

La aceleración y deceleración es lineal y esta teoría se explico en "Embedded Systems Programing" en Enero de del 2005 "Generate stepper motor speed profiels in real time", un articulo del D. Austin. El algoritmo utilizado esta pensado para la parametrización y cálculo en tiempo real, utilizando operaciones de aritmética de coma fija y esta ideado para microcontroladores. En un primer momento utilice cálculos de coma flotante pero esto hacia que se retrasaran demasiado las operaciones, ya que es más costoso el calculo, así que a pesar de perder precisión utilizo coma fija porque la perdida de precisión no es tan significativa.

Un movimiento se dividirá en tres fases, que serán aceleración, velocidad constante y deceleración, como se puede ver a continuación.

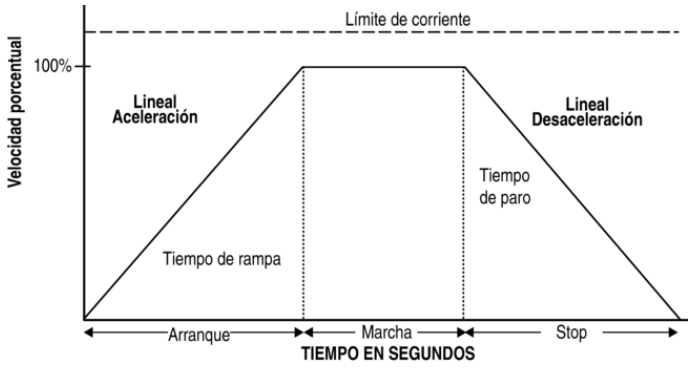

Figura 12: Trayectoria Trapezoidal

En la gráfica anterior tenemos tres puntos críticos que serian los que indicar el fin de la aceleración, el principio de la deceleración y el fin del movimiento. Es necesario conocer esos puntos para poder aplicar cálculos diferentes dependiendo de la etapa donde estemos. El movimiento finalizara cuando se haya completado el recorrido, eso quiere decir cuando hemos recorrido todos los pasos que representan el movimiento. Tanto el punto critico de la aceleración como de la deceleración es el mismo pero a la inversa y este punto se puede calcular con la siguiente formula:

$PasosAce = \frac{vel^2}{2*ace}$

Esta formula se obtiene, si el tiempo en acelerar y los pasos son:

$t = \frac{vel}{ace}$; $n = \frac{ace*t^2}{2angle}$; $n * ace = \frac{vel^2}{2*ace}$;

De aquí se deduce que el número de pasos para alcanzar una velocidad es inversamente proporcional a la aceleración.

Este valor siempre tiene que ser menor que la mitad del espacio total, sinos quiere decir que en lugar de seguir una trayectoria trapezoidal de aceleración sera una un trayectoria triangular y la mitad del recorrido sera el punto critico para pasar de aceleración a desaceleración.

Para controlar la velocidad lo que tendremos que variar es el periodo entre pulsos, dependiendo en la etapa en la que estemos se tendrá que variar de diferentes formas. En la siguiente gráfica se puede observar como se varia la aceleración, la velocidad y la posición.

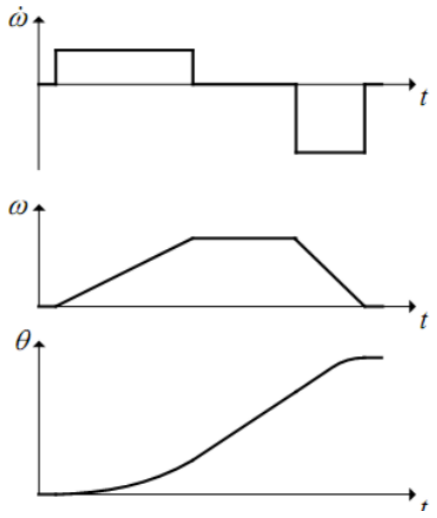

Figura 13: Aceleración, velocidad y posición

Las formulas que representa un movimiento angular, serian el ángulo de paso del motor, posición, y la velocidad son:

$$\alpha = \frac{2\pi}{spr}\,[\text{rad}] \qquad \theta = n\alpha\,[\text{rad}] \qquad \omega = \frac{\alpha}{\delta t}\,[\text{rad/sec}]$$

Figura 14: Formulas movimiento angular

Para la aceleración y frenado tendremos que hacer el siguiente cálculo para saber los tiempo de cada paso, durante la aceleración el tiempo entre pasos se reducirá y durante la desaceleración se aumentara.

$$c_0 = \frac{1}{t_t}\sqrt{\frac{2\alpha}{\dot{\omega}}} \qquad c_n = c_0\left(\sqrt{n+1} - \sqrt{n}\right)$$

Figura 15: Formula aceleración

Como los microcontroladores tiene una capacidad de cálculo limitada y las raíces es una operación muy costosa lo que haremos es utilizar aproximaciones por series de Taylor y solo haremos el calculo inicial con una raíz el resto se hará con aproximaciones por series de Taylor.

Que es una serie de Taylor, es una aproximación de funciones mediante una serie de potencias o sumas de potencias enteras de polinomios, dichas sumas se obtienen mediante las derivadas de la función para un determinado valor y en un entorno donde converjan la serie. En este caso la formula obtenida seria la siguiente:

$$c_n = c_{n-1} - \frac{2c_{n-1}}{4n+1}$$

Figura 16: Formula aproximación de Taylor

Esto tiene un problema introduce un error de 0,44. La forma de corregir este error es multiplicar el calculo inicial Co por 0.676.

En la siguiente gráfica podemos apreciar como se acelera el motor en relación con los pasos y como se va reduciendo el tiempo entre los pasos.

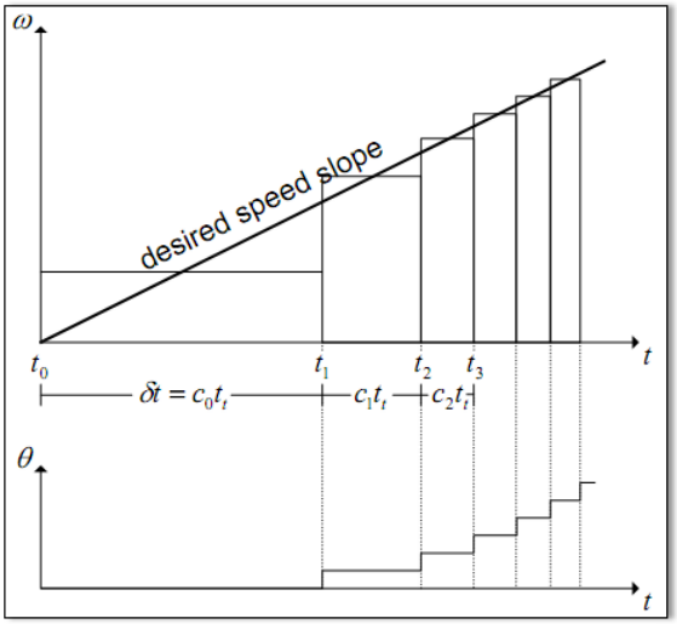

Figura 17: Aceleración en función de los pulsos.

3.3.2. Bresenham

El algoritmo de Bresenham es el encargado de trazar lineas, píxel a píxel, fue diseñado para dibujar gráficos, pero es valido para hacer movimientos lineales con motores paso a paso y se puede extender a cualquier número de motores y que se muevan simultáneamente. Es preciso para crear lineas de rastreo, que convierte mediante rastreo las lineas al utilizar solo cálculos incrementales con enteros.

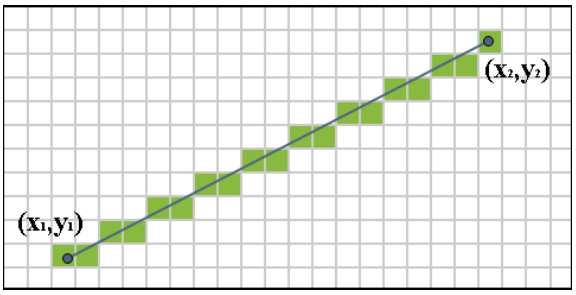

Figura 18: Bresenham lineal

El algoritmo lo que hace es obtener la pendiente de la recta. Esto va a determinar la relación de movimiento entre los dos ejes. Siempre se empezara dando el paso al eje mayor al que tiene más distancia para recorrer y luego mediante un contador se controla a cual de los ejes le toca dar un paso. Por ejemplo si la pendiente de la recta es 1/3 esto quiere decir que cada 3 pasos del Eje Y tendremos que dar un paso en el Eje X.

3.3.3. Movimientos Curvos

Como hemos explicado en el punto anterior hemos utilizado el algoritmo de Bresenham para trazar lineas rectas, pero también se pueden trazar lineas curvas. Para trazar una linea curva necesitamos los siguientes datos, donde empezar, donde terminar y donde esta el centro de la curva. Estos datos los proporciona el G-code.

Para poder hacer la curva con lineas rectas lo que haremos sera fraccionarla en lineas rectas más pequeñas, esto nos da la capacidad de que las rectas sean lo suficientemente cortas como para que no se aprecien. Se definirá un valor que sera el tamaño de las rectas en las que dividiremos la longitud del arco. El problema que nos surge ahora es como obtenemos la longitud del arco, con los datos que tenemos lo que tenemos que utilizar es la ecuación de la longitud de un arco que es:

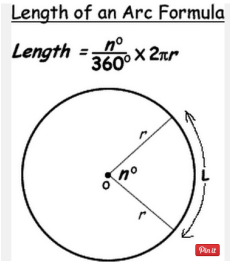

Figura 19: Formula longitud de una curva

Para realizar este calculo la mayoría de lenguajes de programación tienen una función que se denomina arco tangente. Una vez se obtiene la longitud de la curva se divide en pequeñas rectas y esas rectas se pasan al algoritmo de Bresenham y con la acumulación de esas rectas se forma la curva.

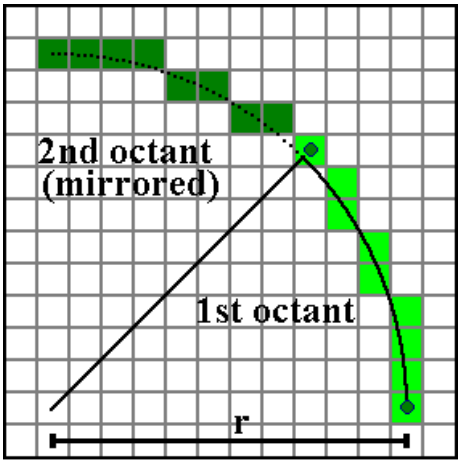

Figura 20: Bresenham curvo

3.3.4. PWM

La modulación por ancho de pulso de una señal, es una técnica en que se modifica el ciclo de trabajo de una señal periódica, para transmitir información. Con un ciclo de trabajo que es el ancho relativo de su parte positiva en relación con el periodo.

$$D = \frac{\tau}{T}$$

D es el ciclo de trabajo

τ es el tiempo en que la función es positiva (ancho del pulso)

T es el período de la función

Figura 21: Formulas PWM

Para este tipo de modulación se utiliza un electrónica que consta de un comparador con dos entradas y una salida. Una de las entradas se conecta a un oscilador de onda de dientes de sierra y la otra para la señal moduladora. La frecuencia de la salida sera igual que la señal de dientes de sierra y el ciclo de trabajo sera como la portadora.

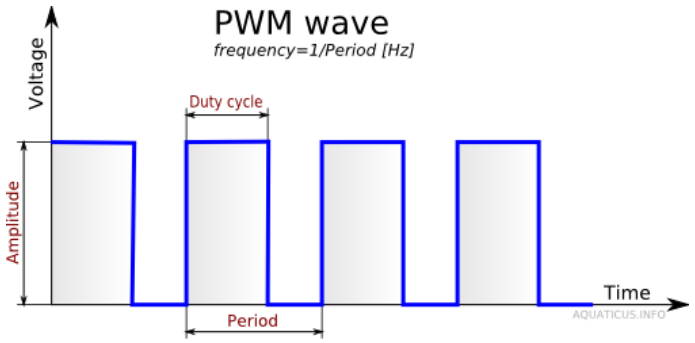

Figura 22: Señal PWM

3.4. Descripción de cada Sketch

En este apartado voy a describir cada uno de los Sketch del firmware. En el firmware podemos encontrar los siguientes Sketch:

1. InterGcode.ino
2. process_buff_gcode.ino
3. servo.ino
4. stepper.ino
5. stepper_cntrl.ino
6. config.ino

3.4.1. InterGcode.ino

Este es el Sketch del programa principal, aquí podemos encontrar las estructura básica de Arduino donde podemos ver el setup, con la inicialización de las bibliotecas y variables necesarias para este Sketch, como el buffer donde almacenaremos los G-codes que vamos a procesar.

```
1  //Libraries
2  #include <TimerOne.h>
3  #define  GPIO2_PREFER_SPEED    1
4  #include "arduino2.h"
5
6  #define COMMAND_SIZE    128 //Buffer size
7
8
9  char palabra[COMMAND_SIZE]; //Buffer gcodes
10 byte serial_count;          //Chars in buffer
11 int no_data = 0;            //No data control
12
13 void setup(){
14
15     //init serial comunication
16     Serial.begin(19200);
17     Serial.println("start");
18
19     //init buffer and pins
20     init_process_gcodes();
21     init_hw_pins();
22
23     //Timer interrupt for stepper motors
24     Timer1.attachInterrupt(step);
25 }
```

Figura 23: setup

En el setup se inicializa la comunicaciones serie, se inicializa el buffer de recepción de G-codes y los pines del Arduino.

```
27 void loop(){
28
29     char c;
30
31     if(Serial.available() > 0){
32
33             c = Serial.read();
34             no_data = 0;
35
36             if(c != '\n'){
37                 palabra[serial_count]=c;
38                 serial_count++;
39             }
40     }
41
42     else{
43         no_data++;
44         delayMicroseconds(100);
45     }
46
47     if(serial_count && (c == '\n'|| no_data > 100)){
48
49             //process gcode
50             process_gcodes(palabra, serial_count);
51
52             //clear command
53             init_process_gcodes();
54     }
55
56     if(no_data > 1000)
57         motors_off();
58 }
```

Figura 24: loop

La función principal del loop es si ha datos disponibles en el puerto serie, almacenarlos en el buffer y después llamar a la función de procesar G-codes. Tiene un contador de espera por si no llegan datos pues apagar los motores.

3.4.2. Process_buff_gcode.ino

Process_gcode

void process_gcodes(char buffer[], int buffer_size)

El método process_gcode es el encargado de interpretar que G-code hay en el buffer y recoger la información necesaria para realizar la acción que indica ese G-code. Por ejemplo si vamos hacer un movimiento tendremos que saber en que ejes, que longitud y que velocidad, después llamaremos el método que traza las lineas rectas. Una cosa destacable de este método es la realizacion de lineas curvas, mediante lineas rectas como hemos explicado teóricamente en el apartado 3.3.3. Este tipo de movimiento se hará con los G-codes G2 y G3 y lo que hace es dividir la curva en lineas rectas y después llamar sucesivamente a move_line para trazar movimientos lineales más pequeños.

Init_process_gcode

void init_process_gcodes()

Método encargado de inicializar el buffer de recepción de G-codes, lo recorre y lo pone todo a 0 y el numero de componentes disponibles en el buffer a 0.

Parse_Number

float parse_Number(char key, char buffer[], int buffer_size)

Parse_Number se encarga de devolver el número que hay detrás de una determinada clave. Los G-codes son comandos compuesto por letras y números, las letras actúan como cabecera y identifican al numero por ejemplo G1 X100 es un G-code de tipo 1, como indica el 1 de después de la G y que tiene que desplazar 100 mm en el eje X.

Find_command

bool find_command(char key, char buffer[], int buffer_size)

Este método es el encargado de averiguar si existe una clave, devolviendo un booleano. Por ejemplo se utiliza para saber si en el buffer hay algún tipo de G-code o es otro tipo de dato que no es valido para el interprete.

3.4.3. Stepper_cntrl.ino

Move_line

void move_line()

Es el método más importante de todo el firmware, es el encargado de general los movimientos lineales de los motores. Aquí podemos encontrar los dos algoritmos imprescindibles para el funcionamiento de este proyecto. Son el algoritmo de Bresenham que es el encargado de coordinar los dos motores si es necesario para hacer un movimiento como ya hemos explicado teóricamente en el punto 3.3.2.

```
32  //master axis (bresemham)
33  max_delta = max(delta_steps.x,delta_steps.y);
34
35  long x_counter = -max_delta/2;
36  long y_counter = -max_delta/2;
37
```

Figura 25: Algoritmo Bresenham

```
45
46      if(x_can_step){
47          x_counter += delta_steps.x;
48          if(x_counter > 0){
49                  noInterrupts();
50                  flag_step++;
51                  interrupts();
52                  x_counter -= max_delta;
53                  if(x_direction)
54                      current_steps.x++;
55                  else
56                      current_steps.x--;
57          }
58      }
59
60      if(y_can_step){
61          y_counter += delta_steps.y;
62          if(y_counter > 0){
63                  noInterrupts();
64                  flag_step++;
65                  interrupts();
66                  y_counter -= max_delta;
67                  if(y_direction)
68                      current_steps.y++;
69                  else
70                      current_steps.y--;
71          }
72      }
```

Figura 26: Algoritmo Bresenham

Como podemos observar en las capturas anteriores, lo que hacemos es iniciar los contadores a la mitad de la distancia máxima en negativo. Luego sumamos la distancia y si es positivo es que tenemos que dar un paso con ese motor y eso lo indica el flag_step, despues de eso le restamos la distancia máxima.

El otro algoritmo relevante que podemos observar en este método es el algoritmo que calcula, los tiempos entre pulsos, como ya hemos explicado en la teoría de 3.3.1 y también en el funcionamiento de motores paso a paso del 2.2.1.

```
75          if(accel_counter <= countLim)
76              newtp = newtp - 2.0*newtp/(4.0*accel_counter+1.0);
77          else if ((countLim < accel_counter) && (accel_counter <= (SMax-countLim)))
78              ;
79          else
80              newtp = newtp + 2.0*newtp/(4.0*(SMax-accel_counter)+1.0);
81
```

Figura 27: Cálculo tiempo entre pasos

Como se aprecia en la imagen anterior, countLim es el que indica el final de la aceleración y también indica el final de la velocidad constante si lo restamos a la distancia total. También se puede observar como el calcula de la aceleración cada vez sera más pequeño, durante el tiempo constante es siempre el mismo serán pulsos constantes y iguales al ultimo pulso de la aceleración y durante el frenado lo que se hará es ir aumentando el tiempo entre pulsos.

Calc_accel_point

long calc_accel_point(long master_axis)

Calc_accel_point es el encargado de calcular el punto donde se termina la aceleración. Y también comprueba que eso suceda antes de la mitad del espacio recorrido.

Set_motion

void set_motion(float x, float y, float z)

El método set_motion es el encargado de fijar las distancias objetivo, de calcular las deltas que son la diferencia entre la distancia actual y la distancia objetivo. También se encarga de transformar todas las distancias a pasos y de calcular la dirección de los motores aparte de accionar el pin que cumple la tarea de asignar la dirección.

```
112 void set_motion(float x, float y, float z){
113
114     //set targets
115     target_units.x = x;
116     target_units.y = y;
117     target_units.z = z;
118
119     //figure our delta
120     delta_units.x = abs(target_units.x - current_units.x);
121     delta_units.y = abs(target_units.y - current_units.y);
122     delta_units.z = abs(target_units.z - current_units.z);
123
124     //transforms steps
125     target_steps.x = X_STEPS_PER_MM *  target_units.x;
126     target_steps.y = Y_STEPS_PER_MM * target_units.y;
127     //target_steps.z =
128
129     delta_steps.x = abs(target_steps.x - current_steps.x);
130     delta_steps.y = abs(target_steps.y - current_steps.y);
131     //delta_steps.z =
132
133     //set direction
134
135     x_direction = (target_units.x >= current_units.x);
136     y_direction = (target_units.y >= current_units.y);
137
138     digitalWrite2f(X_DIR_PIN, x_direction);
139     digitalWrite2f(Y_DIR_PIN, y_direction);
140
141 }
```

Figura 28: setmotion

Este método se invoca desde process_gcode y para preparar el movimiento antes de llamar al metodo move_line que es el el encargado de realizarlo.

Set_position

void set_position(float x, float y, float z)

Método encargado de modificar la posición actual después de hacer un movimiento, se invoca de move_line y también transforma la posición actual en pasos.

```
146 void set_position(float x, float y, float z){
147
148     //set currents
149     current_units.x = x;
150     current_units.y = y;
151     current_units.z = z;
152
153     //transforms steps
154     current_steps.x = X_STEPS_PER_MM *  current_units.x;
155     current_steps.y = Y_STEPS_PER_MM * current_units.y;
156     //current_steps.z =
157
158
159 }
```

Figura 29: setposition

Set_feedrate

void set_feedrate(float feedrate)

Este método también se invoca desde process_gcode para poder preparar la velocidad. Los G-codes utilizan velocidades lineales y mi firmware utiliza velocidad angular con lo cual tendremos que dividir la velocidad por el radio del motor más la polea para sacar la velocidad angular.

```
163 void set_feedrate(float feedrate){
164
165     float radio = 6.15;
166
167     feedrate_angular = (feedrate/radio)*FACTOR_STEPING; //rad/min
168
169 }
```

Figura 30: setfeedrate

Home

void home()

El método home es el encargado de mover todos los ejes a la posición de homming, que seria la inicial para comenzar una secuencia de movimientos. Lo que hace es mover cada eje con una velocidad fijada sin ningún tipo de aceleración, ya que lo

idea para este tipo de movimientos es un velocidad baja para no tener problemas de rebotes al llegar al final del eje. Primero mueve el eje X hasta que el se activa el pin del final de carrera del eje X, lo comprueba con digitalRead2d(X_END_STOP) y después apaga el motor. Hace eso mismo con el eje Y y por último con el servo que utiliza el método servoDown().

```
174 void home(){
175
176     //Servo move up to avoid obstacles
177     servoUp();
178
179     //Move home X
180
181     motors_on();
182     digitalWrite2f(X_DIR_PIN,DIR_HOMING_X);
183
184
185     while(true){
186         if(digitalRead2f(X_END_STOP) == HIGH){
187             motors_off();
188             break;
189         }
190         MakeSpeedStep(FEEDRATE_HOME, X_STEP_PIN);
191     }
192
193
194     //Move home Y
195     digitalWrite2f(Y_DIR_PIN,DIR_HOMING_Y);
196     motors_on();
197     while(true){
198         if(digitalRead2f(Y_END_STOP) == HIGH){
199             motors_off();
200             break;
201         }
202         MakeSpeedStep(FEEDRATE_HOME, Y_STEP_PIN);
203     }
204
205     //Move home Z (servo)
206     servoDown();
207     set_position(0.0,0.0,0.0);
208
209 }
```

Figura 31: home

Este método es invocado si llega un G-code de tipo G28.

3.4.4. Stepper.ino

Init_hw_pins

void init_hw_pins()

Init_hw_pins es el encargado de iniciar los pines con el modo adecuado para que el Arduino sepa si son entradas y salidas, se utiliza pinMode2f(pin,mode). Esto se invoca una sola vez al principio desde el setup.

Motors_on

void motors_on()

Motors_on es el que se encarga de encender los motores paso a paso, se encienden en nivel bajo, con la función digitalWrite2f(ENABLE_PIN,LOW).

Motors_off

void motors_off()

Motors_off es el que se encarga de encender los motores paso a paso, se apaga en nivel alto, con la función digitalWrite2f(ENABLE_PIN,HIGH).

MakeSpeedStep

void MakeSpeedStep(float v, GPIO_pin_t step_pin)

Este método es el que utilizo para dar los pulsos con una velocidad fija, lo utilizo solo para hacer homing ya que no utilizo patrón trapezoidal de velocidad. Con lo cual lo único que hago es hacer un pulso en algo un tiempo constante de pulso, lo pongo a nivel bajo y por ultimo un tiempo contante de espera entre pulsos que se determina por la velocidad suministrada.

```
41
42 void MakeSpeedStep(float v, GPIO_pin_t step_pin){
43
44     digitalWrite2f(step_pin,HIGH);
45
46     delayMicroseconds(1);
47
48     digitalWrite2f(step_pin,LOW);
49
50     delayMicroseconds((1.0/v)*1E6);
51
52 }
```

Figura 32: MakeSpeed

Step

void step()

El método step es uno de los más importantes, ya que es el método que utiliza la interrupción para realizar los pasos desde el método move_line. Lo que hacemos es comprobar si el flag_step es 1 o 2 porque así sabemos que tenemos que hacer un paso con el motor principal o dos paso con ambos motores. Para hacer dos pasos con ambos motores lo que hago es poner los dos a nivel algo a la vez así no perdemos tiempo entre los pasos. En el apartado de interrupciones explicare con más detalle la problemática y porque he utilizado interrupciones.

```
53
54  void step(){
55
56      if(flag_step==1){
57          digitalWrite2f(master_pin,HIGH);
58          delayMicroseconds(1);
59          digitalWrite2f(master_pin,LOW);
60      }
61
62      if(flag_step==2){
63          digitalWrite2f(master_pin,HIGH);
64          digitalWrite2f(slave_pin,HIGH);
65          delayMicroseconds(1);
66          digitalWrite2f(master_pin,LOW);
67          digitalWrite2f(slave_pin,LOW);
68      }
69
70      flag_step = 0;
71  }
```

Figura 33: step

Can_step

bool can_step(GPIO_pin_t end_pin, long max_length_axis, long current, long target, byte direction)

can_step se encarga de comprobar si se tiene que dar un paso más en ese eje. Comprueba que no haya llegado a la posición objetivo, que no se active el final de carrera o que no llegue al tamaño máximo del eje.

```
73
74  bool can_step(GPIO_pin_t end_pin, long max_length_axis, long current, long target, byte direction){
75      if(target == current)
76          return false;
77      else if (digitalRead2f(end_pin) == HIGH && !direction)
78          return false;
79      else if (current >= max_length_axis && direction )
80          return false;
81      return true;
82  }
```

Figura 34: CanStep

3.4.5. Servo.ino

ServoInit

void servoInit()

Método encargado de iniciar el pin del servo, y también la interrupción que controla el servo. Explicare con más detalle el tema de las interrupciones en un apartado especifico.

ISR

ISR(TIMER2_COMPA_vect)

Método que se encarga de calcular y fijar el tiempo de las interrupciones para que el servo se mantenga en la posición deseada.

servoSetPosition

void servoSetPosition(uint16_t highTimeMicroseconds)

Método que invocas par definir la posición, se le pasa un tiempo en microsegundos que sera en tiempo entre pulsos, para el PWM.

servoUp y servoDown

En este firmware solo he diseñado dos posiciones del servo, porque su principal utilidad es mover el eje Z solo para esquivar obstáculos cuando se van hacer movimientos de desplazamiento.

3.4.6. config.ino

Este skech lo utilizo para definir y configurar los diferentes parámetro de la maquina:

1. Parámetros para la transformación de mm a pasos.
2. Parámetro de la división de las curvas en pequeñas rectas.
3. Parámetros de velocidad
4. Parámetros de dirección de homming
5. Parámetros de tamaño máximo de ejes
6. Parámetro de aceleración
7. Los pines de cada uno de las entradas y salidas.

3.5. Interrupciones

3.5.1. Teoría interrupciones

Una interrupción de un microcontrolador es una señal que interrumpe la actividad normal. Hay tres posibles eventos que puedan disparar interrupciones:

1. Un evento hardware, previamente definido.
2. Un evento programado o Timer.
3. Una llamada por software.

Cuando una interrupción se dispara la ejecución normal del programa se suspende pero dejando un puntero para saber donde tiene que volver y continuar con la ejecución normalmente. Salta ejecutar lo que de normal se denomina como Interrupt Service Handler o ISH (Servicio de gestión de interrupción). El concepto de interrupción surge por la necesidad que se tiene por atender a algo inmediatamente esto crea un paralelismo simulado, ya que no es paralelo pero no se ejecuta todo secuencialmente.

Otra de las grandes ventajas de las interrupciones es que nos permiten tener un programa más organizado porque no tenemos que estar comprobando que se tenga que ejecutar algo, simplemente cuando se tiene que ejecutar se ejecuta y continua el programa, ahorrándonos lineas de código y teniendo un código mucho más limpio.

De las tres interrupciones nombradas anteriormente, Arduino solo soporta dos tipos eventos hardware y eventos programados o Timers. Para este proyecto hemos utilizado interrupciones programadas que las explicare con más detalle a continuación.

Una interrupción hardware, esta pensada para reaccionar ante un pulso eléctrico, para que sea lo suficientemente rápida como para darse cuenta. Con lo cual cuando un pin de Arduino y se cumpla la condición de disparo se ejecutara una función. Las posibles condiciones de disparo son:

1. LOW, La interrupción se dispara cuando el pin es LOW.
2. CHANGE, Se dispara cuando pase de HIGH a LOW o viceversa.
3. RISING, Dispara en el flanco de subida (Cuando pasa de LOW a HIGH).
4. FALLING, Dispara en el flanco de bajada (Cuando pasa de HIGH a LOW).
5. Y una solo para el DUE: HIGH se dispara cuando el pin esta HIGH.

Las interrupciones temporizadas que son las que utilizamos en este proyecto tanto para los motores paso a paso como para el PWM del servo. Este tipo de interrupciones se podrían comparar con los que haces con el delay para que se active por ejemplo un led durante un tiempo, pero con la diferencia de que no paramos el programa. Con lo cual podemos hacer otras cosas y cuando llega el momento de la interrupción se ejecuta la función y después vuelve al programa principal.

Los Arduinos disponen de un crital de 16 MHz, con lo cual podríamos fijar teóricamente una interrupción cada 1/16000000 segundos, esto es demasiado rápido. Pero cada timer dispone de un registro interno donde podemos indicar cada cuantos ticks del reloj debe dispararse. Arduino UNO dispone de 3 timers de diferente resolución, dos de 8 bits y uno de 16 bits. Si por ejemplo utilizamos el de 16 bits nos permite contar hasta $2^{16} = 65{,}536$ y si utilizamos el de 8 bits podemos contar hasta 256. Para disponer de más flexibilidad también disponemos de divisores que nos permiten frenar la velocidad del cristal de cuarzo, estos divisores son 1,8,64,256,1024.

3.5.2. Las interrupciones en el firmware

Después de desarrollar la aceleración de un forma secuencial, me di cuenta de que los cálculos era demasiado costosos para el tiempo de duración de los pulsos. Durante la aceleración los cálculos eran tan costosos que cuando pasaba de aceleración a velocidad constante había perdida de pasos ya que durante el periodo de velocidad constate el tiempo entre pulso descendía y hacia que se descortinara el motor. El código de la aceleración de un motor paso a paso es el siguiente:

```
55  while(counter < SMax){
56
57      MakeTimeStep(newtp);
58
59      if( counter <= countLim){
60          newtp = newtp-2.0*newtp/(4.0*counter+1.0);
61
62      }
63
64      else if ((countLim < counter) && (counter <= (SMax-countLim) )){
65          ;
66      }
67      else {
68          newtp=newtp+2.0*newtp/(4.0*(SMax-counter)+1.0);
69      }
70      counter++;
71  }
```

Figura 35: Cálculos del tiempo entre pulsos sin interrupciones

Como podemos observar, hacemos el calculo de cada una de las etapas y de cada uno de los pasos respecto al calculo anterior, y después del calculo hacemos la llamada a un método denominado MakeTimeStep, donde pasamos el tiempo entre pulsos de ese paso, el problema es que el siguiente calculo le va añadir mucho más tiempo del necesario. Esta versión ya esta implementada con coma fija pero aun así los cálculos son lentos.

Motores paso a paso

La única solución posible para este problema son las interrupciones temporizadas. Ya que podemos realizar los cálculos del tiempo entre pulsos y si en cualquier momento se necesita hacer ese pulso se interrumpirá el programa y se realizara el pulso. Para esto hemos utilizado un librería para el timer 1 que es el que más precisión tiene 16 bits, la librería se llama TimerOne. En la siguiente captura lo podemos ver con detalle como lo hacemos.

```
75          if(accel_counter <= countLim)
76              newtp = newtp - 2.0*newtp/(4.0*accel_counter+1.0);
77          else if ((countLim < accel_counter) && (accel_counter <= (SMax-countLim)))
78              ;
79          else
80              newtp = newtp + 2.0*newtp/(4.0*(SMax-accel_counter)+1.0);
81
82          Timer1.setPeriod(newtp/1000.0);
83          Timer1.restart();
84          while(flag_step>0);
85
86          noInterrupts();
87          accel_counter++;
88          interrupts();
89
```

Figura 36: Cálculos del tiempo entre pulsos con interrupciones

Como podemos ver lo que hago es hacer el calculo y modificar el tiempo de la interrupción, mediante setPeriod y reinicio la interrupción mediante restart. Cuando pase ese tiempo calculado entonces llamaremos al método step que hemos explicado anteriormente en el punto 3.4.4. Tenemos que tener varias cosas en cuenta para las interrupciones funcionen bien, una de ellas es que no se puede interrumpir la modificación de accel_counter y tampo los flag_step del algoritmo de Bresenham ya que son los que indican si se tiene que hacer un o dos pasos en la interrupción. Tampoco se puede realizar otra calculo hasta que no se haya realizado el paso y para ello se utiliza el bucle while de espera con el flag_step>0, ya que eso indica que mientras haya un paso pendiente no se puede continuar, de esta forma conseguimos sincronizar los pasos con los cálculos.

Servo

Después de utilizar el timer 1 para los motores paso a paso, resulta que la librería servo de Arduino utiliza ese timer, con lo cual no podía utilizar esa librería. Tuve que utilizar otro timer diferente de menor resolución pero suficiente para el control de PWM de los servos.

Para esta tarea no encontré ninguna librería que funcionara correctamente con el timer 2. Por eso me base en un solución de la empresa Pololu, para la utilización de timer 2 para servos. Los primero que tenemos que hacer es preparar la interrupción. Para ellos utilizamos la función servoInit().

```
62 void servoInit()
63 {
64   digitalWrite2f(SERVO_PIN, LOW);
65   pinMode2f(SERVO_PIN, OUTPUT);
66
67   // Turn on CTC mode.  Timer 2 will count up to OCR2A, then
68   // reset to 0 and cause an interrupt.
69   TCCR2A = (1 << WGM21);
70   // Set a 1:8 prescaler.  This gives us 0.5us resolution.
71   TCCR2B = (1 << CS21);
72
73   // Put the timer in a good default state.
74   TCNT2 = 0;
75   OCR2A = 255;
76
77   TIMSK2 |= (1 << OCIE2A);  // Enable timer compare interrupt.
78   sei();                    // Enable interrupts.
79 }
```

Figura 37: ServInit

Para iniciar una interrupción temporizada tenemos que seguir los siguientes pasos, como se ve en la captura:

1. Iniciar el modo del pin del servo, como salida.
2. Activar el modo CTC (clear timer on compare), este modo lo que hace es activar las interrupciones cuando el contador llega a un valor especifico.
3. También se prescala el timer a 8 para tener un resolución de 0.5 microsegundos.
4. Iniciar el temporizador correctamente.
5. Habilitar la interrupción.

En el registro OCR2A es donde se inicia el contador, cuando se alcanza ese contador se activa la funcion ISR:

```
17  ISR(TIMER2_COMPA_vect)
18  {
19    // The time that passed since the last interrupt is OCR2A + 1
20    // because the timer value will equal OCR2A before going to 0.
21    servoTime += OCR2A + 1;
22
23    static uint16_t highTimeCopy = 3000;
24    static uint8_t interruptCount = 0;
25
26    if(servoHigh)
27    {
28      if(++interruptCount == 2)
29      {
30        OCR2A = 255;
31      }
32
33      // The servo pin is currently high.
34      // Check to see if is time for a falling edge.
35      // Note: We could == instead of >=.
36      if(servoTime >= highTimeCopy)
37      {
38        // The pin has been high enough, so do a falling edge.
39        digitalWrite2f(SERVO_PIN, LOW);
40        servoHigh = false;
41        interruptCount = 0;
42      }
43    }
44    else
45    {
46      // The servo pin is currently low.
47
48      if(servoTime >= 40000)
49      {
50        // We've hit the end of the period (20 ms),
51        // so do a rising edge.
52        highTimeCopy = servoHighTime;
53        digitalWrite2f(SERVO_PIN, HIGH);
54        servoHigh = true;
55        servoTime = 0;
56        interruptCount = 0;
57        OCR2A = ((highTimeCopy % 256) + 256)/2 - 1;
58      }
59    }
60  }
```

Figura 38: ServoISR

Lo que hacemos es si el servo esta a nivel alto, comprobamos que si es momento de ponerlo en nivel bajo. Si esta a nivel bajo a la hora de activar la interrupción, lo que haremos es poner a nivel alto y calcularemos el siguiente 0CR2A respecto al tiempo que se a pasado desde el método servoSetPosition.

```
81  void servoSetPosition(uint16_t highTimeMicroseconds)
82  {
83    TIMSK2 &= ~(1 << OCIE2A); // disable timer compare interrupt
84    servoHighTime = highTimeMicroseconds * 2;
85    TIMSK2 |= (1 << OCIE2A); // enable timer compare interrupt
86  }
```

Figura 39: ServoSPosi

Este método es una simple asignación sobre una variable global volatile, que sera accedida desde ISR, pero por esa razón tenemos que desactivar las interrupciones para que no la asignación se haga correctamente.

3.6. Bibliotecas utilizadas

En este firmware he utilizado dos bibliotecas. TimerOne que es la que he utilizado para la implementación de la interrupción temporizada, para los motores paso a paso como hemos explicado en los puntos 3.3.1 y 3.5. La biblioteca dispone las siguientes funciones:

Función	**Utilidad**
Timer1.initialize(microseconds);	Inicialización del temporizador y la interrupción.
Timer1.setPeriod(microseconds);	Modificar el periodo del temporizador después de iniciar la inrrupción.
Timer1.start();	Inicia el temporizador, comenzado un nuevo periodo.
Timer1.stop();	Detiene el temporizador.
Timer1.restart();	Reinicia el temporizador, desde el principio de un nuevo periodo.
Timer1.resume();	Reanudar la ejecución de un temporizador detenido.
Timer1.pwm(pin, duty);	Configura los pines PWM. Duty varia de 0 a 1023
Timer1.setPwmDuty(pin, duty);	Establecer un nuevo PWM, sin volver a configurar el pin.
Timer1.disablePwm(pin);	Desactivar PWM en un pin.
FTimer1.attachInterrupt(function);	Ejecutar una función cada vez que el período del temporizador acaba. La función se ejecuta como una interrupción, por lo que es necesario tener cuidado especial para compartir cualquier variable.
Timer1.detachInterrupt();	Deshabilitar la interrupción, por lo que la función ya no se ejecuta.

Cuadro 7: Funciones TimerOne

La otra biblioteca utilizada en este firmware es arduino2, su función principal es mejorar la implementación de las entradas y salidas digitales. La razón de utilizar esta librería es por la misma razón que utilizamos interrupciones y cálculos con coma fija, intentar exprimir todo lo posible la velocidad del microcontrolador y intentar conseguir que la interrupciones duren lo menos posible. Resulta que la implementación de esas funciones por el equipo Arduino no es lo más óptima por esa razón se creo esta librería porque utiliza una lógica mucho más veloz de acceso y en esta tabla se puede ver las mejoras:

Arduino uno

Type	**Pin is constant**	**Pin is variable**	**Test program size**
Arduino I/O “as is"	4.1 us	4.1 us	3098 B
Arduino I/O without timer check (see note 1 below)	3.4 us	3.4 us	2998 B
I/O 2 with integer arguments (see note 2 below)	0.8 us	2.0 us (2.8 us)	2876 B (2944 B)
I/O 2 with its native arguments (pin code)	0.6 us	1.1 us (1.9 us)	2856 B (2924 B)

He utilizado las funciones más rápidas para ello simplemente hay que:

1. Importar la librería con: #include “arduino2.h"
2. Declarar los pones con pinMode2f(pin, mode);
3. Utilizar las funciones digitalWrite2f(pin, mode); o digitalRead2f(pin);

4. Descripción del Hardware

La CNC (control numérico por computador) se basa en un robot cartesiano de 3 ejes, este tipo de robots son muy comunes en la industria y cada vez más utilizados en el movimiento maker. Este tipo de robots se basa en que los tres ejes de control son lineales y forman ángulos rectos uno respecto a otro. Esta configuración mecánica permite simplificar las ecuaciones de control. Como ya he indicado se dividen en 3 ejes voy a describir de forma separada cada uno de esos ejes para poder entender su construcción.

4.1. Eje X

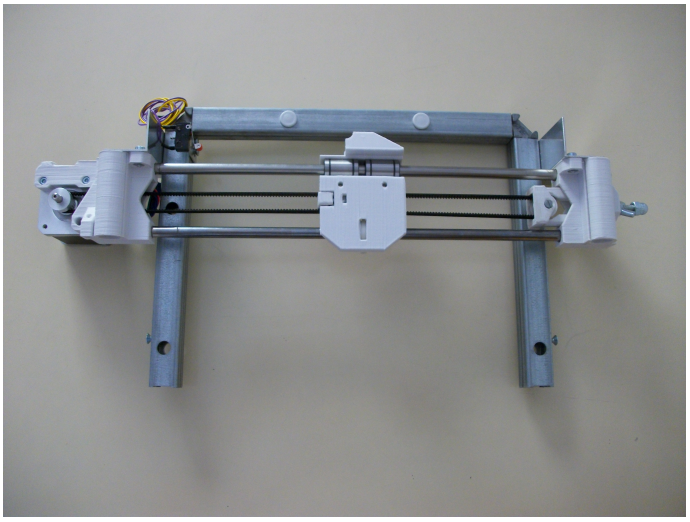

Figura 40: Eje X

Este eje es el portador de la herramienta que se sitúa en el eje z. Esta formado por piezas impresas en 3d y piezas metálicas. Esta sujeto a un marco que es el que da estabilidad a toda la estructura. Este marco esta formado por unas piezas de hierro galvanizado y unas escuadras que facilitan que el marco sea uniforme y este a escuadra. Como la funcionalidad principal de este eje es moverse linealmente, se desplaza sobre unos rodamientos lineales de tipo LM8uu y unas varillas lisas, calibradas y macizas para que sean resistentes. Los rodamientos son autolubricantes y se encarga de un correcto deslizamiento. La tracción se consigue mediante una correa dentada de tipo GT2 y una polea del mismo tipo de diente que va unida al eje del motor paso a paso.

Metálico	Mecánicos	Plástico extruido
Escuadras marco	Poleas GT2 de 20 dientes	Sujeción varillas y tensor
Sujeción final de carrera	Correa GT2 de 6 mm de ancho	Sujeción motor
Barillas lisas	Rodamientos axiales B608zz	Polea B608zz
Tornillos y tuercas	Rodamientos lineales LM8uu	Carro para la herramienta
Marco de sujeción		Tensor

Cuadro 8: Meteriales eje x

4.2. Eje Y

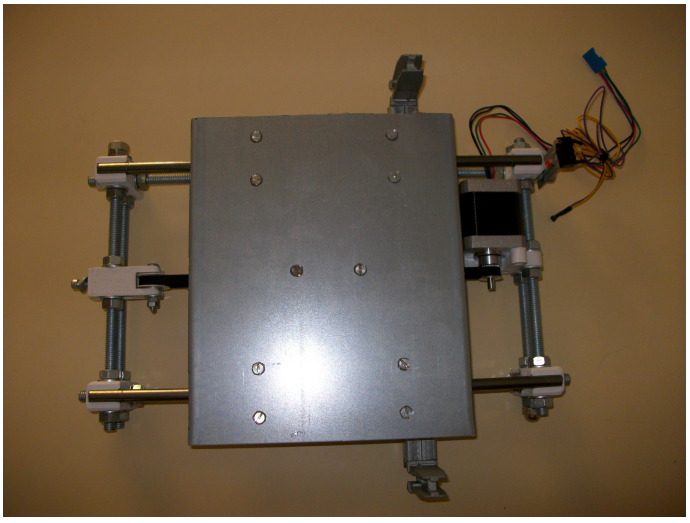

Figura 41: Eje Y

El eje Y es el encargado de la sujeción de la base, donde se ira el objeto que queremos observar con mayor precisión. Esta compuesto de piezas de diferentes materiales desde metálicos hasta plástico extruido por un impresora 3d. También se utilizan elementos mecánicos que ayudan al correcto funcionamiento como pueden ser los rodamientos. Mecánicamente funciona de la misma manera que el eje X, una correa unida al motor mediante un polea dentada es la encargada de ejercer el movimiento lineal. A continuación voy a clasificar los materiales por el tipo de material.

Metálicos	Mecánicos	Plástico extruido
Base	Poleas GT2 de 20 dientes	Patas sujeción base
Varillas roscadas	Correa GT2 de 6 mm de ancho	Sujeción motor
Varillas lisas	Rodamientos axiales B623zz	Polea B623zz
Sujeción final de carrera	Rodamientos lineales LM8uu	Sujeción de base para LM8uu
Parte inferior del marco		Arrastrador de base
Tuercas y tornillos		Tensor y Sujeción polea

Cuadro 9: Meteriales eje y

4.3. Eje Z

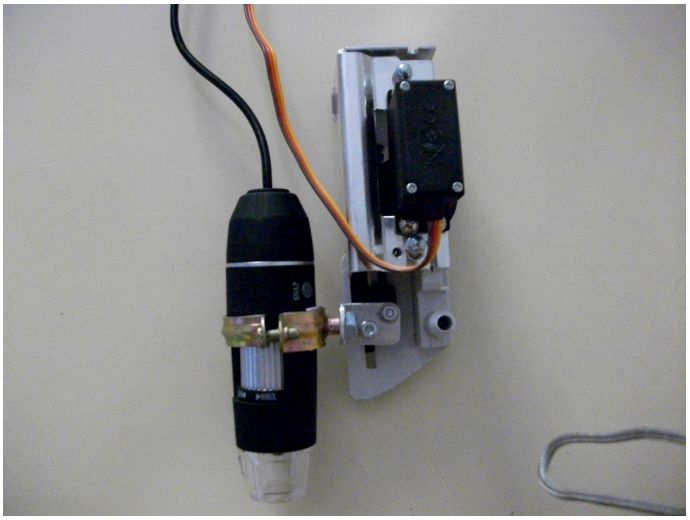

Figura 42: Eje Z

Este eje es el que tiene la función de sujetar la herramienta a sido el más costoso de fabricar y diseñar ya que no se basa en ningún diseño previo, es totalmente original. Se utiliza un servo que acciona una polea dentada que empuja una cremallera, donde esta sujeta una argolla que a su vez aferra la herramienta a utilizar. Tanto el servo como el conjunto de las piezas están sujetos con unas escuadras para dar rigidez a la estructura. En el eje z no hay materiales impresos en 3D pero en un futuro seria mucho más apropiado que lo fueran, son más fáciles de replicar, con lo cual otras personas con impresoras 3D podrían construir la CNC y también si cualquier pieza se daña se puede sustituir. Como no se ha utilizado partes de plástico extruido las piezas se han reciclado de un impresora antigua, obteniendo tanto la cremallera como la polea. Lo materiales utilizados son los listados a continuación.

Metálicos	Mecánicos
Escuadra sujeción al carro	Polea Dentada
Escuadra sujeción cremallera	Cremallera
Argolla sujeción herramienta	

Cuadro 10: Meteriales eje z

5. Conclusiones

Al finalizar el proyecto puedo concluir que todos los objetivos que se plantearon al inicio se han cumplido. Se ha desarrollado un sistema mecánico completo que es capaz de probar como funciona el firmware y que tiene una posible utilidad. Se ha desarrollado y fabricado los componentes electrónicos para el correcto funcionamiento del proyecto. Estas dos cosas completan junto con el firmware, un objetivo oculto, que era una de mis intenciones desde el principio que fuera un proyecto lo más completo posible y que intervinieran diferentes áreas.

El firmware cumple las expectativas, es capaz de gestionar los G-codes, para poder reproducir los movimientos con varios motores, gracias al algoritmo de Bresenham. Los motores tiene una aceleración suave gracias a la trayectoria trapezoidal implementada junto a las interrupciones. Las interrupciones solucionaron uno de los problemas principales del firmware que era que los cálculos del tiempo entre pulsos eran muy costosos y hacían que los pasos de los motores se desincronizaran, al pasar de aceleración a velocidad constante. Para mejorar esta cuestión he utilizado también calculo de coma fija en lugar de coma flotante. Dentro de la implementación del firmware he utilizado un servo para controlar la herramienta y eso me ha permitido aprender como controlar un tipo de motor que no había utilizado antes y entender mucho mejor el concepto de señal PWM.

Por ultimo en el terreno personal estoy muy satisfecho de que me permitan desarrollar este tipo de proyecto, ya que incluye muchas ramas de la informática que me interesan. Al igual que desarrollar este proyecto me ha enseñado como organizarme, ya que en este tipo de proyecto las piezas tiene que ensamblarse y muchas de ellas no funcionan sin las otras y tiene que estar desarrolladas previamente. Estoy satisfecho con mi trabajo porque desde de la investigación realizada para el proyecto no encontré ningún firmware que incluya aceleración, interrupciones y que este realizado por una sola persona.

6. Líneas Futuras

Este proyecto ha cumplido todos sus objetivos iniciales, pero como todo proyecto tiene mejoras y posibles ampliaciones y voy hacer un listado de las que a mi me gustarían, voy a incluir en este listado ampliaciones relacionadas con todo el proyecto desde la electrónica hasta el firmaware, pasando por el hardware.

1. La electrónica se podría integrar en un único PCB, tendríamos que incluir el Atmega328, el Atmega 16U2 para la comunicación USB, alimentación para los motores etc. Esto supondría tener un uso exclusivo para este tipo de maquinas.

2. Utilizar motor paso a paso para eje Z, dando más flexibilidad de movimiento a ese eje y permitiendo que CNC (control numérico por computador) se pudiera utilizar para otras funciones. Esto haría que tuviéramos que modificar el firmware para añadir un motor paso a paso más.

3. Utilizar piezas impresas con tecnología 3D también para el Z. Permitiendo mejor replicación y reparación.

4. Estudiar otra tipos de algoritmos de aceleración para motores paso a paso comprarlos con estos y decidir cual es el más adecuado.

5. Utilizar control de bucle cerrado para los motores paso a paso. Tendríamos que añadir motores con encoders pero esto nos permitiría utilizar PID para el control tanto de velocidad como de posición. Solucionaría el famosos problema de perdida de pasos que tiene este tipo de motores.

6. Crear un software para el análisis de circuitos con el microscopio. Podrías indicar cuales son las zonas a analizar y enviaría los G-codes al interprete para que se desplazara automáticamente y poder comprobar de manera visual o automática si el circuito es correcto.

7. Bibliografía

Referencias

[1] Wikipedia. Definición robot cartesiano: http://es.wikipedia.org/wiki/Robot_de_coordenadas_cartesianas

[2] Wikipedia. Definición Arduino: http://es.wikipedia.org/wiki/Arduino

[3] Wikipedia. Definición motor paso a paso: http://es.wikipedia.org/wiki/Motor_paso_a_paso

[4] Pololus. Información driver stepper: https://www.pololu.com/product/1182/specs http://electronilab.co/tienda/driver-motor-paso-a-paso-1a-a4988-pololu/#tab-additional_information

[5] Wikipedia. Servo: http://es.wikipedia.org/wiki/Servomotor_de_modelismo

[6] UECIDE UECIDE: http://uecide.org/tour

[7] Arduino. Lenguaje Arduino: http://www.arduino.cc/en/pmwiki.php?n=Reference/HomePage

[8] Wikipedia. Series de Taylor: http://es.wikipedia.org/wiki/Serie_de_Taylor

[9] Wikipedi. Algoritmo de Bresenham: http://es.wikipedia.org/wiki/Algoritmo_de_Bresenham

[10] Robert A. Heinlein. Interprete básico: https://www.marginallyclever.com/blog/2013/08/how-to-build-an-2-axis-arduino-cnc-gcode-interpreter/

[11] Robert A. Heinlein. Interprete básico movimientos curvos: https://www.marginallyclever.com/blog/2014/03/how-to-improve-the-2-axis-cnc-gcode-interpreter-to-understand-arcs/

[12] PJRC. Librería TimerOne: https://www.pjrc.com/teensy/td_libs_TimerOne.html

[13] Jan Dolinay. Librería Fast I/0: http://www.codeproject.com/Articles/732646/Fast-digital-I-O-for-Arduino

[14] Prometec. Explicación Interrupciones: http://www.prometec.net/interrupciones/

[15] Prometec. Explicación Interrupciones Temporizadas: http://www.prometec.net/timers/

[16] FirePick. Maquina Pick and Place: http://delta.firepick.org/

[17] . Maquina Pick and Place: http://delta.firepick.org/

[18] FirePick. Maquina Pick and Place: http://delta.firepick.org/

[19] Atmel. Teoría aceleración motores paso a paso: http://www.atmel.com/images/doc8017.pdf

[20] Plastiv. Código aceleración de motores para Atmel: http://code.google.com/p/stepper-motor-controller/source/browse/trunk/+stepper-motor-controller+--username+plastiv%40gmail.com/

[21] GRBL. Interprete G-code Avanzado: https://github.com/grbl/grbl

[22] PJRC. Librería para aceleración de motores: https://www.pjrc.com/teensy/td_libs_AccelStepper.html

[23] Zeytah. Foro conexión driver con Arduino y motor paso a paso: http://cnccontrol.byethost13.com/smf/index.php?PHPSESSID=f8cad81704ab11c9835906985c437595&topic=154.0