UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Grado en Ingeniería Informática

Prototipo de trazador gráfico vertical mediante la utilización de algoritmos de optimización para el delineado.

Estudiante: FRANLY IRIS URBINA FRANCO

Dirigido por: FERNANDO LÓPEZ OSTENERO

Curso: 2021-2022

PROTOTIPO DE TRAZADOR GRÁFICO VERTICAL MEDIANTE LA UTILIZACIÓN DE ALGORITMOS DE OPTIMIZACIÓN PARA EL DELINEADO.

Proyecto de Fin de Grado en modalidad de oferta *específica*

Realizado por: Franly Iris Urbina Franco

Dirigido por: Fernando López Ostenero

Fecha de lectura y defensa: 6 de Octubre de 2022

Agradecimientos

A Luis Javier, porque sin su amor, su felicidad constante, su fortaleza y todo su conocimiento, probablemente yo no sería quien soy hoy y no hubiese llegado dónde he llegado.

A Fernando López por su dedicación, profesionalidad y porque siempre creyó en mí, incluso cuando yo no era capaz de hacerlo.

Al "Akelarre", por su paciencia y amistad incondicional… aunque, sobre todo, por la paciencia.

RESUMEN

Este documento, condensa el desarrollo de un proyecto realizado mediante la plataforma electrónica Arduino, el cual aunara las distintas fases del desarrollo del hardware, firmware y software de un plotter vertical, es decir, un tipo de dispositivo de impresión que, mediante la recepción de una imagen como entrada, genera un código con coordenadas e instrucciones para plasmar la imagen en una superficie vertical, haciendo uso de la gravedad como factor físico, dos motores rotatorios y un servo encargado de la presión de la pluma de impresión.

Se hará un repaso histórico sobre porque nació la necesidad de imprimir y plasmar nuestra información, cómo podemos crear nuestro propio dispositivo de impresión y todo el proceso necesario para que mediante una imagen generemos el código que hará que nuestro dispositivo cobre vida.

PALABRAS CLAVE

Ad-hoc

Arduino

avr-gcc

avr-lib

Bachinter

Binutils

Control decimal numérico

Efecto Raster

Firmware

G-Code

Hardware

Inkscape

Interfase

Makelangelo

Pixelación

Plugin

Plotter

Python

Servo

Steppers

Vectorización

ABSTRACT

This document condenses the development of a project using the Arduino electronic platform, which will combine the different phases of the development of hardware, firmware and software of a vertical plotter, that is, a type of printing device that, by receiving an image as input, generates a code with coordinates and instructions to capture the image on a vertical surface, making use of gravity as a physical factor, two rotary motors and a servo responsible for the pressure of the printing pen.

There will be a historical review of why the need to print and capture our information was born, how we can create our own printing device and the whole process necessary to generate the code that will make our device come to life through an image.

KEYWORDS

Ad-hoc

Arduino

avr-gcc

avr-lib

Bachinter

Binutils

Numerical decimal control

Raster effect

Firmware

G-Code

Hardware

Inkscape

Interphase

Makelangelo

Pixelation

Plugin

Plotter

Python

Servo

Steppers

Vectorization

Contenido

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1.
Introducción y Objetivos.

> ***Optimus, optima, optimum.***
>
> *Para encontrar el origen de este superlativo, debemos remontarnos a los pueblos indoeuropeos --tribus nómadas prehistóricas que partieron del Asia Menor hacia Europa y el Indostán entre mil quinientos y dos mil años antes de Cristo--, que usaban el vocablo op, con el significado de 'producir mucho, en abundancia, más de lo habitual'. Y por si esta abundancia no fuera suficiente, podían añadir a op el sufijo -tamo, dando lugar a op-tamo 'el que produce más'.*
>
> *Como algunos de esos pueblos se fueron a instalar en la Península Itálica, op-tamo acabó incorporado al latín bajo la forma optimus 'el mejor', que llegó hasta nosotros como* ***óptimo*** *[1].*

1.1. Introducción al proyecto

No es un misterio que el lenguaje a lo largo de millones de años de evolución nos ha permitido de linaje en linaje, de pueblo a pueblo y de nación a nación, transmitir nuestros conocimientos, nuestra cultura y avances para que perduren en la humanidad, haciendo que la evolución se considere, no solo parte de nuestra naturaleza, sino condición suficiente y necesaria de nuestra existencia.

Arqueólogos, psicólogos, lingüistas, antropólogos y más profesionales de nuestra era actual, creen situar correctamente el inicio de la comunicación verbal hace menos de 200.000 años conseguida por el homo sapiens [2]. Parece increíble que con un sistema prelingüística básico e impresionantes pinturas rupestres, hayamos pasado a los sistemas de escritura surgidos en la Edad de bronce, entre el 3.500 a.C y el 1.200 a.C. [3]. Durante este periodo de nuestra Prehistoria Reciente, gran parte de las formas de información, conocimiento y arte, eran

transcritos manualmente en pergaminos y superficies como se representa en la Figura 1.1, donde se dejó plasmado la necesidad de comunicarnos y transmitir, de ser nosotros mismos.

Figura 1.1: Detalle de una parte de las pinturas descubiertas en el yacimiento de Atapuerca.

Pero es tanto lo que como especie queremos transmitir y es tanta nuestra sed de conocimiento, que ni los romanos 400 años a. C. con sus moldes de arcilla, ni los chinos en el siglo XI con sus piezas de porcelana podían abastecer nuestra necesidad de comunicarnos, de transmitir. Hacia 1.459 d.C, los libros eran difundidos a través de copias manuscritas de monjes y frailes dedicados exclusivamente al rezo, y la copia de ejemplares por encargo del propio clero o reyes y nobles [4].

Optimus, optima, optimum.

Serían pues, estas palabras las que en 1.438 d.C rondaran los pensamientos de un orfebre alemán que dijo ser capaz de hacer a la vez varias copias de la Biblia en menos de la mitad de tiempo de lo que tardaba en copiar una el más rápido de todos los monjes copistas del mundo cristiano, y que esas

copias no se diferenciarían en absoluto de las manuscritas por ellos. Años de esfuerzo después, sin el reconocimiento que le correspondió y en total bancarrota, Johannes Gutenberg, cuyo retrato podemos apreciar en la Figura 1.2, dio vida a la primera imprenta, confeccionó moldes en madera de cada una de las letras del alfabeto y posteriormente rellenó los moldes con hierro, creando los primeros «tipos móviles». Tuvo que hacer varios modelos de las mismas letras para que coincidiesen todas con todas, en total más de 150 «tipos», imitando perfectamente la escritura de un manuscrito [5][6].

Figura 1.2: Retrato anónimo de Gutenberg datado de 1440.

Primeras imprentas

Su éxito no se hizo esperar y en 1.472, en Segovia, fue instalada la primera imprenta en España, cuya estructura podemos ver reflejada en la Figura 1.3. Anteriormente en toda Europa se habían copiado a mano, del orden de veinte mil libros y durante los 50 años siguientes se imprimieron un estimado de entre 12 y 20 millones de libros.

Esta invención marcó un antes y un después en nuestra historia, la rapidez en la copia de libros se realizó con una velocidad nunca vista y su impacto fue incalculable, la imprenta supuso la revolución más importante en contra de los poderes absolutos (monarquías e iglesia) ya que extendió el conocimiento, algo que

estos poderes guardaron para sí mismos durante los diez siglos que duró la Edad Media [7].

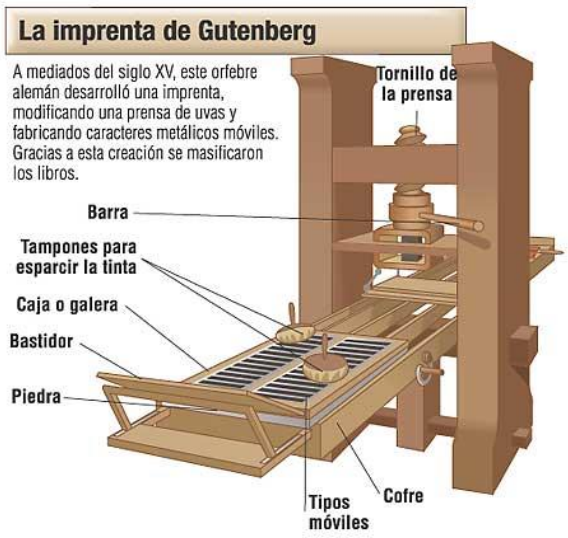


Figura 1.3: Imagen de la estructura física de La imprenta de Gutenberg.

Pocos inventos han tenido la influencia en el ser humano como la creación de la imprenta. A finales del siglo XIX, se perfeccionó el proceso y en 1.885 llegó en manos de Ottmar Mergenthaler la creación de la linotipia, comercializada para la industria de publicación de periódicos, revistas y carteles hasta las décadas de 1.960 y 1.970

Parte indiscutible de la naturaleza humana es la creación, y de lo creado lo mejorable, buscamos en cada una de nuestras obras y en sus procesos la optimización, ser eficientes y mejores. Es por ello por lo que Gutenberg vio la posibilidad de crear la imprenta hace casi 600 años y a su vez, tras años de crecimiento evolutivo, bajo este concepto de eficiencia hemos mejorado incontables tecnologías y creado tantas más.

Orígenes de la impresión

Los nuevos medios de comunicación aparecieron en un momento de cambio acelerado y de comunicaciones más veloces, estos fueron la respuesta a la mayor demanda de información y entretenimiento. Es difícil nombrar a una sola persona como creadora absoluta del artefacto que hoy en día consideramos una impresora, la idea fue evolucionando de la mano de varias personas.

No obstante, podemos decir que el punto de inflexión lo marcó Charles Babbage, quién concibió una máquina de propósito general, que podía ser programada por el usuario para ejecutar un repertorio de instrucciones en el orden deseado. El diseño de 1.837 de la denominada "Máquina Analítica" que se muestra en la Figura 1.4, era capaz de realizar operaciones aritméticas comunes mediante la utilización de tarjetas perforadas y programada en un lenguaje similar al Ensamblador. La Máquina Analítica estaba dotada de una impresora y una campana que anunciaba que el artefacto había terminado su trabajo. Charles Babbage fue considerado un genio y continuó con el refinamiento de su diseño hasta el fin de sus días en 1.871.

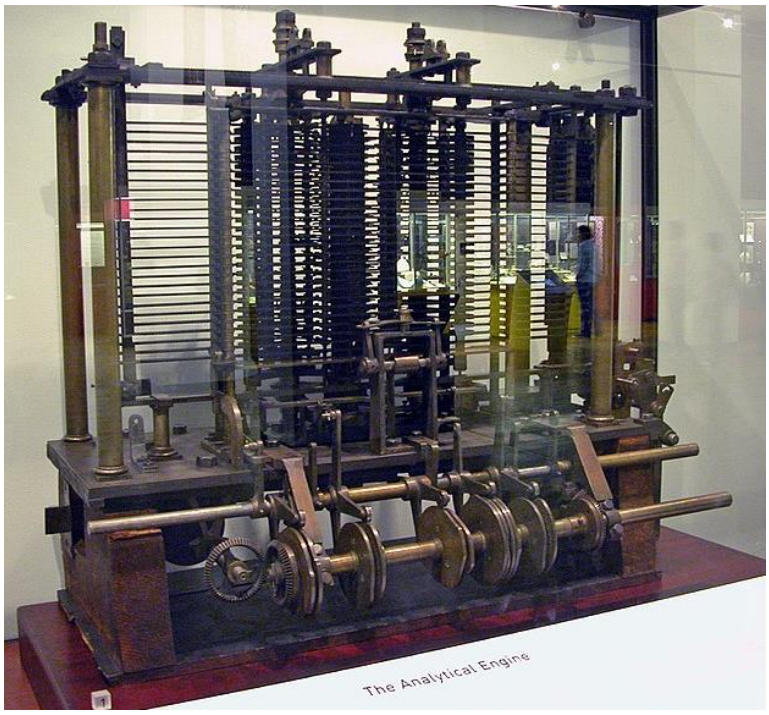


Figura 1.4: Maquina Analítica de Charles Babbage en el Museo de Ciencias de Londres.

A partir del descubrimiento de Charles Babbage, la tecnología de impresión comenzó a desarrollarse a pasos agigantados. En 1.855, David Edward Hughes inventó una especie de telégrafo capaz de transmitir datos mediante un código perforado. Se podría considerar como el primer antecedente de las impresoras modernas y disponía de un teclado con una disposición parecida un piano [8] tal y como se puede apreciar a continuación en la Figura 1.5.

Figura 1.5: Telégrafo impresor de Hughes fabricado por las empresas alemanas Siemens Halske.

En 1.953, Remington Rand crea UNIVAC High Speed Printer, cuya fotografía podemos apreciar en la Figura 1.6. Esta fue la primera impresora de alta velocidad para ser utilizada es un ordenador UNIVAC. Estaba compuesta por cuatro armarios: una fuente de alimentación, la máquina de impresión, un dispositivo de control y un lector de cinta y la cuál, era capaz de imprimir seiscientas líneas de texto por minuto.

Figura 1.6: UNIVAC High Speed Printer.

Cuatro años más tarde, IBM patenta la impresora de matriz de puntos. Que gracias a un cabezal de impresión podía deslizarle lateralmente por el papel. En 1959 inventan los equipos de impresión por líneas. Que tenían la gran ventaja de sobreimprimir, permitiendo una escala de grises y finalmente, en 1969 un ingeniero norteamericano llamado Gary Starkweather creó la primera impresora láser que 2 años después se hizo realidad en el centro de investigaciones Xerox. No obstante, no llegó al mercado hasta 1977 [9].

Exigencias del mercado

La distribución del conocimiento y de la información fue una de las tantas necesidades a cubrir con la invención de la impresión y junto a esto, nuevos desarrollos, arte y publicidad se añadieron al conjunto de demandantes de esta tecnología. Las dimensiones no han sido un impedimento y tan grande los objetivos como su evolución, la impresión de gran formato había llegado. Hablamos de imprimir materiales que son demasiado grandes para los productos de impresión de tamaños comerciales con lo que, hacía falta, el uso de un equipo que pueda acomodarse a formatos de impresión más grandes de lo habitual.

El área del diseño no fue un solicitante más, todo lo contrario, impresoras específicas llamadas plóteres, ploteadoras o trazadores gráficos como el mostrado en la Figura 1.7, fueron creadas para cubrir la necesidad de impresión de planos y posteriormente, gráficos vectoriales. Estos en lugar de usar tóneres, usan lápices, plumas u herramientas de escrituras para el dibujo de múltiples líneas continuas en el papel. La diferencia principal con la impresora normal es que las primeras utilizan líneas continuas para crear las imágenes, mientras que las impresoras utilizan conjuntos de puntos. Es aquí donde desarrollaremos el tema central del presente proyecto, un trabajo que demuestra en síntesis como años de creciente evolución se condensan en uno de tantos productos que han salido al mercado, contribuyendo con el avance comunicativo, social y tecnológico de nuestra era [10].

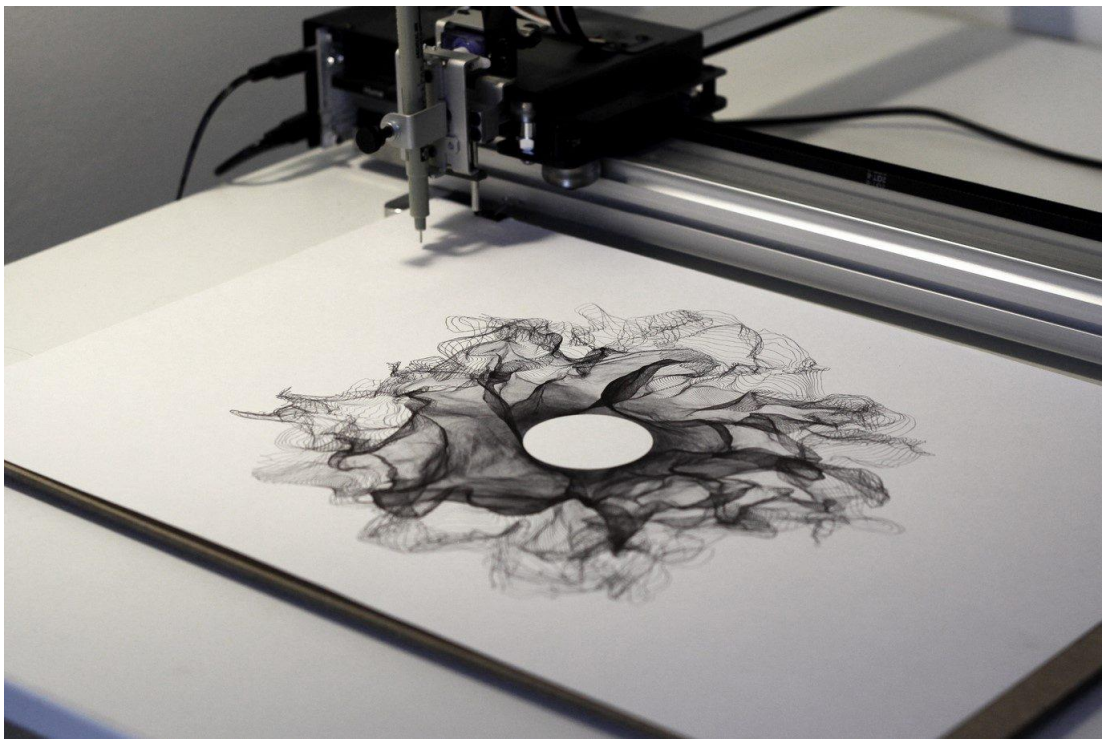


Figura 1.7: Imagen de un plotter realizando un diseño vectorial sobre papel.

1.2. Motivación y objetivos

Cómo hemos podido observar a lo largo de esta introducción, estos dispositivos tienen una trayectoria histórica motivada por una razón de ser: comunicarnos. Estas máquinas han sido fundamentales en el continuo desarrollo de la humanidad, nos han permitido propagar información, diseñar, crear nuevos objetos gracias a las actuales Impresoras 3D, Impresoras Textiles de sublimación e incluso, Impresoras de células y tejidos vivos capaces de contribuir en el área de la medicina y la farmacología.

Estos dispositivos nos han permitido materializar aquello que pensamos. Estamos acostumbrados que en el mundo de la Ingeniería Informática mucho del producto valor sea intangible, líneas de código con grandes contribuciones a la humanidad almacenadas como código binario, por lo que quiero ir un paso más allá con un proyecto el que la física, la informática y el diseño converjan.

Con este proyecto y en mi paso como estudiante de *Grado de Ingeniería Informática en la UNED,* deseo superarme y fusionar los conocimientos adquiridos tanto en el área software como hardware, inspirada en estos

dispositivos que tanta historia albergan, contribuyendo con un prototipo de trazador gráfico que pueda dibujar en superficies verticales las imágenes recibidas creando un código que permita mejorar y reducir los tiempos de impresión de algunos prototipos estudiados a lo largo de la realización de este proyecto.

Se destaca entre las asignaturas estudiadas durante el grado para la realización de este proyecto, las siguientes:

- Introducción a la Ingeniería de Software: Para la creación de los diagramas usados en el diseño.
- Diseño del Software: Aplicada a planificación y análisis del proyecto.
- Gestión de proyecto: Aplicada en la metodología de trabajo.
- Fundamentos Físicos de la Informática: Aplicada a la electrónica del dispositivo.
- Fundamentos de Sistemas Digitales e Ingeniería de Computadores I, II y III: Aplicada sobre el conocimiento de la lógica combinacional y circuitos.
- Teoría de los Lenguajes de Programación: Se aplica el conocimiento adquirido sobre el desarrollo del código.

Que, junto con el resto de las asignaturas estudiadas, condensan todo el conocimiento adquirido para la realización de este trabajo.

1.3. Organización de la memoria

La memoria se encuentra organizada en un total de 9 capítulos. Cada uno de estos posee una breve introducción del contenido y en su desarrollo, se destacan los aspectos más importantes estudiados y vividos en el desarrollo del proyecto, ilustrando cada fase y clasificándolas como Hardware, Firmware y Software. En cada uno de los capítulos se hallará también una breve conclusión que pretende sintetizar lo aprendido para conectarnos al siguiente, todo esto con el fin de enseñar a lo largo de esta memoria, cómo con dedicación, creatividad y los conocimientos adquiridos a través de la carrera universitaria ha conseguido que el desarrollo de este proyecto haya sido posible.

Capítulo 2.
Trazadores gráficos.

"El invento de la imprenta facilitó mucho el manejo de la información."

GEORGE ORWELL

En el capítulo anterior, hemos dado una pequeña introducción acerca de los plotters, dispositivos de salida especiales que se utilizan para producir copias impresas de los grandes gráficos y diseños en papel. A menudo se utilizan para la producción de mapas de construcción, diseños de ingeniería, planos arquitectónicos y gráficos de negocios, pudiendo ser un componente que se añade a un sistema informático o que pueden tener su propia computadora interna.

Su funcionamiento consiste en la interpretación de las órdenes de un ordenador para hacer dibujos vectoriales en papel mediante una pluma o lápiz, por lo que se conoce como impresión libre de impacto, ya que, en lugar de realizar golpes contra la superficie, realiza trazos deslizando un elemento cilíndrico sobre esta. A diferencia de una impresora normal, el plotter puede dibujar líneas continuas directamente a partir de gráficos vectoriales, mientras que las primeras imprimen mediante un conjunto de puntos como ya hemos comentado. Esto genera uno de los puntos débiles de los plotters a comparación de las impresoras, estos son mucho más lentos que porque los movimientos que llevan a cabo con su pluma son muy diferentes que la impresora puede realizar por medio de la impresión por puntos [11] [12].

2.1 Tipología

La diversidad de las necesidades ha dado pie a la creación de distintos tipos de trazadores dada la alta demanda de estos productos y sus objetivos en el mercado, encontramos entre los principales [13]:

2.1.1 Plotters de pluma

Este es el tipo de plotter en el que se centrará nuestro desarrollo, consiste en dispositivo de impresión con un número de cabezales que permiten manejar un número de plumas, bolígrafos, rotuladores o cualquier herramienta de escritura en la cercanía con la superficie de impresión, la cual puede ser regulada por el diseñador. Estos fueron los primeros plotters en salir al mercado y se utilizaban para dibujar planos en ramas destinadas a la arquitectura o topografía, pero hoy en día todavía se siguen utilizando en el diseño como se muestra en la Figura 2.1. Son más lentos, por lo que tardan más en realizar un trabajo, pero ofrecen gran calidad y suavidad en las curvas.

Figura 2.1: Prototipo básico de plotter de pluma.

2.1.2 Plotters de inyección de tinta

Este es uno de los plotters más extendidos. Se parecen a una impresora normal, pero de gran tamaño. Por lo general, su tamaño base es A1. Se podrían definir como una impresora de chorro de tinta, pero a gran formato. La diferencia

entre este tipo de plotter y el resto es que ofrecen impresiones de gran calidad y diversidad en cuanto a colores.

Se les denomina plotters porque son capaces de entender las instrucciones de lenguajes específicos de estos (RD-GL, HP-GL, DMPL, etc.), aunque internamente realizan una conversión de formato vectorial (líneas) a formato ráster (puntos de color). Pueden ser térmicos o piezoeléctricos según la tecnología que utilicen para aplicar las gotas de tinta.

Debido a sus similitudes, tanto la calidad como la velocidad de estos plotters es muy similar a las impresoras de tinta. No suelen fabricarse en tamaños menores de A1 ya que para tal finalidad ya existen las impresoras y podemos hacernos una idea de las dimensiones gracias a la Figura 2.2 a continuación:

Figura 2.2: HP DesignJet Z6100 - Impresora de inyección de tinta de gran formato.

2.1.3 Plotters electrostáticos, láser o térmicos

Su precio es significativamente más elevado que el resto de plotters del mercado dada la velocidad de trabajo, su alta capacidad para el funcionamiento durante horas y el escaso nivel de mantenimiento. Tienen un tamaño de punto menor poseyendo una buena calidad y resisten mejor a la luz y el paso del tiempo.

Su acabado recuerda a la impresión de un fax y no suelen imprimir a color como se puede observar en la Figura 2.3.

Figura 2.3: HP DesignJet T525 - Impresora de inyección de tinta térmica.

2.1.4 Plotters de corte o con corte.

Pudiendo ser manuales o automáticos, son dispositivos similares a los trazadores convencionales, pueden imprimir el diseño que se desee y un cortador (también llamado en cúter), separa el material de impresión utilizado del mantenido en bandeja.

En plotters de cortado dinámico, se intercambia la pluma por una cuchilla pudiendo crear diseños de corte, lo que hace que los acabados siempre sean mucho más profesionales al igual que su diseño tal y como podemos apreciar en la Figura 2.4. Algunos plotters de corte permiten regular la presión que ejerce la cuchilla, estos se utilizan para cortar vinilos, telas, carteles, señales personalizadas y muchos otros tipos de productos siendo utilizado por profesionales de la rotulación, existiendo modelos con la posibilidad de cortar materiales más gruesos como cartulinas o cartones. Su diferencia con el resto es que pueden ser de muchos

tamaños y tipos: de mesa o de rodillo, de corte tangencial, arrastre o cabezal excéntrico y funcionan tanto por arrastre como por fricción.

Figura 2.4: HP DesignJet T120 - Impresora de inyección de tinta térmica con corte.

2.2 Componentes Básicos

Las impresoras plotter son equipos sencillos con pocas piezas como podemos apreciar gracias a la Figura 2.5. Entre los componentes esenciales para su funcionamiento tenemos:

2.2.1 Las cubiertas

Es una parte importante se requieren para proteger todo el sistema de cableado interno que tiene el equipo, así como darle la mayor protección a toda la estructura del plotter. Dependiendo del diseño, también se encargará de resguardar los cartuchos de agentes externos como el polvo, pelusas y cualquier otro objeto y de proteger el mecanismo de movimiento cuando está funcionando.

2.2.2 El soporte móvil

Tiene la función que facilita transportar la máquina de un espacio a otro sin hacer mucho esfuerzo.

2.2.3 Cableado de datos

Se refiere a la parte vital de la máquina, hace que la computadora se comunique con la impresora para recibir la información del diseño que se desea imprimir.

2.2.4 Cable de alimentación

Este cable es el encargado de suministrar electricidad a la impresora para que pueda estar en funcionamiento.

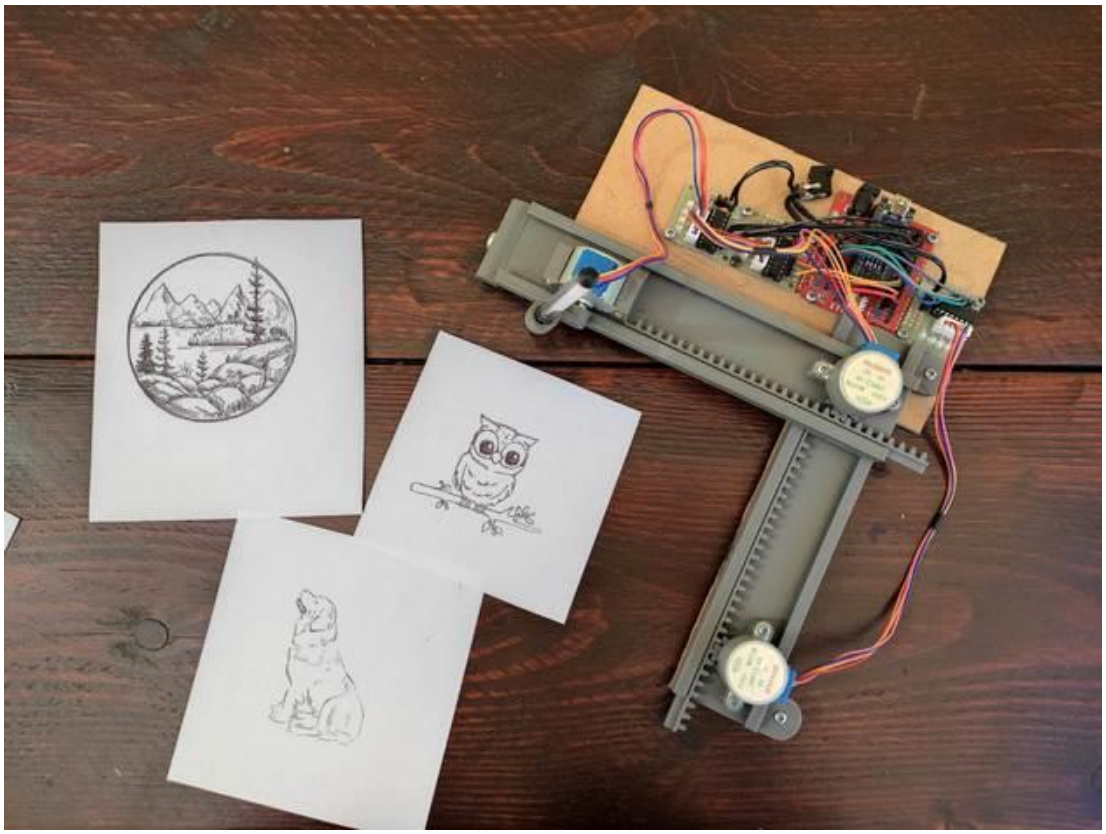


Figura 2.5: Plotter de pluma realizado en Arduino.

2.3 Componentes Extendidos

Definimos como componentes extendidos, aquellos que sirven para la construcción de un plóter y son utilizados hoy en día, más no son prescindibles para una realización funcional del mismo.

2.3.1 Bandeja de entrada.

Es un espacio de almacenamiento para los materiales de impresión, como los rollos de papel, el vinilo entre otros.

2.3.2 El panel LED

Dispositivo de salida dónde se visualizan los indicadores que muestran el estado de la impresora, tales estados pueden ser el de encendido, apagado, proceso actual, almacenaje o cualquiera que el desarrollador desee implementar y considere de utilidad.

2.3.3 Bandeja de salida

Almacenaje que se encarga de sustentar el material sobre el que se ha hecho la impresión una vez que el proceso ha terminado.

2.4 Funcionamiento

Conociendo los componentes que conforman la configuración física de los plotters, basaremos la explicación del funcionamiento en el prototipo para los plotters de pluma vistos en la sección 2.1.1.

Hablamos de sistemas de impresión por impacto, esto quiere decir, que disponemos de un mecanismo que activa y desactiva el contacto de la pluma con el material de impresión mediante la lectura de un código de coordenadas, el cuál dirigirá el plotter a las áreas de impresión junto con la indicación de activación y desactivación de la pluma.

G-Code

G-Code, lenguaje de programación G, o ISO-Code, es el lenguaje de programación más empleado en máquinas de control numérico (CNC) y el más usado para establecer la comunicación mediante la indicación de instrucciones de movimiento, velocidad y trayectoria para hardware en capacidad de desplazarse. Las máquinas típicas controladas por G-Code han sido las fresadoras, cortadoras, tornos e impresoras 3D ya que hoy en día es altamente usado para el diseño vectorial.

Un fichero en G-Code está formado por un conjunto de instrucciones sencillas que indican a una máquina las operaciones que debe realizar. Por ejemplo, desplazar una parte (cabezal, garras, topes), realizar un cambio de herramienta.

Cada línea (llamado bloque) de un programa escrito en G-Code tiene la siguiente forma mostrada en la Figura 2.6 [14]:

```
M05 S0
G90
G1Z0
G21
G1 F1500
G1  X111.8809 Y127.2887
G4 P0
M03 G1Z0 S30
G4 P0.20000000298023224
G1 F1000.000000
G1  X122.8423 Y127.1942
G1  X122.417 Y125.5878
G1  X121.4721 Y123.4617
G1  X120.2437 Y121.7135
G1  X118.8263 Y119.6819
G1  X117.1254 Y118.17
G1  X114.9048 Y116.5164
G1  X112.8259 Y115.3824
G1  X112.0699 Y115.1462
G1  X111.8337 Y115.1462
G1  X111.8809 Y127.2887
G4 P0
M05 S0
G1 F1500
G1  X108.8099 Y132.3914
G4 P0
M03 G1Z0 S30
G4 P0.20000000298023224
G1 F1000.000000
G1  X124.0234 Y132.2969
G1  X125.8661 Y131.4937
G1  X126.7638 Y130.8795
G1  X127.6142 Y129.6983
G1  X128.0867 Y128.2336
G1  X127.8977 Y126.391
G1  X127.1417 Y123.9341
G1  X125.9606 Y121.2411
G1  X124.1652 Y118.4062
G1  X121.3304 Y115.4297
```

Figura 2.6: Extracto de G-Code.

Cada bloque puede tener alguna o todas de esas direcciones, aunque de existir, el orden debe mantenerse. Además, se pueden poner comentarios empezando por ';' como se muestra en la Figura 2.7.

```
1  M05 S0 ;Esto es un comentario
2  G90
3  G1Z0
4  G21
5  G1 F1500
6  G1  X19.2396 Y201.1128
7  G4 P0
8  M03 G1Z0 S30
9  G4 P0.20000000298023224
10 G1 F1000.000000
```

Figura 2.7: Extracto de G-Code con ejemplo de comentario.

Los significados de cada una de estas letras es el siguiente

N	Significado
G	Motion
X	Horizontal distance
Y	Vertical distance
Z	Depth
F	Feed rate
S	Spindle speed
T	Tool selection
M	Miscellaneous functions
I and J	Incremental center of an arc
R	Radius of an arc

Tabla 2.1: Significado standard de cada letra en G-Code.

Cada una de estas letras están pensadas para controlar distintos tipos de máquinas, por lo cual contienen un conjunto de instrucciones muy amplio para poder controlar a todas ellas.

El plotter de pluma no deja de ser una máquina CNC. En este caso el firmware es el que se encarga de interpretar cada línea de G-Code y ejecutar las

acciones oportunas en el plotter, por lo que hablamos de cubrir las funcionalidades de dos partes importantes para el desarrollo del proyecto, el firmware que controlará el hardware y que como bien hemos dicho, es quién interpreta el G-Code y el programa encargado para generar el G-Code, del cual hablaremos en la sección 2.4.2 Plugin G-Code.

Lógicamente, no todas las instrucciones del estándar son de aplicación para el presente proyecto. Este es un listado con algunos de los códigos más importantes y que más frecuentemente encontraremos en el ámbito de impresoras FFF/FDM.

Código	Significado
M0	Parada
M1	Sleep
M2	Fin de programa
M70	Mostrar mensaje en pantalla
M104	Temperatura del extrusor
M106	Velocidad del ventilador
M107	Apagar ventilador
M140 y M190	Temperatura de la cama
M116	Esperar que temperaturas se estabilicen
M112	Parada de emergencia
G0	Movimiento rápido
G1	Movimiento controlado
G4	Pausa
G10	Retracción
G11	Desretracción
G20	Establecer unidades en pulgadas
G21	Establecer unidades en milímetros
G28	Mover al origen (Home)
G29	Autonivelado
G90	Posicionamiento absoluto
G91	Posicionamiento relativo
G92	Establecer posición

Tabla 2.2: Significado standard de códigos en G-Code.

A continuación, presentamos en la Figura 2.8 un ejemplo de fichero G-Code con su correspondiente significado [15].

```
G21 ;trabajar con milímetros
G90 ;usar posicionamiento absoluto
M82 ;colocar el extrusor en posicionamiento absoluto
M107 ;apagar ventilador
G28 X0 Y0 ;mover el extrusor a la posición 0,0 del plano
G28 Z0 ;bajar el extrusor hasta la posición 0 en altura
G92 E0 ;iniciar la extrusión a 0
G1 F200 E3 ;extruir 3 mm de filamento
G92 E0 ;resetear la extrusión a 0
G1 F9000 ;establecer velocidad a 9000 milímetros/minuto
M117 Imprimiendo… ;escribir mensaje en el LCD
G0 F9000 X58.972 Y85.198 Z0.300 ;posicionamiento rápido en
58.972,85.198,0.300
G1 F1200 X60.320 Y84.421 E0.02927 ;establecer velocidad a 1200
milímetros/minuto, posicionar en 60.320,84.421 y extruir a
0.02927
G1 X61.800 Y83.771 E0.05967 ;posicionar en 61.800,83.771 y
extruir a 0.05967
G1 X63.363 Y83.286 E0.09046 ;posicionar en 63.363,83.286 y
extruir a 0.09046
... ;código eliminado hasta primera capa para el ejemplo
M106 S127 ;encender ventilador a la mitad de potencia (127)
G0 F9000 X62.284 Y90.092 Z0.400 ;posicionar rápidamente en
62.284,90.092,0.400
G1 F540 X61.718 Y90.448 E4.19702 ;posicionar en 61.718,90.448 y
extruir a 4.19702
G1 X61.271 Y90.723 E4.20031 ;posicionar en 61.271,90.723 y
extruir a 4.20031
G1 X60.679 Y91.092 E4.20468 ;posicionar en 60.679,91.092 y
extruir a 4.20468
;codigo eliminado hasta final extrusión para el ejemplo
M104 S0 ;apagar extrusor
M140 S0 ;apagar la cama caliente
G1 E-1 F300 ;retraer filamento (para liberar presión)
G28 X0 Y0 ;mover a 0,0
M84 ;apagar motores
```

Figura 2.8: Extracto de G-Code con descripción de sus instrucciones.

Como podemos ver, el G-Code establece las directrices de movimiento y trazado que un mecanismo de impresión debe seguir para plasmar la imagen que recibe como entrada, por lo que su relevancia para determinar la optimización del recorrido de un dibujo es clave si queremos alcanzar los objetivos del proyecto, esta razón hace necesario hablar de cómo obtener este código y mediante qué mecanismos se hará. A continuación, hablaremos de los SVG como ficheros que habitualmente se utilizan como herramienta de entrada para la obtención del G-Code.

Scalable Vector Graphics

Los SVG se basan en un conjunto de gráficos vectoriales 2D con la capacidad de interactuar y ser animados, trazos con un punto de inicio y punto de fin escalables, esto quiere decir, un número inespecífico de puntos que lo conforma, haciendo que se puedan ampliar sin que se sufra perdida de información con el consecuente mantenimiento de calidad de imagen. Cuando se amplía una imagen en formato JPEG o BMP, encontraremos el efecto Raster o mejor conocido como «pixelación» de la imagen. En cambio, en una imagen vectorial los bordes de todos los objetos contenidos en la imagen permanecerán lisos y limpios. Estas diferencias las podemos apreciar en la Figura 2.9 junto a la imagen 2.10, que presenta un diseño realizado en trazos vectoriales.

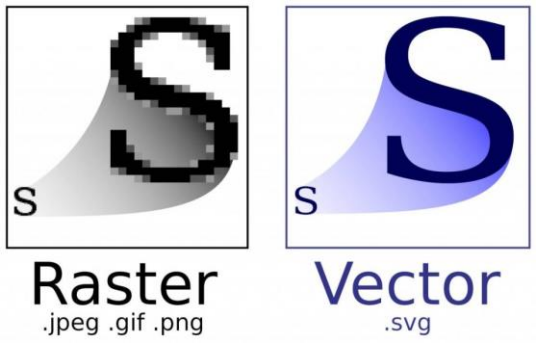


Figura 2.9: Diferencia entre una imagen tipo Raster y tipo Vector.

2.5 Conclusiones

Siendo nuestro objetivo la creación de un plotter vertical, es imprescindible exponer la teoría que conforma las definiciones alrededor del hardware y software para la creación de un prototipo, bases necesarias para la construcción de este. Se incluye la información relacionada con la transformación de un fichero de tipo .svg a G-Code, ya que daremos pie en el siguiente capítulo a la fase de análisis del proyecto, dónde descubriremos las herramientas que nos permiten hacer esta conversión e implementarla junto al resto de componentes con el fin de materializar nuestro proyecto.

Capítulo 3.
Análisis.

> *"Un marco común para los procesos del ciclo de vida de los programas informáticos, con una terminología bien definida, a la que pueda remitirse la industria del software. Contiene procesos, actividades y tareas aplicables durante la adquisición, el suministro, el desarrollo, el funcionamiento, el mantenimiento o la eliminación de sistemas, productos y servicios informáticos. Estos procesos del ciclo de vida se llevan a cabo mediante la participación de los interesados, con el objetivo final de lograr la satisfacción del cliente".*
>
> **ISO/IEC/IEEE 12207:2017**

Tal como lo son los SVG, disponemos de un punto de inicio, lo que queremos, y un punto final, dónde queremos llegar. Todo buen proyecto ha de constituirse siguiendo el denominado ciclo de vida del desarrollo, el cual, contempla las fases necesarias para validar el proyecto y garantizar así que este cumpla con los requisitos para la aplicación y verificación de los procedimientos de desarrollo, asegurándose de que los métodos usados son apropiados para alcanzar con éxito los objetivos de este.

Para alcanzar los objetivos sabemos que será necesaria la construcción del hardware, su firmware y necesitaremos disponer de un software de terceros en la generación del G-Code, necesario para alimentar al hardware de las instrucciones que le guiarán en el proceso de trazado.

En capítulos anteriores hemos definido los antecedentes del proyecto, los cuales nos servirán para gestionar el primero de los pasos en el ciclo de vida del software, El Análisis.

3.1 Hardware

Para los componentes físicos, se han estudiado distintas formas de crear el hardware. Se han analizado otros proyectos similares para tener una idea de

cómo iniciar el proyecto, tal es el caso del proyecto ElectroGeek, donde ofrecen una guía para "Transformar tu pizarra sin usar en un plotter vertical" [23]

En este dispositivo, comentan que usan un Arduino Mega con el firmware de Makelangelo, el cual descargamos y analizamos para estudiar sus características.

En este análisis vemos que no es el único proyecto en el que eligieron Arduino para la construcción del plotter, de hecho, la totalidad de los proyectos analizados se basaron en Arduino para la construcción del plotter, por lo que centré mi análisis en buscar si existían alternativas y por qué no eran usadas.

Básicamente, Arduino es un microcontrolador con el que podemos realizar determinadas funciones y se encuentra entre las plataformas más comunes para la creación de proyecto, como lo son NodeMCU y Raspberry PI. Este último es un microordenador, por lo que su potencia y complejidad son mayores. Aunque podemos trabajar con cualquiera de esas placas, Arduino está especialmente pensada para la creación de proyectos que involucren la electrónica de tareas sencillas que no requieran tareas complejas, además, por medio de la conexión sin necesidad de teclado o ratón, la placa lanzará el firmware guardado, automatizando las tareas de compilación, conexión y sincronización de los dispositivos. Esta es la principal razón por la que Arduino se utilice para este tipo de proyectos, sencillo, rápido y con programación mínima, mientras que con Rasppberry podríamos hacer lo mismo, complicando más la tarea.

Entre las diversas razones dada anteriormente, resulta importante destacar la accesibilidad a los componentes en el mercado, su código abierto, la popularidad y estandarización que nos provee de documentación suficiente para realizar desarrollo, además de su fácil utilización y simplicidad [24].

Componentes

Entre los diversos proyectos que encontramos, destaca **Plotter Vertical Con Arduino** de **MCI electronics** [25]. Este proyecto detalla los pasos para la construcción de un Plotter vertical, desde la creación de las piezas hasta su funcionamiento final y de este diseño podemos ver que entre las piezas claves para la generación de nuestro plotter destacamos la utilización de dos motores paso a paso, un servo, las piezas de cableado y una placa base. Para mantener las piezas unidas y que den lugar al funcionamiento del proyecto, es indispensable el diseño de piezas ad-hoc que consigan la sujeción de la pluma, bolígrafo o rotulador (recordemos que el diseño del plotter será un plotter de pluma), la base del servo

que controle el levantamiento de la pluma, los motores que controlarán la dirección de la pluma y la placa central desde dónde se orquestarán las instrucciones

En la Tabla 3.1 podremos ver una aproximación de los gastos usando esta plataforma, los cuales se compararán con el resultado final de los costes una vez implementado el proyecto, aunque también se añade después de solicitar el presupuesto a una empresa dedicada al diseño e impresión, el precio de las piezas 3D si entregamos el diseño de lo que queremos. El presupuesto total corresponde a los cálculos realizados a fecha de enero de 2022.

Material	**Cantidad**	**Precio Unitario (€)**	**Precio Total (€)**	**Referencia a Julio 2022**
ARDUINO MEGA 2560 R3 ATMEGA2560 COMPATIBLE	**1**	22,95	22,95	https://www.tiendatec.es/maker-zone/placas-arduino/420-arduino-mega-2560-r3-atmega2560-compatible-cable-usb-b-gratis-8404201160013.html
Sensor Shield v5.0 Arduino	**1**	3,00	3,00	https://leantec.es/tienda/sensor-shield-v5-v50-para-arduino-bluetooth-module-servo-uno-mega/
Motor paso a paso 28BYJ-48 + Modulo Driver ULN2003	**2**	3,49	6,98	https://www.tiendatec.es/maker-zone/servos/634-modulo-driver-ul2003-motor-stepper-28byj-48-para-arduino-8406341540007.html
Modulo Adaptador MicroSD	**1**	2,16	2,16	https://www.tiendatec.es/maker-zone/modulos/631-modulo-adaptador-microsd-para-arduino-8406311180004.html
MicroServo 9G SG90	**1**	3,45	3,45	https://www.tiendatec.es/electronica/servos/583-microservo-sg90-8405830020006.html
PACK 100 CABLES DUPONT 20CM M/M	**1**	7,95	7,95	https://www.tiendatec.es/electronica/placas-de-prototipo/cables-de-puentear/280-pack-100-cables-machomacho-para-puentear-en-placa-de-prototipo-arduino-8402800400011.html
PACK 100 CABLES	**1**	7,95	7,95	https://www.tiendatec.es/electronica/placas-de-prototipo/cables-de-puentear/577-pack-

DUPONT 20CM H/H				100-cables-dupont-20cm-hh-para-puentear-en-placa-de-prototipo-arduino-8405770400012.html
Pizarra para rotulador 90 cm x 60 cm repisa madera	**1**	13,55	13,55	https://www.soportesmagneticos.com/pizarras/1-Pizarra-para-rotulador-90-cm-x-60-cm-repisa-madera-3704.html
Pack4 Rotuladores Staedtler pizarra	**1**	7,95	7,95	https://www.abacus.coop/es/rotuladores-staedtler-pizarra-blanca-4-u/1070341.47.html
15 tornillos	**1**	1,47	1,47	https://alitools.io/es/showcase/tornillo-de-cabeza-plana-ultradelgada-para-ordenador-portatil-m5-de-acero-inoxidable-5-50-m1-6-m2-m2-5-m3-m4-m6-negro-phillips-304-uds-4001248931159
Piezas 3D	**3**	30	90	-
TOTAL	-	-	**167,41**	

Tabla 3.1: Coste orientativo en materiales.

3.2 Firmware

Habiendo elegido Arduino como plataforma de implementación, el lenguaje de programación para la comunicación con el hardware es el propio GCC, un lenguaje derivado del C++ que proviene de la adaptación de avr-lib, una librería de C de alta calidad para los microcontroladores. GCC es un conjunto de compiladores que se considera el estándar para los Sistemas Operativos derivados de UNIX y requiere de un conjunto de aplicaciones conocidas como binutils que son unas herramientas de programación para la manipulación de código de objeto (código binario o máquina para que el microcontrolador pueda leer las instrucciones). Está creado para ejecutarse en un sistema como Linux, Windows o macOS y en el proceso, generar código para un microcontrolador AVR, al que se le denomina avr-gcc. Este es el compilador que usa el IDE de Arduino para convertir el código de C++ a un fichero binario (.hex) que se cargará en la memoria flash de la unidad microcontroladora (MCU) que ejecuta las instrucciones. Como

hemos visto en el apartado anterior, la elección de este lenguaje se debe a la elección de la plataforma Arduino para implementar el hardware, destacando la facilidad para generar el código y gracias a las extensas librerías de la que nos beneficiaríamos en la creación del proyecto, ya que nos permitirá simplificar la comunicación con los diferentes componentes hardware [18].

Las órdenes del servo serán sencillas, si queremos activarlo (escribir) el servo se acercará a la pizarra para que no haya distancia entre la pluma y la superficie acrílica. Por el contrario, si queremos dejar de graficar, haremos que el servo rote el cabezal para que la pluma se distancie de la superficie y no marque durante el desplazamiento al siguiente punto de trazado.

Para el movimiento de los motores, el análisis es más complejo. Veremos si hay una distancia entre la posición anterior y la actual mayor a 0, de ser así nos encontraremos en el cuadrante positivo del plano y el motor derecho girará en sentido antihorario para recoger el hilo y subir el plotter.

Este procedimiento es el mismo para cualquier punto de la pizarra e inverso para el motor izquierdo. Si quisiéramos subir el plotter al lado superior izquierdo de la pizarra, el motor izquierdo giraría en sentido horario para poder recoger el hilo. Si quisiéramos una línea recta hacia arriba, ambos motores girarían recogiendo el hilo y estirándolo como mostraremos en la Tabla 3.2 a continuación. Estas premisas se verán reflejadas en el código y en caso de que la distancia entre una coordenada y otra sea muy grande, se dividirá en tantos fragmentos como unidades de medida pueda trazarse en la pizarra en función al diámetro de la polea que se utilice en los motores.

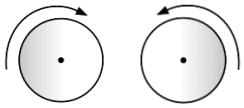
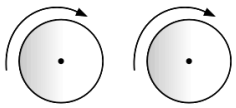
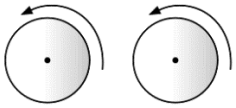
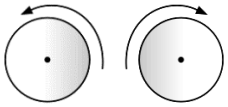
Plano Y	Plano X	Motor Izquierdo	Motor Derecho	Imagen de los motores
+	+	horario	antihorario	
+	-	horario	horario	
-	+	antihorario	horario	
-	-	antihorario	horario	

Tabla 3.2: Análisis del movimiento de los motores para mover el rotulador.

3.3 Software

Para esta fase será necesario contemplar cómo se crean los G-Code y analizar cuál es la manera óptima de conseguir un código de ejecución de instrucciones con el mínimo recorrido y el mayor número de trazado. Aunque en los proyectos anteriores utilizan un software llamado Makelangelo, analizamos la versión 0.9.2 disponible y se puede apreciar una alta complejidad debido al número de ficheros tan elevado de clases en C++ que contienen el código para generar el G-Code.

Durante esta fase, analizamos alternativas y descubrimos que el programa Inkscape dispone de facilidades, en forma de plugins, que nos pueden generar el código que queremos si damos como entrada una imagen vectorial.

Inkscape

Inkscape es un software libre y gratuito para la edición de gráficos vectoriales multiplataforma, actualmente disponible para sistemas operativos como Windows, iOS y Linux que nos permitirá tener una base para el desarrollo de nuestro G-Code. El uso de extensiones y las licencias GNU con la que se manejan la mayoría, nos permite tener acceso a librerías de gran utilidad para

trabajar con nuestros diseños y conseguir nuestro objetivo, concretamente, una librería en Python 2 de licencia libre mostrada en la Figura 3.1 llamada Bachinter [16].

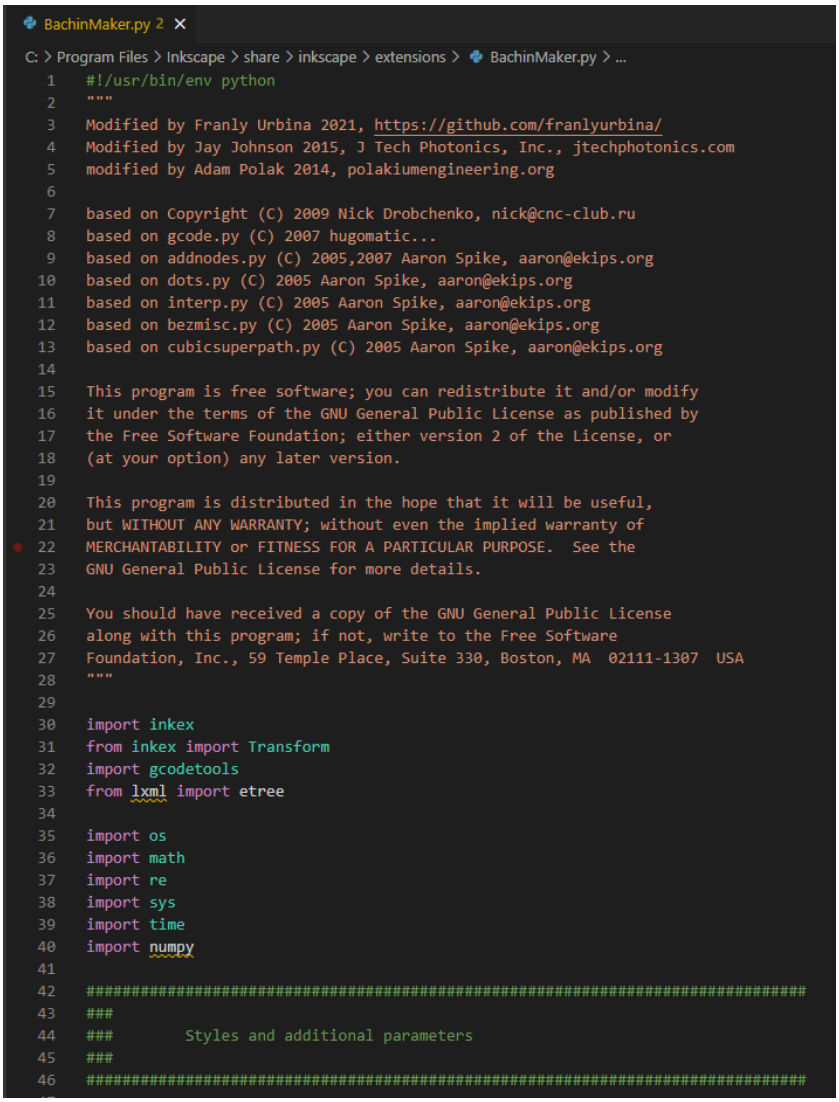


Figura 3.1: Captura de los comentarios de inicio y licencias del plugin Bachinter.

Siguiendo la documentación, todos los datos de configuración para la generación del G-Code, se piden a través de varias ventanas de configuración que se definen por medio de un archivo XML que es tratado por la extensión del programa para asignar los valores de los puntos de coordenadas de inicio y fin o tamaño de la imagen de un fichero SVG como el mostrado en la Figura 3.2, tratando los trazos como funciones en un plano ortonormal, abstrayendo la dimensionalidad del dibujo al ser una imagen vectorial [17].

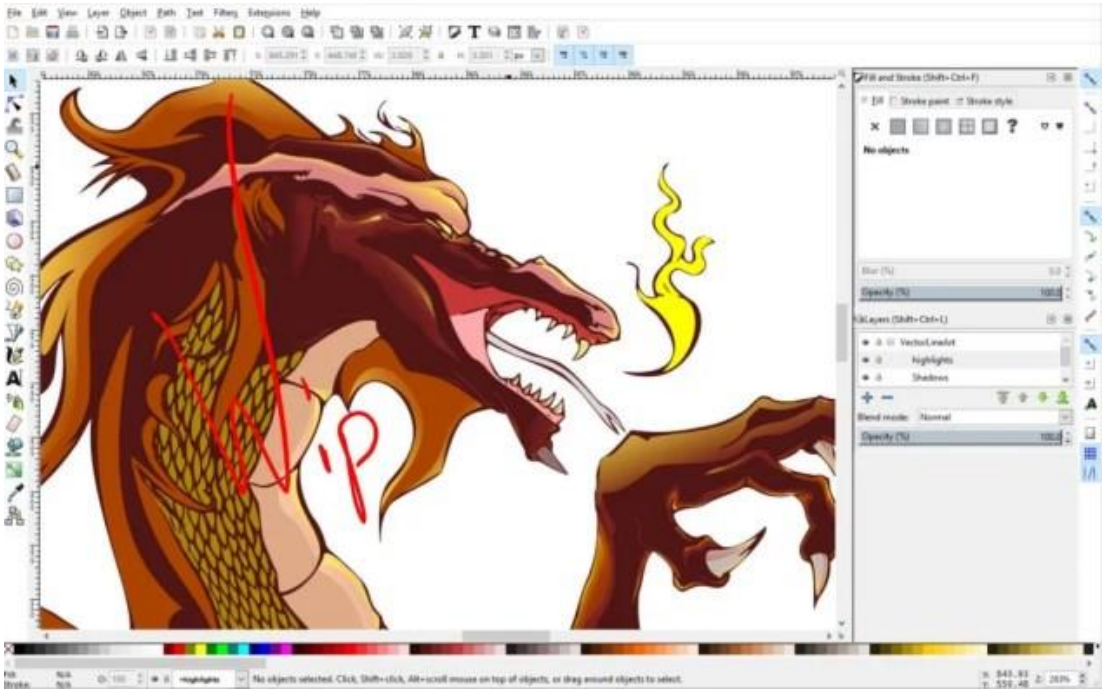


Figura 3.2: Diseño vectorial de un artista gráfico.

3.4 Conclusiones

Hemos estudiado los componentes hardware disponibles, elegido Arduino como plataforma para la construcción del plotter y realizado un presupuesto inicial. También se han visto las diferentes librerías disponibles que posee para facilitarnos la comunicación con los componentes, como anclarlos y también que software nos facilitaría la generación del G-Code que valdrá como entrada para el trazado de la imagen.

Habiendo estudiado la información disponible anteriormente y eligiendo las tecnologías consideradas más ventajosas para la realización del proyecto, en el siguiente capítulo generaremos el diseño para implementar nuestro plotter según los componentes hardware, el firmware y el software analizados, dando paso a los esquemas que nos ayudarán al desarrollo de este.

Capítulo 4.
Diseño.

"Si puedes diseñar una cosa, entonces puedes diseñarlo todo; si lo haces bien, perdurará para siempre"

MASSIMO VIGNELLI

Con el análisis de los componentes y habiendo estudiado las herramientas que queremos aplicar a nuestro proyecto, se procede a establecer los pasos a seguir para conseguir la construcción del proyecto.

4.1 Hardware

Como hemos estudiado en el capítulo anterior, existen distintos proyectos publicados en Arduino dónde la idea base para realizar un plotter que dibuje verticalmente se centra en la utilización de dos motores paso a paso o steppers, un servo, la placa base y una memoria que contendrá la imagen que queremos imprimir como lo es la unidad SD para la lectura de los ficheros. Los componentes hardware se conectarán mediante los pines al microcontrolador, el cual seguirá las instrucciones del firmware, todas estas sujetas mediante piezas ad-hoc hechas con impresora 3D a la pizarra del proyecto.

Empezamos contando con que serán necesarios dos soportes para los motores que guiarán al rotulador dentro de las dimensiones del plano de dibujo, los cuales atornillaremos a la madera de la pizarra. Se toman medidas y se crea el diseño mostrado en la Figura 4.1

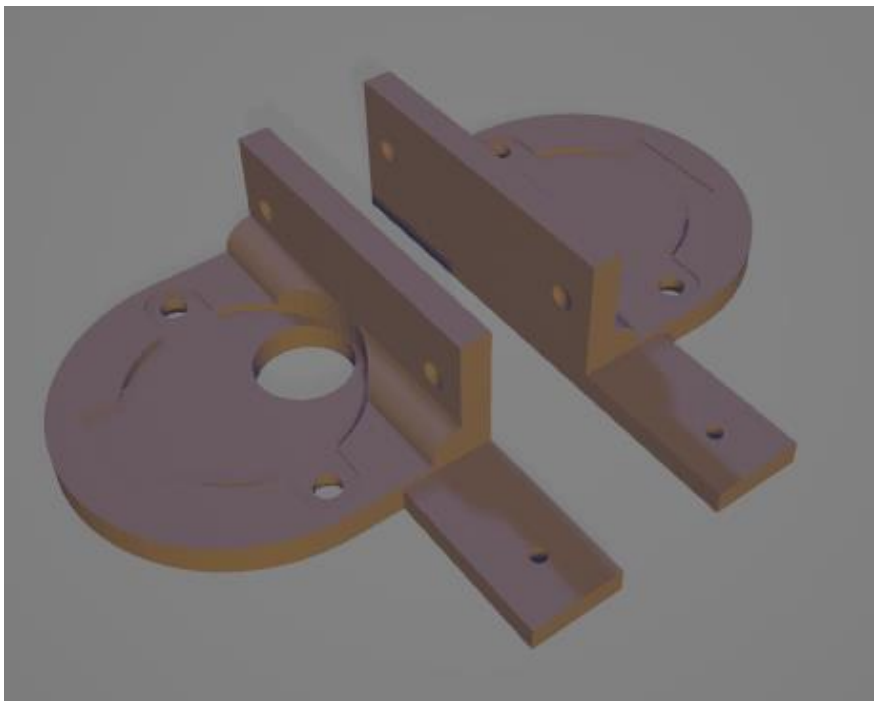


Figura 4.1: Diseño del soporte de los motores paso a paso.

Estos motores tendrán dos poleas, una primera polea de 3.5mm de diámetro mostrada en la Figura 4.2 dónde estará el hilo enrollado y que se irá liberando poco a poco según las coordenadas donde queramos apuntar el rotulador y una segunda polea mostrada en la Figura 4.3 que ayudará a que el hilo mantenga una distancia de 550mm de separación, medida que utilizaremos en el código como guía del eje x en el plano ortonormal.

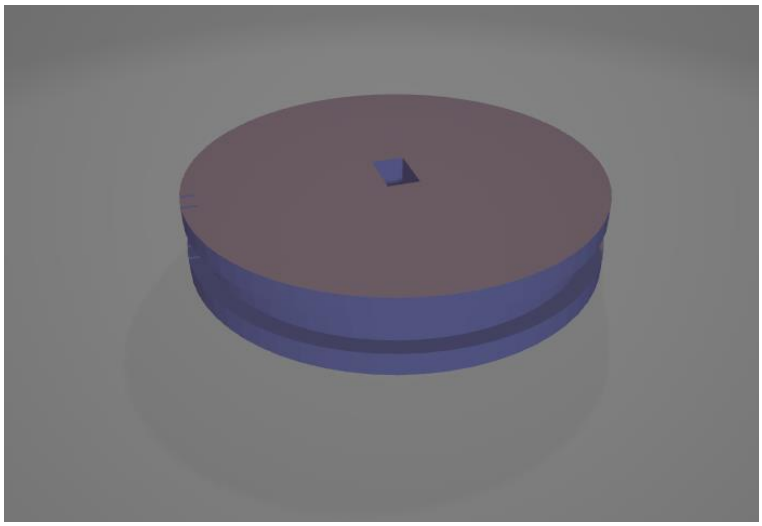


Figura 4.2: Diseño de polea de 3,5 cm.

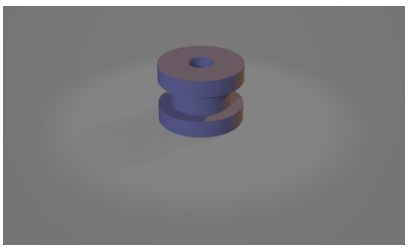


Figura 4.3: Diseño de polea de 1,0 cm.

La Figura 4.4 muestra la base para sujetar el rotulador con los hilos y que dispondrá de una cavidad para el rotulador y otra para sujetar el servo y que se pueda marcar distancia entre la pizarra y la punta del rotulador. A su vez, por diseño, tiene incorporado dos aberturas para añadir peso y evitar movimientos bruscos cuando los hilos se muevan.

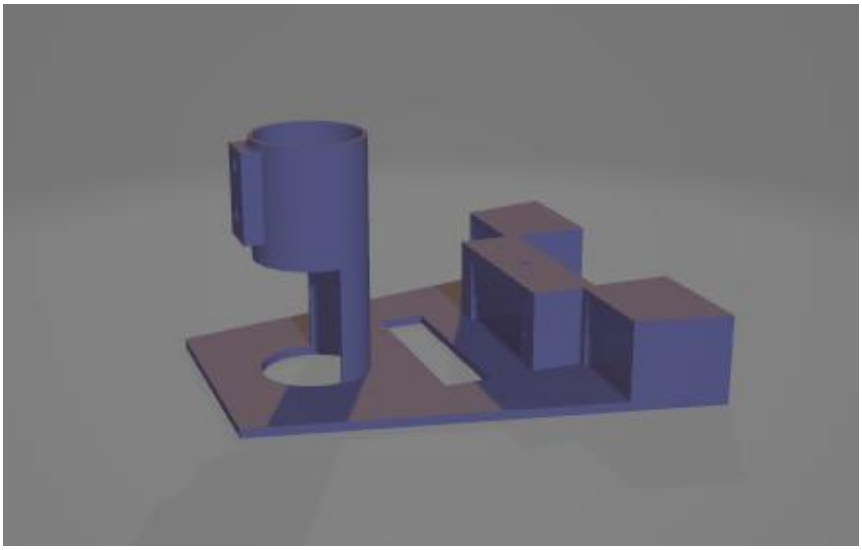


Figura 4.4: Diseño de soporte de rotulador.

Los motores serán ensamblados tal y como muestra la Figura 4.5 y sujetadas con las piezas 3D que hemos visto anteriormente, en el caso de los motores paso-a-paso, la configuración para el motor izquierdo concordará con los pines 2, 3, 5, 6 y el derecho, los pines 7, 8, 9 y 10.

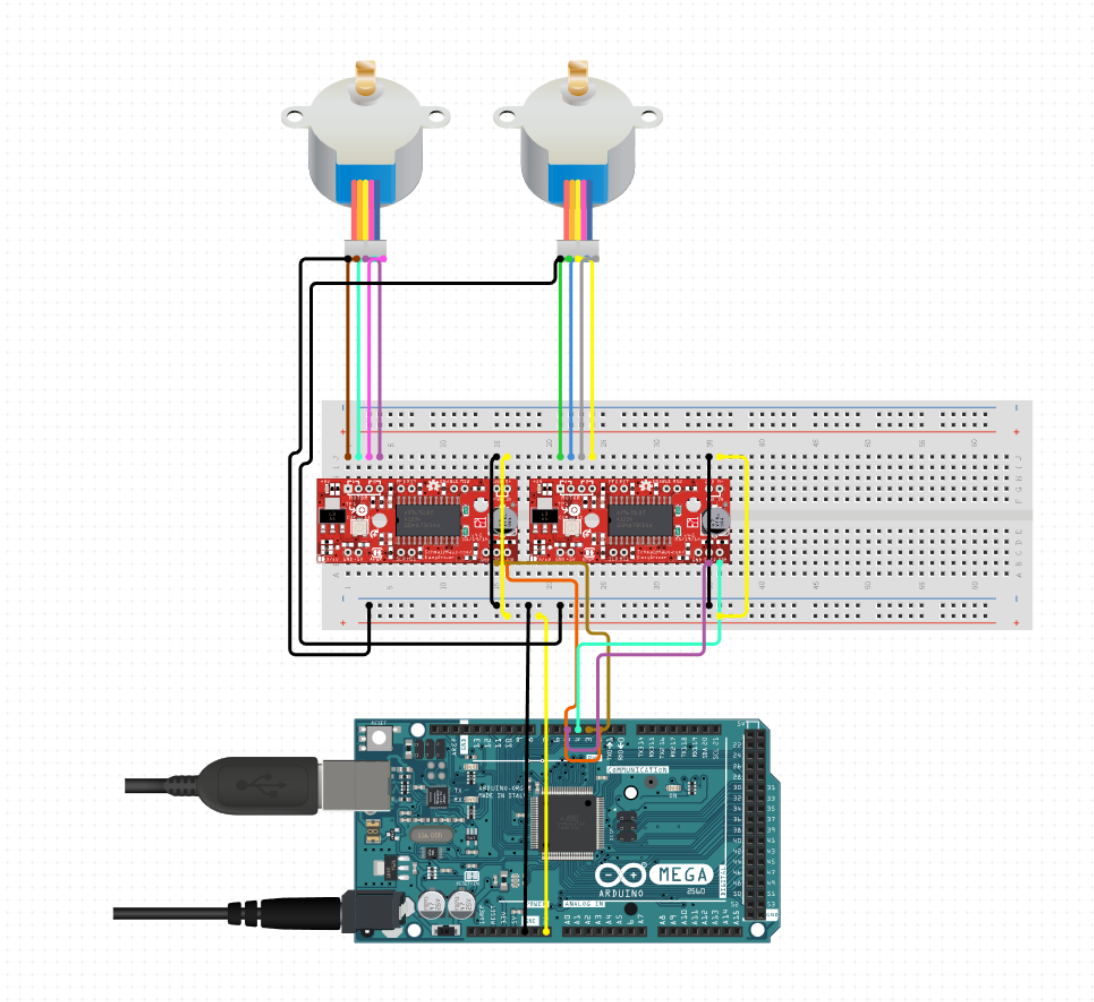


Figura 4.5: Diseño de conexión motor – driver – placa, generado en circuito.io [19].

La tarjeta SD estará colocada en la disposición mostrada a continuación y tal y como muestra la Figura 4.6:

- CS - 53
- SCK - 52
- MOSI - 51
- MISO - 50
- VCC - SD+
- GND - RESET

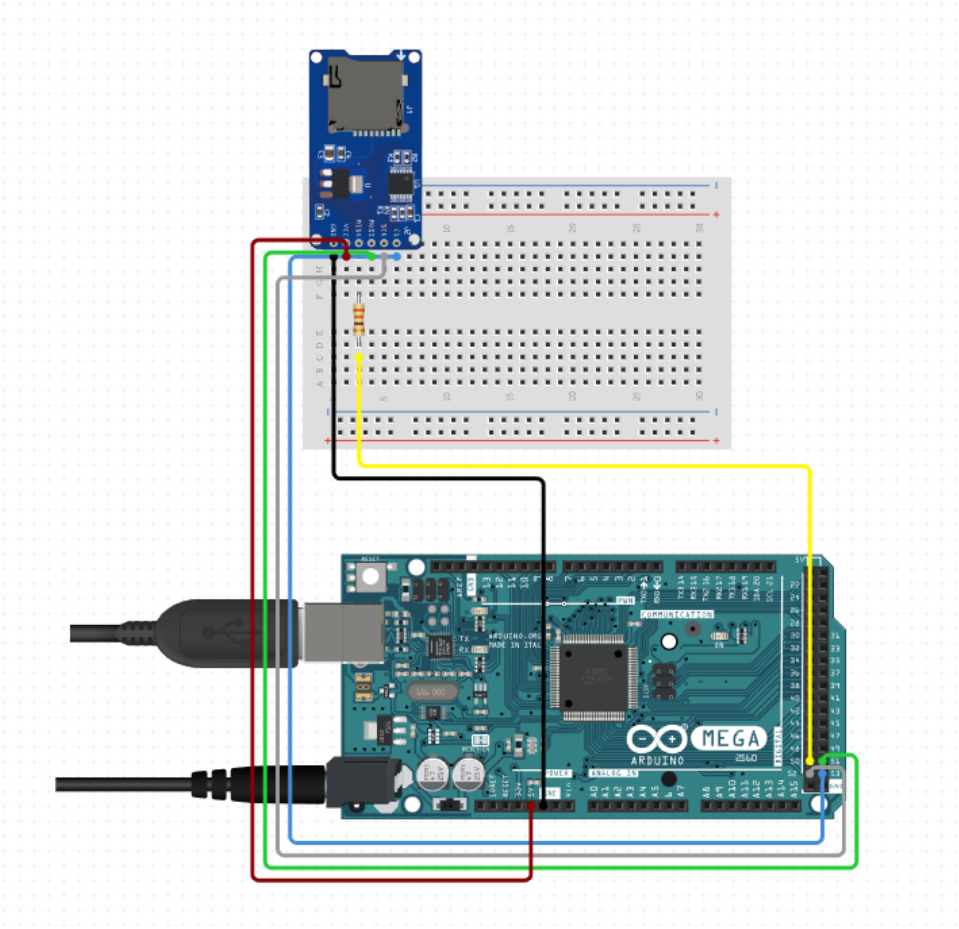


Figura 4.6: Diseño de conexión tarjeta SD – placa, generado en circuito.io [20].

El servo estará colocado en la disposición A0 de la Sensor Shield v5.0 y esta a su vez, estará sobre la placa de Arduino como se muestra en la Figura 4.7

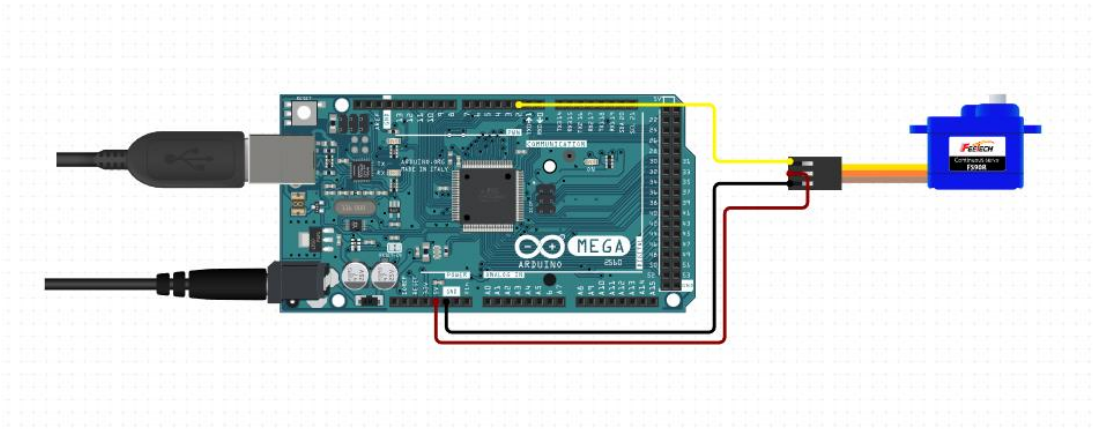


Figura 4.7: Diseño de conexión servo – placa, generado en circuito.io [21].

Conectados todos los componentes, nos queda la siguiente implementación mostrada en la Figura 4.8, cuya fuente de alimentación será la conexión mediante puerto USB al dispositivo que contenga la extensión Bachinter de Inkscape.

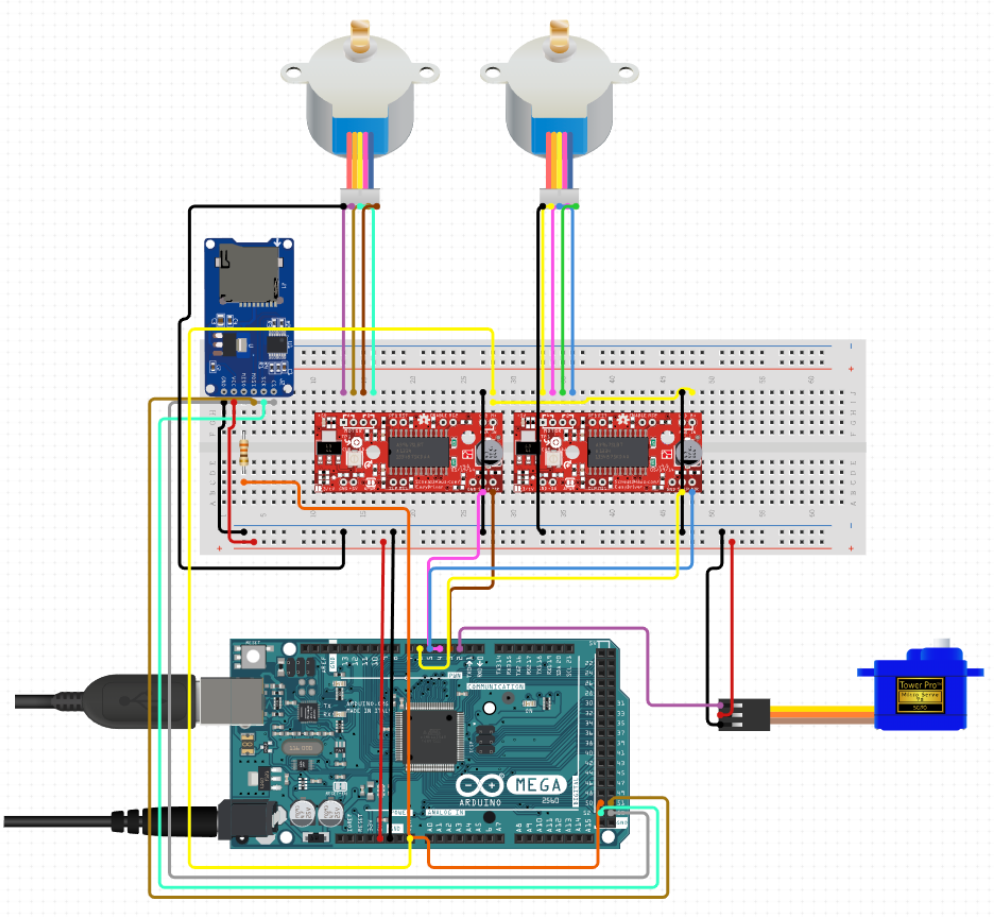


Figura 4.8: Diseño de conexión de componentes generado en circuito.io [22].

Estos componentes por si solos no darán vida al plotter hasta que se ensamblen con las piezas realizadas mediante impresión 3D, las cuales mantendrán la unión del hardware a la pizarra y dónde el rotulador dibujará las imágenes leídas en G-Code mediante el movimiento de los hilos atados a las poleas.

4.2 Firmware

El firmware tendrá por nombre vectorial_draw. El objetivo será leer el fichero que contenga el G-Code y en base a las coordenadas de dirección, girar el eje de los motores paso-a-paso tantas unidades como coordenadas se quieran recorrer. Estas coordenadas no deben ser impresas en su totalidad, ya reflejan en su totalidad la imagen que se desea imprimir más el desplazamiento que debe realizar el plotter hasta la siguiente línea de impresión. Dichas líneas de traslado se deben descartar y aquí entra en juego nuestro servo al cual, se le darán instrucciones de "escribe" y "no escribe" dentro el G-Code. Para esto nos ayudaremos de las librerías Servo.h, SD.h y TinyStepper_28BYJ_48.h, las cuales, nos abstraerán del comportamiento de los componentes y nos ayudarán a simplificar la lógica permitiéndonos enviar las órdenes y validarlas tal y como se muestra en la Figura 4.9 correspondiente a un diagrama de flujo para el seguimiento de instrucciones.

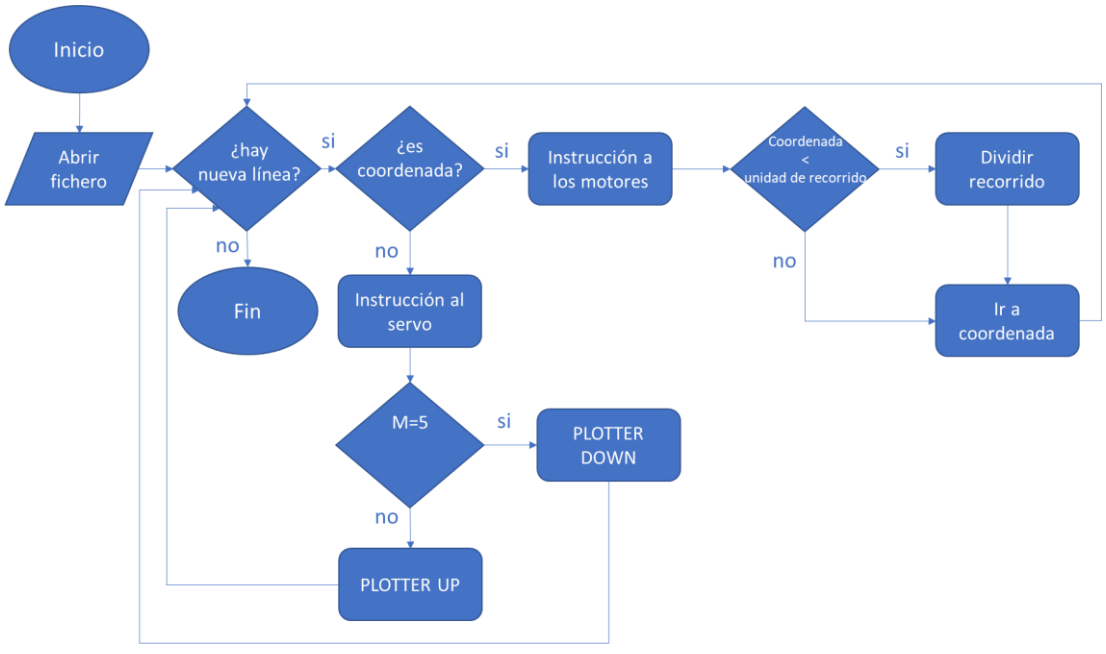


Figura 4.9: Diagrama de flujo del código GCC.

4.3 Software

Se realizó una investigación en la fase de análisis que dio lugar a valorar la extensión Bachinter de Inkscape para la vectorización de imágenes y transformación a G-Code. Lamentablemente esta extensión fue desarrollada utilizando Python 2 y las incompatibilidades del lenguaje hicieron imposible la utilización e instalación de la extensión, por lo que la solución más factible será diseñar un plan de migración del código del software de Python 2 a Python 3, refactorizando el código y adaptándolo a los nuevos estándares y versiones disponibles.

4.4 Conclusiones

Se ha realizado mediante el uso de herramientas de diseño, la simulación de ensamblado de las piezas para conocer las conexiones y tener una visión más clara de los componentes que se necesitan para su ensamblado. A su vez, se han diseñado las piezas que se imprimirán en 3D a medida para el proyecto y su sujeción junto con el diagrama de flujo que deberá seguir el firmware y los pasos indispensables para hacer funcionar el software que elegimos en el Capítulo 3. Si se ha realizado correctamente, pasaremos a la fase de implementación en el siguiente capitulo dónde se reportarán los tiempos realizados y los posibles inconvenientes encontrados.

Capítulo 5.
Implementación.

"Una buena ingeniería de software requiere la diferenciación entre la especificación y la implementación."

ANDREW TANENBAUM

Una vez analizado los diferentes objetivos, ensamblamos las piezas del apartado anterior siguiendo la hoja de características técnicas para el montaje en Arduino para cada uno de los componentes.

5.1 Hardware

El ensamblado empezará por los motores paso a paso, los cuales moverán una polea principal que recogerá el hilo que sujeta el soporte del rotulador. Dichos motores se unirán mediante tornillos y tuercas al soporte y este irá sujeto con la ayuda de dos tirafondos a la madera de la pizarra tal y como muestra la siguiente imagen de la Figura 5.1. Cabe destacar que la polea secundaria más pequeña, se creó con la finalidad de evitar movimientos bruscos en el hilo.

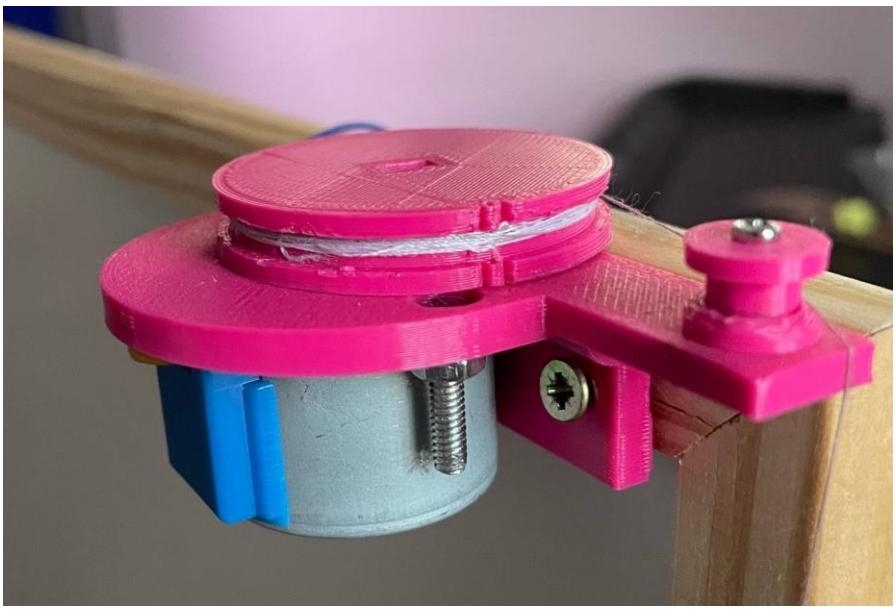


Figura 5.1: Diseño armado del motor paso a paso y las piezas 3D a la pizarra de dibujo.

El soporte del rotulador quedará suspendido por el hilo que mueven los motores paso a paso y llevará consigo el servo que recibirá las ordenes de cuando hay que levantar el rotulador y cuando mantenerlo al nivel de la pizarra. Como se puede observar en la Figura 5.2, en el diseño de este soporte hay dos espacios destinados a añadir peso para mantener el soporte lo más firme y pegado a la pizarra y una tuerca que mantiene fijo al rotulador para evitar su caída durante el proceso de dibujo.

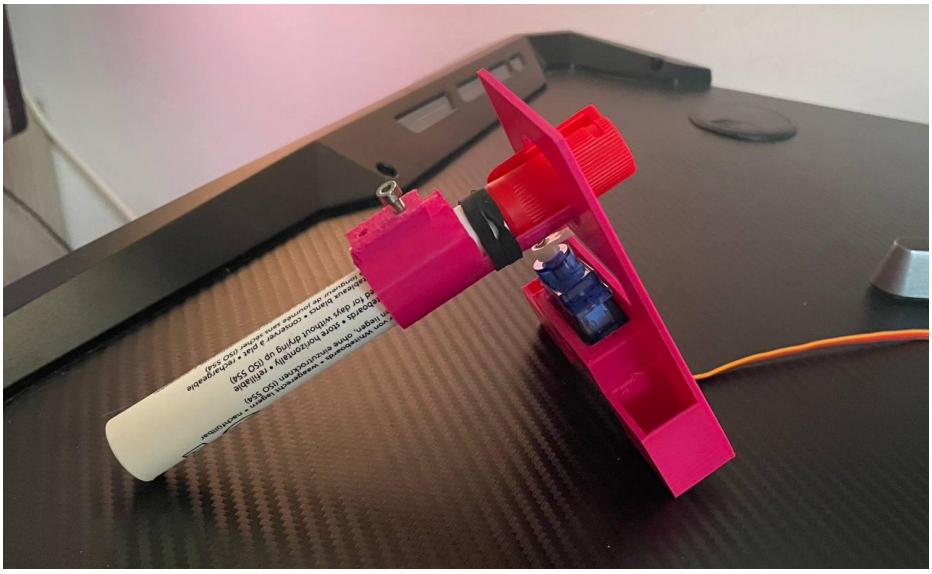


Figura 5.2: Diseño armado del soporte del rotulador y el servo.

5.2 Firmware

Siguiendo el diagrama de flujo diseñado en la Figura 4.9, dada la sencillez de los pasos a seguir, contaremos con un único fichero que se encargará de leer las líneas del archivo G-Code. Siguiendo los estándares, si la línea leída contiene los caracteres M05 el servo debe posicionarse a una distancia que forme un ángulo de 110º respecto a su centro para separarse de la pizarra. Cuando la línea contenga los caracteres M03, el servo se moverá para formar un ángulo de 75º que eliminará la distancia entre el soporte y la pizarra para que el rotulador marque la pizarra y veamos la imagen que forma. Si la línea que se lee en el fichero posee las letras X o Y, significa que hay una coordenada. Esta se restará con la coordenada almacenada de la última lectura y calculando la distancia euclídea entre ambos puntos, se trazará una línea recta.

5.3 Software

En este apartado se centra la mayor complicación encontrada en la realización de este proyecto y es que el software que queríamos utilizar para la generación de G-Code se hizo imposible de instalar y utilizar debido al desarrollo en Python 2, una versión deprecada del lenguaje de programación Python. Este lenguaje puede ser instalado y almacenado en variables de entorno del sistema para que la ejecución del código en diferentes versiones sea transparente para el usuario y así utilizar el software con normalidad. Esta instalación se intentó en diferentes equipos con el mismo resultado, un error de interpretación que barajó la posibilidad de abandonar el software elegido y cambiarlo por otro o realizar una refactorización.

Mediante ensayo de prueba y error en la ejecución del código, nos encontramos con funciones deprecada,.s y líneas de código que generaban errores debido al lenguaje obsoleto que es ahora Python2. Esta incompatibilidad me ha llevado al análisis del software para su corrección si quería seguir el estudio del análisis del Capítulo 3. Eliminaremos funciones que no se utilizan y añadiremos librerías actualizadas al sistema para garantizar un mantenimiento del código. Un aproximado del incremento del tiempo destinado para el proyecto, se ve reflejado en la Tabla 5.1, dónde estudiando el número de líneas de código, la complejidad matemática y los conocimientos del lenguaje del Ingeniero que llevará a cabo esta refactorización, se estima que un total de 14 días naturales será el cálculo más realista para la dedicación en la resolución de este bloqueo.

Cabe destacar que cada día corresponde a una jornada laboral de 8 horas y que cada día se completa cuando se han dedicado las 8 horas estimadas para cada actividad.

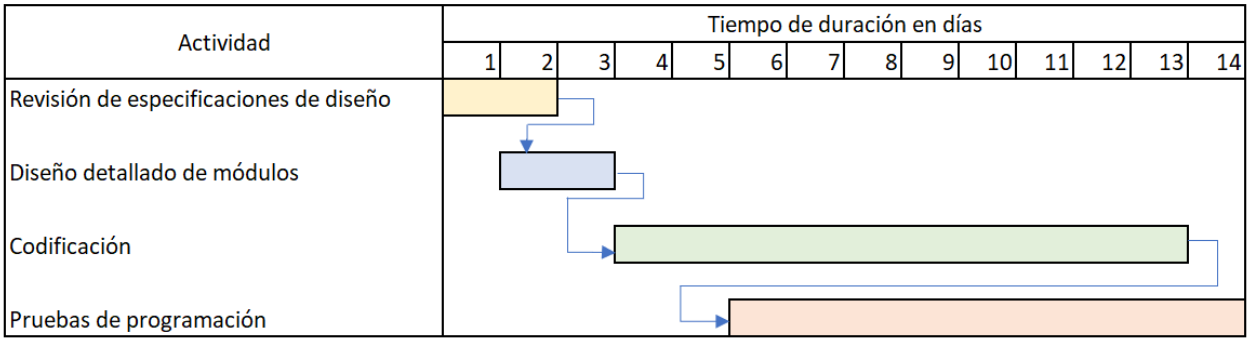


Tabla 5.1 Organización temporal para el proceso de adaptación del lenguaje.

Para interactuar con la extensión, se acoplará el código dentro del directorio indicado en el proceso de instalación [16]. Esto será necesario para que pueda ser reconocido por la aplicación Inkscape.

Este código consta de 5 ficheros importantes que pasamos a comentar y a explicar a continuación tras la refactorización y migración del lenguaje a Python 3.

BachinMaker.inx

Este fichero xml, trata de un fichero de configuración y dependencias mostrado en la Figura 5.3 para la generación de una ventana llamada "G-Code Output", En esta se detallan todos los parámetros que el usuario debe introducir para indicar la posición inicial de la imagen dentro del plano ortonormal representado por la pizarra, también el fichero dónde se almacenará el G-Code e incluso, que letra usaremos para definir la escritura y mover nuestro servo en la pizarra. En síntesis, será el fichero encargado de gestionar la interfase gráfica de la Figura 5.4 que se comunicará con el usuario para recibir los parámetros de entrada que darán como salida, la imagen transformada a G-Code.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<inkscape-extension xmlns="http://www.inkscape.org/namespace/inkscape/extension">
    <_name>Gcode Output</_name>
    <id>BachinMaker</id>
  <dependency type="executable" location="extensions">BachinMaker.py</dependency>
  <dependency type="executable" location="extensions">inkex.py</dependency>

  <param name="laser-command" type="string" _gui-text="Laser On/Pen Down:">M03 G1Z0</param>
  <param name="laser-off-command" type="string" _gui-text="Laser Off/Pen Up:">M05</param>

  <param name="posx" type="int" min="0" max="10000" _gui-text="Start Position X :">0</param>
  <param name="posy" type="int" min="0" max="10000" _gui-text="Start Position Y :">0</param>
  <param name="iscenter" type="boolean" _gui-text="Start Position Move To Center">false</param>

  <param name="travel-speed" type="int" min="0" max="10000" _gui-text="Jogging Speed (mm/min or in/min):
  ">3500</param>
  <param name="laser-speed" type="int" min="0" max="4000" _gui-text="Engraving / Draw Speed (mm/min or
  in/min):">2500</param>
  <param name="laser-power" type="int" min="0" max="20000" _gui-text="Laser Power / Pen Distance S# (
  0-1000):">1000</param>
  <param name="power-delay" type="float" min="0" max="1000" _gui-text="Delay (s):">0.2</param>
  <param name="passes" type="int" min="1" max="100" _gui-text="Repeat Speed (1-100):">1</param>
  <param name="pass-depth" type="float" min="0" max="5" _gui-text="Repeat Distance(1-5) (mm or in):">1</
  param>
  <param name="directory" type="string" _gui-text="Save Path:">D:\output</param>
  <param name="filename" type="string" _gui-text="File Name:">output.nc</param>
  <param name="add-numeric-suffix-to-filename" type="boolean" _gui-text="Add Num To File Name">true</
  param>

  <param name="unit" type="enum" _gui-text="Unit (mm or in):">
    <item value="G21 (All units in mm)">mm</item>
    <item value="G20 (All units in inches)">in</item>


  </param>

  <effect>
    <effects-menu>
      <submenu _name="BachinMaker Tool"/>
    </effects-menu>
    <object-type>path</object-type>
    </effect>

  <script>
    <command reldir="extensions" interpreter="python">BachinMaker.py</command>
  </script>

</inkscape-extension>

```

Figura 5.3: Captura del código del fichero BachinMaker.inx

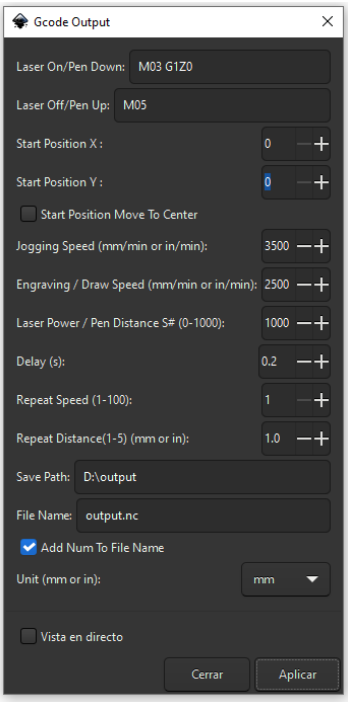


Figura 5.4: Visualización gráfica generada por la configuración del código del fichero BachinMaker.inx

BachinMaker.py

Este otro fichero, contiene funciones estáticas que se centran en la transformación de SVG a G-Code. Es el fichero principal, que quiere decir que es el fichero que se ejecuta y recibe las entradas para el funcionamiento del código y que también gestiona otras configuraciones básicas para la transformación de la imagen, posicionamiento, recorrido y generación de código. Es el Fichero que alimenta el panel de "G-Code Output" y en la imagen de la Figura 5.5 podemos ver la cabecera inicial con la licencia que permite su modificación y adaptación al nuevo lenguaje para el funcionamiento de este proyecto y proyectos futuros de todo quien quiera usar este gran programa.

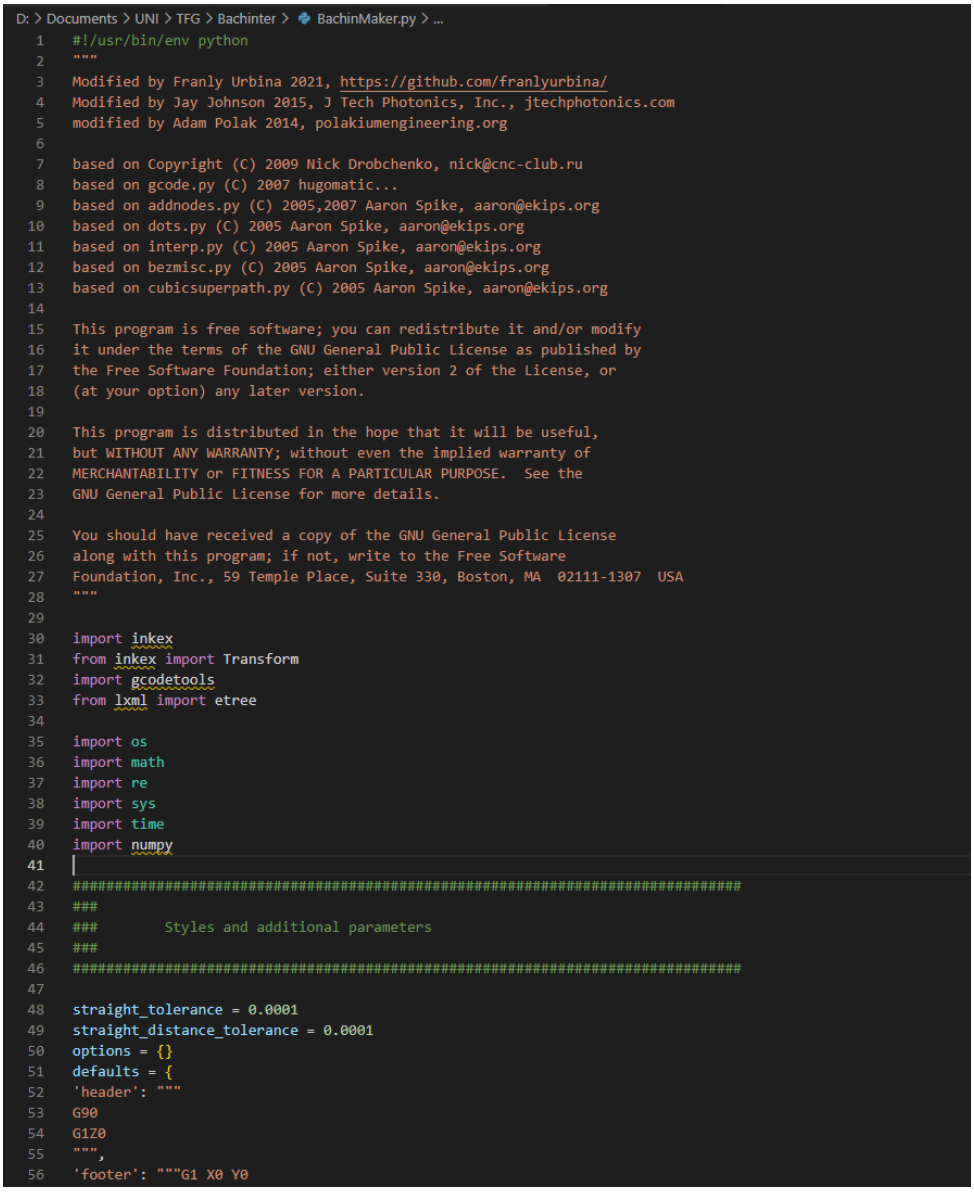


Figura 5.5: Captura del código del fichero BachinMaker.py

BachinMaker_hatch.inx

Fichero xml de configuración y dependencias para la ventana “Filling G-Code” para quienes quieran adaptar sus imágenes con estas funcionalidades. En este fichero mostrado en la Figura 5.6, se detallan los parámetros de entrada que serán recogidos en la ventana que se muestra en la Figura 5.7 y que llamarán al código encontrado el fichero BachinMaker_hatch.py

```
<?xml version="1.0" encoding="UTF-8"?>
<inkscape-extension xmlns="http://www.inkscape.org/namespace/inkscape/extension">
  <_name>Filling Gcode</_name>
  <id>command.BachinMaker Gcode.BachinMaker_hatch</id>
  <dependency type="extension">org.inkscape.output.svg.inkscape</dependency>
  <dependency type="executable" location="extensions">BachinMaker_hatch.py</dependency>
  <dependency type="executable" location="extensions">inkex.py</dependency>
  <dependency type="executable" location="extensions">simplepath.py</dependency>
  <dependency type="executable" location="extensions">simpletransform.py</dependency>
  <dependency type="executable" location="extensions">simplestyle.py</dependency>
  <dependency type="executable" location="extensions">cubicsuperpath.py</dependency>
  <dependency type="executable" location="extensions">cspsubdiv.py</dependency>
  <dependency type="executable" location="extensions">bezmisc.py</dependency>
  <dependency type="executable" location="extensions">plot_utils.py</dependency>

<param name="tab" type="notebook">
  <page name="splash" _gui-text="Image Filling Gocde">
  <_param name="Header" type="description" xml:space="preserve">
This BachinMaker Image Filling Tool will fill your sharp.

He will fill in the pattern which you choose.

The each packet in graphics you selected will be grouped.

  </_param>
  <param name="hatchSpacing" type="float" min="0" max="1000" _gui-text="   Filling Spacing (px)">1</
  param>
  <param name="hatchAngle" type="float" min="-360" max="360" _gui-text="   Filling Angle (°)">45</param>
  <param name="crossHatch" type="boolean" _gui-text="   Interlaced Scanning">false</param>

  <param name="reducePenLifts" type="boolean" _gui-text="   Connected Close End Point">true</param>
  <param name="hatchScope" type="float" min="1.1" max="5.0" _gui-text="         End Point Connect
  Distance (Default: 3)">3.0</param>
  <param name="holdBackHatchFromEdges" type="boolean" _gui-text="   Filling Start With Edge">true</param
  >
  <param name="holdBackSteps" type="float" min="0.1" max="10.0" _gui-text="         Edge Distance (px) (
  Default: 0.2)">0.2</param>
  <param name="tolerance" type="float" min="0.1" max="100" _gui-text="         Tolerance (Default: 20)">
  20.0</param>

  </page>
  <page name="info" _gui-text="About">
  <_param name="aboutpage" type="description" xml:space="preserve">
Hatch spacing is the distance between hatch lines,
measured in units of screen pixels (px). Angles are in
degrees from horizontal; for example 90 is vertical.
```

Figura 5.6: Captura del código del fichero BachinMaker_hatch.inx

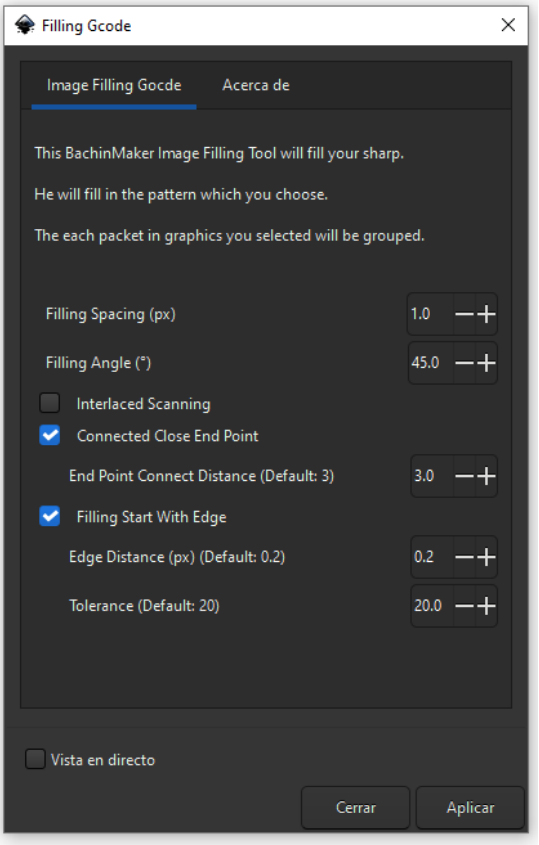


Figura 5.7: Visualización gráfica generada por la configuración del código del fichero BachinMaker_hatch.inx

BachinMaker_hatch.py

Los parámetros anteriormente recogidos pasarán a este fichero con funciones estáticas que basa su contenido en funciones matemáticas centradas en la intersección entre imágenes, transformaciones matriciales y vectoriales para las imágenes, cálculo de la distancia euclídea, calculo lineal, posicionamiento (entre otras funciones) para la generación de imágenes SVG que gestionan la configuración en el panel “Filling G-Code”, un ejemplo de código se muestra en la Figura 5.8 ya migrado a Python 3.

```
677     def addPathVertices( self, path, node=None, transform=None ):
678
679         '''
680         Decompose the path data from an SVG element into individual
681         subpaths, each starting with an absolute move-to (x, y)
682         coordinate followed by one or more absolute line-to (x, y)
683         coordinates.  Each subpath is stored as a list of (x, y)
684         coordinates, with the first entry understood to be a
685         move-to coordinate and the rest line-to coordinates.  A list
686         is then made of all the subpath lists and then stored in the
687         self.paths dictionary using the path's lxml.etree node pointer
688         as the dictionary key.
689         '''
690
691         if ( not path ) or ( len( path ) == 0 ):
692             return
693
694         # parsePath() may raise an exception.  This is okay
695         sp = path.split("\\")[:-1]
696         if ( not sp ) or ( len( sp ) == 0 ):
697             return
698
699         # Get a cubic super duper path
700         p = cubicsuperpath.CubicSuperPath( sp )
701         if ( not p ) or ( len( p ) == 0 ):
702             return
703
704         # Apply any transformation
705         if transform != None:
706             simpletransform.applyTransformToPath( transform, p )
707
708         # Now traverse the simplified path
709         subpaths = []
710         subpath_vertices = []
711         for sp in p:
712             # We've started a new subpath
713             # See if there is a prior subpath and whether we should keep it
714             if len( subpath_vertices ):
715                 if distanceSquared( subpath_vertices[0], subpath_vertices[-1] ) < 1:
716                     # Keep the prior subpath: it appears to be a closed path
717                     subpaths.append( subpath_vertices )
718             subpath_vertices = []
719             subdivideCubicPath( sp, float( self.options.tolerance / 100 ) )
720             for csp in sp:
721                 # Add this vertex to the list of vertices
722                 subpath_vertices.append( csp[1] )
723
724         # Handle final subpath
725         if len( subpath_vertices ):
726             if distanceSquared( subpath_vertices[0], subpath_vertices[-1] ) < 1:
727                 # Path appears to be closed so let's keep it
728                 subpaths.append( subpath_vertices )
```

Figura 5.8: Captura del código del fichero BachinMaker_hatch.py

plot_utils.py

Este fichero contiene funciones estáticas y basa su contenido en operaciones matemáticas de soporte para el cálculo de las dimensiones de las imágenes utilizando distintas unidades de medida. En la Figura 5.9 se muestra una captura de este documento para dar un ejemplo de las funciones que contiene.

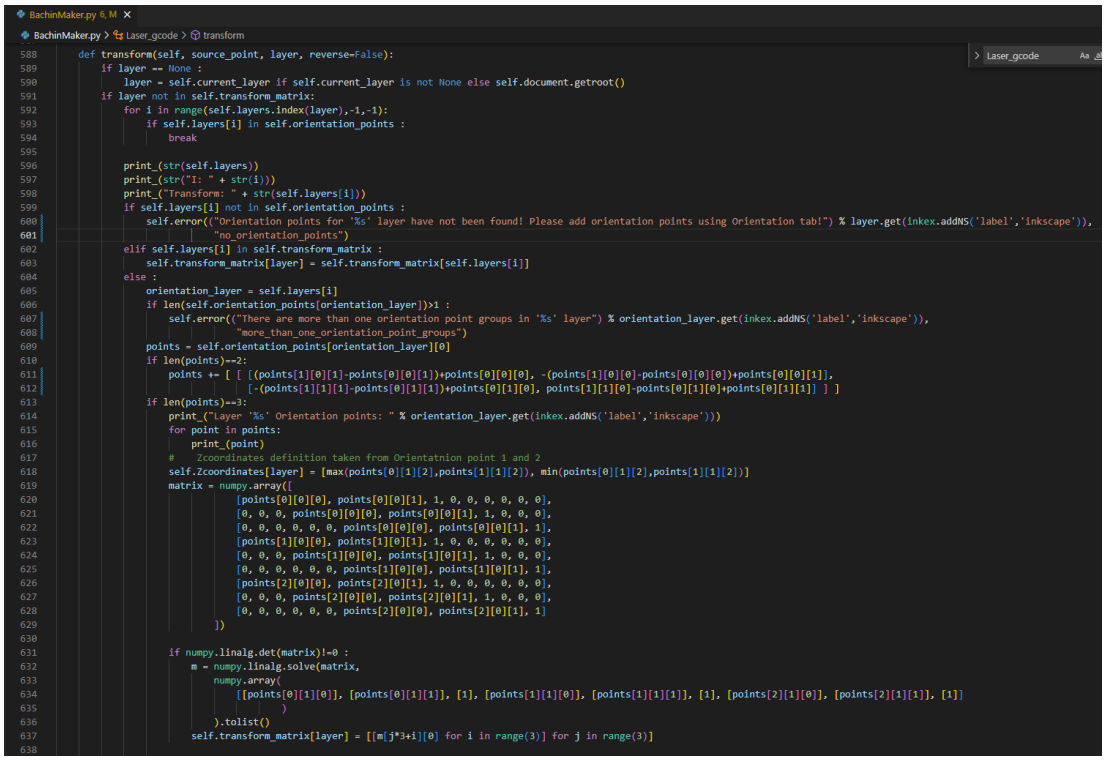
```
33  from math import sqrt
34  import cspsubdiv
35
36  def version():
37      return "0.4"    # Version number for this document
38
39
40  def parseLengthWithUnits( str ):
41      '''
42      Parse an SVG value which may or may not have units attached
43      This version is greatly simplified in that it only allows: no units,
44      units of px, and units of %.  Everything else, it returns None for.
45      There is a more general routine to consider in scour.py if more
46      generality is ever needed.
47      '''
48      u = 'px'
49      s = str.strip()
50      if s[-2:] == 'px':
51          s = s[:-2]
52      elif s[-2:] == 'in':
53          s = s[:-2]
54          u = 'in'
55      elif s[-2:] == 'mm':
56          s = s[:-2]
57          u = 'mm'
58      elif s[-2:] == 'cm':
59          s = s[:-2]
60          u = 'cm'
61      elif s[-1:] == '%':
62          u = '%'
63          s = s[:-1]
64
65      try:
66          v = float( s )
67      except:
68          return None, None
69
70      return v, u
71
72
73  def getLength( altself, name, default ):
74      '''
75      Get the <svg> attribute with name "name" and default value "default"
```

Figura 5.9: Captura del código del fichero plot_utils.py

5.4 Conclusiones

La esquematización de las acciones a seguir en un proyecto, son fundamentales para conseguir un buen resultado, la investigación y comparación de este proyecto con otros, ha ayudado a la elección de las herramientas idóneas en su construcción. Sin lugar a duda, hay que tener en cuenta que se pueden presentar nuevos retos que no estaban contemplados como los problemas con el software que hemos tenido. Fácilmente se pudo tomar la decisión de usar otro programa, pero siendo un software que se utiliza en múltiples plataformas, con una alta demanda y uso por los usuarios, disponiendo del código fuente me vi con

el interés de aportar una solución a la comunidad y a más personas que este cambio de versiones entre lenguajes le ocasionara problemas. Así, me vi sumergida en complejidades matemáticas como las transformaciones lineales matriciales para el escalado de la imagen, teniendo que acudir en más de una oportunidad a lo aprendido en la asignatura de matemáticas de la carrera para expresar en un lenguaje de alto nivel, funciones relacionadas con las transformaciones lineales, las cuales podemos apreciar en el código de la Figura 5.10, consiguiendo la satisfacción de poder decir, esto es ingeniería.




```
    def transform(self, source_point, layer, reverse=False):
        if layer == None :
            layer = self.current_layer if self.current_layer is not None else self.document.getroot()
        if layer not in self.transform_matrix:
            for i in range(self.layers.index(layer),-1,-1):
                if self.layers[i] in self.orientation_points :
                    break

            print_(str(self.layers))
            print_(str("I: " + str(i)))
            print_("Transform: " + str(self.layers[i]))
            if self.layers[i] not in self.orientation_points :
                self.error(("Orientation points for '%s' layer have not been found! Please add orientation points using Orientation tab!") % layer.get(inkex.addNS('label','inkscape')),
                           "no_orientation_points")
            elif self.layers[i] in self.transform_matrix :
                self.transform_matrix[layer] = self.transform_matrix[self.layers[i]]
            else :
                orientation_layer = self.layers[i]
                if len(self.orientation_points[orientation_layer])>1 :
                    self.error(("There are more than one orientation point groups in '%s' layer") % orientation_layer.get(inkex.addNS('label','inkscape')),
                               "more_than_one_orientation_point_groups")
                points = self.orientation_points[orientation_layer][0]
                if len(points)==2:
                    points += [ [ [(points[1][0][1]-points[0][0][1])+points[0][0][0], -(points[1][0][0]-points[0][0][0])+points[0][0][1]],
                                  [-(points[1][1][1]-points[0][1][1])+points[0][1][0], points[1][1][0]-points[0][1][0]+points[0][1][1]] ] ]
                if len(points)==3:
                    print_("Layer '%s' Orientation points: " % orientation_layer.get(inkex.addNS('label','inkscape')))
                    for point in points:
                        print_(point)
                    #    Zcoordinates definition taken from Orientatnion point 1 and 2
                    self.zcoordinates[layer] = [max(points[0][1][2],points[1][1][2]), min(points[0][1][2],points[1][1][2])]
                    matrix = numpy.array([
                                [points[0][0][0], points[0][0][1], 1, 0, 0, 0, 0, 0],
                                [0, 0, 0, points[0][0][0], points[0][0][1], 1, 0, 0],
                                [0, 0, 0, 0, 0, 0, points[0][0][0], points[0][0][1], 1],
                                [points[1][0][0], points[1][0][1], 1, 0, 0, 0, 0, 0],
                                [0, 0, 0, points[1][0][0], points[1][0][1], 1, 0, 0],
                                [0, 0, 0, 0, 0, 0, points[1][0][0], points[1][0][1], 1],
                                [points[2][0][0], points[2][0][1], 1, 0, 0, 0, 0, 0],
                                [0, 0, 0, points[2][0][0], points[2][0][1], 1, 0, 0],
                                [0, 0, 0, 0, 0, 0, points[2][0][0], points[2][0][1], 1]
                            ])

                    if numpy.linalg.det(matrix)!=0 :
                        m = numpy.linalg.solve(matrix,
                            numpy.array(
                                [[points[0][1][0]], [points[0][1][1]], [1], [points[1][1][0]], [points[1][1][1]], [1], [points[2][1][0]], [points[2][1][1]], [1]]
                                    )
                            ).tolist()
                        self.transform_matrix[layer] = [[m[j*3+i][0] for i in range(3)] for j in range(3)]
```

Figura 5.10: Calculo para la transformación lineal de matrices relacionada con las coordenadas de la imagen.

Capítulo 6.
Pruebas.

"Descubrir lo inesperado es más importante que confirmar lo conocido."

GEORGE E. P. BOX

Durante la realización del proyecto, se ha visto como han existido tres fases independientes, que tienen como finalidad converger para generar un único producto. Dicho producto, el plotter vertical, tiene en todo su ciclo la capacidad de fabricar una entrada de datos mediante la lectura de una imagen, transformar los datos en coordenadas y rotar dos motores sincronizando sus movimientos para generar un dibujo. Explicaremos las pruebas realizadas para concluir con la unión de todas estas fases y así generar el producto final mostrado en la Figura 6.1.

Figura 6.1: Plotter vertical trazando la imagen del escudo de la UNED, finalizado y testeado.

6.1 Primer Diseño.

El primer prototipo se realizó con un diseño en paletas de helado para determinar el peso, las longitudes de los hilos tensores, las dimensiones del plotter y las coordenadas de posición de los motores. En la Figura 6.2 a continuación, podemos ver una imagen del primer diseño realizado y como este carecía de estética, no obstante, funcionó a la perfección y dio vida al resto de modelos que se realizaron a partir de este más robusto.

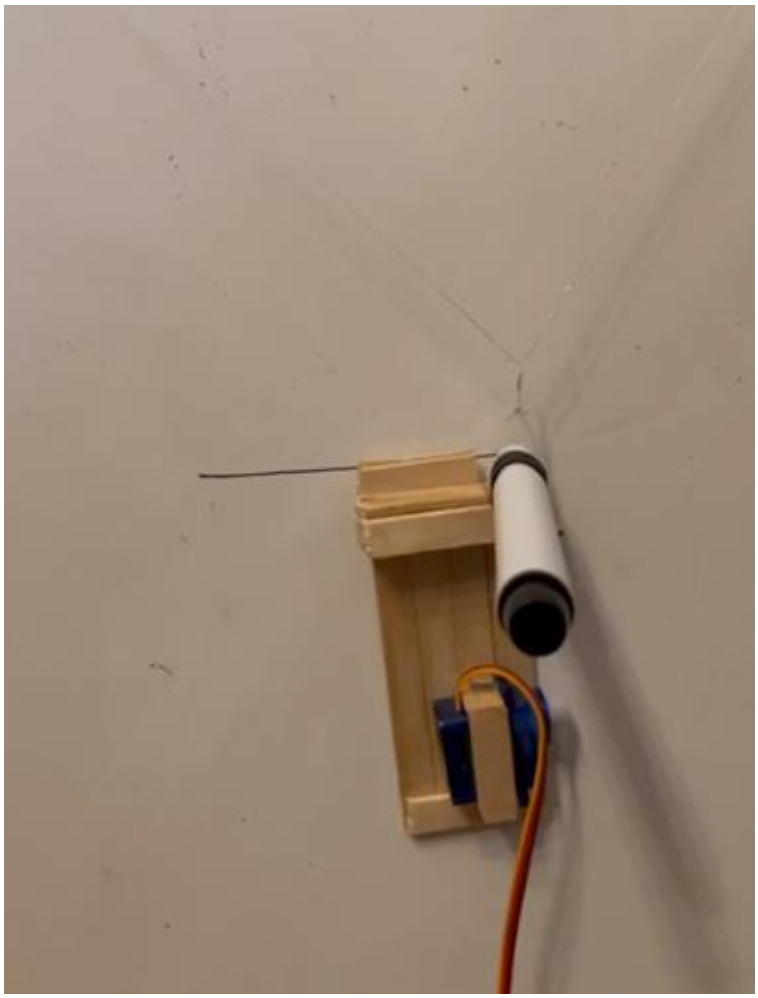


Figura 6.2: Plotter vertical. Primer diseño.

Esta decisión, de realizar con paletas de helado la base que sujetaría el rotulador, fue de gran utilidad para comprender la importancia que juega el equilibrio y la fuerza de gravedad sobre el rotulador. Este primer diseño generaría la confianza suficiente para diseñar mediante los modelos 3D de la Figura 6.3, los componentes hardware que protegerían el plotter y dotarían se sujeción y firmeza el dispositivo.

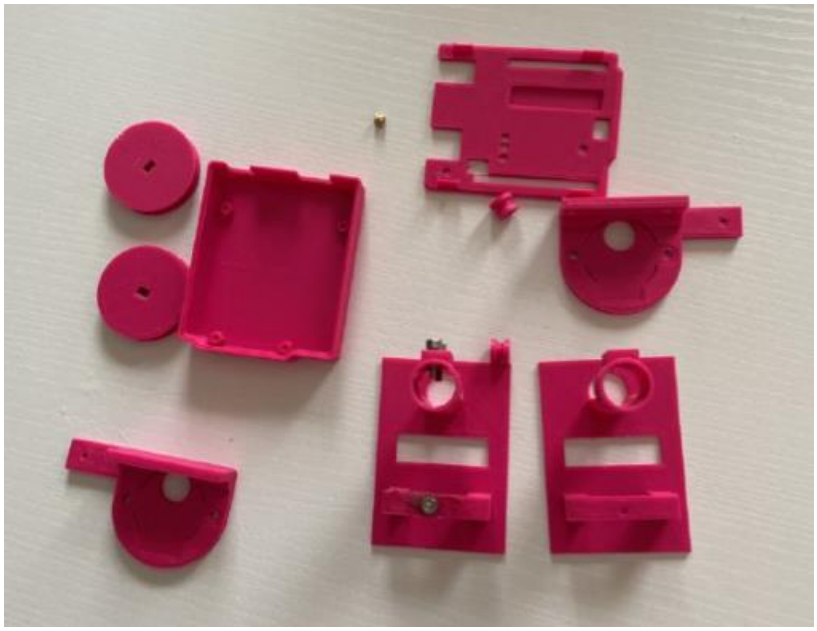


Figura 6.3: Plotter vertical. Primeras piezas 3D.

Teniendo las nuevas piezas en 3D, tuve que enfrentarme a distintos problemas, aunque el diseño con las paletas era ideal tanto en pesos como en diseño, no era nada profesional pero el cambio a piezas de plástico ocasiono una gran inestabilidad en el plotter.

El primero de ellos fue no haber contemplado la posibilidad de pérdida o cambio de rotulador, ni que existe la posibilidad de adquirir uno con distintas dimensiones y peso. Particularmente, mi rotulador de pizarra se agotó y tuve que improvisar con un bolígrafo y papeles mientras rediseñaba las piezas 3D del plotter, haciéndome ver que el peso y tamaño del rotulador, junto con la elección del punto de equilibrio era fundamental para un correcto funcionamiento. En la Figura 6.3 podemos ver un ejemplo de prueba donde se intenta imprimir un conjunto de líneas rectas que, debido al bamboleo del peso, terminan curvándose impidiendo una correcta impresión de la imagen esperada.

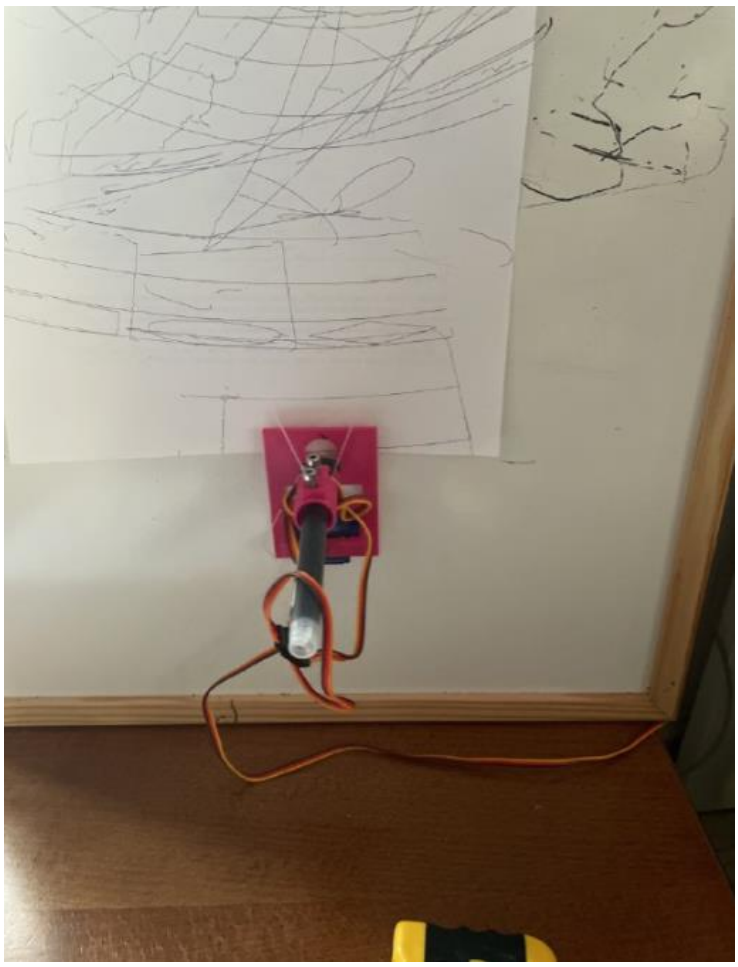


Figura 6.4: Prueba de líneas verticales y horizontales.

Se observa que los pesos no permitían una correcta impresión del dibujo con el nuevo bolígrafo, ya que el desbalance ocasionado por la densidad de los materiales hacía que el plotter fuese susceptible a movimientos muy bruscos por parte del servo, teniendo que modificar el diseño y los grados de rotación determinados en un principio para hacer más suave el levantamiento del rotulador y generando una correcta impresión del dibujo.

Estas correcciones del peso se realizaron y ya se pudo comprobar que la impresión de las imágenes era más nítida y fluida, tal y como podemos apreciar en la Figura 6.4.

Figura 6.5: Correcciones de peso y del punto de equilibrio.

6.2 Evolución de la placa base.

A la par que se corregían los pesos, se analizaba y ajustaba el código que daba vida al plotter y controlaba el movimiento de los motores paso a paso. Durante esta fase, pude notar que la placa Arduino UNO que era recomendada para la realización de un proyecto con estas características y que se muestra en la Figura 6.5, no soportaba diseños grandes dada la complejidad que estos podrían arrastrar, hablamos de un exceso en el número de líneas y coordenadas que el compilador tenía que procesar, por lo que, una vez encontrado el error, se tomó la decisión de acudir a una Arduino MEGA con suficientes conexiones que permitiera el procesamiento del fichero. En la Figura podemos ver la captura realizada a un vídeo que grabó los errores de lectura de la Arduino UNO y llevó a la toma de decisión del cambio de la plaga base.

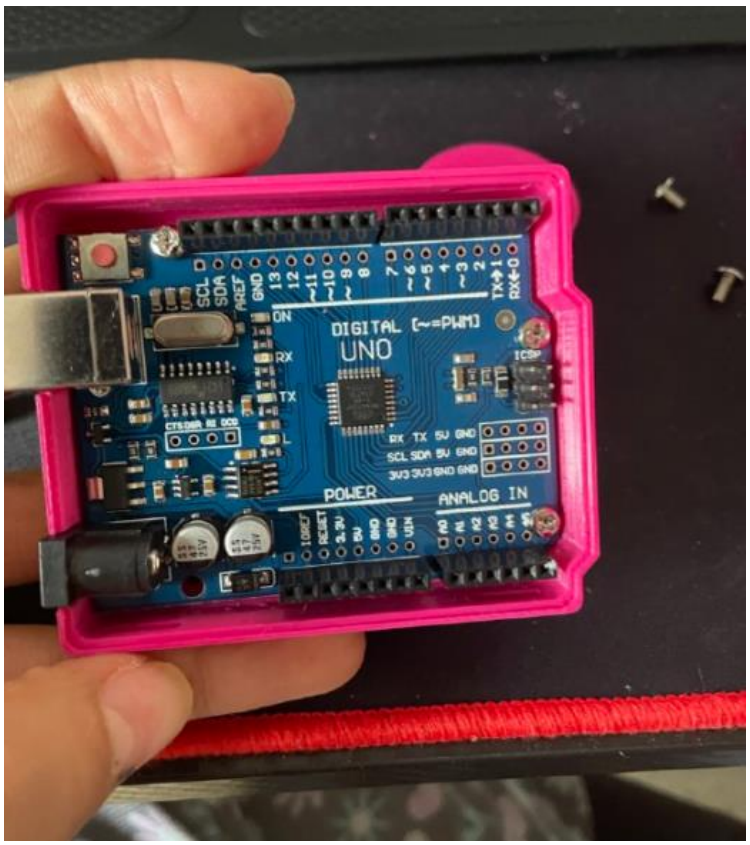


Figura 6.6: Soporte del Arduino UNO.

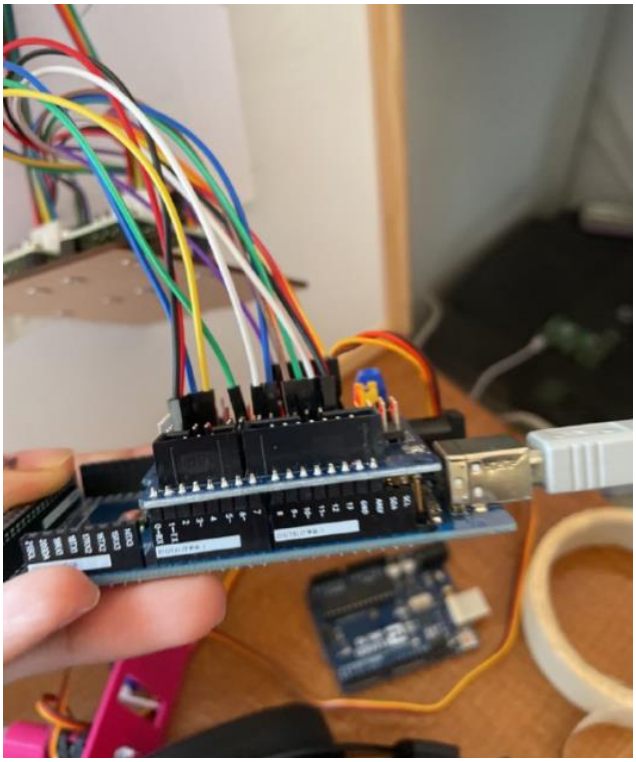


Figura 6.7: Cambio de placa durante la creación del proyecto.

6.3 Diseño final

Una vez realizadas todas las pruebas con el hardware y el firmware, se procede a probar el software con las respectivas funciones que harán la lectura de la imagen y la transformarán a G-Code. En la Figura 6.8, podemos ver un ejemplo de prueba del código encargado de medir la distancia euclídea entre dos coordenadas.

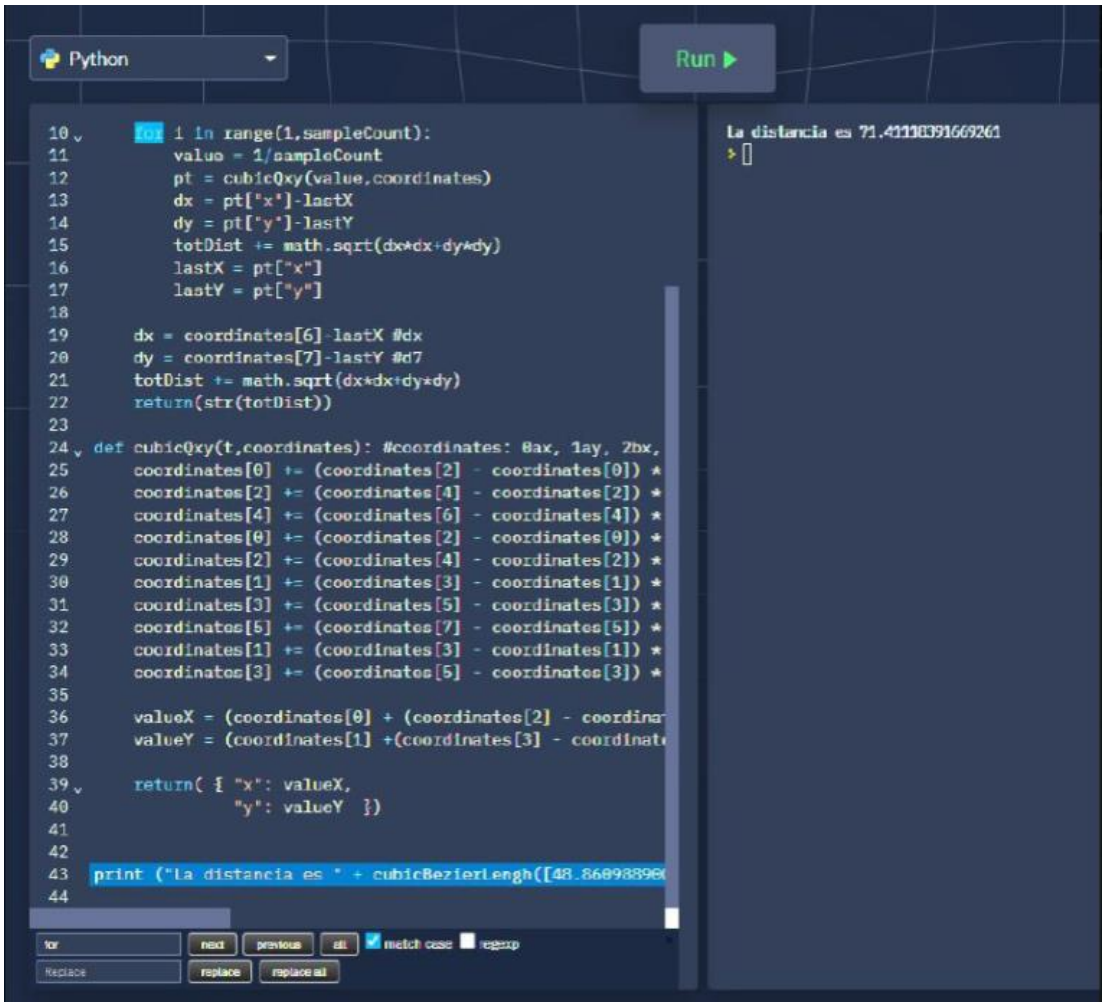


Figura 6.8: Prueba unitaria de la función cubicBezierLengh.

El procedimiento de pruebas se ha realizado mediante la aplicación de pruebas unitarias de las funciones, aislando cada una de ellas y verificando que los valores coinciden con los esperados, generando finalmente, la transformación que vemos en la Figura 6.9, la cual muestra a su izquierda la imagen que se desea imprimir, en medio el código generado por el software y a la derecha el resultado de leer el G-Code generado partiendo la imagen inicial.

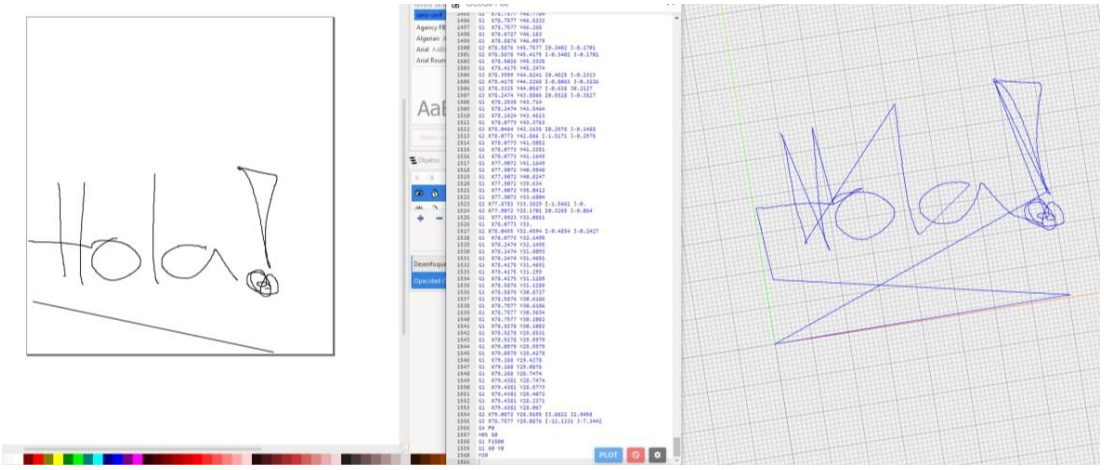


Figura 6.9: Comprobación de la imagen generada.

6.4 Conclusiones

Durante toda la fase del desarrollo, las pruebas jugaron un importante papel ya que, a medida que se avanzaba con el proyecto, más mediciones y más confirmaciones para el correcto funcionamiento del plotter tenían que hacerse. Las pruebas evolucionaron a la par de las correcciones y adaptaciones que estuvieron a la orden del día, sin duda han sido una de las fases más importante y que más tiempo han llevado de todo el proceso de creación de este proyecto.

También habría que destacar que estas pruebas no se ciñeron al diagrama contemplado en la *Tabla 5.1* como hubiese esperado, por el contrario, podríamos hablar de una subdivisión constante del tiempo total estimado desde el inicio del proyecto, dejando un diagrama más parecido al mostrado a continuación en la *Tabla 6.1* y cuyo tema profundizaremos en el siguiente capitulo.

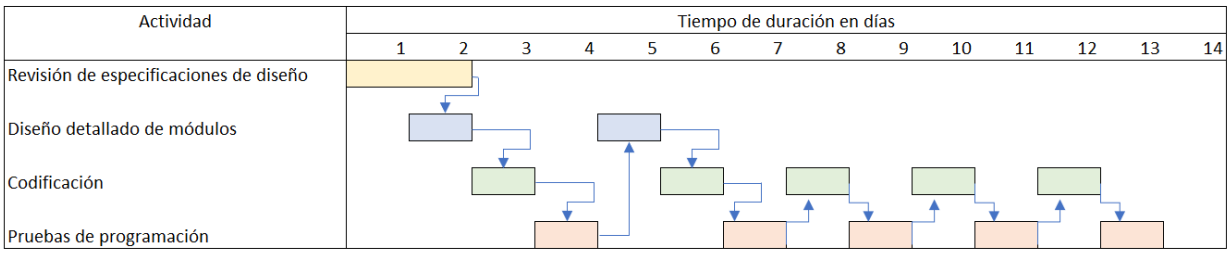


Tabla 6.1 Diagrama de Gantt implementado.

Capítulo 7.
Planificación y coste.

"El costo de una cosa es la cantidad de aquello que yo llamo vida, necesaria para adquirirla, ya sea a corto o a largo plazo."

HENRY DAVID THOREAU

Un proyecto, bien sea un viaje; un juego; una escapada o ante una decisión de la vida; ha de estar acompañado de la planificación. Tener una idea clara de cuál será el inicio y que a dónde pretendemos llegar, es esencial para marcar las pautas que nos llevaran a conseguir lo que buscamos, entender si en el camino es lo que queremos o si ese objetivo es alcanzable.

Formalizando esta definición y extrapolándola a este trabajo, la planificación no ha estado solo en el inicio del proyecto, sino durante todo el proceso de desarrollo, ya que contestemente se han tenido que hacer redefinir los objetivos, contando con hardware que no conocíamos como el Sensor Shield v5.0 debido al cambio de placa para el procesamiento de código, el cambio constante en el diseño de piezas 3D y la necesidad de generar nuestro propio G-Code, son solo unos de los más anecdóticos sucesos encontrados durante la creación de este proyecto.

Los costes finales analizados en el Capítulo 3 distan mucho los realmente generados, sobre todo dada la constante redefinición de las piezas y su impresión, los cuales, encarecieron el presupuesto. Además, en el cálculo establecido para el desarrollo del código, no se contaba con la necesidad de un software que analizara cualquier imagen como entrada, dando lugar a un proyecto mucho más grande que el estimado desde un inicio y, por ende, con unos costes que igualmente por encima de lo estipulado.

7.1 Coste del Software

Es muy común para este tipo de cálculos utilizar la formula del Esfuerzo en personas/año.

$$E = [(L * B^{1/3}) / P]^3 / t^4$$

Donde:

L = Líneas de código funcional

B = Factor de habilidades especiales

P = Parámetro de productividad

T = Duración del proyecto

Fichero	Líneas de código
BachinMaker.inx	43
BachinMaker.py	980
BachinMaker_hatch.inx	77
BachinMaker_hatch.py	1728
plot_utils.py	134
vectorial_draw.ino	194
TOTAL	3.156

Según el código programado, nos encontramos que las líneas de códigos estudiadas y desarrolladas para el proyecto son de 3.156. A esto habría que descontar las líneas comentadas y hacer la estimación del coste, pero personalmente, tras años de estudios con las buenas prácticas y tras mi experiencia personal en el área laboral durante los últimos 8 años en el área de calidad, quiero dar mi visión personal a este calculo de la estimación.

Tanto las líneas comentadas como las líneas de código son básica para los desarrollos en vigencia y futuros evolutivos ya que estas permiten conocer y entender profundamente en que estamos trabajando y cuáles serán nuestros objetivos, como abordarlos y que información de valor existe. Si los comentarios no fuesen valiosos, no deberían existir y el código se convertiría en una estructura inasumible por cualquier programador e ingeniero. Los comentarios son fundamentales y queda bajo la responsabilidad del proveedor si se ha realizado una buena practica empleando y cobrando a un precio razonable y dentro de los valores de mercado por el trabajo realizado.

Aunque esta formula aplica un valor dividido por el tiempo de trabajo, no debemos menospreciar la experiencia de un ingeniero con 10 años en el área laboral frente a un ingeniero que recientemente se incorpora al sector. En estos años se estima que dicha experiencia ha alimentado la visión y abordaje de desarrollos complejos que permitirán en mucho menos tiempo la creación y modificación de los proyectos, pero a su vez, habrán mejorado la simpleza y elegancia de la resolución de los problemas. Se toma en cuenta que los problemas a resolver mediante los distintos lenguajes de programación son agnósticos a dicho problema, por lo que la verbosidad de un lenguaje influirá positiva o negativamente en estos cálculos. Esto no es para nada equilibrado y afecta negativamente a un ingeniero con experiencia si su solución puede encontrarse en 10 líneas de código frente a la misma solución en 300 líneas, en tanto en cuanto la formula apremie positivamente un desarrollo mas engorroso y menos eficiente con estas características.

Mi solución a esta problemática se centra en conocer el valor de mercado promedio de un desarrollador con mi perfil y estudiar el tiempo en horas que se ha tardado en realizar el proyecto mas el coste material. Por simplificar, se estima en base a la Tabla 5.1, que las horas laborales empleadas han sido 8 horas en 14 días, esto hace un total de 112 horas de trabajo. Dado a que el salario medio de un ingeniero con experiencia en España a fecha de septiembre de 2022 es de unos 40.000€/anual, el calculo nos queda en 112 horas * 21,28€/hora, dándonos un total de 2.382,97€ + 167,41€ en concepto de materiales calculados en la Tabla 3.1, generando un total de 2550,38€ para este proyecto.

7.2 Conclusiones

Aunque el cálculo de estimación de esfuerzo ha sido muy popularizado y se encuentra detallada en un sin numero de trabajos y libros, quiero apuntar con este proyecto que el calculo usado con esta formula no se ajusta desde mi punto de vista a los estándares actuales, ya que las empresas con las que he tratado para hacer el cálculo de nuevo personal no lo aplican aunque sea de gran utilidad, principalmente veo que es debido a depender de variables como el valor de habilidades especiales B que podría ajustarse o no a la realidad según quien decida cuál sea su valor. Para formalizar este apunte, si aplicamos la fórmula para el mismo ejemplo en dónde L = 3.156; B=0,75 por tener experiencia; P=10.000 siguiendo una metodología adecuada, y t=3/4 por ser 3 semanas de un mes, obtendríamos que:

$$E = \left(\frac{3156 * (0{,}75)^{\frac{1}{3}}}{10.000} \right)^3 * \frac{1}{(0{,}75)^4} = 0{,}07451$$

Si esto lo multiplicamos por el salario anual que comentamos anteriormente de 40.000€/anual nos deja en 2980.48€. Esto es muy aproximado a los valores de mercado calculados anteriormente con lo que podemos concluir que el valor de mercado de este proyecto estaría entre los 2550,38€ y 2980.48€ aproximadamente.

Capítulo 8.
Conclusiones.

> *"La conclusión es que sabemos muy poco y, sin embargo, es asombroso lo mucho que conocemos. Y más asombroso todavía que un conocimiento tan pequeño nos pueda dar tanto poder."*
>
> **BERTRAND RUSSEL**

Cada uno de estos capítulos ha tenido una frase de entrada que marca con especial detalle un mensaje relacionado con todo lo que cada uno de los capítulos ha enseñado. Particularmente, esta frase si la relacionamos con todos los años de carrera, las asignaturas y todo el proceso para llegar a este proyecto, nos damos cuenta del impacto que tiene la investigación y el desarrollo que se ha llevado a cabo en este trabajo. Empecé mis estudios sin saber que era la programación, mucho menos una ingeniería, 240 créditos después, sigo pensando que aún queda mucho por aprender y por conseguir, no obstante, todo lo que se nos ha enseñado y todo lo que he aprendido con este grado, culmina dándome la suficiente confianza y fortaleza como para determinar que es un proyecto, como iniciarlo y como generar un producto.

El objetivo de este proyecto ha sido crear un dispositivo plotter que pudiese imprimir imágenes en una superficie vertical y que la impresión, siguiese una traza óptima en función de la imagen en lugar de línea a línea como lo haría una impresora convencional. Estos objetivos han sido satisfechos como hemos podido observar en el Capítulo 5 y que, además, se ha complementado con la adaptación de un programa propio para la generación del código vectorial que este y cualquier otro dispositivo puede leer para imprimir imágenes. Se ha hecho uso de las licencias de código libre para mejorar y publicar dicho código contribuyendo con la comunidad de programadores y diseñadores interesados en explotar esta herramienta y se ha realizado una estimación bastante aproximada a los valores de mercado del coste que conllevaría este proyecto.

La finalidad de este documento ha sido detallar los pasos para que cualquiera que lo desee, pueda generar su propio dispositivo aplicando los conocimientos físicos, lógicos y estratégicos que se han aplicado gracias a años de estudios y

perseverancia con el detalle de poder dar pie a mejoras, nuevas ideas y que sea abierto a evoluciones y correcciones futuras, tal y como comentaremos en el siguiente y último capítulo de esta memoria.

Capítulo 9.
Trabajos futuros.

"La mejor manera de predecir el futuro es crearlo."

PETER DRUCKER

A través del desarrollo de este proyecto se han generado distintas oportunidades de crecimiento y evolución del plotter, que han venido de compañeros, amigos y personas ajenas al trabajo. Plasmaremos los mas relevantes que podrían dar una continuidad útil y versátil al dispositivo y que mantendría un proceso de constante desarrollo.

9.1 Adaptación para niños en edad preescolar.

En una de mis conversaciones con compañeros de la universidad, surgió el tema de tratar la necesidad que tienen los nuevos programas educativos de incorporar la informática y la robótica a niños de edad preescolar para que se familiaricen con las nuevas tecnologías. Dado a que la evolución de nuestro sector es rápida y constante, se desea mediante el Ministerio de Educación que estos conocimientos lleguen a las nuevas generaciones con mas facilidad y rapidez por medio de los nuevos profesores y estudiantes de magisterio. Se habló del presente trabajo y la idea de poder darle a los niños un dispositivo fácil, sencillo, entretenido y novedoso para hacer dibujos y que ellos puedan familiarizarse con la robótica, caló por completo entre una de las opciones más viables para presentar como siguiente paso a la evolución del plotter. Esto quiere decir, transformar las cubiertas y proteger al proyecto para que cualquier niño y profesor de infantil pueda hacer uso de el sin miedo a generar incidentes, con un manual de instrucciones apto para todas las edades.

9.2 Incorporación de dispositivo bluetooth para la recepción del código al Firmware.

Esta oportunidad de desarrollo haría más fácil y compacto al dispositivo, ya que si tenemos la oportunidad de enviar el código fuente que necesita la placa Arduino para leer el fichero con las coordenadas que generaría el dibujo mediante bluetooth, nos ahorraríamos el uso de la tarjeta SD para traspasar la información, facilitando notablemente su uso y control de las imágenes que se desean imprimir, ya que se controlaría desde un único ordenador y no desde dos como se hace actualmente al tener que trasladar la tarjeta SD hasta la placa Arduino.

9.3 Lectura y escritura mediante pulsos eléctricos cerebrales.

Un proyecto que me genera une especial entusiasmo, es el unir el presente proyecto con un dispositivo de lectura sensorial que permita el reconocimiento de imágenes mediante pulsos eléctricos y que el plotter pueda dibujar lo que el usuario transmite mediante sus ondas cerebrales, permitiendo a personas que así lo deseen, expresarse mediante el arte de imprimir algo más que las letras predefinidas de un dispositivo.

9.4 Conclusiones

Esta claro que este proyecto puede dar juego a muchas mas combinaciones que permitan su crecimiento, uso y disfrute, pero quiero centrarme en estas tres porque son las que han producido mayor entusiasmo a mi entorno, quienes realmente han deseado lo mejor en todo aquello que me he propuesto. Por mi parte y a nivel personal, el siguiente objetivo de este proyecto será el mencionado en el punto 9.1 como regalo a la profesora de infantil Diana Vides, que quedó encantada con mi proyecto y me animó a concluirlo. Ha estado conmigo durante cada paso que he dado como amiga y compañera por lo que será este un regalo de agradecimiento a ella y a sus niños de infantil por todo el tiempo que ha estado apoyándome y dándome los ánimos para decir, si se puede, esto y más.

Bibliografía

[1] Etimología - El origen de la palabra: óptimo. 1996-2022, de Mag. Ricardo Soca. Sitio web: https://www.elcastellano.org/palabra/%C3%B3ptimo

[2] Origen del lenguaje. 08 junio 2022, de Wikipedia, La enciclopedia libre. Sitio web: https://es.wikipedia.org/wiki/Origen_del_lenguaje

[3] Historia de la escritura. 18 junio 2022, de Wikipedia, La enciclopedia libre. Sitio web: https://es.wikipedia.org/wiki/Historia_de_la_escritura

[4] Johannes Gutenberg inventor de la Imprenta. 08 febrero 2009, de portalgraf.com. Sitio web: https://www.portalgraf.com/historia/johannes-gutenberg#:~:text=Johannes%20Gutenberg%20inventor%20de%20la,o%20de%20reyes%20y%20nobles.

[5] Johannes Gutenberg. 25 junio 2022, de Wikipedia, La enciclopedia libre. Sitio web: https://es.wikipedia.org/wiki/Johannes_Gutenberg

[6] La invención de la imprenta y su impacto en la historia. 11 marzo 2016, de uv.es. Sitio web: https://www.uv.es/uvweb/master-historia-formacion-mundo-occidental/es/blog/invencion-imprenta-impacto-historia-1285960141137/GasetaRecerca.html?id=1285961209839

[7] La Máquina Analítica de Babbage. 07 septiembre 2010, de gtd.es. Sitio web: https://www.gtd.es/es/blog/la-maquina-analitica-de-babbage

[8] La historia de la impresión: ¿Quién fue su inventor? 16 abril 2020, de mastertec.es. Sitio web: https://www.mastertec.es/blog/la-historia-de-la-impresion-quien-fue-su-inventor

[9] UNIVAC High Speed Printer. 11 mayo 2017, de tugurium.com. Sitio web: http://www.tugurium.com/gti/termino.php?Tr=UNIVAC%20High%20Speed%20Printer

[10] Plóter. 21 abril 2022, de Wikipedia, La enciclopedia libre. Sitio web: https://es.wikipedia.org/wiki/Pl%C3%B3ter

[11] ¿QUÉ ES UN PLOTTER? UTILIDADES, TIPOS Y DIFERENCIAS. 28 julio 2020, de digipressystem.com. Sitio web: https://digipressystem.com/que-es-un-plotter-utilidades-tipos-diferencias/

[12] Qué es un plotter y para qué sirve. 2022, de papiroflexiamania.com. Sitio web: https://www.papiroflexiamania.com/que-es-un-plotter-y-para-que-sirve

[13] Tipos de Plotter según tecnología. 2022, de tps-telecon.es. Sitio web: https://www.tps-telecon.es/blog/que-es-un-plotter-tipos-plotters-tamanos

[14] G-code. 10 marzo 2021, de Wikipedia, La enciclopedia libre. Sitio web: https://es.wikipedia.org/wiki/G-code

[15] QUÉ ES EL G-CODE Y SU IMPORTANCIA EN LA IMPRESIÓN 3D. 12 diciembre 2019, de Luis Llamas. Sitio web: https://www.luisllamas.es/que-es-el-g-code-y-su-importancia-en-la-impresion-3d/

[16] Bachin Articles. 26 diciembre 2019 de Inkscape. Sitio web: http://www.bachinmaker.com/index.php?p=85&a=view&r=55

[17] Inkscape, puede crear gráficos vectoriales. 2022, de Inkscape. Sitio web: https://inkscape.es/

[18] Lenguaje de programación de Arduino, estructura de un programa. 2022, de Aprendiendo Arduino. Sitio web: https://aprendiendoarduino.wordpress.com/2015/03/26/lenguaje-de-programacion-de-arduino-estructura-de-un-programa/#:~:text=Para%20programar%20un%20Arduino%2C%20el,posible%20programarlo%20en%20otros%20lenguajes.

[19] Figura 3.13: Diseño de conexión motor – driver – placa, generado en circuito.io. 2022, de Franly Urbina. Sitio web: https://www.circuito.io/app?components=9240,9240,9442,11061

[20] Figura 3.14: Diseño de conexión tarjeta SD – placa, generado en circuito.io. 2022, de Franly Urbina. Sitio web: https://www.circuito.io/app?components=9442,11061,1671987

[21] Figura 3.15: Diseño de conexión servo – placa, generado en circuito.io, 2022 de Franly Urbina. Sitio web: https://www.circuito.io/app?components=9442,10190,11061

[22] Figura 3.16: Diseño de conexión de componentes generado en circuito.io. 2022, de Franly Urbina. *Sitio web:* https://www.circuito.io/app?components=9240,9240,9442,11061,1671987,2345678

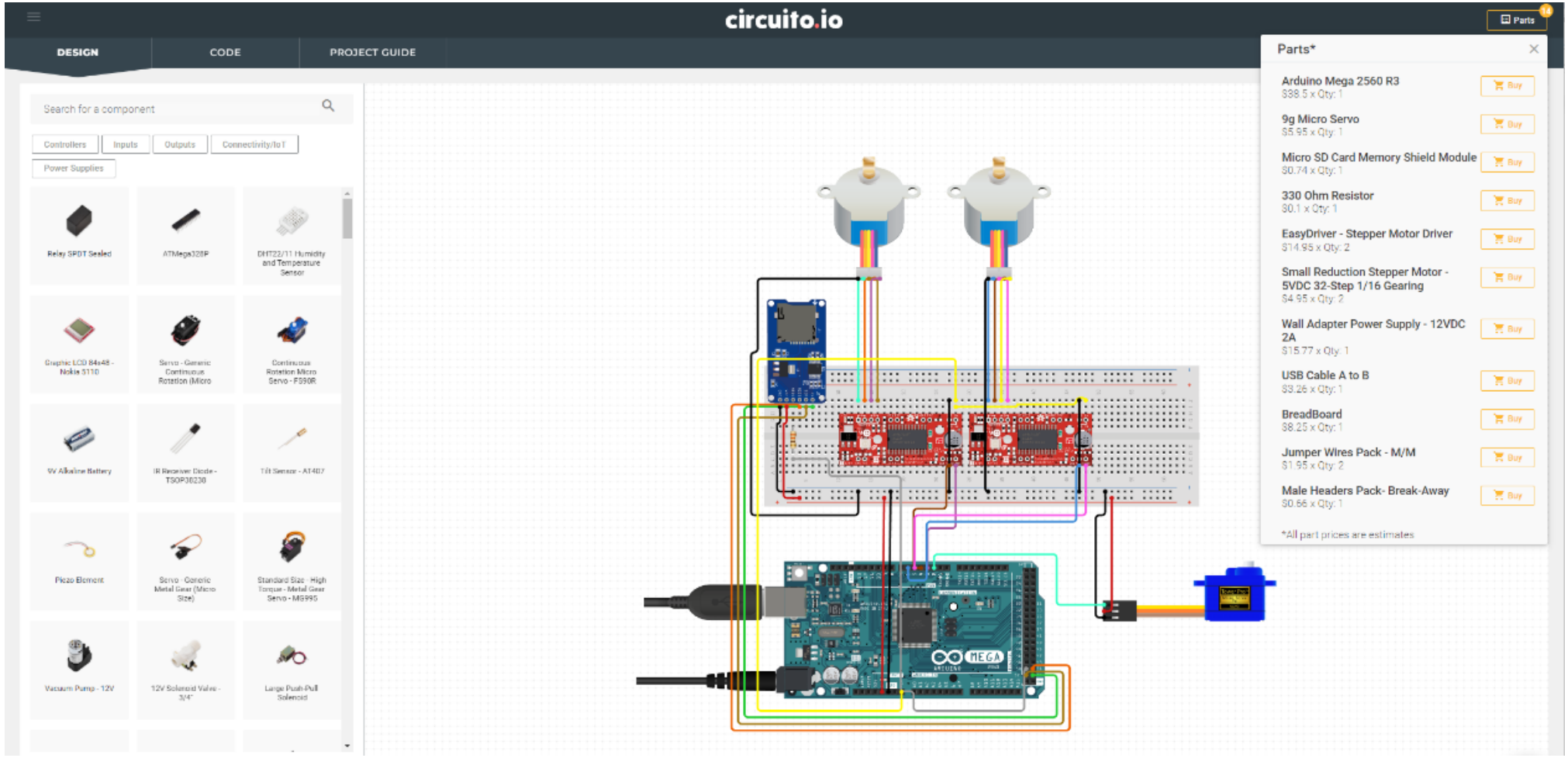
[23] Transforma tu pizarra sin usar en un plotter vertical. 27 marzo 2022, de Electrogeek. Sitio web https://www.electrogeekshop.com/transforma-tu-pizarra-sin-usar-en-un-plotter-vertical/

[24] Arduino vs Raspberry Pi. 4 septiembre 2019, de Jesús Lucas. Sitio web: https://openwebinars.net/blog/arduino-vs-raspberry-pi/#:~:text=Diferencias%20entre%20Arduino%20y%20Raspberry,podemos%20realizar%20con%20un%20ordenador.

[25] Plotter Vertical Con Arduino. 2022, de MCI electronics. Sitio web: https://www.instructables.com/Plotter-Vertical-Con-Arduino/

ANEXO A

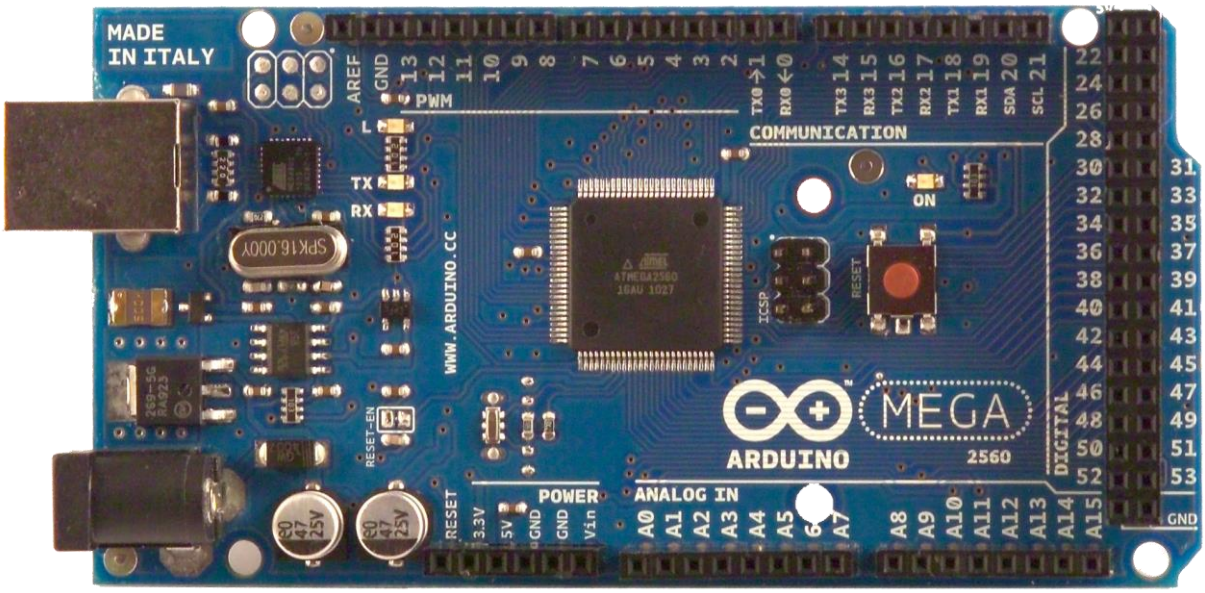
Diseño completo de la conexión de componentes.


circuito.io
DESIGN
CODE
PROJECT GUIDE
Parts
Search for a component
Controllers
Inputs
Outputs
Connectivity/IoT
Power Supplies
Relay SPDT Sealed
ATMega328P
DHT22/11 Humidity and Temperature Sensor
Graphic LCD 84x48 - Nokia 5110
Servo - Generic Continuous Rotation (Micro
Continuous Rotation Micro Servo - FS90R
9V Alkaline Battery
IR Receiver Diode - TSOP38230
Tilt Sensor - AT407
Piezo Element
Servo - Generic Metal Gear (Micro Size)
Standard Size - High Torque - Metal Gear Servo - MG995
Vacuum Pump - 12V
12V Solenoid Valve - 3/4"
Large Push-Pull Solenoid
Parts*
Arduino Mega 2560 R3
$38.5 x Qty: 1
Buy
9g Micro Servo
$5.95 x Qty: 1
Buy
Micro SD Card Memory Shield Module
$0.74 x Qty: 1
Buy
330 Ohm Resistor
$0.1 x Qty: 1
Buy
EasyDriver - Stepper Motor Driver
$14.95 x Qty: 2
Buy
Small Reduction Stepper Motor - 5VDC 32-Step 1/16 Gearing
$4.95 x Qty: 2
Buy
Wall Adapter Power Supply - 12VDC 2A
$15.77 x Qty: 1
Buy
USB Cable A to B
$3.26 x Qty: 1
Buy
BreadBoard
$8.25 x Qty: 1
Buy
Jumper Wires Pack - M/M
$1.95 x Qty: 2
Buy
Male Headers Pack- Break-Away
$0.66 x Qty: 1
Buy
*All part prices are estimates
MEGA

ANEXO B

Arduino MEGA 2560 Datasheet.

Arduino MEGA 2560



CE

Product Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Index

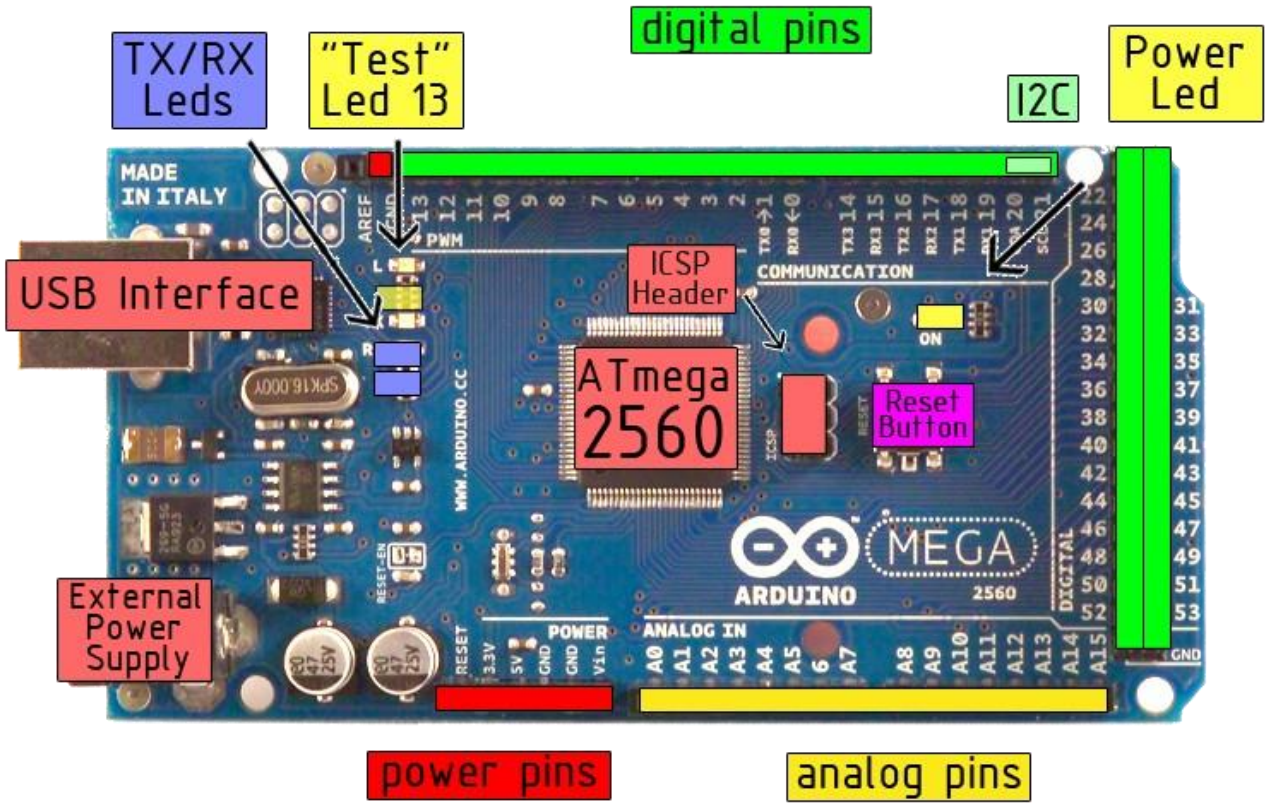
Technical Specification

EAGLE files: arduino-mega2560-reference-design.zip Schematic: arduino-mega2560-schematic.pdf

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

the board



Power

The Arduino Mega2560 can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **I^2C: 20 (SDA) and 21 (SCL).** Support I^2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I^2C pins on the Duemilanove.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().

Communication

- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Mega's digital pins.

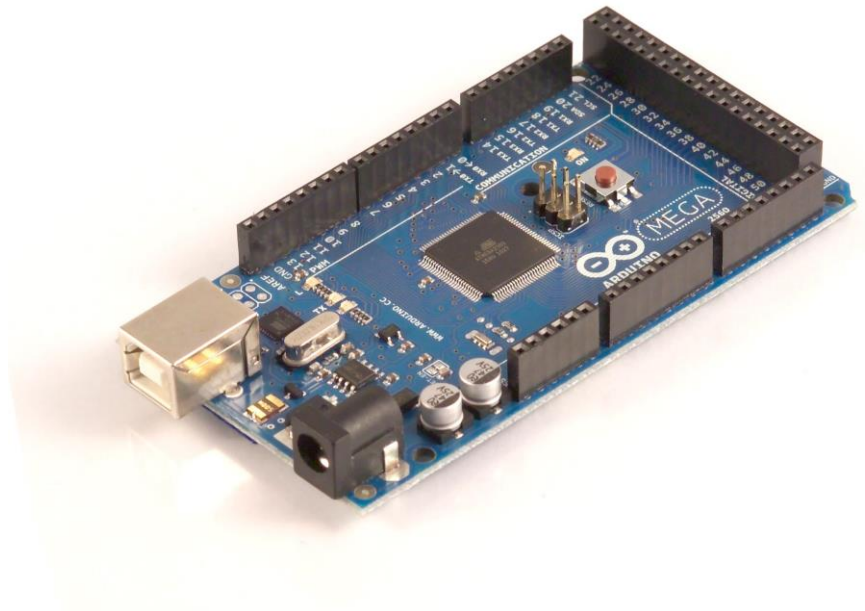
The ATmega2560 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. To use the SPI communication, please see the ATmega2560 datasheet.

Programming

The Arduino Mega2560 can be programmed with the Arduino software (download). For details, see the reference and tutorials.

The Atmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.



Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega is designed to be compatible with most shields designed for the Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega and Duemilanove / Diecimila. **Please note that I²C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).**

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your personal OS. Check on the Arduino site for the latest instructions. *http://arduino.cc/en/Guide/HomePage*

Linux Install

Windows Install

Mac Install

How to use Arduino

Once you have downloaded/unzipped the arduino IDE, you can Plug the Arduino to your PC via USB cable.

Blink led

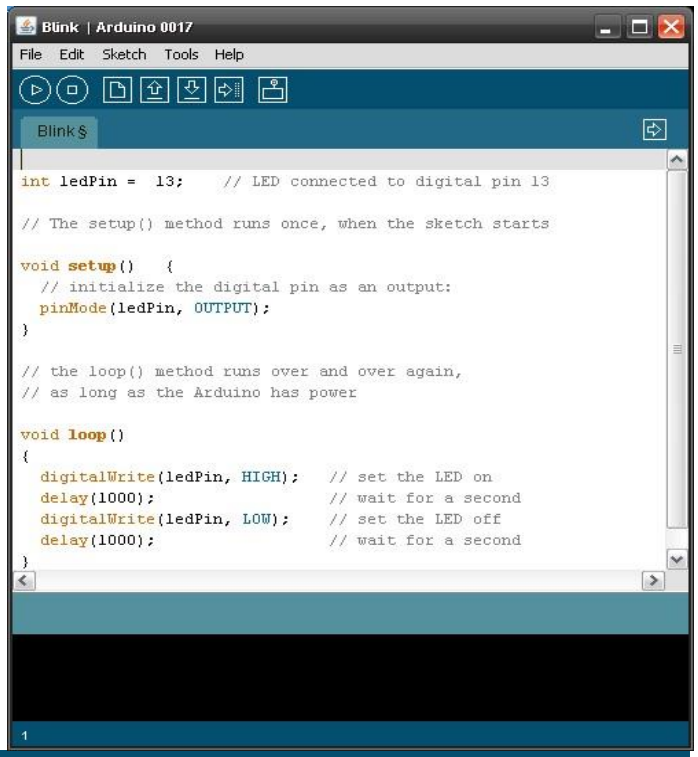
Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well known programming "hello world", select

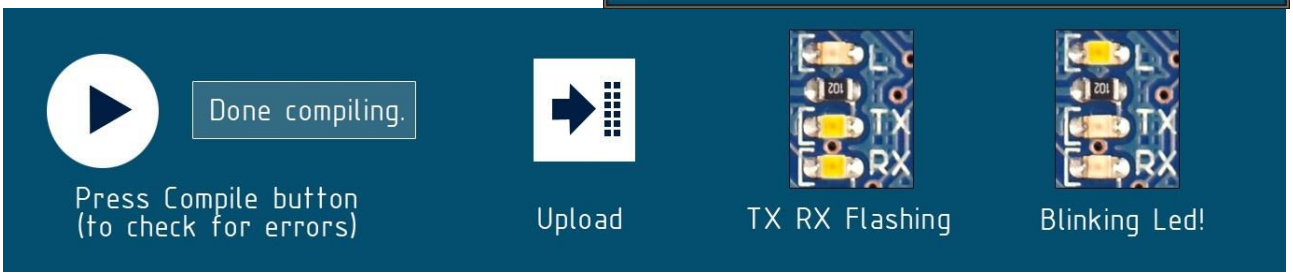
File>Sketchbook>
Arduino-0017>Examples>
Digital>Blink

Once you have your skecth you'll see something very close to the screenshot on the right.

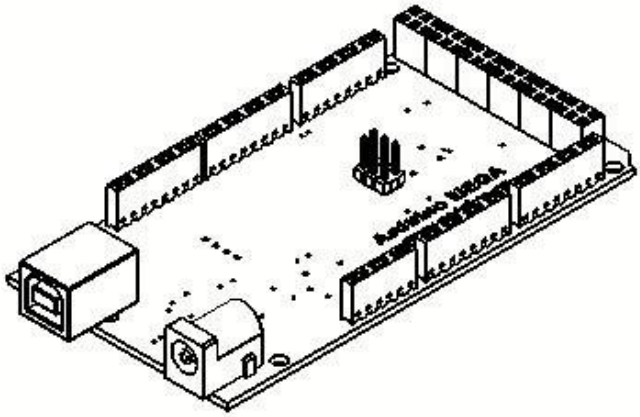
In **Tools>Board** select MEGA

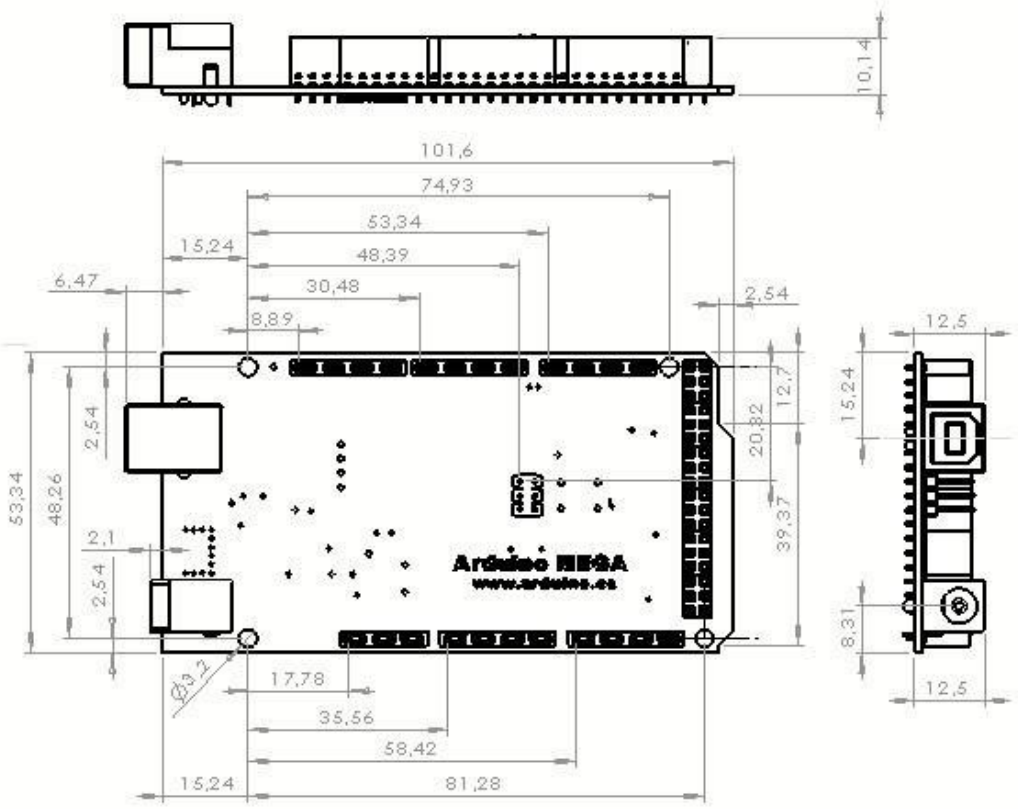
Now you have to go to
Tools>SerialPort
and select the right serial port, the one arduino is attached to.





Dimensioned Drawing





Terms & Conditions

1. Warranties

1.1 The producer warrants that its products will conform to the Specifications. This warranty lasts for one (1) years from the date of the sale. The producer shall not be liable for any defects that are caused by neglect, misuse or mistreatment by the Customer, including improper installation or testing, or for any products that have been altered or modified in any way by a Customer. Moreover, The producer shall not be liable for any defects that result from Customer's design, specifications or instructions for such products. Testing and other quality control techniques are used to the extent the producer deems necessary.

1.2 If any products fail to conform to the warranty set forth above, the producer's sole liability shall be to replace such products. The producer's liability shall be limited to products that are determined by the producer not to conform to such warranty. If the producer elects to replace such products, the producer shall have a reasonable time to replacements. Replaced products shall be warranted for a new full warranty period.

1.3 EXCEPT AS SET FORTH ABOVE, PRODUCTS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS." THE PRODUCER DISCLAIMS ALL OTHER
WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

1.4 Customer agrees that prior to using any systems that include the producer products, Customer will test such systems and the functionality of the products as used in such systems. The producer may provide technical, applications or design advice, quality characterization, reliability data or other services. Customer acknowledges and agrees that providing these services shall not expand or otherwise alter the producer's warranties, as set forth above, and no additional obligations or liabilities shall arise from the producer providing such services.

1.5 The Arduino™ products are not authorized for use in safety-critical applications where a failure of the product would reasonably be expected to cause severe personal injury or death. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Arduino™ products are neither designed nor intended for use in military or aerospace applications or environments and for automotive applications or environment. Customer acknowledges and agrees that any such use of Arduino™ products which is solely at the Customer's risk, and that Customer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

1.6 Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products and any use of Arduino™ products in Customer's applications, notwithstanding any applications-related information or support that may be provided by the producer.

2. Indemnification

The Customer acknowledges and agrees to defend, indemnify and hold harmless the producer from and against any and all third-party losses, damages, liabilities and expenses it incurs to the extent directly caused by: (i) an actual breach by a Customer of the representation and warranties made under this terms and conditions or (ii) the gross negligence or willful misconduct by the Customer.

3. Consequential Damages Waiver

In no event the producer shall be liable to the Customer or any third parties for any special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of the products provided hereunder, regardless of whether the producer has been advised of the possibility of such damages. This section will survive the termination of the warranty period.

4. Changes to specifications

The producer may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." The producer reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

The producer of Arduino ™ has joined the Impatto Zero® policy of LifeGate.it. For each Arduino board produced is created / looked after half squared Km of Costa Rica's forest's.