

Microcontroladores y Electrónica de Potencia

Trabajo integrador:

Impresora Braille



Alumno: Tassara Renzo

Legajo: 12299

Índice

Introducción.....	3
Braille.....	3
Esquema Tecnológico.....	5
Detalles de módulos.....	5
Motores paso a paso.....	5
Driver A4988.....	7
Servomotor SG90.....	8
Teclado matricial.....	9
Final de carrera.....	10
LCD 16x2.....	11
Funcionamiento general.....	11
Explicación de uso de impresora.....	11
Diagrama de bloques del código.....	12
Explicación de código.....	13
Libreria LCD.....	15
main.c.....	16
Variables.....	16
main().....	17
HAL_GPIO_EXTI_Callback().....	17
Manejo_Interrupciones.c.....	19
Variables.....	19
Manejo_Interrupciones().....	20
Texto.c.....	22
Variables.....	22
Texto().....	22
Matriz.c.....	23
Variables.....	23
Armado_Matriz().....	23
Movimiento_Matriz().....	25
Base_de_Datos.c.....	27
Movimiento_Motores.c.....	28
Variables.....	28



Movimiento_Motores()	29
Movimiento_Servo.c	30
Variables	30
Movimiento_Servo()	30
Homing.c	31
Variables	31
Homing()	31
Etapas de montaje y ensayos realizados	33
Estructura	33
Pruebas parciales	35
Drivers y motores PAP	36
Servomotor	38
Conexiones finales	40
Alimentación	41
Resultados, especificaciones finales	42
Metas y objetivos	42
Especificaciones técnicas	43
Precisión	43
Velocidad	43
Conclusiones. Ensayo de ingeniería de producto	44
Referencias	47
Anexos	48

Introducción

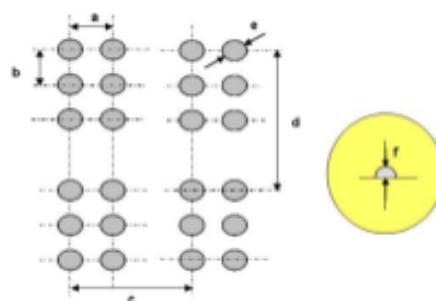
En la actualidad, las personas con discapacidad visual enfrentan dificultades para acceder a libros en formato braille, y solo algunos países cuentan con leyes que establecen la obligación de implementar métodos para que estas personas puedan acceder a cualquier tipo de texto. Es por ello que este proyecto se ha concebido con el objetivo de perfeccionar un sistema que facilite el acceso a impresoras braille, con la visión a largo plazo de integrarlo en impresoras 3D de fácil acceso y coste relativamente bajo, aprovechando de esta forma su mecánica. Esta iniciativa no solo busca eliminar la necesidad de utilizar tintas especiales o realizar mantenimientos complejos en las máquinas, sino también brindar una solución más versátil y eficiente.

Desde hace mucho tiempo, he sido consciente de los desafíos que enfrentan las personas con discapacidad visual, y es por eso que deseo ayudar con mis conocimientos en ingeniería para mejorar su calidad de vida. Aunque hay una variedad de proyectos que buscan hacer más accesible la lectura para estas personas, como los audiolibros y los traductores de texto a braille generados por inteligencia artificial, considero que muchas de las soluciones actuales no son siempre las más idóneas para algunas situaciones. Por ejemplo, no satisfacen las necesidades de aquellos que desean acceder a partes específicas de un texto, como es común en los libros de estudio, o también de aquellos que prefieren la tranquilidad de leer en silencio, sin depender de una voz generada por IA que les lea el contenido. Es en estos aspectos donde creo que se puede innovar y crear soluciones más eficientes y versátiles para estas personas.

Braille

Para el desarrollo de este proyecto, se consultó a una profesora especializada en la enseñanza a personas ciegas, cuya contribución ha sido fundamental. Gracias a su experiencia y conocimiento, se ha obtenido una comprensión más profunda del tema, incluyendo detalles como el tipo de papel utilizado, las dimensiones de los patrones, el método de lectura y, lo más crucial, el proceso de escritura en braille.

Se basó en las recomendaciones de esta profesora, así como en el estudio de libros dedicados al braille y en la información proporcionada por la Organización Nacional de Ciegos Españoles (ONCE), una entidad reconocida por su trabajo en favor de las personas con discapacidad visual. A través de la página web de la ONCE^{*1}, se pudo acceder a dos documentos importantes: el primero (B1) detalla las dimensiones de los patrones^{*2} (figura 1), mientras que el segundo (B2) describe cómo ensamblarlos para representar cada letra en braille^{*3} (figura 2)



a = Distancia horizontal entre los centros de puntos contiguos de la misma celda: de 2,4 a 2,75 mm.
 b = Distancia vertical entre los centros de puntos contiguos de la misma celda: de 2,4 a 2,75 mm.
 c = Distancia entre los centros de puntos idénticos de celdas contiguas: de 6 a 6,91 mm.
 d = Distancia entre los centros de puntos idénticos de líneas contiguas: 10 a 11,26 mm.
 e = Diámetro de la base de los puntos: entre 1,2 y 1,9 mm.
 f = Altura recomendada de los puntos: entre 0,5 y 0,2 mm.

Figura 1

a	b	c	d	e	f	g	h	i	j
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	12	14	145	15	124	1245	125	24	245
k	l	m	n	ñ	o	p	q	r	s
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
13	123	134	1345	12456	135	1234	12345	1235	234
t	u	v	w	x	y	z			
⋮	⋮	⋮	⋮	⋮	⋮	⋮			
2345	136	1236	2456	1346	13456	1356			

Figura 2

Esquema Tecnológico

En la figura 3 se mostrará un diagrama de bloques el cual representa cómo los diferentes módulos interactúan entre sí:

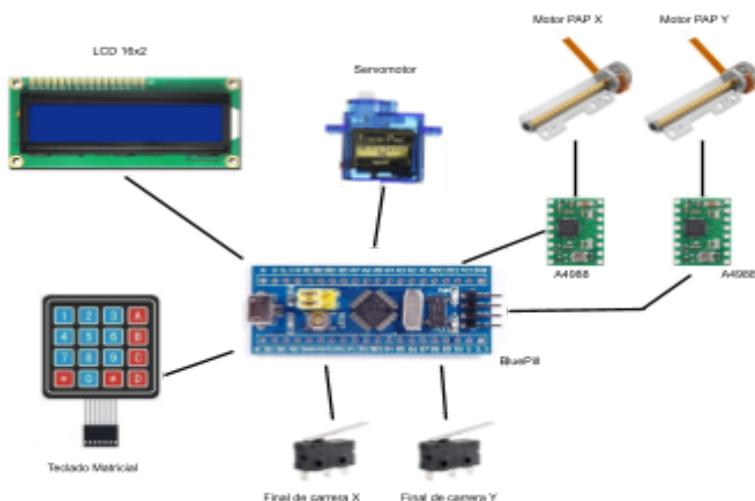


Figura 3

Detalles de módulos

En esta sección se describirán los componentes utilizados con una breve explicación del funcionamiento de cada uno.

Motores paso a paso

Aplicación: Se utilizan para generar el movimiento en el eje X e Y de la impresora.

Se optó por utilizar motores paso a paso (PAP) reciclados de lectores de DVD (figura 4) debido a varias razones. En primer lugar para poder reciclar componentes y aprovechar los recursos existentes. Además estos motores son fáciles de obtener y en este caso, gratuitos. La desventaja de dichos motores es que carecen de un datasheet que indique las características de los motores, como por ejemplo el

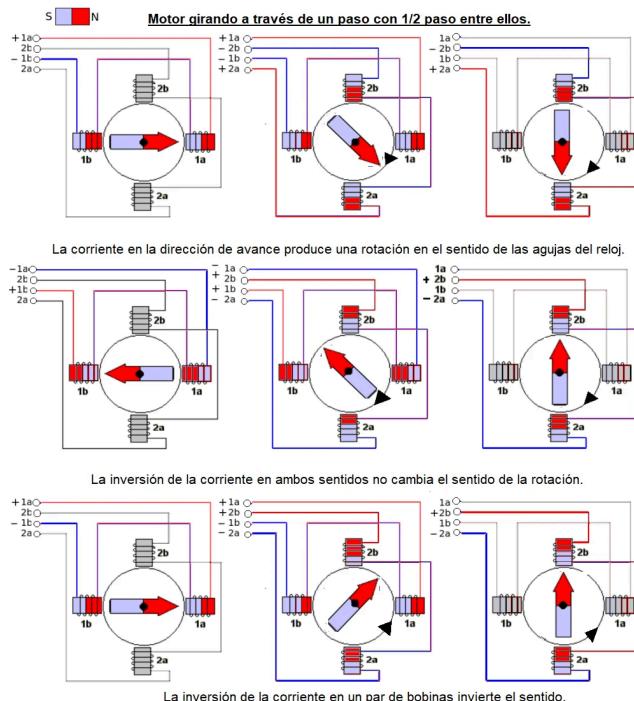
amperaje utilizado. Sin embargo se pudo ajustar el voltaje de referencia de los drivers de manera empírica, sin necesidad de realizar cálculos. Este procedimiento se explicará mejor en [Drivers y motores PAP](#)



Figura 4

Estos motores , originalmente utilizados para el movimiento del láser en el lector de DVD, ofrecen un torque adecuado para la aplicación, sin necesidad de sobredimensionar. Aunque los motores PAP no se destacan por sus velocidades elevadas, cuentan con elevada precisión en su posicionamiento, lo cual los convierte en la elección ideal para esta aplicación.

Los motores paso a paso operan mediante la activación secuencial de bobinas electromagnéticas, que generan campos magnéticos. Estos campos interactúan con los imanes permanentes del rotor, provocando su movimiento. En el caso de los motores paso a paso bipolares, la activación de dos bobinas por fase (o un par de bobinas opuestas) en secuencia controla el movimiento del rotor. El sentido de giro del rotor dependerá del sentido de circulación de corriente. Para una mejor comprensión del funcionamiento de dichos motores, se agregó una secuencia de imágenes que se ve reflejada en la figura 5.



*4

Figura 5

Driver A4988

Aplicación: Se emplearon dos drivers A4988, uno para el motor PAP que gestiona el eje X y otro para el eje Y.

Estos drivers (figura 6) ofrecen un mejor control de los motores, permiten regular el voltaje aplicado y permiten realizar movimientos por micropasos.



Figura 6

Cuando se envía un pulso al motor PAP sin driver, éste gira una cantidad específica de grados determinada por el diseño y las características del motor, dado que el flujo de corriente se genera en una sola una de las bobinas mientras que la otra permanece sin corriente. Sin embargo, mediante los micropasos, el driver es capaz de variar progresivamente esta diferencia de flujos de corriente entre las bobinas, permitiendo así un movimiento más suave y preciso.

Servomotor SG90

Aplicación: El servomotor se empleó para controlar el movimiento del punzón.

Analizando el datasheet^{*5}, éste servo (figura 7) funciona con una alimentación de 5V y su posición se controla mediante una señal de modulación de ancho de pulso (PWM) de 50Hz. Dependiendo del ciclo de trabajo (Duty Cycle) de la señal PWM, el servo puede moverse dentro de un rango de -90° a 90°.



Figura 7

Según el datasheet, para garantizar su correcto funcionamiento, teóricamente el ciclo de trabajo de la señal PWM debe variar entre un valor mínimo y un valor máximo (figura 8).:

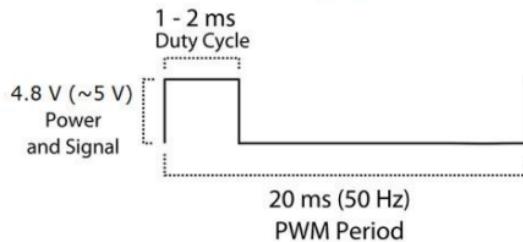


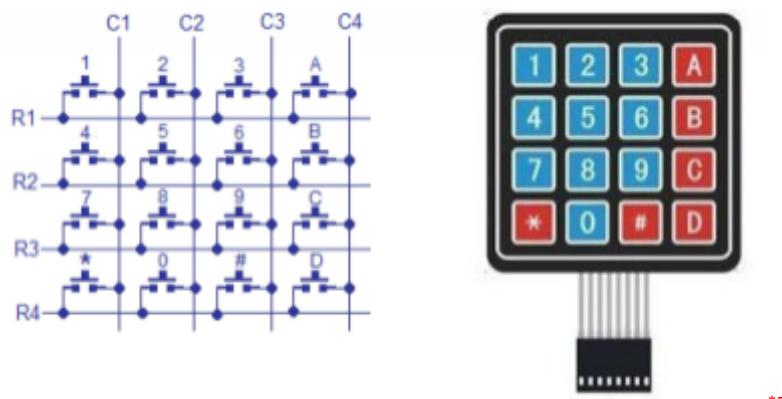
Figura 8

Como se puede apreciar en la imagen anterior, el Duty Cycle debe variar entre un 5% y un 10% para que gire los 180°.

Teclado matricial

Aplicación: El teclado matricial se utiliza para permitirle al usuario realizar las acciones necesarias para generar la impresión en braille deseada.

El circuito de este teclado consta de 16 pulsadores y se organiza en una matriz de 4 filas por 4 columnas, con un total de 8 pines (figura 9).



*7

Figura 9

La principal ventaja de utilizar un teclado matricial en lugar de pulsadores individuales es que se reduce la cantidad de pines necesarios para su conexión al microcontrolador. Sin embargo, es importante tener en cuenta que la velocidad de

procesamiento para determinar qué tecla se ha presionado puede ser más lenta debido a la naturaleza de la lectura secuencial de las columnas y filas.

El funcionamiento del teclado matricial es el siguiente: Cuatro pines están conectados a salidas digitales (OUTPUTS) y se utilizan para activar las columnas del teclado, mientras que otros cuatro pines están configurados como interrupciones externas (EXTI) y se utilizan para leer el estado lógico de las filas. Cuando se presiona alguna tecla, las columnas se activan secuencialmente y se realiza una comparación en cada iteración para determinar qué fila y columna están activadas simultáneamente, identificando así la tecla presionada.

Final de carrera

Aplicación: realizan la acción de homing, esto quiere decir que determina la posición inicial u "origen" de un sistema de coordenadas

Los finales de carrera (figura 10) utilizados fueron reciclados de una impresora de papel.

Consisten en un mecanismo que incluye un actuador mecánico y contactos eléctricos. Cuando se produce el contacto con el actuador del final de carrera, éste se activa mecánicamente cerrando el circuito provocando un cambio de estado. Éstos tienen la conexión normalmente abierta, lo cual significa que cuando no se encuentran presionados, estos se encuentran abiertos.



Figura 10

LCD 16x2

Aplicación: utilizado para generar la interfaz con el usuario que muestra el texto a imprimir, menú de las acciones que puede realizar el usuario, entre otras.

Un LCD (Liquid Crystal Display) funciona mediante la comunicación y el control de los segmentos de cristal líquido para mostrar información visual, como texto, números y gráficos simples. El LCD utilizado tiene una conexión paralela con el microcontrolador, ya que no se disponía del módulo I2C. Dicha conexión se puede ver en la figura 11

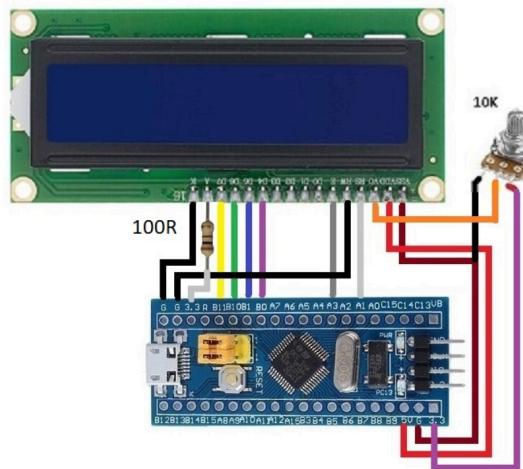


Figura 11

Funcionamiento general

Explicación de uso de impresora

1. Al comenzar, aparecerá un menú en la pantalla donde le indicará al usuario que acciones puede realizar:
 - a. Texto → Usuario podrá escribir texto

- b. Home → Impresora se irá al origen
 - c. Print → Comenzará a imprimir el texto escrito por el usuario
 - d. Back → Volver al menú principal si es que se encuentra en Texto
2. Si el usuario presiona la letra "A" en el teclado matricial, se le dará la opción de escribir el texto que desea imprimir. Este texto se ingresa utilizando los números del teclado, similar al método de escritura en los teléfonos celulares antiguos (figura 12).



Figura 12

- 3. A medida que el usuario escribe el texto, éste se mostrará en la pantalla.
- 4. Una vez que haya terminado de escribir, deberá presionar la tecla "D" para volver al menú principal.
- 5. Luego, para comenzar la impresión, el usuario presionará la tecla "C".
- 6. Antes de imprimir, la impresora se moverá a la posición inicial (homing) y luego iniciará la impresión.
- 7. Además de la impresión, el usuario tendrá la opción de realizar el homing sin necesidad de imprimir previamente. Esto lo logra estando en el menu principal y presionando la tecla "B".

Diagrama de bloques del código

La figura 13 muestra un diagrama de bloques el cual se ve como es el funcionamiento de la impresora. Dicho diagrama se ve qué estados puede tomar la impresora (bloques de color azul) y que estados pueden tomar los motores paso a

paso al momento de estar imprimiendo (bloques de color naranja) mientras que el bloque verde es el movimiento del servo motor.

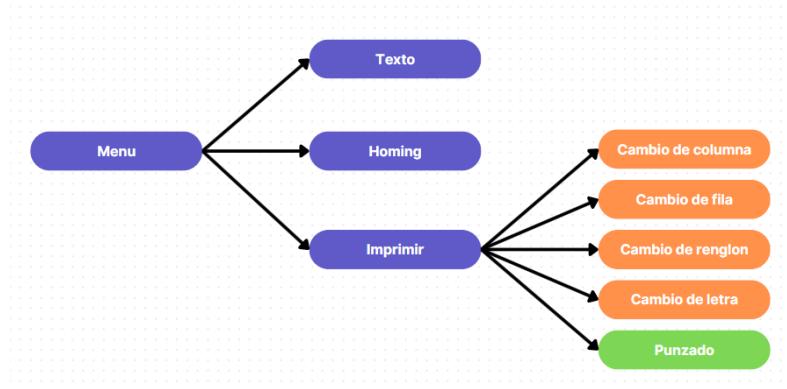


Figura 13

Explicación de código

Para el desarrollo del código se usó el IDE del fabricante del microcontrolador, stm32cubeIDE, el cual resulta sencillo de configurar los pines del microcontrolador, los timers, entre otras cosas.

En la figura 14 se muestra como se configuraron los pines del microcontrolador.

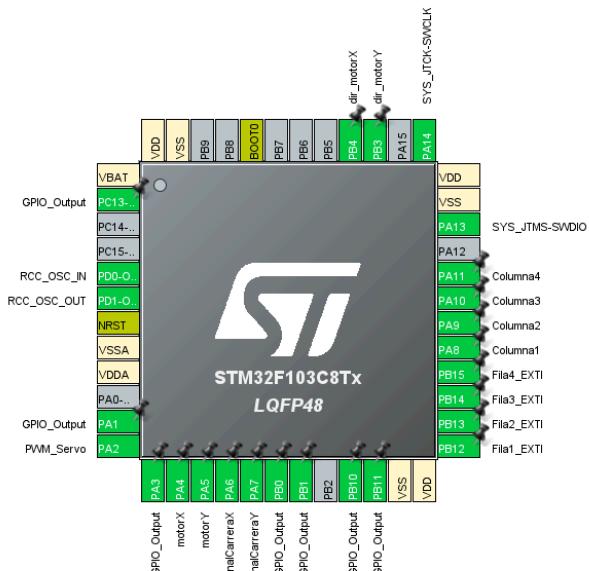


Figura 14

SYS_JTCK-SWCLK → proporciona el reloj para la comunicación de depuración y programación

SYS_JTMS-SWDIO → se utiliza para enviar y recibir datos entre el depurador y el microcontrolador

dir_motorX → Pin digital de salida que indicará la dirección del motor del eje X

dir_motorY → Pin digital de salida que indicará la dirección del motor del eje Y

ColumnaX → Pin digital de salida del teclado matricial

FilaN_EXTI → Pin digital de entrada (habilitado para interrupción externa) del teclado matricial

GPIO_OUTPUT → Pines digitales de salida asociados al LCD

FinalCarreraX → Pin digital de entrada (habilitado para interrupción externa) para el final de carrera del eje X

FinalCarreraY → Pin digital de entrada (habilitado para interrupción externa) para el final de carrera del eje Y

motorX → Pin digital de salida que genera los pulsos para mover el motor del eje X

motorY → Pin digital de salida que genera los pulsos para mover el motor del eje Y

PWM_Servo → Pin generador de señal PWM para el funcionamiento del servo

GPIO_Output (PC13) → Pin asociado al led incorporado en la placa del microcontrolador

Además de configurar los pines, se cambió la frecuencia del clock interno a una frecuencia más elevada, quedando así de 72MHz (figura 15). Esto permite una mayor precisión en el tiempo.

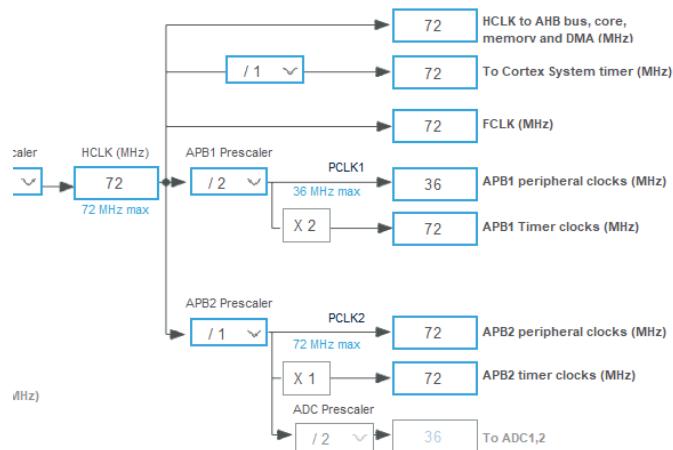


Figura 15

Libreria LCD

La librería utilizada establece una comunicación en paralelo entre el microcontrolador y la pantalla LCD, esto quiere decir que varios bits de datos se transfieren simultáneamente en cada ciclo de transferencia, utilizando varios pines GPIO para representar los bits individuales de datos y los bits de control. En este caso, se utiliza una comunicación paralela de 4 bits, lo que significa que se transfieren 4 bits de datos en cada operación de escritura.

Se optó por la comunicación paralela debido a, como se dijo anteriormente, la falta de disponibilidad del módulo I2C. Aunque esta forma de comunicación requiere más pines para la conexión y no permite conectar múltiples dispositivos en los mismos pines (como lo haría el módulo I2C, que posibilita conectar varios dispositivos y alternar la transferencia de datos), ofrece una ventaja significativa en términos de velocidad. La comunicación en paralelo es más rápida porque permite la transferencia simultánea de los bits de datos, a diferencia de la transferencia secuencial que se realiza en el protocolo I2C.

Las funciones posibles a realizar con esta librería son las siguientes:

Lcd_create: Esta función crea una nueva estructura Lcd_HandleTypeDef y configura el LCD.

Lcd_init: Inicializa el LCD con la configuración especificada en la estructura

Lcd_HandleTypeDef: Esto incluye la configuración de la interfaz de datos (4 bits o 8 bits), la limpieza de la pantalla, la activación del display y la configuración del modo de entrada.

Lcd_int: Convierte un número entero en una cadena de caracteres y la muestra en el LCD.

Lcd_string: Escribe una cadena de caracteres en la posición actual del cursor en el LCD.

Lcd_cursor: Establece la posición del cursor en el LCD para escribir caracteres.

Lcd_clear: Borra la pantalla del LCD y coloca el cursor en la posición inicial.

Lcd_define_char: Define un nuevo carácter personalizado para su uso en el LCD.

Función que se tuvo que agregar a la librería:

Lcd_char: Escribe un solo carácter en la posición actual del cursor en el LCD.

main.c

El main es el código principal, donde de allí se llamarán a todos los otros códigos para sus ejecuciones dependiendo de las acciones a realizar

Variables

Variables globales		
Tipo de dato	Variable	Uso
int8_t	interrupcion	Determina qué acción realizar dependiendo de la interrupción generada
uint32_t	miliseg_prev	Utilizado para evitar rebotes en las interrupciones
uint32_t	miliseg_actual	Utilizado para evitar rebotes en las interrupciones
enum	EstadoImpresora	Determina el estado de la impresora

main()

La función main (figura 16) se encarga de ejecutar acciones dependiendo del estado que se encuentra la variable “estado_impresora”. El estado “*ESPERAR*” es para que el usuario pueda realizar una acción mientras no se está ejecutando otra función. Si el estado es “*MENU*”, se habilita solamente la cuarta columna del teclado matricial, por lo tanto el usuario solo podrá presionar las telas A,B,C y D para poder cambiar el estado de “estado_impresora”.

```

while (1)
{
    if(estado_impresora == TEXTO || estado_impresora == IMPRIMIR || estado_impresora == HOME) {
        Manejo_Interrupciones(interrupcion);
        HAL_GPIO_WritePin(GPIOA, Columna1_Pin, 1); //Asigno valores a pines del teclado matriz
        HAL_GPIO_WritePin(GPIOA, Columna2_Pin, 1);
        HAL_GPIO_WritePin(GPIOA, Columna3_Pin, 1);
        HAL_GPIO_WritePin(GPIOA, Columna4_Pin, 1);
        if (estado_impresora == MENU) {
        }
        else{
            estado_impresora = ESPERAR;
        }
    }
    else if(estado_impresora == MENU) {
        Lcd_clear(&lcd);
        Lcd_cursor(&lcd, 0,0);
        Lcd_string(&lcd, "A)Text   B)Home");
        Lcd_cursor(&lcd, 1,0);
        Lcd_string(&lcd, "C)Print  D)Back");
        HAL_GPIO_WritePin(GPIOA, Columna1_Pin, 0); //Asigno valores a pines del teclado matriz
        HAL_GPIO_WritePin(GPIOA, Columna2_Pin, 0);
        HAL_GPIO_WritePin(GPIOA, Columna3_Pin, 0);
        HAL_GPIO_WritePin(GPIOA, Columna4_Pin, 1); //Solamente habilita las teclas A, B, C y D
        while(estado_impresora == MENU) {
        }
    }
    else if(estado_impresora == ESPERAR){ //Evita la actualización de la pantalla mientras no se
    }
}

```

Figura 16

HAL_GPIO_EXTI_Callback()

Esta función se ejecuta cada vez que hay una interrupción, o sea, una tecla del teclado matricial de las columnas que estén habilitadas o de los finales de carreras.

En la figura 17 se ve la primera parte de esta función la cual ingresa solo si el estado de la impresora es “*ESPERAR*” o “*TEXTO*”. Esta parte permite cambiar el estado de la impresora a “*TEXTO*” (si es que previamente se encontraba en “*ESPERAR*”) y luego poder mandar a la función “Manejo_Interrupcion” la variable

“interrupciones” para así poder realizar diferentes acciones en función de esta variable.

```
USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {

    miliseg_actual = HAL_GetTick();
    if ((miliseg_actual - miliseg_prev > 100)) {
        HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13); //Indicador visual para saber cuando entro a

        if (estado_impresora == ESPERAR || estado_impresora == TEXTO){
            if(GPIO_Pin == Fila1_EXTI_Pin && HAL_GPIO_ReadPin(GPIOB, Fila1_EXTI_Pin)){ //Tuv
                interrupcion = 1;
                estado_impresora = TEXTO;
            }
            else if(GPIO_Pin == Fila2_EXTI_Pin && HAL_GPIO_ReadPin(GPIOB, Fila2_EXTI_Pin)){
                interrupcion = 2;
                estado_impresora = TEXTO;
            }
            else if(GPIO_Pin == Fila3_EXTI_Pin && HAL_GPIO_ReadPin(GPIOB, Fila3_EXTI_Pin)){
                interrupcion = 3;
                estado_impresora = TEXTO;
            }
            if(GPIO_Pin == Fila4_EXTI_Pin && HAL_GPIO_ReadPin(GPIOB, Fila4_EXTI_Pin)){
                interrupcion = 4;
                estado_impresora = TEXTO;
                //estado_impresora = ATRAS;
            }
        }
    }
}
```

Figura 17

La figura 18 muestra la siguiente parte del código, la cual permite cambiar el estado de la impresora cuando previamente dicho estado se encontraba como “MENU”.

```
}
else if(estado_impresora == MENU){
    if(GPIO_Pin == Fila1_EXTI_Pin && HAL_GPIO_ReadPin(GPIOB, Fila1_EXTI_Pin)){ //Tuv
        estado_impresora = TEXTO;
    }
    else if(GPIO_Pin == Fila2_EXTI_Pin && HAL_GPIO_ReadPin(GPIOB, Fila2_EXTI_Pin)){
        estado_impresora = HOME;
    }
    else if(GPIO_Pin == Fila3_EXTI_Pin && HAL_GPIO_ReadPin(GPIOB, Fila3_EXTI_Pin)){
        estado_impresora = IMPRIMIR;
    }
    if(GPIO_Pin == Fila4_EXTI_Pin && HAL_GPIO_ReadPin(GPIOB, Fila4_EXTI_Pin)){
        estado_impresora = MENU;
    }
}
```

Figura 18

La última parte de esta función se encarga de aumentar los contadores homingX y homingY para realizar diferentes acciones en la función [Homing\(\)](#) y de generar un error y dejar de imprimir si es que se presiona alguno de los finales de carrera cuando el estado de la impresora se encuentra en “*IMPRIMIR*”.

```

        }
        if(GPIO_Pin == FinalCarreraX_Pin){
            HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
            if (estado_impresora == IMPRIMIR){
                while(1){} //Se debe reiniciar la impresora
            }
            else if(estado_impresora == HOME){
                if (homingX < 2)
                    homingX++; //Varia variable para saber si se ha llegado al final de carrera
                else{
                    homingX = 0;
                }
            }
        }
        else if(GPIO_Pin == FinalCarreraY_Pin){
            HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
            if (estado_impresora == IMPRIMIR){
                while(1){} //Se debe reiniciar la impresora
            }
            else if(estado_impresora == HOME){
                if (homingY < 2)
                    homingY++;
                else{
                    homingY = 0;
                }
            }
        }
    }
}

```

Figura 19

Manejo_Interrupciones.c

Esta función se encargará de decidir qué acción debe ejecutarse dependiendo de la interrupción generada anteriormente

Variables

Variables		
Tipo de dato	Variable	Uso

uint8_t	filas	Indica cantidad de filas de matriz
uint8_t	columnas	indica cantidad de columnas de matriz
uint8_t	i	Posición de carácter en texto[]
char	texto[]	String donde se escribe el texto a imprimir
uint8_t	cont_espacios	Verifica que "texto" no esté vacía

Manejo_Interrupciones()

Si el estado de la impresora se encuentra como “TEXTO” entonces se empezará a cambiar el valor lógico de cada columna de la matriz de forma secuencial para saber qué pulsador se ha presionado (figura 20). Dependiendo de esto, es el valor que recibirá la función [Texto\(\)](#) y la letra que se escribirá o la acción a realizar (borrar letra o seleccionar letra y pasar a la siguiente).

```

HAL_GPIO_WritePin(GPIOA, Columna1_Pin, 1); //Habilito primera fila
HAL_GPIO_WritePin(GPIOA, Columna2_Pin, 0);
HAL_GPIO_WritePin(GPIOA, Columna3_Pin, 0);
HAL_GPIO_WritePin(GPIOA, Columna4_Pin, 0);

if(pin == 1 && HAL_GPIO_ReadPin(GPIOB, Fila1_EXTI_Pin)){ //Aqui se lee el pin
    texto[i] = Texto(1);
    Lcd_clear(&lcd);
    Lcd_cursor(&lcd, 0,0);
    Lcd_string(&lcd, "Escriba el texto");
    Lcd_cursor(&lcd, 1,0);
    Lcd_string(&lcd, texto);

}
else if(pin == 2 && HAL_GPIO_ReadPin(GPIOB, Fila2_EXTI_Pin)){
    texto[i] = Texto(4);
    Lcd_clear(&lcd);
    Lcd_cursor(&lcd, 0,0);
    Lcd_string(&lcd, "Escriba el texto");
    Lcd_cursor(&lcd, 1,0);
    Lcd_string(&lcd, texto);

}
else if(pin == 3 && HAL_GPIO_ReadPin(GPIOB, Fila3_EXTI_Pin)){
    texto[i] = Texto(3);
    Lcd_clear(&lcd);
    Lcd_cursor(&lcd, 0,0);
    Lcd_string(&lcd, "Escriba el texto");
    Lcd_cursor(&lcd, 1,0);
    Lcd_string(&lcd, texto);

}

```

Figura 20

Si el estado de la impresora se cambia a “IMPRIMIR”, primero se verificará que el usuario haya escrito algo. De ser así, se procede a ingresar a la función [Armado_Matriz\(\)](#) con las variables necesarias (figura 21)

```

else if(estado_impresora == IMPRIMIR) {

    for(int i = 0; i < sizeof(texto); i++){ //Co
        if (texto[i] == ' '){
            cont_espacios++;
        }
    }
    if (cont_espacios == sizeof(texto)-1){
        Lcd_clear(&lcd);
        Lcd_cursor(&lcd, 0,1);
        Lcd_string(&lcd, "Escriba algo");
        HAL_Delay(1000);
        cont_espacios = 0;
        estado_impresora = MENU;
    }
    else{
        Lcd_clear(&lcd);
        Lcd_cursor(&lcd, 0,1);
        Lcd_string(&lcd, "Imprimiendo...");
        Lcd_cursor(&lcd, 1,1);
        Lcd_int(&lcd, sizeof(texto));
        Lcd_cursor(&lcd, 1,10);
        Lcd_int(&lcd, cont_espacios);

        Armado_Matriz(texto, filas, columnas);

        Lcd_clear(&lcd);
        Lcd_cursor(&lcd, 0,1);
        Lcd_string(&lcd, "Retire la hoja");
        HAL_Delay(5000);
        estado_impresora = MENU;
    }
}

```

Figura 21

Si el estado de la impresora es “Home”, procede a ingresar a la función [Homing\(\)](#), el cual realiza los movimientos necesarios para colocar el punzón en la esquina del papel.

```

else if (estado_impresora == HOME)
    estado_impresora = HOME;
    Lcd_clear(&lcd);
    Lcd_cursor(&lcd, 0,1);
    Lcd_string(&lcd, "Homing...");
    Homing();
    Lcd_clear(&lcd);
    Lcd_cursor(&lcd, 0,1);
    Lcd_string(&lcd, "Home");
    Lcd_cursor(&lcd, 1,1);
    Lcd_string(&lcd, "finalizado");
    HAL_Delay(2000);
    estado_impresora = MENU;
}

```

Figura 22

Texto.c

Esta función se encarga de seleccionar la letra correspondiente dependiendo la tecla pulsada y la cantidad de veces que se pulsó.

Variables

Variables		
Tipo de dato	Variable	Uso
uint8_t	cont[10]	contadores para indicar cuantas veces se presiono la tecla
char	caracter	indica caracter a devolver

Texto()

En la figura 23 se muestra una parte del código de “Texto()” el cual es el encargado de devolver el carácter correspondiente a la función

[Manejo_Interrupciones\(\)](#)

```
char Texto(int numero) {
    switch(numero){
        case(0):
            if (cont[0] == 0){
                caracter = ' ';
                return caracter;
            }
            //el caso solo se usa para
            break;

        case(1):
            break;
        case(2):
            if (cont[2] == 0){
                caracter = 'a';
                cont[2]++;
                return caracter;
            }
            else if(cont[2] == 1){
                caracter = 'b';
                cont[2]++;
                return caracter;
            }
            else if(cont[2] == 2){
                caracter = 'c';
                cont[2] = 0;
                return caracter;
            }
            break;
        case(3):
    }
}
```

Figura 23

Como se pudo ver, presionando el 0, solo se genera el espacio, el 1 no genera ningún carácter, si se presiona una vez el “2” se escribe la letra “a”, dos veces la “b”, tres veces la “c” y cuatro veces vuelve a escribir la “a” y, como se vio anteriormente, `texto[i] = Texto(...)`, por lo que tomará el valor de lo que devuelve esta función. Esta metodología se usó para el resto de las teclas. Sin embargo, cuando se presiona “#” se resetea el contador (`cont[0]`) y devuelve el último carácter seleccionado previamente (figura 24).

```

case(11):
    for (int k = 0; k <= 9;k++){
        cont[k] = 0;
    }
    return caracter;

```

Figura 24

Matriz.c

Variables

Variables		
Tipo de dato	Variable	Uso
<code>uint8_t</code>	<code>m</code>	Cambio de submatriz
<code>uint8_t</code>	<code>filasSubmatriz</code>	Cantidad de filas de patrón Braille por letra
<code>uint8_t</code>	<code>columnasSubmatriz</code>	Cantidad de columnas de patrón Braille por letra
<code>uint8_t</code>	<code>1</code>	Contador de cantidad de “1” en la matriz

Armado_Matriz()

Esta función es la encargada de crear una matriz de ceros y luego cambiar algunos elementos a “1” según corresponda. Si hay un “0”, entonces en esa

posición no hay que hacer un punzado en el papel y si hay un “1” si lo hace (figura 25).

```
void Armado_Matriz(char texto[], int filasMatriz, int columnasMatriz) { //  
    // Creo la matriz de ceros  
    uint8_t matriz[filasMatriz][columnasMatriz];  
  
    //Completo con 1 y 0 la matriz segun el texto escrito  
    for (int i = 0; i < filasMatriz; i += filasSubmatriz) [  
        for (int j = 0; j < columnasMatriz; j += columnasSubmatriz) {  
            m++; //Cambio de submatriz (letra)  
            for (int k = 0; k < filasSubmatriz; k++) {  
                for (int l = 0; l < columnasSubmatriz; l++) {  
                    matriz[i + k][j + l] = Base_de_Datos(texto[m-1], k, l);  
                }  
            }  
        }  
    }  
    m = 0; //reinicio la variable m  
    Movimiento_Matriz(filasMatriz, columnasMatriz, matriz);  
}
```

Figura 25

Como se puede ver, se completan con “1” y “0” analizando submatrices de 3x2 (tamaño del patrón braille) y los caracteres de “texto”. Para una mejor comprensión, en la figura 26 se ve como es el recorrido para completar la matriz.

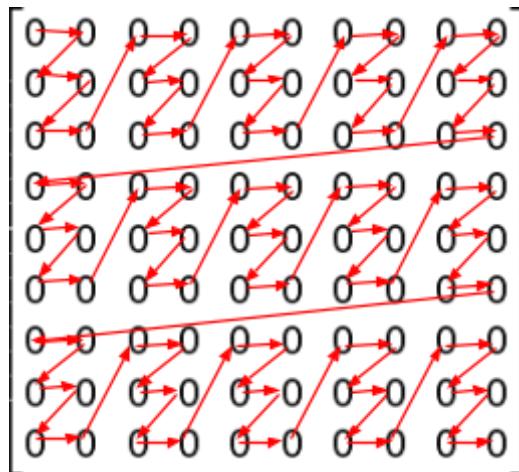


Figura 26

Movimiento_Matriz()

Esta función es la encargada de ejecutar las acciones correspondientes según el valor de los elementos de la matriz y de la posición.

Primero se realiza la acción de “Homing”, el cual se debió cambiar el estado de la impresora para que se realice correctamente. Esta acción se ejecuta antes de realizar la impresión por si el usuario no hizo homing previamente o movió los motores de forma manual. Luego se separa de los finales de carrera (offset) y se cambia el estado de la impresora (figura 27).

```
void Movimiento_Matriz(int filasMatriz, int columnasMatriz,uint8_t matriz[filasMatriz][columnasMatriz]) {
    estado_impresora = HOME; //Realizo Homing antes de comenzar la impresion
    Homing();

    //Pegueño movimiento para separarse de los finales de carrera ('OFFSET')
    HAL_GPIO_WritePin(GPIOB, dir_motorX_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, dir_motorY_Pin, GPIO_PIN_RESET);
    Movimiento_Motores(CAMBIO_FILA);
    Movimiento_Motores(CAMBIO_COLUMNNA);
    HAL_GPIO_WritePin(GPIOB, dir_motorX_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, dir_motorY_Pin, GPIO_PIN_RESET);
    estado_impresora = IMPRIMIR;
```

Figura 27

Luego empieza la impresión, el cual se alterna el estado de los motores según los desplazamiento que deba hacer (figura 28).

```
for (int i = 0; i < filasMatriz; i++) {
    if (contador_fila == 3){
        Movimiento_Motores(CAMBIO_RENGLON);
        contador_fila = 0;
    }
    else{
        Movimiento_Motores(CAMBIO_FILA);
        contador_fila++;
    }
    if ((i == 0) || (i%2 == 0)){
        for (int j = 0; j < columnasMatriz; j++){
            if ((j == 0) || (j%2 == 0)){
                Movimiento_Motores(CAMBIO_COLUMNNA);
            }
            else if (j%2 != 0){
                Movimiento_Motores(CAMBIO_LETRA);
            }
        }
    }
}
```

Figura 28

Una de las formas más eficientes de realizar el desplazamiento es por filas, como se muestra en la figura 29

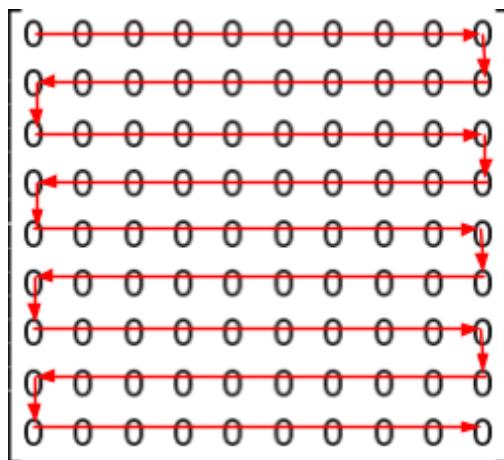


Figura 29

Luego de cada movimiento, se verifica si la matriz contiene un “1” en esa posición. De ser así, se procede a realizar el punzado (figura 30). Luego de realizarlo, verifica si tiene que realizar más punzados. De no ser así, finaliza la impresión.

```

if ((i == 0) || (i%2 == 0)){
    for (int j = 0; j < columnasMatriz; j++)[]

        if ((j == 0) || (j%2 == 0)){
            Movimiento_Motores(CAMBIO_COLUMNAS);
        }
        else if (j%2 != 0){
            Movimiento_Motores(CAMBIO_LETRA);
        }

        if (matriz[i][j] == 1){
            Movimiento_Servo();
            matriz[i][j] = 0;
            for (int fila = 0; fila < filasMatriz; fila++) { //Corroborar
                for (int columna = 0; columna < columnasMatriz; columna++) {
                    if (matriz[fila][columna] == 1) {
                        contador_elementos++;
                    }
                }
            }
            if (contador_elementos == 0){
                i = filasMatriz - 1;
                j = columnasMatriz - 1;
                estado_impresora = HOME;
            }
            else{
                contador_elementos = 0;
            }
        }
}

```

Figura 30

Luego procede a hacer homing (figura 31).

```

    Homing(); //Realizo Homing()
    estado_impresora = IMPRIMIR;

```

Figura 31

Base_de_Datos.c

Analizando el bucle “*for*” en Armado_Matriz() se puede ver que dependiendo del carácter actual que está analizando el bucle, el valor de “*k*” y el valor de “*l*”, es el valor que obtendrá el elemento de la matriz (figura 32):

```

int Base_de_Datos(char caracter,int k, int l){
    switch (caracter){

```

Figura 32

Por ejemplo si analizamos el caso en que el carácter actual es ‘h’ (figura 33), entonces la submatriz quedará de tal forma que los “1” corresponden a los sectores donde se hace el punzado y los “0” donde no se hace nada, tal como indica la figura 34 la cual fue detallada en la sección [Braille](#).

```

case('h'):
    if (k == 0 && l == 0){
        return 1;
    }
    else if (k == 0 && l == 1){
        return 0;
    }
    else if (k == 1 && l == 0){
        return 1;
    }
    else if (k == 1 && l == 1){
        return 1;
    }
    else if (k == 2 && l == 0){
        return 0;
    }
    else if (k == 2 && l == 1){
        return 0;
    }
    break;
}

```

Figura 33

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \quad \begin{matrix} h \\ \bullet \\ \bullet \end{matrix}$$

Figura 34

Movimiento_Motores.c

Aquí se encuentra la función “Movimiento_Motores()”, la cual es la responsable de los movimientos y direcciones de los motores X e Y.

Variables

Variables		
Tipo de dato	Variable	Uso
const uint16_t	pxmmX	Pasos/mm del motor X
const uint16_t	pxmmY	Pasos/mm del motor Y
uint8_t	velocidad	describe la velocidad de motores PAP
const uint8_t	a	Distancia entre puntos según La ONCE
const uint8_t	b	Distancia entre puntos según La ONCE

const uint8_t	c	Distancia entre puntos según La ONCE
const uint8_t	d	Distancia entre puntos según La ONCE

Movimiento_Motores()

Dependiendo del estado en que se encuentren los motores PAP es el movimiento que realizará (figura 35).

```
void Movimiento_Motores(int accion){
    switch(accion){
        case CAMBIO_RENGLON: //Cambio de renglon
            for (int i = 0; i < pxmmY*d; i++){
                HAL_GPIO_TogglePin(GPIOA, motorY_Pin);
                HAL_Delay(velocidad);
            }
            //break; //break esta comentado porque genera un error en la compilacion

        case CAMBIO_FILA: //Cambio de fila
            HAL_GPIO_TogglePin(GPIOB, dir_motorX_Pin); //
            for (int i = 0; i < pxmmY*b; i++){
                HAL_GPIO_TogglePin(GPIOA, motorY_Pin);
                HAL_Delay(velocidad);
            }
            //break;

        case CAMBIO_COLUMNAS: //Cambio de columna
            for (int i = 0; i < pxmmX*a; i++){
                HAL_GPIO_TogglePin(GPIOA, motorX_Pin);
                HAL_Delay(velocidad);
            }
            //break;

        case CAMBIO_LETRA://Cambio de letra
            for (int i = 0; i < pxmmX*(c-a); i++){
                HAL_GPIO_TogglePin(GPIOA, motorX_Pin);
                HAL_Delay(velocidad);
            }
            //break;
    }
}
```

Figura 35

Las distancias recorridas son las correspondientes a las que se explicaron en la sección [Braille](#) del presente informe. Para esto se definieron las variables ingresando el valor en milímetros según indica la documentación de La ONCE^{*2} y

luego se multiplican a la variable que indica los pasos por milímetro de cada motor cuando los drivers están conectados de tal forma que genere microsteps de 1/16.

Movimiento_Servo.c

Variables

Variables		
Tipo de dato	Variable	Uso
uint8_t	vel_servo	Variable dentro de delay que representa velocidad
uint16_t	t_reposo	Tiempo de reposo del punzón sobre hoja
uint16_t	pos_inicial	Posición inicial de servo
uint16_t	pos_actual	Posición actual del servo
uint16_t	por_final	Posición final donde hace contacto con la hoja

Movimiento_Servo()

Esta función se encarga del movimiento del servomotor para realizar el punzado (figura 36).

```
void Movimiento_Servo() {
    pos_actual = pos_inicial;

    while(pos_actual <= pos_final){
        HAL_Delay(vel_servo);
        TIM2->CCR3 = pos_actual;
        pos_actual += 5;      //Incremento
    }

    HAL_Delay(t_reposo);

    while (pos_actual >= pos_inicial){
        HAL_Delay(vel_servo);
        TIM2->CCR3 = pos_actual;
        pos_actual -= 5;      //Decremento
    }
}
```

Figura 36

Primero se actualiza la variable posición actual. Luego entra al primer bucle “while” donde el Duty Cycle incrementará. Dicha velocidad de incremento dependerá de dos valores, “vel_servo” y de pos_inicial += 5.

La variable “pos_final” representará la presión con la que el punzón deja la marca sobre la hoja, por lo tanto al aumentar este valor, más notable será la marca. Sin embargo hay que tener cuidado con dicho valor ya que al aumentarlo mucho, el punzón ejercerá mucha fuerza a la base donde se encuentra la hoja, pudiendo romper alguna pieza.

Finalmente el servo volverá a su posición inicial, donde el bucle “while” es el mismo que el anterior, pero la variable “pos_actual” decrementará hasta llegar a la posición inicial.

Homing.c

Este archivo contiene la función encargada de realizar la acción “Homing”. Dicha acción sirve para referenciar el punzón y poder comenzar con la impresión en la posición correcta.

Variables

Variables		
Tipo de dato	Variable	Uso
uint8_t	homingX	indicador logico de final de carrera X
uint8_t	homingY	indicador logico de final de carrera Y
uint16_t	velocidad_homing	describe la velocidad de motores PAP

Homing()

Al comenzar se inicializan las variables “homingX” y “homingY” iguales a “1” y se setean las direcciones de los motores PAP para que vayan en dirección a los

finales de carrera. Al presionar los finales de carrera se produce una interrupción la cual incrementa la variable correspondiente (figura 37).

```

void Homing(){
    homingX = 1;
    homingY = 1;

    //=====Homing rápido=====
    HAL_GPIO_WritePin(GPIOB, dir_motorX_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOB, dir_motorY_Pin, GPIO_PIN_RESET);

    while(homingX == 1 || homingY == 1){
        if (homingX == 1){
            HAL_GPIO_TogglePin(GPIOA, motorX_Pin);
        }
        if (homingY == 1){
            HAL_GPIO_TogglePin(GPIOA, motorY_Pin);
        }
        HAL_Delay(velocidad1_homing);
    }
}

```

Figura 37

Una vez que ambas variables sean iguales a “2”, se procede a hacer un cambio de sentido de los motores alejándose una determinada distancia y luego se hace un acercamiento fino hacia los finales de carrera con una menor velocidad (figura 38).

```

//==Retroceso==
HAL_GPIO_WritePin(GPIOB, dir_motorX_Pin, GPIO_PIN_SET); //
HAL_GPIO_WritePin(GPIOB, dir_motorY_Pin, GPIO_PIN_SET);
for (int i = 0; i < 1000; i++){
    HAL_GPIO_TogglePin(GPIOA, motorX_Pin);
    HAL_GPIO_TogglePin(GPIOA, motorY_Pin);
    HAL_Delay(velocidad1_homing);
}

//==Homing lento==
HAL_GPIO_WritePin(GPIOB, dir_motorX_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(GPIOB, dir_motorY_Pin, GPIO_PIN_RESET);

while(homingX == 2 || homingY == 2){
    if (homingX == 2){
        HAL_GPIO_TogglePin(GPIOA, motorX_Pin);
    }
    if (homingY == 2){
        HAL_GPIO_TogglePin(GPIOA, motorY_Pin);
    }
    HAL_Delay(velocidad2_homing);
}

```

Figura 38

Etapas de montaje y ensayos realizados

En esta sección del informe, se detallan las diversas etapas de montaje y los ensayos realizados como parte del proceso de desarrollo de la “Impresora Braille”. El montaje y los ensayos cumplen un rol muy importante en la evaluación del rendimiento y la calidad final de la impresora.

En primer lugar, se presenta una descripción general de las etapas de montaje, la cual describe cómo se hicieron las diferentes piezas. Posteriormente, se detallan los diferentes ensayos realizados para evaluar el funcionamiento y la fiabilidad de la impresora.

Estructura

Se imprimieron en 3D todas las piezas para el armado de la estructura (figura 39).



Figura 39

Este diseño ha sido descargado a través de la página thingiverse.com^{*1} el cual originalmente es para realizar una CNC Plotter con láser.

Luego se imprimió el mecanismo para utilizar el servo como punzón (figura 40), también descargado a través de la página thingiverse.com^{*2}. Dicho mecanismo

cuenta con un engrane el cual desplaza linealmente una cremallera. En el extremo de dicha cremallera, se colocó la punta de una lapicera, ya que es metálica, resistente y cumple con las dimensiones del punzado según La ONCE (detallado en la sección de [Braille](#))

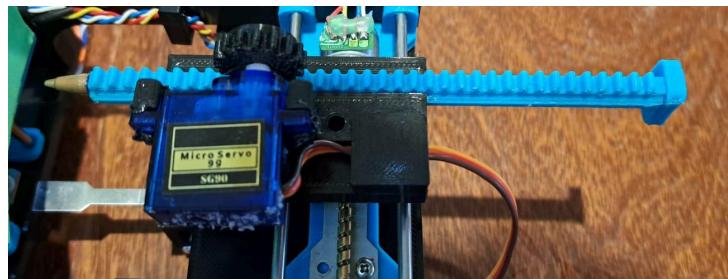


Figura 40

Además se ha diseñado una estructura para que todas las conexiones queden dentro de una caja y la impresora quede sobre esta. Este diseño se hizo en Solid Edge como se muestra en la figura 41 . El diseño cuenta con seis piezas las cuales se ensamblan unas con otras para formar las cajas.

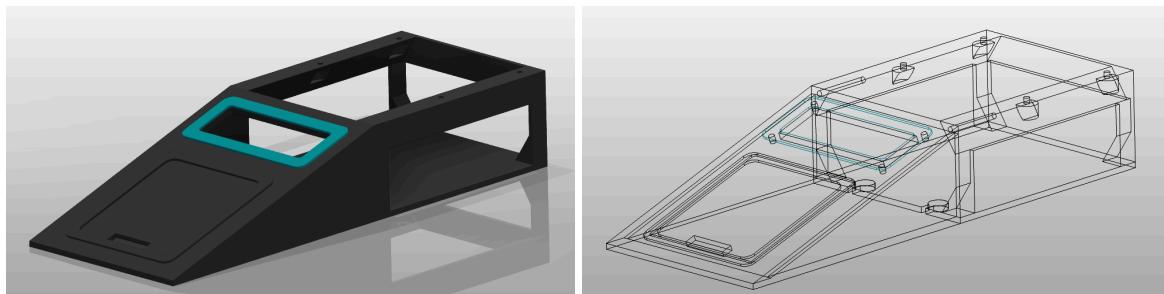


Figura 41

El patrón que se encuentra en los costados se realizó buscando la imagen en Google, luego se convirtió el archivo de .jpg a .svg el cual permitió trabajarla en tinkercad.com y se modificó según las medidas necesarias (figura 42)

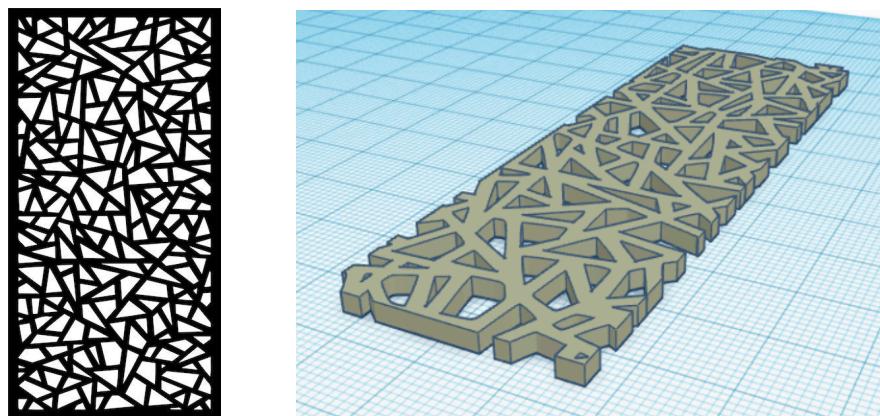


Figura 42

Además se tuvo que diseñar una pieza la cual sostiene los finales de carrera (figura 43). Dicha pieza también se diseñó en Solid Edge.

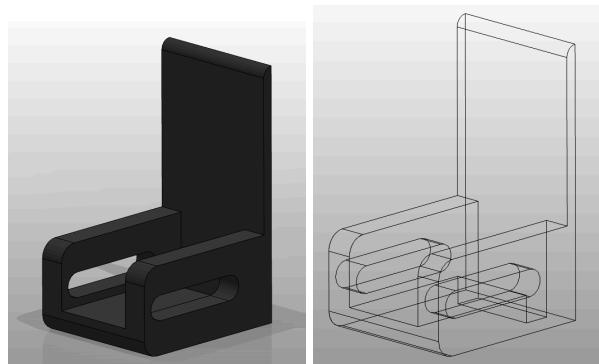


Figura 43

Pruebas parciales

Primero se realizaron conexiones las cuales permitieron hacer todas las pruebas necesarias para calibrar y saber cómo funcionaban los diferentes componentes utilizados.

Drivers y motores PAP

Se configuraron los micropasos a 1/16 mediante la conexión de los pines ms1, ms2 y ms3 como se especifica en la tabla (figura 44) extraída del datasheet del controlador ⁶:

MS1	MS2	MS3	Microstep Resolution	Excitation Mode
L	L	L	Full Step	2 Phase
H	L	L	Half Step	1-2 Phase
L	H	L	Quarter Step	W1-2 Phase
H	H	L	Eighth Step	2W1-2 Phase
H	H	H	Sixteenth Step	4W1-2 Phase



Figura 44

Idealmente, se debería ajustar el voltaje de referencia de los drivers utilizando la fórmula proporcionada por el datasheet:

$$I_{TripMAX} = V_{REF}/(8 \times R_S)$$

Sin embargo, como no se disponía de datos precisos sobre la corriente máxima utilizada por los motores ($I_{TripMAX}$), se optó por un enfoque práctico que se detalla a continuación:

1. Se implementó un código simple en el microprocesador (Calibracion_PAP.c) para simular una señal pulsante utilizando HAL_Delay y se generó un 1 lógico en un pin del microcontrolador para indicar la dirección del motor (figura 45)

```
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET);
    if (contador == 0){
        for (int i = 0; i < 1100; i++){
            HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
            HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_9);
            HAL_Delay(1);
            HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
            HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_9);
            HAL_Delay(1);
        }
        contador = 1;
    }
    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

Figura 45

Como se puede apreciar, se generó un indicador visual prendiendo y apagando el pin 13 del microcontrolador, que corresponde al led integrado del mismo.

El número 1100 corresponde a la cantidad de micropasos que tiene que generar para que el desplazamiento traslacional sea de 10mm. Dicho valor ha sido regulado mediante prueba y error midiendo con un calibre pie de rey.

2. Una vez cargado el código, se procedió a hacer las conexiones correspondientes (figura 46).

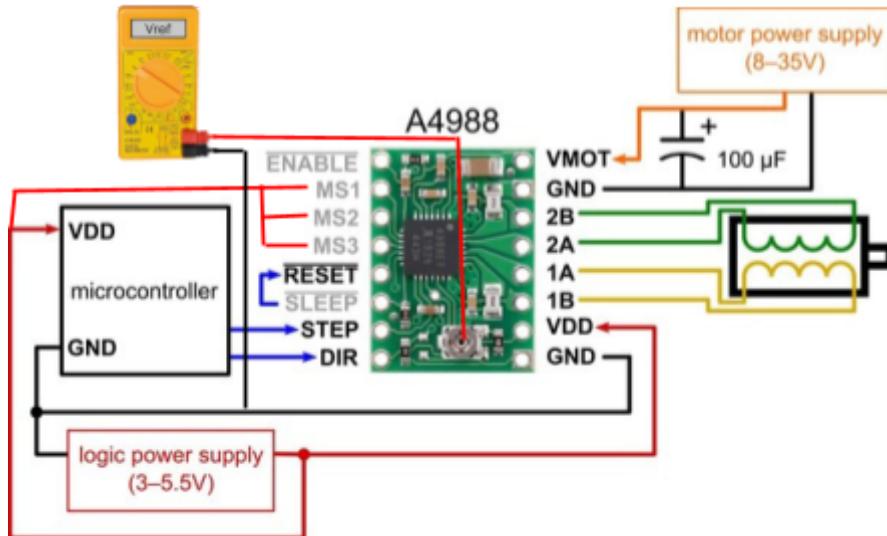


Figura 46

3. Por último, ya con el código en funcionamiento, se aumentó el voltaje de referencia (V_{ref}) gradualmente, partiendo de $V_{ref} = 0V$, hasta que el motor comenzó a moverse sin problemas.

El valor final de V_{ref} para cada motor fue de 0.15V. A partir de este valor, se estima que la corriente máxima consumida por los motores es de aproximadamente:

$$I_{max} = V_{ref}/(8 * R_s) = 0.15V/(8 * 0.1\Omega) = 0.1875 A$$

Servomotor

Antes de implementar el servo al proyecto, se realizó una prueba para variar dicho PWM y ver su funcionamiento. El procedimiento de la prueba realizada se describe a continuación:

1. En el IDE de stm32cube se configuró para que la frecuencia del oscilador interno sea de 72MHz

2. Se configuró para que se utilice el timer número 2 (TIM2) para generar una señal PWM en un pin (figura 47)

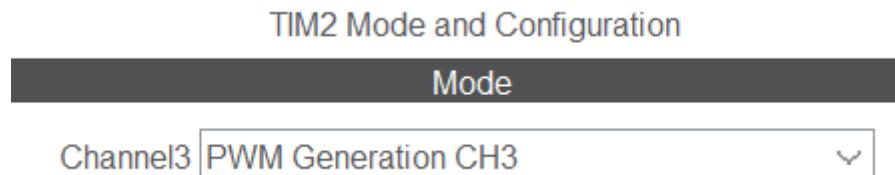


Figura 47

3. Se configuró un preescalador de 64 y un contador de periodo de 22500. Estos valores se eligieron en base a la siguiente ecuación:

$$f_{pwm} = \frac{f_{clk}}{(PSC-1)*ARR} = \frac{72MHz}{(65-1)*22500} = 50Hz$$

Siendo:

- f_{pwm} = Frecuencia de pwm
- f_{clk} = Frecuencia del reloj interno (clock)
- PSC = Pre-escalador (Prescale)
- ARR = Contador (Auto Reload Register)

4. Se generó el código que se muestra en la figura 48 (Prueba_Servo):

```

HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3); // Iniciar la generación de la señal PWM
TIM2->CCR3 = 500; // Establecer el valor de comparación

while (1)
{
    HAL_Delay(10);

    TIM2->CCR3 += 50; // Incrementar el valor de comparación
    if (TIM2->CCR3 > 2450)
    {
        TIM2->CCR3 = 550; // Restablecer el valor de comparación a 550
        HAL_Delay(500);
    }
}

```

Figura 48

El cual la explicación se puede ver como comentarios dentro del código.

5. Como se puede ver, el valor de comparación varía entre el 0.024% y el 0.11% del Duty Cycle (550 y 2450 respectivamente). Con estos valores se logró una variación de entre -90° y 90°.

Conexiones finales

Para lograr las conexiones, se reutilizaron cables los cuales han sido conseguidos en los cables de teléfono. A diferencia de los clásicos “Dupont” que se suelen usar para Arduino, estos son en su totalidad de aluminio, lo cual es una desventaja ya que tiene una mayor resistencia. Sin embargo, para la aplicación funcionan bien ya que se permite cortarlos a la longitud que uno desea y soldarlos de manera fácil. Ademas vienen de color rojo, azul y blanco, lo cual dichos colores se aprovecharon para diferenciar las conexiones al polo positivo de 5V o 3.3V (cable rojo), polo negativo GND (cable azul) y de señal (cable blanco).

Primero se cortaron todos los cables los cuales corresponden a las conexiones en la protoboard (como por ejemplo bluepill con drivers A4988) y se realizaron las conexiones con los drivers previamente calibrados (figura 49)

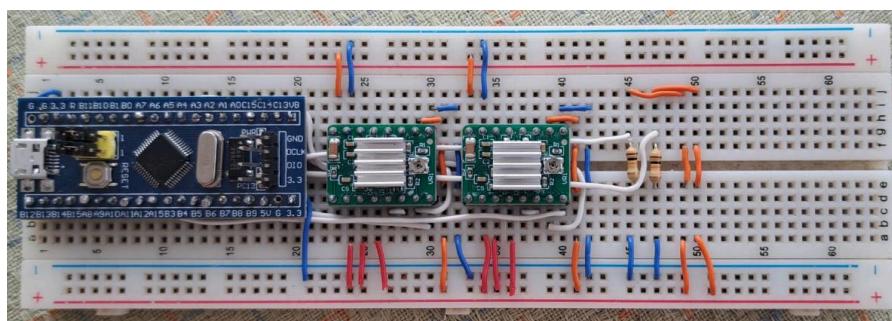


Figura 49

A los finales de carrera se les conectó una resistencia pull down de 10000 ohms (figura 50)

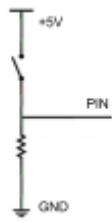


Figura 50

Luego se conectaron los motores, finales de carrera, teclado matricial y lcd (figura 51)



Figura 51

Alimentación

Se reutilizó una fuente de alimentación de una computadora en desuso. Para que sea fácil de utilizar, se diseñó en Solid Edge una caja y se hicieron las conexiones correspondientes para las diferentes salidas de voltaje (figura 52).



Figura 52

De esta fuente, se utilizó un puerto USB para alimentar el microcontrolador, la salida de 5V para alimentar el servomotor, los drivers A4988, la pantalla LCD y para obtener un 1 lógico en los finales de carrera y los pines ms1, ms2 y ms3 de los drivers y la otra salida regulada a 9V para los motores PAP..

Resultados, especificaciones finales

Metas y objetivos

Para el proyecto se han logrado muchos objetivos planteados desde el principio. Sin embargo el diagrama de bloques se ha ido modificando continuamente debido a que se pensaban diferentes formas en el transcurso de la realización del proyecto para que el código sea prolíjo, entendible y con un funcionamiento más óptimo.

Con respecto a los componentes utilizados no se realizaron cambios. Desde un principio se pensó en utilizar dos motores PAP reutilizados, un servomotor, una bluepill, dos drivers A4988, una pantalla LCD de 16x2, un teclado matricial de 4x4 y dos finales de carrera.

Sin embargo, al comenzar el proyecto, se planeó agregar una opción la cual el usuario pueda cambiar algunas variables desde el teclado matricial, como por ejemplo la velocidad de los motores o la presión del punzado. Además también se

había planeado utilizar FreeRTOS en Homing.c. Estas dos ideas que se pensaron desde el principio no se lograron realizar debido a que se tuvo como prioridad el funcionamiento correcto de la impresora el cual demoró más de lo previsto.

También hubo unos cambios en el diseño de la estructura. Al principio se pensaba que era mejor que quedaran las partes por separado (la impresora, las conexiones en protoboard y el teclado con el LCD). Pero luego se pensó que era mejor que todo quede como una sola estructura, debido a que facilita el transporte de la impresora y evitará desconexiones, ya que las conexiones en la protoboard no son soldaduras y corren el riesgo de desconectarse.

Por último se hicieron modificaciones en las máquinas de estado, las cuales resultaron ser más útiles y mejoró el orden dentro del código.

Especificaciones técnicas

Precisión

Con respecto a la precisión del proyecto, se pueden hacer muchas mejoras. Por ejemplo las correderas que se deslizan por el eje de los motores PAP podrían ser de mejor calidad, ya que estos fueron impresos en 3D y se tuvieron que hacer algunas modificaciones para que se deslice de una manera fluida. Cuando los motores PAP tienen velocidades altas no son capaces de mover correctamente los ejes, por lo tanto puede generarse una pérdida de pasos en dichos motores.

Sin embargo, si la velocidad es controlada, la impresora imprime como corresponde ya que se calibraron los pasos por mm de cada motor para que se mueva la distancia necesaria según "[La ONCE](#)".

Velocidad

Como se dijo anteriormente, las velocidades están limitadas debido a que las correderas no son de buena calidad, Sin embargo estas se podrían mejorar de tal forma que la velocidad quede solamente limitada a la velocidad máxima de los motores PAP.

Además dichas velocidades y la velocidad del servomotor se podrían mejorar utilizando diferentes rampas de aceleración y luego interpolando dicha rampa regular la velocidad. Por ejemplo se podría usar una interpolación quíntuple (figura 53)

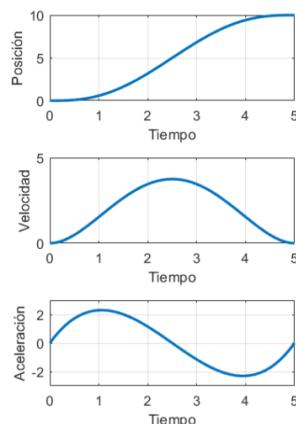


Figura 53

Por lo tanto los motores no se exigirán mucho al momento de moverse de un punto a otro debido a que las aceleraciones son más suaves.

Conclusiones. Ensayo de ingeniería de producto

El proyecto ha alcanzado resultados altamente satisfactorios, cumpliendo con los objetivos iniciales establecidos. A lo largo del proceso, se enfrentaron diversas dificultades, tales como problemas mecánicos en la impresora y desafíos en el proceso de desoldado de piezas para su reutilización. Sin embargo, estas dificultades sirvieron como oportunidades de aprendizaje significativas.

Desde el inicio, se enfocó en desarrollar una programación estructurada con el propósito de facilitar futuras mejoras y la replicabilidad del proyecto para otros usuarios. Sin embargo, lo más destacado de este proyecto radica en su potencial para beneficiar a comunidades que enfrentan limitaciones, brindándoles herramientas que pueden mejorar su calidad de vida. Aunque estos proyectos no

resuelven todos los problemas, sí representan una mejora significativa, y creo firmemente en la contribución individual al bienestar común.

Para convertir este proyecto en una solución industrial, se requieren varias mejoras que se planean implementar en un futuro próximo:

1. Mejora de Trayectoria:

Actualmente, la trayectoria no es óptima y podría mejorarse de muchas maneras distintas. Dos de ellas podrían ser:

- Al finalizar con el último punzado de la fila, el punzón podría avanzar a la siguiente. La figura 54 ilustra la comparación entre la trayectoria actual y la posible mejora.

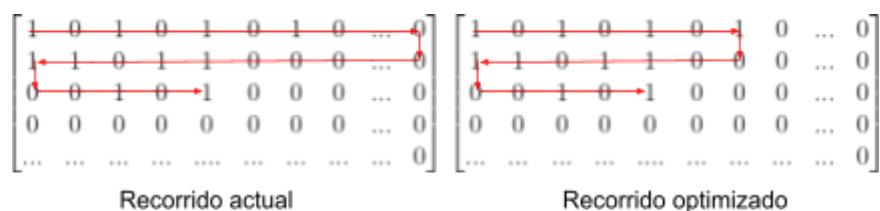


Figura 54

- Se podría implementar un algoritmo de búsqueda A* para calcular la ruta más óptima teniendo en cuenta la heurística y el costo del camino.

2. Actualización de Componentes:

- Integrar motores PAP Nema 17 permitiría alcanzar mayores velocidades y manejar una carga útil superior. Esta mejora posibilitaría redimensionar la impresora para trabajar con hojas de papel de mayor tamaño y, por ende, imprimir más caracteres.
- Como se dijo anteriormente, cambiar las piezas que se deslizan sobre las varillas por rodamientos lineales y carril que encasta en el rotor del motor PAP por alguna pieza de mejor calidad.
- Reemplazar el teclado matricial por uno de computadora aceleraría el proceso de escritura. Asimismo, añadir letras braille en las teclas y un parlante para proporcionar retroalimentación auditiva permitiría que personas no videntes utilicen la impresora.

3. Mejora de Velocidades:

Como se dijo anteriormente, implementar rampas de aceleración mejorarían el movimiento de los motores, evitando picos de aceleración y garantizando un desplazamiento más suave.

4. Control de Motores PAP con PWM:

La utilización de modulación por ancho de pulsos (PWM) para controlar los motores ofrecería un mayor grado de precisión en las velocidades, ya que actualmente se genera una señal pulsante cambiando el estado del pin y utilizando la función HAL_Delay, la cual puedo obtener una frecuencia máxima de pulsos de 1000Hz, ya que el valor mínimo que se podría colocar en la función es de un 1ms. Utilizando una señal PWM se podría ajustar dicha frecuencia y se agregaría un contador de pulsos para controlar la posición.

5. Agregado de contador de pulsos en motores PAP

Al agregar un contador de pulsos en los motores PAP, además de poder utilizar una señal PWM controlada, permitirá agregar una detección de errores. Por ejemplo si no toca un final de carrera antes de contar cierta cantidad de pulsos, se pausaría el movimiento de la impresora, ya que hubo una pérdida de pasos o el motor está detenido por algún objeto externo que impide su movimiento.

6. Pulsador de parada de emergencia

Mientras se imprime, debería estar habilitado algún pulsador el cual provoque una interrupción y detiene la impresión. Esto puede ser beneficioso debido a que si el usuario ve que durante la impresión hay algún problema con los motores o está saliendo mal la impresión por algún motivo, este tendrá la posibilidad de pararla de una manera sencilla y reiniciarla sin necesidad de desconectar la placa y escribir el texto nuevamente.

7. Cambio de drivers

Los drivers A4988 son uno de los que menos micropasos ofrece. Para mejorar esto se podrían cambiar por los DRV8825 que ofrece unos micropasos de hasta 1/64 o utilizar un driver de la serie TMC, los cuales ofrecen unos micropasos

de 1/128. A mayor micropasos, mayor precisión en los movimientos y más silenciosa será la impresora.

Referencias

*1) <https://www.once.es/>

*2)

<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiFr9ut55aEAxXmtJUCHQPADUcQFnoECBsQAQ&url=https%3A%2F%2Fwww.once.es%2Fservicios-sociales%2Fbraille%2Fdocumentos-tecnicos%2Fdocumentos-tecnicos-relacionados-con-el-braille%2Fdocumentos%2Fb1-parametros-dimensionales-del-braille-2%2F%40%40download%2Ffile%2FB1%2520Par%25C3%25A1metros%2520dimensionales%2520del%2520braille%2520OV1.pdf&usg=AOvVaw2AZlf6QskpMhvtuFMarTP7&opi=89978449>

*3)

<https://www.once.es/servicios-sociales/braille/comision-braille-espanola/documentos-tecnicos/documentos-tecnicos-relacionados-con-el-braille/documentos/b2-signografia-basica-lenguas-cooficiales>

*4)

<https://solectroshop.com/es/blog/como-funcionan-los-motores-pap-que-son-los-microsteps-n90>

*5)

https://www.kjell.com/globalassets/mediaassets/701916_87897_datasheet_en.pdf?ref=4287817A7A

*5)

https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf

*6)

http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf

*7)

<https://tienda.starware.com.ar/producto/teclado-membrana-matricial-generico-4x4-numerico-adhesivo/>

Anexos

*1) <https://www.thingiverse.com/thing:3521286>

*2) <https://www.thingiverse.com/thing:3170748>