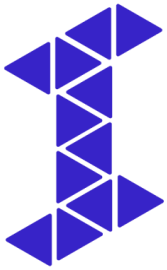


Facultad de
INGENIERÍA
Universidad Nacional de Lomas de Zamora

Informe de Práctica Profesional Supervisada

Alumno: Jonatan Bogado
DNI: 34255930

Empresa: A.D. Barbieri S.A.

Tutor Institucional: Nicolás Barcia
Posición: Gerente de Ingeniería

Tutor Académico: Ing. Cristian Lukaszewicz
Datos:

Firma tutor Institucional: _____

Índice

1. Reservado a la Facultad para evaluación

2. Lugar en donde he realizado la PPS

Realicé mi Práctica Profesional Supervisada en la empresa A.D. Barbieri S.A., ubicada en Burzaco, partido de Almirante Brown, Provincia de Buenos Aires, Argentina. Esta planta industrial forma parte del Parque Industrial de Burzaco, un polo productivo destacado en la región.

La empresa cuenta con una planta de gran dimensión, equipada con líneas de producción modernas orientadas a la fabricación de perfiles de acero galvanizado para sistemas constructivos en seco. Las instalaciones están organizadas de manera eficiente, permitiendo un flujo continuo de materiales desde el ingreso de la materia prima hasta el producto final.

Dentro de la planta se encuentran máquinas perfiladoras, máquinas de corte hidráulico, sistemas de transporte y manipulación de materiales, y diversos equipos automatizados. El área en la que realicé la práctica está dedicada a mantenimiento e innovación tecnológica, donde se trabajan proyectos de mejora sobre equipos existentes, como el que integré.

El personal de la empresa se organiza en turnos y mantiene una jornada laboral de lunes a viernes, de 8 a 17 horas. Durante mi estadía, pude interactuar con técnicos, ingenieros y operarios altamente capacitados, lo cual me permitió aprender tanto desde el punto de vista técnico como humano.

Me llamó especialmente la atención el compromiso de la empresa con la mejora continua y la búsqueda constante de innovación. Este ambiente propicio me permitió aplicar conocimientos adquiridos durante mi formación académica y a la vez adquirir nuevas competencias técnicas y profesionales.

3. Presentación

3.1. Introducción

Durante mi Práctica Profesional Supervisada en A.D. Barbieri S.A., participé en la actualización de una máquina de corte hidráulico para su uso en obras, desarrollada por la misma empresa. El proyecto consistió en incorporar mejoras tecnológicas que permitieran automatizar el proceso de alimentación y posicionamiento de perfiles metálicos, logrando así un corte más preciso, eficiente y seguro.

Mi tarea principal fue diseñar e implementar un sistema de control que utiliza motores paso a paso, sensores inductivos y una interfaz de usuario, permitiendo al operario configurar los parámetros del corte de forma sencilla. Este trabajo forma parte de un proceso de modernización de los equipos productivos de la planta, orientado a la automatización y digitalización industrial.

3.2. Objetivo

El objetivo principal del proyecto fue automatizar el proceso de ingreso, posicionamiento y corte de perfiles metálicos en la máquina, mediante el uso de tecnologías accesibles y eficientes. A través del desarrollo de un sistema de control programable, se buscó mejorar la precisión del corte, reducir errores humanos, facilitar la operación de la máquina e incrementar la seguridad del operario.

3.3. Alcance del proyecto

El proyecto abarcó el diseño, implementación y prueba de un sistema electrónico y de control que permite:

- Controlar tres motores paso a paso para la tracción del perfil.
- Detectar la posición del perfil mediante dos sensores inductivos.
- Configurar parámetros de corte (tipo de perfil, longitud de entrada y longitud de corte) mediante una interfaz con display y encoder rotativo.
- Iniciar el proceso de corte mediante un pulsador verde tipo hongo, y detenerlo de forma segura con un pulsador de emergencia.

No formó parte del alcance la modificación del sistema hidráulico de corte ni la implementación de conectividad remota, aunque el sistema quedó preparado para futuras expansiones.

3.4. Resumen

El proyecto consistió en modernizar la máquina de corte hidráulico mediante la integración de un sistema electrónico de control que automatiza el proceso de alimentación y posicionamiento del perfil metálico. Se incorporaron sensores inductivos, motores paso a paso, un encoder rotatorio y una interfaz de usuario con pantalla LCD, permitiendo que el operario configure los cortes de forma precisa y segura. También se añadieron dispositivos de control externo para iniciar y detener el proceso, mejorando así la ergonomía y la seguridad de la operación. Esta solución representa un paso importante hacia la transformación digital de los equipos de planta, alineándose con los principios de la Industria 4.0.

4. Resolución del Proyecto

Durante el desarrollo de mi práctica profesional en la empresa A.D. Barbieri S.A., me asignaron la tarea de actualizar e incorporar funcionalidades a una máquina de corte hidráulico, con el objetivo de convertirla en un sistema más automatizado, preciso y eficiente.

4.1. Relevamiento y análisis inicial

El primer paso fue analizar el funcionamiento de la máquina en su estado original. La máquina realizaba cortes de perfiles metálicos mediante una matriz accionada por una unidad hidráulica, pero no contaba con un sistema automático de posicionamiento ni tracción del perfil, ni con una interfaz de usuario moderna.

Me reuní con el equipo técnico de la empresa para comprender los requerimientos del sistema y definir qué tipo de mejoras eran necesarias para cumplir con los objetivos de precisión, seguridad y facilidad de uso. A partir de esto, elaboré una propuesta de automatización basada en el uso de microcontroladores, motores paso a paso, sensores y una interfaz gráfica de usuario.

4.2. Diseño e integración del sistema mecánico y electrónico

Diseñe un sistema de ingreso automático del perfil metálico utilizando dos motores paso a paso NEMA 17 que traccionan el material desde el frente de la máquina, empujándolo hacia la zona de corte. Para verificar que el perfil llegue correctamente a la zona de trabajo, integré un sensor inductivo antes de la matriz de corte.

Una vez que el perfil atraviesa la matriz, otro motor paso a paso, también NEMA 17, asistido por un encoder rotativo, continúa traccionando el perfil para asegurar un avance estable y constante. También coloqué un segundo sensor inductivo en la salida para confirmar el paso completo del perfil.

El control de estos motores y sensores se realiza mediante una placa ESP32-WROOM-32, la cual programé para recibir los parámetros definidos por el usuario, mover los motores en función de la longitud deseada y gestionar los sensores para asegurar condiciones seguras antes del corte.

4.2.1. Especificaciones Técnicas

Para llevar adelante el proyecto, trabajé sobre la base mecánica de la máquina, que ya contaba con un sistema hidráulico de corte, pero sin automatización en el ingreso ni en el posicionamiento del perfil. El sistema que diseñe incluye los siguientes componentes principales:

- **3 motores paso a paso NEMA 17:**
  - Dos ubicados en la entrada del perfil para generar tracción y alineación.
  - Uno ubicado después del área de corte, equipado con un encoder rotativo para retroalimentación de movimiento y asistencia en la salida del perfil.
- **2 sensores inductivos tipo NPN normalmente abiertos:**
  - Uno antes de la matriz de corte, para detectar la llegada del perfil.
  - Otro después del área de corte, para verificar el paso completo del mismo.
- **Encoder rotatorio incremental:** Utilizado como elemento de medición del perfil.

4.2.2. Características del perfil a traccionar

- Tipo de material: Acero galvanizado (densidad aproximada: 7.85 g/cm³)
- Longitud estándar: 6 metros (aunque la máquina puede trabajar hasta 12 m)
- Dimensiones de perfiles a mover: desde PGU 60 hasta PGC 250
- Peso aproximado:
    - Un perfil PGU 60 de 6 metros puede pesar alrededor de 5 kg
    - Un perfil PGC 250 de 6 metros puede pesar hasta 20 kg

Para este proyecto vamos a considerar el caso más exigente:

- Perfil de 6 metros de largo y 20 kg de peso

4.2.3. Cálculo de la fuerza de tracción necesaria

Lo importante no es tanto el peso total sino la fuerza de fricción entre el perfil y la superficie, ya que el movimiento es horizontal y no se requiere levantar el perfil, sino arrastrarlo.

Fuerza de tracción = fricción estática

$$F_{tracción} = \mu * N$$

Donde:

- μ: coeficiente de fricción entre acero y acero en seco ≈ 0.5. (Si hay rodillos, puede bajar hasta 0.05 – 0.1)
- N: fuerza normal = peso = masa · gravedad
- $m = 20\,kg$
- $g = 9.81\,m/s^2$

Entonces:

$$F_{tracción} = 0.5 * (20 * 9.81) = 0.5 * 196.2 = 98.1\ N$$

Necesitamos al menos 100 N de fuerza de tracción para arrastrar el perfil más grande sin rodillos. Si usamos rodillos o guías, el coeficiente de fricción baja a 0.1 y la fuerza sería de solo 20 N.

4.2.4. Conversión de fuerza a torque

Suponiendo que el motor mueve un rodillo de tracción con radio (r = 2 cm = 0.02 m):

$$T = F * r$$

$$T = 100\ N * 0.02\ m = 2\ Nm$$

El sistema requiere al menos 2 Newton·metro (Nm) de torque por motor en condiciones adversas (sin rodillos).

4.2.5. Elección del motor

Un motor NEMA 17 estándar entrega entre 0.4 y 0.7 Nm directamente.

Para alcanzar los 2 Nm requeridos, se usan:

- Reducción mecánica (gearbox): Por ejemplo, una reducción 5:1 aumenta el torque 5 veces.
- Uso de 2 motores en paralelo para compartir carga

Cálculo final:

Con un motor de 0.5 Nm, y una relación de reducción 5:1, se obtiene:

$$T_{salida} = 0.5\ Nm * 5 = 2.5\ Nm$$

Cumple perfectamente con el requerimiento de tracción máxima.

4.2.6. Conclusión técnica

La elección de motores NEMA 17 con acople a sistemas de reducción mecánica (5:1) permite alcanzar un torque de tracción superior a 2 Nm, lo cual garantiza la capacidad de arrastrar perfiles de hasta 250 mm de ancho y 20 kg de peso sobre superficie metálica, incluso sin asistencia de rodillos. Si se incorpora un sistema de rodillos, la carga efectiva baja considerablemente, permitiendo un margen de seguridad mayor y mejor eficiencia energética.

4.2.6.1. Integración mecánica

El sistema de tracción que diseñe está compuesto por tres motores paso a paso modelo NEMA 17 – 17HS4401. Dos de ellos se ubican en la entrada del perfil, y son responsables de hacer avanzar el mismo hacia el área de corte. El tercero se posiciona a la salida del área de corte y actúa como soporte de tracción adicional, incluyendo un encoder rotativo acoplado en su parte superior para medir el avance real del perfil y corregir posibles desviaciones.

La estructura se diseñó teniendo en cuenta el peso y las dimensiones de los perfiles. Para el perfil de mayor peso (PGC 250, 6 m de largo y aproximadamente 20 kg), calculé la fuerza de tracción necesaria considerando el rozamiento acero-acero. Estimé una fuerza máxima de

100 N, lo cual representa un torque requerido de aproximadamente 2 Nm en el rodillo de tracción. Con el uso de motores NEMA 17 (0.147 Nm) y una reducción mecánica de 5:1, obtengo un torque de salida de aproximadamente 0.735 Nm por motor, lo cual es suficiente para el trabajo en conjunto o con asistencia de rodillos.

4.2.6.2.Integración electrónica

Para el control de los motores utilicé drivers DRV8825, los cuales permiten corriente de hasta 2.2 A por fase y ofrecen microstepping hasta 1/32, lo que mejora notablemente la precisión del movimiento. Los motores y drivers son alimentados por una fuente switching de 24 V y 5 A, lo que garantiza un rendimiento óptimo y buena respuesta en carga.

El sistema cuenta con una pantalla TFT ILI9341 de 2.4 pulgadas donde visualizo y configuro los parámetros de corte. Esta interfaz es manipulada por medio de un encoder rotativo KY-040, que permite seleccionar la longitud deseada del perfil a cortar, el tipo de perfil y la cantidad de cortes. Para iniciar el proceso, instalé un pulsador tipo hongo verde, que activa el programa de corte, siempre que haya una configuración cargada. También incluí un pulsador de emergencia, que detiene todo el sistema ante cualquier falla o situación de riesgo.

El control de todo el sistema lo realiza un microcontrolador ESP32-WROOM-32, que ejecuta el programa que desarrollé, y que en el futuro permitirá además enviar métricas de uso a través de WiFi a una app o dashboard.

4.2.6.3. Sensado y control

Para mejorar el control de posición del perfil, instalé dos sensores inductivos: uno antes del área de corte y otro después. Estos sensores permiten verificar la presencia del perfil y confirman su paso correcto por la matriz antes de activar el corte hidráulico. El encoder rotativo adicional también contribuye a medir el avance real, validando que el perfil haya sido posicionado con precisión antes de cada ciclo de corte.

4.3. Desarrollo de la interfaz de usuario

Para facilitar la interacción con la máquina, implementé una interfaz gráfica con una pantalla TFT ILI9341 de 2.4". Esta pantalla permite al operario:

- Navegar entre diferentes pantallas usando un encoder rotativo KY-040,
- Seleccionar el tipo de perfil a cortar,
- Ingresar la longitud deseada de corte,
- Ingresar la longitud del perfil original,
- Confirmar y lanzar el proceso de corte.

También diseñé la interfaz para mostrar alertas, mensajes de estado y validaciones en tiempo real. Para iniciar el proceso de corte, el operario debe presionar un pulsador tipo hongo color verde, pero solo si el sistema detecta que se ha cargado un programa de corte válido y los sensores confirman que el perfil está correctamente posicionado. Incorporé además un pulsador de emergencia que detiene inmediatamente el sistema ante cualquier evento anormal.

4.3.1. Pines utilizados

- Pines para el display ILI9341

TFT_DC 2 - display ILI9341
TFT_RST 4 - display ILI9341
TFT_CS 15 - display ILI9341
SPI_SCK 18 - display ILI9341
SPI_MISO 19 - display ILI9341
SPI_MOSI 23 - display ILI9341

- Pines para los motores paso a paso (NEMA)
STEP_MOTOR_3 5 - NEMA
ENABLE_MOTORS_PIN 17 - NEMA
STEP_MOTOR_1 0 - NEMA
DIR_MOTOR_2 21 - NEMA
STEP_MOTOR_2 22 - NEMA
DIR_MOTOR_1 26 - NEMA
DIR_MOTOR_3 32 - NEMA

- Pines para el encoder del perfil
PROFILE_ENCODER_B 16 - encoder
PROFILE_ENCODER_A 25 - encoder

- Pines para los sensores inductivos
SENSOR_IND_1 36 - sensores inductivos
SENSOR_IND_2 39 - sensores inductivos

- Pines de Pulsadores cabeza de hongo
BTN_EMERGENCY 13 - Botones
BTN_START 27 - Botones

- Encoder del menú de usuario. Pines para el encoder rotatorio del menú (KY-040)
ENCODER_SW_PIN 33 - Encoder display
ENCODER_CLK_PIN 34 - Encoder display
ENCODER_DT_PIN 35 - Encoder display

- Boton exclusivo menú
  BTN_MENU 1 - Botones

- Pin para la cizalla hidráulica
  CIZALLA_PIN 12 - Cizalla

- Pin para el sensor de fin de carrera del pistón hidráulico
  CIZALLA_RETRAIDA_SENSOR_PIN 30 - Final Carrera

- Pin para el chip select de la tarjeta SD
  SD_CS 14 - SC

4.4. Programación y control

4.4.1. Software utilizado

Todo el sistema fue programado en el entorno Arduino IDE, utilizando múltiples bibliotecas como Adafruit_ILI9341 para manejar la pantalla, AccelStepper para controlar los motores paso a paso y ESP32Encoder para leer el encoder rotativo.

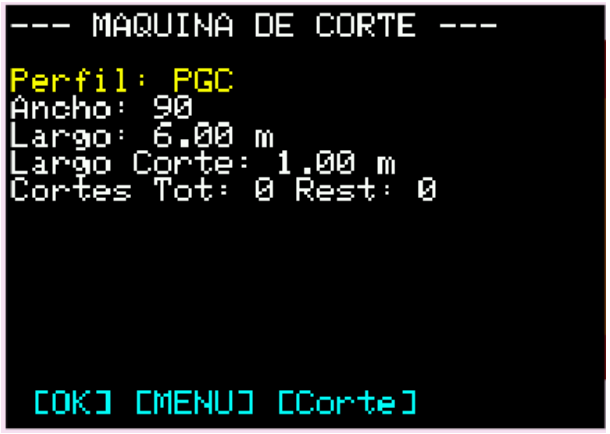
El código fue estructurado en estados que representan cada pantalla o situación del sistema. Esto incluye:

- Estado de configuración,
- Estado de espera,
- Estado de ejecución del corte,
- Estado de error o emergencia.

Me aseguré de implementar lógica de seguridad para que el sistema solo pueda realizar movimientos si las condiciones de sensores y parámetros son las correctas.

4.4.2. Diagramas de pantallas

Pantalla 1: Estado del corte



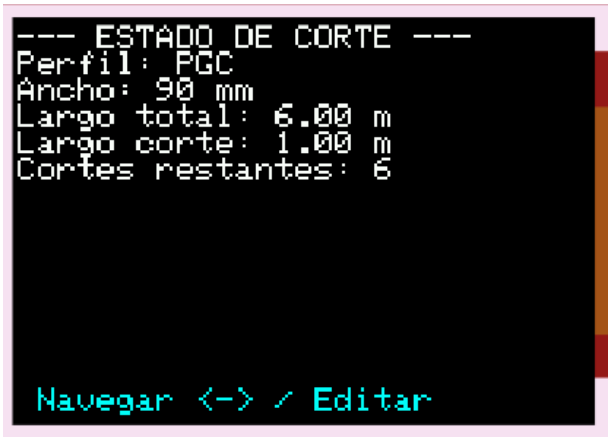
Esta es la pantalla principal del sistema. En ella se visualiza en tiempo real:

- El tipo de perfil seleccionado (por ejemplo: PGU 60, PGC 70, etc.).
- La longitud de corte programada.
- La longitud total del perfil ingresado (largo comprado).
- La cantidad restante de cortes pendientes, la cual se va decrementando automáticamente conforme avanza el proceso de corte.

Además, se muestra un estado dinámico del sistema que puede alternar entre:

- "Corte en proceso" cuando la máquina está trabajando.
- "Retirar perfil sobrante" cuando finaliza un corte.
- "Ingresar nuevo perfil" cuando se completan los cortes programados.
- En estado inactivo, no se muestra ninguna leyenda de estado.

Pantalla 2: Programación del corte

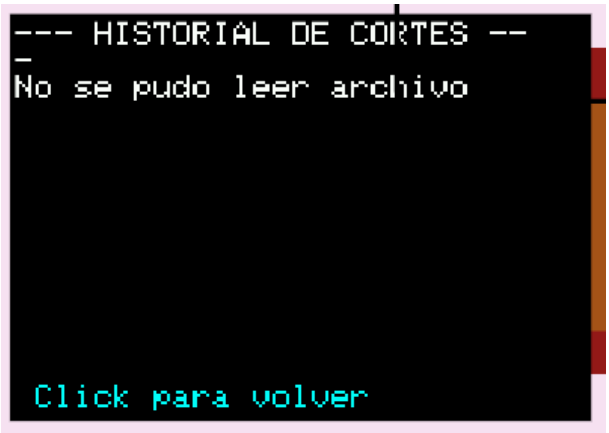


Desde esta pantalla el usuario puede configurar los parámetros del corte. Al presionar el switch del encoder ingreso al modo de navegación, donde puedo moverme entre los siguientes ítems:

- Tipo de perfil
- Longitud del perfil ingresado
- Longitud de corte
- Cantidad de cortes
- Regresar

Al pulsar nuevamente el switch sobre cualquier ítem, ingreso al modo de edición. Allí puedo girar el encoder para modificar el valor del campo seleccionado (por ejemplo, cambiar el tipo de perfil entre PGU 60 y PGC 250, o modificar la longitud del corte). Una vez elegido el valor deseado, presiono nuevamente el switch para guardar el cambio. Si regreso al primer ítem y selecciono "Regresar", el sistema vuelve a la pantalla principal.

Pantalla 3: Histórico de cortes

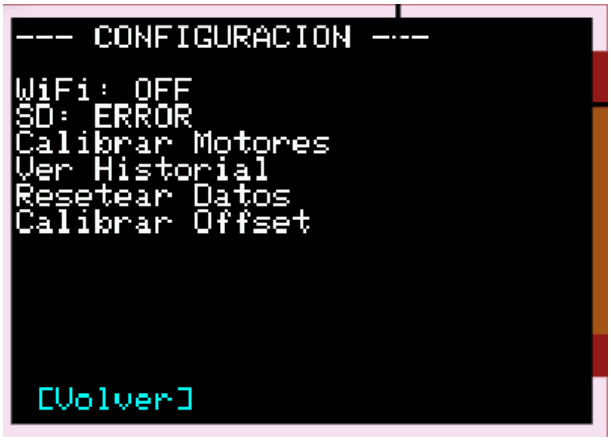


En esta pantalla se muestra un listado ordenado cronológicamente en forma descendente con los registros de los cortes realizados. Para cada corte almacenado se detalla:

- Fecha y hora del inicio de la operación.
- Tipo de perfil cortado.
- Longitud del corte realizada.
- Cantidad de cortes ejecutados con esa configuración.

Estos datos se almacenan automáticamente en una tarjeta microSD conectada al sistema, lo que permite su recuperación o análisis posterior en una computadora. Esto resulta útil para llevar un control de producción y trazabilidad.

Pantalla 4: Configuraciones



Desde esta pantalla es posible visualizar y configurar opciones del sistema:

- Estado de la conexión WiFi.
- Dirección IP asignada al ESP32.

- Estado de la conexión Bluetooth (pensada para uso futuro).
- Estado del almacenamiento en la tarjeta SD.

Estas opciones me permiten verificar que el sistema esté correctamente conectado a la red local y que pueda ser ampliado en futuras versiones para comunicación remota o integración con una app o servidor.

4.5. Pruebas y validación

Realicé múltiples pruebas en taller para verificar el funcionamiento de cada parte del sistema por separado:

- Movimiento de motores paso a paso y precisión de posicionamiento,
- Lectura confiable de los sensores inductivos,
- Detección precisa mediante el encoder rotativo,
- Fluidez en la navegación por menús y validación de entradas.

Una vez superadas las pruebas individuales, realicé pruebas integradas en conjunto con el equipo técnico. Verificamos que el sistema realizará los cortes de forma autónoma y segura, respetando las longitudes ingresadas por el operario.

5. Conclusiones

La realización de este proyecto de actualización y automatización de una máquina de corte hidráulico en la empresa A.D. Barbieri S.A. me permitió integrar de forma práctica los conocimientos adquiridos a lo largo de la carrera de Ingeniería Mecatrónica, enfrentándome a un desafío real de diseño, implementación y validación de un sistema complejo que combina mecánica, electrónica, programación y control.

Durante el desarrollo del sistema, apliqué conceptos fundamentales de distintas asignaturas de la carrera. Desde el análisis de fuerza y torque necesario para la selección adecuada de los motores paso a paso, hasta la elección de drivers, sensores y la estructura mecánica necesaria para garantizar la correcta tracción del perfil. También implementé una interfaz gráfica en una pantalla TFT ILI9341 controlada desde un ESP32, permitiendo al operario configurar parámetros como tipo de perfil, longitud de corte y cantidad de cortes a realizar, utilizando un sistema de menús interactivos controlado mediante un encoder rotativo.

En cuanto al software, diseñé un programa modular y escalable, empleando una estructura basada en máquina de estados, y desarrollé funcionalidades como el almacenamiento del historial de cortes, el control de seguridad mediante sensores inductivos y pulsadores de emergencia, y la capacidad de conectividad inalámbrica, todo dentro de las limitaciones de un sistema embebido. Este proyecto también me permitió comprender con mayor profundidad la importancia de optimizar el uso de memoria y recursos en plataformas de bajo consumo.

Las competencias que desarrollé incluyen:

- Integración multidisciplinaria de sistemas mecánicos, electrónicos y de control.
- Capacidad de análisis técnico para la selección de componentes.

- Programación de sistemas embebidos en lenguaje C/C++.
- Lectura e interpretación de documentación técnica industrial.
- Comunicación efectiva de ideas técnicas a través de interfaces visuales y diagramas.
- Planificación y gestión de un proyecto desde su concepto hasta su implementación.

En definitiva, esta experiencia fortaleció mi perfil como ingeniero mecatrónico, acercándome a las necesidades reales de la industria, mejorando mi autonomía, mi capacidad de resolución de problemas, y mi visión integral de los sistemas automatizados modernos. Este proyecto representa no solo una aplicación concreta de mis conocimientos, sino también una valiosa oportunidad de crecimiento profesional.

6. Anexos

6.1. Código del programa

csf_mk2.ino

```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ILI9341.h>
#include <ESP32Encoder.h>
#include <SD.h>
#include <AccelStepper.h>
#include <Wire.h>
#include <Bounce2.h>

// Pines para el display ILI9341
#define TFT_CS 15
#define TFT_DC 2
#define TFT_RST 4
#define SPI_SCK 18
#define SPI_MOSI 23
#define SPI_MISO 19
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_RST);

// Encoder del menú de usuario. Pines para el encoder rotatorio del menú (KY-040)
#define ENCODER_CLK_PIN 34
#define ENCODER_DT_PIN  35
#define ENCODER_SW_PIN  33
ESP32Encoder menuEncoder;
Bounce menuButtonDebouncer = Bounce();

#define BTN_MENU 0
Bounce menuScreenButtonDebouncer = Bounce();
```

```
// Pines para los motores paso a paso (NEMA)
#define DIR_MOTOR_1 26 // Motor de Tracción 1
#define STEP_MOTOR_1 1
#define DIR_MOTOR_2 21 // Motor de Tracción 2
#define STEP_MOTOR_2 22
#define DIR_MOTOR_3 32 // Motor de Tracción 3 (salida)
#define STEP_MOTOR_3 5
#define ENABLE_MOTORS_PIN 17

// Crear objetos AccelStepper para los motores
AccelStepper motor1(AccelStepper::DRIVER, STEP_MOTOR_1, DIR_MOTOR_1);
AccelStepper motor2(AccelStepper::DRIVER, STEP_MOTOR_2,
DIR_MOTOR_2);
AccelStepper motor3(AccelStepper::DRIVER, STEP_MOTOR_3,
DIR_MOTOR_3);

// Pines para el encoder del perfil
#define PROFILE_ENCODER_A 25
#define PROFILE_ENCODER_B 16
ESP32Encoder profileEncoder;
float stepsPerMeter = 6366.0;

// Pines para los sensores inductivos
#define SENSOR_IND_1 36
#define SENSOR_IND_2 39

// Pines de botones
#define BTN_START 27
#define BTN_EMERGENCY 13
Bounce startButtonDebouncer = Bounce();
Bounce emergencyButtonDebouncer = Bounce();
```

```
// Pin para la cizalla hidráulica
#define CIZALLA_PIN 12

// Pin para el sensor de fin de carrera del pistón hidráulico
#define CIZALLA_RETRAIDA_SENSOR_PIN 30

// Variables necesarias para el corte
const unsigned long CUT_ACTIVATION_TIME_MS = 1000; // Tiempo de
bajada de cizalla en ms
const unsigned long PISTON_RETURN_TIMEOUT_MS = 5000; // Tiempo
máximo para retorno del pistón

// Pin para el chip select de la tarjeta SD
#define SD_CS 14

// Definiciones de estado de la máquina
enum MachineState {
  IDLE,
  WAITING_FOR_PROFILE,
  MOVING_TO_OFFSET,
  READY_TO_CUT,
  PERFORMING_CUT_CYCLE,
  PAUSED,
  EMERGENCY_STOP,
  ERROR
};
MachineState currentState = IDLE;

//int currentScreen = 0;
const int NUM_SCREENS = 4;
bool editing = false;
int editOption = 0;
bool wifiEnabled = false;
```

```
bool sdInitialized = false;

// Estructuras de datos para los perfiles
enum ProfileType { PGC, PGU };
String profileTypeStrings[] = {"PGC", "PGU"};
int profileWidths[] = {60, 70, 90, 100, 140, 150, 200, 250}; // Anchos
disponibles

struct ProfileSettings {
  ProfileType type;
  int widthIndex;
};

// Variables para la configuración actual
ProfileSettings currentProfileSettings = {PGC, 2}; // Perfil inicial: PGC 90
(índice 2)
float currentLength = 6.00;     // Largo total del perfil en metros
float currentCutLength = 1.00;  // Largo de corte en metros
int totalCuts = 0;              // Cantidad total de cortes a realizar para el perfil
actual
int cutsRemaining = 0;          // Cortes restantes para el perfil actual

// Variables para el menú y display
int currentMenuOption = 0;
const int NUM_MAIN_OPTIONS = 5; // Perfil, Ancho, Largo, Largo Corte,
Cantidad Cortes
int currentScreen = 0; // 0: Principal, 1: Datos Wifi, 2: Config
bool editingValue = false; // Indica si estamos editando un valor con el
encoder

// Variables de offset para la cizalla
long matrixOffsetSteps = 0; // Offset en pasos del encoder desde la
posición de los dos sensores hasta la cizalla
```

```
/*void drawHomeScreen() {
  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(0, 0);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println("--- MAQUINA DE CORTE ---");
  tft.println("");

  // Perfil
  if (currentMenuOption == 0 && currentScreen == 0 && !editingValue)
tft.setTextColor(ILI9341_YELLOW); else tft.setTextColor(ILI9341_WHITE);
  tft.print("Perfil: ");
  tft.println(profileTypeStrings[currentProfileSettings.type]);

  // Ancho
  if (currentMenuOption == 1 && currentScreen == 0 && !editingValue)
tft.setTextColor(ILI9341_YELLOW); else tft.setTextColor(ILI9341_WHITE);
  tft.print("Ancho: ");
  tft.println(profileWidths[currentProfileSettings.widthIndex]);

  // Largo
  if (currentMenuOption == 2 && currentScreen == 0 && !editingValue)
tft.setTextColor(ILI9341_YELLOW); else tft.setTextColor(ILI9341_WHITE);
  tft.print("Largo: ");
  tft.print(currentLength, 2); tft.println(" m");

  // Largo Corte
  if (currentMenuOption == 3 && currentScreen == 0 && !editingValue)
tft.setTextColor(ILI9341_YELLOW); else tft.setTextColor(ILI9341_WHITE);
  tft.print("Largo Corte: ");
  tft.print(currentCutLength, 2); tft.println(" m");
```

```
  // Cantidad de Cortes
  if (currentMenuOption == 4 && currentScreen == 0 && !editingValue)
tft.setTextColor(ILI9341_YELLOW); else tft.setTextColor(ILI9341_WHITE);
  tft.print("Cortes Tot: "); tft.print(totalCuts);
  tft.print(" Rest: "); tft.println(cutsRemaining);

  tft.setCursor(0, tft.height() - 20);
  tft.setTextColor(ILI9341_CYAN);
  tft.print(" [OK] [MENU] [Corte]"); // Placeholder para botones físicos o
táctiles
}*/

void drawHomeScreen() {
  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(0, 0);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println("--- ESTADO DE CORTE ---");
  tft.println("Perfil: PGC");
  tft.println("Ancho: 90 mm");
  tft.println("Largo total: 6.00 m");
  tft.println("Largo corte: 1.00 m");
  tft.println("Cortes restantes: 6");
  tft.setCursor(0, tft.height() - 20);
  tft.setTextColor(ILI9341_CYAN);
  tft.print(" Navegar <-> / Editar");
}

void drawEditScreen() {
  const char* labels[] = {"Perfil: PGC", "Ancho: 90 mm", "Largo total: 6.00 m",
"Corte: 1.00 m", "Cantidad: 6"};
  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(0, 0);
```

```
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println("--- CONFIGURAR CORTE ---");
  for (int i = 0; i < 5; i++) {
    if (editOption == i && editing) {
      tft.setTextColor(ILI9341_YELLOW);
    } else if (editOption == i) {
      tft.setTextColor(ILI9341_CYAN);
    } else {
      tft.setTextColor(ILI9341_WHITE);
    }
    tft.println(labels[i]);
  }
  tft.setCursor(0, tft.height() - 20);
  tft.setTextColor(ILI9341_CYAN);
  tft.print(" Click para seleccionar/modificar");
}

void drawHistoryScreen() {
  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(0, 0);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println("--- HISTORIAL DE CORTES ---");
  File file = SD.open("/cuts_history.txt");
  if (file) {
    int line = 0;
    while (file.available() && line < 5) {
      String entry = file.readStringUntil('\n');
      tft.setCursor(0, 40 + line * 20);
      tft.setTextSize(1);
      tft.println(entry);
      line++;
```

```
    }
    file.close();
  } else {
    tft.println("No se pudo leer archivo");
  }
  tft.setCursor(0, tft.height() - 20);
  tft.setTextColor(ILI9341_CYAN);
  tft.print(" Click para volver");
}

void drawWifiScreen() {
  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(0, 0);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println("--- DATOS WIFI ---");
  tft.println("");
  tft.println("SSID: TuRedWifi");
  tft.println("IP: 192.168.1.100");
  tft.println("");
  //tft.println("Conectando..."); // O estado actual
  tft.setCursor(0, tft.height() - 20);
  tft.setTextColor(ILI9341_CYAN);
  tft.print(" [Volver]");
}

void drawConfigScreen() {
  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(0, 0);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println("--- CONFIGURACION ---");
  tft.println("");
```

```
  tft.print("WiFi: "); tft.println(wifiEnabled ? "ON" : "OFF");
  tft.print("SD: "); tft.println(sdInitialized ? "OK" : "ERROR");
  tft.println("Calibrar Motores");
  tft.println("Ver Historial");
  tft.println("Resetear Datos");
  tft.println("Calibrar Offset"); // Opción para calibrar el offset de la matriz
  tft.println("");
  tft.setCursor(0, tft.height() - 20);
  tft.setTextColor(ILI9341_CYAN);
  tft.print(" [Volver]");
}

void updateDisplay() {
  switch (currentScreen) {
    case 0: drawHomeScreen(); break;
    case 1: drawHistoryScreen(); break;
    case 2: drawWifiScreen(); break;
    case 3: drawConfigScreen(); break;
    // Agrega más pantallas según sea necesario
  }
}

/*void handleMenuEncoder() {
  static long oldEncoderPos = 0;
  long newEncoderPos = menuEncoder.getCount();

  if (newEncoderPos != oldEncoderPos) {
    if (editingValue) {
      // Estamos editando un valor (Largo o Largo Corte)
      float step = 0.01; // Paso de 0.01 metros
      if (newEncoderPos > oldEncoderPos) {
        // Aumentar valor
        if (currentMenuOption == 2) currentLength += step;
```

```
      else if (currentMenuOption == 3) currentCutLength += step;
    } else {
      // Disminuir valor
      if (currentMenuOption == 2) currentLength -= step;
      else if (currentMenuOption == 3) currentCutLength -= step;
    }
    // Asegurarse de que los valores no sean negativos
    if (currentLength < 0) currentLength = 0;
    if (currentCutLength < 0) currentCutLength = 0;
    // Recalcular totalCuts cuando se cambian Largo o Largo Corte
    if (currentCutLength > 0) {
      totalCuts = floor(currentLength / currentCutLength);
      cutsRemaining = totalCuts;
    } else {
      totalCuts = 0;
      cutsRemaining = 0;
    }

  } else {
    // Estamos navegando en el menú
    if (newEncoderPos > oldEncoderPos) {
      currentMenuOption++;
    } else {
      currentMenuOption--;
    }
    // Limitar opciones del menú según la pantalla actual
    if (currentScreen == 0) { // Pantalla principal
      if (currentMenuOption >= NUM_MAIN_OPTIONS) currentMenuOption =
0;
      if (currentMenuOption < 0) currentMenuOption = NUM_MAIN_OPTIONS
- 1;
    } else if (currentScreen == 1) { // Pantalla Wifi - no navegable
      currentMenuOption = 0;
```

```
    } else if (currentScreen == 2) { // Pantalla Config (ejemplo para 4 opciones)
      currentMenuOption %= 4;
      if (currentMenuOption < 0) currentMenuOption += 4;
    }
   }
   oldEncoderPos = newEncoderPos;
   updateDisplay();
  }
}

// Función para manejar el botón del encoder (seleccionar/entrar/salir edición)
void handleEncoderButton() {
  // Cuando se presiona el botón del encoder
  if (currentScreen == 0) { // Pantalla principal
   if (currentMenuOption == 0) { // Perfil
    if (!editingValue) {
     currentProfileSettings.type = (currentProfileSettings.type == PGC) ? PGU : PGC;
    }
   } else if (currentMenuOption == 1) { // Ancho
    if (!editingValue) {
     currentProfileSettings.widthIndex = (currentProfileSettings.widthIndex + 1) % (sizeof(profileWidths) / sizeof(profileWidths[0]));
    }
   } else if (currentMenuOption == 2 || currentMenuOption == 3) { // Largo o Largo Corte
    editingValue = !editingValue; // Alternar entre modo edición y navegación
    if (editingValue) {
     // Resetear el contador del encoder para empezar a editar desde 0
     menuEncoder.clearCount();
```

```
    }
  } else if (currentMenuOption == 4) { // Cantidad de Cortes (no editable directamente)
    // Podrías implementar una lógica para reiniciar el contador si es necesario.
  }
 } else if (currentScreen == 1 || currentScreen == 2) { // Pantallas Wifi o Configuración
  // Aquí puedes implementar la lógica de "Volver" o seleccionar opciones de configuración
  // Por simplicidad, un toque podría volver a la pantalla principal
  currentScreen = 0;
  currentMenuOption = 0; // Resetear opción al volver
 }
 updateDisplay();
}*/

void handleMenuEncoder() {
 static long oldEncoderPos = 0;
 long newEncoderPos = menuEncoder.getCount();
 static unsigned long lastChangeTime = 0;
 unsigned long now = millis();

 if (abs(newEncoderPos - oldEncoderPos) >= 4 && (now - lastChangeTime) > 150) {
  lastChangeTime = now;
  if (currentScreen == 1 && editing) {
   if (newEncoderPos > oldEncoderPos) editOption++;
   else editOption--;
   if (editOption < 0) editOption = 4;
   if (editOption > 4) editOption = 0;
  } else if (!editing) {
   if (newEncoderPos > oldEncoderPos) currentScreen++;
```

```
    else currentScreen--;
    if (currentScreen < 0) currentScreen = NUM_SCREENS - 1;
    if (currentScreen >= NUM_SCREENS) currentScreen = 0;
   }
   oldEncoderPos = newEncoderPos;
   menuEncoder.clearCount();
   updateDisplay();
  }
}

void handleEncoderButton() {
  if (menuButtonDebouncer.fell()) {
   if (currentScreen == 0) {
    currentScreen = 1;
    editOption = 0;
    editing = false;
   } else if (currentScreen == 1) {
    editing = !editing;
   } else if (currentScreen == 3) {
    wifiEnabled = !wifiEnabled;
    currentScreen = 0;
   } else {
    currentScreen = 0;
   }
   menuEncoder.clearCount();
   updateDisplay();
  }
}

// Función para cambiar de pantalla (ejemplo, un botón físico de "MENU")
void cycleScreens() {
  currentScreen = (currentScreen + 1) % 3; // 0:Principal, 1:Wifi, 2:Config
  currentMenuOption = 0; // Resetear la opción al cambiar de pantalla
```

```
  editingValue = false; // Asegurarse de que no estamos editando al
cambiar de pantalla
  updateDisplay();
}


void setupMotors() {
  motor1.setMaxSpeed(1000.0);    // Ajusta la velocidad máxima según tus
NEMA y driver
  motor1.setAcceleration(500.0); // Ajusta la aceleración
  motor2.setMaxSpeed(1000.0);
  motor2.setAcceleration(500.0);
  motor3.setMaxSpeed(1000.0);
  motor3.setAcceleration(500.0);

  pinMode(ENABLE_MOTORS_PIN, OUTPUT);
  digitalWrite(ENABLE_MOTORS_PIN, LOW); // Activar drivers (común para
muchos drivers)
}

// Función para mover el perfil una distancia específica desde la posición
actual del encoder del perfil
void moveProfileBy(float distance_m) {
  long targetSteps = (long)(distance_m * stepsPerMeter);
  long currentSteps = profileEncoder.getCount(); // Tomar la lectura actual
como referencia

  motor1.moveTo(currentSteps + targetSteps);
  motor2.moveTo(currentSteps + targetSteps);
  motor3.moveTo(currentSteps + targetSteps);

  Serial.print("Moviendo perfil a ");
  Serial.print(distance_m);
```

```
  Serial.println(" metros.");

  while (motor1.distanceToGo() != 0 || motor2.distanceToGo() != 0 ||
motor3.distanceToGo() != 0) {
    motor1.run();
    motor2.run();
    motor3.run();
  }
  Serial.println("Movimiento completado.");
}

// Función para realizar el corte
void performCut() {
  Serial.println("Realizando corte...");
  // 1. Detener motores de tracción
  motor1.stop(); motor2.stop(); motor3.stop();
  while(motor1.isRunning() || motor2.isRunning() || motor3.isRunning()){
    motor1.run(); motor2.run(); motor3.run();
  }

  // 2. Ejecutar pistón hidráulico (activar relé para bajar la cizalla)
  digitalWrite(CIZALLA_PIN, HIGH); // Cizalla BAJA
  Serial.println("Cizalla bajando...");
  delay(CUT_ACTIVATION_TIME_MS); // Espera el tiempo de bajada y corte

  // 3. Desactivar el pistón para que retorne a su posición inicial
  digitalWrite(CIZALLA_PIN, LOW); // Cizalla SUBE (se desactiva la presión
para que retorne)
  Serial.println("Cizalla subiendo...");

  // 4. Esperar la lectura del sensor de fin de carrera de retorno (pistón
arriba)
  unsigned long startTime = millis();
```

```
  bool pistonRetracted = false;
  while (millis() - startTime < PISTON_RETURN_TIMEOUT_MS) {
   if (digitalRead(CIZALLA_RETRAIDA_SENSOR_PIN) == HIGH) { // Asume
HIGH cuando el pistón está arriba
    pistonRetracted = true;
    break; // El pistón ha retornado
   }
   delay(10); // Pequeña espera para no saturar el CPU
  }

  if (!pistonRetracted) {
   handleError("Fallo: El piston de la cizalla no retorno a su posicion inicial.");
   return; // Detiene el ciclo si hay un error
  }
  Serial.println("Cizalla en posicion inicial (arriba).");

  // 5. Registrar el corte en la SD
  logCut(currentProfileSettings.type,
profileWidths[currentProfileSettings.widthIndex], currentCutLength);

  // 6. Decrementar cortes restantes
  if (cutsRemaining > 0) {
   cutsRemaining--;
  }
  updateDisplay(); // Actualizar la pantalla con los cortes restantes
  Serial.println("Corte completado.");
}


// Función para verificar si ambos sensores detectan el perfil
bool bothSensorsActive() {
  return (digitalRead(SENSOR_IND_1) == HIGH &&
digitalRead(SENSOR_IND_2) == HIGH); // O LOW, según tus sensores
```

```
}

// Función para verificar si el primer sensor detecta el perfil
bool firstSensorActive() {
  return (digitalRead(SENSOR_IND_1) == HIGH); // O LOW, según tu sensor
}

// Función de calibración del offset de la matriz (se llamaría desde el menú de Configuración)
void calibrateMatrixOffset() {
  Serial.println("Iniciando calibracion de offset. Coloque un perfil y presione Start.");
  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(0,0); tft.setTextColor(ILI9341_WHITE); tft.println("Calibrando Offset...");
  tft.println("Mueva perfil manualmente hasta que ambos sensores esten activos.");
  tft.println("Luego presione Start.");

  while (!bothSensorsActive()) {
    if (startButtonDebouncer.fell()) { // Si se presiona Start durante la calibración
      Serial.println("Detectado Start button durante calibracion");
      break; // Sale del bucle, asume que el perfil está posicionado
    }
    delay(50);
  }

  matrixOffsetSteps = profileEncoder.getCount(); // Guarda el conteo del encoder
  Serial.print("Offset de matriz calibrado: ");
  Serial.print(matrixOffsetSteps);
  Serial.println(" pasos.");
```

```
  logCalibration("Matrix Offset:" + String(matrixOffsetSteps));

  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(0,0); tft.setTextColor(ILI9341_WHITE); tft.println("Calibracion
OK!");
  tft.print("Offset: "); tft.print(matrixOffsetSteps); tft.println(" pasos");
  delay(2000);
  updateDisplay(); // Volver a la pantalla principal
}


void initSDCard() {
  if (!SD.begin(SD_CS)) {
    Serial.println("Fallo al inicializar la tarjeta SD!");
    return;
  } else {
    Serial.println("Tarjeta SD inicializada.");
  }
}

void logData(String filename, String data) {
  File dataFile = SD.open(filename, FILE_APPEND);
  if (dataFile) {
    dataFile.println(data);
    dataFile.close();
    Serial.print("Datos guardados en ");
    Serial.println(filename);
  } else {
    Serial.print("Error al abrir ");
    Serial.print(filename);
    Serial.println(" para escritura.");
  }
}
```

```
void logMachineState(String state) {
  String logEntry = String(millis()) + "," + state;
  logData("/machine_state.txt", logEntry);
}

void logError(String errorMsg) {
  String logEntry = String(millis()) + "," + errorMsg;
  logData("/errors.txt", logEntry);
}

void logCalibration(String calibrationData) {
  String logEntry = String(millis()) + "," + calibrationData;
  logData("/calibrations.txt", logEntry);
}

void logCut(ProfileType type, int width, float length) {
  String logEntry = String(millis()) + "," + profileTypeStrings[type] + "," +
String(width) + "," + String(length, 2);
  logData("/cuts_history.txt", logEntry);
}


void handleStartButton() {
  if (startButtonDebouncer.fell()) { // Solo si el botón acaba de ser
presionado
    if (currentState == IDLE) {
      Serial.println("Inicio de proceso solicitado.");
      logMachineState("START_REQUESTED");
      // Reiniciar contador de cortes si se inició un nuevo proceso
      if (currentCutLength > 0) {
        totalCuts = floor(currentLength / currentCutLength);
        cutsRemaining = totalCuts;
```

```
    } else {
      totalCuts = 0;
      cutsRemaining = 0;
    }
    updateDisplay(); // Actualizar la pantalla con los nuevos conteos

    currentState = WAITING_FOR_PROFILE;
    Serial.println("Estado: Esperando Perfil.");
    tft.fillScreen(ILI9341_BLACK);
    tft.setCursor(0,0); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2);
    tft.println("Esperando perfil...");
    tft.setTextSize(1);
    tft.println("Introduzca el perfil hasta el primer sensor.");

  } else if (currentState == PERFORMING_CUT_CYCLE) {
    // Si ya está cortando, el botón de inicio podría ser una
pausa/reanudación
    // Por ahora, lo dejaremos para "inicio" solamente
    Serial.println("Maquina ya en proceso de corte.");
  } else if (currentState == EMERGENCY_STOP) {
    Serial.println("La maquina esta en parada de emergencia. Resetea
primero.");
    // No hacer nada, requiere resetear la emergencia
  }
 }
}

void handleEmergencyButton() {
 if (emergencyButtonDebouncer.fell()) { // Solo si el botón acaba de ser
presionado
  if (currentState != EMERGENCY_STOP) {
    currentState = EMERGENCY_STOP;
    Serial.println("!!! PARADA DE EMERGENCIA ACTIVADA !!!");
```

```
    logError("EMERGENCY_STOP_ACTIVATED");
    // Detener inmediatamente todos los motores
    motor1.stop(); motor2.stop(); motor3.stop(); // Detiene los movimientos actuales
    digitalWrite(ENABLE_MOTORS_PIN, HIGH); // Desactiva los drivers de los motores
    digitalWrite(CIZALLA_PIN, LOW); // Asegura que la cizalla esté apagada
    tft.fillScreen(ILI9341_RED);
    tft.setCursor(0, tft.height()/2 - 10);
    tft.setTextColor(ILI9341_WHITE);
    tft.setTextSize(3);
    tft.println("EMERGENCIA");
    tft.setTextSize(2);
    tft.setCursor(0, tft.height()/2 + 20);
    tft.println("Presione Reset para continuar"); // Placeholder para un botón de reset físico
   }
  }
}

// Función para reiniciar la máquina desde la parada de emergencia
void resetEmergency() {
  if (currentState == EMERGENCY_STOP) {
   Serial.println("Reseteando de emergencia...");
   digitalWrite(ENABLE_MOTORS_PIN, LOW); // Reactiva los drivers
   currentState = IDLE;
   updateDisplay(); // Vuelve a la pantalla principal
   logMachineState("EMERGENCY_RESET");
  }
}

// Función para gestionar errores (simplificado, se puede expandir)
void handleError(String errorMsg) {
```

```
  Serial.print("ERROR: ");
  Serial.println(errorMsg);
  logError(errorMsg);
  currentState = ERROR;
  tft.fillScreen(ILI9341_RED);
  tft.setCursor(0,0); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2);
  tft.println("ERROR CRITICO!");
  tft.println(errorMsg);
  tft.println("Reinicie la maquina.");
  // Considerar un loop infinito o un reinicio forzado en errores críticos.
  while(true){
    // Esperar reinicio manual o un botón de reset
  }
}


void setup() {
  Serial.begin(115200);
  Serial.println("Iniciando maquina de corte...");

  // Configurar pines de IO
  pinMode(ENABLE_MOTORS_PIN, OUTPUT);
  pinMode(CIZALLA_PIN, OUTPUT);
  digitalWrite(CIZALLA_PIN, LOW); // Asegura que la cizalla está apagada al
inicio

  // Configurar botones con Bounce2
  startButtonDebouncer.attach(BTN_START, INPUT_PULLUP);
  startButtonDebouncer.interval(25); // Antirebote de 25ms
  emergencyButtonDebouncer.attach(BTN_EMERGENCY, INPUT_PULLUP);
  emergencyButtonDebouncer.interval(25);
  menuButtonDebouncer.attach(ENCODER_SW_PIN, INPUT_PULLUP);
  menuButtonDebouncer.interval(25);
```

```
  menuScreenButtonDebouncer.attach(BTN_MENU, INPUT_PULLUP);
  menuScreenButtonDebouncer.interval(25);

  // Configurar pin del sensor de fin de carrera de la cizalla
  pinMode(CIZALLA_RETRAIDA_SENSOR_PIN, INPUT_PULLUP); // O INPUT si
tiene pull-up/down externo

  // Inicializar display
  tft.begin();
  tft.setRotation(3); // Ajusta la rotación si es necesario
  tft.fillScreen(ILI9341_BLACK);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.setCursor(0, 0);
  tft.println("Inicializando...");

  // Inicializar encoders
  menuEncoder.attachHalfQuad(ENCODER_CLK_PIN, ENCODER_DT_PIN);
  menuEncoder.clearCount();
  profileEncoder.attachHalfQuad(PROFILE_ENCODER_A,
PROFILE_ENCODER_B);
  profileEncoder.clearCount();

  // Inicializar sensores inductivos
  pinMode(SENSOR_IND_1, INPUT);
  pinMode(SENSOR_IND_2, INPUT);

  // Inicializar motores
  setupMotors();

  // Inicializar SD
  initSDCard();
```

```
  updateDisplay(); // Dibujar la pantalla inicial
  logMachineState("POWER_ON");
}

void loop() {
  // Actualizar estados de los botones con Bounce2
  startButtonDebouncer.update();
  emergencyButtonDebouncer.update();
  menuButtonDebouncer.update();
  menuScreenButtonDebouncer.update();

  // Manejo de botones
  if (startButtonDebouncer.fell()) {
    handleStartButton();
  }
  if (emergencyButtonDebouncer.fell()) {
    handleEmergencyButton();
  }
  if (menuButtonDebouncer.fell()) {
    handleEncoderButton();
  }
  if (menuScreenButtonDebouncer.fell()) {
    cycleScreens();
  }

  handleMenuEncoder(); // Leer y procesar el encoder del menú
  handleEncoderButton();

  Serial.print("Encoder perfil: ");
  Serial.println(profileEncoder.getCount());

  // Lógica principal de la máquina basada en el estado
  switch (currentState) {
```

```
  case IDLE:
    // Esperar comando de inicio o cambios de configuración
    break;

  case WAITING_FOR_PROFILE:
    if (firstSensorActive()) {
      Serial.println("Primer sensor detecta perfil. Moviendo a posicion de
calibracion.");
      currentState = MOVING_TO_OFFSET;
      tft.fillScreen(ILI9341_BLACK);
      tft.setCursor(0,0); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2);
      tft.println("Alineando perfil...");
    }
    break;

  case MOVING_TO_OFFSET:
    if (!bothSensorsActive()) {
      // Mover lentamente hasta que ambos sensores detecten
      motor1.setSpeed(100); motor2.setSpeed(100); motor3.setSpeed(100);
// Velocidad lenta
      motor1.runSpeed(); motor2.runSpeed(); motor3.runSpeed();
    } else {
      motor1.stop(); motor2.stop(); motor3.stop();
      while(motor1.isRunning() || motor2.isRunning() || motor3.isRunning()){
// Esperar que se detengan
        motor1.run(); motor2.run(); motor3.run();
      }
      Serial.println("Ambos sensores detectan perfil. Perfil en offset de
referencia.");
      profileEncoder.setCount(matrixOffsetSteps); // Reinicia el contador
del perfil para que la "distancia 0" sea la matriz
      currentState = READY_TO_CUT;
      Serial.println("Estado: Listo para Cortar.");
```

```
      tft.fillScreen(ILI9341_BLACK);
      tft.setCursor(0,0); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2);
      tft.println("Perfil listo!");
      tft.setTextSize(1);
      tft.println("Presione Start para iniciar el ciclo.");
    }
    break;

  case READY_TO_CUT:
    if (startButtonDebouncer.fell()) { // Si se presiona Start de nuevo
      if (cutsRemaining > 0) {
        currentState = PERFORMING_CUT_CYCLE;
        Serial.println("Iniciando ciclo de corte...");
        logMachineState("CUT_CYCLE_STARTED");
        // Mover para el primer corte
        moveProfileBy(currentCutLength); // Mueve el perfil la longitud
deseada
        performCut(); // Realiza el corte
      } else {
        Serial.println("No hay cortes pendientes o la longitud de corte es
cero.");
        currentState = IDLE;
        updateDisplay();
      }
    }
    break;

  case PERFORMING_CUT_CYCLE:
    if (cutsRemaining > 0) {
      // Verificar si todavía hay perfil para cortar (el primer sensor debe
seguir activo)
      if (firstSensorActive()) {
        // Mover para el siguiente corte
```

```
        moveProfileBy(currentCutLength);
        performCut();
       } else {
        Serial.println("Perfil agotado o retirado. Fin del ciclo.");
        handleError("Perfil agotado o retirado."); // Considerar como error o
fin de operación
        currentState = IDLE; // O ERROR si lo consideras crítico
        updateDisplay();
       }
     } else {
      Serial.println("Todos los cortes completados. Fin del proceso.");
      currentState = IDLE; // Vuelve al estado inactivo
      updateDisplay();
     }
     break;

   case PAUSED:
     // Lógica de pausa
     break;

   case EMERGENCY_STOP:
     // No hacer nada, esperar que se reinicie manualmente o se resuelva la
emergencia
     break;

   case ERROR:
     // En este estado, la máquina está detenida hasta que se resuelva el
error
     break;
  }
}
```

6.2. Imágenes de Referencia