

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE CULIACÁN

DICTAMEN DE APROBACIÓN DE CONTENIDO
DEL DOCUMENTO DE TESIS

Culiacán, Sinaloa, 04/junio/2025

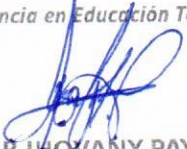
JOSE ANGEL ALCARAZ VEGA
JEFA DEL DEPTO. DE METAL-MECÁNICA
PRESENTE.

Con relación a la tesis de grado de licenciatura titulada **[Instrumentación y Adquisición de Datos para un Proceso de Tratamiento Térmico de Metales]** derivada de la LGAC Automatización, presentada el 10 de diciembre de 2024 por **Felipe de Jesús Rodríguez Murillo**, egresado del programa de Ingeniería **Mecatrónica** de esta institución con número de control 19170480, informamos a usted que el comité de tesis designado efectuó la revisión del trabajo de tesis en el marco del documento normativo vigente "Manual de Lineamientos Académico-Administrativos del Tecnológico Nacional de México", capítulo 14 Lineamiento para la Titulación Integral, y de acuerdo con lo establecido por el Comité Académico en minuta de la reunión realizada el viernes 28 de marzo de 2025 para la documentación del "Proceso de Validación de Tesis de Licenciatura para el Repositorio Institucional".

Al respecto, se dictamina que el contenido del documento de tesis cumple con todos los requisitos establecidos correspondientes, el dictamen del comité es: **APROBADO EN TODAS SUS PARTES**, por lo que se recomienda continuar con el proceso de validación para su publicación en el Repositorio Institucional.

ATENTAMENTE

Excelencia en Educación Tecnológica®


OMAR JHOVANY PAYAN SERRANO
ASESOR DE TESIS

Vo. Bo.
Revisores del Comité de Tesis


Ricardo Ernesto Domínguez Rodríguez


Ismael Díaz Peña

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE CULIACÁN

CONSTANCIA DE CUMPLIMIENTO DEL
NIVEL DE ORIGINALIDAD REQUERIDO

Culiacán, Sinaloa, 04/junio/2025

El que suscribe, jefe de Oficina de Proyectos de Investigación del Departamento de Metal-Mecánica:

HACE CONSTAR

Que, con relación a la revisión de la tesis titulada **Instrumentación y Adquisición de Datos para un Proceso de Tratamiento Térmico de Metales** derivada de la LGAC Automatización, presentada el 10 de diciembre de 2024 por **Felipe de Jesús Rodríguez Murillo**, egresado del programa de Ingeniería **Mecatrónica** de esta institución con número de control 19170480, se hace constar que el documento obtuvo un **porcentaje de similitud del 19%**, el cual **cumple** con el rango de los parámetros de similitud y la configuración de los filtros y umbrales de exclusión establecidos por el Comité Académico en minuta de la reunión realizada el viernes 28 de marzo de 2025 para la documentación del Proceso de Validación de Tesis de Licenciatura para el Repositorio Institucional. Anexo reporte de similitud.

Se expide la presente constancia en la ciudad de Culiacán Rosales, Sinaloa, a los cuatro días del mes de junio del año dos mil veinticinco.

ATENTAMENTE

Excelencia en Educación Tecnológica®


OMAR JHOVANY PAYÁN SERRANO
JEFE DE OFICINA DE PROYECTOS DE
INVESTIGACIÓN

Vo. Bo.


JOSE ANGEL ALCARAZ VEGA
JEFE DE METAL-MECÁNICA

LICENCIA DE USO OTORGADA POR **Felipe de Jesús Rodríguez Murillo** de nacionalidad Mexicana mayor de edad, con domicilio ubicado **calle Av. Coras 2820, Col. Industrial El Palmito, C.P. 80160, Culiacán, Sinaloa**, en; en mi calidad de titular de los derechos patrimoniales y morales y autor de la tesis denominada **"Instrumentación y adquisición de datos para un proceso de tratamientos térmico de metales"** en adelante **"LA OBRA"** quien paratodos los fines del presente documento se denominará **"EL AUTOR Y/O EL TITULAR"**, a favor del Instituto Tecnológico de Culiacán del Tecnológico Nacional de México, la cual se registrá por las cláusulas siguientes:

PRIMERA –OBJETO: **"EL AUTOR Y/O TITULAR"**, mediante el presente documento otorga al Instituto Tecnológico de Culiacán del Tecnológico Nacional de México, licencia de uso gratuita e indefinida respecto de **"LA OBRA"**, para almacenar, preservar, publicar, reproducir y/o divulgar la misma, con fines académicos, por cualquier medio en forma física y a través del repositorio institucional y del repositorionacional, éste último consultable en la página: (<https://www.repositorionacionalcti.mx/>).

SEGUNDA - TERRITORIO: La presente licencia se otorga, de manera no exclusiva, sin limitación geográfica o territorial alguna, de manera gratuita e indefinida.

TERCERA -ALCANCE: La presente licencia contempla la autorización para formato uso de **"LA OBRA"** en cualquier formato o soporte material y se extiende a la utilización, de manera enunciativa más no limitativa alos siguientes medios: óptico, magnético, electrónico, virtual (red), mensaje de datos o similar conocido por conocerse en medio óptico, magnético, electrónico, en red, mensajes de datos o similar, conocido o por conocerse.

CUARTA – EXCLUSIVIDAD: La presente licencia de aquí establecida no implica exclusividad en favor del Instituto Tecnológico de Culiacán; por lo tanto, **"EL AUTOR Y/O TITULAR"** conserva los derechos ~~patrimoniales~~ y morales de **"LA OBRA"**, objeto del presente documento.

QUINTA – CRÉDITOS: El Instituto Tecnológico de Culiacán y/o el Tecnológico Nacional de México reconoce que el **"AUTOR Y/O TITULAR"** es el único, primigenio y perpetuo titular de los derechos morales sobre **"LA OBRA"**; por lo tanto, siempre deberá otorgarle los créditos correspondientes por la autoría de la misma.

SEXTA – AUTORÍA: **"EL AUTOR Y/O TITULAR"** manifiesta ser el único titular de los derechos de autor que derivan de **"LA OBRA"** y declara que el material objeto del presente fue realizado por él, sin violentar o usurpar derechos de propiedad intelectual de terceros; por lo tanto, en caso de controversia sobre los mismos, se obliga a ser el único responsable.

Dado en la ciudad de Culiacán Rosales, Sinaloa, a los días 6 del mes de junio del 2025.

"EL AUTOR Y/O TITULAR"

FELIPE DE JESUS RODRIGUEZ MURILLO

"EL INSTITUTO TECNOLÓGICO DE CULIACÁN"

FRANCISCA PIÑA ZAZUETA
DIRECTORA

TECNOLÓGICO NACIONAL DE MEXICO

Instituto Tecnológico de Culiacán



TITULACIÓN INTEGRAL

OPCION:

TESIS PROFESIONAL

NOMBRE DEL PROYECTO:

**INSTRUMENTACIÓN Y ADQUISICIÓN DE DATOS PARA UN
PROCESO DE TRATAMIENTO TÉRMICO DE METALES.**

ALUMNO:

Rodríguez Murillo Felipe de Jesús.

CARRERA:

Ingeniería Mecatrónica.

Num. De Control.

19170480.

Culiacán, Sinaloa (2025)



Instituto Tecnológico de Culiacán

DIVISION DE ESTUDIOS PROFESIONALES
ASUNTO: **Autorización de Impresión**

Culiacán, Sinaloa, 24/03/2025 10:59

C. FELIPE DE JESUS RODRIGUEZ MURILLO
PASANTE(S) DE LA CARRERA DE INGENIERÍA MECATRÓNICA
P R E S E N T E.

De acuerdo con fallo emitido por la Comisión Revisora Integrada por:

Cédula Profesional

Presidente:	Doctor en Ciencias de la Ingeniería Omar Jhovany Payan Serrano	11461565
Secretario:	Maestro en Ciencias en Materiales Ricardo Ernesto Domínguez Rodríguez	5583942
Vocal:	Doctor en Ingeniería de Materiales Ismael Díaz Peña	9798300

Y considerando que reúne los requisitos establecidos en la Opción: **TITULACIÓN INTEGRAL: TESIS PROFESIONAL "INSTRUMENTACIÓN Y ADQUISICIÓN DE DATOS PARA UN PROCESO DE TRATAMIENTOS TÉRMICO DE METALES"**, damos a usted nuestra autorización para que proceda a imprimir su trabajo profesional.

Sin otro particular, quedo de usted.

ATENTAMENTE

Excelencia en Educación Tecnológica®

LIC. CONCEPCIÓN MENDOZA ROSALES

JEFA DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CULIACÁN

DEPARTAMENTO DE DIVISIÓN DE
ESTUDIOS PROFESIONALES



2025
Año de
La Mujer
Indígena

Juan de Dios Bátiz 310 Pte. Col. Guadalupe C.P. 80220
Culiacán, Sinaloa. Tel. 667-454-0100
tecnm.mx www.culiacan.tecnm.mx



DEDICATORIA

El presente trabajo lo dedico principalmente a mi familia, quienes son los que siempre me han brindado su apoyo a lo largo de mi carrera, formación profesional y personal. Gracias a ellos siempre he podido contar con todos los recursos necesarios para cruzar todas y cada una de mis etapas académicas y personales, hasta encontrarme aquí redactando este documento para la culminación de mi etapa universitaria. Además de brindarme todo lo que me fue necesario en recursos, también me dieron su apoyo incondicional y amor cuando la situación no era la mejor o presentaba diferentes problemas, hasta poder decir que sin su ayuda en algunas ocasiones salir de dichas situaciones pudo haber sido más complicado. Los quiero mucho y muchas gracias por estar a mi lado cuando más los he necesitado. Los amo.

También quiero dedicarles este trabajo a mis amigos, a quienes, a lo largo de los años en igual de separarnos, nos volvimos más unidos, que a pesar de no estar presentes todos los días en nuestras vidas, sé que cuando los necesite estarán para mí, porque cuando uno de ellos ocupe de mí, estaré ahí. Los quiero mucho.

Por último, también dedico este trabajo a mi pareja actual, porque a pesar de no saber lo que el futuro nos depara, también es cierto que me ofreciste tu apoyo incondicional, y en momentos difíciles de mi vida, eso fue de una gran ayuda. Te amo.

INSTRUMENTACIÓN Y ADQUISICIÓN DE DATOS PARA UN PROCESO DE TRATAMIENTO TÉRMICO DE METALES

RESUMEN

En el laboratorio de tratamientos térmicos de metales del Tecnológico Nacional de México campus Culiacán, se encuentra una mufla con una alta capacidad de calentamiento, sin embargo, no se cuenta con una herramienta capaz de medir la temperatura interna de la probeta en análisis, por lo que, los alumnos no pueden realizar estos tratamientos a un nivel profesional. En el presente trabajo se desarrolló un dispositivo mecatrónico, que, mediante dos termopares tipo K, un par de módulos *MAX6675* y un *ESP32*, capta la temperatura interna de una probeta de acero *AISI 1040* y la muestra en una gráfica al igual que su diferencial, en una página *web*, la cual, esta alojada en un servidor asíncrono creado por el microcontrolador, cuyas peticiones y eventos son manejados mediante los *WebSockets*. En la programación también se incluye la generación de un *DNS* propio para el sitio *web*, como también la creación de una red *WiFi* para que los usuarios conectados a esta sean quienes puedan acceder a la interfaz *web*. Toda esta información del ensayo es manejada mediante objetos tipo *JSON* y por vectores ambos dinámicos para que tomen el tamaño en función de la duración del ensayo. Por último, para la elaboración de este dispositivo, se diseñó su circuito esquemático y *PCB*, para posteriormente mecanizarlo mediante una fresadora *CNC*, y la caja donde fue almacenado, de igual manera, se diseñó en 3D para su posterior impresión.

Palabras clave: *ESP32*, *Termopar*, *MAX6675*, *Arduino*, *ESPAsyncWebServer*, *WebSockets*.

CONTENIDO

1	INTRODUCCIÓN	1
1.1	Antecedentes	1
1.2	Descripción del problema.....	2
1.3	Hipótesis o preguntas de investigación	3
1.4	Objetivo general	3
1.5	Objetivos específicos	3
1.6	Justificación	3
1.7	Alcance.....	4
2	FUNDAMENTO TEÓRICO	5
2.1	Tratamientos térmicos en aceros.....	5
2.1.1	Templado	5
2.2	Acero ASTM A36.....	8
2.3	Acero 1040	8
2.4	Horno mufla.....	9
2.4.1	Principio de funcionamiento de un horno mufla.....	10
2.5	Termopar	11
2.5.1	Principio de funcionamiento del termopar	12
2.5.2	Tipos de termopares	13
2.6	MAX6675.....	15
2.6.1	Composición del módulo MAX6675.....	16
2.6.2	Funcionamiento del módulo MAX6675.....	17
2.7	Protocolo <i>SPI</i>	18

2.8	Microcontroladores.....	19
2.8.1	ESP32	20
2.9	IDE de programación	23
2.9.1	IDE de Arduino.....	25
2.9.2	Visual Studio Code	30
2.10	HTML.....	32
2.11	JavaScript.....	34
2.11.1	Chart.js.....	36
2.12	CSS	37
2.13	PCB	39
2.13.1	Proteus.....	40
2.13.2	FlatCAM	41
2.14	CNC.....	42
2.14.1	Candle.....	46
2.15	Impresoras 3D.....	47
2.15.1	Impresoras FDM o FFF.....	47
2.16	UltiMaker Cura	50
2.17	SolidWorks	50
3	PROCEDIMIENTO	52
3.1	Elaboración de la página <i>web</i>	52
3.1.1	Desarrollo del archivo <i>HTML</i>	52
3.1.2	Elaboración del archivo <i>CSS</i>	54
3.1.3	Creación del archivo <i>JavaScript</i>	58
3.2	Programación del ESP32.....	68

3.3	Elaboración de la <i>PCB</i>	78
3.3.1	Esquemático	78
3.3.2	<i>PCB Layout</i>	79
3.4	Generación de código G	81
3.5	Construcción de la <i>PCB</i>	86
3.6	Caja del dispositivo mecatrónico.....	93
3.6.1	Modelado 3D de la caja y tapadera	93
3.6.2	Impresión 3D de la caja y tapadera	94
3.7	Ensamble final del proyecto	97
4	RESULTADOS	100
4.1	Preparación del dispositivo mecatrónico	100
4.2	Prueba del dispositivo mecatrónico	101
4.3	Gráficas finales.....	105
5	CONCLUSIONES.....	107
6	COMPETENCIAS DESARROLLADAS.....	109
7	FUENTES DE INFORMACIÓN.....	110
8	ANEXOS	117
8.1	Instalación de la extensión <i>Live Server</i> en el <i>IDE</i> de <i>Visual Studio Code</i>	117
8.2	Código <i>HTML</i>	118
8.3	Código <i>CSS</i>	121
8.4	Descarga de librería <i>Chart.js</i>	124
8.5	Código <i>JavaScript</i>	124
8.6	Soporte para <i>ESP32</i> en <i>Arduino IDE</i>	147
8.7	Instalación de librerías de <i>Arduino</i>	149

8.7.1	Librería Wifi.h	149
8.7.2	Librería ESPAsyncWebServer.h.....	150
8.7.3	Librería SPIFFS.h	151
8.7.4	Librería WebSocketsServer.h.....	153
8.7.5	Librería SPI.h	153
8.7.6	Librería ESPmDNS.h	153
8.7.7	Librería ArduinoJson.h.....	154
8.8	Código del <i>ESP32</i>	154
8.9	Creación del microcontrolador <i>ESP32</i> en <i>Proteus</i>	176
8.10	Diseño 3D de la caja y tapadera del dispositivo mecatrónico.....	178

ÍNDICE DE FIGURAS Y TABLAS

Figura 1: Diagrama hierro-carbono.	6
Figura 2: Representación gráfica de las etapas de calentamiento para un temple.	7
Figura 3: Ejemplo de horno mufla.	9
Figura 4: Componentes de un horno mufla.	10
Figura 5: Ejemplos de termopares.	11
Figura 6: Diagrama interno de un termopar.	11
Figura 7: Representación del voltaje en presencia de calor.	12
Figura 8: Liberación o absorción de temperatura causada por voltaje.	13
Figura 9: Representación del efecto <i>Thomson</i>	13
Figura 10: Módulo MAX6675.	16
Figura 11: Diagrama de conexión del módulo MAX6675.	17
Figura 12: Definición de los bits enviados por el módulo MAX6675.	17
Figura 13: Diagrama de pulsos en la comunicación <i>SPI</i>	19
Figura 14: Diagrama de pines del <i>ESP32-DevKitC</i>	23
Figura 15: Ejemplo de la interfaz del <i>IDE</i> de <i>Arduino</i>	25
Figura 16: Declaración de la librería <i>WiFi.h</i> en <i>Arduino</i>	26
Figura 17: Declaración de la librería <i>ESPAsyncWebServer.h</i> en <i>Arduino</i>	27
Figura 18: Declaración de la librería <i>WebSocketsServer.h</i> en <i>Arduino</i>	28
Figura 19: Declaración de la librería <i>SPIFFS.h</i> en <i>Arduino</i>	28
Figura 20: Declaración de la librería <i>SPI.h</i> en <i>Arduino</i>	29
Figura 21: Declaración de la librería <i>ESPmDNS.h</i> en <i>Arduino</i>	29
Figura 22: Declaración de la librería <i>ArduinoJson.h</i> en <i>Arduino</i>	29
Figura 23: Declaración de la librería <i>vector</i> en <i>Arduino</i>	30
Figura 24: Interfaz gráfica del <i>IDE</i> de <i>Visual Studio Code</i>	30
Figura 25: Logotipo de <i>HTML</i>	33
Figura 26: Estructura de un archivo <i>HTML</i>	34

Figura 27: Logotipo de <i>JavaScript</i> .	34
Figura 28: Logotipo de la librería <i>Chart.js</i> .	36
Figura 29: Ejemplo de un gráfico mixto hecho con <i>Chart.js</i> .	37
Figura 30: Logotipo de <i>CSS</i> .	38
Figura 31: Placa de circuito impreso.	39
Figura 32: Baquelita virgen de una sola cara.	40
Figura 33: Interfaz gráfica del software de diseño de <i>PCB</i> de <i>Proteus</i> .	41
Figura 34: Interfaz gráfica del software <i>FlatCAM</i> .	42
Figura 35: Enrutador <i>CNC</i> modelo KT-1325.	43
Figura 36: Cortador <i>CNC</i> de plasma.	44
Figura 37: Cortadora láser <i>CNC</i> de <i>CO2</i> .	44
Figura 38: Torno <i>CNC</i> .	45
Figura 39: Fresadora <i>CNC</i> .	45
Figura 40: Interfaz gráfica de <i>Candle</i> con un proyecto de ejemplo.	46
Figura 41: Impresora <i>FFF</i> del tipo cartesiana.	48
Figura 42: Impresora <i>FFF</i> del tipo <i>CoreXY</i> .	48
Figura 43: Impresora <i>FFF</i> del tipo <i>delta</i> .	49
Figura 44: Impresora <i>FFF</i> del tipo polar.	49
Figura 45: Interfaz gráfica de <i>UltiMaker Cura</i> .	50
Figura 46: Interfaz gráfica de <i>SolidWorks</i> .	51
Figura 47: Contenido de la etiqueta <i>head</i> .	53
Figura 48: Contenido de las etiquetas <i>body</i> .	54
Figura 49: Color de fondo para la etiqueta <i>body</i> en el archivo <i>CSS</i> .	55
Figura 50: Propiedades de las etiquetas de los títulos.	55
Figura 51: Propiedades de la etiqueta <i>td</i> para las columnas de las tablas.	55
Figura 52: Características de estilo para los selectores o <i>comboboxes</i> .	56
Figura 53: Propiedades para las cajas o entradas de texto.	56
Figura 54: Propiedades para la clase <i>boton</i> .	57
Figura 55: Propiedades para los <i>canvas</i> .	57
Figura 56: Evento para que el <i>script</i> se ejecute una vez cargado el <i>DOM</i> .	58

Figura 57: Variables utilizadas a lo largo del <i>BackEnd</i> de la página <i>web</i>	58
Figura 58: Constantes de elementos de la página <i>web</i>	59
Figura 59: Evento de cambio para el <i>combobox</i> de la cantidad de termopares.	59
Figura 60: Función para asignar el evento de descarga a los botones de las gráficas.	61
Figura 61: Evento clic para el botón de empezar.	62
Figura 62: Funciones para deshabilitar algunos botones y la información del ensayo.	62
Figura 63: Evento clic para el botón de reiniciar.	63
Figura 64: Función para la destrucción de las gráficas.	64
Figura 65: Función para habilitar el ingreso de datos.	64
Figura 66: Evento clic para el botón finalizar.	65
Figura 67: Función para habilitar los botones de control del sitio <i>web</i>	65
Figura 68: Función para obtener las gráficas antes de mostrarlas en la <i>web</i>	66
Figura 69: Sitio <i>web</i> en la espera del ingreso de la información.	67
Figura 70: Figuras usadas en la programación del <i>ESP32</i>	69
Figura 71: Definición de pines <i>CS</i> para módulos <i>MAX6675</i>	69
Figura 72: Declaración de los tres <i>LEDs</i> indicadores.	69
Figura 73: Declaración de variables que se usarán en la lógica del programa.	70
Figura 74: Vectores para el almacenamiento de los datos de las mediciones.	70
Figura 75: Declaración de los servidores y sus puertos.	71
Figura 76: Esquemático del circuito del dispositivo mecatrónico.	78
Figura 77: Diseño de la <i>PCB</i>	80
Figura 78: Generación de archivos para el mecanizado de la <i>PCB</i>	80
Figura 79: Archivo <i>Gerber</i> de la capa de cobre inferior de la <i>PCB</i>	81
Figura 80: Calculo del diámetro efectivo de corte de la broca.	81
Figura 81: Geometría de las rutas de la broca para la eliminación de cobre.	82
Figura 82: Código <i>G</i> de la placa generado por <i>FlatCAM</i>	83
Figura 83: Cara inferior de la <i>PCB</i> junto a los taladros a realizar.	84
Figura 84: Tabla de dimensiones de las brocas a usar en el proyecto.	84
Figura 85: Código <i>G</i> para las trayectorias de perforación.	85
Figura 86: Trayectorias en código <i>G</i> para perforar la <i>CNC</i>	85

Figura 87: Código G para el corte de la <i>PCB</i>	86
Figura 88: Fresadora CNC 1610.	87
Figura 89: Configuración para la comunicación con la máquina <i>CNC</i>	87
Figura 90: Código G para las pistas de la <i>PCB</i>	88
Figura 91: Botón para crear un mapa de alturas.	88
Figura 92: Mapa de alturas obtenido automáticamente.....	89
Figura 93: Eliminación del cobre excedente de la placa de circuito.	89
Figura 94: Perforaciones para puentes de la cara superior.	90
Figura 95: <i>PCB</i> perforada y cortada.	91
Figura 96: Placa de circuito impreso limpia para soldadura.	91
Figura 97: <i>PCB</i> finalizada.....	92
Figura 98: Plano de la caja donde se encontrará el dispositivo.	93
Figura 99: Plano de la tapadera de la caja.	94
Figura 100: Modelos <i>STL</i> para su impresión 3D.	95
Figura 101: Parámetros de impresión.	96
Figura 102: Impresión 3D de la caja para el dispositivo mecatrónico.	96
Figura 103: Indicadores LED para interfaz.	97
Figura 104: Placa de circuito fijada dentro de la caja impresa en 3D.....	97
Figura 105: Indicadores LED montados en la tapadera de la caja impresa en 3D.	98
Figura 106: Dispositivo con los termopares tipo K y el cable de alimentación conectados.	98
Figura 107: Dispositivo mecatrónico cerrado.....	99
Figura 108: Termopar tipo K.	100
Figura 109: Serigrafiado para la identificación del termopar principal y secundario.	100
Figura 110: Probeta de hacer AISI 1040 con barreno de 1/16 de pulgada.	101
Figura 111: Probeta con el termopar K en el centro de ésta, dentro de la mufla.	101
Figura 112: Panel de control del horno mufla.	102
Figura 113: Interfaz gráfica del dispositivo mecatrónico con la configuración elegida.	102
Figura 114: Nombre de la red <i>WiFi</i> generada por el dispositivo mecatrónico.	103
Figura 115: Gráfica de la temperatura interna de la probeta.	103
Figura 116: Gráfica del diferencial de temperatura interna de la probeta.	104

Figura 117: Probeta sumergida en agua a temperatura ambiente para su enfriamiento. .	104
Figura 118: Gráfica de la temperatura interna de la probeta al finalizar el ensayo.	105
Figura 119: Gráfica del diferencial de temperatura interna de la probeta al finalizar el ensayo.....	106
Figura 120: Ubicación del ícono de extensiones del <i>IDE</i> de <i>Visual Studio Code</i>	117
Figura 121: Resultado de la búsqueda de la extensión <i>Live Server</i>	117
Figura 122: Botón de instalación de la extensión <i>Live Server</i>	118
Figura 123: Dirección <i>URL</i> de la librería <i>Char.js</i>	124
Figura 124: Dirección <i>URL</i> para el gestor de tarjetas de <i>ESP32</i> para <i>Arduino</i>	147
Figura 125: Menú desplegable de opción "Archivo" de <i>Arduino IDE</i>	147
Figura 126: Ventana de preferencias de <i>Arduino IDE</i> con el gestor de tarjetas <i>ESP32</i> . ..	148
Figura 127: Ubicación del gestor de tarjetas de <i>Arduino IDE</i>	148
Figura 128: Instalación del paquete de tarjetas <i>ESP32</i> por <i>Espressif Systems</i>	149
Figura 129: Instalación de librería <i>WiFi.h</i>	149
Figura 130: Repositorio <i>GitHub</i> para la descarga de la librería <i>ESPAsyncWebServer</i>	150
Figura 131: Hipervínculos de las dependencias de la librería según la tarjeta usada.	150
Figura 132: Ubicación de la opción para la instalación manual de librerías.	151
Figura 133: Mensaje para una instalación correcta de la librería.	151
Figura 134: Instalación de librería <i>SPIFFS.h</i>	152
Figura 135: Ubicación de la opción " <i>ESP32 Sketch Data Upload</i> ".	152
Figura 136: Instalación de librería <i>SPI.h</i>	153
Figura 137: Instalación de librería <i>ESPmDNS.h</i>	153
Figura 138: Instalación de librería <i>ArduinoJson.h</i>	154
Figura 139: Botón de descarga para crear el <i>ESP32</i> en <i>Proteus</i>	176
Figura 140: Selección de opción "Make Package" para generar el <i>PCB</i> del <i>ESP32</i>	177
Figura 141: Selección de opción "Make Device" para generar el esquemático del <i>ESP32</i>	177
Figura 142: Diseños en 3D en el software <i>SolidWorks</i> para la caja y tapadera.	178

Tabla 1: Composición química del acero ASTM A36.	8
Tabla 2: Composición química del acero ASTM A36.	9
Tabla 3: Tabla de especificaciones de los termopares J, K y T.	14

1 INTRODUCCIÓN

1.1 Antecedentes

Desde hace tiempo el tratamiento térmico de templado ha sido objeto de estudio, debido a que este proceso térmico es crucial en la industria metalúrgica. Dicho proceso consta de un calentamiento controlado durante un tiempo específico y también de un enfriamiento rápido para así mejorar sus propiedades mecánicas (Saldaña, 2022).

Algunas de las propiedades mecánicas a mejorar pueden ser una mayor dureza, tenacidad o resistencia al desgaste, para ello, el material debe de pasar por un cambio en su microestructura, a su vez, para lograrlo, el tratamiento térmico se debe de llevar a cabo sometiendo al material a ciertas temperaturas, controlando en todo momento las velocidades de calentamiento y enfriamiento, así como también los tiempos de sostenimiento (Romero & Vergara, 2018).

Es por ello por lo que el monitorear la temperatura de la probeta a tratar durante el proceso térmico es algo vital, lo que crea la necesidad de utilizar sensores y dispositivos electrónicos capaz de procesar la información obtenida, esto se puede ver reflejado en el trabajo de Agama, Saa, Baldeón & Bajaña (2022) en “Servidor Web local con Autenticación y Administración de Usuarios basado en ESP32 para Viviendas Inteligentes” en donde se menciona que el Internet de las Cosas (*IoT*, por sus siglas en inglés), es un recurso muy utilizado en la actualidad, ya que gracias a este han aparecido una gran diversidad de aparatos electrónicos capaces de ser controlados desde un *smartphone* que cuenten con la conectividad a la red de internet o inclusive sin ella, sin embargo, la mayoría de estos equipos son propiedad intelectual de un fabricante en especial, por lo que la utilización de estos se convierte en un ecosistema cerrado y de alto costo, presentando así una alternativa con un microcontrolador de bajo costo llamado *ESP32*. En este trabajo además también se nos plantea el uso de este dispositivo como un servidor tipo *Access Point* para presentar una interfaz web a la que el usuario puede conectarse desde un navegador web para monitorear el estado de distintos sensores y

controlar varios recursos, todo eso usando librerías propias del microcontrolador relacionadas a servidores y servidores asíncronos.

Otros autores han utilizado ese mismo recurso para crear servidores con ese microcontrolador para la creación de interfaces con el usuario para control y monitoreo, tales como Jorda (2023) en “Estudio, propuesta y prototipado de solución para la geolocalización y orientación en interiores para personas con discapacidad” en donde utilizando el protocolo *WebSocket ESPAsynWebServer* crea instancias asíncronas con la finalidad de esperar el recibimiento de peticiones de conexión mediante *HTTP*. Y Garrido, Sepúlveda & Vega (2021) en “Diseño e implementación de un sistema inalámbrico de comunicación entre un ambiente virtual y un agente en el mundo real” utilizan el protocolo *WebSocket* para permitir una comunicación bidireccional entre el cliente y el servidor, esto mediante eventos del mismo protocolo para así lograr que el dispositivo ejecutara las acciones correctas en función de las lecturas de los sensores.

Por último, Velázquez & Martínez (2018) en “FABRICACIÓN E INSTRUMENTACIÓN DE HORNO CON PERFIL ELIPSOIDAL PARA CALIBRACIÓN DE TERMOPAR, CON APLICACIÓN A PROCESOS DE CORTE CONTINUO” mencionan un módulo conversor análogo para termopares tipo K llamado MAX6675, el cual utilizan para lograr obtener la temperatura del termopar en conjunto de un microcontrolador *Arduino*, que si bien no es el mismo que el *ESP32* sí utilizan el mismo entorno y lenguaje de programación.

1.2 Descripción del problema

Actualmente, en el laboratorio de tratamientos térmicos de metales del Tecnológico Nacional de México campus Culiacán, se cuenta con una mufla capaz de realizar tratamientos térmicos para metales, debido a que cuenta con una capacidad de calentamiento alta, sin embargo, aunque dicho dispositivo cuente con un control de temperatura ideal para su función, este mismo, no cuenta con ninguna herramienta que nos brinde información sobre la temperatura interna de la probeta metálica durante el proceso de templado, por consiguiente, el ingeniero en preparación, es incapaz de realizar un templado ingenieril y realiza dicho proceso de forma informal, lo cual es grave, debido a que en el

mundo laboral y profesional los templados y tratamientos térmicos a metales se realizan de la forma más controlada y profesional posible, por lo que el ingeniero debe de prepararse para conocer el proceso de realizar dichos tratamientos.

1.3 Hipótesis o preguntas de investigación

El desarrollo de un dispositivo de medición de temperatura basado en el uso del microcontrolador ESP32, módulos MAX6675 y termopares tipo k, con una interfaz web para el control y visualización de datos, permitirá crear una herramienta versátil y adaptable para monitorear tratamientos térmicos ingenieriles.

1.4 Objetivo general

Permitir medir, graficar y guardar la temperatura interna a la que se encuentra la probeta dentro de una mufla cada periodo de tiempo para un temple correcto.

1.5 Objetivos específicos

- Medir la temperatura interna de la probeta.
- Graficar las mediciones de temperatura.
- Graficar el diferencial de temperatura.
- Guardar las gráficas para su análisis.

1.6 Justificación

Este proyecto permite realizar temples a metales con mayor precisión y de carácter ingenieril, gracias al monitoreo en tiempo real de la temperatura interna del metal. Por lo que, el ingeniero en formación podrá adquirir las habilidades y conocimientos necesarios para realizar este tipo de tratamientos térmicos de la forma más profesional posible, además, gracias a las gráficas, podrá

realizar un análisis exhaustivo del comportamiento del metal a lo largo del proceso, es su calentamiento y su enfriamiento.

1.7 Alcance

Este proyecto partirá desde la obtención del diferencial de potencial entregado por un termopar tipo K, o por dos termopares según lo decida el estudiante, de la temperatura interna de un metal dentro de una mufla, hasta el graficar dicho dato en dos gráficas, una para la temperatura con relación al tiempo y la segunda para el diferencial de temperatura a lo largo del tiempo para cada termopar, además de si se realiza el estudio con dos termopares, también se graficará la comparación de las temperaturas (se juntan las gráficas de temperatura individuales en una sola) y la comparación de los diferenciales de temperatura (se juntan las gráficas del diferencial de temperatura en una sola). Se utilizará un *ESP32* para crear un servidor de tipo *Access Point* para así alojar una página web creada con los lenguajes *HTML*, *JavaScript* y *CSS*, en donde se graficarán mediante la librería *CHART.JS* los resultados de la medición con uno o dos termopares tipo K y los conversores MAX6675.

2 FUNDAMENTO TEÓRICO

2.1 Tratamientos térmicos en aceros

Los tratamientos térmicos en aceros son procesos utilizados con el propósito de modificar o alterar las propiedades físicas y mecánicas del material en cuestión. Estas técnicas involucran el calentamiento y el enfriamiento, los cuales deben de ser estrictamente controlados, para así obtener características específicas deseadas según el proceso, como lo puede ser una mayor dureza, resistencia, tenacidad y durabilidad (Saldaña, 2022).

Para lograr aplicar correctamente un tratamiento térmico es de suma importancia el conocer las características del material a someter, así como también su estructura y composición química. Otro punto que procurar es el de que toda la sección de la pieza sometida alcance la temperatura de transformación necesaria al mismo tiempo, esto se debe a que al lograrlo se obtiene una estructura final homogénea, por lo anterior se entiende que la geometría de la pieza es otro factor para considerar y de gran importancia. Ya que, con una pieza de sección transversal delgada, el calentamiento debe de ser lento para evitar distorsiones y tensiones térmicas (Romero & Vergara, 2018).

En el presente trabajo se analizará el comportamiento de los aceros ASTM A36 y 1040, esto debido a que son con los que se cuentan en el laboratorio del instituto, además de que el tratamiento térmico que se les aplicará será un proceso de templado, por último, el medio que se usará para calentar los materiales será una mufla.

2.1.1 Templado

El proceso térmico de templado consiste en calentar el acero a una temperatura crítica y luego enfriarlo rápidamente utilizando un aceite o agua como medio de enfriamiento. Esta técnica da como

resultado que la estructura del acero sea martensítica, por lo que la dureza es mayor al igual que su fragilidad y resistencia, es decir, este proceso se utiliza con el fin de aumentar la dureza y resistencia del acero, pero a costa de la tenacidad (las fases de este proceso junto a sus propiedades y temperaturas se muestra en el diagrama de la figura 1). En este proceso la precisión con la que se controla el enfriamiento es crítico para así obtener las propiedades deseadas (Saldaña, 2022).

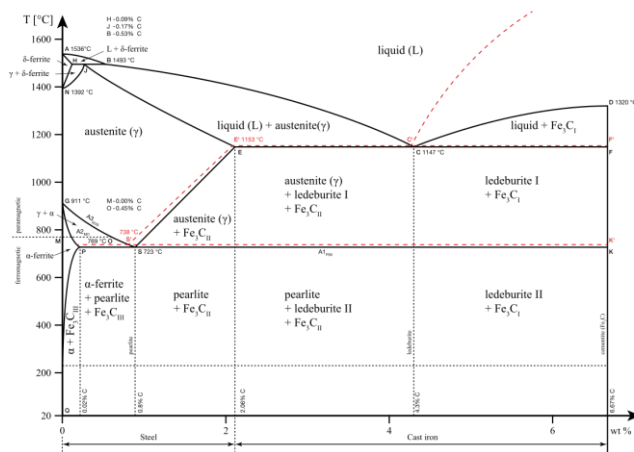


Figura 1: Diagrama hierro-carbono.

Fuente: https://es.wikipedia.org/wiki/Diagrama_hierro-carbono

Según Hernández & Laverde (2019), el templado es un tratamiento térmico en el que el acero debe calentarse hasta una temperatura óptima durante un determinado tiempo, para después enfriarlo rápidamente a una velocidad mayor que la crítica, la cual es la velocidad mínima de enfriamiento. Además, mencionan que el templado se caracteriza por enfriamientos rápidos que pueden ser de manera continua o escalonados. Por último, estos autores establecen los objetivos de realizar un templado al acero, los cuales son:

- Aumentar la dureza y resistencia mecánica.
- Disminuir la tenacidad.
- Disminuir el alargamiento unitario.
- Modificar algunas propiedades eléctricas, magnéticas y químicas.

2.1.1.1 Etapas del templeado

Para el proceso térmico del templeado se dividen en 3 etapas:

1. Calentamiento: En esta etapa el acero debe de calentarse lenta y uniformemente, esto con el objeto de evitar distorsión del material o en algunos casos la fractura de este cabe destacar que la velocidad en cuestión se determina por la conductividad térmica del acero sometido, así como también por su geometría, siempre buscando que la temperatura interna sea lo más igual a la superficial. En la figura 2 se representa gráficamente el comportamiento de esta etapa (González, 2022).
2. Mantenimiento o permanencia: En esta etapa, se mantiene el acero a una temperatura deseada y de forma constante, durante un tiempo establecido, el cual lo determina su composición química y su sección transversal. Lo anterior puede verse en la figura 2 (González, 2022).
3. Enfriamiento: Esta etapa consiste en enfriar el material a una velocidad dada por las propiedades finales deseadas y por el medio de enfriamiento que permita acercarse lo más posible a dicha velocidad. En la figura 2 se puede observar el enfriamiento de forma gráfica (González, 2022).

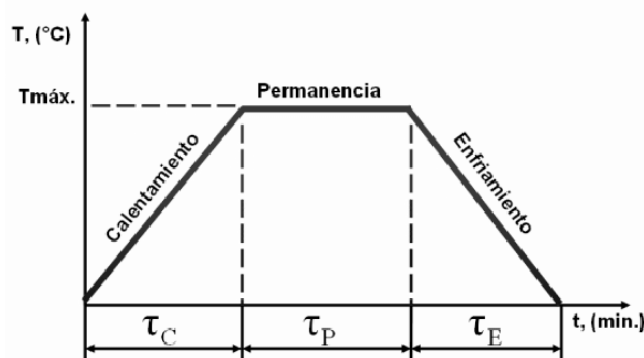


Figura 2: Representación gráfica de las etapas de calentamiento para un temple.

Fuente: https://www.researchgate.net/figure/Figura-2-Eschema-de-representacion-general-de-tratamiento-termico-En-el-caso-del_fig2_308417027

2.2 Acero ASTM A36

Es un acero estructural al carbón (su composición química se muestra en la tabla 1), el cuál es uno de los más utilizados a nivel mundial y es un recurso para la construcción de estructuras metálicas, puentes, torres de energía, torres para comunicación y edificaciones remachadas, atornilladas o soldadas, herrajes eléctricos y señalización (Hernández & Laverde, 2019).

De los aceros al carbono, el ASTM A36 en cuestión, es una de las variedades más comunes debido en parte por su bajo costo. Además, de ofrecer una excelente resistencia y fuerza para un acero al carbono y aleación (Pérez & Capus, 2020).

Tabla 1: Composición química del acero ASTM A36.

Elemento	Contenido (porcentaje)
Carbono, C.	0.25
Manganeso, Mn.	-
Fósforo, P.	0.04
Azufre, S.	0.05
Silicio, Si.	0.40
Cobre, Cu.	0.20

Fuente: Elaboración propia tomada de Hernández & Laverde (2019).

2.3 Acero 1040

El acero 1040 contiene alrededor de un 0.40% de carbono, este material posee propiedades mecánicas sólidas mientras mantiene una buena maquinabilidad. A su vez, su menor contenido de dicho componente químico le confiere una mayor tenacidad, lo que puede traer ventajas a la hora de aplicarse en ambientes donde le exijan al acero una mayor capacidad de deformarse antes de fracturarse, en la tabla 2 se especifica la composición química para este acero (Metalúrgica Daniele, 2024).

Tabla 2: Composición química del acero ASTM A36.

Elemento	Contenido (porcentaje)
Carbono, C.	0.37 – 0.44
Manganeso, Mn.	0.60 – 0.90
Fósforo, P.	0.04
Azufre, S.	0.05

Fuente: Elaboración propia tomada de <https://sisa1.com.mx/wp-content/uploads/Aceros-SISA->

[Tabla-de-Aceros-SAE-AISI-Servicio-Industrial-S.A.-de-C.V..pdf](#)

2.4 Horno mufla

Un horno mufla es aquel que puede generar altas temperaturas y funciona para calentar diversos materiales que pueden ser sometidos a diferentes tratamientos térmicos, un ejemplo de estos se muestra en la figura 3. Estos hornos muflas poseen distintas aplicaciones; como puede ser en la metalúrgica, secado y calcinación de precipitados, aleaciones de metal, templado y otras que necesiten un tratamiento de calor (Monteros, 2016). Cabe destacar que, los hornos muflas permiten un aumento de la temperatura de forma regulada y progresiva gracias a sus mecanismos de control (Bonilla, Gutiérrez, Osorio & Sánchez, 2020).



Figura 3: Ejemplo de horno mufla.

Fuente: <https://estudiantesenlaboratoristaquimico.blogspot.com/2013/12/la-mufla.html>

2.4.1 Principio de funcionamiento de un horno mufla

Estos hornos funcionan gracias a la aplicación de energía térmica suministrada por resistencias eléctricas que están colocadas en las paredes laterales y en la parte posterior de la mufla, las cuales pueden trabajar durante mucho tiempo. Estas resistencias transmiten con gran eficacia el calor obedeciendo la primera ley de la termodinámica, del cuerpo con mayor temperatura al cuerpo de menor temperatura (Monteros, 2016).

Las muflas se componen de ciertos elementos necesarios para su funcionamiento correcto (véase la figura 4):

- Gabinete interno.
- Gabinete externo.
- Contrapuerta.
- Panel de control.

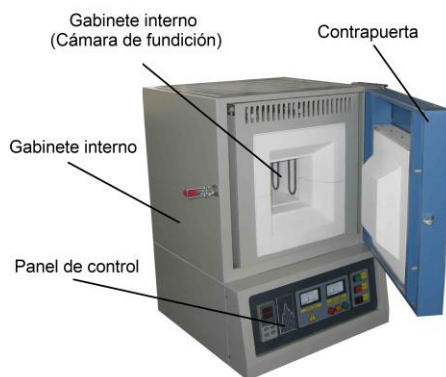


Figura 4: Componentes de un horno mufla.

Fuente: Elaboración propia tomada de

<https://estudiantesenlaboratoristaquimico.blogspot.com/2013/12/la-mufla.html>

2.5 Termopar

El termopar o también llamado termocupla, es un instrumento cuya función es la de medir la temperatura, su composición se basa en dos alambres de diferentes metales unidos por uno de sus extremos (Cali, 2022). Unos ejemplos de estos sensores se muestran en la figura 5.



Figura 5: Ejemplos de termopares.

Fuente: <https://www.tempyco.com/termopares>

Los termopares se basan en una corriente eléctrica del orden de los milivoltios que fluye a través de un circuito continuo conformado por dos alambres metálicos diferentes. Y es gracias a la temperatura presente en esta unión (conocida como “junta caliente”, mostrada en la figura 6), que dicha corriente varía en su señal (Monsalve, 2019).



Figura 6: Diagrama interno de un termopar.

Fuente: <https://www.ingmecafenix.com/automatizacion/sensores/termopar/>

Además Nova (2020), menciona que para medir adecuadamente la variable temperatura con este transductor, es necesario aislar la termo resistencia tanto térmica como eléctricamente. Los materiales más comunes en el aislamiento son:

- Teflón.
- Asbesto.
- Fibra de vidrio.
- Cerámica.

2.5.1 Principio de funcionamiento del termopar

El termopar funciona bajo el efecto *Seebeck*, el cual menciona que, al unir dos metales distintos, estos generan un potencial por lo que, al variar la temperatura en la junta caliente, existe una diferencia de potencial proporcional a la temperatura en el otro extremo del termopar (Cali, 2022). Lo anterior se muestra en la figura 7.

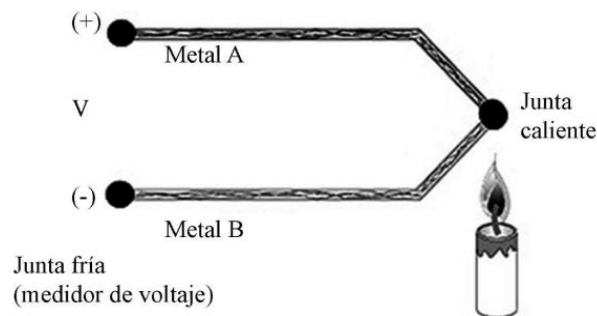


Figura 7: Representación del voltaje en presencia de calor.

Fuente:

https://amyd.quimica.unam.mx/pluginfile.php/9863/mod_resource/content/1/5%20Teoria.pdf

Además del efecto *Seebeck*, otros fenómenos presentes en el funcionamiento del termopar son los efectos *Peltier* y *Thomson*. El efecto *Peltier* se refiere al cambio de la temperatura, ya sea liberación o absorción, producida al circular una corriente eléctrica por una unión entre dos metales o

semiconductores distintos entre sí (Azorín, 2018). El cambio entre la absorción o liberación de la temperatura se representa en la figura 8.

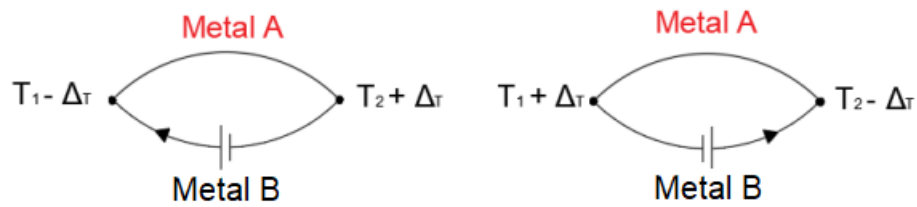


Figura 8: Liberación o absorción de temperatura causada por voltaje.

Fuente: Azorín (2018).

Mientras que el efecto *Thomson* también se manifiesta en la absorción o la liberación de calor cuando existe presencia de una corriente eléctrica circulando por un metal homogéneo en el que se presenta un gradiente de temperatura entre sus extremos, véase figura 9 (Azorín, 2018).

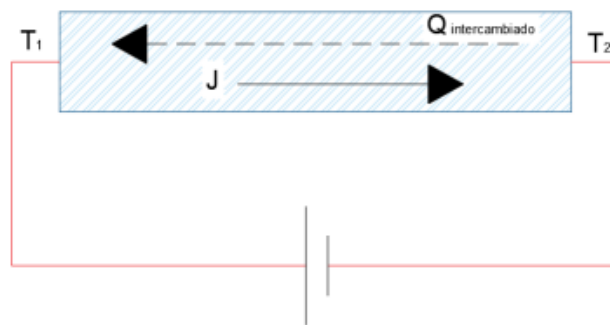


Figura 9: Representación del efecto *Thomson*.

Fuente: Azorín (2018).

2.5.2 Tipos de termopares

Los termopares más utilizados en diferentes aplicaciones son los tipos J, K y T, cada uno de ellos posee un rango de medición de temperatura y un entorno de aplicación distintos (Guerrero & Arias, 2021). Esta información se muestra en la tabla 3.

Según la norma ANSI/ASTM E-230 estandariza los diferentes termopares y codificando su color para el termopar, cables y conectores, véase la tabla para la información de los termopares J, K y T.

Tabla 3: Tabla de especificaciones de los termopares J, K y T.

Código de color ANSI	Combinación de Aleación.		Comentarios Entorno Cable Pelado	Rango de temp. máximo grado termopar	FEM(mV) en todo el rango de temp. máximo
	Positivo	Negativo			
J	HIERRO Fe (Magnético)	CONSTANTÁN CUPRONÍQUEL Cu-Ni	Reductor, vacío, inerte. Uso limitado en entornos oxidantes a altas temperaturas. No se recomienda para temperaturas bajas.	-210 a 1200°C -346 a 2193°F	-8.095 a 69.553
K	CHROMEOMEGA NICROMO Ni-Cr	ALOMEGA NIQUEL ALUMINIO Ni-Al	Oxidante limpio e inerte. Uso limitado en vacío o atmósfera reductora. Amplio rango de temperatura, la calibración más popular.	-270 a 1372 °C -454 a 2501°F	-6.458 a 54.886
T	COBRE Cu	CONSTANTÁN CUPRONÍQUEL Cu-Ni	Ligeramente oxidante. Bueno en condiciones de humedad presente. Aplicaciones a temperaturas bajas y criogénicas	-270 a 400°C -454 a 752°F	-6.258 a 20.872

Fuente: Guerrero & Arias (2021).

2.5.2.1 Termopar tipo J

Los termopares tipo J se conforman por un conductor positivo de Hierro y un conductor negativo de Constantán (aleación compuesta de 55% de Cobre y 45% de Níquel). Presentan una limitación en entornos oxidantes a temperaturas altas y no se recomienda su aplicación en temperaturas bajas. Además, es sensible magnéticamente debido a su composición de Hierro (Guerrero & Arias, 2021).

Cabe destacar que este tipo de termopares son fabricados con la finalidad de utilizarse en vacío, ambientes reductores o en ambientes de gas inerte (Nova, 2020).

2.5.2.2 Termopar tipo K

Los termopares tipo K están compuestos por un elemento positivo de *Chromel* (compuesto de 90% de Níquel y 10% de Cromo) y su elemento negativo de *Alumel* (compuesto de 95% de Níquel, 2% de Manganeseo, 2% de Aluminio y 1% de Silicio) (Guerrero & Arias, 2021).

Este tipo de termopares son utilizados en ambientes oxidantes o de gas inerte, a su vez, se evita el uso en ambientes donde exista la presencia de azufre, porque dicho elemento ocasiona la oxidación del termopar en cuestión. Este tipo suele utilizarse en aplicaciones donde la temperatura será mayor a 550° C (Nova, 2020).

2.5.2.3 Termopar tipo T

El termopar tipo T está compuesto por un elemento positivo de 100% Cobre y su elemento negativo de Constantán, a diferencia de los dos tipos de termopares anteriores, el tipo T presenta una sensibilidad a los campos magnéticos (Guerrero & Arias, 2021).

El termopar tipo T son diseñados para trabajar en temperaturas bajo los 0° C, además de poder utilizarse en ambientes con atmósferas reductoras, oxidantes o de gas inerte. Por último, este tipo de termopares son característicos de soportar la oxidación frente a atmósferas de alta humedad (Nova, 2020).

2.6 MAX6675

El transmisor MAX6675 permite conectar un termopar tipo K, este componente incluye una compensación de junta fría, además de que su propósito es el de convertir la señal analógica entregada por el sensor de temperatura a una señal digital con una excelente resolución de 12 bits, por lo que tiene un rango desde los 0° hasta los 1023° C. El protocolo de comunicación es *SPI*, y por último la resolución de la temperatura entregada por este módulo es de 0.25° C (Domínguez, 2018). En la figura 10, se muestra la representación física de este componente.



Figura 10: Módulo MAX6675.

Fuente: <https://www.geekfactory.mx/producto/max6675-modulo-interfaz-para-termopar-tipo-k/>

2.6.1 Composición del módulo MAX6675

En adición a lo antes mencionado, este módulo es un circuito integrado que está constituido por elementos electrónicos necesarios para amplificar, compensar y convertir una señal análoga de entrada a una señal digital en su salida (Narváez, 2022).

El módulo se constituye por 7 pines (véase figura 11), los cuales son: $T+$, $T-$, V_{cc} , GND , SO , SCK y CS , donde cada uno se conecta a (Gómez & Silva, 2019):

- $T+$: Entrada positiva del termopar K.
- $T-$: Entrada negativa del termopar K.
- V_{cc} : Se conecta al pin positivo de la fuente.
- Gnd : Se conecta al pin negativo de la fuente.
- SO : Salida en serie del módulo a entrada del microcontrolador.
- SCK : Reloj serial.
- CS : *Chip Selected*. Configuración baja, selecciona el módulo y ordena una salida que se sincronice con el reloj.

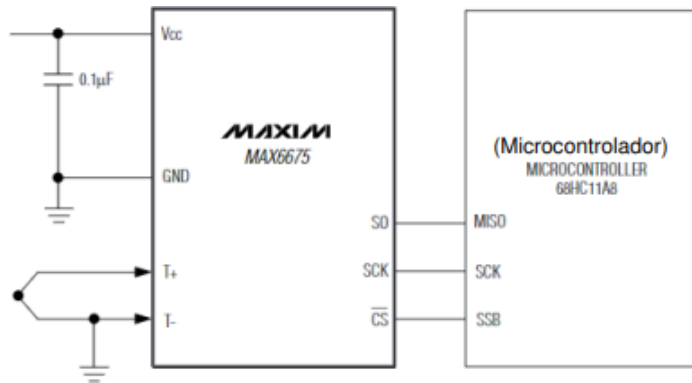


Figura 11: Diagrama de conexión del módulo MAX6675.

Fuente: Narváez (2022).

2.6.2 Funcionamiento del módulo MAX6675

Como bien se ha mencionado, el protocolo de comunicación de este módulo es por medio del protocolo serial *SPI*. Como puede apreciarse en la figura 12, la señal se conforma por 16 bits, de los cuales los primeros 3 son de estado, el bit número uno se utiliza para el estado del módulo, mientras que el bit dos, es para ver el estado del termopar, es decir, si este funciona o no, y el bit tres se utiliza para encender el módulo (Velázquez & Martínez, 2018).

En cuanto a los otros 12 bits restantes, son utilizados para la lectura de la temperatura, como lo mencionó anteriormente.

BIT	DUMMY SIGN BIT	12-BIT TEMPERATURE READING												THERMOCOUPLE INPUT	DEVICE ID	STATE
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	MSB											LSB		0	Three- state

Figura 12: Definición de los bits enviados por el módulo MAX6675.

Fuente: <https://www.geekfactory.mx/producto/max6675-modulo-interfaz-para-termopar-tipo-k/>

2.7 Protocolo SPI

El protocolo *SPI* (*Serial Peripheral Interface*) es un estándar de comunicaciones el cual es usado principalmente para transferir información entre circuitos integrados en equipos electrónicos. La estructura de este protocolo es la de un bus serie síncrono, en otras palabras, emplea una línea de reloj con la finalidad de sincronizar las transmisiones de datos entre dispositivos. Las cuales se llevan a cabo por dos líneas, una que es para la transmisión del sistema maestro hacía el esclavo y la otra para transferir del esclavo al maestro. Dado a que como su nombre lo indica, la transmisión de esta información es en serie, un único bit del contenido del mensaje es enviado en cada ciclo del reloj. Por lo anterior, para finalizar o empezar una transmisión por medio del bus, se debe de incluir una línea de selección de chip con el propósito de utilizarse como identificación del dispositivo que se desea para realizar el intercambio de la información (Grajales & Monsalve, 2019).

La sincronización y la transmisión de datos se realiza gracias a cuatro señales, las cuales son (Grajales & Monsalve, 2019):

- *SCLK* (*Clock*): Es aquel pulso que pauta la sincronización. Con cada uno de los pulsos del reloj, se lee o se realiza el envío de un bit.
- *MOSI* (*Master Output Slave Input*): Es la salida de datos del maestro y la entrada al esclavo. También le llaman *SIMO*.
- *MISO* (*Master Input Slave Output*): Es la salida de datos del esclavo y la entrada del maestro. También le llaman *SOMI*.
- *CS/Select*: Es para seleccionar un esclavo, o para que el maestro le ordene al esclavo activarse. También le llaman *SSTE*.

En la siguiente figura 13, se observa gráficamente la forma de transmisión del protocolo SPI.

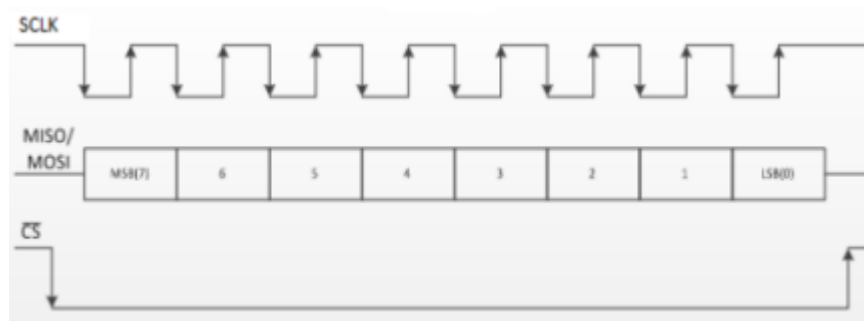


Figura 13: Diagrama de pulsos en la comunicación SPI.

Fuente: Grajales & Monsalve, (2019).

2.8 Microcontroladores

Un microcontrolador es un computador de limitadas prestaciones, el cual se encuentra contenido en un chip de un circuito integrado y tiene como objetivo el de gobernar una sola tarea. Para que el microcontrolador sea capaz de realizar las actividades que se desea es necesario grabar en su memoria de programa una serie de instrucciones que conformen el programa de la aplicación (Grajales & Monsalve, 2019).

De igual forma, Guerreiro & Arias (2021) definen a un microcontrolador como un circuito integrado programable conformado internamente por todos los componentes que forman a un computador. El microprocesador en cuestión al estar programado correctamente es capaz de llevar a cabo sistemáticamente la tarea específica la cual se le fue asignada.

Los mismos autores mencionan que las partes básicas de un microcontrolador son:

- **Memoria ROM:** Es aquella memoria de solo lectura, la cual es utilizada para el almacenamiento de los datos del microcontrolador.
- **Memoria RAM:** Es la memoria de almacenamiento aleatorio, siendo esta la memoria principal que utiliza el microcontrolador, porque es donde se llevan a cabo las operaciones de lectura y escritura del proceso.

- Puertos de entrada y salida: Es la interfaz del microcontrolador, es decir, es donde se conectan aquellos elementos externos que interactúan con el microcontrolador en cuestión.
- Lógica de control: Hace referencia al programa del microcontrolador, ya que define cómo responderá el sistema ante cambios o eventos condicionales.

2.8.1 ESP32

ESPRESSIF (2024), es la empresa creadora del microcontrolador ESP32 entre otros, define a este microcontrolador como una *MCU* (unidad de microcontrolador, por sus siglas en inglés) repleta de funciones que además cuenta con una conectividad *Wi-Fi* y *Bluetooth* integrada para una amplia gama de aplicaciones.

El mismo creador menciona que las características principales del ESP32 son las siguientes:

- Diseño robusto: Este microcontrolador se puede aplicar de forma confiable en ambientes industriales, ya que cuenta con una temperatura de funcionamiento desde los -40° hasta 125° C. Además, gracias a sus circuitos de calibración avanzados, el ESP32 puede erradicar de manera dinámica las imperfecciones del circuito externo y poder adaptarse a cambios en condiciones extremas.
- Consumo ultra bajo de energía: Ya que fue diseñado para dispositivos móviles, dispositivos electrónicos portátiles y aplicaciones de *IoT*, se logra un consumo de energía ultra bajo.
- Alto nivel de integración: Este microcontrolador está integrado con conmutadores de antena incorporados, amplificador de potencia, amplificador de bajo ruido, filtros y módulos de administración de energía, por lo que el ESP32 agrega una versatilidad y funcionalidad invaluables a sus aplicaciones de requisitos mínimos de placa de circuito impreso.

- Chip híbrido *Wi-Fi* y *Bluetooth*: El ESP32 puede funcionar tanto como un sistema completamente independiente, así como también como un esclavo de otro microcontrolador, ya que puede proporcionar su funcionalidad *Wi-Fi* y *Bluetooth* a través de sus interfaces *SPI/SDIO* o *I2C/UART*.

Por último, menciona que cuenta con varios componentes en venta y uno de ellos son los *DevKits* o placas de desarrollo, siendo algunas de ellas las siguientes:

- ESP32-DevKitC.
- ESP32-DevKitM-1.
- ESP32-PICO-KIT.
- ESP32-PICO-V3-ZERO-DevKit (ACK DevKit).
- ESP32-Vaquita-DSPG.

2.8.1.1 ESP32-DevKitC

Es una placa de desarrollo de bajo tamaño y de nivel básico la cual forma parte de la serie ESP32. Además, como se muestra en la figura 14, este módulo tiene un amplio conjunto de periféricos y la distribución de sus pines está optimizada para crear diferentes prototipos sin problemas (ESPRESSIF, 2024).

Ikiss (2020) menciona que el ESP32 incorpora la tecnología *Bluetooth* y *Wi-Fi* que sumado con su pequeño tamaño lo convierte en un microcontrolador ideal para interconectarse con otros dispositivos o fácil de integrar en placas de circuitos impresos.

De igual forma Peris (2020) concuerda con el hecho de que el módulo ESP32 es un dispositivo realmente útil y con una gran variedad de aplicaciones gracias a las antenas integradas de *Bluetooth* y *Wi-Fi*. Porque gracias a dichas antenas puede comunicarse con otros dispositivos de manera inalámbrica, siempre y cuando también cuente con alguna de las dos antenas. Gracias a lo

anterior, este módulo puede brindar ese carácter *IoT* al sistema deseado, siendo además de las mejores opciones para ello debido a su bajo precio y consumo energético.

Las especificaciones del ESP32-DevKitC, son las siguientes (Peris, 2020):

- CPU: Microprocesador de 32-bit Xtensa LX6 de un solo núcleo o doble y puede operar de los 160 hasta 240 MHz y puede rendir hasta 600 DMPIS.
- Memoria: 520 KiB SRAM.
- Conectividad inalámbrica vía *Wi-Fi* y *Bluetooth*.
- 36 pines *GPIO*.
- 4 interfaces periféricas *SPI*.
- 2 interfaces periféricas *I2C*.
- 3 interfaces periféricas *UART*.
- 2x8 bits *DACs*.
- Soporta todas las características de seguridad estándar tales como; *IEEE 802.11*, incluyendo *WFA*, *WPA/WPA2* y *WAPI*.
- 12 bit *SAR ADC* de hasta 18 canales.
- Frecuencia de reloj ajustable desde 80 hasta 240 MHz.
- Controlador host *SD/SDIO/CE-ATA/MMC/eMMC*.
- Controlador esclavo *SDIO/SPI*.
- Incluye sensor de efecto hall y de temperatura interna del chip.
- Preamplificador analógico de ultra baja potencia.

visual, el autocompletado específico para el lenguaje y la comprobación de errores a medida que se escribe el código.

- Automatización de las compilaciones locales: estas son herramientas que ayudan a automatizar aquellas tareas fáciles pero repetitivas como parte de la creación de una compilación local del software para que lo pueda utilizar el desarrollador, como lo es la compilación del código fuente de la computadora en código binario, el empaquetado del código mencionado y la ejecución de pruebas automatizadas.
- Depurador: es un programa que funciona para probar otros softwares y así mostrar la ubicación de un error en el código fuente de forma gráfica.

Por otro lado, Amazon Web Services (2022) menciona que las razones más populares por las que los desarrolladores utilizan IDEs de programación son las siguientes:

- Automatización de la edición de código: dado a que un *IDE* conoce las reglas de estructura de los lenguajes de programación para generar instrucciones, contiene así muchas funciones inteligentes para escribir y editar automáticamente el código.
- Resaltado de sintaxis: como lo mencionó Red Hat anteriormente, el *IDE* tiene la capacidad de cambiar el estilo de la letra de algunas palabras dentro del código fuente, como también de cambiar su color, por lo que se convierten en pistas visuales para errores sintácticos y también para hacer más legible el código.
- Finalización de código inteligente: un *IDE* puede proporcionar sugerencias para terminar una instrucción en el código cuando el desarrollador empieza a escribir.
- Refactorización del soporte: es el proceso de reestructuración del código con el propósito de hacerlo más eficaz y legible sin cambiar su lógica, que el *IDE* puede realizar de forma automática hasta cierto punto.

- **Compilación:** un *IDE* compila o convierte el código de un lenguaje de alto nivel a uno tan simplificado que el sistema operativo donde se ejecutará lo pueda entender (lenguaje máquina).
- **Pruebas:** algunos *IDEs* permiten al desarrollador automatizar pruebas, para comprobar la funcionalidad de su código.
- **Depuración:** una de las características más valiosas del *IDE* es que son capaces de seguir el código línea a línea conforme el código se ejecuta y así revelar errores y poder corregirlos.

2.9.1 IDE de Arduino

La empresa *Arduino* (2019) define al *IDE* de *Arduino* como un software constituido por el conjunto de herramientas que le permiten al programador desarrollar y grabar todo el código necesario para que la placa de desarrollo *Arduino* funcione como el desarrollador lo deseé. Dicho *IDE* permite escribir, depurar, editar y grabar el software llamado “*sketches*” de una forma sencilla, como se percibe en la figura 15.



Figura 15: Ejemplo de la interfaz del *IDE* de *Arduino*.

Fuente: Elaboración propia.

De la misma forma, este *IDE* con ayuda de núcleos de terceros puede utilizarse para programar otras placas de desarrollo y no solo las propias de *Arduino*. Este *software* es una aplicación multiplataforma desarrollada en el lenguaje *Java* y, a su vez, permite los lenguajes de programación *C* y *C++* bajo ciertos criterios de estructuración de códigos (Peris, 2020).

El mismo autor menciona que la estructura de un programa en el *IDE* de *Arduino* es la siguiente:

- Una definición de las librerías de programación que se utilizarán.
- Una función de ajuste o preparación previa al bucle principal.
- Una función de bucle principal la cual se ejecuta de manera infinita.

2.9.1.1 Librería WiFi.h

Esta librería propia de *Arduino* se basa en la funcionalidad del *shield Arduino WiFi* y permite la conexión a redes tanto locales como a redes de internet. Con este recurso se pueden instanciar servidores, clientes, enviar y recibir paquetes *UDP* a través de *WiFi*, también puede conectarse tanto a redes abiertas, así como también, a encriptadas (*WEP*, *WPA*) y, por último, esta librería es capaz de gestionar *DNS* (Arduino, 2015). En la figura 16 se muestra cómo incluirla en un *sketch*.

```
#include <WiFi.h>
```

Figura 16: Declaración de la librería WiFi.h en *Arduino*.

Fuente: Elaboración propia.

2.9.1.2 Librería ESPAsyncWebServer.h

Es una librería *Open Source*, desarrollada por el autor lacamera, que permite la configuración de un servidor *HTTP* y *WebSocket* asíncrono, además cuenta con la característica de poder atender a varios clientes de manera simultánea (Agama, Saa, Baldeón & Bajaña, 2022).

ESPAsyncWebServer es un *WebSocket* desarrollado para las placas de desarrollo *ESP32* y *ESP8266*. Cuya principal funcionalidad es la de manejar distintas instancias asíncronas esperando a recibir peticiones de conexión mediante *HTTP*. Lo anterior da la posibilidad de poder mediante el mismo puerto enviar y recibir peticiones gestionando automáticamente la disponibilidad de recursos de *CPU* y de memoria (Jordà, 2023). Para incluir esta librería a un *sketch*, se debe hacer como en la figura 17.

```
#include <ESPAsyncWebServer.h>
```

Figura 17: Declaración de la librería ESPAsyncWebServer.h en *Arduino*.

Fuente: Elaboración propia.

2.9.1.3 Librería WebSocketsServer.h

Esta librería (figura 18) permite utilizar los *Websockets*, los cuales posibilitan una comunicación bidireccional y simultánea entre los dispositivos conectados entre sí, como puede ser el servidor y el cliente. Dicha conexión comienza con una solicitud realizada por un cliente, la cual el servidor por medio del evento *handshake* acepta la petición del cliente y permite que la comunicación entre ambos siga abierta. Es por lo anterior que, mediante eventos la información es enviada y recibida por ambas partes, por lo que, tanto el servidor como el cliente esperan en todo momento algún evento y actúan de acuerdo con ello (Garrido, Sepúlveda & Vega, 2021).

Así mismo, dentro del mismo trabajo de Garrido, Sepúlveda & Vega (2021), mencionan los principales eventos de comunicación escuchados por el servidor los cuales son:

- Evento *Listening*: Establece la dirección y puerto por el cual se llevará a cabo la conexión.

- Evento *Connection*: Petición de conexión de un cliente.
- Evento *Message*: Indica que se ha recibido un mensaje.
- Evento *Close*: Se presenta cuando uno de los clientes ha cerrado la comunicación.
- Evento *Error*: Se activa cuando un error es detectado en la comunicación.

```
#include <WebSocketsServer.h>
```

Figura 18: Declaración de la librería WebSocketsServer.h en *Arduino*.

Fuente: Elaboración propia.

2.9.1.4 Librería SPIFFS.h

Esta librería realmente es un complemento de *Arduino* la cual funciona para empaquetar la carpeta de datos al nivel del *sketch* en la imagen del sistema de archivos *SPIFFS* y la carga en la memoria *flash* del *ESP32* (Agama, Saa, Baldeón & Bajaña, 2022).

SPIFFS (*Serial Peripheral Interface Flash File System*) es un sistema de archivos basado en *flash*, el cual fue diseñado para ser utilizado específicamente en microcontroladores. Este sistema brinda una forma muy conveniente de almacenar y gestionar diferentes archivos dentro de la memoria *flash* del microcontrolador. Por ejemplo, permite almacenar y gestionar los archivos necesarios para una interfaz *web* (como lo son archivos *html*, *css* y *javascript*) directamente en el microcontrolador o placa de desarrollo (Beltrán, 2024). Para utilizar esta librería se debe de incluir como lo indica la figura 19.

```
#include <SPIFFS.h>
```

Figura 19: Declaración de la librería SPIFFS.h en *Arduino*.

Fuente: Elaboración propia.

2.9.1.5 Librería SPI.h

Esta librería, véase figura 20 para su inclusión, permite la comunicación con dispositivos mediante el protocolo *SPI* (protocolo explicado previamente en el presente trabajo), utilizando la placa de desarrollo como el dispositivo controlador (Arduino, 2024).

```
#include <SPI.h>
```

Figura 20: Declaración de la librería SPI.h en *Arduino*.

Fuente: Elaboración propia.

2.9.1.6 Librería ESPmDNS.h

Esta librería utiliza *mDNS* el cual es un servicio *UDP* de multidifusión que se utiliza para proporcionar un servicio de red local y descubrimiento de *host* (ESPRESSIF, 2023). En la figura 21 se muestra cómo se debe incluirse esta librería.

```
#include <ESPmDNS.h>
```

Figura 21: Declaración de la librería ESPmDNS.h en *Arduino*.

Fuente: Elaboración propia.

2.9.1.7 Librería ArduinoJson.h

Es una librería JSON C++ para *Arduino* y también para *IoT* (Internet de las Cosas) (Benoît, 2014). Para incluir esta librería al *sketch* se debe de seguir la figura 22.

```
#include <ArduinoJson.h>
```

Figura 22: Declaración de la librería ArduinoJson.h en *Arduino*.

Fuente: Elaboración propia.

2.9.1.8 Librería vector

Permite la creación de contenedores de secuencia los cuales apuntan a una matriz externa de estilo C asignada estáticamente. El tamaño máximo de estos contenedores llamados vectores, se fija en el momento de la compilación, pero el tamaño de estos puede cambiar al insertar o extraer elementos. La asignación de memoria estática se utiliza con el objeto de evitar problemas de asignación dinámica en procesadores integrados muy pequeños (Polidoro, 2016). Seguir la figura 23 para utilizar la librería.

```
#include <vector>
```

Figura 23: Declaración de la librería vector en *Arduino*.

Fuente: Elaboración propia.

2.9.2 Visual Studio Code

Es un editor de código fuente, es ligero pero eficaz y se ejecuta en el escritorio. Inicialmente cuenta con soporte integrado para *JavaScript*, *TypeScript* y *Node.js*, además de que cuenta con una gran variedad de extensiones para otros lenguajes de programación y de entornos de ejecución como puede ser: *C++*, *C#*, *Java*, *Python*, etc. (Microsoft, 2024). En la figura 24 se muestra la ventana inicial del *IDE* en cuestión.

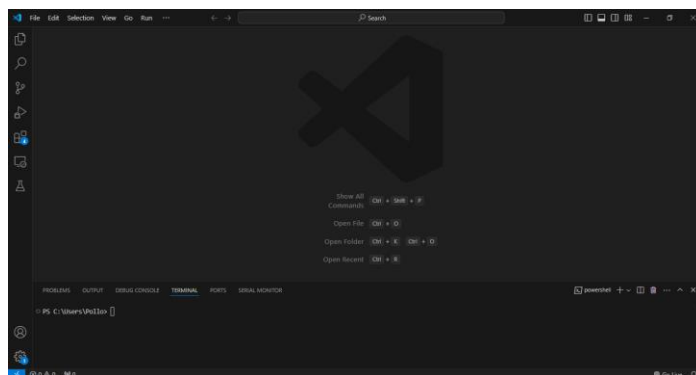


Figura 24: Interfaz gráfica del *IDE* de *Visual Studio Code*.

Fuente: Elaboración propia.

Las principales características que nos menciona la propia marca Visual Studio Code (Visual Studio Code, 2015) son:

- Codificar con *GitHub Copilot*: El desarrollador puede escribir código de una forma mucho más rápida e inteligente gracias a esta herramienta la cual es una IA, que presenta sugerencias automáticas para ayudar a codificar mediante la comprensión del contexto previo y que, además, mediante el chat en línea le permite al desarrollador generar ediciones de forma iterativa y obtener respuestas rápidas y directamente al código fuente de su autoría.
- Programar en cualquier idioma: *VS Code* es compatible y soporta el desarrollo en casi cualquier lenguaje de programación más relevantes. Varios de ellos ya vienen incluidos desde su instalación como lo es *JavaScript*, *TypeScript*, *CSS* y *HTML*, pero se pueden descubrir más extensiones para otros lenguajes en la *VS Code Marketplace*.
- Desarrollar con extensiones: Sin importar el nivel del que el desarrollador tenga, es decir, siendo novato o experto, *VS Code* posee una gran variedad de extensiones que apoyan al programador desde lo visual hasta lo más específico.
- Totalmente personalizable: Este *IDE* brinda al desarrollador la libertad de personalizar la interfaz de usuario con libertad para que así se pueda adaptar a tu estilo. Puede cambiar el color de los temas, sincronizar sus configuraciones y, por último, en *VS Code* se pueden crear perfiles con un conjunto de personalizaciones distintas para cambiar entre ellos con rapidez y de forma sencilla.
- Codificar en cualquier lugar: El desarrollador puede programar en cualquier lugar, es decir, puede estar conectado a la nube, en un repositorio remoto o incluso desde el navegador.

Además de sus características también mencionan las funciones integradas con las que *VS Code* cuenta, las cuales son:

- Terminal integrada: El desarrollador puede usar su *Shell* preferido entre *zsh*, *pwsh* o *git bash*.
- Ejecutar código: Permite ejecutar y depurar el código fuente.
- Control de versiones: Cuenta con un soporte para *git* y muchos otros proveedores de control de fuente.
- Historial local: Posee un historial rastreado automáticamente para no perder los cambios.
- Temas: Cuenta con una gran variedad de estilos predeterminados para adaptarse a la personalidad del desarrollador.
- Accesibilidad: Experiencia optimizada para lectores de pantalla, temas de alto contraste y navegación solo con el teclado.
- Soporte web: El desarrollador mediante cualquier dispositivo con conexión a internet puede acceder a su código fuente.

2.10 HTML

En la actualidad se utiliza *HTML* (figura 25) como lenguaje de desarrollo web. Este término proviene de *HyperText Markup Language* por sus siglas en inglés, y su traducción en español sería lenguaje de marcado de hipertexto. La función principal de este lenguaje es la de indicarle al navegador cómo debe formatear estructuralmente una página *web* al momento de cargarla (Luna, 2019).



Figura 25: Logotipo de *HTML*.

Fuente: <https://www.linkedin.com/pulse/html-css-javascript-rahul-roy>

HTML no es un lenguaje de programación, esto debido a que un lenguaje de programación se caracteriza por tener variables, ciclos, tipos de datos, entre otras propiedades propias. En cambio, *HTML* se basa en el uso de etiquetas, atributos y estilos (Luján, 2016).

Así mismo, en el libro de Luna (2019), menciona que el *HTML* tiene una estructura la cual es la siguiente (en la figura 26 se muestra la estructura con las etiquetas en el *IDE* de *VS Code*):

- `<!DOCTYPE html>`: Esta etiqueta le establece al navegador *web* qué versión de *HTML* se utilizó para desarrollar la página que está a punto de cargar.
- `<html>`: Esta etiqueta menciona cuándo comienza el contenido desarrollado en el lenguaje *HTML*. Dicho contenido se tiene que iniciar con la etiqueta `<html>` y también debe de marcarse su finalización con la otra etiqueta `</html>`, lo anterior establece que todo lo que se encuentre entre esas dos etiquetas en cuestión el navegador debe de interpretarlo como etiquetas *HTML*. Si en dado caso el navegador no pueda interpretar alguna de esas etiquetas dentro del contenido, solo las representará como texto plano.
- `<head>`: Normalmente se utiliza para declarar el contenido principal de la página *HTML*, como lo es; el formato de los caracteres utilizados a lo largo del contenido, el título de la página, referencias a archivos externos como a hojas de estilo CSS o a

archivos *JavaScript*, entre otros recursos para complementar a la página. De igual forma al final de esta parte de la estructura debe de cerrarse con la etiqueta `</head>`.

- `<body>`: Dentro de estas dos etiquetas (también cierra con un `</body>`) se incluye todo el contenido de la parte gráfica de la página *HTML*; títulos, imágenes, textos, audios, videos, entre otros.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9
10 </body>
11 </html>
```

Figura 26: Estructura de un archivo *HTML*.

Fuente: Elaboración propia.

2.11 JavaScript

Es un lenguaje de programación que utilizan los desarrolladores web para lograr que sus páginas *web* sean más interactivas. Lo anterior puede ir desde actualizar la fuente de la letra utilizada hasta mostrar animaciones y mapas interactivos. Este lenguaje es una de las principales tecnologías de la *World Wide Web* (Amazon Web Services, 2022).



Figura 27: Logotipo de *JavaScript*.

Fuente: <https://www.linkedin.com/pulse/html-css-javascript-rahul-roy>

Comúnmente este lenguaje es abreviado como *JS* (figura 27) y es del tipo interpretado. A pesar de ser débilmente tipado y dinámico, también se define como un lenguaje orientado a objetos (Luna, 2019).

Luna (2019) también menciona en su libro las principales 10 ventajas de utilizar *JavaScript*, las cuales son:

1. Es un lenguaje de programación sencillo y poderoso.
2. Cuando se encuentra integrado en los motores *web*, ejecuta rápidamente su sintaxis.
3. Actualmente ya se encuentra integrado en los navegadores web más populares.
4. Es versátil para el desarrollo *web* dinámico y para aplicaciones móviles.
5. Es soportado por todos los dispositivos móviles de la actualidad.
6. Permite desarrollar aplicaciones móviles híbridas.
7. Es multiplataforma (computadoras, celulares, hardware, etc.).
8. Elimina el peso adicional que generaría un *framework*, disminuyendo a su vez el consumo de ancho de banda,
9. Se relaciona estrechamente con *HTML* y *CSS*.
10. Este lenguaje es la base de los *frameworks* más populares de la actualidad.

Por último, en *JavaScript* existen las librerías las cuales son conjuntos de fragmentos de código escritos anteriormente que cualquier desarrollador puede volver a utilizar para realizar algunas funciones deseadas. El código de estas librerías se conecta con el resto del código del proyecto según lo necesite el programador. Algunos usos más comunes de estas librerías son (Amazon Web Services, 2022):

- Visualización de datos.

- Manipulación de *DOM*.
- Formularios.
- Funciones matemáticas y de texto.

2.11.1 Chart.js

Es una librería de gráficos de *JavaScript* para desarrolladores, la cual es la más popular actualmente según las estrellas en *GitHub* (alrededor de sesenta mil) y el número de descargas en *npm* (aproximadamente dos millones cuatrocientos mil descargas semanales). Además, esta librería es de código abierto, posee una licencia *MIT*, es muy permisiva y una gran comunidad activa le brinda soporte constantemente (Char.js, 2024). En la figura 27 se muestra el logotipo de esta librería.



Figura 28: Logotipo de la librería *Chart.js*.

Fuente: <https://www.chartjs.org/>

Chart.js brinda un conjunto de tipos de gráficos, complementos y opciones de personalización. Además de los tipos de gráficos ya incluidos, el desarrollador puede utilizar otros adicionales creados por la comunidad, y a su vez, es posible combinar más de un tipo de gráfico en un gráfico mixto (figura 29). Los tipos son (Chart.js, 2024):

- Gráfico de área.
- Gráfico de barras.

- Gráfico de burbujas.
- Gráficos circulares y de anillos.
- Gráfico de líneas.
- Gráfico de área polar.
- Gráfico de radar.
- Gráfico de dispersión.



Figura 29: Ejemplo de un gráfico mixto hecho con *Chart.js*.

Fuente: <https://www.chartjs.org/>

2.12 CSS

La hoja de estilo en cascada o CSS (por sus siglas en ingles de *Cascading Style Sheets*), véase figura 30, es un lenguaje que se utiliza para definir el estilo de un documento cuya estructura está escrita en *HTML*. La idea principal de utilizar CSS es la de separar la estructura de un documento de su presentación, es decir, definir la información del estilo en un archivo diferente al archivo *HTML* (W3C, 2015).



Figura 30: Logotipo de CSS.

Fuente: <https://www.linkedin.com/pulse/html-css-javascript-rahul-roy>

CSS funciona para determinar el diseño de los documentos electrónicos, lo anterior con ayuda de unas simples instrucciones, las cuales se presentan en código fuente claro, los diferentes componentes de un sitio *web*, como lo son el diseño, color y la tipografía, entre otros, pueden adaptarse y modificarse a placer. Es importante mencionar que, gracias a las hojas de estilo, la estructura semántica y el contenido del documento *web* no se ven afectados (IONOS, 2021).

La implementación de las hojas de estilo en una página web conlleva una serie de ventajas, algunas de ellas son (IONOS, 2021):

- Permite controlar algunas características de diseño de forma centralizada, es decir, una serie de elementos similares, como imágenes, pertenecientes a un mismo documento pueden ser reconocidos y formateados con un solo comando.
- Las instrucciones CSS se guardan en un archivo diferente, lo cual evita malas prácticas y a su vez esta misma hoja puede utilizarse para diferentes documentos *HTML*.
- Se pueden definir o crear animaciones o representaciones gráficas.

2.13 PCB

Printed Circuit Board o *PCB* (Placa de Circuito Impreso, en su traducción al español) es un circuito cuyos componentes y conductores se encuentran contenidos dentro de una estructura mecánica. Las principales funciones conductoras de una *PCB* consisten en trazos de cobre, terminales y conductores planos. A su vez, la estructura general de la placa es chapada y cubierta por una capa de soldadura no conductora acompañada de un serigrafiado de los componentes, como puede observarse en la figura 31 (Peterson, 2023).

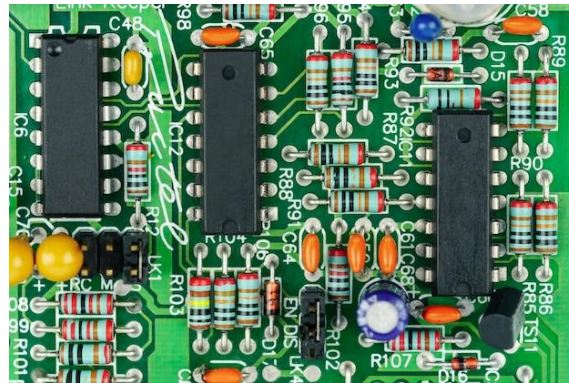


Figura 31: Placa de circuito impreso.

Fuente: https://www.freepik.es/fotos-premium/primer-plano-placa-circuito-impreso-componentes-como-resistencias-circuitos-integrados_21869142.htm

Así mismo, Peterson (2023) menciona que las placas de circuito impreso están construidas por capas que alternan entre cobre y material no conductor alternándose entre sí. En el proceso de fabricación de estas placas, se graban las capas del material conductor dejando caminos o trazos entre los componentes para lograr la interconexión de estos.

En la actualidad, los dos tipos de materiales más comunes y que son muy útiles para emplearlos como tarjetas de impresión o trazados de los circuitos impresos son la fibra fenólica (también conocida coloquialmente como “baquelita”) y la fibra de vidrio (Morales & Solis, 2018).

Estos materiales cuentan ya sea con una o dos caras cubiertas con una delgada capa de cobre, véase figura 32, en la cual se graba el circuito. Ambas ofrecen características físicas

adecuadas para poder realizar dicho proceso de manufactura, como lo es la resistencia al calor, alta rigidez al montar los componentes electrónicos y facilidad de corte para su diseño en cualquier dimensión y geometría (Morales & Solis, 2018).



Figura 32: Baquelita virgen de una sola cara.

Fuente: <https://ja-bots.com/producto/baquelita-virgen-10x10/>

2.13.1 Proteus

Es un *software* enfocado en la ejecución de proyectos de construcción de equipos electrónicos en todas sus etapas, las cuales son; diseño del esquema electrónico, programación del *software* de la *PCB* a construir, simulación del proyecto, depuración de errores, documentación y construcción (Morales & Solis, 2018).

El *software Proteus PCB* combina dos módulos de la aplicación en general, los cuales son el módulo de captura esquemática y el módulo de diseño de *PCB* para así, con el producto proporcionar una herramienta asequible, potente y fácil de usar para la creación y diseño de una *PCB* como puede observarse en la figura 33 (Proteus, 2018).

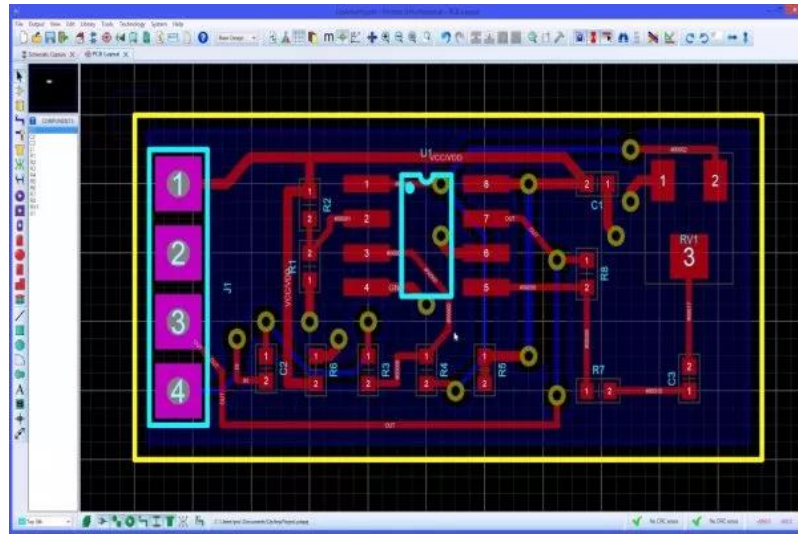


Figura 33: Interfaz gráfica del software de diseño de *PCB* de *Proteus*.

Fuente: <https://www.labcenter.com/pcb/>

2.13.2 FlatCAM

En la actualidad el fresado *PCB*, también llamado grabado mecánico, es el proceso de manufactura para la creación de placas de circuito impreso más rápido, limpio, de alto rendimiento y económico (FlatCAM, 2014).

FlatCAM es un *software* que permite llevar diseños de circuitos a una fresadora *CNC*. Esta aplicación puede abrir archivos del tipo *Gerber*, *Excellon* o *G-code*, además de poder editarlo o, por lo contrario, crearlo desde cero, para al final obtener un archivo *G-code*. Una de las tareas para las que *FlatCAM* es perfecto es la del enrutamiento de aislamiento (figura 34). Este programa es de código abierto, está escrito en el lenguaje de programación *Python* y funciona sin problemas en la mayoría de los sistemas operativos, algunas de sus características con las siguientes (FlatCAM, 2014):

- Espectador: Se puede visualizar archivos *Gerber*, *Drill* y *G-Code*. Por lo que el usuario puede saber cómo se verá la *PCB* y cómo será maquinada.

- Enrutamiento de aislamiento: Permite la creación de trayectorias de herramientas de grabado para así devastar alrededor de los trayectos de las conexiones entre componentes. Se puede establecer varias pasadas para espacios amplios, además de limpiar áreas particulares de cobre deseadas.
- Dos caras: Se puede llevar a cabo la creación de placas de circuito de dos caras.
- Características de potencia: Mediante una consola *TCL* se proporciona una increíble flexibilidad para que los usuarios puedan automatizar e implementar sus propias funciones.

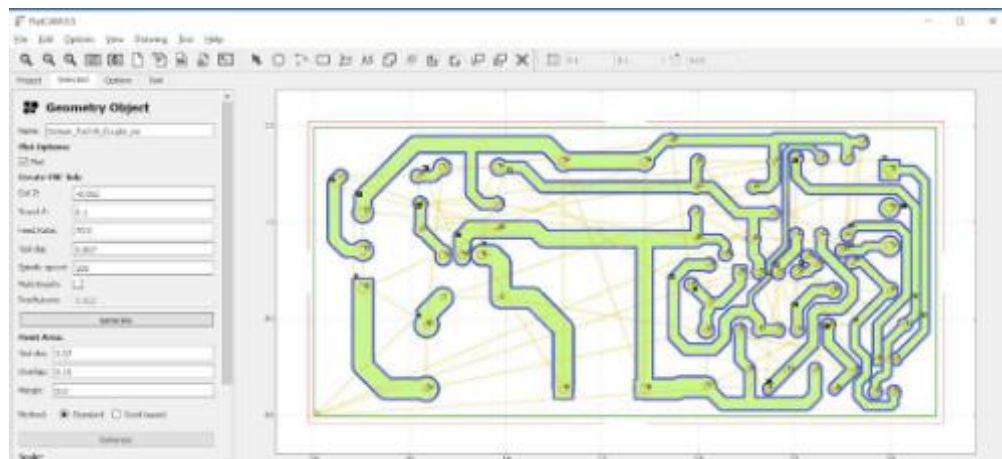


Figura 34: Interfaz gráfica del software *FlatCAM*.

Fuente: https://hobbycnc.com/pc_board_isolation_routing/pc-board-isolation-routing-flatcam/

2.14 CNC

Las máquinas *CNC* (Control Numérico Computarizado, por sus siglas en inglés), son dispositivos de fresado automatizados, los cuales fabrican componentes industriales sin la necesidad de la ayuda o la operación continua de la mano humana. Estas herramientas utilizan instrucciones codificadas enviadas a través de una computadora, lo que permite el fabricar piezas con gran precisión y rapidez (Dassault Systèmes, 2022).

Una pieza clave de estas máquinas es el controlador *CNC*, el cual es un componente cuyo propósito es el de interpretar los códigos del programa, otorgándole así a la máquina la capacidad de accionarse en un orden secuencial dado. Para lograr lo anterior, trabaja con una serie de motores y componentes de accionamiento para el desplazamiento de los ejes de la máquina (Salvador, 2020).

El control numérico computarizado es, como ya se mencionó, un sistema que permite el control en todo momento de la posición de un elemento físico, dando origen así al mecanizado *CNC*. Este es un proceso que se ha convertido en una tecnología indispensable para las industrias en la actualidad, ya que trae consigo un aumento en la productividad y reduce los costes de fabricación. Consiste en diseñar un programa con una serie de órdenes las cuales determinen la posición y rotación de la herramienta siempre de forma secuencial. Gracias a ello, se puede controlar la posición y velocidad de desplazamiento exacta (Boyfre, 2023).

Debido a que esta tecnología es muy popular y en ocasiones necesaria en la actualidad, existen una gran variedad de máquinas *CNC*, algunas de estas son (Autodesk, 2019):

- Enrutadores *CNC*: Son utilizados normalmente en la carpintería y pueden variar su tamaño y estilo, es decir pueden ser desde tres grados de libertad hasta los cinco grados y ser del tamaño de un escritorio hasta una habitación completa. En la figura 35 se muestra un ejemplo.



Figura 35: Enrutador *CNC* modelo KT-1325.

Fuente: <https://sinosupply.com.mx/producto/cnc-router-kt-1325>

- Cortadoras de plasma *CNC*: Se utilizan para cortar metales conductores mediante la aplicación de un chorro de plasma a altas temperaturas como se aprecia en la figura 36.



Figura 36: Cortador *CNC* de plasma.

Fuente: <https://www.hypertherm.com/es/landing-pages/choosing-a-cnc-plasma-table/>

- Cortadoras láser *CNC*: Permiten el cortar una gran variedad de materiales y espesores que pueden llegar hasta el metal, lo anterior depende de la potencia del láser. En la figura 37 se muestra un ejemplo de estas máquinas.



Figura 37: Cortadora láser *CNC* de CO₂.

Fuente: <https://sideco.com.mx/>

- Tornos *CNC*: Rotan a altas velocidades la pieza de trabajo y mediante diversas herramientas de corte se crean piezas con un perfil de revolución como se ve en la figura 38.

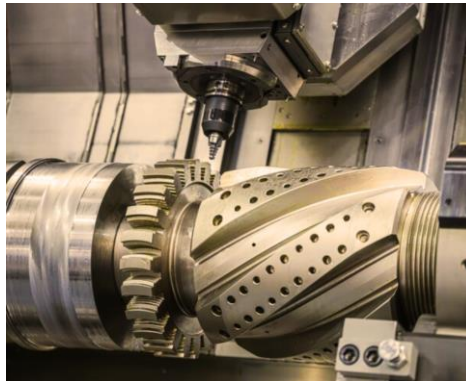


Figura 38: Torno *CNC*.

Fuente: <https://www.zintilon.com/es/blog/cnc-lathe-machines-from-simple-turning-to-complex-parts/>

- Fresadoras *CNC*: Son altamente versátiles e incluyen una gran variedad de configuraciones de su fresado que pueden ir desde los tres hasta los cinco grados de libertad. Un ejemplo de este tipo de máquinas se muestra en la figura 39.



Figura 39: Fresadora *CNC*.

Fuente: <https://sideco.com.mx/>

- Fresado y torneado *CNC*: Combinan las dos máquinas anteriores en un solo centro de mecanizado.

2.14.1 Candle

Es un *software* de código abierto el cual es utilizado en las máquinas herramientas *CNC*. Brinda la posibilidad de procesar archivos de código G, así como también tener un monitoreo visual a tiempo real (López, 2022).

Candle se utiliza para controlar *CNC*, pero solo aquellas que se basen en *GRBL* (Denvi, 2023). *GRBL* es un *software* gratuito y de código abierto, el cual es de alto rendimiento y de bajo consumo para el control de *CNC* (Siatras, 2023).

Las principales funciones de *Candle*, que pueden verse en la figura 40, son (Denvi, 2023):

- Permite el control de una máquina *CNC* basada en *GRBL*, lo anterior mediante comandos de consola, botones y/o teclado numérico.
- Monitorear del estado en tiempo real de la máquina *CNC*.
- Carga, edición, guardado y envío de archivos de código G a la máquina *CNC*.
- Brinda la posibilidad de visualizar los archivos de código G.

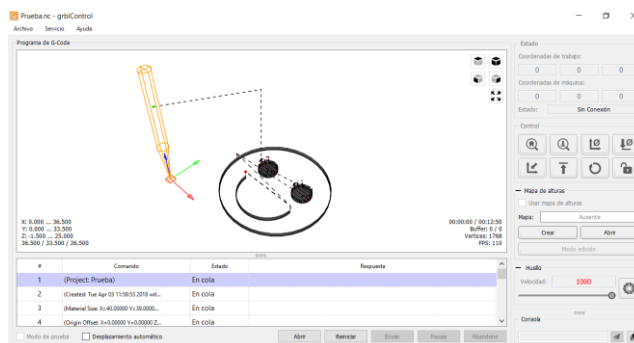


Figura 40: Interfaz gráfica de *Candle* con un proyecto de ejemplo.

Fuente: Elaboración propia.

2.15 Impresoras 3D

La impresión 3D es un proceso de fabricación por adición y automatizado, en este se involucra una impresora 3D la cual crea un modelo físico basado en un objeto o modelo 3D. Hay diferentes tipos de impresoras 3D, sin embargo, todas se basan en el mismo principio el cual es; crear un objeto agregando capas encima o debajo de la capa previa existente correspondiente a la sección transversal del modelo 3D (Prusa, 2021).

Para poder llevar a cabo la impresión en 3D es fundamental generar el código G, para ello mediante un *software* en específico, el modelo digital en 3D se corta en cientos de capas finas que posteriormente se convierten en código G, para que así la impresora sea capaz de saber con precisión cuándo y dónde depositar el material (Dassault Systèmes, 2022).

2.15.1 Impresoras FDM o FFF

FDM son las siglas en inglés para Modelado por Deposición Fundida (es una marca registrada por Stratasys) y *FFF* es el acrónimo en inglés para Fabricación por Filamento Fundido (el cual es de código abierto). Este tipo de impresoras 3D son todas aquellas que calientan o derriten polímeros que son empujados a través de una boquilla de impresión hasta la plataforma base de construcción del modelo físico. El material para imprimir se proporciona en bobinas de 1.75 mm hasta 2.85 mm de grosor. La diferencia entre los tipos de impresoras de este principio de funcionamiento radica en cómo se mueve la boquilla o extrusora y la plataforma base de impresión. Los tipos de impresoras 3D cuyo principio de funcionamiento es por *FFF* son (Prusa, 2021):

- Impresoras cartesianas: Se caracterizan por tener al menos tres motores que se encargan de mover las partes de la impresora a lo largo del sistema de coordenadas cartesianas (véase figura 41): Y para adelante y atrás, X para izquierda y derecha y Z para arriba y abajo.

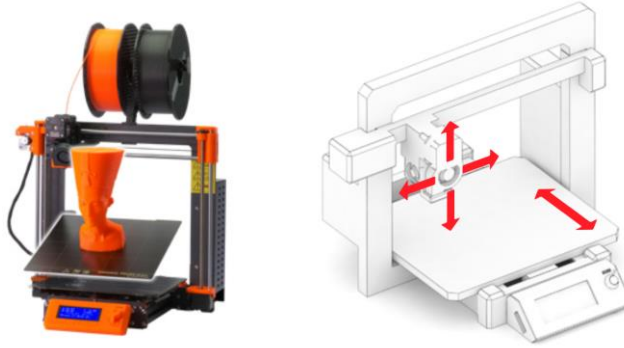


Figura 41: Impresora *FFF* del tipo cartesiana.

Fuente: Prusa (2021).

- Impresoras *CoreXY*: Estas impresoras utilizan el mismo sistema de coordenadas que las impresoras cartesianas, determinadas por dos motores y correas de distribución los ejes Y y X, la altura de impresión se determina con Z. La diferencia de este tipo de impresoras es que el movimiento X o Y dependen el uno del otro, gracias a la relación entre sus correas, véase en la figura 42. En este caso la base de impresión subirá o bajará para encontrarse con la extrusora.

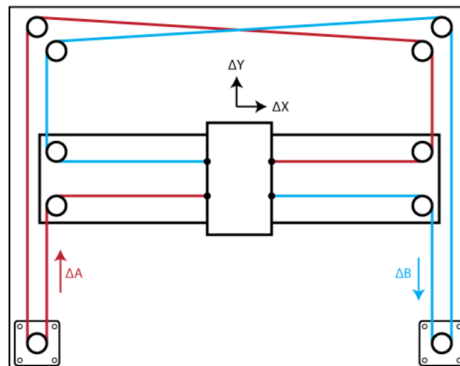


Figura 42: Impresora *FFF* del tipo *CoreXY*.

Fuente: Prusa (2021).

- Impresoras *delta*: La superficie o base de impresión es fija y tres motores son los encargados de posicionar el extrusor a lo largo de las coordenadas cartesianas (figura

43). Dichos motores controlan tres pilares y estos mismos están conectados a la extrusora mediante brazos. El nombre de este tipo de impresoras proviene de la forma triangular que forman los brazos que sostienen a la boquilla.

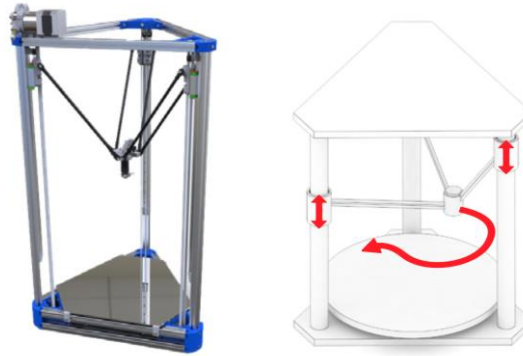


Figura 43: Impresora *FFF* del tipo *delta*.

Fuente: Prusa (2021).

- Impresoras polares: Esta impresora, a diferencia de las anteriores, utiliza el sistema de coordenadas polares. En este caso la base de impresión además de moverse hacia los lados también posee la característica de rotar. Por lo que, la extrusora solo se mueve hacia arriba y abajo. En la figura 44 se muestra un ejemplo de este tipo de impresoras.



Figura 44: Impresora *FFF* del tipo polar.

Fuente: Prusa (2021).

2.16 UltiMaker Cura

Es un *software* de impresión 3D gratuito, fácil de usar y de código abierto. Brinda un ajuste fino del modelo digital 3D proporcionado con más de cuatrocientas configuraciones para así lograr obtener los mejores resultados de corte e impresión, en la figura 45 se muestra su interfaz gráfica. Con esta aplicación cualquier diseño *CAD* generado con *SolidWorks*, *Autodesk Inventor*, entre otros, en un formato *STL*, *OBJ*, *X3D*, *3MF*, *BMP*, *GIF*, *JPG*, *PNG*, puede ser convertido a código G gracias al flujo de trabajo integrado con el que cuenta Cura (UltiMaker, 2019).

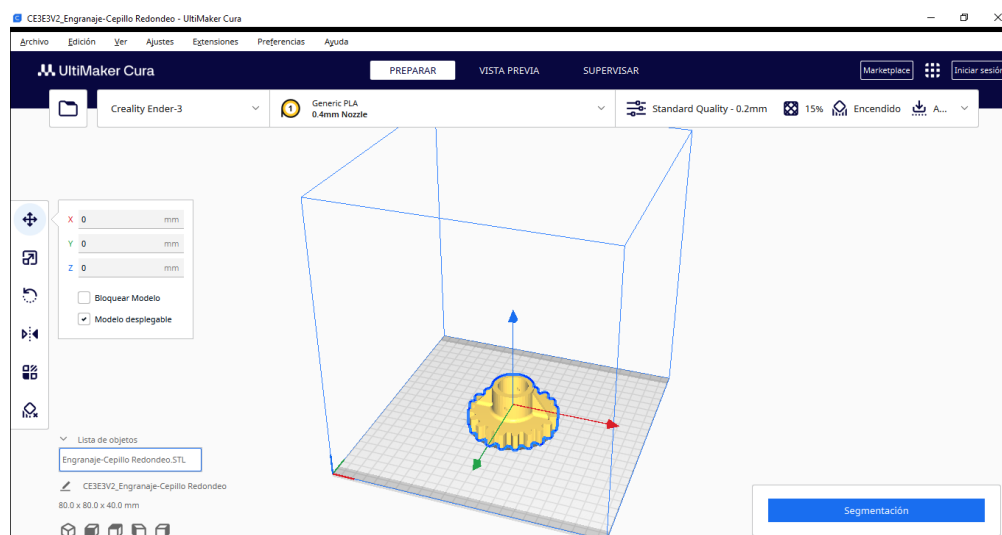


Figura 45: Interfaz gráfica de *UltiMaker Cura*.

Fuente: Elaboración propia.

2.17 SolidWorks

Es un *software* de diseño *CAD* 3D (Diseño Asistido por Computadora) usado para modelado y creación de piezas y ensamblajes en 3D y planos en 2D. Esta aplicación ofrece un gran abanico de soluciones para cubrir los diferentes aspectos implicados en el proceso del desarrollo de un producto (véase su interfaz en la figura 46). Este *CAD* ofrece la posibilidad de crear, diseñar, simular, fabricar, publicar y gestionar los datos del proceso de diseño (SolidBi, 2018).

SolidWorks como se mencionó anteriormente se basa en el diseño 3D, pero para ello, la creación de piezas se basa en croquis 2D, los cuales son dibujos, que mediante las diferentes herramientas ofrecidas por el software es capaz de darles un espesor y generar un volumen obteniendo como resultado un modelo 3D, y al unir diferentes modelos es posible la creación de un ensamblaje 3D, así como también la generación de dibujos 2D a partir de tanto los modelos y ensamblajes 3D (Dassault Systèmes, 2018).

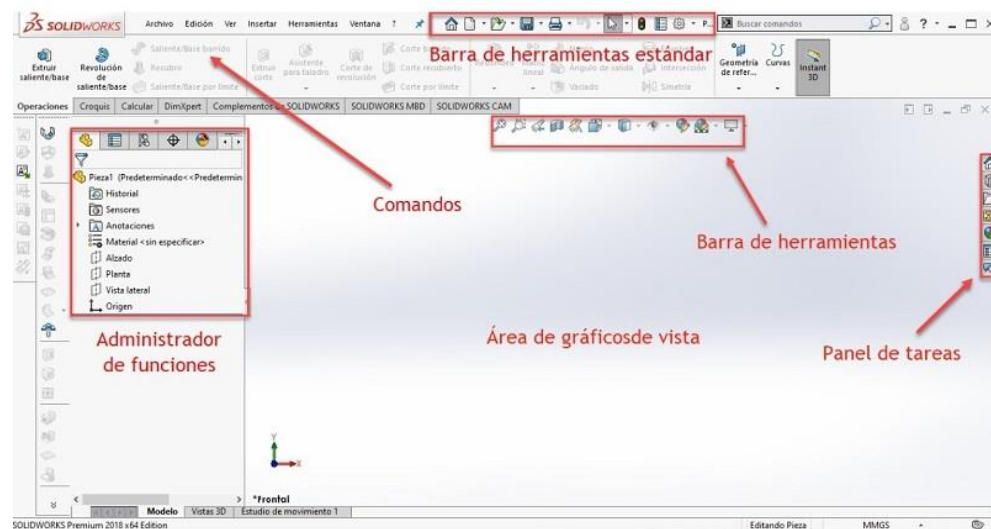


Figura 46: Interfaz gráfica de *SolidWorks*.

Fuente: <https://www.3dcadportal.com/articulos/disenio-3d-y-analisis-estructural-de-un-patin-electrico>

3 PROCEDIMIENTO

3.1 Elaboración de la página web

La elaboración de la página web para el proyecto se desarrollará en el *IDE* de *Visual Studio Code* y también con la ayuda de la extensión *Live Server* (anexo 8.1), la cual será de ayuda para la visualización de la estructura de la página web en desarrollo, ya que permite crear un servidor local para el proyecto web.

La estructura del proyecto web se dividirá en cuatro archivos (uno del tipo *HTML*, uno del tipo *CSS* y dos del tipo *JavaScript*) dentro de una carpeta llamada “data”, esto porque será necesario más adelante con la herramienta de la librería para *Arduino SPIFFS*.

3.1.1 Desarrollo del archivo *HTML*

En este archivo colocamos las etiquetas necesarias para lograr el diseño deseado, partiendo desde la estructura básica para este tipo de archivos (véase figura 26). Para visualizar mejor el código diríjase al anexo 8.2.

Dentro de las etiquetas *html* (figura 47) este contenido el tipo de la codificación de caracteres del documento siendo este de tipo *UTF-8*, lo que nos permite utilizar caracteres especiales y acentos. También se incluye el título de la página web que aparecerá en la pestaña del navegador utilizado. Mediante el atributo *viewport* definimos la escala inicial y el ancho de la página, para la escala se define de 1.0 lo cual se traduce en un 100%, lo cual significa que el *zoom* inicial de la web será de dicho porcentaje, mientras que para el ancho se establece que debe coincidir con la anchura del dispositivo en el que se muestre la página. Por último, se enlazan tres archivos (uno del tipo *CSS* y dos del tipo *JavaScript*) los cuales se encargarán del diseño y funcionalidad de la web.

```

<head>
  <meta charset="utf-8">
  <title>TermoparK</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="estilo.css">
  <script type="text/javascript" src="chart.min.js"></script>
  <script type="text/javascript" src="graficas.js"></script>
</head>

```

Figura 47: Contenido de la etiqueta *head*.

Fuente: Elaboración propia.

Las etiquetas *body* están compuestas por el contenido del sitio web (figura 48), en este caso se le agregó a la propiedad de anchura la característica de *fit-content*, la cual se encargará de adaptar el ancho del contenido de acuerdo con las demás etiquetas.

Las principales etiquetas son las que definen a una tabla y a un divisor. Se definen tres propiedades para esa tabla; el ancho, la posición y el borde. Para el primero se define que será del 100%, es decir, será del mismo ancho que el *body*, para el segundo que la posición del contenido dentro de la tabla estará en el centro y, por último, se establece que los bordes tendrán el tamaño de un píxel, esto para que se vea gráficamente las limitaciones espaciales de la gráfica.

La composición de la tabla se basa en dos renglones, el primero para colocar el título (por ese motivo se le coloca el identificador “título”) el cual es “WebServer ESP32 para TermoparK” con la etiqueta especial *h1* para definirlo como un título de primer nivel, además se le atribuye la propiedad *colspan* con la cantidad de cinco, esto para que concuerde con el número de columnas para el siguiente renglón. Como se acaba de mencionar, este segundo renglón se conforma por cinco columnas dentro de las cuales se encuentran diferentes componentes. La primera columna (con un tamaño de 20% para definir que el tamaño es el mismo para todas las columnas) contiene a un *combobox* con la leyenda “Cantidad de termopares” (de color gris claro o #d8d8d8 en hexadecimal) y con el identificador de “cantidad” el cual será utilizado para establecer el número de termopares que se utilizarán (solo se puede elegir entre uno o dos). La segunda columna se conforma por la leyenda “Tiempo de muestreo (segundos)” (el color es el mismo que el de la leyenda anterior) y la caja de texto con el identificador “muestreo” en el cual se encuentra una función que se encarga de validar que solo

se puedan ingresar números (también la propiedad *numeric* ayuda a que la caja de texto solo acepte caracteres numéricos). En cuanto a las siguientes tres columnas las conforman tres botones, uno para cada columna, los cuales pertenecen a la clase “boton”, y sus identificadores al igual que sus nombres son; empezar, reinicio y finalizar, respectivamente, siendo este último de un color rojo.

Por último, para el divisor con identificador “contenido” solo contiene a otro divisor con la propiedad *text-align: center* para centrar el título de segundo nivel “Selecciona la cantidad de termopares” (dado por la etiqueta *h2*).

```

<body style="width: fit-content;">
  <table width="100%" align="center" border="1">
    <tr id="titulo">
      <td colspan="5">
        <h1>WebServer ESP32 para TermoparK</h1>
      </td>
    </tr>
    <tr>
      <td style="width: 20%;">
        <div>
          <p style="color: #8d8d8d; margin: 1%;>
            Cantidad de termopares
          </p>
          <select id="cantidad" style="margin: 1%;>
            <option value="0">Selecione...</option>
            <option value="1">1</option>
            <option value="2">2</option>
          </select>
        </div>
      </td>
      <td style="width: 20%;">
        <div>
          <p style="color: #8d8d8d; margin: 1%;>
            Tiempo de muestreo (segundos)
          </p>
          <input id="muestreo" type="text" inputmode="numeric" pattern="d+"
            oninput="this.value = this.value.replace(/[^0-9]/g, '')"/>
        </div>
      </td>
      <td style="width: 20%;">
        <button class="boton" id="empezar">Empezar</button>
      </td>
      <td style="width: 20%;">
        <button class="boton" id="reinicio">Reinicio</button>
      </td>
      <td style="width: 20%;">
        <button class="boton" style="background-color: #c00000;" id="finalizar">Finalizar</button>
      </td>
    </tr>
  </table>
  <div id="contenido">
    <div style="text-align: center;">
      <h2>Selecciona la cantidad de termopares</h2>
    </div>
  </div>
</body>

```

Figura 48: Contenido de las etiquetas *body*.

Fuente: Elaboración propia.

3.1.2 Elaboración del archivo CSS

En este documento se colocarán propiedades y características que sirvan de apoyo al archivo *HTML* para lograr que la página *web* se vea mejor (el contenido de este archivo puede visualizarse por completo en el anexo 8.3).

Para la etiqueta *body* se define un color de gris oscuro para su fondo (código hexadecimal #424242), lo anterior puede verse en la figura 49.

```
body {
  background-color: #424242;
}
```

Figura 49: Color de fondo para la etiqueta *body* en el archivo CSS.

Fuente: Elaboración propia.

Como se observa en el figura 50, los títulos del primer al tercer nivel se establece que el texto estará alineado al centro, su color de fondo gris oscuro (con código hexadecimal #474747), el color blanco usado para las letras (su código hexadecimal es #ffffff), la forma en la que se mostrarán los elementos será según la propiedad *inline-block*, lo que permite que los títulos se puedan colocar en línea con otros elementos, por lo que el ancho de los títulos se ajustará al texto y no por toda la línea, por último, se define un espacio de 1% alrededor de los títulos para que el color de fondo de los mismos se expanda un poco.

```
h1, h2, h3 {
  text-align: center;
  background-color: #474747;
  color: #ffffff;
  display: inline-block;
  padding: 1%;
}
```

Figura 50: Propiedades de las etiquetas de los títulos.

Fuente: Elaboración propia.

La etiqueta *td* que se utiliza para establecer una columna de una tabla solo se declara una propiedad, siendo esta que los elementos estén alineados al centro, véase la figura 51.

```
td {
  text-align: center;
}
```

Figura 51: Propiedades de la etiqueta *td* para las columnas de las tablas.

Fuente: Elaboración propia.

Para los *comboboxes* (se usa la etiqueta *select* como se ve en la figura 52) se define un color gris claro para su fondo (con un código hexadecimal #aeaeae) y se ajusta a un tamaño mediano la letra.

```
select {  
  background-color: #aeaeae;  
  font-size: medium;  
}
```

Figura 52: Características de estilo para los selectores o *comboboxes*.

Fuente: Elaboración propia.

Las entradas de texto o cajas de texto con etiqueta *input* (figura 53), tendrán un color gris claro para su fondo al igual que los selectores, así mismo, su tamaño de letra también será de uno mediano, el ancho de la caja de texto está dada por setenta píxeles y el texto dentro de esta entrada estará centrado.

```
input {  
  background-color: #aeaeae;  
  font-size: medium;  
  width: 70px;  
  text-align: center;  
}
```

Figura 53: Propiedades para las cajas o entradas de texto.

Fuente: Elaboración propia.

Para la clase de los botones se definen múltiples cosas, como se puede observar en la figura 54. Primero con *.boton* se establece un estilo base, al cual se le eliminan los bordes, el color de texto es negro, el tamaño de la fuente será mediano, gracias a la propiedad *padding* se agrandan los botones añadiendo un relleno interno del tamaño de los píxeles mostrados en la figura 54 o en el anexo 8.3, con la propiedad *pointer* ocasiona que el cursor cambie a una mano cuando se posiciona arriba de un botón, se redondean las esquinas de los botones con un tamaño de cinco píxeles, y el color de fondo para los botones será de un gris claro (código #aeaeae en hexadecimal). Gracias a la instrucción en CSS *.boton:hover* se puede declarar que cuando el cursor este sobre un botón, el color

del fondo del mismo se cambie a un gris oscuro (código #7e7e7e en hexadecimal). Y por último, mediante la instrucción `.boton:active` se define que al presionar el botón el tamaño del mismo se reduzca un 10%.

```
.boton {
  border: none;
  color: black;
  font-size: medium;
  padding: 14px 28px;
  cursor: pointer;
  border-radius: 5px;
  background-color: #aeaeae;
}
.boton:hover {
  background-color: #7e7e7e;
}
.boton:active {
  transform: scale(0.9);
}
```

Figura 54: Propiedades para la clase boton.

Fuente: Elaboración propia.

Para terminar, en la figura 55 se muestran las propiedades para las etiquetas *canvas*, las cuales son que el ancho de este ocupe el 100% del espacio disponible y mediante con la propiedad *height* al dejarla en automático se asegura la relación de aspecto de los *canvas*, también se establece que el ancho máximo de los mismos no deberá de superar el 100% y, por último, al colocar la propiedad *margin* en automático, se centran los *canvas* en sus contenedores.

```
canvas {
  width: 100%;
  height: auto;
  max-width: 100%;
  margin: auto;
}
```

Figura 55: Propiedades para los canvas.

Fuente: Elaboración propia.

3.1.3 Creación del archivo *JavaScript*

Antes de desarrollar este archivo es importante descargar la librería de *Chart.js* como se indica en el anexo 8.4. Y otro aspecto a mencionar es que el código explicado a continuación se encuentra en su totalidad en el anexo 8.5.

La función principal de este documento será un evento, el cual se encargará de detectar cuando el documento se halla cargado en su totalidad antes de ejecutar su código (la función se visualiza en la figura 56).

```
> document.addEventListener('DOMContentLoaded', function(){ ...  
});
```

Figura 56: Evento para que el *script* se ejecute una vez cargado el *DOM*.

Fuente: Elaboración propia.

Dentro de la función lo primero que se declaran son las variables que se utilizarán a lo largo del código (ver figura 57). El primer conjunto de variables que son declaradas, son del tipo arreglo y serán aquellas referentes al tiempo, temperatura, diferencial de temperatura y el tiempo del diferencial de temperatura, para cada uno de los dos termopares. El siguiente conjunto son las variables que se requieren para crear las gráficas mostradas en el sitio *web*. Inmediatamente después se declara una bandera la cual al ser de tipo booleano será de ayuda para identificar la cantidad de termopares indicada por el usuario. Por último, se declara una variable la cual será usada para establecer la conexión *WebSocket* con el *ESP32* utilizando el *localhost* y el puerto número 81.

```
//Arreglos de tiempo, temperatura y diferenciales de los termopares  
let tTmk1, tempTmk1, dFTmk1, tDFTmk1 = [];  
let tTmk2, tempTmk2, dFTmk2, tDFTmk2 = [];  
//Variables para las gráficas  
let cnfgTmk1, cnfgDFTmk1 = '';  
let cnfgTmk2, cnfgDFTmk2 = '';  
let cnfgCompTemp, cnfgCompDf = '';  
let chartTmk1, chartDFTmk1, chartTmk2, chartDFTmk2, chartCompTemp, chartCompDf = '';  
//Bandera para identificar el uso de 1 termopar o de 2  
let bFlag = false;  
//Variable para establecer la conexión WebSocket con el ESP32 mediante el puerto 81  
let socketConnection = new WebSocket('ws://' + location.hostname + ':81/');
```

Figura 57: Variables utilizadas a lo largo del *BackEnd* de la página *web*.

Fuente: Elaboración propia.

Después de las anteriores declaraciones de variables, se crean más, sin embargo, estas son del tipo constante (figura 58), debido a que se usan para seleccionar elementos de la página *web* mediante sus identificadores, guardando así el *combobox* con el identificador *cantidad*, la caja de texto con identificador *muestreo* y un divisor declarador en el archivo *html* con el identificador *contenido*. De igual forma se seleccionan los tres botones con los identificadores *empezar*, *reinicio* y *finalizar*.

```
//Variables de configuración de la página
const cantidad = document.getElementById('cantidad');
const muestreo = document.getElementById('muestreo');
const contenido = document.getElementById('contenido');

//Variables para trabajar con los botones
const botonEmpezar = document.getElementById("empezar");
const botonReinicio = document.getElementById("reinicio");
const botonFinalizar = document.getElementById("finalizar");
```

Figura 58: Constantes de elementos de la página *web*.

Fuente: Elaboración propia.

Un objetivo del código es poder cambiar la interfaz *web* en función de la cantidad de termopares a usar, para ello es necesaria la creación de un evento enfocado en el cambio del valor del *combobox*, así que para lograrlo es necesario usar la constante *cantidad*, ya que apunta a ese elemento *html*, y añadirle un evento del tipo *change* para así cuando el usuario seleccione un valor, este se guarde en la variable *cantidadSeleccionada* e inmediatamente después ejecute la función *actualizarContenido* mandando como parámetro la variable mencionada. El fragmento de código puede verse en la figura 59.

```
//Evento para cambiar la página según cuántos termopares se usen
cantidad.addEventListener('change', function() {
  let cantidadSeleccionada = cantidad.value;
  actualizarContenido(cantidadSeleccionada);
});
```

Figura 59: Evento de cambio para el *combobox* de la cantidad de termopares.

Fuente: Elaboración propia.

La función *acualizarContenido*, se compone de una estructura *if* de tres casos, en el primero se compara si el parámetro enviado es igual a dos y de ser así la bandera toma un estado de verdadero y posteriormente se modifica el divisor con identificador *contenido* reemplazando lo

existente con una tabla que ocupa el ancho del documento *html*, se alinea al centro su contenido y sus bordes son del tamaño de un píxel. La tabla consta de seis filas y una sola columna y en cada fila se crea una gráfica y un botón de descarga para la misma, el orden de las gráficas es; la gráfica de la temperatura del primer termopar seguida del gráfico de su diferencial de temperatura, posteriormente se replica lo mismo, pero en función del segundo termopar, y al final se grafican las comparaciones entre las temperaturas de los dos termopares, y por separado, la comparación de sus diferenciales de temperatura. Después de graficar se ejecuta la función `asignarEventosDeDescarga`. Sin embargo, si el parámetro no es dos, en un segundo caso se compara para saber si es igual a uno, de serlo, la bandera toma un estado falso y al igual que el caso anterior, se crea una tabla con las mismas propiedades, pero siendo solamente de dos filas y una columna, en donde solo se grafican la temperatura y el diferencial de temperatura de un solo termopar, así mismo también se ejecuta la función `asignarEventosDeDescarga`. Por último, si por alguna razón ninguno de los dos casos se cumple, la estructura se encarga de darle un estado falso a la bandera y se modifica el contenido regresándolo a su estado inicial. Por lo extenso de esta función favor de dirigirse al anexo 8.5 en caso de necesitar visualizar el código.

La función `asignarEventosDeDescarga`, la cual es mencionada anteriormente, tiene el objeto de, como su nombre lo indica, asignar un evento a cada uno de los botones de descarga de las gráficas, para así descargarla en formato de imagen (*PNG*). Para lograr lo anterior, como puede verse en la figura 60, primeramente, se seleccionan todos los elementos *HTML* que correspondan a la clase botón y, además, poseen un identificador, una vez agrupados todos los botones, se itera sobre los mismos gracias a la función *forEach*, con la finalidad de asignar un evento de pulsación a cada botón. Dentro de ese evento, lo primero que se realizará es asignar el valor del identificador del botón en cuestión a la variable constante con nombre `canvasId`, después de eso, a otra variable constante de nombre `canvas` se guardará el elemento *HTML* con el mismo identificador almacenado en la variable `canvasId`. Posteriormente, se creará un nuevo elemento *HTML* del tipo de enlace `<a>` para poder activar la descarga y se almacenará en otra variable del tipo constante con el nombre `link`, luego al atributo *href* (atributo necesario para que el navegador trate a lo almacenado como contenido descargable) de la variable mencionada, se guardará la gráfica almacenada en la variable `canvas`

después de convertirla en una imagen de formato *PNG*, a continuación gracias al atributo *download* se le puede asignar un nombre a la imagen que será descargada, el cual estará dado por el identificador de la gráfica y seguido por la extensión *.png*. Por último, gracias al método *click()* de la misma variable, se simulará que se pulsa el enlace creado para dar inicio a la imagen generada.

```
//Función para agregar el evento de descarga de cada gráfica a su botón correspondiente
function asignarEventosDeDescarga() {
  document.querySelectorAll('.boton[id]').forEach(button => {
    button.addEventListener('click', function() {
      const canvasId = this.getAttribute('id');
      const canvas = document.getElementById(canvasId);
      const link = document.createElement('a');
      link.href = canvas.toDataURL('image/png');
      link.download = canvasId + '.png';
      link.click();
    });
  });
}
```

Figura 60: Función para asignar el evento de descarga a los botones de las gráficas.

Fuente: Elaboración propia.

La interfaz web se compone de tres botones para controlar el estudio de la temperatura, el primero siendo el botón empezar. A dicho botón se le asigna un evento para cuando sea pulsado, véase figura 61, que a su vez se compone de una estructura *if-else*, la cual compara que la cantidad de termopares no sea igual a cero, como también su tiempo entre mediciones, de no ser así se mostrará una alerta en el navegador indicando que se debe de proveer la información requerida para poder empezar con el estudio. Por otro lado, si la comparación de la estructura es verdadera, se ejecutarán dos funciones; *deshabilitarControl* y *deshabilitarCofig*, las cuales serán explicadas más adelante. Después de eso, se crea un *JSON* para enviarlo al *ESP32*, los datos dentro de este objeto es el indicador con nombre *configWeb*, la cantidad de termopares, el tiempo de muestreo, el valor booleano de falso para la bandera *botonEstados* (para indicar si los botones están habilitados) y, por último, el valor booleano de verdadero de la bandera *bEmpezar* (para indicar si se debe de medir la temperatura). Finalmente, gracias a la variable *socketConnection* y su método *send*, es por lo que se puede enviar al *JSON* anterior al microcontrolador. Inmediatamente después de enviar el objeto se ejecuta la función *obtenerGraficas*.

```

//Evento del botón de empezar para iniciar el estudio de temperatura
botonEmpezar.addEventListener('click', function() {
    if (cantidad.value !== '0' && muestreo.value !== 0) {
        deshabilitarControl();
        deshabilitarCofig();
        //Mandar los datos de configuración al esp32
        const jsonData = JSON.stringify({
            indicador: "configWeb",
            cantidad: cantidad.value,
            muestreo: muestreo.value,
            botonEstados: false,
            bEmpezar: true
        });
        socketConnection.send(jsonData);
        obtenerGraficas();
    } else {
        alert("Llena la información requerida para poder empezar");
    }
});

```

Figura 61: Evento clic para el botón de empezar.

Fuente: Elaboración propia.

Las funciones deshabilitarControl y deshabilitarCofig, tienen la misma finalidad, deshabilitar elementos *HTML*, mientras que la primera bloquea y oscurece los botones de empezar y reiniciar, pero permite el uso del botón finalizar, la segunda bloquea y oscurece el selector de termopares y la caja de texto para el tiempo de muestreo. El fragmento de código anterior puede observarse en la figura 62.

```

//Función para deshabilitar los botones de control de la página web //Función para deshabilitar los controles de configuración de la página web
function deshabilitarControl() {
    botonEmpezar.disabled = true;
    botonEmpezar.style.backgroundColor = "#606060";

    botonReinicio.disabled = true;
    botonReinicio.style.backgroundColor = "#606060";

    botonFinalizar.disabled = false;
    botonFinalizar.style.backgroundColor = "rgb(205, 0, 0)";
}
function deshabilitarCofig() {
    cantidad.disabled = true;
    cantidad.style.backgroundColor = "#606060";
    cantidad.style.color = "#d8d8d8";

    muestreo.disabled = true;
    muestreo.style.backgroundColor = "#606060";
    muestreo.style.color = "#d8d8d8";
}

```

Figura 62: Funciones para deshabilitar algunos botones y la información del ensayo.

Fuente: Elaboración propia.

El segundo botón con el nombre de reinicio también posee un evento para cuando es pulsado (figura 63). Dentro de tal evento, lo primero que se realizará es la limpieza de los arreglos de tiempo, temperatura y diferenciales, así como también se limpian las variables necesarias para la realización de la gráfica. Posteriormente, al botón empezar se le actualiza su texto con “Empezar”. Luego se ejecutan dos funciones; destruirGraficos y habilitarConfig. Después, se reiniciarán los campos de

entrada para la cantidad de termopares y el tiempo de muestreo, al igual que la bandera bFlag ya que tomará un estado falso. Al igual que el botón de empezar, el botón reinicio crea un *JSON* para enviarlo al microcontrolador, en este caso el nombre del indicador es de resetWeb, la cantidad de termopares y el tiempo de muestreo es cero y el estado para los botones es de verdadero. Dicho objeto se envía al *ESP32* de la misma forma explicada anteriormente. Por último, se ejecuta la función actualizarContenido con la cantidad de termopares actual para que la página *web* regrese a su estado inicial.

```
//Evento del botón de reinicio para volver al estado inicial de la página web
botonReinicio.addEventListener('click', function() {
    tTmk1, tempTmk1, dFTmk1, tDfTmk1 = [];
    tTmk2, tempTmk2, dFTmk2, tDfTmk2 = [];
    cnfgTmk1, cnfgDfTmk1, cnfgTmk2, cnfgDfTmk2 = '';
    cnfgCompTemp, cnfgCompDf = '';

    botonEmpezar.innerHTML = "Empezar";

    destruirGraficos();
    habilitarConfig();
    cantidad.value = "0";
    muestreo.value = 0;
    bFlag = false;
    //Reiniciar los datos de configuración al esp32
    const jsonData = JSON.stringify({
        indicador: "resetWeb",
        cantidad: "0",
        muestreo: "0",
        botonEstados: true
    });
    socketConnection.send(jsonData);
    actualizarContenido(cantidad.value);
});
```

Figura 63: Evento clic para el botón de reiniciar.

Fuente: Elaboración propia.

La función destruirGraficos usada previamente, como su nombre lo indica, se utilizará para eliminar los gráficos creados en el sitio *web*, se encargará de verificar si existe algún gráfico y de ser así, utilizará el método *destroy* de la librería *Chart.js* para borrarlo. La programación de lo anterior se muestra en la figura 64.

```
//Función para destruir las gráficas que existan
function destruirGraficos() {
  if (chartTmk1) chartTmk1.destroy();
  if (chartTmk2) chartTmk2.destroy();
  if (chartDfTmk1) chartDfTmk1.destroy();
  if (chartDfTmk2) chartDfTmk2.destroy();
  if (chartCompTemp) chartCompTemp.destroy();
  if (chartCompDf) chartCompDf.destroy();
}
```

Figura 64: Función para la destrucción de las gráficas.

Fuente: Elaboración propia.

En cuanto a la función `habilitarConfig` (figura 65), tiene como finalidad la de habilitar el uso de los elementos *HTML* para la selección de la cantidad de termopares y el ingreso del tiempo de muestreo, así como también de regresar el color inicial de estos.

```
//Función para habilitar los controles de configuración de la página web
function habilitarConfig() {
  cantidad.disabled = false;
  cantidad.style.backgroundColor = "#aeaeae";
  cantidad.style.color = "black";

  muestreo.disabled = false;
  muestreo.style.backgroundColor = "#aeaeae";
  muestreo.style.color = "black";
}
```

Figura 65: Función para habilitar el ingreso de datos.

Fuente: Elaboración propia.

De igual manera que los dos botones anteriores, este con nombre de finalizar posee un evento para detectar cuándo es pulsado, ver figura 66. Al cumplirse dicho suceso, la función ejecutará una función de confirmación, la cual consta de preguntar al usuario mediante una ventana emergente del navegador si desea finalizar el estudio de temperatura o no, el resultado booleano lo guardará en una bandera llamada `bFin`, para después ser evaluada en una estructura *if-else*. En caso de que el usuario cancele la finalización del estudio, el programa solo imprimirá el mensaje de que no se finalizó en la consola de las herramientas del programador. Por otro lado, si el usuario decide aceptar la finalización, lo primero en suceder será que se cambiará el nombre del botón de empezar a retomar. Después se ejecutará la función `habilitarControl` e inmediatamente se creará un *JSON* para mandarlo al *ESP32* de igual forma que se realizó con los botones anteriores, el contenido de este consta del

indicador con nombre finalizar, el valor booleano de falso para la bandera bEmpezar y el valor booleano verdadero para el estado de los botones.

```
//Evento del botón Finalizar para detener el estudio de temperatura
botonFinalizar.addEventListener('click', function() {
    let bFin = confirm("¿Estás seguro de querer finalizar?");
    if (bFin) {
        botonEmpezar.innerHTML = "Retomar";
        habilitarControl();
        const jsonData = JSON.stringify({
            indicador: "finalizar",
            bEmpezar: false,
            botonEstados: true
        });
        socketConnection.send(jsonData);
    } else {
        console.log("No se finalizó");
    }
});
```

Figura 66: Evento clic para el botón finalizar.

Fuente: Elaboración propia.

La función habilitarControl tiene como función principal, como lo dice su nombre, habilitar los botones de control de la página web y regresarlos al color original de estos, en la figura 67 se muestra cómo se habilita y cambia el color de los tres botones.

```
//Función para habilitar los botones de control de la página web
function habilitarControl() {
    botonEmpezar.disabled = false;
    botonEmpezar.style.backgroundColor = "#aeaeae";

    botonReinicio.disabled = false;
    botonReinicio.style.backgroundColor = "#aeaeae";

    botonFinalizar.disabled = true;
    botonFinalizar.style.backgroundColor = "rgb(155, 0, 0)";
}
```

Figura 67: Función para habilitar los botones de control del sitio web.

Fuente: Elaboración propia.

Siguiendo con el proceso de la programación, en el evento del botón empezar se ejecuta la función obtenerGráficas, cuyo propósito es el de crear todas las gráficas necesarias para mostrar los datos de las mediciones tanto de un solo termopar como de los dos. En la figura 68 se puede observar que para la creación de las gráficas se llama a la función crearGrafico donde los parámetros utilizados son los arreglos de temperatura y tiempo del termopar número uno, en caso de que la bandera bFlag

tenga un valor booleano verdadero, el proceso anterior se repite para el segundo termopar, así como también, se usa la función `crearGraficoComp` para crear las gráficas donde se comparan los resultados de ambos termopares. Por último, se ejecuta la función `graficar`.

```
//Función para crear todas las gráficas
function obtenerGráficas() {
    cnfgTmk1 = crearGrafico(tTmk1, tempTmk1);
    cnfgDfTmk1 = crearGrafico(tDfTmk1, dfTmk1);

    if (bFlag) {
        cnfgTmk2 = crearGrafico(tTmk2, tempTmk2);
        cnfgDfTmk2 = crearGrafico(tDfTmk2, dfTmk2);

        cnfgCompTemp = crearGraficoComp(tTmk1, tempTmk1, tempTmk2);
        cnfgCompDf = crearGraficoComp(tDfTmk1, dfTmk1, dfTmk2);
    }
    graficar();
}
```

Figura 68: Función para obtener las gráficas antes de mostrarlas en la web.

Fuente: Elaboración propia.

La función `crearGrafico` (anexo 8.5) utiliza la librería *Chart.js* para crear gráficas de línea para representar la relación entre la temperatura y el tiempo del termopar. Los parámetros que se reciben son dos, un arreglo de tiempo que serán los valores para el eje X y otro arreglo con los valores de temperatura para representarlos en el eje Y. Lo primero que realizará la función es crear un objeto constante llamado `datos` para definir lo que se graficará, la clave *labels* con el valor de tiempo, significa que este arreglo será usado como etiquetas en el eje X, mientras que la clave *datasets* se compone de un conjunto de datos; *data* para indicar que los valores de temperatura serán usados de etiquetas a lo largo del eje Y, y *borderColor* para definir un color azul para la línea del gráfico. Lo segundo en definirse es otro objeto constante llamado `config`, dentro del cual se establecen las características para las gráficas. Por la clave *type* con un valor *line* se declara que el gráfico será de tipo línea y la variable `datos` creada anteriormente se asigna a los valores de la gráfica mediante la llave *data*. En cuanto a las opciones, se usa la llave *scales* para configurar ambos ejes, para X se añade el título de “Tiempo (t)” y con color gris para su cuadrícula, mientras que para Y se usa el título “Temperatura (°C)” y el color de su cuadrícula esta dado por una estructura *if-else*, en la que para los valores diferentes a cero será de un color gris, mientras que para el valor cero será de un color blanco semitransparente. Por último, la función retorna el objeto `config` para su uso.

La siguiente función con nombre `crearGraficoComp` (anexo 8.5), es similar a la anteriormente explicada, sin embargo, los parámetros recibidos son tres arreglos, el de tiempo y los dos de temperaturas de los termopares. Para los conjuntos de datos de las temperaturas se nombran según su termopar gracias a la llave *label* (ya sea “Termopar 1” o “Termopar 2”), para los valores de cada conjunto se utiliza su arreglo, para ambos se desactivan los rellenos que se crean debajo de las líneas al momento de graficar (así no interfiere una línea con la otra) gracias a la llave *fill* y también se agrega una pequeña curva en las gráficas para suavizar los giros, y por último, la línea para el primer termopar será de un color rojo, mientras que para el segundo termopar será de un color verde. El tercer conjunto de datos solo se compone de el arreglo tiempo para usar sus valores como etiquetas en el eje X, y de contener los dos conjuntos de datos previamente creados, en un solo arreglo para poder graficarlas en el mismo gráfico. En el objeto `config` se establece lo mismo que en la función anterior, con una sola diferencia en la cuadrícula del eje Y, ya que consta de dos estructuras *if-else*, una para el grosor de las líneas, el cual se mantendrá igual para todos los valores excepto cuando es cero, ya que este se vuelve más grueso para resaltarlo. Y la segunda estructura es para definir el color, de igual forma, para cualquier valor la cuadrícula será del mismo color gris, menos para cuando es cero, en dicho caso será de un tono azul. Al final, la función devuelve el objeto `config` para su uso.

Hasta este momento el sitio web configurado se puede observar mediante el uso de la extensión *Live Server*, el cual se muestra en la figura 69.



Figura 69: Sitio *web* en la espera del ingreso de la información.

Fuente: Elaboración propia.

En cuanto a la parte final del código *JavaScript*, se define un evento, con nombre `socketConnection.onmessage` (anexo 8.5), que se encarga de manejar exclusivamente los mensajes recibidos desde el microcontrolador *ESP32*. Esta función se ejecuta automáticamente cada vez que

recibe un mensaje y lo primero que realizará es transformar el objeto JSON recibido como parámetro a un objeto *JavaScript* y almacenarlo en la variable *data* para así poder acceder su información y propiedades. Una propiedad de dicho objeto es el campo llamado *indicador*, el cual determinará el evento que se manejará, lo anterior en conjunto con una estructura *switch*. Esta estructura examina el valor del indicador y ejecuta uno de los tres posibles casos. El primero dado por “inicio”, se encarga de establecer en los campos de cantidad y muestreo (el *combobox* y la caja de texto) los valores enviados por el microcontrolador, siendo la cantidad de termopares usados y el tiempo de muestreo, respectivamente, después se encarga de ejecutar la función *actualizarContenido* con la cantidad de termopares que se están utilizando para el estudio de la temperatura y, por último, si el valor booleano del parámetro *botonEstados* es falso, ejecuta la función *deshabilitarControl*, en caso contrario, ejecuta *habilitarControl*. El segundo caso dado por “medición”, se encargará de actualizar las variables globales para los valores del termopar, siendo estos, el de tiempo, temperatura, tiempo del diferencial de temperatura y el diferencial de temperatura, después ejecuta las funciones *deshabilitarCofig*, *destruirGraficos* y *obtenerGraficas*, y al final, cambia el texto del botón *empezar* por “Retomar”. Por último, el tercer caso dado por “medicion2”, al igual que el caso anterior, actualizará los valores de las variables globales, pero ahora para los dos termopares, sin embargo, ejecutará las mismas funciones y también cambiará el texto del botón *empezar*.

3.2 Programación del ESP32

La programación del microcontrolador en cuestión se llevará a cabo mediante el *IDE* de programación de *Arduino*, para ello, es necesario realizar la instalación del soporte de *Arduino* para *ESP32* (anexo 8.6).

Ya que se cuente con el soporte instalado en la opción Herramientas > Placa, se seleccionará el microcontrolador “ESP32 Dev Module” para continuar con la programación.

Cabe aclarar que la ubicación de este *sketch* de *Arduino* se debe de ubicar al mismo nivel que la carpeta “data” creada para la página web, esto con la finalidad de que la herramienta *SPIFFS* funcione correctamente.

Lo primero en declararse en la programación, son las librerías que se usarán a lo largo del proyecto, siendo estas las mostradas en la figura 70. Estas librerías son instaladas en el anexo 8.7.

```
//Librerías utilizadas
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <SPIFFS.h>
#include <WebSocketsServer.h>
#include <SPI.h>
#include <ESPmDNS.h>
#include <ArduinoJson.h>
#include <vector>
```

Figura 70: Figuras usadas en la programación del ESP32.

Fuente: Elaboración propia.

Lo siguiente en el código es la definición de los pines *Chip Select* (CS) del módulo MAX6675 para los dos termopares que se utilizarán, para ello se declaran de la forma mostrada en la figura 71.

```
// Definir pines de Chip Select para cada MAX6675
#define MAX6675_CS1 5
#define MAX6675_CS2 4
```

Figura 71: Definición de pines CS para módulos MAX6675.

Fuente: Elaboración propia.

Después se declaran los *LEDs* indicadores con los que contará el proyecto, para ello se establecen tres variables (figura 72), uno para cada *LED*, de tipo constante y entero de ocho *bits*.

```
//Definir pin de los indicadores
const uint8_t ledPinEncendido = 2;
const uint8_t ledPinCensando = 16;
const uint8_t ledPinError = 17;
```

Figura 72: Declaración de los tres *LEDs* indicadores.

Fuente: Elaboración propia.

Inmediatamente después se crean las variables que se usarán en la lógica del programa, mostradas en la figura 73, dentro de las cuales está una bandera de tipo booleano para iniciar o finalizar con el ciclo de mediciones de temperatura, todas las variables para llenar los arreglos de información para ambos termopares, una bandera de tipo entero de ocho *bits* para saber si existen mediciones de temperatura previas o no, una segunda bandera de tipo booleano para indicar si están bloqueados o no los controles de la página web y dos últimas variables del tipo entero de ocho y dieciséis *bits*, una para indicar saber la cantidad de termopares establecidos en la web y la segunda para el tiempo de muestreo del estudio.

```
//Variable para iniciar el ciclo del estudio
bool bEmpezar = false;

//Variables para los arreglos
uint8_t tiempo = 0;
uint8_t tiempoAnterior = 0;
float tempAnterior = 0;
float tempAnterior2 = 0;
uint8_t deltaT = 0;
float deltaTemp = 0;
float deltaTemp2 = 0;
float derivada = 0;
float derivada2 = 0;

//Variable para saber si hay o no mediciones
uint8_t bMediciones = 0;

//Variables de configuración de la página Web
bool botonEstados = true;
uint8_t cantidadTermopares = 0;
uint16_t tiempoMuestreo = 0;
```

Figura 73: Declaración de variables que se usarán en la lógica del programa.

Fuente: Elaboración propia.

En adición a lo anterior, también son declarados los vectores que serán utilizados más adelante para el almacenamiento de los datos de las mediciones de el o los termopares, todos estos vectores son definidos como se muestra en la figura 74.

```
//Vectores para almacenar temperaturas, tiempo y diferenciales
std::vector<float> tTmk1;
std::vector<float> tempTmk1;
std::vector<float> dfTmk1;
std::vector<float> tTmk2;
std::vector<float> tempTmk2;
std::vector<float> dfTmk2;
```

Figura 74: Vectores para el almacenamiento de los datos de las mediciones.

Fuente: Elaboración propia.

Por último, con respecto a las declaraciones del programa, se declaran los servidores que se utilizarán para gestionar las solicitudes *web*, así como también las conexiones *WebSockets*. Para la gestión de las solicitudes se usará un servidor asíncrono y mediante el puerto 80 y para la conexión *WebSocket* será mediante el puerto 81, la declaración anterior se muestra en la figura 75.

```
//Declarar los servidores y sus puertos
AsyncWebServer server(80);
WebSocketsServer websockets(81);
```

Figura 75: Declaración de los servidores y sus puertos.

Fuente: Elaboración propia.

Lo siguiente en el código es la función *setup*, la cual se muestra en el anexo 8.8 debido a ser tan extensa. Dentro de esta función, se configura un puerto serial el cual puede ser de ayuda para *debuggear* en caso de un error y su velocidad será de 115200 baudios. Después, los tres *LEDs* que se usarán se declaran como salidas y se apagan (en caso de estar encendidos). Luego, se inicializa el bus *SPI* ya que este será usado para la comunicación con los módulos *MAX6675* de los termopares, es por ello por lo que los pines *CS* de estos se declaran como salidas y enviamos una salida alta para ambos pines para deseleccionarlos hasta que la medición sea inicializada. Posteriormente, se configura el servidor *WiFi* del *ESP32* para que éste funcione como un punto de acceso creando una red *WiFi* con el nombre “MedicionTermoparK” y la contraseña “LaboratorioTermo2024”. Inmediatamente después, se configura el *DNS* mediante el servicio *Multicast DNS*, lo cual permitirá ingresar al microcontrolador mediante el nombre de dominio “termopar.local” y no mediante su *IP* (esto seguirá siendo posible, pero para practicidad se recomienda el uso del *DNS*), en caso de que sufra un error al configurar el *DNS*, el sistema entrará en un bucle infinito indicando esto al usuario mediante el encendido del *LED* por nombre *ledPinError* y escribiendo un mensaje de error en la consola serial, en caso de que la configuración es exitosa se apaga el *LED* del error y se muestra un mensaje del estado en la consola serial.

Lo siguiente en inicializarse dentro de esta función es el sistema de archivos *SPiffs*, en el cual, si existe un error al montar dicho sistema, se muestra un mensaje de error en la consola serial y

se enciende el *LED* por nombre `ledPinError`. Siguiendo con este sistema de archivos, se configuran las diferentes rutas en el servidor del microcontrolador para montar archivos específicos almacenados en la memoria *flash* del *ESP32* mediante la solicitud HTTP de tipo *GET*, las rutas mencionadas son las siguientes:

- `/`: Monta como página principal el archivo con nombre `index.html`.
- `/estilo.css`: Monta como hoja de estilos el archivo de nombre `estilo.css`.
- `/char.min.js`: Monta la librería JS de nombre `chart.min.js`.
- `/graficas.js`: Monta el archivo JS de la programación *BackEnd* que tiene el nombre `gráficas.js`.

Después de montar los archivos necesarios para el sitio *web*, dentro de la función *setup* se agrega un evento para manejar el error 404, el cual se presenta cuando se ingresa a una página no existente, el nombre de la función que será ejecutada cuando esto suceda tiene el nombre `notFound`. Por último, se inicia tanto el servidor *web* como el servidor *WebSocket*, así mismo se configura una función llamada `websocketEvent` para manejar los eventos de este servidor.

Lo siguiente en el código es la función *loop*, para que el microcontrolador ejecute repetidamente todo lo que se encuentre dentro de esta función. Lo primero que se debe de realizar es ejecutar el método `websockets.loop()`, debido a que este es el encargado de mantener activas las conexiones del servidor *WebSocket*, así como de también manejar la comunicación en tiempo real. Después se enciende el *LED* `ledPinEncendido` para indicar al usuario que el sistema ha iniciado.

La estructura principal de esta función es un *if* debido a que mientras la validación de la bandera `bEmpezar` no sea verdadera, lo único que se ejecutará será el método previamente mencionado después de apagar el *LED* con nombre `ledPinCensado`. Por otro lado, cuando la bandera sea verdad iniciará el proceso de muestreo de temperatura. Dentro de dicho proceso lo primero en realizarse es el encendido del *LED* por nombre `ledPinCensado` para indicar al usuario que el estudio de la temperatura está en ejecución, después se declara un cronómetro gracias a la

función *millis* y una variable entera de 32 *bits* llamada “prevMillis”, este cronómetro estará dentro de otra estructura *if* para que solo se ejecute cuando el tiempo transcurrido menos el tiempo anterior almacenado sea igual o mayor al tiempo de muestro ingresado por el usuario (todo esto en milisegundos, el usuario agrega la cantidad en segundos es por eso que la variable se multiplica por mil). Cuando lo anterior se cumple, se presenta otra estructura *if-e/se* para separar el funcionamiento del programa para uno o dos termopares (esta cantidad la agrega el usuario en el sitio *web*).

Si la cantidad de termopares solo es uno, lo primero en realizarse es guardar el tiempo de la función *millis* en la variable *prevMillis* para que el cronómetro pueda seguir funcionando, después se establece con el número uno la variable “bMediciones” para indicar que ya hay mediciones de temperatura y que solo se utiliza un termopar. Posteriormente se crea una variable llamada “temperatura1” del tipo flotante en el que se almacena el resultado de la función “leer_termopar” en el que se envía como parámetro la variable MAX667_CS1. Luego, mediante el método “push_back” de los vectores *tempTmk1* y *tTmk1*, se añaden los valores de las variables “temperatura1” y “tiempo” respectivamente. Después, se calcula el diferencial de temperatura, si y solo si, el tiempo no es cero. Una vez con el cálculo hecho, igual que anteriormente, se añade el valor de la variable “derivada” al vector *dfTmk1* y se ejecuta la función “enviarMedición” junto a un parámetro verdadero. Por último, se reasignan los valores a las variables que fungen de apoyo para el cálculo del diferencial de temperatura y el valor del tiempo.

Por otro lado, si la cantidad de termopares son dos, la estructura de lo programado en el caso anterior es la misma, pero con más datos, por ejemplo. La variable *prevMillis* vuelve a guardar el valor del tiempo de la función *millis*. Sin embargo, ahora la variable “bMediciones” se establece con el valor del número dos, para indicar de igual manera que ya hay mediciones de temperatura pero que ahora se utilizan dos termopares. Después, se crean dos variables de tipo flotante “temperatura1” y “temperatura2” donde su valor será el resultado de la función “leer_termopar” pero el parámetro para dicha función será distinto para cada una, para la primera será “MAX667_CS1” mientras que para la otra será “MAX667_CS2”. Luego se añaden los resultados a los vectores *tempTmk1* y *tempTmk2* y la variable “tiempo” se añade al vector *tTmk1*. Posteriormente se hace el cálculo de los dos diferenciales

de temperatura (tanto para el termopar 1 como para el termopar 2), de igual forma solo si el tiempo no es cero, y se añaden a sus respectivos vectores (dfTmk1 y dfTmk2). Por último, se ejecuta la misma función “enviarMedicion” pero ahora con un parámetro falso, y de igual forma, se reasignan los valores de las variables de apoyo para el cálculo de los diferenciales de temperatura y el valor del tiempo.

Con lo anterior se culmina la programación de la función *loop*, sin embargo, se mencionan dos funciones dentro de la misma “leer_termopar” y “enviarMedicion”. La primera consiste en una función cuyo principal objetivo es el de obtener la medición de temperatura de un termopar mediante el protocolo *SPI*, para elegir cuál de los dos termopares será del que se obtenga dicho dato, se tendrá que pasar como parámetro de la función el pin CS del módulo *MAX6675*. Lo primero en declararse es una variable llamada “v” del tipo entero de dieciséis *bits* para almacenar los datos obtenidos por el módulo. Lo siguiente es seleccionar el módulo del termopar en cuestión, es por eso que se envía una señal baja y posteriormente se establece una pausa de un milisegundo para asegurar la selección correcta. Luego, mediante el método “transfer” se envía un *byte* a través del bus *SPI* al módulo del termopar para que este, a su vez regrese un *byte* con información de la medición y almacenarlo en la variable creada, inmediatamente después se desplaza el contenido de “v” ocho bits a la izquierda para repetir el proceso anterior, es decir, mediante el operador *OR* y el método “transfer” obtener un segundo *byte* de información del módulo *MAX6675*. Continuando, se deselecta el chip enviando una señal alta debido a que ya se obtuvo la información necesaria de la medición. Mediante una estructura *if* se valida si el *bit* número dos es igual a uno, recordemos que este bit en el módulo *MAX6675* es el encargado de indicar si existe un problema con el termopar, de ser así se enciende el *LED* de nombre “ledPinError”, se imprime un mensaje de error en la consola serial y la función retorna *NAN* (*Not-A-Number*), de no haber ningún problema con el termopar, la función solo continúa desplazando hacia la derecha y eliminando los tres *bits* menos significativos de la variable “v”, ya que no son relevantes para la obtención de la temperatura. Por último, la función retorna el producto de la variable “v” y el valor 0.25, porque así se obtiene el valor de temperatura (se multiplica por dicho valor, ya que la variable representa la temperatura en “pasos” de 0.25 °C).

La segunda función con nombre “enviarMedicion” tiene como objetivo el enviar todos los datos de la medición de uno o los dos termopares, esto dependiendo del valor booleano obtenido como parámetro. Es por lo anterior que la función comienza con una estructura *if-else*, si el estado de la variable “bandera” es verdadero, significa que solo se enviarán los datos de un solo termopar, caso contrario, si el estado es falso, se enviará la información de los dos termopares. Ambos casos siguen una misma estructura, solo varían en la cantidad de información. Para ambos se crea un objeto *JsonDocument* de nombre “medicion”, que será usado para construir el *JSON* de información, posteriormente a dicho objeto se añade un campo llamado “indicador”, que actuará como identificador como se vio en el *BackEnd* del sitio *web*, con un valor de “medicion” para el caso de un solo termopar y un valor “medicion2” para el caso del envío de ambos. Lo siguiente es la creación de múltiples arreglos *JSON* que se añaden al *JSON* creado previamente, estos arreglos se crean a partir de la iteración de los vectores con la información de la medición esto junto al método “add” para ingresar uno a uno los valores al arreglo. Los arreglos creados para el envío de la información de un solo termopar son:

- temp1: Contiene los datos de temperaturas del termopar 1 y su campo identificador es “tempTmk1”.
- tiempo1: Contiene los datos de tiempo del termopar 1 y su campo identificador es “tTmk1”.
- dfTemp1: Contiene los datos del diferencial de temperatura del termopar 1 y su campo identificador es “dfTmk1”.

Mientras que los arreglos creados para el envío de la información para los dos termopares son los que se acaban de mencionar, pero además se suman los siguientes:

- temp2: Contiene los datos de temperaturas del termopar 2 y su campo identificador es “tempTmk2”.
- dfTemp2: Contiene los datos del diferencial de temperatura del termopar 2 y su campo identificador es “dfTmk2”.

El arreglo de tiempo para el segundo termopar no es enviado, porque el intervalo de tiempo de muestreo para ambos termopares es el mismo.

Por último, para ambos casos se crea una variable de tipo *String* llamada “datas” para poder serializar el objeto “medicion”, es decir, se convierte el objeto *JSON* a una cadena de texto, cadena que mediante el método “broadcastTXT” envía a través del *WebSockets* a todos los clientes conectados. Al final, solo se imprime en la consola serial esa misma cadena.

Continuando con el código, lo siguiente en declararse es la función que gestionará los eventos que suceden en el *WebSocket* del *ESP32*. Para esta función llamada “webSocketEvent” se establecen cuatro parámetros de entrada:

- num: Es el identificador o número de cliente asignado por el servidor *WebSocket*.
- type: Es el tipo de evento que ocurrió (puede ser una conexión nueva, una desconexión o un envío de mensaje).
- payload: Es el contenido del mensaje recibido en formato de *byte*.
- length: Es la longitud del mensaje recibido.

Esta función se compone principalmente de una estructura *switch* para evaluar el tipo de evento que suceda. En el caso de que un cliente del servidor se ha desconectado, se ejecuta el caso “WStype_DISCONNECTED”, en este evento solo se imprimirá un mensaje en el monitor serial de que un cliente se ha desconectado (acompañado de su identificador).

Para el caso “WStype_CONNECTED”, este se activa cuando se detecta la conexión un cliente nuevo. Para temas de *debug* (en caso de necesitarse) se obtiene la *IP* de dicho cliente y se imprime en el monitor serial, junto a su identificador y al parámetro *payload*. Después, se crea un objeto *JSON* llamado “configuración” para enviar la configuración actual de la página web al cliente que se acaba de conectar. Este objeto se compone de cuatro claves. La primera clave “indicador” tiene el valor “inicio” que funcionará como identificador, la segunda clave “cantidadTermopares” con el valor de la variable “cantidadTermopares” convertido a *String*, será para indicar cuántos termopares se están

utilizando (en caso de que ya haya sido iniciado el estudio), la tercera clave “tiempoMuestreo” toma el valor de la variable del mismo nombre y lo convierte a tipo *String* para hacer saber el tiempo actual entre mediciones, y la cuarta clave “botonEstados” envía el valor booleano de la variable con el mismo nombre. Posteriormente, se realiza la serialización de este objeto, es decir, mediante la función “serializeJson” se convierte el objeto *JSON* a una cadena de texto y, como ya se mencionó, mediante el método “broadcastTXT” se envía dicha cadena de texto a todos los clientes conectados. Y lo último de este caso, es la comparación de la variable “bMediciones”, ya que, si dicha variable posee un valor de uno o dos, se ejecuta la función “enviarMediciones” con un estado verdadero o falso, respectivamente, en caso de que la variable este vacía, no se ejecuta nada y solo termina el caso.

El siguiente caso para esta función “WStype_text”, corresponde a la recepción de un mensaje dado por el cliente al microcontrolador a través de la página *web*. La primera acción que realizará el *ESP32* será de imprimir el mensaje obtenido en el monitor serial, para posteriormente procesar el objeto *JSON* recibido mediante la función “deserializeJson” para convertirlo en un objeto del tipo *JsonDocument* y de esa forma poder analizar el contenido del mensaje. Al momento de realizar la deserialización, si presenta un error, se imprimirá dicho mensaje en el monitor serial y se encenderá el *LED* de nombre “ledPinError”. Por otro lado, si la conversión es exitosa, iniciará una estructura *if* donde se encargará de detectar qué valor contiene el mensaje en el campo llamado “indicador”, existen tres posibilidades:

1. “configweb”: Se actualizan los valores de las variables cantidadTermopares, tiempoMuestreo, botonEstados y bEmpezar.
2. “resetWeb”: Además de actualizar las variables cantidadTermopares, tiempoMuestro y botonEstados, se ejecuta una función llamada “reset”.
3. “finalizar”: Solamente se actualizan las variables bEmpezar y botonEstados.

Por último, si por algún motivo el valor del campo no concuerda con ninguna de las tres posibilidades, el sistema solo imprimirá el mensaje “Ni un caso” en el monitor serial y seguirá con la ejecución del programa.

La función “reset” mencionada anteriormente, solo consiste en igualar a cero todas las variables (bMediciones, tiempo, tiempoAnterior, tempAnterior, deltaT, deltaTemp y derivada), además de limpiar con el uso del método “clear” todos los vectores (tTmk1, tempTmk1, dfTmk1, tTmk2, tempTmk2 y dfTmk2).

Lo último dentro de esta programación del microcontrolador es el manejador del evento de “error 404”, dicho problema ocurre cuando un cliente intenta acceder a una ruta o página que no existe en el servidor, y cuando lo anterior sucede, el programa se encargará de redireccionar al cliente a la página raíz del sitio *web*.

3.3 Elaboración de la PCB

3.3.1 Esquemático

Este proyecto se diseñará con el *software* de *proteus 8* pero para ello será necesaria la creación del componente *ESP32*, porque este microcontrolador no se encuentra dentro de la biblioteca de componentes de la aplicación, para ello siga los pasos del anexo 8.8.

Una vez con el recurso instalado se diseña el esquemático, mostrado en la figura 76, en el cual se utilizan dos conectores de cinco filas y una columna (para indicar la conexión de los dos módulos *MAX6675*), tres conectores de dos filas y una columna (para indicar la conexión de los tres *LEDs*), y el microcontrolador *ESP32*.

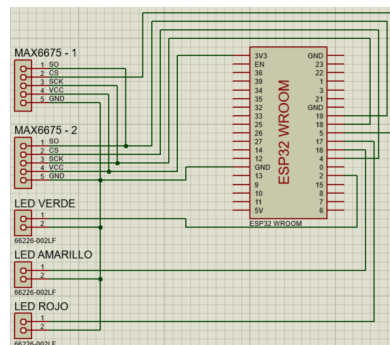


Figura 76: Esquemático del circuito del dispositivo mecatrónico.

Fuente: Elaboración propia.

Los pines *SO* y *SCK* de los módulos *MAX6675*, están conectados a los pines físicos 31 y 30 del microcontrolador, porque estos son los reservados para utilizar el protocolo *SPI*. Los pines *CS* son conectados a las entradas *GPIO* 5 y 4, como se estableció en la programación del *Arduino IDE*. Los pines *VCC* son conectados a la salida de 3.3 V del microcontrolador. Por otro lado, los pines de los *LEDs*, como se mencionó en la programación, son conectados a las entradas *GPIO* 2, 16 y 17. Por último, se unen todos los negativos para evitar problemas relacionados a no unificar las tierras.

3.3.2 PCB Layout

Una vez que se conectan todos los componentes de la placa, lo siguiente es dirigirse a la pestaña de “PCB Layout” para construir y diseñar la *PCB*. En este apartado se colocan los componentes, se unen las conexiones con pistas, además de poder modificar el tamaño de los orificios y las huellas de los pines.

Para este dispositivo se diseñó la *PCB* con componentes *through hole*, es decir, se tomará en cuenta la utilización de componentes que se colocarán en la superficie de la vaquelita y se soldarán por debajo de la misma. Además de utilizar la capa de cobre existente como la pista negativa, para así facilitar el maquinado de la placa de circuito. El tamaño aproximado de la *PCB* final, se establece de 100mm de largo y 5mm de ancho. Otra cosa que mencionar es que se utilizarán tres puentes (representados de color rojo) porque no fue posible diseñar una placa de circuito con la totalidad de sus pistas en la capa inferior de cobre. El diseño del circuito se muestra en la figura 77.

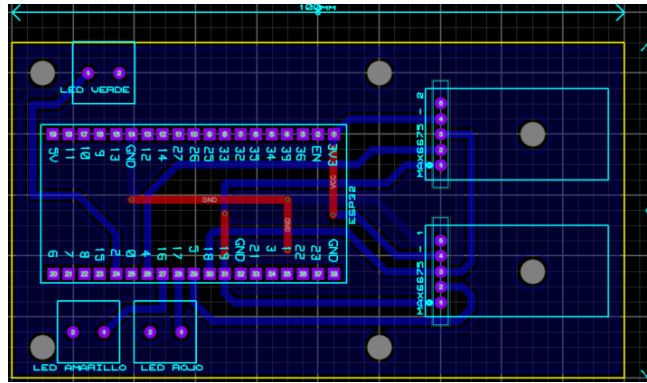


Figura 77: Diseño de la PCB.

Fuente: Elaboración propia.

Para continuar con la fabricación de la placa de circuito impreso es necesario exportar una serie de archivos del proyecto de *Proteus*, para ello se selecciona la opción de la ruta Salida > Generar un archivo Gerber/Excellon.

Una vez con la ventana abierta, es necesario colocar un nombre deseado a los archivos, así como también de agregar la ruta donde se guardarán los archivos que se generen. Los archivos que se seleccionarán son los siguientes: Cobre inferior y Taladro. También se selecciona la opción “Espejado” y el formato de *Gerber* será “RS274X”, la configuración se muestra en la figura 78.

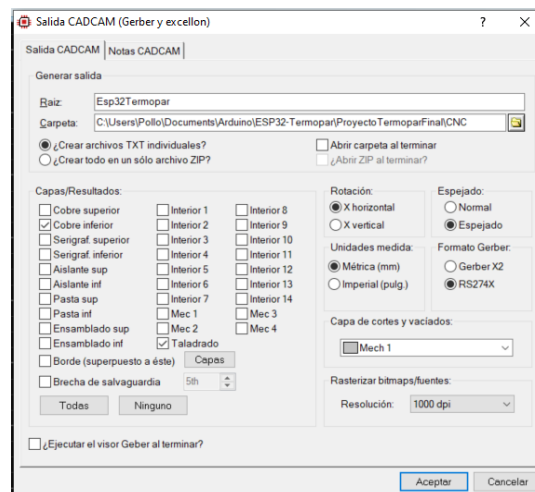


Figura 78: Generación de archivos para el mecanizado de la PCB.

Fuente: Elaboración propia.

3.4 Generación de código G

Para generar el código G que será necesario para mecanizar la placa de circuito impresa, se utilizará el *software* gratuito de *FlatCAM*, en este abriremos el archivo *Gerber* generado para la capa de cobre inferior de la placa. El cobre que se preservará se representa con un color verde en esta aplicación, lo anterior se muestra en la figura 79.

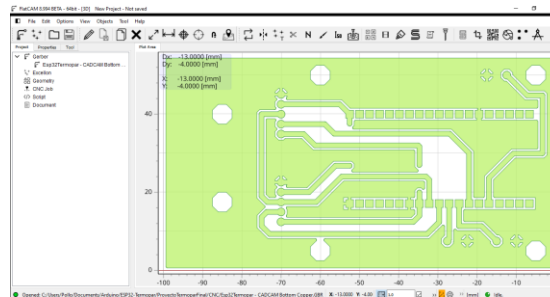


Figura 79: Archivo *Gerber* de la capa de cobre inferior de la *PCB*.

Fuente: Elaboración propia.

Lo siguiente en este proceso es trazar las rutas por las que la *CNC* deberá de seguir para eliminar las zonas blancas y así crear las pistas de la *PCB*. Para ello, se utilizará la herramienta “Non-Copper Clearing” que es nativa de este *software*. Donde el primer parámetro que debe ser agregado es el diámetro efectivo de corte de la broca a utilizar, en este caso se utilizará una broca de tipo V cuya punta de es 0.2 mm y su ángulo es de 20°, por lo que, para saber este dato, se utilizará la calculadora de *FlatCAM* (figura 80).

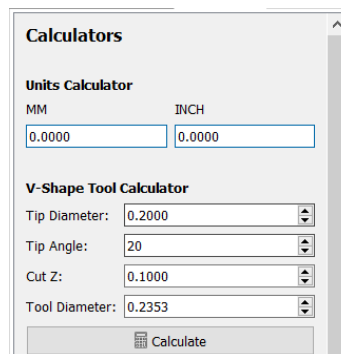


Figura 80: Calculo del diámetro efectivo de corte de la broca.

Fuente: Elaboración propia.

El resultado fue un diámetro efectivo de 0.235 mm, por lo que se agrega en el parámetro “Tool Dia” y se pulsa el botón “Search and Add” para agregar la broca. El siguiente parámetro que editar es “Overlap”, este hace referencia a la superposición que tendrá una ruta sobre otra (se expresa en porcentaje), para este caso se usará una superposición del 40% para evitar tener una mala eliminación de cobre. El próximo parámetro es “Method” el cual hace referencia al método que el software utilizará para la generación de las trayectorias de la broca, se utilizará “Standard”. Por último, el parámetro “Margin” es el margen que la aplicación tendrá para eliminar el cobre alrededor de los límites de la placa, ya que esto no es una necesidad en esta aplicación, se colocará un valor de cero. Después de todo simplemente se hace clic en el botón “Generate Geometry”, las líneas de color rojo mostradas en la figura 81, son las rutas que tendrá la broca.

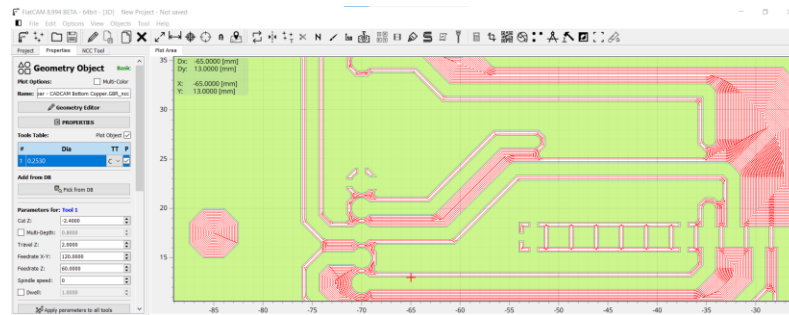


Figura 81: Geometría de las rutas de la broca para la eliminación de cobre.

Fuente: Elaboración propia.

Lo siguiente a realizar dentro de las propiedades de la geometría obtenida, es indicar la profundidad de corte que se tendrá, como se mencionó en el cálculo del diámetro efectivo de corte, será de 0.1 mm. El próximo parámetro es la distancia de seguridad en el eje Z, es decir, es la distancia que la CNC levantará la broca para cambiar de trayectoria, será de 2 mm para evitar colisiones en el mecanizado. La velocidad de movimiento en el plano X-Y se quedará lo predeterminado (120 mm/min), sin embargo, para el eje Z se usará una velocidad de 25 para evitar dañar la broca al momento de perforar. Por último, la velocidad de giro del motor se configurará a mil, y se presiona el botón “Generate CNCJob object” para generar el código G (este se muestra en la figura 82).

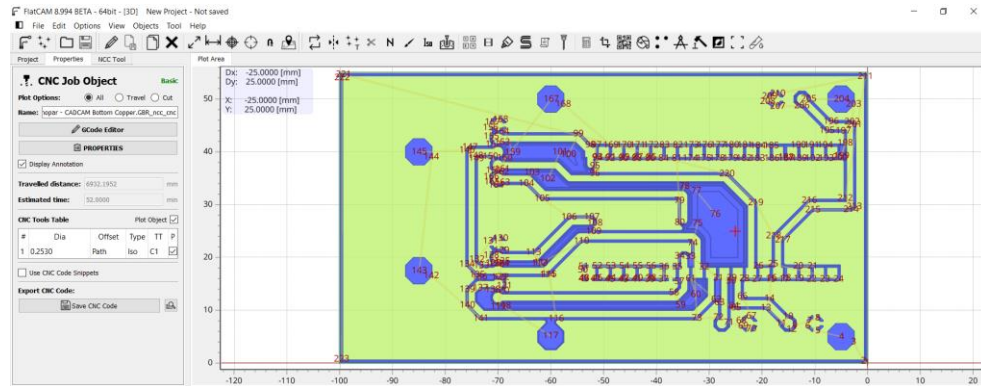


Figura 82: Código G de la placa generado por *FlatCAM*.

Fuente: Elaboración propia.

El color azul de la figura 82, representa las trayectorias que seguirá la máquina *CNC* en un plano tridimensional, además de eso, *FlatCAM* también brinda una aproximación del tiempo que tendrá el mecanizado, el cual será de aproximadamente 52 minutos. Al final se presiona el botón “Save CNC Code” para guardar el archivo con el código G necesario para mecanizar las pistas de la placa de circuito.

Después de tener el código G para la eliminación del exceso de cobre de la cara inferior, es necesario importar el archivo *Excellon* de los taladros que tendrá la placa para iniciar su configuración y generar su código G al igual que las pistas. En el proyecto podemos deshabilitar la geometría y el código G que acabamos de generar simplemente dando clic derecho y seleccionar la opción “Disable Plot” para poder tener un área de trabajo limpia y evitar confusiones, en la figura 83 se puede observar cómo se ve el proyecto con la importación de los taladros (representados por un color rojo).

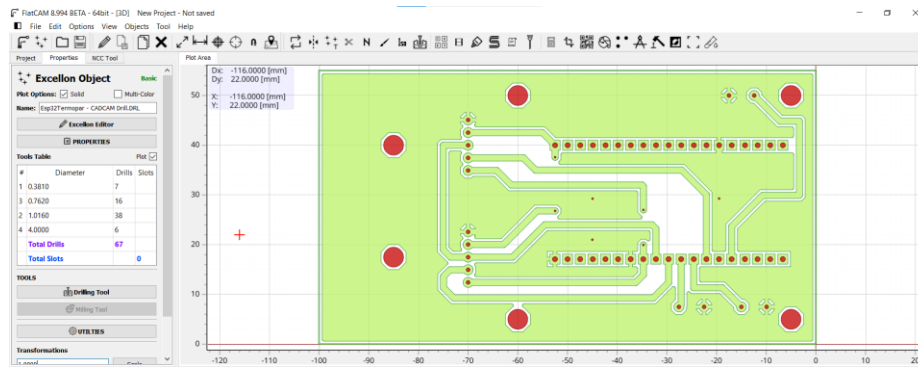


Figura 83: Cara inferior de la *PCB* junto a los taladros a realizar.

Fuente: Elaboración propia.

Para modificar los parámetros del objeto *Excellon*, primero se debe de pulsar en el botón “Drilling Tool”, dentro de esta opción se selecciona el diámetro de la broca (la tabla de la figura 84) con la que se trabajará.

Drilling Tool Basic			
EXCELLON:			
Esp32Termopar - CAD/CAM Drill.DRL			
#	Diameter	Drills	Slots
1	0.381	7	
2	1.016	38	
3	0.762	16	
4	4.0	6	
Total Drills		67	
Total Slots			0

Tool order: ☒ No ☐ Forward ☐ Reverse

Figura 84: Tabla de dimensiones de las brocas a usar en el proyecto.

Fuente: Elaboración propia.

Después de lo anterior, ya se puede ingresar la profundidad de corte para perforar la placa, siendo de 2.5 mm. La distancia de seguridad en el Eje Z será de 2 mm y la velocidad de este mismo eje será de 25 mm/min. La velocidad de giro de la herramienta volverá a ser de 1000. Por último, se presiona el botón “Generate CNCJob object” para generar el código G de la perforación de la *PCB*, esto puede verse en la figura 85.

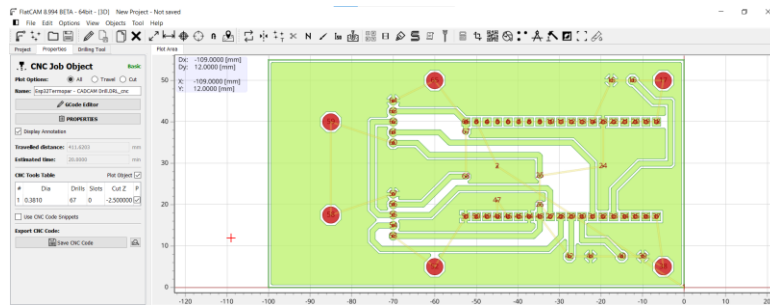


Figura 85: Código G para las trayectorias de perforación.

Fuente: Elaboración propia.

Lo anterior debe repetirse para cada una de las brocas generando así el código G para cada dimensión de broca. Excepto para la broca de 4 mm, ya al no contar con una de ese tamaño se mecanizarán las perforaciones con una broca más pequeña, para ello, en las propiedades del archivo *Excellon* se selecciona la broca de 4 mm y se presiona el botón “UTILITIES” y dentro de estas opciones en el parámetro “Milling Diameter” colocamos el valor de 1 mm para utilizar una broca de ese grosor, después se presiona el botón “Mill Drills” y genera la geometría para el mecanizado de todas esas perforaciones, por lo que, igual que anteriormente, se tiene que agregar una profundidad a esa geometría colocando una profundidad de 2.5 mm, 2 mm de seguridad en el eje Z, una velocidad en los plano X-Y y Z de 25 mm/min y de 1000 para la velocidad de giro de la herramienta, y se presiona el botón “Generate CNCJob object” para tener el código G, en la figura 86 se muestra el código G para todas las perforaciones a realizar.

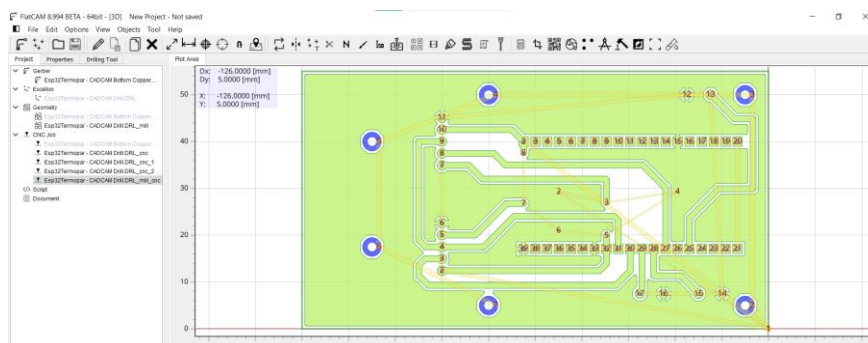


Figura 86: Trayectorias en código G para perforar la CNC.

Fuente: Elaboración propia.

Antes de terminar, falta generar un código G para el corte de la placa, para ello, con la herramienta de corte “Cutout PCB” se indicará como referencia el archivo *Gerber* de las pistas, se usará una broca de 1 mm de diámetro para efectuar este corte a una profundidad de 2.5 mm y se desactivará la casilla “Multi-Depth” debido a que el tajo será con solo una trayectoria a una sola profundidad. En este caso no se generarán puentes para mantener fijada la placa, porque esta se encuentra sujeta en su totalidad. Se presiona el botón “Generate Geometry” (el que tiene una equis como icono) y se procede a dar la profundidad de corte a la geometría como se ha hecho anteriormente y se genera el código G, al final debería verse como en la figura 87.

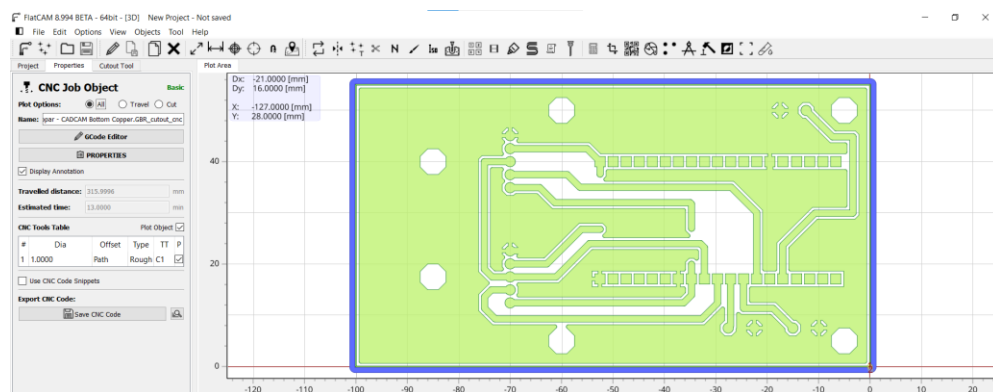


Figura 87: Código G para el corte de la *PCB*.

Fuente: Elaboración propia.

Finalmente, solo hace falta la descarga de todos los códigos G generados para proceder con el mecanizado de la *PCB* en la máquina *CNC*.

3.5 Construcción de la *PCB*

Para realizar el grabado del circuito de la placa, se utilizará una Fresadora *CNC* (figura 88) junto al software *Candle*.



Figura 88: Fresadora CNC 1610.

Fuente: https://articulo.mercadolibre.com.mx/MLM-726063865-maquina-cnc-grabadora-cortadora-cnc-1610-laser-2500mw-usb-_JM

Una vez colocada la baquelita con la capa de cobre hacía arriba en la plataforma de trabajo de la CNC, se debe abrir el software de *Candle* para el control de la máquina. Dentro del *software* en la opción Servicio > Opciones, se debe de seleccionar el puerto COM al que está conectada la máquina y seleccionar la cantidad de baudios para la comunicación, en este caso el puerto será el 10 y la velocidad de 115200, véase figura 89.

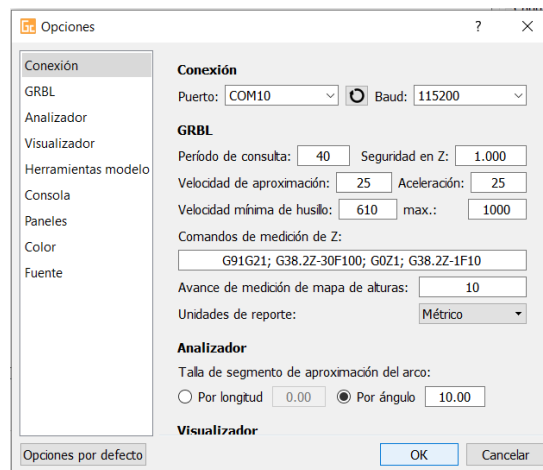


Figura 89: Configuración para la comunicación con la máquina CNC.

Fuente: Elaboración propia.

En la aplicación con el botón “Abrir” se debe seleccionar el código G correspondiente a las pistas, véase el ejemplo de la figura 90.

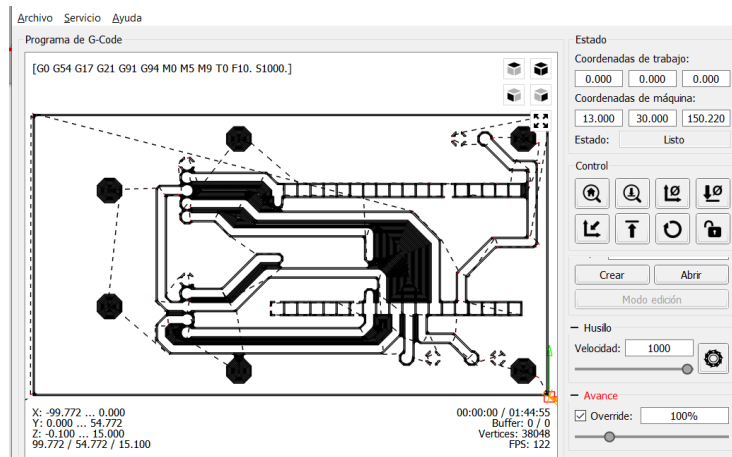


Figura 90: Código G para las pistas de la PCB.

Fuente: Elaboración propia.

Se debe de posicionar a la CNC manualmente en el origen de la placa, para ello, se puede utilizar el control manual que ofrece la aplicación. Una vez posicionada en el origen en el plano X-Y de manera manual, el eje z se puede auto nivelar con la opción “Medición de Z” que ofrece el *software*, es importante mencionar que, para utilizar esta herramienta, se debe de conectar mediante cables u otro elemento los pines A5 de la tarjeta de control uno a la broca y otro a la capa de cobre de la placa.

Ahora ya con la máquina en su origen en los tres ejes cardinales, se debe de crear un mapa de alturas, esto con la finalidad de asegurar un correcto mecanizado, ya que es normal que las baquelitas no tengan una correcta rectificación. Para generar este mapa, se debe de presionar el botón “Crear”, el botón se muestra en la figura 91.

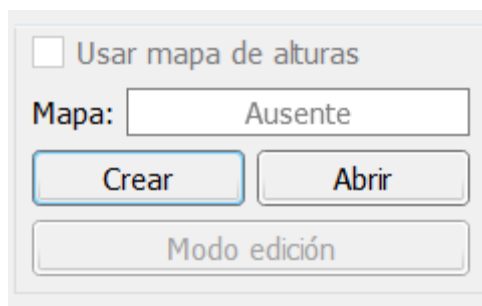


Figura 91: Botón para crear un mapa de alturas.

Fuente: Elaboración propia.

Posteriormente, se debe de presionar el botón “Auto” para que el *software* automáticamente genere un área para el mapa del tamaño exacto de la placa. En esta ocasión debido al tamaño de la placa, se propone el uso de 36 puntos de medición para la generación del mapa. Una vez con estos dos parámetros establecidos se presiona el botón “Medición” para que *Candle* de manera automática realice las 36 mediciones, en la figura 92 se muestra el mapa de alturas generado.

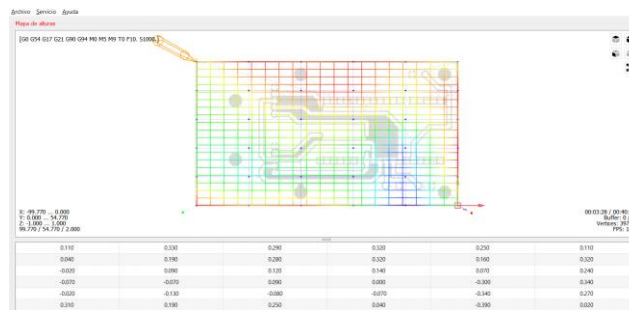


Figura 92: Mapa de alturas obtenido automáticamente.

Fuente: Elaboración propia.

Ya con el mapa de alturas obtenido, solo es necesario pulsar en el botón “Modo edición” para salir de la interfaz del mapa de alturas, y una vez afuera marcar la casilla “Usar mapa de alturas” para que así el *software* corrija el código G en base a las mediciones del mapa. Posteriormente se enciende el husillo e inicia el proceso de grabado pulsando en el botón “Enviar”.

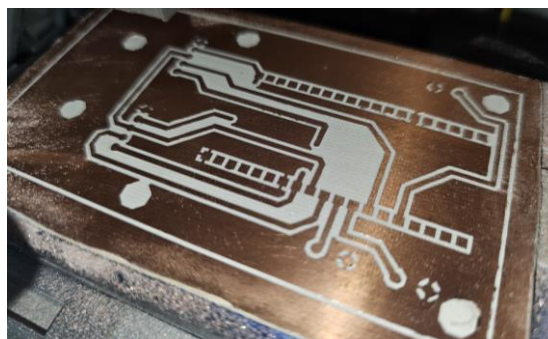


Figura 93: Eliminación del cobre excedente de la placa de circuito.

Fuente: Elaboración propia.

Una vez con las pistas de la capa inferior mecanizadas, véase figura 93, la broca debe de sustituirse para continuar con la perforación de esta. Primeramente, se utilizará la broca de 0.7 mm de grosor para perforar aquellos orificios en los que se colocarán los puentes de la parte superior de la *PCB*.

Una vez con la herramienta sustituida, en la interfaz de *Candle* se debe de abrir el archivo del código G de estas perforaciones, posterior a eso, se calibra el origen para el eje Z, como ya se ha mencionado, y sin la necesidad de utilizar un mapa de alturas, porque recordemos que son perforaciones, se inicia con el proceso al presionar el botón “Enviar”, obteniendo como resultado la figura 94.

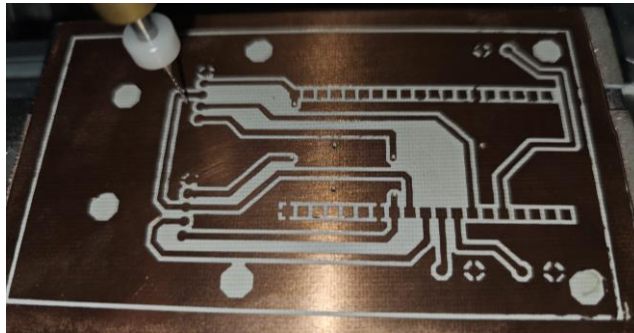


Figura 94: Perforaciones para puentes de la cara superior.

Fuente: Elaboración propia.

El proceso anterior se debe de seguir para todas las perforaciones de la placa de circuito impreso con las medidas requeridas, en este caso se utilizará la broca de 0.9 mm para la elaboración de los agujeros de los componentes y una broca de 1 mm para realizar los agujeros de 4 mm, por último, esta misma broca se utilizará para realizar el corte de la *PCB*, teniendo como resultado la figura 95.

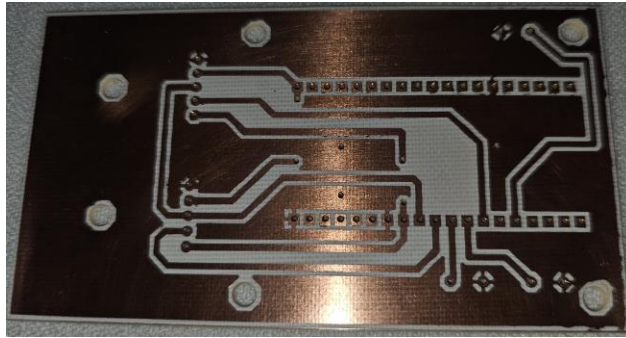


Figura 95: *PCB* perforada y cortada.

Fuente: Elaboración propia.

Para finalizar con el mecanizado de la placa de circuito, solo será necesario de lijar con una lija de grano fino ambas superficies, para así eliminar cualquier protuberancia que se haya originado con las brocas. Después con un cepillo se frota la cara inferior (donde se ubica el cobre) para remover los restos de cobre que hayan podido quedar entre las pistas. Y así, con un paño ligeramente húmedo de alcohol isopropílico, se limpia la superficie para obtener una *PCB* lista para el proceso de soldadura, figura 96.

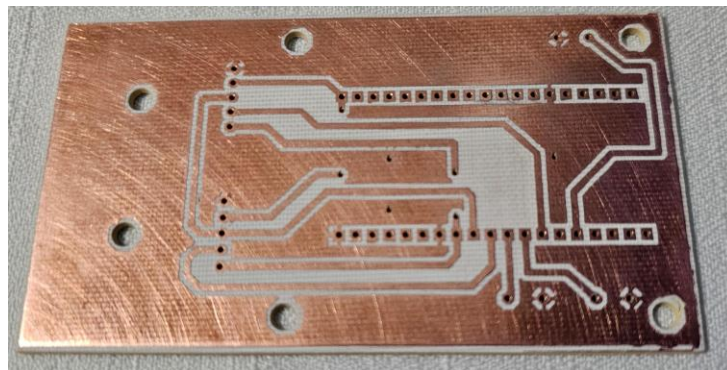


Figura 96: Placa de circuito impreso limpia para soldadura.

Fuente: Elaboración propia.

Para continuar con el proceso de la elaboración de la *PCB*, lo siguiente a realizar es a soldadura de los componentes de esta, para ello seguiremos el diagrama eléctrico de la figura 76.

El orden de los componentes por soldar será seleccionado por su tamaño, para facilitar así la fabricación, es por ello por lo que el orden será:

1. Puentes de cable calibre 22.
2. Borneras de conexión 2x1.
3. Módulos *MAX6675*.
4. Pines de conexión tipo hembra (sobre estos irá el microcontrolador).

Con el orden establecido, se inicia con el proceso, cabe mencionar que para la soldadura se utilizará estaño 60/40 de 1 mm de diámetro.

En la figura 97 se puede apreciar la *PCB* ya finalizada y con los componentes soldados y colocados. Con esto la placa de circuito ya se encuentra lista para utilizarla para el estudio de la temperatura, cabe mencionar que los pines utilizados para soldar los módulos *MAX6675* no fueron removidos para que en caso de un error de funcionamiento del dispositivo pueda ser de fácil acceso para realizar un *debug*.

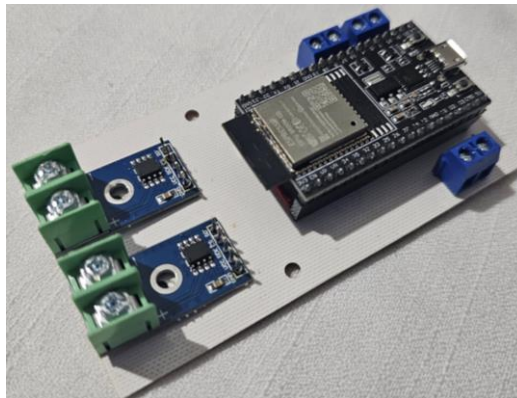


Figura 97: *PCB* finalizada.

Fuente: Elaboración propia.

3.6 Caja del dispositivo mecatrónico

3.6.1 Modelado 3D de la caja y tapadera

Para finalizar, el dispositivo creado será almacenado en una caja impresa en 3D, la cual será modelada mediante el *software CAD SolidWorks*.

El plano de la carcasa creada puede visualizarse en la figura 98, en donde se muestra a detalle la información necesaria para su modelado 3D. Las dimensiones serán de 121 mm de largo, 76 mm de ancho y 37 mm de alto, además el grosor de las paredes será de 8 mm, esto con el objeto de que la caja sea robusta para su manipulación en el laboratorio. En las esquinas de esta se colocaron cuatro agujeros pasados de 4 mm de diámetro para asegurar la tapadera (figura 99), mientras que en la base de la carcasa se colocaron otros cuatro agujeros, pero ahora de 5 mm de diámetro para colocar en estos la *PCB* desarrollada previamente. Por último, se realizaron tres cortes cuadrados, dos en la cara frontal para que las conexiones de los módulos *MAX6675* puedan salir, y uno en la cara trasera para permitir la conexión del cable que energiza al microcontrolador, las medidas de estos son 17 mm x 8 mm y 20 mm x 15 mm, respectivamente.

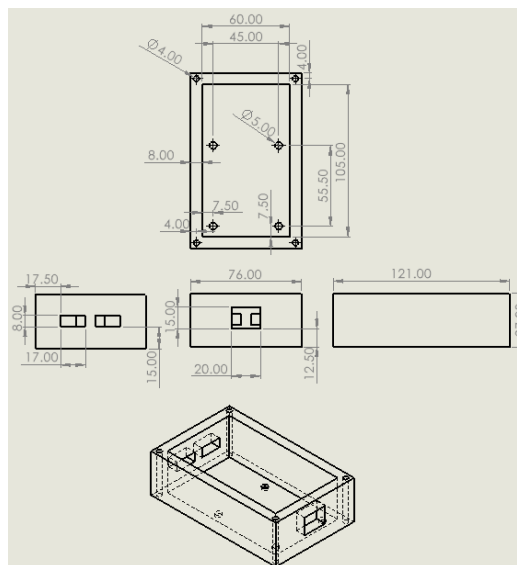


Figura 98: Plano de la caja donde se encontrará el dispositivo.

Fuente: Elaboración propia.

Las dimensiones para el largo y ancho de la tapadera son los mismos que la caja, así como también de las cuatro perforaciones necesarias para los tornillos en las esquinas. Para este diseño (figura 99) se optó por una tapadera con dos alturas distintas, esto para facilitar la tarea de alinear los orificios de las esquinas, dichas alturas serán de 4 mm y 3 mm dando como resultado una altura de 7 mm, y las dimensiones de la segunda altura serán de 58 mm x 103 mm. Al final se colocaron tres agujeros más con una dimensión de 5.5 mm de diámetro, ya que en estos se insertarán a presión los tres *LEDs* que utiliza el dispositivo mecatrónico, se optó por separar el color verde de los otros dos (rojo y amarillo), porque el primero solo indica que el proyecto está encendido, mientras que los otros dos funcionan como indicador de error (para el color rojo) y para indicar cuando el sistema se encuentre en operación (para el color amarillo).

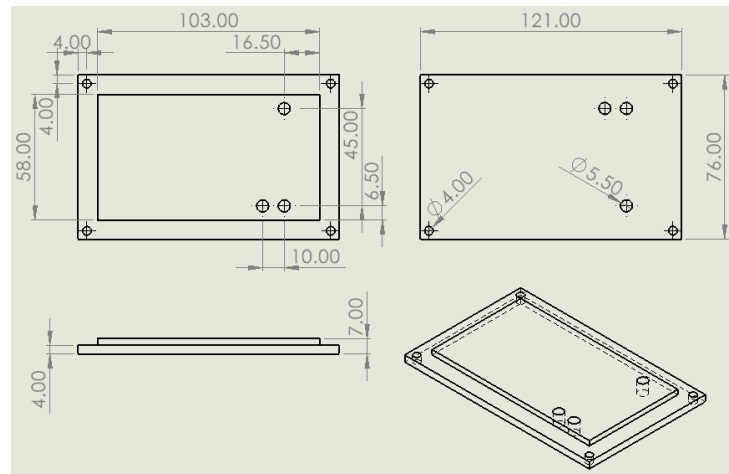


Figura 99: Plano de la tapadera de la caja.

Fuente: Elaboración propia.

Los diseños 3D de los planos mostrados por las figuras 98 y 99 se muestran en el anexo 8.10.

3.6.2 Impresión 3D de la caja y tapadera

Una vez con los diseños creados y convertidos al tipo de archivo *STL*, se deben de importar en el software *UltiMaker CURA*, el cual se encargará de generar el código G necesario para la impresión de los dos componentes, en la figura 100 se visualiza lo anterior.

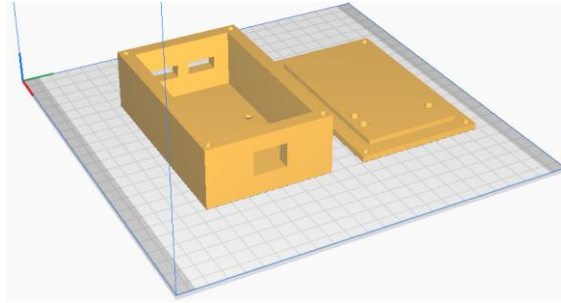


Figura 100: Modelos *STL* para su impresión 3D.

Fuente: Elaboración propia.

La impresora que se utilizará será una de la marca *Creality*, específicamente el modelo *Ender 3*, siendo esta una impresora *FFF* de tipo cartesiana. En cuanto al material de impresión se optó por *PETG*, esto debido a su facilidad de impresión y por su alta resistencia, siendo esto último una característica importante para que la caja pueda soportar las condiciones de uso en el laboratorio, además de ser de color naranja para su fácil identificación. En relación con lo anterior, se usará una temperatura de impresión de 230° C y una de 85° C para la cama de la impresora. Como parámetros adicionales; la altura de capa se definió de 0.2 mm, el número de líneas en las paredes es de tres, la densidad de relleno será de un 20% y el patrón de este será del tipo “subdivisión cúbica”, las características mencionadas se muestran en la figura 101.

Calidad ▼			
Altura de capa	 	0.2	mm
Paredes ▼			
Grosor de la pared		1.6	mm
Recuento de líneas de pared	f_x	3	
Expansión horizontal		0.0	mm
Superior o inferior ◀			
Relleno ▼			
Densidad de relleno		20.0	%
Patrón de relleno	f_x	Subdivisión cúbica	▼
Material ▼			
Temperatura de impresión	f_x	230.0	°C
Temperatura de la placa de impresión	 	85.0	°C

Figura 101: Parámetros de impresión.

Fuente: Elaboración propia.

Una vez con los parámetros establecidos, solo es necesario pulsar el botón “Segmentación” para que el *software* se encargue de generar el código G, con las características anteriormente dadas, para la impresora. Por lo que, este archivo se guarda en la tarjeta *microSD* propia de esta máquina y se procede con el proceso de impresión, obteniendo como resultado la figura 102.



Figura 102: Impresión 3D de la caja para el dispositivo mecatrónico.

Fuente: Elaboración propia.

3.7 Ensamble final del proyecto

Lo primero en realizarse será la soldadura de alambre estañado calibre 22 en los pines de los tres *LEDs* indicadores, usando color rojo para identificar su ánodo y negro para su cátodo, los tramos a soldar serán de 10 cm, para permitir la manipulación de la tapadera cuando se retire, los diodos *LED* soldados pueden verse en la figura 103.

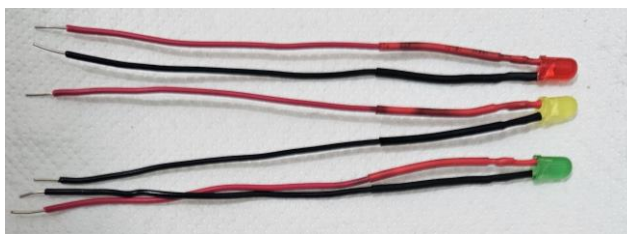


Figura 103: Indicadores LED para interfaz.

Fuente: Elaboración propia.

Luego, la *PCB* se fijará dentro de la caja mediante cuatro tornillos (M3 x 15 x 0.5 mm) y cuatro tuercas, como puede verse en la figura 104.



Figura 104: Placa de circuito fijada dentro de la caja impresa en 3D.

Fuente: Elaboración propia.

Posteriormente, se conectarán los *LEDs* a las tres borneras siguiendo el diagrama de la figura 76, la polaridad de la conexión se indicó con un plumón permanente en la *PCB*. Después, estos serán colocados a presión en los orificios de la tapadera (ver figura 105), para así lograr que se muestren al usuario una vez cerrado el dispositivo mecatrónico.

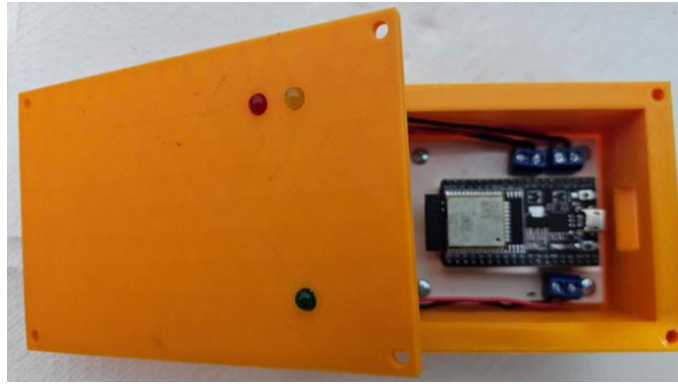


Figura 105: Indicadores LED montados en la tapadera de la caja impresa en 3D.

Fuente: Elaboración propia.

Antes de cerrar el dispositivo, se conectarán los dos termopares tipo K en los módulos *MAX6675* obedeciendo la polaridad, al mismo tiempo se conectará el cable micro *USB* al microcontrolador *ESP32*, el resultado de lo anterior se muestra en la figura 106.

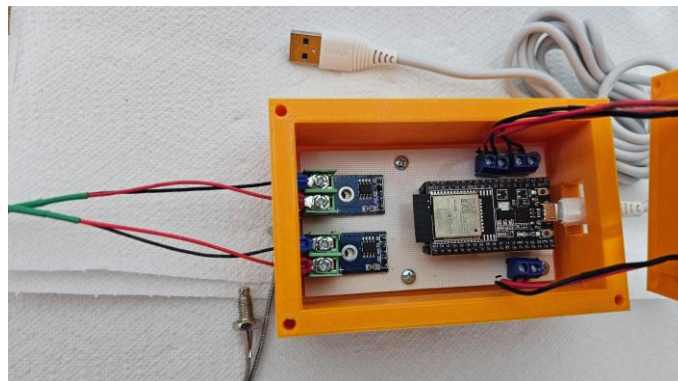


Figura 106: Dispositivo con los termopares tipo K y el cable de alimentación conectados.

Fuente: Elaboración propia.

Por último, la tapadera será fijada a la caja mediante cuatro tronillos (M3 x 50 x 0.5 mm) y cuatro tuercas, obteniendo como resultado final la figura 107.

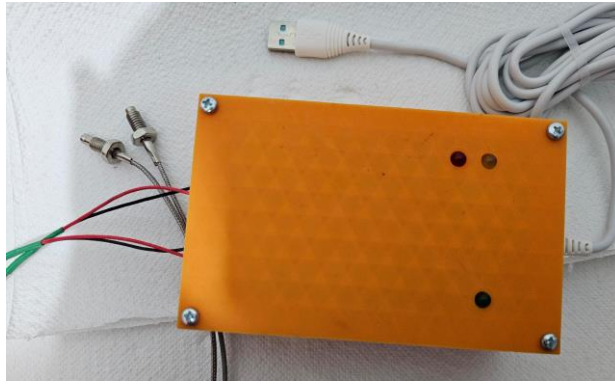


Figura 107: Dispositivo mecatrónico cerrado.

Fuente: Elaboración propia.

4 RESULTADOS

4.1 Preparación del dispositivo mecatrónico

Con la finalidad de utilizar el dispositivo para la medición de los tratamientos térmicos de los metales en el laboratorio, fue necesario reemplazar los termopares conectados en el capítulo anterior por el mostrado en la figura 108, el cual se trata de un termopar tipo K de 1/16 de pulgada por 2 metros de largo con conector macho mini.

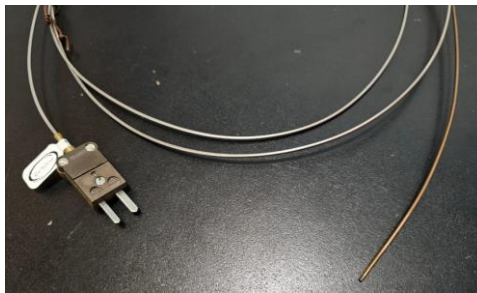


Figura 108: Termopar tipo K.

Fuente: Elaboración propia.

Otro detalle agregado al dispositivo para la facilitación de su manipulación y así que el alumno no confunda las conexiones de los termopares, es el serigrafiado mostrado en la figura 109, cuyo objetivo es el de identificar el termopar número uno y dos.



Figura 109: Serigrafiado para la identificación del termopar principal y secundario.

Fuente: Laboratorio de tratamientos térmicos del instituto.

4.2 Prueba del dispositivo mecatrónico

Lo primero realizado para el ensayo de la medición de temperatura, fue la elaboración de la probeta que se sometió al estudio, ésta en particular cuenta con un barreno de 1/16 de pulgada en uno de sus extremos (como se muestra en la figura 110), para que así el termopar pueda ser insertado dentro y medir la temperatura interna del metal.



Figura 110: Probeta de hacer AISI 1040 con barreno de 1/16 de pulgada.

Fuente: Laboratorio de tratamientos térmicos del instituto.

Posteriormente, el extremo del termopar fue colocado dentro del barreno de la probeta, para así colocar esta última dentro de la mufla, debido a que una vez que se someta a calentamiento, su manipulación sería más compleja, lo anterior se muestra en la figura 111.



Figura 111: Probeta con el termopar K en el centro de ésta, dentro de la mufla.

Fuente: Laboratorio de tratamientos térmicos del instituto.

Una vez la probeta dentro de la mufla, la puerta fue cerrada y mediante el panel de control de temperatura, se fijó un *setpoint* de 251 °C, como se puede observar en la figura 112.



Figura 112: Panel de control del horno mufla.

Fuente: Laboratorio de tratamientos térmicos del instituto.

Mientras la probeta era ingresada a la mufla, se configuró el ensayo de la temperatura del dispositivo mecatrónico desarrollado mediante su interfaz (figura 113), dentro de la cual se estableció el uso de un solo termopar y un tiempo de muestreo de un segundo.



Figura 113: Interfaz gráfica del dispositivo mecatrónico con la configuración elegida.

Fuente: Laboratorio de tratamientos térmicos del instituto.

Para lograr acceder a la interfaz gráfica del dispositivo, fue necesario conectarse a la red *WiFi* generada por el microcontrolador bajo el nombre “MedicionTermoparkK” (figura 114), utilizando a su vez la contraseña “LaboratorioTermo2024”. Con lo anterior realizado, en una pestaña en el buscador,

se colocó la URL “termopar.local” y así ya se tuvo control del dispositivo (esto se explica a profundidad en el capítulo 3.2).

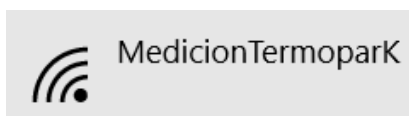


Figura 114: Nombre de la red *WiFi* generada por el dispositivo mecatrónico.

Fuente: Laboratorio de tratamientos térmicos del instituto.

Después de aproximadamente un minuto y medio iniciado el ensayo, la gráfica de temperatura obtenida se muestra en la figura 115. En la cual se observó como la temperatura interna de la probeta empezó en 29 °C y continuaba incrementando progresivamente hasta los 36 °C en el segundo 84.

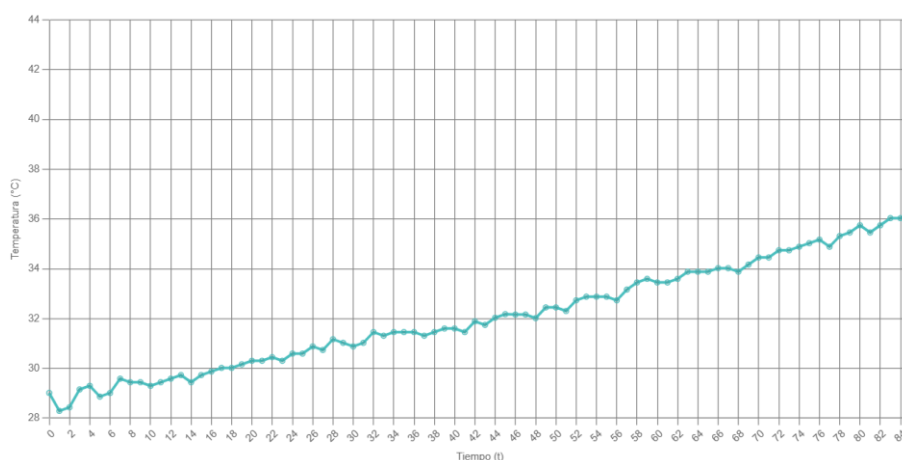


Figura 115: Gráfica de la temperatura interna de la probeta.

Fuente: Laboratorio de tratamientos térmicos del instituto.

Así mismo, la gráfica del diferencial de temperatura de la figura 116, mostró los incrementos o decrementos en la temperatura interna de la probeta segundo a segundo, siendo los cambios más grandes de 1.5 °C y en otros de 0 °C, es decir, la temperatura interna se mantenía constante.

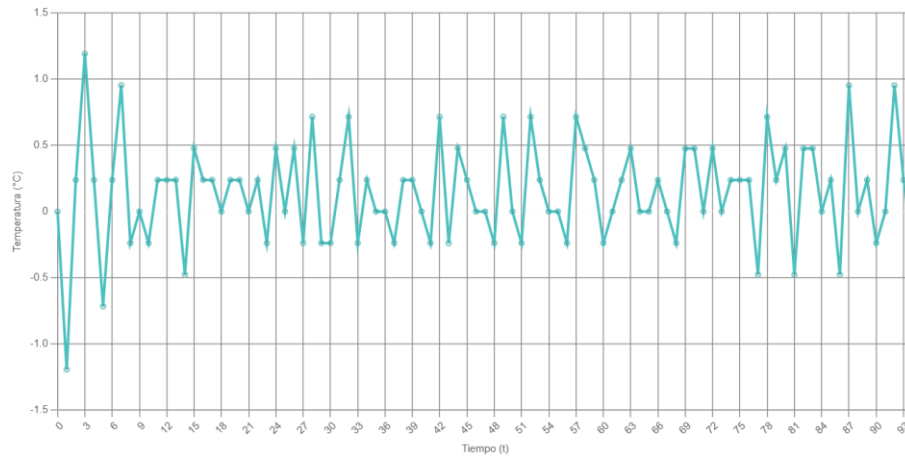


Figura 116: Gráfica del diferencial de temperatura interna de la probeta.

Fuente: Laboratorio de tratamientos térmicos del instituto.

Luego de 27 minutos, se abrió la puerta de la mufla para retirar la probeta mediante el uso de unas tenazas para forja. Inmediatamente después que la probeta fue retirada, se sumergió en agua a temperatura ambiente para realizar el enfriamiento del acero, véase la figura 117.



Figura 117: Probeta sumergida en agua a temperatura ambiente para su enfriamiento.

Fuente: Laboratorio de tratamientos térmicos del instituto.

4.3 Gráficas finales

Las gráficas obtenidas al finalizar el ensayo de temperatura se muestran en las figuras 118 para la gráfica de temperatura y 119 para la gráfica del diferencial de temperatura. En la primera se mostró como la temperatura interna del acero continuaba incrementando hasta una máxima de 210 °C, pero en el momento donde la probeta fue sumergida en el agua, la gráfica cayó en picado hasta estabilizarse a los 26.5 °C solamente transcurridos 70 segundos.

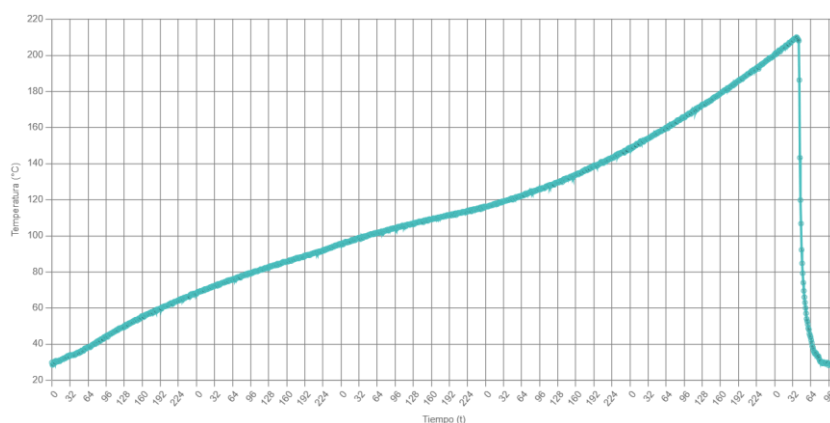


Figura 118: Gráfica de la temperatura interna de la probeta al finalizar el ensayo.

Fuente: Laboratorio de tratamientos térmicos del instituto.

De la misma forma, en la figura 119, los diferenciales de la temperatura interna de la probeta seguían un ritmo constante, es decir, los incrementos eran muy parecidos a lo largo del tiempo, hasta que el acero fue sumergido en el agua. En ese momento, el diferencial presentado llegó a ser de hasta -43 °C, es decir, la temperatura interna del acero en un segundo bajó 43 °C, pasando de 186.25 °C a 143.25 °C. Posteriormente, los diferenciales se volvieron cada vez más pequeños conforme se acercaba a la temperatura del agua.

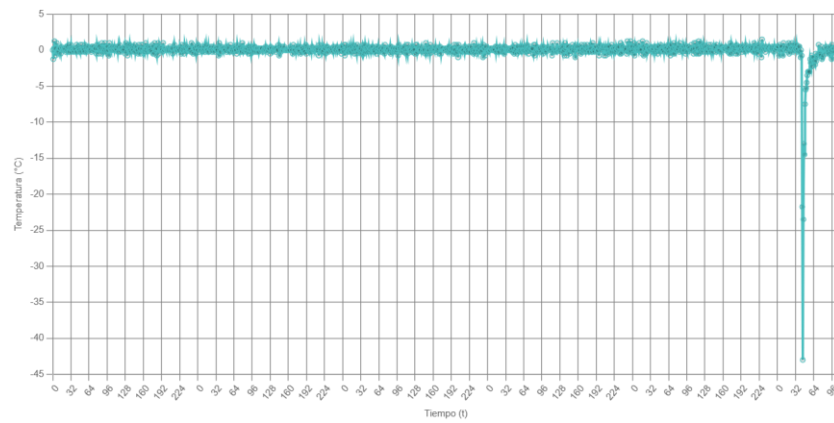


Figura 119: Gráfica del diferencial de temperatura interna de la probeta al finalizar el ensayo.

Fuente: Laboratorio de tratamientos térmicos del instituto.

5 CONCLUSIONES

Al finalizar el presente proyecto, como se muestra a lo largo de los resultados de esta, tanto la hipótesis como los objetivos se cumplieron, ya que, el dispositivo mecatrónico final, permite la medición de la temperatura interna de la probeta de acero AISI 1040 mediante el uso en conjunto de un termopar tipo K, un módulo *MAX6675* y un microcontrolador *ESP32*, el cual a su vez muestra las mediciones de temperatura en una página *web* según el tiempo de muestreo configurado por el usuario en esta misma interfaz, además estos datos son representados en dos gráficas, una para la temperatura respecto al tiempo, y otra para el diferencial de la temperatura respecto al tiempo, por último, gracias a un botón ubicado debajo de las mismas, es posible descargar una imagen para guardar el ensayo en caso de ser requerido para su posterior análisis.

Así como los objetivos de la investigación se cumplieron, surgieron nuevas ideas para la innovación del dispositivo. Una de las ideas es el manejo de errores del proyecto, ya que, a pesar de detectar si el termopar es desconectado durante el ensayo o si el servidor no se creó correctamente u otro error sucede, la única forma actual de comunicarlo al usuario es mediante un indicador *LED* de color rojo, esto es un problema, porque aunque sea comunicado que sucedió un error, no se especifica cuál, por lo que una solución viable sería la adición de una pantalla *LCD* que funja como interfaz para los errores u otra información a comunicar al usuario, de igual forma estos mensajes de errores pueden ser agregados en la interfaz *web*.

Otra recomendación para el proyecto es la opción para cambiar el tiempo de muestreo del ensayo o de permitir el agregar un tiempo de muestro especial para la etapa de enfriamiento, esto debido a que si bien, las etapas de calentamiento y mantenimiento del acero son lentas, el tiempo muestreo puede ser alto, sin embargo, cuando se trata del enfriamiento, esta etapa sucede muy rápido, por lo que un tiempo de muestreo corto sería lo ideal. Por ende, la combinación de estos dos tiempos de muestreo diferentes puede dar como resultado gráficas más legibles y de mayor calidad.

La siguiente recomendación para el proyecto sería la adaptación de la página *web* para que pueda ser vista correctamente por cualquier dispositivo, es decir, hacer la interfaz *web* dinámica para que sin importar desde qué dispositivo acceda el usuario ésta se vea correctamente.

En cuanto a las gráficas recomendaría en darles la característica de ser reajustables o permitir realizar un acercamiento para así lograr observarlas con más detalle de ser necesario.

La última recomendación para este proyecto sería en la adición de una opción para exportar los datos de las gráficas en forma de tablas, ya que, esta característica podría brindar la posibilidad de trabajar con esta información de forma exacta.

Por último, los resultados mostrados por el dispositivo fueron excelentes, esto debido a que durante su prueba en la mufla y midiendo la temperatura interna de la probeta no se detectaron interferencias en las mediciones ni tampoco fue complicado el uso del dispositivo, por lo que, será una herramienta muy útil y versátil en el laboratorio de tratamientos térmicos para que los alumnos de la universidad puedan realizar estos tratamientos a un nivel ingenieril al tener una información tan importante como lo es la temperatura interna del metal con el que se trabaja.

6 COMPETENCIAS DESARROLLADAS

Durante las diferentes etapas de la elaboración del dispositivo mecatrónico, me vi en la necesidad de superarme en cada uno de los ámbitos, ya que, si bien conocía cada una de las áreas en las que llegué a trabajar, mi deseo de realizar una herramienta útil para el laboratorio me orilló a sobrepasar mis límites.

La primera competencia que fue necesario desarrollar, era la programación y uso en general del microcontrolador *ESP32*, si bien, ya había programado en el *Arduino IDE* con anterioridad, jamás lo había utilizado para dicho microcontrolador. De la misma manera, tuve que investigar y aprender a utilizar las librerías presentadas a lo largo del proyecto y adaptarlas a las necesidades de este.

La siguiente competencia aplicada fue la elaboración de la placa de circuitos impresos mediante una fresadora *CNC*. En esta área, a pesar de en lo personal haber tenido práctica a lo largo de mi vida académica, aun así, adquirí nuevo conocimiento y experiencia en el diseño de *PCB* mediante los *softwares Proteus y FlatCAM*, como lo es particularmente en los parámetros para lograr un mejor mecanizado de las pistas y barrenados.

El diseño mecánico 3D y la impresión 3D, fueron otras competencias aplicadas en la elaboración de este trabajo, sin embargo, estas habilidades en cuestión han sido las que más me han acompañado en mi formación profesional, por lo que no supuso un reto aplicarlas correctamente.

Por otro lado, la competencia más desafiante en este proyecto fue sin duda alguna la programación de la página *web*, así como también del manejo del servidor creado mediante el microcontrolador. Fue complicado debido a mi poca experiencia en esta área, pero gracias a ello, me fue posible adquirir esta nueva habilidad que sin duda es muy útil en la actualidad.

7 FUENTES DE INFORMACIÓN

1. Monteros Vizcaíno, L. A. (2016). *Diseño y construcción de un horno mufla para la realización de tratamientos térmicos en aceros* [Tesis de pregrado, Universidad Técnica del Norte]. Recuperado de <https://repositorio.utn.edu.ec/handle/123456789/4592>
2. González Cardona, U. M. (2022). *EVALUACIÓN DEL DESEMPEÑO DEL PROCESO DE TEMPLE DE UN ACERO 4140R, VARIANDO LAS CONDICIONES DE AGITACIÓN, TIEMPO DE USO Y CONCENTRACIÓN DE NANOPARTÍCULAS DE UN ACEITE DE USO INDUSTRIAL* [Tesis de posgrado, Tecnológico Nacional de México]. Recuperado de <https://rinacional.tecnm.mx/jspui/handle/TecNM/8050>
3. Pérez Salinas, C. F. & Capus Solano, E. S. (2020). *Análisis de la resistencia al desgaste abrasivo del acero al carbono ASTM A-36 y del AISI 1045, con tratamiento superficial, para utilizar en la reparación de cucharones de retroexcavadoras* [Tesis de licenciatura, Universidad Técnica de Ambato]. Recuperado de <https://repositorio.uta.edu.ec/jspui/handle/123456789/30699>
4. Hernández Hurtado, J. D. & Laverde Castillo, L. F. (2019). *Estudio y caracterización del acero A36 tratado térmicamente como sustituto para la fabricación de tanques mezcladores de hormigón* [Tesis de licenciatura, Escuela politécnica nacional]. Recuperado de <http://bibdigital.epn.edu.ec/handle/15000/20406>
5. Saldaña Celis, D. R. (2022). *Modelado y caracterización del proceso de temple* [Tesis de licenciatura, Universidad nacional de Trujillo]. Recuperado de <https://hdl.handle.net/20.500.14414/18397>
6. Romero Hurtado, D. A. & Vergara Morales, D. (2018). *Influencia del tratamiento térmico de temple con austenización parcial, criogenia y revenido sobre la resistencia a la tensión y la*

- dureza de un acero con contenido de carbono superior al 0,5%* [Tesis de licenciatura, Universidad distrital Francisco José de Caldas]. Recuperado de <http://hdl.handle.net/11349/14177>
7. Bonilla Castañeda, S. M., Gutiérrez Quiceno, K. J. , Osorio Vélez, F. J. & Sánchez Escobar, M. V. (2020). *Módulo III. Equipos de laboratorio*. Recuperado de <https://media2.utp.edu.co/archivos/M%C3%B3dulo%20III.%20Equipos%20de%20Laboratorio.pdf>
 8. Metalúrgica Daniele. (2024). *¿Es relevante la diferencia del acero SAE 1045 y del SAE 1040 en la práctica?* [Metalúrgica Daniele blog]. Recuperado de <https://www.metalurgicadaniele.com.ar/blog/es-relevante-la-diferencia-del-acero-sae-1045-y-del-sae-1040-en-la-practica>
 9. Azorín Penalva, A. (2018). *Efecto Peltier y desarrollo de posibles aplicaciones técnicas* [Tesis de licenciatura, Universidad Politécnica de València]. Recuperado de <http://hdl.handle.net/10251/109025>
 10. Cali Flores, A. E. (2022). *DISEÑO E IMPLEMENTACIÓN DE UNA HERRAMIENTA INTERACTIVA DE INSTRUMENTACIÓN VIRTUAL PARA LAS PRÁCTICAS DEL LABORATORIO DE SENSORES Y TRANSDUCTORES RELACIONADAS CON LA VARIABLE TEMPERATURA* [Tesis de licenciatura, Escuela Politécnica Nacional]. Recuperado de <http://bibdigital.epn.edu.ec/handle/15000/23323>
 11. Nova Esquivel, E. A. (2020). *Diseño de un plan de mantenimiento preventivo para la estandarización en los rangos de medición de temperatura a sensores termopares en una línea de extrusión de PET reciclado utilizando la Norma ISO 10 012* [Tesis de maestría, Universidad de San Carlos de Guatemala]. Recuperado de <http://www.repositorio.usac.edu.gt/id/eprint/15055>
 12. Guerrero Quezada, G. I. & Arias Pozo, P. A. (2021). *Diseño y construcción de un banco didáctico de instrumentación para la calibración de los termopares tipo J, K & T, y control de*

un horno de tres etapas [Tesis de licenciatura, Escuela Superior Politécnica de Chimborazo]. Recuperado de <http://dspace.esPOCH.edu.ec/handle/123456789/15571>

13. Narváez Pilamunga, E.J. (2022). *Implementación del sistema de monitoreo y supervisión de temperaturas de dos sistemas de refrigeración del Laboratorio de Tecnología Industrial* [Tesis de licenciatura, Escuela Politécnica Nacional]. Recuperado de <http://bibdigital.epn.edu.ec/handle/15000/22839>
14. Domínguez Enríquez, J. L. (2018). *Integración y caracterización de sistemas criogénicos de 150 a 4 kelvin y diseños de monturas térmicas* [Proyecto de residencias, Tecnológico Nacional de México]. Recuperado de <http://repositoriodigital.tuxtla.tecnm.mx/xmlui/handle/123456789/3855>
15. Velázquez Villegas, A. E. & Martínez Ramírez, I. (2018). *Fabricación e instrumentación de horno con perfil elipsoidal para calibración de termopar, con aplicación a procesos de corte continuo* [Proyecto universitario, Universidad de Guanajuato]. Recuperado de <http://repositorio.ugto.mx/handle/20.500.12059/5782>
16. Gómez Gaviria, J. S. & Silva Lima, N. A. (2019). *Diseño e Implementación de un Sistema de Energía Solar Fotovoltaico para la Alimentación de Una Planta de Potabilización de Agua* [Tesis de licenciatura, Universidad de los Llanos]. Recuperado de <https://repositorio.unillanos.edu.co/handle/001/1439>
17. Grajales Eraz, Y. & Monsalve Vargas, J. F. (2019). *DISEÑO DE UN INSTRUMENTO DE ADQUISICIÓN DE TEMPERATURA A TRAVÉS DE TERMOCUPLA TIPO K Y MICROCONTROLADOR USANDO EL PROTOCOLO DE COMUNICACIÓN SPI* [Trabajo de especialización, Escuela Tecnológica Instituto Técnico Central]. Recuperado de <http://repositorio.itc.edu.co/handle/001/378>
18. Ikiss, J. (2020). *SISTEMA DE ADQUISICIÓN DE DATOS CON ESP32* [Trabajo final de grado, Universitat Politècnica de Catalunya]. Recuperado de <http://hdl.handle.net/2117/344400>

19. ESPRESSIF. (2024). *ESP32*. Recuperado de <https://www.espressif.com/en/products/socs/esp32>
20. ESPRESSIF. (2024). *ESP32-DevKitC*. Recuperado de <https://www.espressif.com/en/products/devkits/esp32-devkitc>
21. Amazon Web Services. (2022). *¿Qué es un entorno de desarrollo integrado (IDE)?*. Recuperado de <https://aws.amazon.com/es/what-is/ide/>
22. Red Hat. (2023). *¿Qué es y para qué sirve un IDE?*. Recuperado de <https://www.redhat.com/es/topics/middleware/what-is-ide>
23. Arduino. (2019). *Software de Arduino*. Recuperado de <https://arduino.cl/programacion/>
24. Peris Martínez, J. (2020). *SISTEMA DE MONITORIZACIÓN INALÁMBRICA DE TEMPERATURA MEDIANTE SENSOR DE INFRARROJOS Y MICROCONTROLADOR ESP32* [Tesis de maestría, Universitat Politècnica de València]. Recuperado de <http://hdl.handle.net/10251/153154>
25. Arduino. (2015). *Wi-Fi*. Recuperado de <https://www.arduino.cc/reference/en/libraries/wifi/>
26. Agama Chico, A. J., Saa Ayala, J. F., Baldeón Navarrete, E. G. & Bajaña Ortiz, O. A. (2023). *Servidor Web local con Autenticación y Administración de Usuarios basado en ESP32 para Viviendas Inteligentes* [Revista científica, Journal of Science and Research, 8(1), 148–166]. Recuperado a partir de <https://revistas.utb.edu.ec/index.php/sr/article/view/2852>
27. Jordà Juan, J. (2023). *Estudio, propuesta y prototipado de solución para la geolocalización y orientación en interiores para personas con discapacidad* [Proyecto de fin de grado, Universitat Politècnica de València]. Recuperado de <http://hdl.handle.net/10251/192643>
28. Garrido Castañeda, S. I., Sepúlveda Cervantes, G. & Vega Alvarado, E. (2021). *Diseño e implementación de un sistema inalámbrico de comunicación entre un ambiente virtual y un agente en el mundo real*. En Congreso Estudiantil de Inteligencia Artificial Aplicada a la

- Ingeniería y Tecnología (pp. 55-59). FES Cuautitlán, UNAM, Estado de México. Recuperado de <https://virtual.cuautitlan.unam.mx/intar/wp-content/uploads/sites/14/2022/01/Dise%C3%B1o-e-implementaci%C3%B3n-de-un-sistema-inal%C3%A1mbrico-de-comunicaci%C3%B3n-entre-un-ambiente-virtual-y-un-agente-en-el-mundo-real.pdf>
29. Beltrán Galán, J. (2024). *CONTROL DE NIVEL EN UN SISTEMA DE TANQUES INTERCONECTADOS*. Recuperado de <https://hdl.handle.net/10953.1/24155>
 30. ESPRESSIF. (2023). *Servicio mDNS*. Recuperado de <https://docs.espressif.com/projects/esp-protocols/mdns/docs/latest/en/index.html>
 31. Benoît, B. (2014). *ArduinoJson*. Recuperado de <https://github.com/bblanchon/ArduinoJson>
 32. Polidoro, P. (2023). *Vector*. Recuperado de <https://github.com/janelia-arduino/Vector>
 33. Arduino. (2024). *SPI*. Recuperado de <https://www.arduino.cc/reference/en/language/functions/communication/spi/>
 34. Microsoft. (2024). *Visual Studio: IDE y Editor de código para desarrolladores de software y Teams*. Recuperado de <https://visualstudio.microsoft.com/es/#vscode-section>
 35. Visual Studio Code. (2015). *Code faster with AI*. Recuperado de <https://code.visualstudio.com/>
 36. Luján Castillo, J. D. (2016). *HTML5, CSS Y JAVASCRIPT Crea tu web y apps con el estándar de desarrollo*. RC Libros. Recuperado de https://rclibros.es/wp-content/uploads/2016/01/capitulo_9788494345098.pdf
 37. Amazon Web Services. (2022). *¿Qué es JavaScript (JS)?*. Recuperado de <https://aws.amazon.com/es/what-is/javascript/>
 38. Luna, F. O. (2019). *JavaScript: aprende a programar en el lenguaje de la Web*. 1ª ed. Ciudad Autónoma de Buenos Aires: Six Ediciones. Recuperado de https://books.google.es/books?hl=es&lr=lang_es&id=SqikDwAAQBAJ&oi=fnd&pg=PA4&dq=ht

ml+css+y+javascript&ots=pA7fZ-

9nEv&sig=Ksg6T13HqDYiXJJROILOmH8IXpA#v=onepage&q=html%20css%20y%20javascrip
t&f=false

39. Char.js. (2024). *Chart.js*. Recuperado de <https://www.chartjs.org/docs/latest/>
40. WC3. (2015). *Es/CSS*. Recuperado de <https://www.w3.org/wiki/Es/CSS>
41. IONOS. (2021). *¿Qué es CSS? Definición y aplicación*. Recuperado de <https://www.ionos.es/digitalguide/paginas-web/disenio-web/que-es-css/#c358784>
42. Peterson, Z. (2023). *¿Qué es un PCB o Placa de Circuito Impreso?*. Recuperado de <https://resources.altium.com/es/p/what-is-a-pcb>
43. Morales, R. A. & Solis Salinas, R. J. (2018). *Desarrollo de una metodología para la elaboración de tarjetas PCB de forma óptima*. Universidad Nacional de Ingeniería. Recuperado de <https://core.ac.uk/download/pdf/250145125.pdf>
44. Proteus. (2018). *Proteus PCB Design*. Recuperado de https://www.labcenter.com/proteus_pcb/
45. FlatCAM. (2014). *FlatCAM: Free and Open-source PCB CAM*. Recuperado de <http://flatcam.org/>
46. Dassault Systèmes. (2022). *Mecanizado CNC*. Recuperado de <https://www.3ds.com/es/make/guide/process/cnc-machining>
47. Salvador, J. (2020). *Máquinas CNC: Todo lo que necesitas saber*. Recuperado de <https://www.stanser.com/como-funciona-una-maquina-cnc-maquinas-cnc/>
48. Boyfre. (2023). *Mecanizado CNC – ¿Qué es y cómo funciona?*. Recuperado de <https://www.boyfre.com/mecanizado-cnc-que-es-y-como-funciona/>

49. Autodesk. (2019). *PROGRAMACIÓN DE CNC*. Recuperado de <https://www.autodesk.com/mx/solutions/cnc-programming>
50. López Mera, G. H. (2022). *Manual de usuario de Router CNC*. Universidad Nacional Autónoma de México. Recuperado de http://olimpia.cuautitlan2.unam.mx/pagina_ingenieria/industrial/prac/prac/MANUAL_USUARIO_ROUTER_CNC%2021022023.pdf
51. Denvi. (2023). *Candle*. Recuperado de <https://github.com/Denvi/Candle?search=1>
52. Siatras, N. (2023). *GRBL*. Recuperado de <https://github.com/gnea/grbl/wiki>
53. Dassault Systèmes. (2022). *Impresión 3D*. Recuperado de <https://www.3ds.com/es/make/guide/process/3d-printing>
54. Prusa. (2021). *Tipos de impresoras y sus diferencias*. Recuperado de https://help.prusa3d.com/es/article/tipos-de-impresoras-y-sus-diferencias-_112464
55. UltiMaker. (2019). *UltiMaker Cura 5.8*. Recuperado de <https://ultimaker.com/es/software/ultimaker-cura/>
56. SolidBi. (2018). *SOLIDWORKS. Qué es y para qué sirve*. Recuperado de <https://solid-bi.es/solidworks/>
57. Dassault Systèmes. (2018). *INTRODUCCIÓN A SOLIDWORKS*. Recuperado de https://my.solidworks.com/solidworks/guide/SOLIDWORKS_Introduction_ES.pdf
58. Ulrich, K., & Eppinger, S. (2013). *Diseño y desarrollo de productos* (5ª edición ed.). México, D.F.: Mc Graw Hill.

8 ANEXOS

8.1 Instalación de la extensión *Live Server* en el *IDE* de *Visual Studio Code*.

Es una extensión que permite la creación de un servidor local para montar la página *web* en el mismo.

Primero se tiene que dirigir al apartado de extensiones del *IDE* presionando sobre el ícono de la figura 120 o mediante el atajo mostrado en la misma figura.

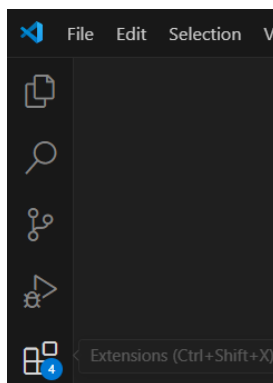


Figura 120: Ubicación del ícono de extensiones del *IDE* de *Visual Studio Code*.

Fuente: Elaboración propia.

Después, en la caja de texto debe colocar el nombre de la extensión (*Live Server*) y pulsar en la extensión deseada (figura 121).

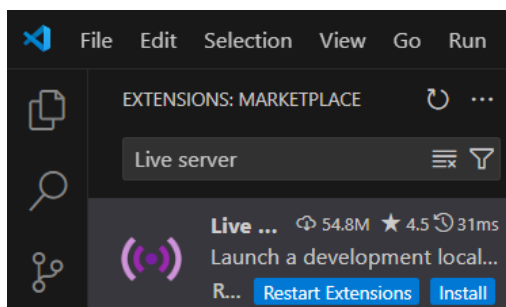


Figura 121: Resultado de la búsqueda de la extensión *Live Server*.

Fuente: Elaboración propia.

Por último, solo debe presionar el botón con la leyenda “Install” para proceder con la instalación (figura 122).

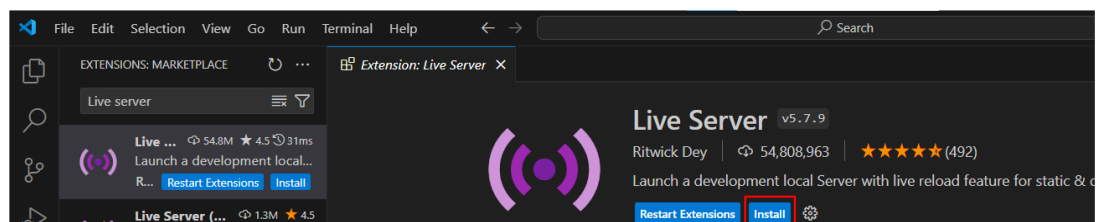


Figura 122: Botón de instalación de la extensión *Live Server*.

Fuente: Elaboración propia.

En algunas ocasiones la instalación requiere el reinicio del *IDE* para permitir el funcionamiento de la extensión, si el *software* lo solicita se debe de proceder con el reinicio.

8.2 Código *HTML*

El código *HTML* se puede descargar desde el repositorio https://github.com/Felipe-Rodriguez/Esp32_TermoparkK. De igual manera se incluye el código en este anexo.

```
<!DOCTYPE html>

<html lang="es">

<head>

  <meta charset="utf-8">

  <title>Termopark</title>

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <link rel="stylesheet" href="estilo.css">

  <script type="text/javascript" src="chart.min.js"></script>
```

```

<script type="text/javascript" src="graficas.js"></script>

</head>

<body style="width: fit-content;">

<table width="100%" align="center" border="1">

  <tr id="titulo">

    <td colspan="5">

      <h1>WebServer ESP32 para Termopark</h1>

    </td>

  </tr>

  <tr>

    <td style="width: 20%;">

      <div>

        <p style="color: #d8d8d8; margin: 1%;">

          Cantidad de termopares

        </p>

        <select id="cantidad" style="margin: 1%;">

          <option value="0">Seleccione...</option>

          <option value="1">1</option>

          <option value="2">2</option>

```

```

        </select>

    </div>

</td>

<td style="width: 20%;">

    <div>

        <p style="color: #d8d8d8; margin: 1%;">

            Tiempo de muestreo (segundos)

        </p>

        <input id="muestreo" type="text" inputmode="numeric" pattern="\d*"

            oninput="this.value = this.value.replace(/[^0-9]/g, '')"/>

    </div>

</td>

<td style="width: 20%;">

    <button class="boton" id="empezar">Empezar</button>

</td>

<td style="width: 20%;">

    <button class="boton" id="reinicio">Reinicio</button>

</td>

<td style="width: 20%;">

```

```

                <button class="boton" style="background-color: rgb(205, 0, 0);"
id="finalizar">Finalizar</button>

            </td>

        </tr>

    </table>

    <div id="contenido">

        <div style="text-align: center;">

            <h2>Selecciona la cantidad de termopares</h2>

        </div>

    </div>

</body>

</html>

```

8.3 Código CSS

El código CSS se puede descargar desde el repositorio https://github.com/Felipe-Rodriguez/Esp32_Termopark. De igual manera se incluye el código en este anexo.

```

body {

    background-color: #424242;

}

```

```

h1, h2, h3 {

```

```
text-align: center;

background-color: #474747;

color: #ffffff;

display: inline-block;

padding: 1%;

}
```

```
td {

text-align: center;

}
```

```
select {

background-color: #aeaeae;

font-size: medium;

}
```

```
input {

background-color: #aeaeae;

font-size: medium;

width: 70px;
```

```
    text-align: center;

}

.boton {

    border: none;

    color: black;

    font-size: medium;

    padding: 14px 28px;

    cursor: pointer;

    border-radius: 5px;

    background-color: #aeaeae;

}

.boton:hover {

    background-color: #7e7e7e;

}

.boton:active {

    transform: scale(0.9);

}

canvas {
```

```
width: 100%;

height: auto;

max-width: 100%;

margin: auto;

}
```

8.4 Descarga de librería *Chart.js*

Para realizar la descarga de la librería *Chart.js* es necesario dirigirse a la siguiente página <https://cdnjs.com/libraries/Chart.js/3.8.0>, dentro de esta se puede descargar la versión que prefiera, en el presente proyecto se utilizó la versión 3.8.0.

Después, para generar un documento con la librería y poder utilizarla sin conexión a internet, se debe de copiar la dirección *URL* de la figura 123, y abrirla en una pestaña del buscador, luego se copia todo el contenido de esta y se pega en un archivo nuevo llamado “char.min.js” (debe ser un archivo tipo .js).



Figura 123: Dirección *URL* de la librería *Char.js*.

Fuente: Elaboración propia.

8.5 Código *JavaScript*

El código *JavaScript* se puede descargar desde el repositorio https://github.com/Felipe-Rodriguez/Esp32_Termopark. De igual manera se incluye el código en este anexo.

```
document.addEventListener('DOMContentLoaded', function(){
```

```

//Arreglos de tiempo, temperatura y diferenciales de los termopares

let tTmk1, tempTmk1, dfTmk1, tDfTmk1 = [];

let tTmk2, tempTmk2, dfTmk2, tDfTmk2 = [];

//Variables para las gráficas

let cnfgTmk1, cnfgDfTmk1 = "";

let cnfgTmk2, cnfgDfTmk2 = "";

let cnfgCompTemp, cnfgCompDf = "";

let chartTmk1, chartDfTmk1, chartTmk2, chartDfTmk2, chartCompTemp, chartCompDf = "";

//Bandera para identificar el uso de 1 termopar o de 2

let bFlag = false;

//Variable para establecer la conexión WebSocket con el ESP32 mediante el puerto 81

let socketConnection = new WebSocket('ws://' + location.hostname + ':81/');

//Variables de configuración de la página

const cantidad = document.getElementById('cantidad');

const muestreo = document.getElementById('muestreo');

const contenido = document.getElementById('contenido');

//Variables para trabajar con los botones

const botonEmpezar = document.getElementById("empezar");

```

```

const botonReinicio = document.getElementById("reinicio");

const botonFinalizar = document.getElementById("finalizar");


//Evento para cambiar la página según cuántos termopares se usen

cantidad.addEventListener('change', function() {

    let cantidadSeleccionada = cantidad.value;

    actualizarContenido(cantidadSeleccionada);

});


//Función encargada de actualizar la página con la cantidad de termopares

function actualizarContenido(valor){

    if (valor == '2') {

        bFlag = true;

        contenido.innerHTML = `

            <table width="100%" align="center" border="1">

                <tr>

                    <td id="Tmk1" width="100%">

                        <h2 style="margin: 0;">Termopark 1</h2><br>

                        <h3>Gráfica de temperatura</h3><br>

                        <canvas id="graficaTmk1"></canvas><br>

```

<button style="margin: 1%;" class="boton"

id="graficaTmk1">Descargar</button>

</td>

</tr>

<tr>

<td width="100%">

<h3>Gráfica del diferencial de temperatura</h3>

<canvas id="graficaDfTmk1"></canvas>

<button style="margin: 1%;" class="boton"

id="graficaDfTmk1">Descargar</button>

</td>

</tr>

<tr>

<td id="Tmk2" width="100%">

<h2 style="margin: 0;">Termopark 2</h2>

<h3>Gráfica de temperatura</h3>

<canvas id="graficaTmk2"></canvas>

<button style="margin: 1%;" class="boton"

id="graficaTmk2">Descargar</button>

</td>

</tr>

```

<tr>

<td width="100%">

<h3>Gráfica del diferencial de temperatura</h3>

<canvas id="graficaDfTmk2"></canvas><br>

<button style="margin: 1%;" class="boton"
id="graficaDfTmk2">Descargar</button>

</td>

</tr>

<tr>

<td colspan="6">

<h2>Comparación de temperaturas</h2><br>

<canvas id="graficaCompTemp"></canvas><br>

<button style="margin: 1%;" class="boton"
id="graficaCompTemp">Descargar</button><br>

<h2>Comparación de diferenciales de temperaturas</h2><br>

<canvas id="graficaCompDf"></canvas><br>

<button style="margin: 1%;" class="boton"
id="graficaCompDf">Descargar</button>

</td>

</tr>

</table>

```

```

        `;

        asignarEventosDeDescarga();

    } else if (valor == '1') {

        bFlag = false;

        contenido.innerHTML = `

            <table width="100%" align="center" border="1">

                <tr>

                    <td id="Tmk1" width="100%">

                        <h3>Gráfica de temperatura</h3><br>

                        <canvas id="graficaTmk1"></canvas><br>

                        <button style="margin: 1%;" class="boton"
id="graficaTmk1">Descargar</button>

                    </td>

                </tr>

                <tr>

                    <td width="100%">

                        <h3>Gráfica del diferencial de temperatura</h3><br>

                        <canvas id="graficaDfTmk1"></canvas><br>

                        <button style="margin: 1%;" class="boton"
id="graficaDfTmk1">Descargar</button>

```

```

        </td>

    </tr>

</table>

`;

asignarEventosDeDescarga();

} else {

    bFlag = false;

    contenido.innerHTML = `

        <div style="text-align: center;">

            <h2>Selecciona la cantidad de termopares</h2>

        </div>

    `;

}

}

```

//Función para agregar el evento de descarga de cada gráfica a su botón correspondiente

```

function asignarEventosDeDescarga() {

    document.querySelectorAll('.boton[id]').forEach(button => {

        button.addEventListener('click', function() {

            const canvasId = this.getAttribute('id');

```

```

const canvas = document.getElementById(canvasId);

const link = document.createElement('a');

link.href = canvas.toDataURL('image/png');

link.download = canvasId + '.png';

link.click();

});

});

}

//Evento del botón de empezar para iniciar el estudio de temperatura

botonEmpezar.addEventListener('click', function() {

    if (cantidad.value !== '0' && muestreo.value !== 0) {

        deshabilitarControl();

        deshabilitarCofig();

        //Mandar los datos de configuración al esp32

        const jsonData = JSON.stringify({

            indicador: "configWeb",

            cantidad: cantidad.value,

            muestreo: muestreo.value,

            botonEstados: false,

```

```

        bEmpezar: true

    });

    socketConnection.send(jsonData);

    obtenerGraficas();

} else {

    alert("Llena la información requerida para poder empezar");

}

});

```

//Evento del botón de reinicio para volver al estado inicial de la página web

```

botonReinicio.addEventListener('click', function() {

    tTmk1, tempTmk1, dfTmk1, tDfTmk1 = [];

    tTmk2, tempTmk2, dfTmk2, tDfTmk2 = [];

    cnfgTmk1, cnfgDfTmk1, cnfgTmk2, cnfgDfTmk2 = "";

    cnfgCompTemp, cnfgCompDf = "";

    botonEmpezar.innerHTML = "Empezar";

    destruirGraficos();

    habilitarConfig();

```

```

cantidad.value = "0";

muestreo.value = 0;

bFlag = false;

//Reiniciar los datos de configuración al esp32

const jsonData = JSON.stringify({

    indicador: "resetWeb",

    cantidad: "0",

    muestreo: "0",

    botonEstados: true

});

socketConnection.send(jsonData);

actualizarContenido(cantidad.value);

});

//Evento del botón Finalizar para detener el estudio de temperatura

botonFinalizar.addEventListener('click', function() {

    let bFin = confirm("¿Estás seguro de querer finalizar?");

    if (bFin) {

        botonEmpezar.innerHTML = "Retomar";

        habilitarControl();

```

```
const jsonData = JSON.stringify({

    indicador: "finalizar",

    bEmpezar: false,

    botonEstados: true

});

socketConnection.send(jsonData);

} else {

    console.log("No se finalizó");

}

});
```

//Función para crear todas las gráficas

```
function obtenerGráficas() {

    cnfgTmk1 = crearGrafico(tTmk1, tempTmk1);

    cnfgDfTmk1 = crearGrafico(tDfTmk1, dfTmk1);

    if (bFlag) {

        cnfgTmk2 = crearGrafico(tTmk2, tempTmk2);

        cnfgDfTmk2 = crearGrafico(tDfTmk2, dfTmk2);
```

```

        cnfgCompTemp = crearGraficoComp(tTmk1, tempTmk1, tempTmk2);

        cnfgCompDf = crearGraficoComp(tDfTmk1, dfTmk1, dfTmk2);

    }

    graficar();

}

```

//Función para graficar

```

function graficar() {

    const graphTmk1 = document.querySelector("#graficaTmk1");

    chartTmk1 = new Chart(graphTmk1, cnfgTmk1);

    const graphDfTmk1 = document.querySelector("#graficaDfTmk1");

    chartDfTmk1 = new Chart(graphDfTmk1, cnfgDfTmk1);

    if (bFlag) {

        const graphTmk2 = document.querySelector("#graficaTmk2");

        chartTmk2 = new Chart(graphTmk2, cnfgTmk2);

        const graphDfTmk2 = document.querySelector("#graficaDfTmk2");

        chartDfTmk2 = new Chart(graphDfTmk2, cnfgDfTmk2);
    }
}

```

```

const graphCompTemp = document.querySelector("#graficaCompTemp")

chartCompTemp = new Chart(graphCompTemp, cnfgCompTemp);

const graphCompDf = document.querySelector("#graficaCompDf");

chartCompDf = new Chart(graphCompDf, cnfgCompDf);

}

}

```

//Función reusable para crear gráficas relación temperatura-tiempo

```
function crearGrafico(tiempo, temperatura){
```

```
    const datos = {
```

```
        labels: tiempo,
```

```
        datasets: [{
```

```
            data: temperatura,
```

```
            borderColor: 'rgb(75, 192, 192)',
```

```
        ]}
```

```
    };
```

```
    const config = {
```

```
        type: 'line',
```

```
        data: datos,
```

```

options: {

  scales: {

    x: {

      title: {

        display: true,

        text: 'Tiempo (t)'

      },

      grid: {

        color: 'rgba(127.5, 127.5, 127.5, 1)',

      },

    },

    y: {

      title: {

        display: true,

        text: 'Temperatura (°C)'

      },

      grid: {

        color: function(e) {

          if (e.tick.value === 0) {

            return 'rgba(255, 255, 255, 0.5)';

```

```

        }

        else {

            return 'rgba(127.5, 127.5, 127.5, 1)';

        }

    }

}

},

plugins: {

    legend: {

        display: false

    }

}

}

};

return config;

}

```

//Función reusable para crear gráficas comparativas entre 2 valores y el tiempo

```
function crearGraficoComp(tiempo, tempTmk1, tempTmk2){
```

```
const dataset1 = {  
  
  label: "Termopar 1",  
  
  data: tempTmk1,  
  
  borderColor: 'rgba(248, 37, 37, 0.8)',  
  
  fill: false,  
  
  tension: 0.1  
  
};
```

```
const dataset2 = {  
  
  label: "Termopar 2",  
  
  data: tempTmk2,  
  
  borderColor: 'rgba(69, 248, 84, 0.8)',  
  
  fill: false,  
  
  tension: 0.1  
  
};
```

```
const datos = {  
  
  labels: tiempo,  
  
  datasets: [dataset1, dataset2]  
  
};
```

```

const config = {

  type: 'line',

  data: datos,

  options: {

    scales: {

      x: {

        title: {

          display: true,

          text: 'Tiempo (t)'

        },

        grid: {

          color: 'rgba(127.5, 127.5, 127.5, 1)',

        }

      },

      y: {

        title: {

          display: true,

          text: 'Temperatura (°C)'

        },

      },

```

```

grid: {

    lineWidth: function(e) {

        if (e.tick.value === 0) {

            return '3';

        } else {

            return '1';

        }

    },

    color: function(e) {

        if (e.tick.value === 0) {

            return 'rgba(69, 140, 248, 0.8)';

        } else {

            return 'rgba(127.5, 127.5, 127.5, 1)';

        }

    }

}

};

```

```

    return config;

}

//Función para deshabilitar los botones de control de la página web

function deshabilitarControl() {

    botonEmpezar.disabled = true;

    botonEmpezar.style.backgroundColor = "#606060";


    botonReinicio.disabled = true;

    botonReinicio.style.backgroundColor = "#606060";


    botonFinalizar.disabled = false;

    botonFinalizar.style.backgroundColor = "rgb(205, 0, 0)";

}

//Función para habilitar los botones de control de la página web

function habilitarControl() {

    botonEmpezar.disabled = false;

    botonEmpezar.style.backgroundColor = "#aeaeae";

```

```

    botonReinicio.disabled = false;

    botonReinicio.style.backgroundColor = "#aeaeae";


    botonFinalizar.disabled = true;

    botonFinalizar.style.backgroundColor = "rgb(155, 0, 0)";

}


//Función para deshabilitar los controles de configuración de la página web

function deshabilitarCofig() {

    cantidad.disabled = true;

    cantidad.style.backgroundColor = "#606060";

    cantidad.style.color = "#d8d8d8";


    muestreo.disabled = true;

    muestreo.style.backgroundColor = "#606060";

    muestreo.style.color = "#d8d8d8";

}


//Función para habilitar los controles de configuración de la página web

function habilitarConfig() {

```

```

cantidad.disabled = false;

cantidad.style.backgroundColor = "#aeaeae";

cantidad.style.color = "black";


muestreo.disabled = false;

muestreo.style.backgroundColor = "#aeaeae";

muestreo.style.color = "black";

}


//Evento para recibir la información del ESP32

socketConnection.onmessage = function(event)

{

    var data = JSON.parse(event.data);


    switch (data.indicador) {

        case "inicio":

            cantidad.value = data.cantidadTermopares;

            muestreo.value = data.tiempoMuestreo;

            actualizarContenido(data.cantidadTermopares);

            (data.botonEstados == false) ? deshabilitarControl() : habilitarControl();

```

```
break;
```

```
case "medicion":
```

```
    tTmk1 = data.tTmk1;
```

```
    tempTmk1 = data.tempTmk1;
```

```
    tDfTmk1 = data.tTmk1;
```

```
    dfTmk1 = data.dfTmk1;
```

```
    deshabilitarCofig();
```

```
    destruirGraficos();
```

```
    obtenerGraficas();
```

```
    botonEmpezar.innerHTML = "Retomar";
```

```
break;
```

```
case "medicion2":
```

```
    tTmk1 = data.tTmk1;
```

```
    tempTmk1 = data.tempTmk1;
```

```
    tDfTmk1 = data.tTmk1;
```

```
    dfTmk1 = data.dfTmk1;
```

```
    tTmk2 = data.tTmk1;
```

```
    tempTmk2 = data.tempTmk2;
```

```

        tDfTmk2 = data.tTmk1;

        dfTmk2 = data.dfTmk2;

        deshabilitarCofig();

        destruirGraficos();

        obtenerGraficas();

        botonEmpezar.innerHTML = "Retomar";

        break;
    }
}

```

//Función para destruir las gráficas que existan

```

function destruirGraficos() {

    if (chartTmk1) chartTmk1.destroy();

    if (chartTmk2) chartTmk2.destroy();

    if (chartDfTmk1) chartDfTmk1.destroy();

    if (chartDfTmk2) chartDfTmk2.destroy();

    if (chartCompTemp) chartCompTemp.destroy();

    if (chartCompDf) chartCompDf.destroy();

}

```

});

8.6 Soporte para ESP32 en Arduino IDE

Para realizar la instalación se debe dirigir a la página oficial de *ESPRESSIF* <https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html> en la cual se copiará el URL mostrado en la figura 124.

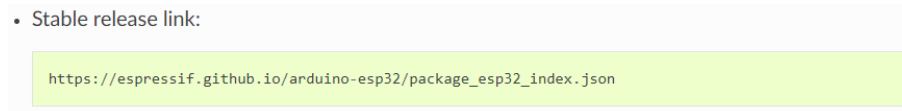


Figura 124: Dirección URL para el gestor de tarjetas de ESP32 para Arduino.

Fuente: Elaboración propia.

Ya con la dirección copiada, se debe de abrir la interfaz de *Arduino* (puede ser cualquier versión, si y solo si, es igual o mayor a la 1.8) y dentro de esta, abrir la ventana de preferencias que se encuentra dentro de Archivo > Preferencias (ver figura 125).

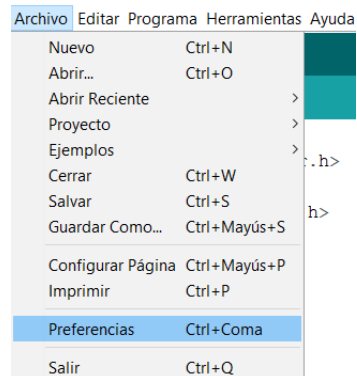


Figura 125: Menú desplegable de opción “Archivo” de Arduino IDE.

Fuente: Elaboración propia.

Dentro de preferencias, debe de pegar el enlace copiado en la opción “Gestor de URLs Adicionales de Tarjetas”, como se muestra en la figura 126.

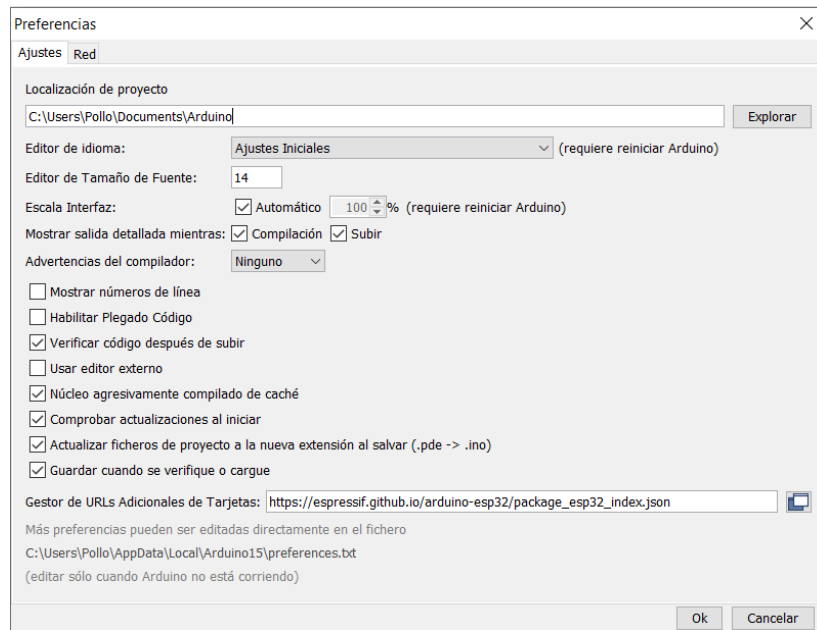


Figura 126: Ventana de preferencias de *Arduino IDE* con el gestor de tarjetas ESP32.

Fuente: Elaboración propia.

Una vez con la dirección correctamente pegada, debe de pulsarse el botón con la leyenda “Ok” y dirigirse a la opción “Gestor de tarjetas” ubicada en Herramientas > Placa: > Gestor de tarjetas (figura 127)

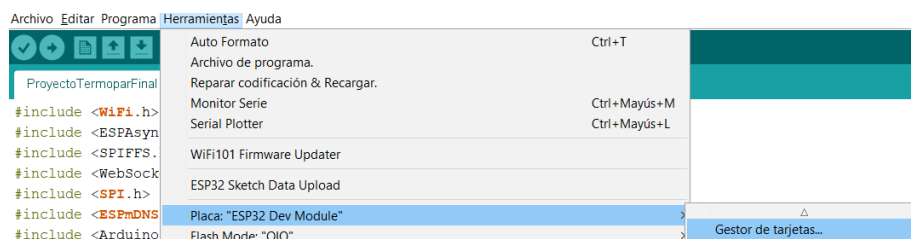


Figura 127: Ubicación del gestor de tarjetas de *Arduino IDE*.

Fuente: Elaboración propia.

Con el gestor de tarjetas abierto, se busca el microcontrolador *ESP32* y se instala el paquete elaborado por *Espressif Systems* (ver figura 128). La versión para instalar puede ser cualquiera, pero la más actual cuando se empezó con el desarrollo de esta aplicación era la 3.0.1.

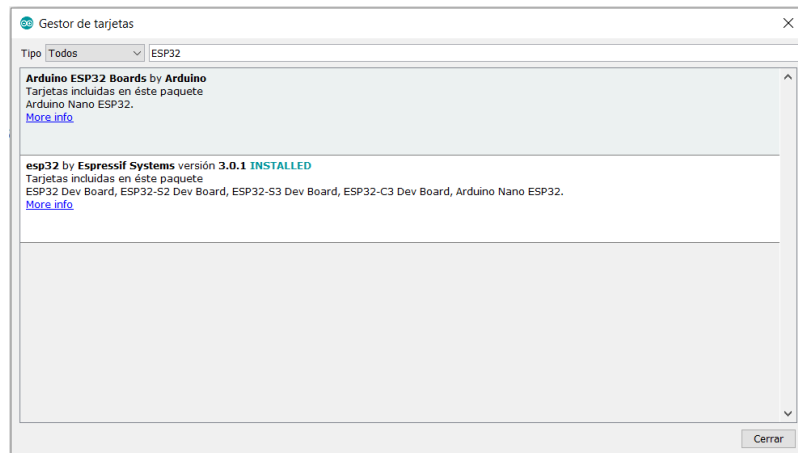


Figura 128: Instalación del paquete de tarjetas *ESP32* por *Espressif Systems*.

Fuente: Elaboración propia.

Finalmente, se dirige a Herramientas > Placa:, y se selecciona el *DevKit* o la versión de *ESP32* con la que se cuente.

8.7 Instalación de librerías de *Arduino*

8.7.1 Librería *Wifi.h*

Debe dirigirse a la opción Programa > Incluir Librería > Gestionar Librerías, después debe buscarse la librería en cuestión y realizar la instalación de esta, véase en figura 129.

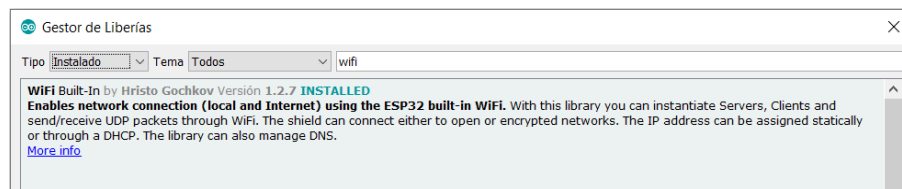


Figura 129: Instalación de librería *WiFi.h*.

Fuente: Elaboración propia.

8.7.2 Librería ESPAsyncWebServer.h

Para realizar la instalación de esta librería, cabe aclarar que no son propias de *Arduino* sino desarrolladas por la comunidad, primero es necesario dirigirse al siguiente repositorio en *GitHub* <https://github.com/me-no-dev/ESPAsyncWebServer> y descargar el archivo *ZIP* de la rama (figura 130).

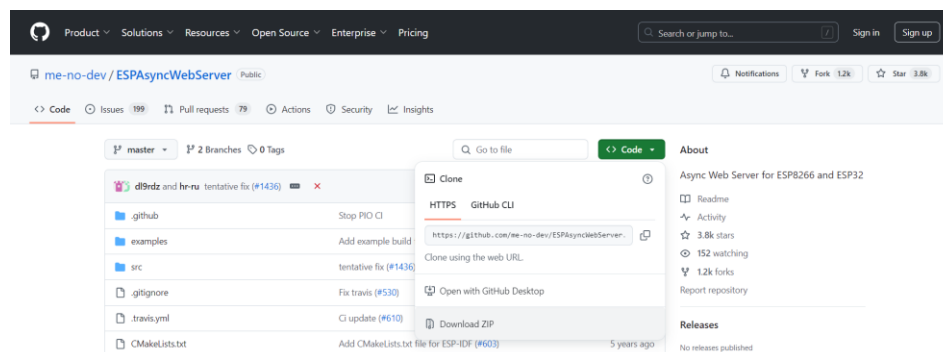


Figura 130: Repositorio *GitHub* para la descarga de la librería *ESPAsyncWebServer*.

Fuente: Elaboración propia.

Una vez con la librería descargada, es necesario presionar en el hipervínculo “*AsyncTCP*” (figura 131), debido a que es otra librería la cual es necesaria para el correcto funcionamiento de la primera, es decir, es una dependencia.

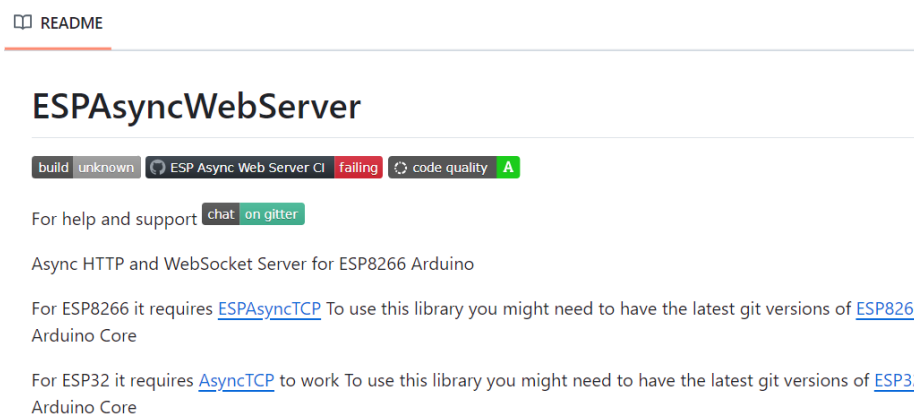


Figura 131: Hipervínculos de las dependencias de la librería según la tarjeta usada.

Fuente: Elaboración propia.

Una vez en este segundo repositorio, se realiza la descarga del archivo *ZIP* de la misma manera que para la librería *ESPAsyncWebServer*.

Con las dos librerías ya descargadas, lo siguiente es dirigirse a la ruta Programa > Incluir Librería > Añadir biblioteca .ZIP ... en *Arduino IDE* (la ruta se muestra en la figura 132).

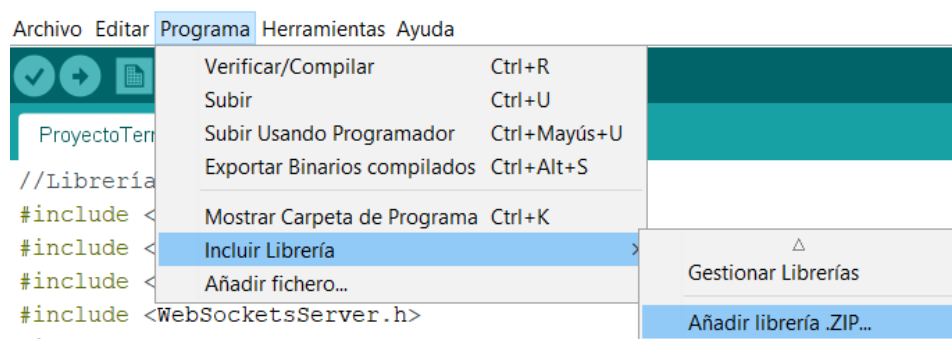


Figura 132: Ubicación de la opción para la instalación manual de librerías.

Fuente: Elaboración propia.

Una vez con la ventana de selección abierta, se debe de abrir un archivo *ZIP* a la vez y validar que aparezca el mensaje de que se instaló correctamente (figura 133).

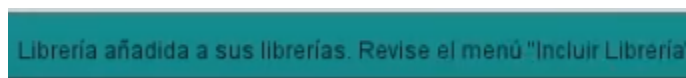


Figura 133: Mensaje para una instalación correcta de la librería.

Fuente: Elaboración propia.

8.7.3 Librería SPIFFS.h

La instalación de esta librería se realiza desde el gestor de librerías de *Arduino IDE*, por lo que debe dirigirse a la opción Programa > Incluir Librería > Gestionar Librerías, después debe buscarse la librería en cuestión y realizar la instalación de esta, véase en figura 134.

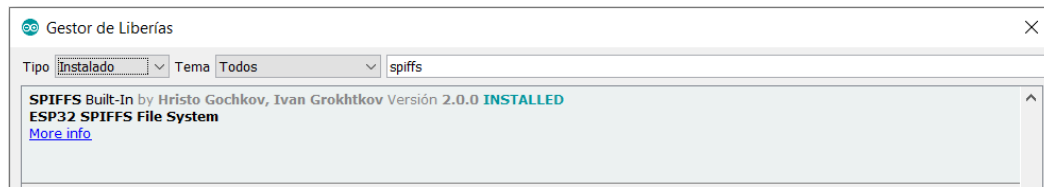


Figura 134: Instalación de librería *SPIFFS.h*.

Fuente: Elaboración propia.

En adición a lo anterior, es necesaria la instalación de un plugin para *Arduino IDE* el cual permitirá empaquetar una carpeta con el nombre “data” en una imagen del sistema de archivos *SPIFFS* y cargarla a la memoria *flash* del *ESP32*. Para ello, se descargará el componente *ZIP* de la rama del siguiente repositorio de *GitHub* <https://github.com/me-no-dev/arduino-esp32fs-plugin>.

Con el componente descargado, se debe de dirigir a la locación del proyecto (puede visualizarse en la opción de preferencias el *Arduino IDE*) y crear una carpeta con el nombre “tools” (en caso de ya tener una carpeta con ese nombre, se omite ese paso).

Dentro de la carpeta *tools* se debe de descomprimir el archivo *ZIP* descargado anteriormente. Al abrir el *Arduino IDE* y crear un proyecto con un *DevKit* del *ESP32* seleccionado, en el apartado Herramientas, debe de aparecer una opción llamada “*ESP32 Sketch Data Upload*” indicando que la instalación del *plugin* se realizó correctamente (figura 135).

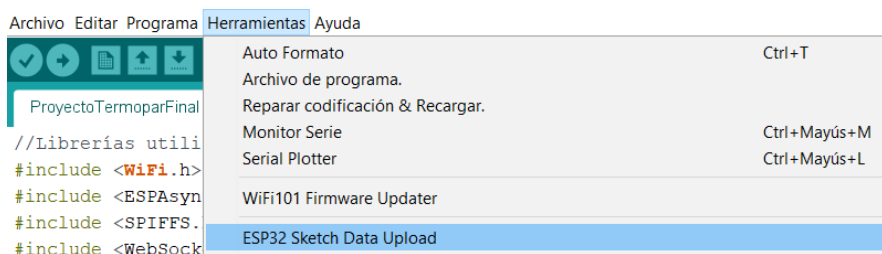


Figura 135: Ubicación de la opción “*ESP32 Sketch Data Upload*”.

Fuente: Elaboración propia.

8.7.4 Librería WebSocketsServer.h

Para la instalación de esta librería, como se ha realizado anteriormente, se descargará del siguiente repositorio de *GitHub* <https://github.com/Links2004/arduinoWebSockets>. Después de realizar la descarga del archivo *ZIP*, se debe de dirigir a la ruta Programa > Incluir Librería > Añadir biblioteca .ZIP ... y abrir el componente descargado. Al final solo se debe de validar que la librería se instaló correctamente con el mensaje de la figura 133.

8.7.5 Librería SPI.h

La instalación de esta librería se realiza desde el gestor de librerías de *Arduino IDE*, por lo que debe dirigirse a la opción Programa > Incluir Librería > Gestionar Librerías, después debe buscarse la librería en cuestión y realizar la instalación de esta, véase en figura 136.

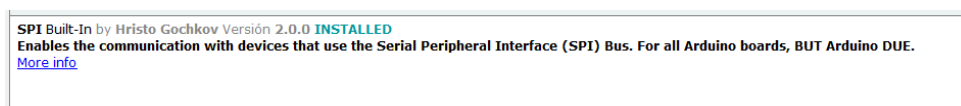


Figura 136: Instalación de librería *SPI.h*.

Fuente: Elaboración propia.

8.7.6 Librería ESPmDNS.h

La instalación de esta librería se realiza desde el gestor de librerías de *Arduino IDE*, por lo que debe dirigirse a la opción Programa > Incluir Librería > Gestionar Librerías, después debe buscarse la librería en cuestión y realizar la instalación de esta, véase en figura 137.

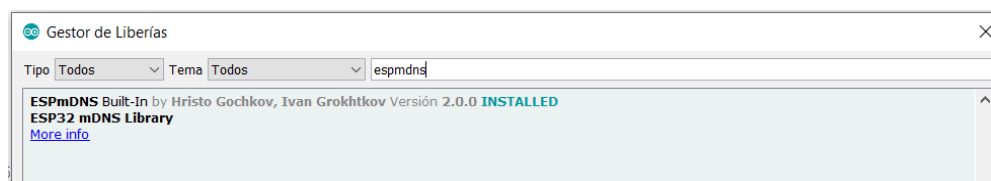


Figura 137: Instalación de librería *ESPmDNS.h*.

Fuente: Elaboración propia.

8.7.7 Librería ArduinoJson.h

La instalación de esta librería se realiza desde el gestor de librerías de *Arduino IDE*, por lo que debe dirigirse a la opción Programa > Incluir Librería > Gestionar Librerías, después debe buscarse la librería en cuestión y realizar la instalación de esta, véase en figura 138.

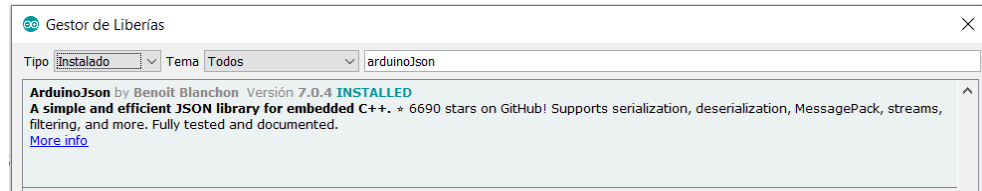


Figura 138: Instalación de librería *ArduinoJson.h*.

Fuente: Elaboración propia.

8.8 Código del ESP32

El código para el ESP32 se puede descargar desde el repositorio https://github.com/Felipe-Rodriguez/Esp32_Termopark. De igual manera se incluye el código en este anexo.

```
//Librerías utilizadas

#include <WiFi.h>

#include <ESPAsyncWebServer.h>

#include <SPIFFS.h>

#include <WebSocketsServer.h>

#include <SPI.h>

#include <ESPmDNS.h>

#include <ArduinoJson.h>

#include <vector>
```

```
// Definir pines de Chip Select para cada MAX6675
```

```
#define MAX6675_CS1 5
```

```
#define MAX6675_CS2 4
```

```
//Definir pin de los indicadores
```

```
const uint8_t ledPinEncendido = 2;
```

```
const uint8_t ledPinCensando = 16;
```

```
const uint8_t ledPinError = 17;
```

```
//Variable para iniciar el ciclo del estudio
```

```
bool bEmpezar = false;
```

```
//Variables para los arreglos
```

```
uint8_t tiempo = 0;
```

```
uint8_t tiempoAnterior = 0;
```

```
float tempAnterior = 0;
```

```
float tempAnterior2 = 0;
```

```
uint8_t deltaT = 0;
```

```
float deltaTemp = 0;
```

```
float deltaTemp2 = 0;
```

```
float derivada = 0;
```

```
float derivada2 = 0;
```

```
//Variable para saber si hay o no mediciones
```

```
uint8_t bMediciones = 0;
```

```
//Variables de configuración de la página Web
```

```
bool botonEstados = true;
```

```
uint8_t cantidadTermopares = 0;
```

```
uint16_t tiempoMuestreo = 0;
```

```
//Vectores para almacenar temperaturas, tiempo y diferenciales
```

```
std::vector<float> tTmk1;
```

```
std::vector<float> tempTmk1;
```

```
std::vector<float> dfTmk1;
```

```
std::vector<float> tTmk2;
```

```
std::vector<float> tempTmk2;
```

```
std::vector<float> dfTmk2;
```

```
//Delcarar los servidores y sus puertos
```

```
AsyncWebServer server(80);
```

```
WebSocketsServer websockets(81);
```

```
void setup(void) {
```

```
    //Declaración del puerto serial
```

```
    Serial.begin(115200);
```

```
    //Declaración de los indicadores
```

```
    pinMode(ledPinEncendido, OUTPUT);
```

```
    pinMode(ledPinCensando, OUTPUT);
```

```
    pinMode(ledPinError, OUTPUT);
```

```
    digitalWrite(ledPinEncendido, LOW);
```

```
    digitalWrite(ledPinCensando, LOW);
```

```
    digitalWrite(ledPinError, LOW);
```

```
    //Declaraciones para los termopares
```

```
    SPI.begin(); // Iniciar el bus SPI
```

```
    pinMode(MAX6675_CS1, OUTPUT);
```

```
    pinMode(MAX6675_CS2, OUTPUT);
```

```
digitalWrite(MAX6675_CS1, HIGH); // Deseleccionar el chip 1
```

```
digitalWrite(MAX6675_CS2, HIGH); // Deseleccionar el chip 2
```

```
//Configuración del servidor del ESP32
```

```
WiFi.softAP("MedicionTermopark", "LaboratorioTermo2024");
```

```
Serial.println("\nsoftAP");
```

```
Serial.println(WiFi.softAPIP());
```

```
//Configuración del DNS
```

```
if (!MDNS.begin("termopar")) {
```

```
    Serial.println("Error setting up MDNS responder!");
```

```
    digitalWrite(ledPinError, HIGH);
```

```
    while (1) {
```

```
        delay(1000);
```

```
    }
```

```
}
```

```
digitalWrite(ledPinError, LOW);
```

```
Serial.println("mDNS responder started");
```

```
//Montaje del directorio SPIFFS
```

```
if (!SPIFFS.begin(true)) {
```

```
    Serial.println("Error al montar SPIFFS");
```

```
    digitalWrite(ledPinError, HIGH);
```

```
    return;
```

```
}
```

```
//Activar los archivos que se utilizarán en la página web (montados previamente con SPIFFS)
```

```
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
```

```
    { request -> send(SPIFFS, "/index.html", "text/html"); });
```

```
server.on("/estilo.css", HTTP_GET, [](AsyncWebServerRequest *request)
```

```
    { request->send(SPIFFS, "/estilo.css", "text/css"); });
```

```
server.on("/chart.min.js", HTTP_GET, [](AsyncWebServerRequest *request)
```

```
    { request->send(SPIFFS, "/chart.min.js", "application/javascript"); });
```

```
server.on("/graficas.js", HTTP_GET, [](AsyncWebServerRequest *request)
```

```
    { request->send(SPIFFS, "/graficas.js", "application/javascript"); });
```

```
//Agregar evento de página no encontrada (error 404)
```

```
server.onNotFound(notFound);
```

```

//Iniciar el servidor

server.begin();

Serial.println("Server iniciado");


//Iniciar el servidor webSocket

websockets.begin();

websockets.onEvent(webSocketEvent);

}


void loop(void) {

    //Declaramos el gestor de conexiones webSockets

    websockets.loop();


    //Encendemos el led de on para saber que el sistema ha iniciado

    digitalWrite(ledPinEncendido, HIGH);


    //Inicio del bucle para el estudio de temperatura

    if (bEmpezar)

    {

```

```

//Encendido del led censado

digitalWrite(ledPinCensando, HIGH);


//Declaración del cronómetro con millis

static uint32_t prevMillis = 0;

if (millis() - prevMillis >= (tiempoMuestreo * 1000))

{

    //Inicio del bucle para un solo termopar

    if (cantidadTermopares == 1)

    {

        prevMillis = millis();


        //Variable para saber que hay datos para un solo termopar

        bMediciones = 1;


        //Lectura del termopar 1

        float temperatura1 = leer_termopar(MAX6675_CS1);


        //Se añade el valor de temperatura y tiempo al final de cada vector respectivamente

        tempTmk1.push_back(temperatura1);

```

```

tTmk1.push_back(tiempo);

//Calculo del diferencial de temperatura en (iniciando cuando existen 2 mediciones)

if (tiempo != 0)

{

    deltaT = tiempo - tiempoAnterior;

    deltaTemp = temperatura1 - tempAnterior;

    derivada = deltaTemp / deltaT;

}

//Se añade el valor del diferencial al vector correspondiente

dfTmk1.push_back(derivada);

enviarMedicion(true);


//Reasignación de valores a las variables de apoyo

tiempoAnterior = tiempo;

tempAnterior = temperatura1;

tiempo = tiempo + tiempoMuestreo;

}

// Inicio del bucle para 2 termopares

```

```

else {

    prevMillis = millis();

    //Variable para saber que hay datos para un solo termopar

    bMediciones = 2;

    //Lectura de los termopares

    float temperatura1 = leer_termopar(MAX6675_CS1);

    float temperatura2 = leer_termopar(MAX6675_CS2);


    //Se añade el valor de temperatura y tiempo al final de cada vector respectivamente

    tempTmk1.push_back(temperatura1);

    tempTmk2.push_back(temperatura2);

    tTmk1.push_back(tiempo);


    //Calculo del diferencial de temperatura en (iniciando cuando existen 2 mediciones)

    if (tiempo != 0)

    {

        deltaT = tiempo - tiempoAnterior;

        deltaTemp = temperatura1 - tempAnterior;

        deltaTemp2 = temperatura2 - tempAnterior2;

        derivada = deltaTemp / deltaT;
    }
}

```

```

        derivada2 = deltaTemp2 / deltaT;

    }

    //Se añade el valor del diferencial al vector correspondiente

    dfTmk1.push_back(derivada);

    dfTmk2.push_back(derivada2);

    enviarMedicion(false);

    //Reasignación de valores a las variables de apoyo

    tiempoAnterior = tiempo;

    tempAnterior = temperatura1;

    tempAnterior2 = temperatura2;

    tiempo = tiempo + tiempoMuestreo;

}

}

} else {

    digitalWrite(ledPinCensando, LOW);

}

}

void websocketEvent(uint8_t num, WStype_t type, uint8_t *payload, size_t length) {

```

```

switch (type) {

    //Evento para cliente desconectado

    case WStype_DISCONNECTED:

        Serial.printf("[%u] Desconectado!\n", num);

        break;

    case WStype_CONNECTED:

    {

        //Obtención de la ip del cliente

        IPAddress ip = websockets.remoteIP(num);

        Serial.printf("[%u] Conectado en %d.%d.%d.%d url: %s\n", num, ip[0], ip[1], ip[2], ip[3],
payload);

        //Creación de Json con los datos de la configuración de la página web

        JsonDocument configuracion;

        configuracion["indicador"] = "inicio";

        configuracion["cantidadTermopares"] = String(cantidadTermopares);

        configuracion["tiempoMuestreo"] = String(tiempoMuestreo);

        configuracion["botonEstados"] = botonEstados;

        //Serialización del Json

```

```

String jsonConfig;

serializeJson(configuracion, jsonConfig);

//Envío del Json serializado

websockets.broadcastTXT(jsonConfig);

//Comparación de existencia de datos en el estudio, para recargas de página o clientes
nuevos

if (bMediciones == 1)

{

    enviarMedicion(true);

} else if (bMediciones == 2) {

    enviarMedicion(false);

}

}

break;

case WStype_TEXT:

{

    //Obtención del Json enviado por la página web

    Serial.printf("[%u] Mensaje recibido: %s\n", num, payload);

```

```

String msg = String((char *)(payload));

Serial.print(msg);


//Creación de un Json

JsonDocument doc;


//Deserializar el Json recibido

DeserializationError error = deserializeJson(doc, payload);


//Validación para la correcta deserialización

if (!error)

{

    //Evento enviado al empezar o retomar el estudio

    if(doc["indicador"] == "configWeb")

    {

        // Extraer los valores del JSON

        cantidadTermopares = doc["cantidad"].as<int>();

        tiempoMuestreo = doc["muestreo"].as<int>();

        botonEstados = doc["botonEstados"].as<bool>();

        bEmpezar = doc["bEmpezar"].as<bool>();
    }
}

```

```

}

//Evento enviado al reiniciar la página web

else if (doc["indicador"] == "resetWeb")

{

    // Extraer los valores del JSON

    cantidadTermopares = doc["cantidad"].as<int>();

    tiempoMuestreo = doc["muestreo"].as<int>();

    botonEstados = doc["botonEstados"].as<bool>();

    reset();

}

//Evento enviado al finalizar el estudio

else if (doc["indicador"] == "finalizar")

{

    bEmpezar = doc["bEmpezar"].as<bool>();

    botonEstados = doc["botonEstados"].as<bool>();

}

//Evento default

else {

    Serial.println("Ni un caso");

}

```

```

    }

    //Se ejecuta cuando la deserealización del Json falla

    else

    {

        Serial.print("Error al analizar el JSON: ");

        Serial.println(error.c_str());

        digitalWrite(ledPinError, HIGH);

    }

}

break;

}

}

```

```

//Evento de error de página no encontrada (error 404)

```

```

void notFound(AsyncWebServerRequest *request) {

    // Redirigir al cliente a la página principal ("/")

    request->redirect("/");

}

```

```

//Función para obtener la temperatura del termopar deseado

```

```

double leer_termopar(int cs_pin) {

    uint16_t v;

    digitalWrite(cs_pin, LOW); // Seleccionar el chip

    delay(1);

    v = SPI.transfer(0x00); // Leer el primer byte

    v <<= 8;

    v |= SPI.transfer(0x00); // Leer el segundo byte

    digitalWrite(cs_pin, HIGH); // Deseleccionar el chip

    if (v & 0x4) {

        // El bit 2 indica si el termopar está desconectado

        digitalWrite(ledPinError, HIGH);

        Serial.println("Error con el termopar");

        return NAN;

    }

    v >>= 3; // Descartar los 3 bits inferiores

```

```

    return v * 0.25; // Convertir a temperatura
}

//Función para enviar las mediciones de un solo termopar o ambas
void enviarMedicion(bool bandera) {

    //Comparación para identificar la cantidad de termopares
    if (bandera)

    {

        //Creación de un Json
        JsonDocument medicion;

        medicion["indicador"] = "medicion";

        //Creamos un arreglo dentro del Json para cada uno de los vectores
        JsonArray temp1 = medicion["tempTmk1"].to<JsonArray>();

        for (int i = 0; i < tempTmk1.size(); i++)

        {

            temp1.add(tempTmk1[i]);

        }
    }
}

```

```
JSONArray tiempo1 = medicion["tTmk1"].to<JSONArray>();
```

```
for (int i = 0; i < tTmk1.size(); i++)
```

```
{
```

```
    tiempo1.add(tTmk1[i]);
```

```
}
```

```
JSONArray dfTemp1 = medicion["dfTmk1"].to<JSONArray>();
```

```
for (int i = 0; i < dfTmk1.size(); i++)
```

```
{
```

```
    dfTemp1.add(dfTmk1[i]);
```

```
}
```

```
//Serealización y envio del Json
```

```
String datas;
```

```
serializeJson(medicion, datas);
```

```
websockets.broadcastTXT(datas);
```

```
Serial.println(datas);
```

```
} else { //Enviar la información de los dos termopares
```

```
//Creación de un Json
```

```

JsonDocument medicion;

medicion["indicador"] = "medicion2";


//Creamos un arreglo dentro del Json para cada uno de los vectores

JsonArray temp1 = medicion["tempTmk1"].to<JsonArray>();

for (int i = 0; i < tempTmk1.size(); i++)

{

    temp1.add(tempTmk1[i]);

}


JsonArray tiempo1 = medicion["tTmk1"].to<JsonArray>();

for (int i = 0; i < tTmk1.size(); i++)

{

    tiempo1.add(tTmk1[i]);

}


JsonArray dfTemp1 = medicion["dfTmk1"].to<JsonArray>();

for (int i = 0; i < dfTmk1.size(); i++)

{

    dfTemp1.add(dfTmk1[i]);

}

```

```
}
```

```
JSONArray temp2 = medicion["tempTmk2"].to<JSONArray>();
```

```
for (int i = 0; i < tempTmk2.size(); i++)
```

```
{
```

```
    temp2.add(tempTmk2[i]);
```

```
}
```

```
JSONArray dfTemp2 = medicion["dfTmk2"].to<JSONArray>();
```

```
for (int i = 0; i < dfTmk2.size(); i++)
```

```
{
```

```
    dfTemp2.add(dfTmk2[i]);
```

```
}
```

```
//Serealización y envio del Json
```

```
String datas;
```

```
serializeJson(medicion, datas);
```

```
websockets.broadcastTXT(datas);
```

```
Serial.println(datas);
```

```

    }

}

//Función para reiniciar los valores de las variables y vectores

void reset() {

    bMediciones = 0;

    tiempo = 0;

    tiempoAnterior = 0;

    tempAnterior = 0;

    deltaT = 0;

    deltaTemp = 0;

    derivada = 0;

    tTmk1.clear();

    tempTmk1.clear();

    dfTmk1.clear();

    tTmk2.clear();

    tempTmk2.clear();

    dfTmk2.clear();

}

```

8.9 Creación del microcontrolador *ESP32* en *Proteus*

Para crear este componente se debe de dirigir a la siguiente página web <https://sites.google.com/view/mv-technologie-ecu/esquematicos-electronicos>, dentro de la misma, se descargará un archivo *ZIP* el cual contendrá los archivos necesarios para agregar al *ESP32* al directorio de componentes, en la figura 139 se muestra el botón de descarga del componente.



Figura 139: Botón de descarga para crear el *ESP32* en *Proteus*.

Fuente: <https://sites.google.com/view/mv-technologie-ecu/esquematicos-electronicos>

Una vez descargado, se extrae el contenido y se abre el archivo “ESQUEMATICO ESP32” con el *software* de *Proteus*. Después en la pestaña “PCB Layout” se selecciona todo el componente, se copia y pega en el mismo archivo, después se borra el anterior, y luego se presiona el clic derecho para seleccionar la opción “Make Package”, lo anterior se muestra en la figura 140.

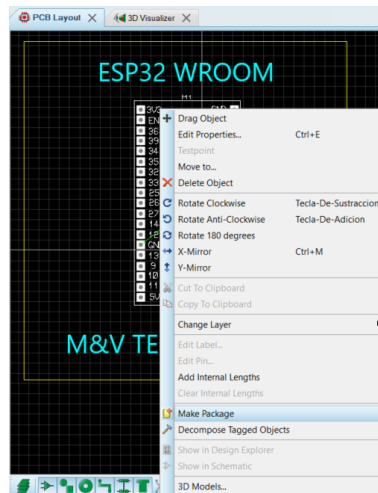


Figura 140: Selección de opción "Make Package" para generar el PCB del ESP32.

Fuente: Elaboración propia.

Dentro de la ventana solo se acepta y se confirma la acción para guardar el diseño. Luego, en la pestaña "Schematic Capture" se presiona clic derecho al componente y se selecciona la opción "Make Device", como se muestra en la figura 141, y se acepta la ventana emergente (en caso de aparecer).

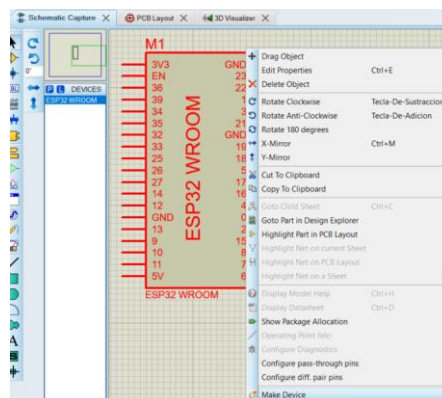


Figura 141: Selección de opción "Make Device" para generar el esquemático del ESP32.

Fuente: Elaboración propia.

Por último, solo se presiona el botón de siguiente hasta crear el dispositivo y ya se puede usar los recursos para diseñar una PCB y esquemático para el microcontrolador ESP32.

8.10 Diseño 3D de la caja y tapadera del dispositivo mecatrónico

Se agrega la figura 142 para ilustrar el diseño *CAD* de la caja y la tapadera del diseño mecatrónico.

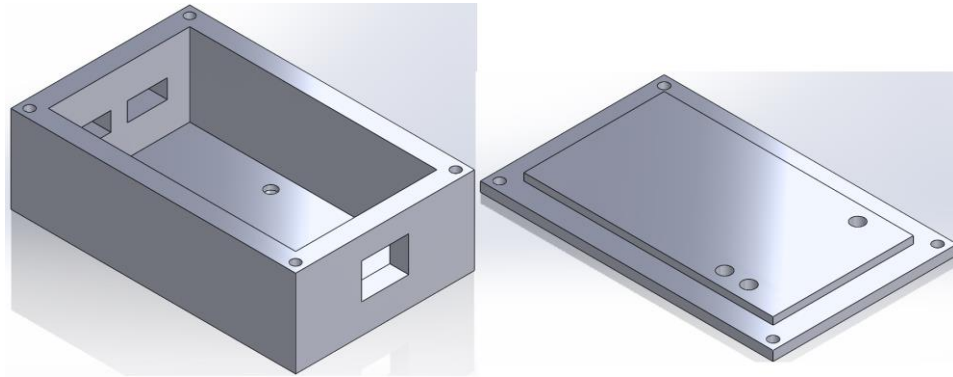


Figura 142: Diseños en 3D en el *software SolidWorks* para la caja y tapadera.

Fuente: Elaboración propia.