Informe Técnico: Implementación y Análisis del PT100 en Circuitos de Amplificadores Operacionales

Santiago I. Flores Chavez
CUI: 20210197
Instrumentación Física 1

Indice

Introducción

1.1. Contexto del PT100 y su importancia en mediciones de temperatura.

La temperatura es una magnitud omnipresente en la instrumentación de medición y control. Diversos dispositivos como resistencias (RTD), termistores, termopares y semiconductores como diodos se utilizan para detectar cambios y medir temperatura absoluta. Entre ellos, el sensor de temperatura RTD de platino destaca por ser el más preciso, lineal y estable a lo largo del tiempo. Su mejora constante en tecnología refuerza la calidad en mediciones de temperatura. Específicamente, el RTD de platino de 100Ω se ha consolidado como un estándar en mediciones por su estabilidad y linealidad en un amplio rango de temperaturas.

1.2. Justificación del uso de circuitos con amplificadores operacionales para el PT100.

Al medir temperaturas con el RTD de platino, es crucial condicionar y traducir su señal analógica al dominio digital. En este contexto, los amplificadores operacionales juegan un papel esencial. Estos componentes no sólo amplifican las señales débiles provenientes del sensor, sino que también pueden configurarse para crear fuentes de corriente estables, esenciales para el correcto funcionamiento del PT100. Además, al evitar el autocalentamiento y asegurar una correcta excitación, la precisión en las mediciones puede ser de hasta ±4,3°C en un rango de temperatura de -200°C a 800°C.

1.3. Objetivos de la práctica.

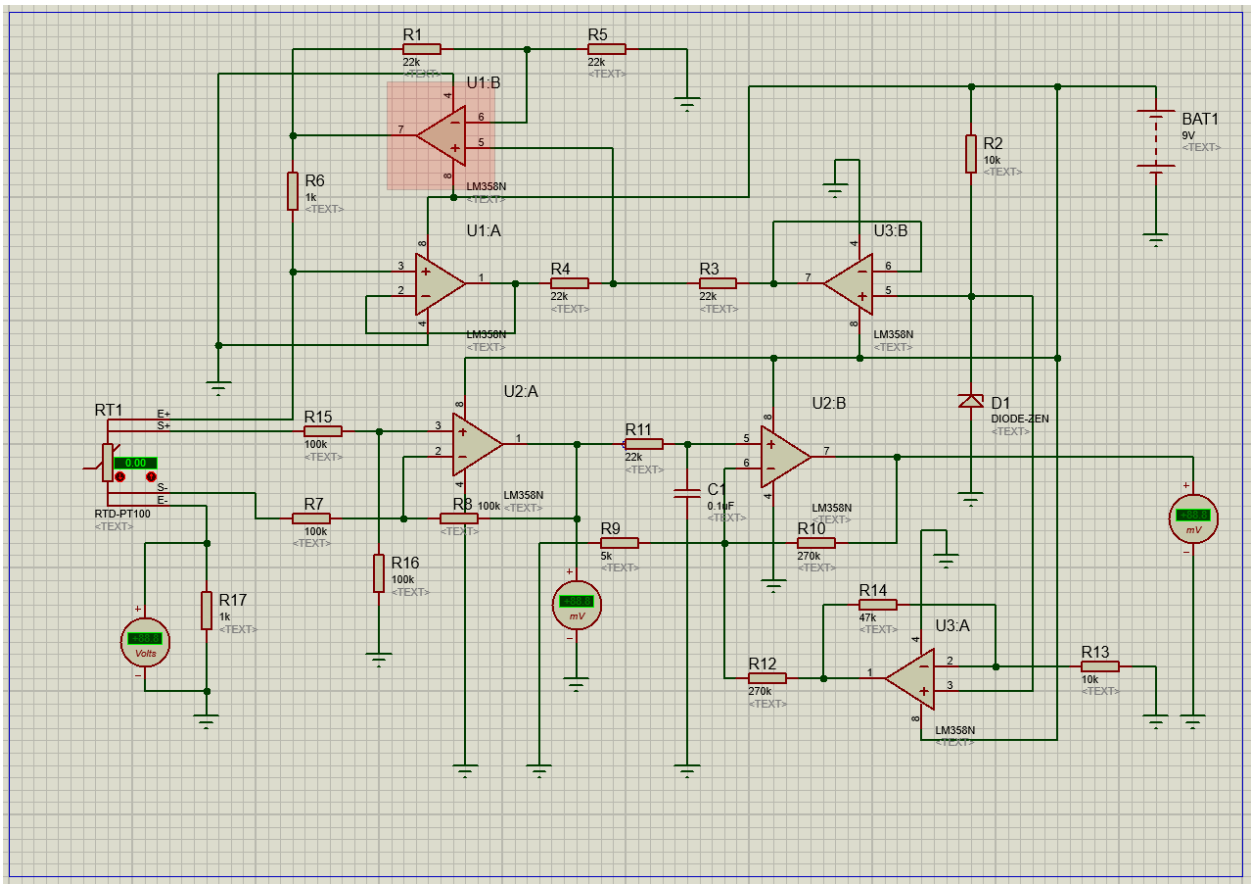
- Comprender las propiedades y características fundamentales de los sensores RTD de platino, así como sus aplicaciones en la medición de temperatura.
- Diseñar un circuito basado en amplificadores operacionales que mida la temperatura utilizando un PT100 de 4 hilos, donde la salida del circuito varía en el rango de 0V a 5V para temperaturas entre -50°C y 50°C.
- Simular el circuito diseñado utilizando el software de simulación de circuitos eléctricos en Proteus.
- Utilizar el software Python para recopilar y procesar datos de temperatura obtenidos a través del circuito diseñado.
- Implementar un programa que mida la temperatura ambiente a una tasa de un dato por segundo durante 24 horas y guarde estos datos en un archivo CSV.
- Representar gráficamente los datos de temperatura recopilados, así como su derivada, para un análisis visual más fácil.
- Comparar los datos de temperatura recopilados con datos estándar para validar la precisión y eficacia del diseño del circuito y el programa implementado.

Marco Teórico

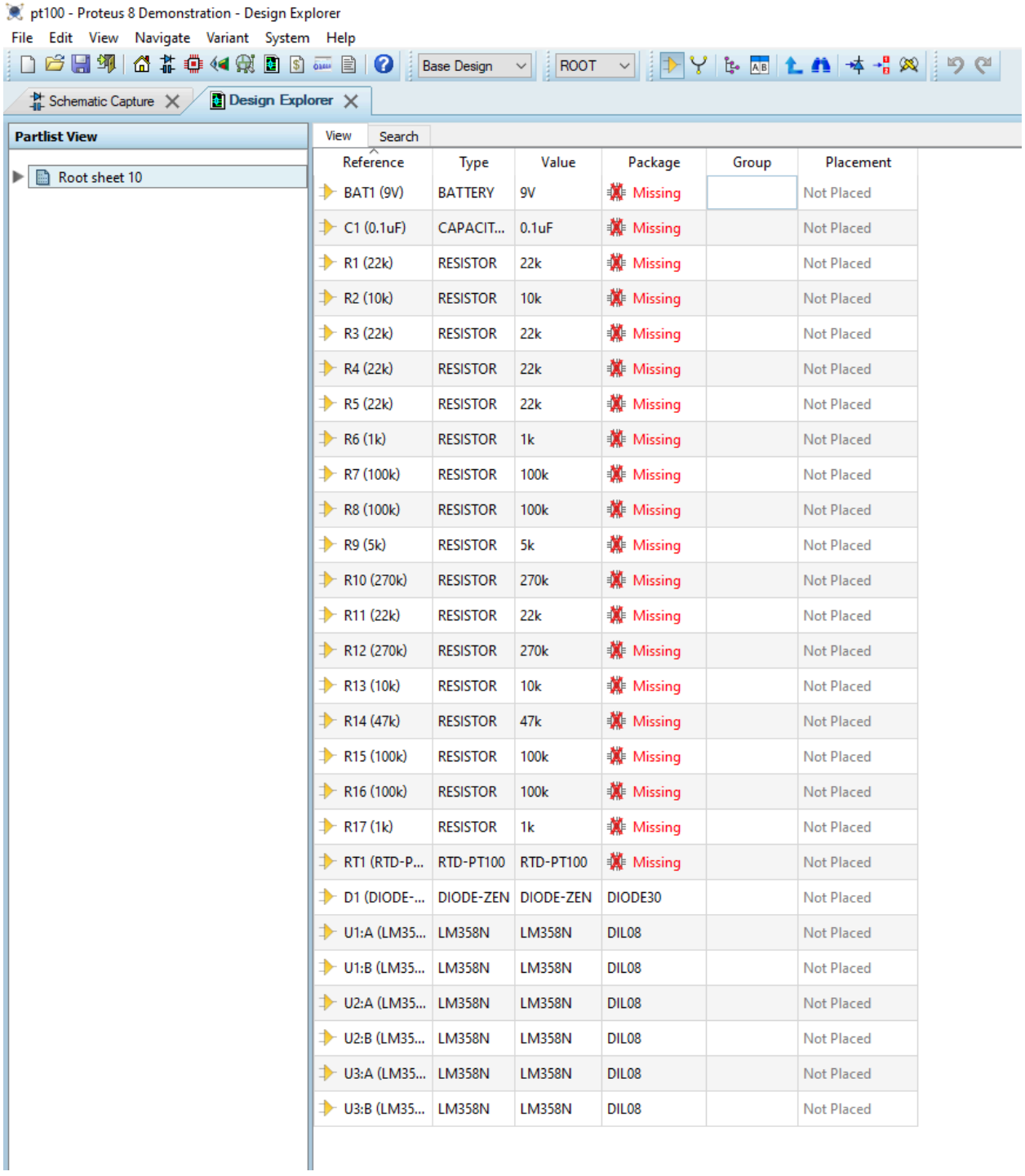
La medición de temperatura es vital en sistemas de instrumentación. Se utilizan diversos elementos para esta tarea, siendo las resistencias (RTD), en particular el RTD de platino, destacadas por su precisión, estabilidad y linealidad. La base de su funcionamiento radica en que cambios de temperatura alteran su resistencia eléctrica. El RTD de platino de 100Ω es notable debido a su linealidad y estabilidad en un extenso rango de temperaturas. Es esencial alimentar los RTD con una corriente adecuada para evitar recalentamientos. La relación entre resistencia y temperatura de los RTD puede describirse con ecuaciones. En contextos de rangos más amplios, se emplea la ecuación de Callendar-Van Dusen. La implementación de RTD en sistemas conlleva el uso de fuentes de corriente, amplificadores y sistemas de adquisición de datos.

Diseño del Circuito

3.1. Diagrama esquemático del circuito.



Simulación del Circuito



Reference	Type	Value	Package	Group	Placement
BAT1 (9V)	BATTERY	9V	Missing		Not Placed
C1 (0.1uF)	CAPACIT...	0.1uF	Missing		Not Placed
R1 (22k)	RESISTOR	22k	Missing		Not Placed
R2 (10k)	RESISTOR	10k	Missing		Not Placed
R3 (22k)	RESISTOR	22k	Missing		Not Placed
R4 (22k)	RESISTOR	22k	Missing		Not Placed
R5 (22k)	RESISTOR	22k	Missing		Not Placed
R6 (1k)	RESISTOR	1k	Missing		Not Placed
R7 (100k)	RESISTOR	100k	Missing		Not Placed
R8 (100k)	RESISTOR	100k	Missing		Not Placed
R9 (5k)	RESISTOR	5k	Missing		Not Placed
R10 (270k)	RESISTOR	270k	Missing		Not Placed
R11 (22k)	RESISTOR	22k	Missing		Not Placed
R12 (270k)	RESISTOR	270k	Missing		Not Placed
R13 (10k)	RESISTOR	10k	Missing		Not Placed
R14 (47k)	RESISTOR	47k	Missing		Not Placed
R15 (100k)	RESISTOR	100k	Missing		Not Placed
R16 (100k)	RESISTOR	100k	Missing		Not Placed
R17 (1k)	RESISTOR	1k	Missing		Not Placed
RT1 (RTD-P...	RTD-PT100	RTD-PT100	Missing		Not Placed
D1 (DIODE-...	DIODE-ZEN	DIODE-ZEN	DIODE30		Not Placed
U1:A (LM35...	LM358N	LM358N	DIL08		Not Placed
U1:B (LM35...	LM358N	LM358N	DIL08		Not Placed
U2:A (LM35...	LM358N	LM358N	DIL08		Not Placed
U2:B (LM35...	LM358N	LM358N	DIL08		Not Placed
U3:A (LM35...	LM358N	LM358N	DIL08		Not Placed
U3:B (LM35...	LM358N	LM358N	DIL08		Not Placed

Programación y Registro de Datos

5.1. Descripción de la herramienta de conexión remota y su propósito.

AnyDesk es una herramienta de conexión remota utilizada para acceder y controlar computadoras desde otras ubicaciones. En este experimento, se utiliza AnyDesk para conectarse a la computadora identificada como 1031513726. La clave para el acceso es "instrumentacion1". El propósito principal de esta conexión es ejecutar el programa de adquisición de datos de temperatura ambiente a una tasa de un dato por segundo durante 24 horas.

5.2. Programa desarrollado para la adquisición de datos de temperatura.

Se empleó un programa basado en Python, que utiliza librerías como numpy para operaciones matemáticas, matplotlib para graficación, datetime para gestionar fechas y tiempos, csv para manipular archivos CSV y uLogg para interactuar con el dispositivo de medición. El programa está diseñado para adquirir datos de temperatura del ambiente, almacenar estos datos en un archivo CSV y, finalmente, graficarlos junto con su derivada.

5.3. Características y estructura del archivo CSV generado.

El archivo CSV, denominado 'temperatura_data.csv', está estructurado en dos columnas: 'timestamp', que registra la marca temporal de la medición, y 'temperature', que almacena el valor de la temperatura medida en grados Celsius. Este archivo se actualiza a una tasa de un dato por segundo durante las 24 horas de experimentación.

5.4. Código comentado (línea por línea).

Se usó el código proporcionado por el Dr. Juan Ernesto Palo Tejada proporcionado en el informe para realizar esta práctica, en el cual se realizaron las modificaciones necesarias para obtener los datos según los requisitos necesarios, se comentó el código linea por linea, el codigo se puede encontrar en el **Anexo 2** de este informe o acceder vía internet a través del siguiente enlace, el código fue almacenado en un repositorio público en GitHub: https://github.com/IsSantiagoFL/physical_instrumentation_1/blob/main/alpha.py

5.5. Propuesta de mejora: Optimización y robustez del código propuesto.

A partir del código proporcionado por el docente, se buscó optimizar la estructura y funcionalidad para satisfacer las necesidades del experimento. Se introdujeron funciones para modularizar tareas específicas, nombres de variables y constantes más descriptivos, y tiempos de lectura y frecuencias de intervalo adaptados a las necesidades del experimentador.

Además, se realizaron mejoras en la visualización gráfica, incluyendo el uso de leyendas, para una interpretación más intuitiva de los resultados. Este enfoque de reestructuración permitió generar un código más limpio, fácil de leer y robusto, con un potencial significativo para futuras mejoras y adaptaciones.

Un código alternativo, que representa una versión más robusta y potente de este experimento, ha sido desarrollado y compartido en un repositorio público de GitHub, en el siguiente enlace (también se puede encontrar en el **Anexo 1**) https://github.com/IsSantiagoFL/physical_instrumentation_1/blob/main/pt100_8.1.0-RC.py

Análisis Gráfico de los Datos

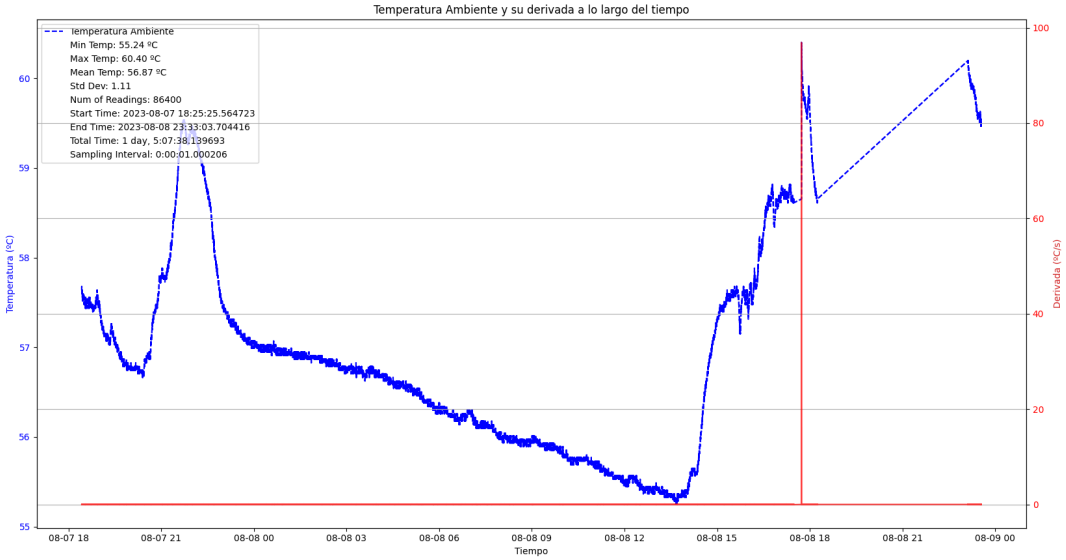
6.1. Herramientas y métodos utilizados para graficar.

Se utilizó la biblioteca matplotlib.pyplot de Python para la generación y visualización de gráficos. Los datos se procesaron mediante NumPy para calcular las diferencias de temperatura y tiempo, y posteriormente, obtener la derivada de la temperatura con respecto al tiempo.

6.2. Gráfica de temperatura versus tiempo y su derivada correspondiente.

La gráfica muestra en el eje x el tiempo y en el eje y la temperatura en grados Celsius. Se incluye una línea secundaria que representa la derivada de la temperatura con respecto al tiempo, ofreciendo una visión más detallada

de la tasa de cambio de la temperatura. Esta derivada se muestra en un eje vertical adicional en la misma gráfica para su fácil comparación.



La gráfica se puede visualizar mejor en el siguiente enlace:
https://drive.google.com/file/d/1GVOC27yd9l4mpjpVKWqe_w96BAOL_tJW/view?usp=sharing
y en el **Anexo 3,** ya que se muestra en su tamaño completo.

6.3. Interpretación y análisis de los resultados gráficos.
La derivada de la gráfica nos brinda información sobre la rapidez con la que varía la temperatura en el tiempo. Un valor positivo en la derivada indica un incremento en la temperatura, mientras que un valor negativo señala una disminución o enfriamiento. La magnitud de estos valores representa la velocidad de dicho cambio: valores más elevados indican variaciones más abruptas y viceversa. Al analizar conjuntamente la gráfica original y su derivada, se logra una comprensión más profunda no sólo del comportamiento de la temperatura sino también de la celeridad con la que se produce ese comportamiento.

Comparativa con Datos Externos

7.1. Datos de temperaturas máximas y mínimas proporcionadas por SENAMHI para Arequipa.
Temperatura máxima registrada: 27.9°C
Temperatura mínima registrada: 10.4°C

7.2. Comparación y análisis de las diferencias y/o coincidencias entre datos experimentales y datos externos.
Nuestros resultados experimentales presentan notables diferencias respecto a los datos proporcionados por SENAMHI. Observamos que las temperaturas mínimas y máximas que registramos son considerablemente más altas, con una temperatura mínima de 55.24°C y máxima de 60.40°C. Además, la temperatura promedio que obtuvimos es de 56.87°C, con una desviación estándar de 1.11°C.
Estas discrepancias podrían deberse a errores en el instrumento de medición, en la calibración, o en la metodología empleada.
Los datos en CSV, se pueden encontrar en el siguiente enlace:
https://drive.google.com/file/d/1IM-YUYUcYBQbjci8l3ipUR3DuQ1809JY/view?usp=sharing

Conclusiones

8.1. Síntesis de los hallazgos y aprendizajes de la práctica.

El sensor RTD de platino PT100 ha demostrado ser un instrumento de alta precisión y fiabilidad para la medición de temperaturas en un amplio rango.

Los circuitos con amplificadores operacionales son cruciales para el adecuado procesamiento y amplificación de las señales provenientes del PT100, y su correcta implementación influye significativamente en la precisión de las mediciones.

La simulación del circuito en Proteus y la programación en Python han permitido validar, procesar y visualizar los datos de forma efectiva. Sin embargo, se observaron discrepancias significativas entre los datos experimentales y los proporcionados por SENAMHI, lo que sugiere la necesidad de revisar y perfeccionar aspectos del diseño y la implementación.

El análisis gráfico, incluyendo la derivada de la temperatura respecto al tiempo, proporciona una comprensión más detallada y profunda de la variabilidad y la tasa de cambio de la temperatura en un periodo determinado.

8.2. Implicaciones de los resultados en futuras investigaciones o aplicaciones.

Los resultados indican la importancia de una correcta calibración y validación del sistema de medición para asegurar la precisión en aplicaciones reales.

Las discrepancias encontradas resaltan la necesidad de una evaluación minuciosa de los componentes y el software utilizado.

Este estudio sienta las bases para futuras investigaciones que busquen optimizar o adaptar sistemas basados en PT100 para diferentes aplicaciones y condiciones.

Recomendaciones

9.1. Mejoras propuestas para futuras implementaciones o prácticas similares.

- Es crucial realizar una calibración del PT100 en condiciones controladas antes de su uso en mediciones reales para garantizar su precisión.

- Se sugiere revisar y, si es necesario, mejorar la alimentación y estabilidad del circuito para evitar posibles interferencias o ruidos que puedan afectar las mediciones.

- La integración de técnicas avanzadas de filtrado de señales podría mejorar la precisión y reducir el ruido en los datos recopilados.

- Asegurarse de que el software de recopilación y análisis de datos esté optimizado y libre de errores, posiblemente mediante revisiones de código por pares o pruebas extensivas.

9.2. Recomendaciones para la interpretación y uso de los datos recopilados.

- Al analizar los datos, es esencial considerar las posibles fuentes de error o variabilidad, incluyendo factores ambientales, errores de instrumentación o calibración.

- Los datos recopilados deben compararse con fuentes confiables y estandarizadas, como SENAMHI, para validar su precisión.

- Antes de aplicar los datos en contextos prácticos o decisiones importantes, es vital llevar a cabo análisis estadísticos y técnicos para garantizar su confiabilidad.

9.3 Recomendaciones en el código.

- ***Calibración de la Temperatura en el Código:*** Se identificaron lecturas de temperatura excepcionalmente altas en comparación con los datos proporcionados por el SENAMHI. Se recomienda realizar una revisión exhaustiva de la calibración de temperatura en el código. Si tras revisar la calibración se confirma que esta es correcta, se deberán considerar otros factores que puedan estar influyendo en estas lecturas atípicas.

- ***Revisión de Funciones de Derivadas en Python:*** Es esencial revisar y validar las funciones en Python responsables de calcular las derivadas. Se detectaron potenciales inconsistencias que podrían afectar la precisión de los resultados.

- ***Optimización de Cálculos con NumPy:*** Consideramos que el módulo NumPy no se está aprovechando al máximo en el código actual. NumPy ofrece herramientas eficientes para cálculos matemáticos y podría ser más adecuado para calcular derivadas y otras operaciones matemáticas en este contexto.

- ***Ampliación y Mejora del Código:*** El código actual presenta una amplia gama de oportunidades para mejoras y optimizaciones futuras. Se recomienda una revisión integral para identificar áreas de mejora y garantizar que el sistema sea escalable y eficiente en el futuro.

Anexo 1

```
# Importar los módulos necesarios
import numpy as np # Para realizar operaciones matemáticas, arrays, matrices, etc.
import matplotlib.pyplot as plt # Para la creación de gráficos y visulizarlos
from datetime import datetime # Para trabajar con fechas y tiempos
from datetime import timedelta # Para trabajar con fechas y tiempos
from uLogg import Meter # Para interactuar con el datalogger
import csv # Importar el modulo csv para guardar los datos en un .csv

# Definir las constantes:
PORT = 'COM15' # Indica al puerto al que se conectara el dispositivo
CONFIG_FILE = 'config1.txt' # Indica el archivo que contiene la configuración del dispositivo

# Crea una función que devuelve "D_TIME" a partir de lo que el usuario desee
def get_sampling_interval():
	'''Solicita al usuario el intervalo de muestreo (tiempo entre lecturas) en segundos'''
	try:
		D_TIME = float(input("Ingrese el intervalo de muestre en segundos (por ejemplo, 1.0 para un dato
por segundo): "))
		return D_TIME
	except ValueError:
		print("Por favor, ingrese un valor numérico válido.")
		return get_sampling_interval()

# Funcion: piude al usuario el tiempo de toma de datos y lo convierte a segundos.
def get_duration_in_seconds():
	'''Solicita al usuario la duración y la unidad (segundos, mimutos u horas)
	y devuelve la duración total en segundos'''

	print("¿Durante cuanto tiempo desea recolectar datos?")
	print("Primero seleccione la unidad de tiempo:")
	print("1: Segundos")
	print("2: Minutos")
	print("3: Horas")

	choice = input("Ingrese su opción (1/2/3): ")

	if choice == "1":
		try:
			duration = float(input("Ingrese la duración en segundos: "))
			return duration
		except ValueError:
			print("Por favor, ingrese un valor numérico válido.")
			return get_duration_in_seconds()

	elif choice == "2":
		try:
			duration = float(input("Ingrese la duración en minutos: "))
			return duration * 60 # Convertimos minutos a segundos
		except ValueError:
			print("Por favor, ingrese un valor numérico válido.")
			return get_duration_in_seconds()

	elif choice == "3":
		try:
			duration = float(input("Ingrese la duración en horas: "))
```

```
                return duration * 3600 # Convertimos horas a segundos
            except ValueError:
                print("Por favor, ingrese un valor numérico válido.")
                return get_duration_in_seconds()
        else:
            print("Opción no valida, Por favor, selecione 1, 2 o 3.")
            return get_duration_in_seconds()

# Configuración del dispositivo en una función:
def configure_device(port, config_file): # Los parametros necesarios sera, el puerto y el archivo de configuración
    '''Configura el dispositivo.'''
    # A continuación de inicializa el dispostivo creando una instancia de la clase "Meter"
    mt = Meter(Puerto = port) # Se indica que se usara el puerto "port" ya definido para comunicarse con el dispositivo
    mt.ReadConfFile(config_file) # Se usa un método para configurar el dispositivo usando el archivo de configuración especificado
    return mt # Devuelve el objeto "mt" que es el dispotivo ya configurado y conectado, listo para ser usado.

# Adquisición de datos con el dispositivo ya configurado:
def collect_data(device, num_readings): # Parametros de entrada: Dispositivo previamente configurado y numero de lecturas a tomar
    '''Recolecta datos del dispositivo.'''
    data = [[],[]] # Se inicializa una lista llamada "data", es una lista bidimensional para almacenar los datos del dispositivo
    cont = 0 # Se inicializa el contador "cont" en cero, para rastrear el conjunto de datos recolectados
    time_wait = 0 # Se inicializa la varialbe a cero, Es el tiempo que espera antes de tomar la siguiente lectura

    while cont < num_readings: # Comienza un bucle que se ejecuta mientras el contador cont sea menor que num_readings.
        time_now = datetime.now()           # Obtiene el tiempo actual y lo almacena en la variable time_now.
        time_unix = time_now.timestamp() # Convierte time_now a una marca de tiempo UNIX y lo almacena en time_unix.

        if time_unix >= time_wait: # Verifica si la marca de tiempo UNIX actual es mayor o igual a time_wait. Si es cierto, es hora de recolectar un nuevo conjunto de datos.
            time_wait = time_unix + D_TIME # Actualiza la variable time_wait agregándole D_TIME, que es una constante definida previamente. Esto establece el próximo punto en el tiempo en el que se debería recolectar otro conjunto de datos.
            dt = device.GetOneDat(False) # Llama al método GetOneDat del objeto device (que representa el dispositivo) para obtener un conjunto de datos.
            # Los datos seran almacenados en la lista "data":
            data[0].append(dt[0]) # Agrega el primer elemento del conjunto de datos (dt[0]) a la primera lista en data.
            data[1].append((40.5844E-3)*dt[1]-22.9740E1) # Multiplica el segundo elemento del conjunto de datos (dt[1]) por 40.5844E-3, le resta 22.9740E1, y añade el resultado a la segunda lista en data.
            cont += 1 # Incrementa el contador cont en 1, indicando que se ha recolectado un conjunto adicional de datos.
            print(f'{cont} {str(data[0][-1])[:-5]} {data[1][-1]:.1f}') # imprimir información en la consola: numero actual de conjunto de datos recolectados, ultimo valor agregado a las listas de data.

    return data # devuelve la lista data que contiene los datos recolectados.

# Guarda los datos recolectados en un archivo .csv
def save_to_csv(data, filename="data.csv"): # Aquí se esta definicinedo la función
    '''Guarda los datos en un archivo CSV.'''
```

```
    with open(filename, 'w', newline='') as file: # Se esta creando el archivo con permisos de escritura
        writer = csv.writer(file) # creamos el objeto "writer" que nos permite escribir datos en ela archivo
        writer.writerow(["Tiempo", "Temperatura (ºC)"])  # Escribir encabezados
        for i in range(len(data[0])): # El bucle iterara cada elemento de "data" donde estan los datos guardados
            writer.writerow([data[0][i], data[1][i]]) # y luego los ira escibiendo en el archivo CSV

# Se define la función para crear las graficas de los datos obtenidos:
def plot_data(data): # El unico parametro de entrada que acepta es "data" la lista para almacenar los datos obtenidos por el dispositivo.
        """Grafica los datos recolectados y su derivada."""
        # Obteniendo las diferencias de tiempo y temperatura.
        time_deltas = np.diff(data[0]) # Diferencias consecutivas en el tiempo
        temp_deltas = np.diff(data[1]) # Diferencias consecutivas en la temperatura

        # Convertir time_deltas a segundos (es una lista de objetos timedelta)
        time_deltas_in_seconds = [delta.total_seconds() for delta in time_deltas]

        # Calculamos la derivada (pendiente) de la temperatura respecto al tiempo
        derivatives = temp_deltas / np.array(time_deltas_in_seconds)

        # Graficamos los datos originales (Temperatura)
        fig, ax1 = plt.subplots()

        ax1.set_xlabel('Tiempo')
        ax1.set_ylabel('Temperatura (ºC)', color='#0000FF')
        ax1.plot(data[0], data[1], label='Temperatura Ambiente', color='#0000FF', linestyle='--')
        ax1.tick_params(axis='y', labelcolor='#0000FF')

        # Creando un segundo eje para la derivada
        ax2 = ax1.twinx()
        ax2.set_ylabel('Derivada (ºC/s)', color = 'tab:red')
        ax2.plot(data[0][1:], derivatives, label='Derivada', color='red', alpha=0.75)  # Color rojo con transparencia
        ax2.tick_params(axis='y', labelcolor='red')

        # Configurando el aspecto visual
        plt.title('Temperatura Ambiente y su derivada a lo largo del tiempo') # Título del gráfico
        fig.tight_layout()
        # plt.legend() # Mostrar leyenda
        plt.grid(True) # Mostrar una grilla para facilitar la lectura

        # Definimos las líneas de la gráfica
        temp_handle, = ax1.plot(data[0], data[1], label='Temperatura Ambiente', color='#0000FF', linestyle='--')

        # Definimos las leyendas usando líneas invisibles
        labels = [
    f"Min Temp: {min_temp:.2f} ºC",
    f"Max Temp: {max_temp:.2f} ºC",
    f"Mean Temp: {mean_temp:.2f} ºC",
    f"Std Dev: {std_temp:.2f}",
    f"Num of Readings: {num_readings}",
    f"Start Time: {start_time}",
    f"End Time: {end_time}",
    f"Total Time: {total_time}",
    f"Sampling Interval: {sampling_interval}"
        ]
        handles = [temp_handle] + [ax1.plot([], [], marker="", linestyle="", label=label)[0] for label in labels]
```

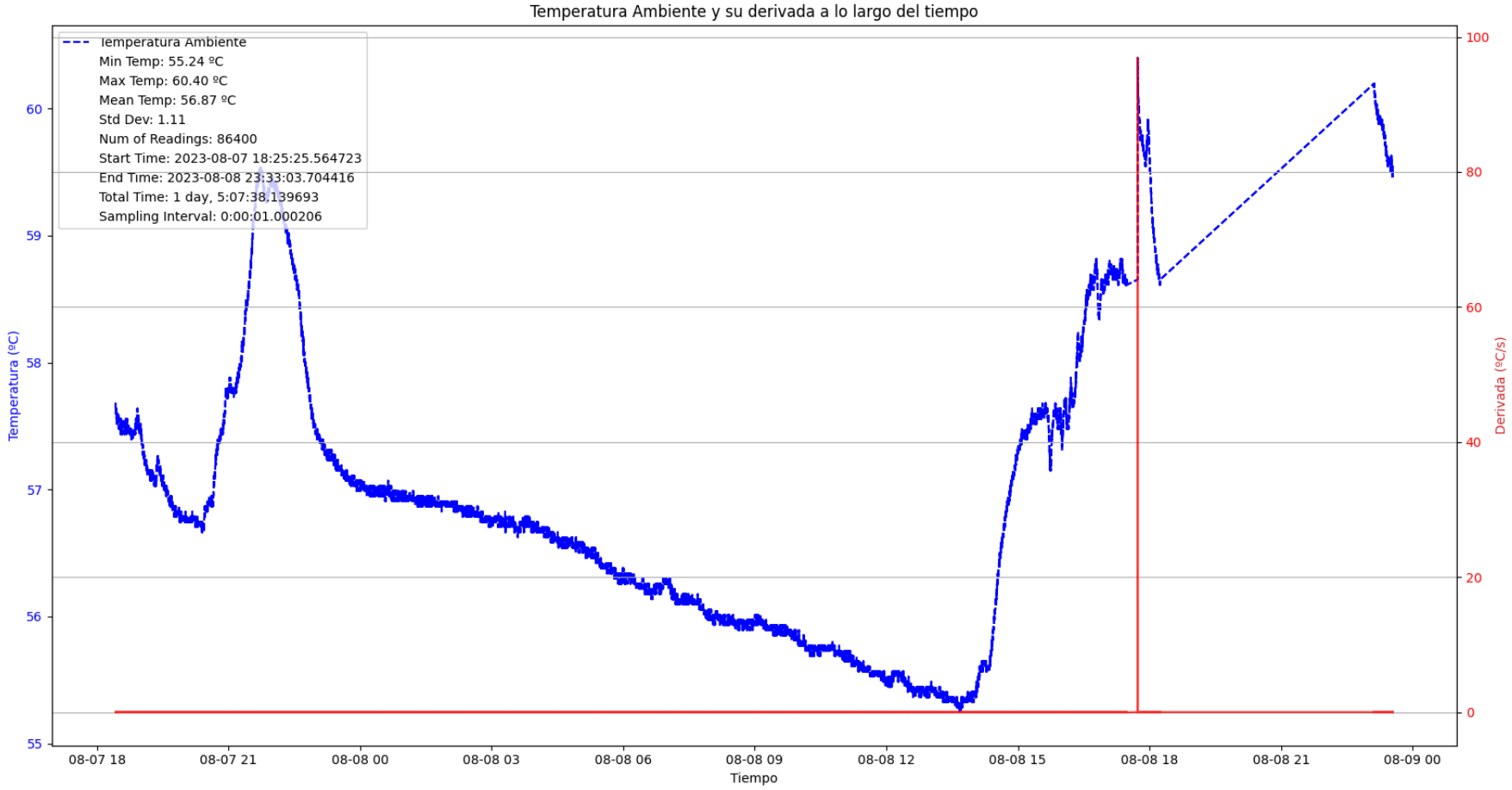
```
        # Mostramos la leyenda con todas las etiquetas en el eje izquierdo
        ax1.legend(handles=handles, loc="upper left")


        # para visualizar el gráfico en una ventana emergente.
        plt.show() # Esta función muestra el gráfico y permite al usuario interactuar con él


if __name__ == "__main__":
        # Solicitamos el intervalo de muestreo al usuario:
        D_TIME = get_sampling_interval()

        # Calcularemos el número total de lecturas que tomara el dispositivo.
        time_seconds = get_duration_in_seconds()
        NUM_READINGS = int(time_seconds / D_TIME) # divismos el tiempo de muestreo entre el intervalo de
muestreo, ambos en unidades de segundos.

        # Configuración del dispositivo:
        device = configure_device(PORT, CONFIG_FILE) # Se configura el dispositivo y se guarda en nla variable
"device"

        # Adquisición de datos
        data = collect_data(device, NUM_READINGS) # Se hace al adquisiscion de datos y se guarda en la varible
"data"

        # Calculando las estadísticas
        start_time = data[0][0]
        end_time = data[0][-1]
        total_time = end_time - start_time
        sampling_interval = data[0][1] - data[0][0]
        temperatures = np.array(data[1])
        min_temp = np.min(temperatures)
        max_temp = np.max(temperatures)
        mean_temp = np.mean(temperatures)
        std_temp = np.std(temperatures)
        num_readings = len(temperatures)

        # Guardar los datos en un archivo CSV:
        save_to_csv(data, "temperatura_ambiente.csv")

        # Finalización y graficación
        device.ClosePortLogg() # Se cierra la comunicación, la conexción con el dispositivo "device", liberando el
recurso asociado
        plot_data(data) # Se grafica los datos guardados en la variable "data"
```

Anexo 2

```
# Importamos la librería numpy para operaciones matemáticas
import numpy as np
# Importamos la librería matplotlib para graficar
import matplotlib.pyplot as plt
# Importamos funciones de datetime para manejar fechas y tiempos
from datetime import datetime, timedelta
# Importamos la librería csv para escribir y leer archivos CSV
import csv
# Importamos la clase Meter de la librería uLogg para manejar el dispositivo de medición
from uLogg import Meter

# Verificamos si este script se está ejecutando como el principal
if __name__ == "__main__":

    # Definimos el intervalo de tiempo entre mediciones, en este caso 1 segundo
    Dtime = 1
    # Inicializamos una variable para esperar el siguiente tiempo de medición
    TimeWait = 0
    # Contador para llevar registro de cuántas mediciones se han realizado
    cont = 0
    # Definimos el tiempo total de medición, en este caso 24 horas convertidas a segundos
    total_time = 24 * 60 * 60

    # Instanciamos el objeto Meter para comunicarnos con el dispositivo a través del puerto 'COM15'
    mt = Meter(Puerto='COM15')
    # Leemos el archivo de configuración para configurar el dispositivo
    mt.ReadConfFile('config1.txt')

    # Inicializamos una lista para almacenar los datos de tiempo y temperatura
    data = [[], []]

    # Abrimos (o creamos) un archivo CSV en modo escritura
    with open('temperatura_data.csv', 'w', newline='') as csvfile:
        # Definimos los nombres de las columnas del archivo CSV
        fieldnames = ['timestamp', 'temperature']
        # Creamos un escritor de diccionarios para el archivo CSV
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        # Escribimos la cabecera del archivo CSV
        writer.writeheader()

        # Mientras no hayamos alcanzado el tiempo total de medición
        while cont < total_time:
            # Obtenemos la fecha y hora actual
            TimeNow = datetime.now()
            # Convertimos la fecha y hora actual a un timestamp (segundos desde 1970)
            TimeUnix = TimeNow.timestamp()

            # Si el timestamp actual es mayor o igual al tiempo de espera
            if TimeUnix >= TimeWait:
                # Actualizamos el tiempo de espera sumándole el intervalo de tiempo
                TimeWait = TimeUnix + Dtime
                # Obtenemos un dato del dispositivo
                dt = mt.GetOneDat(False)
                # Calculamos la temperatura usando la fórmula dada
                temperature = (40.5844E-3) * dt[1] - 22.9740E1
```

```
            # Añadimos el dato de tiempo y temperatura a nuestra lista de datos
            data[0].append(dt[0])
            data[1].append(temperature)
            # Escribimos el dato de tiempo y temperatura en el archivo CSV
            writer.writerow({'timestamp': str(data[0][-1])[:-5], 'temperature': temperature})
            # Incrementamos el contador de mediciones
            cont += 1
            # Imprimimos el número de medición, el dato de tiempo y la temperatura
            print(f'{cont} {str(data[0][-1])[:-5]} {temperature:.1f}')

# Cerramos la comunicación con el dispositivo
mt.ClosePortLogg()

# Comenzamos a graficar
# Creamos una figura y un eje para la gráfica
fig, ax1 = plt.subplots()

# Definimos el color para la gráfica de temperatura
color = 'tab:red'
# Establecemos las etiquetas de los ejes
ax1.set_xlabel('Time')
ax1.set_ylabel('Temperature', color=color)
# Graficamos los datos de tiempo y temperatura
ax1.plot(data[0], data[1], color=color)
# Establecemos el color de las marcas del eje y
ax1.tick_params(axis='y', labelcolor=color)

# Creamos un segundo eje y para la derivada de la temperatura
ax2 = ax1.twinx()
# Definimos el color para la gráfica de la derivada
color = 'tab:blue'
# Establecemos la etiqueta del eje y para la derivada
ax2.set_ylabel('Derivative', color=color)
# Graficamos los datos de tiempo y la derivada de la temperatura
ax2.plot(data[0], np.gradient(data[1]), color=color)
# Establecemos el color de las marcas del eje y para la derivada
ax2.tick_params(axis='y', labelcolor=color)

# Mostramos la gráfica
plt.show()
```

Anexo 3



https://drive.google.com/file/d/1GVOC27yd9l4mpjpVKWqe_w96BAOL_tJW/view?usp=sharing