5-12-2025

Proyecto 2_AB Mediciòn de temperatura del aire

Integrantes del equipo:

Aguirre Yáñez Andrik Jesus

Cano García Eduardo

Hernández Pérez Uziel Celestino

Orozco García Julieta Elizabeth

Grado y grupo: 7º6

Docente: Efrén Fitz Rodríguez

Introducción

La medición de la temperatura del aire en entornos agrícolas y de invernadero es crítica para comprender procesos fisiológicos de cultivos, balance de energía y decisiones de manejo productivo; sin embargo, obtener lecturas correctas depende no solo del sensor utilizado sino también de las condiciones de radiación, ventilación y del tamaño del elemento sensible (Universidad Autónoma Chapingo, 2025). Por ello, los experimentos que comparan distintos escenarios de exposición y protección del sensor —sensor expuesto, sensor protegido pero ventilado, y sensor protegido con aspiración (psicrómetro)— permiten identificar sesgos sistémicos y cuantificar errores debidos a radiación de onda corta y larga, así como la influencia del flujo de aire sobre la señal térmica (WMO, 2008). Este proyecto replica esas tres condiciones simultáneamente para analizar la magnitud de las diferencias en la temperatura del bulbo seco (Tbs) y, en el escenario psicrométrico, la temperatura de bulbo húmedo (Tbh) y las propiedades psicrométricas derivadas (Universidad Autónoma Chapingo, 2025).

La instrumentación seleccionada combina termistores NTC de 10 kΩ como elementos sensibles principales y un sensor digital DHT11 como referencia de humedad y temperatura, lo que permite comparar mediciones analógicas (NTC) y digitales en condiciones reales de campo. Los NTC son apropiados por su alta sensibilidad y bajo costo, pero requieren modelado no lineal (por ejemplo, la ecuación de Steinhart–Hart) para convertir resistencia a temperatura con exactitud metrológica; además, su respuesta depende de montaje y flujo de aire, por lo que su protección física y la ventilación son factores determinantes en la calidad de la medición (Vishay Intertechnology, 2018; Steinhart & Hart, 1968). Complementariamente, el uso de un módulo RTC y almacenamiento en microSD garantiza trazabilidad temporal y persistencia de datos durante el muestreo prolongado requerido por la práctica (Maxim Integrated, 2015; Catalex Electronics, 2019).

La incorporación de sensores digitales como el DHT11 complementa el sistema al ofrecer una referencia rápida de temperatura y humedad, contribuyendo al proceso de verificación y control de calidad de las mediciones obtenidas mediante el método psicrométrico. De igual manera, el uso de módulos de reloj en tiempo real (RTC) y sistemas de almacenamiento en tarjeta microSD permite cumplir con los requisitos de registro continuo, indispensable para el análisis posterior y la comparación entre equipos, especialmente en contextos académicos y de investigación. La automatización de estos sistemas mediante microcontroladores como Arduino facilita la integración, despliegue y digitalización de los datos, asegurando su relevancia en el ámbito de los biosistemas inteligentes, donde la medición ambiental es un componente esencial para la toma de decisiones basada en datos (Maxim Integrated, 2015).

Materiales

Sensores:

- Termistores NTC 10 kΩ × 4 (NTC1 expuesto, NTC2 protegido, NTC3 psicrómetro seco, NTC4 psicrómetro húmedo).
- DHT11 (sensor integrado de T y RH).

Electrónica y módulos:

- Módulo microSD y tarjeta microSD.
- Módulo RTC (DS3231).
- Módulo LCD (I2C, 16×2 o 20×4).

Conectividad y prototipado:

- Placa microcontroladora (ESP32/Arduino compatible) — conexiones provistas abajo.
- Protoboard y cables DuPont.
- Resistencias 10 kΩ (divisor para cada NTC y pull-up del DHT11).

Alimentación y mecánica:

- Fuente de poder 3–5 V (según microcontrolador).
- Caja de cartón o botella plástica (para protección/radiación).
- Mecha textil y pequeño depósito de agua (para bulbo húmedo).
- Mechero/ventilador pequeño (para generar flujo > 3 m/s).
- Silicón para fijaciones.

Equipo adicional

- Computadora para programación.
- Regla, cutter, cinta adhesiva.

1.1 Diseño general del sistema

1. Selección y organización de sensores
Se emplean cuatro termistores NTC de 10 kΩ como sensores principales distribuidos en los tres escenarios de medición de temperatura:

- NTC 1 (Expuesto): mide la temperatura directamente bajo radiación.
- NTC 2 (Protegido): mide la temperatura dentro de un abrigo o pantalla de radiación.
- NTC 3 (Psicrómetro – Bulbo seco): mide la temperatura en instrumento ventilado.
- NTC 4 (Psicrómetro – Bulbo húmedo): envuelto con mecha húmeda para generar enfriamiento por evaporación.

Además, se utiliza un sensor DHT11 como instrumento de referencia para comparar valores de temperatura y humedad relativa entre los tres escenarios.

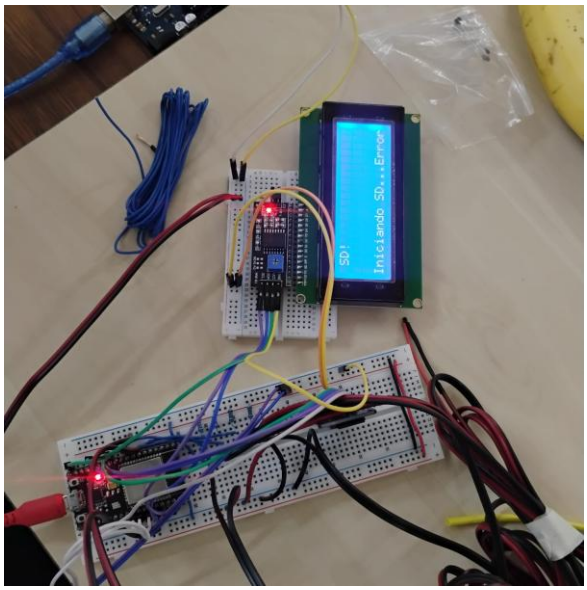


Fig 1. Circuito montado. Tomada por autores

2. Diseño del sistema de medición bajo tres escenarios

Para el NTC expuesto, el sensor se coloca directamente al aire libre, sin protección contra radiación, con el fin de registrar el efecto directo de la radiación solar en la medición de temperatura.

Para el NTC protegido, se emplea una caja de cartón modificada que actúa como abrigo de radiación, permitiendo la entrada de aire pero bloqueando radiación directa. Esta condición permite evaluar el impacto de sombras ventiladas sobre la precisión de la medición.

Para el escenario psicrométrico, el NTC destinado al bulbo húmedo se equipa con una mecha textil conectada a un depósito de agua elaborado con una botella reciclada, manteniendo humedad constante. Un ventilador pequeño provee un flujo de aire superior a 3 m/s, condición necesaria para que las mediciones de bulbo seco y bulbo húmedo sean confiables.

3. Programación y registro de datos

El sistema se programa para realizar un muestreo cada 1–2 segundos, almacenando temporalmente las lecturas en memoria volátil del microcontrolador.

Cada 10 minutos, se calcula un promedio de entre 600 y 3000 lecturas, el cual se almacena en la memoria microSD junto con la fecha y hora proporcionadas por el módulo RTC (DS3231).

Simultáneamente, la pantalla LCD I2C despliega en tiempo real las temperaturas de cada escenario (T expuesta, T protegida, Tbs, Tbh), además de la humedad relativa y la hora actual.

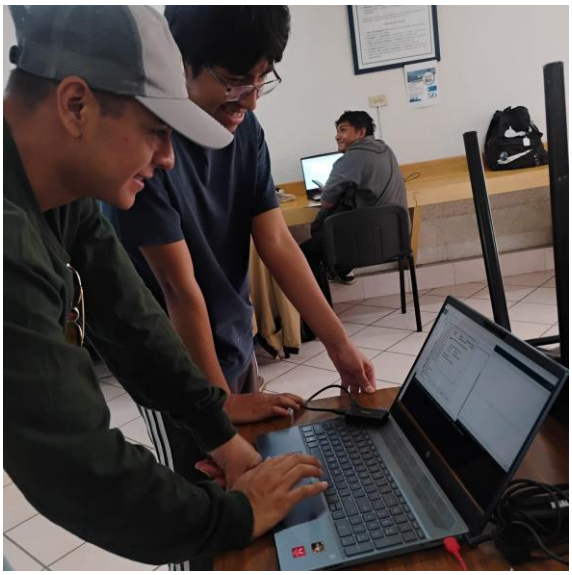


Fig 2. Programación y registro de datos. Tomada por autores

4. Instalación en campo

El sistema se instala a una distancia mínima de 2 metros del microcontrolador para evitar interferencias térmicas.
Los sensores son colocados en un área abierta para asegurar una adecuada ventilación, especialmente en el escenario psicrométrico.

Fig 3. Instalación en campo. Tomada por autores

1.2 Procedimiento experimental

1. Preparación de sensores

- Verificar la continuidad y el valor nominal de los NTC, confirmando que presenten aproximadamente 10 kΩ a 25 °C, acorde a sus hojas técnicas.
- Preparar la mecha del sensor de bulbo húmedo, asegurando que mantenga contacto constante con el agua del depósito.

2. Montaje y verificación del circuito

- Elaborar una simulación inicial del circuito en Proteus para validar conexiones.
- Revisar los divisores resistivos de cada NTC (resistencia fija de 10 kΩ por sensor).
- Conectar el DHT11, el RTC y el módulo SD según la distribución de pines proporcionada.
- Confirmar en el monitor serial que las lecturas de los NTC y del DHT11 son correctas antes de montar la carcasa física.

3. Calibración inicial

- Comparar las lecturas de los NTC con las del sensor digital DHT11 en un ambiente controlado (25–27 °C).
- Aplicar correcciones por software (offset) si existe una desviación sistemática en alguno de los sensores.
- Probar la ecuación de Steinhart–Hart para asegurar que los NTC reportan temperaturas consistentes.

4. Instalación en campo

- Fijar todos los sensores en sus respectivas configuraciones: expuesto, protegido y psicrómetro.
- Asegurar que exista flujo de aire constante en el escenario psicrométrico.
- Conectar el sistema a la fuente de alimentación y verificar actividad del RTC y correcto funcionamiento del módulo microSD.
- Iniciar el registro continuo según los parámetros del proyecto.

5. Comparación de resultados

- Con los datos que se obtuvieron de las mediciones, se hace una comparación con los datos medidos y proporcionados por el Doctor Efrén Fitz. Esta comparación se muestra en el apartado de resultados.

Nota: Cabe mencionar que esta tabla con los resultados se agregarà a los archivos a subir con el nombre de DatosProyecto_2.

Instrumentación

La instrumentación adopta cuatro termistores NTC de 10 kΩ como elementos sensibles principales para medir la temperatura del bulbo seco en los tres escenarios y la temperatura del bulbo húmedo en el psicrómetro; cada termistor se configura en un divisor resistivo con una resistencia fija de 10 kΩ para maximizar la sensibilidad alrededor de 25 °C y proporcionar una señal analógica proporcional a la resistencia del sensor, conforme a las recomendaciones técnicas de fabricantes de NTC y guías de diseño de sensores (Vishay Intertechnology, 2018; TDK/EPCOS, 2020).7

La conversión de la lectura analógica a temperatura se realiza siguiendo el flujo ADC → cálculo de resistencia → ecuación de Steinhart–Hart → conversión a °C; específicamente, tras leer el valor ADC se calcula la resistencia R_{NTC} mediante la relación:

$$R_{NTC} = R_{fijo}\left(\frac{1023}{adc} - 1\right) \text{ (para ADC de 10 bits)}$$

Ec 1. Resistencia R_{NTC}

Y posteriormente se aplica la ecuación:

$$\frac{1}{T} = A + B\ln(R) + C[\ln(R)]^3$$

Ec 2. Para obtener temperatura

Con los coeficientes A = 0.001129148, B = 0.000234125 y C = 0.0000000876741, para obtener la temperatura en kelvin y luego en °C; este enfoque proporciona la precisión requerida para análisis psicrométricos y es la práctica estándar en instrumentación con termistores (Steinhart & Hart, 1968; Arduino, 2019).

En el escenario psicrométrico, el termistor destinado al bulbo húmedo (NTC4) se envuelve con una mecha textil conectada por capilaridad a un depósito de agua reciclado, permitiendo la evaporación constante. El sistema se coloca en un recinto ventilado artificialmente con un ventilador que asegura un flujo de aire mayor a 3 m/s, condición indispensable para que la diferencia entre la temperatura del bulbo seco (NTC3) y del bulbo húmedo represente correctamente la depresión psicrométrica. Estas condiciones reproducen las recomendaciones establecidas para psicrómetros aspirados en mediciones meteorológicas. (WMO, 2008; Universidad Autónoma Chapingo, 2025).

Una vez obtenidas las temperaturas Tbs y Tbh mediante los termistores, la humedad relativa se calcula utilizando la ecuación psicrométrica operativa:

$$\mathrm{HR} = \frac{e_w - A\,P\,\Delta t}{e_d} \times 100,$$

Ec 3. Para calcular humedad relativa

En la cual e_w es la presión de vapor de saturación evaluada en la Tbh, e_d es la presión de saturación en Tbs, Δt es la depresión del bulbo (Tbs − Tbh), P es la presión atmosférica y A es la constante psicrométrica. Tanto e_w como $e_d$se obtienen mediante ecuaciones empíricas como la formulación de Magnus o Bolton, ampliamente aceptadas en el cálculo de variables psicrométricas. La constante psicrométrica depende de condiciones experimentales específicas, por lo que se adopta el valor estándar recomendado para psicrómetros ventilados. Esta ecuación permite obtener la humedad relativa a partir de las temperaturas medidas, cumpliendo con los estándares meteorológicos de cálculo. (Bolton, 1980; WMO, 2008).

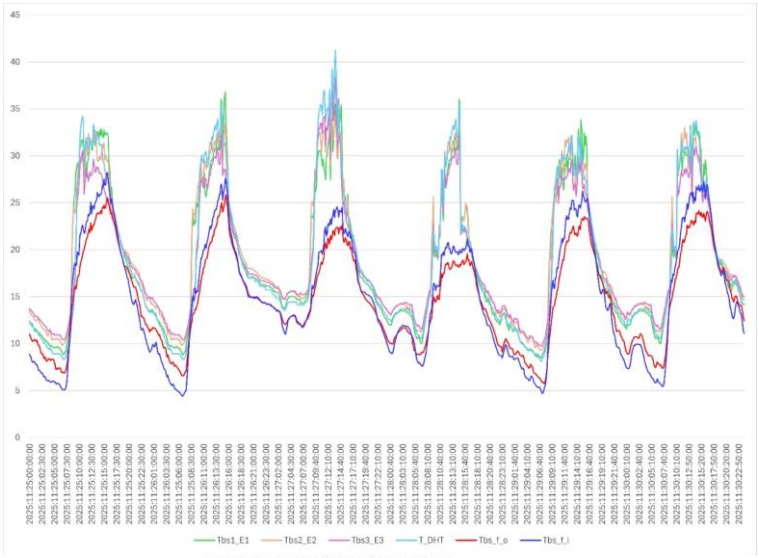
Como sensor de referencia se emplea el DHT11, conectado al pin digital asignado. Este sensor entrega valores rápidos de temperatura y humedad relativa con una precisión moderada, pero su estabilidad lo convierte en una herramienta útil para comparación y validación de las mediciones obtenidas mediante los termistores. (Aosong Electronics, 2018).

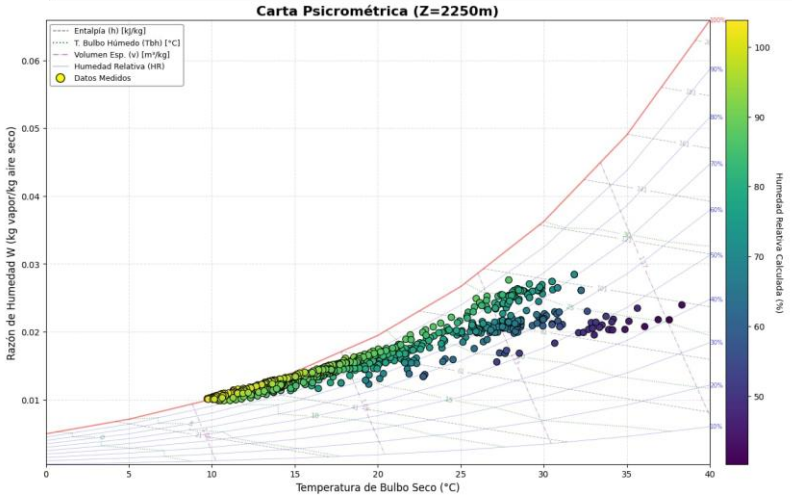
Para la sincronización temporal, el módulo RTC DS3231 se conecta mediante comunicación I2C, proporcionando marcas de tiempo precisas para cada lectura registrada. La deriva temporal del RTC es extremadamente baja (±2 ppm), lo que asegura la validez cronológica del registro continuo de datos. El módulo microSD, por su parte, se comunica mediante el protocolo SPI y permite almacenar miles de líneas de datos en formato .csv, facilitando el análisis posterior y garantizando non-volatilidad. (Maxim Integrated, 2015; Catalex Electronics, 2019).

Finalmente, la pantalla LCD I2C se utiliza para mostrar en tiempo real los valores de temperatura de los cuatro termistores, la humedad relativa calculada, así como la fecha y hora actual del RTC. Su uso reduce la necesidad de monitoreo mediante computadora y facilita la verificación de campo. La comunicación I2C permite compartir el bus con el RTC sin interferencias, simplificando el cableado. Los sensores se montan a más de 2 m del microcontrolador para evitar interferencias térmicas y se instalan en campo siguiendo los criterios de ventilación y protección requeridos por cada escenario. (Hitachi, 2006; Universidad Autónoma Chapingo, 2025).

NTC1 (expuesto) → Pin 33; NTC2 (protegido) → Pin 25; NTC3 (psicrómetro seco) → Pin 35; NTC4 (psicrómetro húmedo) → Pin 34; DHT11 → Pin 32; SD (MOSI, MISO, SCK, CS) → Pins 12,13,14,15 (CS); RTC (I2C) → Pins 18 (SDA), 19 (SCL), + alimentación en Pin 5; LCD (I2C) → SDA Pin 21, SCL Pin 22. Este mapeo debe implementarse en el firmware respetando las librerías SPI / I2C del microcontrolador seleccionado y asegurando que los niveles lógicos y referencias de voltaje sean compatibles entre módulos (Arduino, 2019; Maxim Integrated, 2015).

Resultados





La gráfica ilustra el comportamiento psicrométrico del aire en la zona de estudio basándose en las mediciones de temperatura de bulbo seco (T_{bs}) y bulbo húmedo (T_{bh}). Destacan los siguientes fenómenos:

1. Se observa una tendencia entre la temperatura de bulbo seco y la humedad relativa. A temperaturas bajas ($10 - 15°C$), la masa de aire se encuentra próxima a la saturación ($HR > 90\%$), mientras que, al aumentar la temperatura ($> 25°C$), la capacidad de contener agua del aire aumenta, provocando un descenso en la humedad relativa hasta el $20 - 30\%$.

2. En el extremo inferior izquierdo de la curva, los datos convergen con la línea de saturación ($HR = 100\%$). Esto indica que, durante las temperaturas mínimas registradas, el aire alcanzó su temperatura de punto de rocío.

3. A diferencia de un proceso de calentamiento sensible puro (que sería una línea horizontal), la nube de puntos muestra una pendiente positiva ascendente. La razón de humedad aumenta de $\approx 0.010\ a\ 0.025$ kg_v/kg_{as} conforme aumenta la temperatura.

Conclusión

La instrumentación y evaluación de los sensores en los tres escenarios de medición permitieron analizar de manera precisa cómo las condiciones ambientales y físicas influyen directamente en la lectura de la temperatura del aire. La comparación entre un sensor expuesto, uno protegido y un sistema psicrométrico ventilado evidenció que la radiación solar y la falta de ventilación pueden generar sesgos significativos en la medición, mientras que el uso del psicrómetro con flujo de aire controlado proporciona valores más representativos del estado real del ambiente. El empleo de termistores NTC de 10 kΩ junto con la ecuación de Steinhart–Hart permitió obtener temperaturas confiables, mientras que la aplicación de la ecuación psicrométrica para calcular la humedad relativa, complementada por la lectura del DHT11, fortaleció la validación de los resultados. En conjunto, el sistema instrumentado demostró ser una herramienta accesible, funcional y precisa para el análisis microclimático, además de representar una experiencia fundamental en el desarrollo de habilidades de medición, automatización y procesamiento de datos dentro del área de biosistemas.

Bibliografía

Aosong Electronics. (2018). *DHT11 humidity & temperature sensor datasheet*. Aosong Electronics Co. Ltd.

Arduino. (2019). *Analog input pins* [Documentation]. https://www.arduino.cc

Bolton, D. (1980). The computation of equivalent potential temperature. *Monthly Weather Review, 108*(7), 1046–1053.

Bureau of Meteorology. (2003). *Psychrometric equations and humidity calculations*. Australian Government.

Catalex Electronics. (2019). *Micro SD card module datasheet*. Catalex Technology Ltd.

Hitachi. (2006). *HD44780U LCD controller/driver datasheet*. Hitachi Semiconductor.

Steinhart, J. S., & Hart, S. R. (1968). Calibration curves for thermistors. *Journal of Research of the National Bureau of Standards, 73*(1), 27–35.

Stull, R. (2011). *Meteorology for scientists and engineers* (3rd ed.). University of British Columbia Press.

TDK/EPCOS. (2020). *NTC thermistors – Application notes*. TDK Corporation.

Universidad Autónoma Chapingo. (2025). *Proyecto 02 – Medición de temperatura en tres escenarios* [Guía de laboratorio].

Vishay Intertechnology. (2018). *NTC thermistors: Selection guide*. Vishay Intertechnology Inc.

World Meteorological Organization. (2008). *Guide to meteorological instruments and methods of observation* (WMO-No. 8). WMO.

Anexos

Còdigo carta_psricromètrica 2

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D  # Necesario para la leyenda personalizada
import numpy as np
import csv
import os
from calculos import CalculadoraPsicrometrica  # Asegúrate de importar la clase correcta

# ========= CONFIGURACIÓN DE LA CARTA =========
Z = 2250  # (msnm)
Tbs_vals = [t for t in range(0, 45, 5)]  # Rango eje X
phi_vals = [i / 100 for i in range(10, 101, 10)]  # Curvas de HR

# ========= FUNCIÓN GENERADORA DE LA CARTA (FONDO) =========
def generar_carta_psicrometrica(z, Tbs_vals, phi_vals):
    datos = {"W": [], "H": [], "Veh": [], "Tbh": []}
    for hr in phi_vals:
        Ws, Hs, Vhs, Tbh_lineas = [], [], [], []
        for T in Tbs_vals:
            calc = CalculadoraPsicrometrica(z, T, hr)
            res = calc.calcular_todo()
            Ws.append(res["W_kgkg"])
            Hs.append(res["h_kJkg"])
            Vhs.append(res["veh_m3kg"])
            Tbh_lineas.append(res["Tbh_C"])
        datos["W"].append(Ws)
        datos["H"].append(Hs)
        datos["Veh"].append(Vhs)
        datos["Tbh"].append(Tbh_lineas)
    return datos

# ========= FUNCIÓN DE LECTURA (Tbs y Tbh) =========
def leer_tbs_tbh(filepath):
    tbs_list = []
    tbh_list = []
    nombres_tbs = ['tbs', 't_seca', 't_bulbo_seco', 'temp', 't']
    nombres_tbh = ['tbh', 't_humeda', 't_bulbo_humedo', 'wet', 'tw']

    if not os.path.exists(filepath):
        print(f"Error: El archivo {filepath} no existe.")
        return [], []

    with open(filepath, 'r', newline='', encoding='utf-8') as f:
        sample = f.read(1024)
        delim = ';' if ';' in sample and ',' not in sample else ','
        f.seek(0)
        reader = csv.reader(f, delimiter=delim)
        rows = list(reader)
        if not rows: return [], []

        header = [h.strip().lower() for h in rows[0]]
        idx_tbs = -1
        idx_tbh = -1

        for i, col in enumerate(header):
            if col in nombres_tbs: idx_tbs = i
```

```
            if col in nombres_tbh: idx_tbh = i

        if idx_tbs == -1: idx_tbs = 0
        if idx_tbh == -1: idx_tbh = 1

        for row in rows[1:]:
            try:
                if len(row) > max(idx_tbs, idx_tbh):
                    val_tbs = float(row[idx_tbs])
                    val_tbh = float(row[idx_tbh])
                    tbs_list.append(val_tbs)
                    tbh_list.append(val_tbh)
            except ValueError:
                continue
    return tbs_list, tbh_list

# ========= PROCESAMIENTO PRINCIPAL =========
print("Generando carta psicrométrica...")
carta = generar_carta_psicrometrica(Z, Tbs_vals, phi_vals)

archivo_datos = "datos_temperatura_p2.csv"
tbs_exp, tbh_exp = leer_tbs_tbh(archivo_datos)

w_calc = []
hr_calc = []

print(f"Procesando {len(tbs_exp)} puntos experimentales...")
for t_seca, t_hum in zip(tbs_exp, tbh_exp):
    calc = CalculadoraPsicrometrica(Z, t_seca, 0)
    hr_reales = calc.calcular_hr_psicrometrica(t_hum)
    w = calc.calcular_razon_humedad()
    w_calc.append(w)
    hr_calc.append(hr_reales)

# ========= GRAFICAR =========
plt.figure(figsize=(14, 10))
plt.title(f"Carta Psicrométrica (Z={Z}m)", fontsize=16, fontweight='bold')
plt.xlabel("Temperatura de Bulbo Seco (°C)", fontsize=12)
plt.ylabel("Razón de Humedad W (kg vapor/kg aire seco)", fontsize=12)

# --- 1. Dibujar líneas de HR (Fondo) ---
for i, hr in enumerate(phi_vals):
    color_linea = 'blue' if hr < 1.0 else 'red'
    grosor = 1.5 if hr == 1.0 else 0.6
    alpha_val = 0.3 if hr < 1.0 else 0.5
    plt.plot(Tbs_vals, carta["W"][i], color=color_linea, alpha=alpha_val,
linewidth=grosor)

    if len(carta["W"][i]) > 0:
        plt.text(Tbs_vals[-1], carta["W"][i][-1], f"{int(hr * 100)}%",
                 fontsize=7, color=color_linea, alpha=0.7, verticalalignment='center')

# Preparación para contornos
H = np.array(carta["H"])
W = np.array(carta["W"])
T, PHI = np.meshgrid(Tbs_vals, phi_vals)

# --- 2. LÍNEAS DE ENTALPÍA (h) ---
```

```
niveles_h = np.arange(np.nanmin(H), np.nanmax(H), 20)
cs_h = plt.contour(T, W, H, niveles_h, colors='black', linestyles='--', alpha=0.3,
linewidths=0.8)
plt.clabel(cs_h, inline=True, fmt='%d', fontsize=7, colors='black', manual=False)

# --- 3. LÍNEAS DE BULBO HÚMEDO (Tbh) ---
Tbh = np.array(carta["Tbh"])
niveles_tbh = np.arange(0, 35, 5)
cs_tbh = plt.contour(T, W, Tbh, niveles_tbh, colors='green', linestyles=':', alpha=0.5,
linewidths=1)

# CORRECCIÓN AQUÍ: Eliminamos "fontweight='bold'" que causaba el error
plt.clabel(cs_tbh, inline=True, fmt='%d', fontsize=8, colors='green')

# --- 4. LÍNEAS DE VOLUMEN ESPECÍFICO (Veh) ---
Veh = np.array(carta["Veh"])
niveles_v = np.linspace(np.nanmin(Veh), np.nanmax(Veh), 6)
cs_v = plt.contour(T, W, Veh, niveles_v, colors='purple', linestyles='-.', alpha=0.4,
linewidths=0.8)
plt.clabel(cs_v, inline=True, fmt='%.2f', fontsize=7, colors='purple')

# --- 5. LEYENDA PERSONALIZADA ---
from matplotlib.lines import Line2D

leyenda_elementos = [
    Line2D([0], [0], color='black', linestyle='--', linewidth=1, alpha=0.6,
label='Entalpía (h) [kJ/kg]'),
    Line2D([0], [0], color='green', linestyle=':', linewidth=1.5, alpha=0.8, label='T.
Bulbo Húmedo (Tbh) [°C]'),
    Line2D([0], [0], color='purple', linestyle='-.', linewidth=1, alpha=0.6,
label='Volumen Esp. (v) [m³/kg]'),
    Line2D([0], [0], color='blue', lw=1, alpha=0.3, label='Humedad Relativa (HR)'),
    Line2D([0], [0], marker='o', color='w', markerfacecolor='yellow',
markeredgecolor='black', markersize=10,
           label='Datos Medidos')
]

plt.legend(handles=leyenda_elementos, loc='upper left', fontsize=9, framealpha=0.9,
edgecolor='gray')

# --- 6. Graficar tus puntos experimentales ---
if len(tbs_exp) > 0:
    sc = plt.scatter(tbs_exp, w_calc, c=hr_calc, cmap='viridis',
                     s=60, edgecolors='black', linewidths=0.8, zorder=10)

    cbar = plt.colorbar(sc, pad=0.02)
    cbar.set_label("Humedad Relativa Calculada (%)", rotation=270, labelpad=15)
else:
    print("Advertencia: No se encontraron datos para graficar.")

plt.grid(True, linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```

Código de cálculos

```
import math
import csv
```

```
import os
import numpy as np
import matplotlib.pyplot as plt

class CalculadoraPsicrometrica:
    """
    Clase para realizar cálculos psicrométricos del aire húmedo (por muestra).
    RA en J/kg*K.
    """

    RA = 287.055  # J/kg*K, Constante del Gas para el Aire Seco

    def __init__(self, z, tbs, hr):
        """
        Args:
            z (float): Altitud sobre el nivel del mar (m)
            tbs (float): Temperatura de bulbo seco (°C)
            hr (float): Humedad relativa (0-1) o (0-100).
                        NOTA: Si vas a calcular HR desde bulbo húmedo,
                    puedes iniciar esto con 0 y luego llamar a calcular_hr_psicrometrica.
        """
        self.z = float(z)
        self.tbs = float(tbs)
        # Normalizar hr: si viene mayor a 1 se asume porcentaje 0-100
        self.hr = float(hr) / 100.0 if float(hr) > 1.0 else float(hr)
        self.patm = None
        self.tbsk = None
        self.pv = None
        self.pvs = None
        self.dpva = None
        self.w = None
        self.ws = None
        self.mu = None
        self.veh = None
        self.h = None
        self.tpr = None
        self.tbh = None  # Temperatura de bulbo húmedo

    def calcular_presion_atmosferica(self):
        """Calcula la presión atmosférica en kPa en función de la altitud (msnm)."""
        self.patm = 101.325 * (1 - (2.25577 * 10 ** -5) * self.z) ** 5.2529
        return self.patm

    def convertir_temperatura_kelvin(self):
        """Convierte la temperatura de Celsius a Kelvin."""
        self.tbsk = 273.15 + self.tbs
        return self.tbsk

    def calcular_presion_vapor_saturado(self, temperatura=None):
        """
        Calcula la presión de vapor saturado.
        Devuelve pvs en Pa.
        """
        if temperatura is None:
            temperatura = self.tbs
            temp_k = self.tbsk if self.tbsk is not None else 273.15 + self.tbs
        else:
            temp_k = 273.15 + temperatura
```

```
        if -100 < temperatura < 0:
            pvs = math.exp(
                (-(5.6745359 * 10 ** 3) / temp_k) + 6.3925247 -
                ((9.6778430 * 10 ** -3) * temp_k) +
                ((6.2215701 * 10 ** -7) * (temp_k) ** 2) +
                ((2.0747825 * 10 ** -9) * (temp_k) ** 3) -
                ((9.484024 * 10 ** -13) * (temp_k) ** 4) +
                (4.1635019 * math.log(temp_k))
            )
        elif 0 <= temperatura < 200:
            pvs = math.exp(
                (-(5.8002206 * 10 ** 3) / temp_k) + 1.3914993 -
                ((48.640239 * 10 ** -3) * temp_k) +
                ((41.764768 * 10 ** -6) * (temp_k) ** 2) -
                ((14.452093 * 10 ** -9) * (temp_k) ** 3) +
                (6.5459673 * math.log(temp_k))
            )
        else:
            raise ValueError(f"Temperatura {temperatura}°C fuera del rango válido (-100 a
200°C)")

        # Guardar en self.pvs solo si se calculó para la temperatura principal
        if temperatura == self.tbs:
            self.pvs = pvs

        return pvs

    def calcular_presion_vapor(self):
        """Calcula la presión de vapor Pv en Pa usando hr * pvs."""
        if self.pvs is None:
            self.calcular_presion_vapor_saturado()
        self.pv = self.hr * self.pvs
        return self.pv

    def calcular_deficit_presion_vapor(self):
        """Calcula el déficit de presión de vapor (Pvs - Pv) en Pa."""
        if self.pvs is None:
            self.calcular_presion_vapor_saturado()
        if self.pv is None:
            self.calcular_presion_vapor()
        self.dpva = self.pvs - self.pv
        return self.dpva

    def calcular_razon_humedad(self, presion_vapor=None, presion_atmosferica=None):
        """
        Calcula la razón de humedad W (kg_vapor/kg_aire_seco).
        presion_vapor en Pa (si no se pasa, usa self.pv).
        presion_atmosferica en kPa (si no se pasa, usa self.patm).
        """
        if presion_vapor is None:
            if self.pv is None:
                self.calcular_presion_vapor()
            presion_vapor = self.pv
        if presion_atmosferica is None:
            if self.patm is None:
                self.calcular_presion_atmosferica()
            presion_atmosferica = self.patm
```

```
        pv_kpa = presion_vapor / 1000
        return 0.621945 * (pv_kpa / (presion_atmosferica - pv_kpa))

    def calcular_grado_saturacion(self):
        """Calcula mu = W/Ws."""
        if self.w is None:
            self.w = self.calcular_razon_humedad(self.pv, self.patm)
        if self.ws is None:
            if self.pvs is None:
                self.calcular_presion_vapor_saturado()
            self.ws = self.calcular_razon_humedad(self.pvs, self.patm)
        self.mu = self.w / self.ws if self.ws != 0 else None
        return self.mu

    def calcular_volumen_especifico(self):
        """Calcula el volumen específico del aire húmedo (m3/kg_as)."""
        if self.tbsk is None:
            self.convertir_temperatura_kelvin()
        if self.patm is None:
            self.calcular_presion_atmosferica()
        if self.w is None:
            self.w = self.calcular_razon_humedad()

        # patm en kPa -> convertir a Pa multiplicando por 1000
        self.veh = ((self.RA * self.tbsk) / (self.patm * 1000.0)) * ((1 + 1.6087 * self.w)
/ (1 + self.w))
        return self.veh

    def calcular_entalpia(self, temperatura=None, razon_humedad=None):
        """Calcula la entalpía en kJ/kg (valores aproximados)."""
        if temperatura is None:
            temperatura = self.tbs
        if razon_humedad is None:
            if self.w is None:
                self.w = self.calcular_razon_humedad()
            razon_humedad = self.w

        self.h = (1.006 * temperatura) + razon_humedad * (2501 + 1.805 * temperatura)
        return self.h

    def calcular_temperatura_punto_rocio(self):
        """Calcula temperatura del punto de rocío Tpr en ºC (aprox.)"""
        if self.pv is None:
            self.calcular_presion_vapor()

        # Estas fórmulas usan ln(pv) con pv en Pa
        if -60 < self.tbs < 0:
            self.tpr = -60.450 + 7.0322 * math.log(self.pv) + 0.3700 * (math.log(self.pv))
** 2
        elif 0 <= self.tbs < 70:
            self.tpr = -35.957 - 1.8726 * math.log(self.pv) + 1.1689 * (math.log(self.pv))
** 2
        else:
            # Si está fuera de rango, lo dejamos None pero no rompemos
            self.tpr = None
        return self.tpr
```

```
    def calcular_temperatura_bulbo_humedo(self, tolerancia=0.001, max_iteraciones=100):
        """
        Calcula la temperatura de bulbo húmedo (Tbh) por bisección (si tienes HR y buscas
Tbh).
        Devuelve °C.
        """
        if self.patm is None:
            self.calcular_presion_atmosferica()
        if self.w is None:
            self.calcular_presion_vapor()
            self.w = self.calcular_razon_humedad(self.pv, self.patm)

        def funcion_objetivo(tbh_prueba):
            pvs_tbh = self.calcular_presion_vapor_saturado(tbh_prueba)
            ws_tbh = self.calcular_razon_humedad(pvs_tbh, self.patm)
            # Ecuación psicrométrica aproximada (unidades SI)
              numerador = ((2501 - 2.326 * tbh_prueba) * ws_tbh - 1.006 * (self.tbs -
tbh_prueba))
            denominador = (2501 + 1.86 * self.tbs - 4.186 * tbh_prueba)
            w_calculada = numerador / denominador
            return w_calculada - self.w

        tbh_min = -50.0
        tbh_max = self.tbs
        f_min = funcion_objetivo(tbh_min)
        f_max = funcion_objetivo(tbh_max)

        if f_min * f_max > 0:
            # fallback empírico si no cambia de signo
            self.tbh = self.tbs - (1 - self.hr) * (self.tbs - 14) / 3
            return self.tbh

        iteracion = 0
        while iteracion < max_iteraciones:
            tbh_prueba = (tbh_min + tbh_max) / 2.0
            error = funcion_objetivo(tbh_prueba)
            if abs(error) < tolerancia:
                self.tbh = tbh_prueba
                return self.tbh
            if error > 0:
                tbh_max = tbh_prueba
            else:
                tbh_min = tbh_prueba
            iteracion += 1

        self.tbh = (tbh_min + tbh_max) / 2.0
        return self.tbh

    # ------------------------------------------------------------------------
    # NUEVO MÉTODO AGREGADO: CALCULAR HR DESDE BULBO HÚMEDO
    # ------------------------------------------------------------------------
    def calcular_hr_psicrometrica(self, t_humedo):
        """
        Calcula la Humedad Relativa (HR) y Presión de Vapor (Pv) partiendo de
        la temperatura de bulbo húmedo conocida.

        Args:
            t_humedo (float): Temperatura de bulbo húmedo (°C)
```

```
        Returns:
            float: Humedad Relativa en %
        """
        # Asegurarnos de tener la presión atmosférica calculada
        if self.patm is None:
            self.calcular_presion_atmosferica()

        # Convertir Patm de kPa (usado en la clase) a Pa (necesario para la fórmula)
        patm_pa = self.patm * 1000.0

        # 1. Calcular Pvs a la temperatura de BULBO HÚMEDO (e'_w)
        pvs_humedo = self.calcular_presion_vapor_saturado(temperatura=t_humedo)

        # 2. Calcular Pvs a la temperatura de BULBO SECO (e_s)
        # Nota: llamamos al método interno, no sobreescribimos self.pvs principal todavía
        pvs_seco = self.calcular_presion_vapor_saturado(temperatura=self.tbs)

        # 3. Constante psicrométrica (A)
        # 0.000666 para ventilación forzada (velocidad > 2.5 m/s)
        # 0.000800 para ventilación natural
        # Ajusta según tu sensor. Asumimos 0.000666 estándar aspiro-psicrómetro.
        A = 0.000666

        # 4. Calcular Presión de Vapor Real (Pv) usando Ecuación Psicrométrica
        # Pv = Pvs_hum - A * Patm * (Tbs - Tbh)
        pv_real = pvs_humedo - A * patm_pa * (self.tbs - t_humedo)

        # Guardar valores calculados en la instancia
        self.pv = pv_real
        self.tbh = t_humedo
        self.pvs = pvs_seco  # Actualizamos pvs de la clase al de bulbo seco

        # 5. Calcular HR = Pv / Pvs_seco
        if pvs_seco != 0:
            self.hr = pv_real / pvs_seco
        else:
            self.hr = 0

        # Retornar porcentaje
        return self.hr * 100

    def calcular_todo(self):
        """
        Ejecuta todos los cálculos en orden y devuelve un diccionario de resultados.
        """
        try:
            self.calcular_presion_atmosferica()
            self.convertir_temperatura_kelvin()
            self.calcular_presion_vapor_saturado()

            # Si YA tenemos Pv calculado (por ejemplo, usando calcular_hr_psicrometrica
antes),
            # no necesitamos recalcularlo con HR.
            if self.pv is None:
                self.calcular_presion_vapor()

            self.calcular_deficit_presion_vapor()
```

```
            self.w = self.calcular_razon_humedad(self.pv, self.patm)
            self.ws = self.calcular_razon_humedad(self.pvs, self.patm)

            self.calcular_grado_saturacion()
            self.calcular_volumen_especifico()
            self.calcular_entalpia()
            self.calcular_temperatura_punto_rocio()

            # Si no tenemos Tbh (porque entramos con HR), la calculamos iterativamente
            if self.tbh is None:
                self.calcular_temperatura_bulbo_humedo()

            return {
                'patm_kPa': self.patm,
                'pv_Pa': self.pv,
                'pvs_Pa': self.pvs,
                'dpva_Pa': self.dpva,
                'W_kgkg': self.w,
                'Ws_kgkg': self.ws,
                'mu': self.mu,
                'veh_m3kg': self.veh,
                'h_kJkg': self.h,
                'Tpr_C': self.tpr,
                'Tbh_C': self.tbh
            }

        except Exception as e:
            raise RuntimeError(f"Error en los cálculos: {e}")

# -----------------------
# Funciones auxiliares para procesamiento vectorial y archivos
# -----------------------

def _ensure_list(x):
    """Si x no es iterable (o es str), lo convierte en lista de un elemento."""
    if x is None:
        return []
    if isinstance(x, (list, tuple)):
        return list(x)
    # strings considered single scalar
    return [x]

def calcular_vectorial(z, tbs_list, hr_list):
    """
    Calcula propiedades psicrométricas para listas de tbs y hr.
    Args:
        z (float): elevación msnm
        tbs_list (iterable): temperaturas bulbo seco (°C)
        hr_list (iterable): humid relativa (0-1 o 0-100)
    Returns:
        list of dicts: resultados por muestra
    """
    tbs_l = _ensure_list(tbs_list)
    hr_l = _ensure_list(hr_list)

    if len(tbs_l) != len(hr_l):
        raise ValueError("tbs_list y hr_list deben tener la misma longitud.")
```

```
    resultados = []
    for tbs, hr in zip(tbs_l, hr_l):
        calc = CalculadoraPsicrometrica(z, tbs, hr)
        res = calc.calcular_todo()
        # añadir datos de entrada para referencia
        res['Tbs_C'] = float(tbs)
        res['HR_frac'] = float(hr) / 100.0 if float(hr) > 1.0 else float(hr)
        resultados.append(res)
    return resultados

def leer_csv_o_txt(filepath, delim=None, encabezados_esperados=None):
    """
    Lee un CSV o TXT delimitado y busca columnas para Tbs y HR.
    """
    if encabezados_esperados is None:
        encabezados_esperados = {
            'tbs': ['Tbs', 'tbs', 'T_bulbo_sec', 'Tbulbo', 'T'],
            'hr': ['HR', 'hr', 'Humedad', 'humedad', 'phi', 'phi%', 'hum%']
        }

    with open(filepath, 'r', newline='', encoding='utf-8') as f:
        sample = f.read(2048)
        f.seek(0)
        # detectar delimitador sencillo
        if delim is None:
            if ',' in sample:
                delim = ','
            elif '\t' in sample:
                delim = '\t'
            else:
                delim = ','

        reader = csv.reader(f, delimiter=delim)
        rows = list(reader)
        if not rows:
            return [], []

        header = [h.strip() for h in rows[0]]
        # intentar identificar columnas por nombre
        col_tbs = None
        col_hr = None
        for i, col in enumerate(header):
            low = col.lower()
            for candidate in encabezados_esperados['tbs']:
                if candidate.lower() == low:
                    col_tbs = i
            for candidate in encabezados_esperados['hr']:
                if candidate.lower() == low:
                    col_hr = i

        data_rows = rows[1:] if col_tbs is not None and col_hr is not None else rows  #
si no hay header, leer todo
        tbs_list = []
        hr_list = []
        for r in data_rows:
            # proteger si fila corta
            if col_tbs is not None and col_hr is not None:
```

```
                try:
                    tbs_list.append(float(r[col_tbs]))
                    hr_list.append(float(r[col_hr]))
                except Exception:
                    # intentar saltar filas corruptas
                    continue
            else:
                # si no hay encabezado, intentar tomar las primeras 2 columnas como tbs,hr
                if len(r) >= 2:
                    try:
                        tbs_list.append(float(r[0]))
                        hr_list.append(float(r[1]))
                    except Exception:
                        continue

    return tbs_list, hr_list

def procesar_archivo(filepath, z, delim=None, encabezados_esperados=None, guardar_salida=None):
    """
    Lee un archivo (CSV/TXT). Devuelve resultados vectoriales.
    Si el archivo es .xls/.xlsx intentará usar openpyxl/xlrd si están disponibles.
    guardar_salida: ruta de archivo CSV donde escribir resultados (opcional).
    """
    ext = os.path.splitext(filepath)[1].lower()
    if ext in ['.csv', '.txt']:
        tbs_list, hr_list = leer_csv_o_txt(filepath, delim=delim, encabezados_esperados=encabezados_esperados)
    elif ext in ['.xls', '.xlsx']:
        # intentar usar librería si está instalada
        try:
            if ext == '.xlsx':
                import openpyxl
                wb = openpyxl.load_workbook(filepath, read_only=True, data_only=True)
                ws = wb.active
                rows = list(ws.values)
                # asumir primera fila header si strings
                header = rows[0]
                tbs_list = []
                hr_list = []
                # intentar detectar columnas por nombre
                headers = [str(h).strip().lower() if h is not None else '' for h in header]
                col_tbs = None
                col_hr = None
                for i, col in enumerate(headers):
                    if col in ('tbs', 't', 't_bulbo_sec', 'tbulbo'):
                        col_tbs = i
                    if col in ('hr', 'humedad', 'phi', 'phi%'):
                        col_hr = i
                for r in rows[1:]:
                    try:
                        if col_tbs is not None and col_hr is not None:
                            tbs_list.append(float(r[col_tbs]))
                            hr_list.append(float(r[col_hr]))
                        else:
                            tbs_list.append(float(r[0]))
                            hr_list.append(float(r[1]))
                    except Exception:
```

```
                    continue
            else:
                import xlrd
                wb = xlrd.open_workbook(filepath)
                sh = wb.sheet_by_index(0)
                rows = [sh.row_values(i) for i in range(sh.nrows)]
                        tbs_list,  hr_list  =  leer_csv_o_txt(filepath,  delim=delim,
encabezados_esperados=encabezados_esperados)
        except Exception:
            raise RuntimeError("No se pudo leer archivo Excel: instala 'openpyxl' (xlsx)
o convierte el archivo a CSV.")
    else:
       raise ValueError("Extensión no soportada. Use .csv, .txt, .xls o .xlsx (o convierta
a .csv).")

    # ahora calcular vectorial
    resultados = calcular_vectorial(z, tbs_list, hr_list)

    # si se solicita guardar salida, escribir CSV con columnas ordenadas
    if guardar_salida:
        campos = ['Tbs_C', 'HR_frac', 'patm_kPa', 'pv_Pa', 'pvs_Pa', 'dpva_Pa',
                  'W_kgkg', 'Ws_kgkg', 'mu', 'veh_m3kg', 'h_kJkg', 'Tpr_C', 'Tbh_C']
        with open(guardar_salida, 'w', newline='', encoding='utf-8') as f:
            writer = csv.DictWriter(f, fieldnames=campos)
            writer.writeheader()
            for r in resultados:
                # filtrar sólo campos en 'campos'
                fila = {k: (r.get(k) if k in r else None) for k in campos}
                writer.writerow(fila)

    return resultados
```