Proyecto final
Control de invernadero

Cruz Calderón Jorge Luis, Frías Hernández Camille Emille Román,
Sánchez Estrada Angel Isaac

El presente proyecto busca gestionar de manera eficiente las condiciones internas del entorno controlado, asegurando la regulación de temperatura, la irrigación y el flujo de aire, mediante la integración de sensores y actuadores.

El sistema contará con funciones automatizadas, como el encendido y apagado del sistema de irrigación, el control de temperatura a través de un controlador PID, y la administración de la velocidad del ventilador mediante señales PWM. Adicionalmente, se implementará un servidor web que permitirá el acceso remoto a las funciones del invernadero y la consulta de gráficas históricas que registran las variables monitoreadas y las acciones realizadas.

El proyecto incluye tanto el diseño de hardware, que comprende la conexión de sensores y actuadores, como el desarrollo de software para la programación del sistema embebido, garantizando una solución práctica, robusta y accesible para usuarios con conocimientos básicos en electrónica y sistemas embebidos.

1. Objetivos

- Diseñar e implementar un sistema embebido que permita la supervisión y control remoto de un invernadero más enfocado en las plantas suculentas, integrando sensores y actuadores para gestionar de manera precisa las variables ambientales internas.
- Desarrollar una solución de software y hardware que incorpore un controlador PID para regular la temperatura, así como señales PWM para administrar la velocidad del ventilador, asegurando un funcionamiento eficiente y confiable.
- Proveer un sistema de visualización de datos históricos mediante gráficas almacenadas, además de un servidor web local, para facilitar el análisis y la toma de decisiones sobre el funcionamiento del invernadero.

2. Antecedentes

Adaptabilidad y Eficiencia de las Suculentas en Espacios Reducidos: Diseño y Gestión de un Invernadero Controlado

En este proyecto se plantea usar la planta suculenta debido a los cuidados que requiere, ya que se caracteriza por su bajo consumo de recursos, tanto naturales como de mantenimiento. Esta elección resulta particularmente adecuada para un entorno de estudiante o en espacios reducidos, como un departamento, donde el tiempo y los recursos disponibles para el cuidado de plantas suelen ser limitados.

Las suculentas, gracias a su capacidad natural para almacenar agua en sus hojas, tallos o raíces, prosperan con un riego esporádico y no necesitan de un sustrato complejo, lo que facilita su cuidado. Además, estas plantas toleran ambientes con baja humedad relativa, entre el 10 % y el 30 %, y se adaptan fácilmente a invernaderos con condiciones controladas, sin requerir sistemas complicados de irrigación

o ventilación. Estas características las hacen ideales para espacios donde la optimización de recursos es crucial (Nabors, 2006).

El diseño de un invernadero para plantas suculentas debe considerar los aspectos clave de ventilación, irrigación y control de temperatura, ya que estos influyen directamente en su crecimiento.

Las suculentas requieren una iluminación intensa para un crecimiento óptimo. El uso de una lámpara incandescente de 100W proporciona la intensidad lumínica necesaria. Sin embargo, estas lámparas emiten una cantidad significativa de calor, lo que puede dañar las plantas si se colocan demasiado cerca. Por lo que, una lámpara incandescente de 100W puede colocarse a una distancia de 30cm de las plantas, proporcionando la intensidad lumínica adecuada sin el riesgo de sobrecalentamiento (Saltón Verde, 2022).

Las suculentas prosperan en un rango de temperatura diurna de 21°C a 29°C y nocturna de 10°C a 15°C. Es esencial monitorear la temperatura dentro del invernadero, especialmente al utilizar fuentes de luz que emiten calor. El uso de lámparas incandescentes puede elevar la temperatura interna, por lo que es crucial asegurarse de que no exceda los 30°C para evitar el estrés térmico en las plantas (Garvillo, 2023).

Una ventilación adecuada es vital para mantener niveles óptimos de temperatura y humedad, así como para prevenir la acumulación de patógenos. La instalación de dos ventiladores en la parte superior del invernadero ayudará a promover la circulación del aire y a disipar el calor generado por la lámpara. Es recomendable que los ventiladores operen de manera continua o en intervalos regulares para asegurar una renovación constante del aire (Bribiesca, R., 2020).

El método de riego subterráneo es una técnica que suministra agua directamente al sistema radicular mediante emisores enterrados. Este método minimiza pérdidas por evaporación, lo que lo convierte en una opción eficiente para las suculentas, especialmente en ambientes controlados como invernaderos. Al permitir que el agua llegue directamente a las raíces, este sistema evita el desperdicio hídrico, una ventaja significativa en regiones con limitaciones de agua (Lucas et al., 2015).

Protocolos

El bus 1-Wire es un protocolo de comunicación serial desarrollado por Dallas Semiconductor, ahora parte de Maxim Integrated, que permite la interconexión de dispositivos digitales utilizando una única línea de datos y una referencia a tierra común. En una configuración maestro-esclavo, el dispositivo maestro controla la comunicación y puede interactuar con múltiples esclavos, cada uno con una dirección única de 64 bits, lo cual facilita la identificación y el acceso a cada dispositivo en el bus (Linke, 2008).

Por otro lado, el bus I2C, desarrollado por Philips, permite la interconexión de múltiples dispositivos mediante dos líneas, SDA (datos) y SCL (reloj). Este protocolo soporta varios modos de velocidad y puede conectar múltiples dispositivos maestros y esclavos, cada uno identificado por una dirección única de 7 o 10 bits (NXP Semiconductors, 2021).

Componentes

El sensor DS18B20 es un sensor de temperatura digital que utiliza el bus 1-Wire para comunicarse con dispositivos maestros como la Raspberry Pi. Este sensor, que tiene un rango de temperatura de -55°C a +125°C y una precisión de ±0.5°C, se puede alimentar directamente desde la línea de datos, simplificando el diseño del sistema (Maxim Integrated, 2015).

El TRIAC es un dispositivo semiconductor utilizado para controlar flujos de baja potencia en un circuito CA. Los TRIACS pueden conducir en ambos sentidos de la corriente, permitiendo la modulación del ciclo completo de la señal alterna (Boylestad y Nashelsky, 2015).

El IRLZ44N es un transistor MOSFET de canal N diseñado principalmente para aplicaciones de conmutación y control de potencia. Este componente se destaca por su capacidad para manejar altas corrientes de hasta 47 amperios y su baja resistencia en estado de conducción, lo que lo hace eficiente

en términos de disipación de energía. Gracias a su voltaje de umbral bajo, puede ser controlado directamente mediante señales de nivel lógico, lo cual facilita su integración en circuitos digitales (Philips Semiconductors, 1999).

El MOC3021 es un optoacoplador diseñado para aislar eléctricamente circuitos de control de corriente directa y dispositivos de corriente alterna. Este dispositivo emplea un diodo emisor de infrarrojos y un interruptor bilateral de silicio que, juntos, permiten controlar cargas como motores, lámparas o solenoides. Una de sus principales ventajas es su capacidad para bloquear voltajes de hasta 400 voltios, proporcionando un aislamiento seguro de hasta 5300 voltios RMS (Fairchild Semiconductor, 2003).

Por otro lado, los optoacopladores o optoaisladores son dispositivos que permiten la transmisión de señales entre dos circuitos. El dispositivo consta de un LED y un fotodiodo, donde el LED emite luz en señal de entrada activando el fotodiodo de salida. El optoacoplador 4N25 es un dispositivo electrónico ampliamente utilizado que combina un diodo emisor de luz (LED) y un fototransistor NPN en un encapsulado DIP de 6 pines, con el propósito de proporcionar aislamiento eléctrico entre dos circuitos. Entre sus características principales destacan su capacidad de manejar tensiones de colector-emisor de hasta 70 V, así como una corriente de entrada típica de 10 mA para activar el LED interno (Mishra y Singh, 2018; Vishay Semiconductors, 2010).

El sensor de humedad del suelo FC-28 es un dispositivo ampliamente utilizado para medir la humedad en aplicaciones agrícolas, como los sistemas de riego automatizados en invernaderos. Este sensor opera mediante la medición de la conductividad eléctrica entre sus electrodos, la cual varía según el contenido de agua en el suelo. Ofrece tanto una salida analógica como una digital, lo que facilita su integración con microcontroladores. Su implementación permite monitorear continuamente la humedad del suelo, lo que resulta fundamental para optimizar el uso del agua y evitar problemas como el riego excesivo o la deshidratación de las plantas (Marrero Ramírez et al., 2021).

Control de un foco incandescente de 100 W mediante PID

El control PID es ampliamente utilizado para regular dispositivos que requieren precisión, como un foco incandescente de 100 W. Este sistema ajusta la potencia en función de la diferencia entre el valor deseado y el valor medido, minimizando el error actual y acumulado. El control PID ofrece una respuesta dinámica y estable al regular variables como la temperatura, gracias a su capacidad de ajustar automáticamente los parámetros proporcional, integral y derivativo. En este caso, la señal del PID puede controlar un TRIAC para modular gradualmente la intensidad lumínica del foco y, en consecuencia, la temperatura generada (Dorf y Bishop, 2011).

Control de los ventiladores mediante PWM

La Modulación por Ancho de Pulso (PWM) es una técnica eficiente para controlar dispositivos como ventiladores. El PWM permite variar la velocidad de un motor ajustando el ciclo de trabajo de una señal digital, lo que regula la potencia entregada sin pérdidas significativas de energía. En este caso, la frecuencia de la señal debe seleccionarse adecuadamente para evitar ruido y garantizar un funcionamiento suave. Este método se implementa fácilmente mediante un microcontrolador y un transistor o driver de motor, permitiendo ajustar la velocidad del ventilador según los requerimientos del sistema (Rashid, 2013).

Lista de Componentes

Se asume que el alumno cuenta con una Raspberry Pi con sistema operativo Raspberry Pi OS Lite (2023–05–03 o anterior) e intérprete de Python instalado. Se aconseja encarecidamente el uso de Git como programa de control de versiones.

Dispositivos principales

- 1 Raspberry Pi Placa Base Modelo B / 8GB

- 1 microcontrolador RP2040 (Raspberry Pico) con firmware MicroPython precargado

Accesorios y conectividad

- 1 cable USB-C con soporte para datos
- 1 conector DIL con cable plano tipo listón para el GPIO de la Raspberry Pi
- Cables y conectores varios
- 10 Borneras de 2
- 2 Borneras de 3

Alimentación y energía

- 1 fuente de alimentación regulada a 5V y al menos 2 amperios de salida
- 1 fuente de alimentación regulada a 12V y al menos 1 amperio de salida

Resistencias

- 1 resistencia de 68 kΩ ($\frac{1}{4}$ Watt)
- 4 resistencias de 10 kΩ ($\frac{1}{4}$ Watt)
- 1 resistencia de 4.7 kΩ ($\frac{1}{4}$ Watt)
- 1 resistencia de 1 kΩ (1 Watt)
- 4 resistencias de 470 Ω ($\frac{1}{4}$ Watt)
- 1 resistencia de 330 Ω ($\frac{1}{4}$ Watt)

Componentes electrónicos adicionales

- 1 TRIAC BT137
- 4 diodos 1N4007 o puente rectificador equivalente
- 1 optoacoplador 4N25
- 1 optoacoplador MOC 3021
- 1 LED ultrabillante de 5 mm
- 3 transistores IRLZ44

Sistema de ventilación y enfriamiento

- 2 ventiladores plásticos 510-740 de 4”12V RAD (8x8x2.5 cm)

Sistema de bombeo

- 1 mini bomba de agua 6-12V (3W, 0.3Mpa)
- 4 metros de manguera para bomba de agua de 0.9 a 1 cm

Iluminación

- 1 foco incandescente de 100W (NO AHORRADOR NI LED)
- 1 socket para foco incandescente

Otros

- 1 protoboard o circuito impreso equivalente
- 1 sensor de humedad del suelo FC-28 Higrómetro
- 4 sensores de temperatura DS18B20

3. Descripción del funcionamiento de los componentes

Control del foco

Al estar trabajando con un circuito que requiere una parte en AC y otra en DC, es necesario rectificar la señal. Para ello se utilizaron diodos 1N4007, debido a su capacidad de soportar hasta 1000 V, lo cual es más que suficiente para la señal de AC en México, que es de 120 V. Por otro lado, el amperaje especificado para ellos es de un máximo de 1 A. Al usar un foco de 100 W y despejar de la fórmula $I = \frac{P}{V}$, obtenemos un resultado de menos de un amperio, 0,833, por lo que estos diodos son ideales para la rectificación de esta señal.

Se utiliza un TRIAC (Triode for Alternating Current) que permite conmutar el paso de la corriente en un circuito de AC. En este caso, este circuito enciende el foco, cumpliendo la función de un interruptor, similar a un transistor en DC o a un relé, pero con la ventaja de ser más duradero y de poder funcionar con una señal de un microcontrolador, incluso con una corriente muy pequeña, permitiendo su activación controlada.

Con el fin de aislar correctamente las señales de AC y DC, se utilizaron dos optoacopladores: el 4N25 y el MOC3021. De esta forma, podemos detectar el cruce por cero de la señal rectificada. Asimismo, se puede enviar una señal al tiristor para activarlo y permitir el paso de la corriente al foco. El MOC3021 fue utilizado por tener un tiristor interno que le permite manejar los cambios de fase de una señal de AC sin perjudicar su funcionamiento, de forma que puede mandar una pequeña señal de AC al TRIAC, que manejará la señal directa de AC. Por su parte, el 4N25, al tener un circuito simple con un LED y un fototransistor, permite que el microcontrolador pueda detectar cada vez que el LED se apague, lo cual indica que la señal ha pasado por cero.

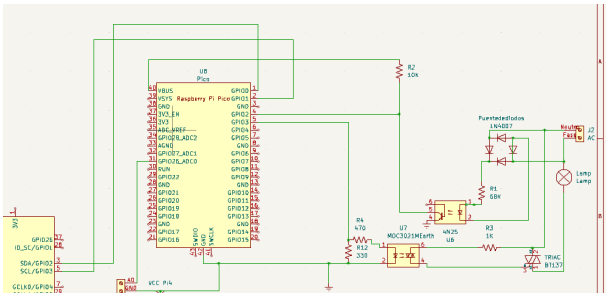

Figura 1: Circuito de control de foco

Control de motores

Los motores son actuadores que requieren una gran corriente para vencer la inercia y comenzar su trabajo. Con el fin de no exigir demasiado al microcontrolador y evitar dañarlo, se utilizan transistores

en modo interruptor con una fuente externa que pueda satisfacer las necesidades de los actuadores, como amperajes relativamente altos (1 A) y un voltaje diferente al de salida de los microcontroladores.

Para este circuito se utilizaron transistores IRLZ44, ya que los motores de los ventiladores requieren 12 V DC y un amperaje de 220 mA, según la hoja de especificaciones. Por lo tanto, se empleó un transistor en modo interruptor para cada ventilador, así como para la bomba de 12 V y 250 mA. Para proteger al microcontrolador, y aprovechando la capacidad de los transistores de activarse con una señal de bajo amperaje, se utilizó una resistencia de 470 Ω y otra resistencia de 10 kΩ entre el source y el gate para evitar ruido en la señal, asegurando que se apaguen y enciendan según lo deseado.

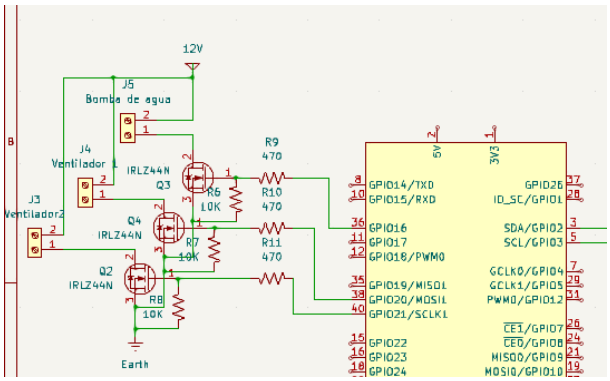

Figura 2: Circuito del control de los motores

Sensado

El sistema incluye dos tipos principales de sensores: sensores de temperatura y un sensor de humedad.

Los sensores de temperatura son del tipo DS18B20, que operan mediante el protocolo One Wire. Esto permite realizar varias mediciones usando solamente un pin de señal, sin la necesidad de preocuparse por la sincronización de datos ni por la conversión analógica a digital. Estos sensores utilizan una resistencia de 470 Ω conectada a VCC, según lo recomendado en la hoja de datos.

Por su parte, para la medición de humedad se utilizó un higrómetro FC-28, que permite obtener lecturas tanto analógicas como digitales. Esto facilita una mejor aproximación de la humedad dentro del invernadero. Además, su cableado es simple, utilizando únicamente un pin ADC proporcionado por la Pico, obteniendo un porcentaje de humedad útil para decidir el riego.

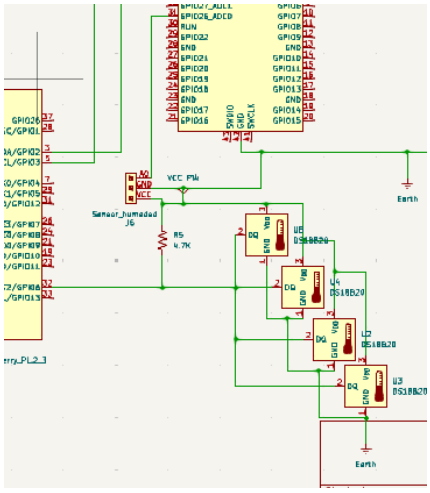

Figura 3: Circuito del control de los sensores

4. Descripción del funcionamiento de la tarjeta controladora y circuito microcontrolador

En este sistema se emplean dos tarjetas: una Raspberry Pi 4 y una Raspberry Pi Pico, cada una con funciones específicas dentro del diseño.

La Raspberry Pi 4 se encarga de ejecutar los procesos principales, incluyendo el cálculo y la activación de los actuadores. Esta tarjeta realiza los cálculos del error y las correcciones correspondientes para el control PID, además de gestionar el sensado de los termómetros mediante el protocolo One Wire. También es responsable de la interfaz gráfica para el servidor, permitiendo la visualización y el control centralizado de la mayoría de las señales del sistema.

Por otro lado, la Raspberry Pi Pico tiene como función principal la gestión de señales críticas enviadas a la Raspberry Pi 4. Esto incluye la detección del cruce por cero en el circuito del foco y el envío de la señal de retorno desde el dimmer al MOC. Gracias a sus puertos ADC, la Raspberry Pi Pico también realiza la medición de los niveles de humedad, proporcionando una señal digital (0 o 1) a la Raspberry Pi 4 para su procesamiento, contribuyendo así al comportamiento automatizado del sistema.

5. Cuidado de la salud y advertencias de riesgos

El diseño e implementación de un sistema de invernadero utilizando componentes electrónicos y eléctricos requiere adoptar medidas de seguridad rigurosas para proteger tanto al usuario como al equipo.

En primer lugar, se deben manipular con precaución las fuentes de alimentación y los dispositivos conectados a la corriente eléctrica. Nunca se debe operar con las manos húmedas ni en superficies mojadas, ya que esto aumenta significativamente el riesgo de electrocución. Asimismo, es fundamental utilizar protectores contra sobrecargas para evitar picos de voltaje que puedan dañar el sistema o causar incendios. Los cables y conectores deben revisarse regularmente para garantizar que no presenten desgaste o daño que pueda resultar en cortocircuitos o pérdida de funcionalidad.

El foco incandescente de 100W puede representar un riesgo, ya que alcanza temperaturas elevadas durante su operación. El contacto directo con este elemento puede causar quemaduras graves, y su proximidad a materiales inflamables aumenta el peligro de incendio. Por ello, se debe instalar en un soporte resistente al calor y mantenerlo alejado de otros componentes sensibles al calor, como sensores o cables.

Los sistemas de ventilación y bombeo implican riesgos mecánicos y eléctricos. Los ventiladores en funcionamiento pueden causar lesiones si se tocan accidentalmente, mientras que la bomba de agua, al operar con líquidos, presenta el peligro de cortocircuitos si las conexiones eléctricas no están adecuadamente aisladas de la humedad. Es recomendable utilizar carcasas protectoras para los ventiladores y garantizar que las conexiones eléctricas estén adecuadamente aisladas.

Trabajar con estos dispositivos requiere un entorno limpio, seco y bien ventilado, así como un conocimiento básico de las especificaciones técnicas de cada componente. Tener acceso a un interruptor de seguridad para desactivar rápidamente el sistema en caso de emergencia, además de un kit de primeros auxilios, puede ser crucial para abordar cualquier eventualidad.

6. Información sobre el cuidado de los componentes electrónicos delicados

El cuidado adecuado de los componentes electrónicos es fundamental para garantizar su correcto funcionamiento y prolongar su vida útil, especialmente en proyectos donde se utilizan dispositivos sensibles como los incluidos en el sistema de invernadero. En primer lugar, es necesario considerar las descargas electrostáticas (ESD), ya que estas representan un riesgo significativo para componentes como la Raspberry Pi y el microcontrolador RP2040. Para mitigar este riesgo, se recomienda trabajar en superficies

antiestáticas, utilizar una pulsera antiestática conectada a tierra y manipular los dispositivos únicamente por los bordes, evitando el contacto directo con pines o terminales expuestos.

Los TRIAC, optoacopladores y transistores requieren un manejo delicado, asegurando siempre que sus terminales estén aislados y protegidos contra picos de corriente. Para ello, se sugiere utilizar fusibles o circuitos de protección adicionales. El sistema de ventilación y bombeo también presenta cuidados específicos. Los ventiladores deben mantenerse libres de polvo y montados de manera firme para evitar vibraciones que puedan dañar otros componentes. Asimismo, la bomba de agua debe operarse con precaución, evitando su funcionamiento en seco y asegurando que las conexiones eléctricas estén debidamente selladas para prevenir filtraciones y cortocircuitos.

El foco incandescente debe mantenerse a una distancia prudente de los componentes electrónicos para evitar daños por exposición al calor. Además, debe evitarse su manipulación mientras está encendido o caliente, debido al riesgo de quemaduras. Por último, el sensor de humedad del suelo FC-28 requiere una limpieza regular para evitar corrosión y debe utilizarse dentro de sus parámetros operativos, evitando inmersiones profundas en agua, ya que podrían dañarlo.

7. Configuración de la tarjeta controladora y microcontrolador

Para la configuración de los protocolos de comunicación I2C y One-Wire en la Raspberry Pi, se siguieron los pasos descritos a continuación:

1. Acceder al menú de configuración de la Raspberry Pi utilizando el comando:

   ```
   sudo raspi-config
   ```

2. Dentro del menú de configuración, seleccionar la opción **Interface Options**.
3. Habilitar el protocolo I2C seleccionando la opción **I2C** y confirmando su activación.
4. Repetir el proceso para habilitar el protocolo One-Wire seleccionando la opción **One-Wire**.
5. Guardar los cambios y salir del menú de configuración.
6. Reiniciar la Raspberry Pi para aplicar los cambios ejecutando el comando:

   ```
   sudo reboot
   ```

7. Verificar que los protocolos estén habilitados correctamente:
   - Para I2C, instalar la utilidad `i2c-tools` si aún no está instalada:

     ```
     sudo apt-get install -y i2c-tools
     ```

     Luego, utilizar el comando:

     ```
     i2cdetect -y 1
     ```

     Esto mostrará una tabla con los dispositivos conectados en el bus I2C.
   - Para One-Wire, verificar la presencia del archivo de dispositivo asociado en el sistema de archivos en la ruta:

     `/sys/bus/w1/devices`

Donde los sensores o dispositivos conectados deberían aparecer listados.

Con estos pasos, los protocolos I2C y One-Wire quedan configurados y listos para su uso en la Raspberry Pi. Esto permite la comunicación con los sensores de temperatura y la Raspberry Pi Pico.

7.1. Inicio automático

Para la ejecución automática, se debe acceder nuevamente a la configuración de la Raspberry Pi utilizando el comando:

```
sudo raspi-config
```

De ahí se elegirá la opción **System Options** → **Boot/Autologin** y seleccionar la opción que mejor se ajuste a las necesidades.

Se reiniciará para guardar las configuraciones y, una vez que vuelva a encender, se creará un archivo bash que ejecutará el archivo del proyecto con un código similar al siguiente:

```
#!/bin/bash

# Navegar al directorio donde está el archivo final.py
cd /ruta/del/programa

# Ejecutar el programa en Python
python3 programa.py

# Fin del script
exit 0
```

Se le otorgarán permisos de ejecución con:

```
chmod +x /home/pi/miprograma.sh
```

Luego, se modificará el archivo `rc.local` agregando la siguiente línea antes del `exit` del archivo:

```
/home/pi/programa.sh
```

8. Desarrollo de los componentes de software

Control del sistema de irrigación y humedad

Cruz Calderón Jorge Luis

Como se investigó en los antecedentes, la planta suculenta es capaz de soportar humedades bajas de alrededor del 10 % y 30 %. Y, en base a esto, se creó la función `controlar_humedad` en la Pico, que gestiona el nivel de humedad mediante la activación o desactivación de una salida según los valores obtenidos del sensor. En primer lugar, evalúa si el porcentaje de humedad es inferior al umbral mínimo (10 %).

Si esta condición se cumple, activa la salida a un estado alto (1) y emite un mensaje indicando que la humedad es baja. En cambio, si la humedad excede el umbral máximo (30 %), desactiva la salida estableciendo un estado bajo (0). Finalmente, si la humedad se encuentra dentro del rango aceptable, la función asegura que la salida permanezca desactivada. De esta forma, el sistema garantiza el control eficiente de la humedad, manteniéndola dentro de los límites especificados para una suculenta.

```
1 def controlar_humedad(humedad, pin, min_humedad=10, max_humedad=30):
```

```
    if humedad < min_humedad:
        pin.value(1)
        print("Humedad baja. Activando salida...")
    elif humedad > max_humedad:
        pin.value(0)
        print("Humedad alta. Desactivando salida...")
    else:
        print("Humedad dentro del rango adecuado.")
        pin.value(0)
```

Código 1: Función para el control de humedad

En la Raspberry Pi, la función `solve_humidity` es responsable de gestionar la activación del sistema de irrigación en función de una señal de entrada detectada en el pin denominado `PIN_INTERRUPT`. En primer lugar, verifica si dicho pin se encuentra en un estado alto, indicando la necesidad de iniciar el riego. En este caso, la función activa el pin de salida para habilitar el sistema de irrigación, manteniéndolo encendido durante un segundo utilizando un temporizador o función de retardo. Posteriormente, desactiva el pin de salida para detener la irrigación.

Si, por el contrario, la señal en `PIN_INTERRUPT` está en estado bajo, la función garantiza que el sistema de irrigación permanezca apagado al desactivar el pin de salida. De esta manera, la función asegura un control eficiente y reactivo del riego, basado en las condiciones detectadas por el sensor.

```
def solve_humidity():
    if GPIO.input(PIN_INTERRUPT) == GPIO.HIGH:
        GPIO.output(PIN_OUTPUT, GPIO.HIGH)
        print("Interrupci n detectada, PIN 16 en ALTO")
        time.sleep(1)
        GPIO.output(PIN_OUTPUT, GPIO.LOW)
    else:
        GPIO.output(PIN_OUTPUT, GPIO.LOW)
```

Código 2: Función activar la irrigación

Control de temperatura del invernadero utilizando PID a través del foco incandescente

Frías Hernández Camille Emille Román y Sánchez Estrada Ángel Isaac

El sistema implementa un controlador PID para regular la temperatura del invernadero ajustando la potencia del foco incandescente a través del protocolo I^2C.

La clase `PIDController` es el núcleo del sistema de control proporcional-integral-derivativo (PID). Utiliza la fórmula:

$$\text{salida} = K_p \cdot \text{error} + K_i \cdot \text{integral} + K_d \cdot \text{derivada}$$

Donde:

- `Kp`: Coeficiente proporcional.
- `Ki`: Coeficiente integral.
- `Kd`: Coeficiente derivativo.
- `error`: Diferencia entre el setpoint y el valor actual.
- `integral`: Acumulación de errores pasados.
- `derivada`: Cambio del error con respecto al tiempo.

- `calculate()`: Ajusta la potencia del sistema y la limita entre 0 % y 100 %.

```
def calculate(self, setpoint, actual_value, dt=3.0):
    error = setpoint - actual_value
    self.integral += error * dt
    derivative = (error - self.previous_error) / dt
    output = self.Kp * error + self.Ki * self.integral + self.Kd * derivative
    self.previous_error = error
        # Limitar salida entre 0% y 100%
   output = max(0, min(100, output))
   return round(output)
```

Código 3: Actualización de errores

- `write_power(pwr)`: Envía el valor de potencia al dispositivo esclavo a través de I^2C.

```
def write_power(pwr):
    try:
        data = struct.pack('<f', pwr)
        msg = smbus2.i2c_msg.write(SLAVE_ADDR, data)
        i2c.i2c_rdwr(msg)
    except Exception as e:
        print(f"Error en la escritura I2C: {e}")
```

Código 4: Envío de la potencia

- `control_temperature(setpoint)`: Lee la temperatura actual, calcula la potencia ajustada usando el controlador PID y la envía al esclavo.

```
def control_temperature(setpoint):
    pid_controller = PIDController()  # Usa las constantes predeterminadas
    try:
        actual_temp = read_temperature()
        if actual_temp is None:
            print("No se obtuvo una lectura v lida. Intentando de nuevo...")
            return None

        adjusted_power = pid_controller.calculate(setpoint, actual_temp)
        write_power(adjusted_power)

        print(f"Setpoint: {setpoint} C  | Actual: {actual_temp:.2f} C  | Potencia:
            {adjusted_power}%")
        return adjusted_power
    except Exception as e:
        print(f"Error en el control PID: {e}")
        return None
```

Código 5: Método para utilizar el PID

El control del foco está dado por un dispositivo esclavo I^2C que controla un dimmer para ajustar la intensidad de un foco, además de monitorear la humedad en un hilo separado.

Se utiliza un programa PIO (Programmable I/O) para sincronizar el control del foco con el cruce por cero de la corriente alterna, ajustando el retraso según la potencia recibida.

El dispositivo esclavo I^2C, implementado con la librería `i2cslave.py`, recibe valores de potencia, los valida y ajusta el brillo del foco utilizando el dimmer.

Se emplea la librería de Mauricio Matamoros para usar la Raspberry Pi Pico como dispositivo esclavo I^2C.

De forma que el flujo es el siguiente:

1. El script `PID.py` calcula el nivel de potencia necesario para mantener la temperatura deseada y lo envía al esclavo I^2C.

2. El script `main.py` recibe el valor de potencia y ajusta el brillo del foco mediante el dimmer.

3. Simultáneamente, un hilo de monitoreo mide la humedad, garantizando la operación concurrente del sistema.

Control de potencia del ventilador mediante PWM

Cruz Calderón Jorge Luis y Sánchez Estrada Angel Isaac

Para ajustar la potencia de los ventiladores mediante PWM, se creó la función `set_motor_power`. Cuando se llama a esta función, primero verifica que el valor de potencia, representado por el parámetro `power`, esté comprendido entre 0 y 100. Este valor se mide en porcentaje.

- 0 % significa que el motor estará completamente apagado.
- 100 % significa que el motor estará funcionando a toda su potencia.

Si el valor es válido, la función utiliza el comando `ChangeDutyCycle` del objeto `pwm`. Este comando ajusta el ciclo de trabajo, que es el porcentaje del tiempo que la corriente está .[en]cendida"durante un período de tiempo.

- Si `power` es 50, el motor recibirá corriente la mitad del tiempo y girará a la mitad de su potencia máxima.
- Si `power` es 100, el motor recibirá corriente todo el tiempo y girará a su máxima velocidad.

Si el valor de `power` está fuera del rango permitido, la función mostrará un mensaje de error para indicar que la potencia debe ser un número entre 0 y 100.

```
1
2 def set_motor_power(pwm, power):
3
4     if 0 <= power <= 100:
5         pwm.ChangeDutyCycle(power)
6     else:
7         print("Error: La potencia debe estar entre 0 y 100.")
```

Código 6: Manejo de la potencia del ventilador mediante PWM

Programado de ciclos de temperatura e irrigado

Cruz Calderón Jorge Luis

La función `get_setpoint` determina el valor de la temperatura deseada (*setpoint*) que será utilizado en el sistema de control térmico. Su funcionamiento se basa en dos posibles escenarios: el uso de un valor personalizado definido por el usuario o un valor predeterminado que varía según el ciclo diurno o nocturno.

En primer lugar, la función verifica si existe un valor personalizado asignado a la variable global `custom_setpoint`. Si dicho valor no es nulo, la función retorna directamente este valor, dado que se asume que el usuario ha decidido establecer manualmente la temperatura deseada. Este comportamiento prioriza la configuración manual sobre la automática, permitiendo mayor flexibilidad.

En caso de que no se haya definido un valor personalizado, la función procede a calcular el *setpoint* basándose en la hora actual del sistema. Para ello, utiliza `time.localtime().tm_hour`, que devuelve la hora en formato de 24 horas.

- Si la hora está entre las 06:00 y las 20:00, se retorna un valor de 25 °C, que representa la temperatura predeterminada para el período diurno.

- Si la hora está fuera de este rango (20:00 a 06:00), se retorna un valor de 12.5 °C, que corresponde a la temperatura predeterminada para el período nocturno.

```
def get_setpoint():

    global custom_setpoint
    if custom_setpoint is not None:
        return custom_setpoint

    current_hour = time.localtime().tm_hour
    if 6 <= current_hour < 20:  # De 6:00 AM a 8:00 PM
        return 25.0  # Temperatura diurna
    else:  # De 8:00 PM a 6:00 AM
        return 12.5  # Temperatura nocturna
```

Código 7: Ciclo de temperatura

El sistema de irrigación no utiliza un ciclo fijo, sino un control dependiente del nivel de humedad del suelo. Si la tierra no está lo suficientemente húmeda, se enviará una señal que activará la irrigación. Cuando el sensor de humedad supere el 30 % o alcance una humedad adecuada, la irrigación se detendrá automáticamente.

Servidor web y gráfica con histórico de temperatura, irrigación y acciones tomadas

Frías Hernández Camille Emille Román y Sánchez Estrada Angel Isaac

Se realizó un código para el servidor el cual tuviera como objetivo el llamar a un html para darle formato a la página.

```
class ControlServer(BaseHTTPRequestHandler):
    def _serve_ui_file(self):
        with open("indexv2.html", "r") as f:
            content = f.read()
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()
        self.wfile.write(bytes(content, "utf-8"))
```

Código 8: Llamada a HTML

Este mismo gestionará la llamada de funciones para darle la responsividad entre servidor y html leyendo su contenido y descripciones.

```
 if action == "update_setpoint":
                # Actualiza el setpoint
                update_custom_setpoint(float(value))
                # Inicia el sistema PID si no est  activo
                if not main_thread or not main_thread.is_alive():
                    main_thread = Thread(target=main, daemon=True)
                    main_thread.start()
                response["message"] = "Setpoint actualizado e inicio del control PID"
```

Código 9: Llamada a funciones a través del servidor

Finalmente el servidor se actualizará para mostrar las temperaturas que se irán actualizando conforme al tiempo de ejecución del mismo con el fin de que el usuario pueda monitorear en todo momento la temperatura.

```
self.send_response(200)
            self.send_header("Content-type", "application/json")
            self.end_headers()
            self.wfile.write(bytes(json.dumps(response), "utf-8"))
        except Exception as e:
```

```
            print("Error procesando la solicitud:", e)
            self.send_response(500)
            self.end_headers()
```

Código 10: Actualización del servidor web

9. Integración de los componentes en una solución de software

Para crear una solución de software, los comportamientos del sistema se separaron en distintas funciones. Esto permite llamarlas según sea necesario para representar comportamientos programados según las necesidades específicas de las plantas propuestas. Asimismo, esta estructura facilita la integración con el servidor, ya que las funciones controlan cada actuador del sistema de forma modular.

También se desarrolló una función independiente para cada motor, lo que permite controlarlos de manera individual. Esto habilita comportamientos más complejos y específicos según las condiciones del sistema.

```
def encender_ambos_motores():

    set_motor_power(pwm_motor_1, 100)
    set_motor_power(pwm_motor_2, 100)
    print("Ambos motores encendidos al 100%")

def apagar_ambos_motores():

    set_motor_power(pwm_motor_1, 0)
    set_motor_power(pwm_motor_2, 0)
    print("Ambos motores apagados")
```

Código 11: Control de encendido y apagado del motor

El sistema permite realizar ajustes en la potencia del motor mediante PWM a una frecuencia fija. Esto reduce el ruido a cambio de una ligera pérdida de potencia.

```
def ajustar_potencia_motor(motor, potencia):

    set_motor_power(motor, potencia)
    print(f"Potencia del motor ajustada a {potencia}%")
```

Código 12: Ajusta la potencia del motor

La lectura de los sensores se realiza a partir de los archivos correspondientes dentro de la carpeta `w1/devices`. Durante este proceso, se asegura que los valores sean válidos y que la lectura se haya realizado correctamente antes de agregarlos al cálculo del promedio.

```
def sensor_temperature(sensor_file):

    try:
        with open(sensor_file, 'r') as file:
            lines = file.readlines()

        if lines[0].strip().endswith('YES'):
            temperature_data = lines[1].split("t=")
            if len(temperature_data) == 2:
                temperature_c = float(temperature_data[1]) / 1000.0
                return temperature_c
```

Código 13: Lectura de la temperatura

10. Link del video demostrativo

Video demostrativo

11. Link del repositorio

GitHub

12. Conclusiones

Cruz Calderón Jorge Luis:

El desarrollo del invernadero representó un reto que abarcó desde el diseño y ensamblaje físico de la maqueta hasta la implementación de funciones para su correcto funcionamiento. Basándonos en una investigación previa sobre las necesidades específicas de las suculentas, logramos establecer ciclos diurnos y nocturnos óptimos, así como un control eficiente de la humedad mediante sensores. Aunque no se implementó un ciclo estricto de irrigación, el sistema diseñado responde automáticamente a las lecturas del sensor de humedad, garantizando un uso eficiente del agua.

El proyecto destacó por la integración de múltiples componentes, como el control de temperatura mediante un sistema PID y la regulación del flujo de los ventiladores a través de PWM, permitiendo crear un entorno óptimo para el desarrollo de las suculentas.

Uno de los principales desafíos durante el desarrollo del proyecto fue el manejo y la integración adecuada de los diversos componentes electrónicos, como MOSFETs, optoacopladores y sensores. La precisión en la conexión y ensamblaje de estos elementos resultó crucial, ya que cualquier error podía generar fallas significativas en el sistema, ocasionando retrasos al identificar y corregir los problemas. Estas dificultades resaltaron la importancia de seguir estrictamente las especificaciones técnicas de cada componente y realizar pruebas constantes en cada etapa de ensamblaje.

Por otro lado, la implementación del servidor web fue otro desafío destacado. Implicó garantizar la integración adecuada entre las funciones del sistema y su representación en el servidor, lo que exigió una verificación constante de las conexiones para evitar fallos que comprometieran los circuitos y el funcionamiento general del sistema.

Frías Hernández Camille Emille Román:

Durante el desarrollo de esta actividad se implementó una solución al problema del invernadero, integrando tanto conocimientos de software como de hardware con el objetivo de diseñar un sistema sencillo y económico.

En la parte de hardware, el desarrollo fue relativamente sencillo, siendo el principal reto la disposición física de los componentes. Un desafío particular fue el espacio limitado en el circuito de corriente alterna (AC), ya que la proximidad entre los pines de las borneras incrementaba el riesgo de un posible cortocircuito. Esto requirió especial cuidado en la organización y aislamiento de las conexiones.

Por otro lado, el desarrollo de los códigos necesarios para implementar comportamientos mínimos a nivel de señales resultó manejable. Este enfoque permitió realizar pruebas individuales de los distintos módulos, garantizando su correcto funcionamiento antes de proceder con la integración final. El comportamiento integrado fue ajustado específicamente a las necesidades de las suculentas, basándose en investigación previa.

La mayor cantidad de problemas se presentó durante la implementación del servidor web. Fue necesario adaptar las soluciones de software existentes para que fueran compatibles con la lógica y la arquitectura del servidor, lo que demandó ajustes significativos en el diseño.

Finalmente, las constantes del sistema fueron ajustadas adecuadamente para resolver la lenta transmisión de energía calorífica en el sistema. Esto implicó aumentar la ganancia proporcional en el controlador PID, permitiendo que el foco incrementara rápidamente su potencia y alcanzara eficientemente la temperatura deseada.

Sánchez Estrada Angel Isaac:

En este proyecto se desarrolló un sistema de invernadero que integra conceptos clave vistos en clase, como el control de potencia mediante PID aplicado a un foco, controladores de irrigación, monitoreo de humedad y modulación por ancho de pulso (PWM) para ventiladores. Estas implementaciones permitieron no solo un manejo eficiente de los recursos, sino también la creación de un entorno óptimo para el crecimiento de plantas suculentas.

La elección de las suculentas fue particularmente adecuada debido a su resistencia y facilidad de cuidado, ya que demandan pocos recursos y prosperan en condiciones controladas con riegos esporádicos y un rango de temperaturas bien definido. La precisión de los sistemas implementados permitió incluso considerar la posibilidad de reproducir estas plantas de manera eficiente, maximizando los beneficios de un entorno monitoreado.

Uno de los mayores retos del proyecto fue el ensamblaje de los circuitos, incluyendo su soldado, pruebas y corrección de errores. Cualquier falla requería identificar y analizar las posibles causas, lo cual se evidenció especialmente en la configuración del módulo controlador del foco. A esto se sumaron los problemas derivados del código, como la gestión de delays, las configuraciones del servidor, y la correcta vinculación de funciones con los elementos desplegados en el HTML, incluyendo la integración de hojas de estilo en CSS. La verificación constante de cada componente y su interacción fue crucial para garantizar un sistema funcional.

Además, fue indispensable consultar las hojas de especificaciones de cada componente utilizado, como focos, ventiladores, resistencias y MOSFETs. Esto permitió entender sus características individuales y su comportamiento en conjunto, asegurando que todos los elementos interactuaran de manera adecuada y eficiente dentro del sistema. Sin este análisis detallado, habría sido imposible garantizar un funcionamiento seguro y óptimo del invernadero.

Sin duda, este proyecto representó un desafío integral que abarcó tanto el ensamblaje físico como la depuración y optimización de software. Aunque el proceso fue complejo y exigente, los conocimientos adquiridos fueron significativos. Reconozco, sin embargo, que aún hay áreas por mejorar, como la optimización del código, la depuración más eficiente y la gestión de hilos para un funcionamiento más fluido y sin retrasos.

13. Cuestionario

1. ¿Qué factores deben considerarse en el diseño de un invernadero para plantas suculentas? Los factores a considerar incluyen la ventilación, la iluminación, el riego y el control de temperatura.

2. ¿Cómo contribuye el sensor DS18B20 al monitoreo de temperatura en un sistema de invernadero? Este sensor digital mide temperaturas entre -55°C y +125°C con una precisión de ±0.5°C. Su diseño permite alimentarlo directamente desde la línea de datos, simplificando su integración en sistemas basados en microcontroladores como la Raspberry Pi.

3. ¿De qué trata el protocolo 1-Wire y cómo se relaciona con el sensor DS18B20? El protocolo 1-Wire es una interfaz serial que permite la comunicación entre dispositivos utilizando una sola línea de datos y una referencia a tierra. El sensor DS18B20 utiliza este protocolo para transmitir datos de temperatura al sistema maestro, simplificando el diseño del sistema.

4. ¿Qué papel desempeña el TRIAC en el control de dispositivos de corriente alterna dentro del invernadero? El TRIAC regula el flujo de corriente alterna, permitiendo controlar dispositivos como lámparas incandescentes. Su capacidad para modular el ciclo completo de la señal facilita la regulación de la intensidad lumínica y, en consecuencia, de la temperatura en el sistema de invernadero.

5. ¿Qué es el transistor MOSFET IRLZ44N y para qué aplicaciones es más adecuado? El IRLZ44N es un transistor MOSFET de canal N diseñado para aplicaciones de conmutación y control de

potencia. Es adecuado para manejar altas corrientes, hasta 47 amperios, y su baja resistencia lo hace eficiente en términos de disipación de energía. Su capacidad para ser controlado mediante señales de nivel lógico facilita su integración en sistemas digitales.

Referencias

Boylestad, R. L., & Nashelsky, L. (2015). *Electronic Devices and Circuit Theory* (11.ª ed.). Pearson.

Bribiesca, R. (2020). *Manejo de la temperatura en el invernadero*. Consultado el 12 de noviembre de 2024, desde https://agrofacto.com/manejo-temperatura-invernadero/

Dorf, R. C., & Bishop, R. H. (2011). *Sistemas de Control Moderno* (12ª). Pearson Educación. Consultado el 21 de noviembre de 2024, desde https://www.academia.edu/35915273/Sistemas_de_Control_Moderno_Richard_Dorf

Fairchild Semiconductor. (2003). *MOC3021 Datasheet*. Consultado el 24 de noviembre de 2024, desde https://www.alldatasheet.es/datasheet-pdf/pdf/53870/FAIRCHILD/MOC3021.html

Garvillo. (2023). *La mejor temperatura para las suculentas: una guía para un control óptimo*. Consultado el 12 de noviembre de 2024, desde https://garvillo.com/es/mejor-temperatura-para-suculentas/

Linke, B. (2008, junio). Overview of 1-Wire Technology and Its Use [Maxim Integrated]. https://www.analog.com/media/en/technical-documentation/tech-articles/guide-to-1wire-communication--maxim-integrated.pdf

Lucas, F. J., Martínez-Álvarez, V., & Valiente, M. (2015). *Subsurface drip irrigation vs. surface drip irrigation in tomato*. Consultado el 12 de noviembre de 2024, desde https://repositorio.upct.es/server/api/core/bitstreams/52874a73-c7f9-4283-824d-9ab2ce4da923/content

Marrero Ramírez, S., González Palau, I., León Segovia, M. A., & Suárez Vinueza, R. E. (2021). Control de humedad y consumo de agua en un invernadero. *Ciencia, Universidad Técnica de Cotopaxi*. Consultado el 20 de noviembre de 2024, desde https://www.academia.edu/80463745/Control_de_humedad_y_consumo_de_agua_en_un_invernadero

Maxim Integrated. (2015). *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*. Consultado el 15 de noviembre de 2024, desde https://www.analog.com/media/en/technical-documentation/data-sheets/DS18B20.pdf

Mishra, M., & Singh, S. P. (2018). *Power Electronics: Circuits, Devices and Applications*. McGraw Hill Education.

Nabors, M. W. (2006). *Introducción a la botánica* (1.ª ed.). Addison-Wesley. Consultado el 12 de noviembre de 2024, desde https://www.academia.edu/44352742/Introducci%C3%B3n_a_la_Bot%C3%A1nica_Murray_W_Nabors

NXP Semiconductors. (2021). *UM10204 I2C-bus specification and user manual*. Consultado el 16 de noviembre de 2024, desde https://www.nxp.com/docs/en/user-guide/UM10204.pdf

Philips Semiconductors. (1999). *IRFZ44N Datasheet*. Consultado el 24 de noviembre de 2024, desde https://www.alldatasheet.com/datasheet-pdf/pdf/17807/PHILIPS/IRFZ44N.html?utm_source=chatgpt.com

Rashid, M. H. (2013). *Electrónica de Potencia: Circuitos, Dispositivos y Aplicaciones* (4ª). Pearson Educación.

Saltón Verde. (2022). *Calculadora de iluminación LED: Potencia, PPF, PPFD y más*. Consultado el 12 de noviembre de 2024, desde https://saltonverde.com/calculadora-de-iluminacion-led/

Vishay Semiconductors. (2010). *4N25, 4N26, 4N27, 4N28: Optocoupler, Phototransistor Output, with Base Connection [Hoja de datos]*. Consultado el 15 de noviembre de 2024, desde https://www.vishay.com/docs/83725/4n25.pdf

A. Programa Control

```
import threading
import RPi.GPIO as GPIO
import time
import logging
from motorPWM import setup_motor, set_motor_power, cleanup
from Temperature import read_temperature
from PID import control_temperature, write_power
from funciones import ciclo, update_custom_setpoint

# Configuraci n del logging
logging.basicConfig(
    filename='function_calls.log',
    level=logging.INFO,
    format='%(asctime)s - %(message)s'
)

PIN_MOTOR_1 = 20
PIN_MOTOR_2 = 21
PIN_INTERRUPT = 26
PIN_OUTPUT = 16

# Configuraci n de GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Configuraci n de motores
try:
    pwm_motor_1 = setup_motor(PIN_MOTOR_1)
    pwm_motor_2 = setup_motor(PIN_MOTOR_2)
    print("Motores inicializados correctamente.")
except Exception as e:
    print(f"Error al inicializar motores: {e}")
    pwm_motor_1 = pwm_motor_2 = None

def motor_derecho(potencia):
    if pwm_motor_1:
        logging.info(f'Funci n llamada: motor_derecho con potencia={potencia}')
        set_motor_power(pwm_motor_1, potencia)
    else:
        logging.error("pwm_motor_1 no inicializado.")

def motor_izquierdo(potencia):
    if pwm_motor_2:
        logging.info(f'Funci n llamada: motor_izquierdo con potencia={potencia}')
        set_motor_power(pwm_motor_2, potencia)
    else:
        logging.error("pwm_motor_2 no inicializado.")

def ambos_motore(potencia):
    if pwm_motor_1 and pwm_motor_2:
        logging.info(f'Funci n llamada: ambos_motore con potencia={potencia}')
        set_motor_power(pwm_motor_1, potencia)
        set_motor_power(pwm_motor_2, potencia)
    else:
        logging.error("Motores no inicializados.")

def bomba(senal):
    logging.info(f'Funci n llamada: bomba con senal={senal}')
    if senal == 1:
        GPIO.output(PIN_OUTPUT, GPIO.HIGH)
    else:
        GPIO.output(PIN_OUTPUT, GPIO.LOW)

def potencia_foco(potencia):
    logging.info(f'Funci n llamada: potencia_foco con potencia={potencia}')
    write_power(potencia)

def set_PID(temp):
```

```
69      logging.info(f'Funci n llamada: set_PID con temp={temp}')
70      control_temperature(temp)
71
72  def temperatura():
73      logging.info('Funci n llamada: temperatura')
74      return read_temperature()
75
76  def predeterminado():
77      logging.info('Revirtiendo a ciclo de setpoint predeterminado')
78      update_custom_setpoint(None)
79
80  def set_value(temp):
81      if temp is not None and not (0 <= temp <= 100):
82          logging.warning(f'Setpoint inv lido: {temp}')
83          raise ValueError("Setpoint debe estar entre 0 y 100.")
84      update_custom_setpoint(temp)
```

B. Programa Funciones

```
import threading
import RPi.GPIO as GPIO
import time
from motorPWM import setup_motor, set_motor_power
from Temperature import read_temperature
from PID import control_temperature, PIDController

pid_controller = PIDController(kp=3.0, ki=2.5, kd=1.5)

# Configuraci n de GPIO
PIN_MOTOR_1 = 20
PIN_MOTOR_2 = 21
PIN_INTERRUPT = 26
PIN_OUTPUT = 16

# Limpieza inicial para evitar conflictos
GPIO.cleanup()  # Fuerza la limpieza de cualquier configuraci n previa
GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN_INTERRUPT, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(PIN_OUTPUT, GPIO.OUT)

# Variables globales para PWM
pwm_motor_1 = None
pwm_motor_2 = None

def initialize_pwm():
    """Inicializa los pines PWM con manejo de errores y verificaci n."""
    global pwm_motor_1, pwm_motor_2

    try:
        # Limpia objetos PWM existentes si es necesario
        if pwm_motor_1 is not None:
            pwm_motor_1.stop()
            pwm_motor_1 = None
        if pwm_motor_2 is not None:
            pwm_motor_2.stop()
            pwm_motor_2 = None

        # Inicializa PWM solo si no est n configurados
        pwm_motor_1 = setup_motor(PIN_MOTOR_1)
        pwm_motor_2 = setup_motor(PIN_MOTOR_2)

    except RuntimeError as e:
        print(f"Error al inicializar PWM: {e}")
        GPIO.cleanup()  # Limpieza adicional en caso de error
        raise

def cleanup_pwm():
    """Limpia los recursos PWM y GPIO."""
    global pwm_motor_1, pwm_motor_2
    if pwm_motor_1 is not None:
        pwm_motor_1.stop()
        pwm_motor_1 = None
    if pwm_motor_2 is not None:
        pwm_motor_2.stop()
        pwm_motor_2 = None
    GPIO.cleanup()

# Nueva variable global para almacenar la temperatura personalizada
custom_setpoint = None

def get_setpoint():
    global custom_setpoint
    if custom_setpoint is not None:
        return custom_setpoint  # Usar el valor personalizado si est  definido

    current_hour = time.localtime().tm_hour
    if 6 <= current_hour < 20:  # De 6:00 AM a 8:00 PM
```

```
        return 12.0  # Temperatura diurna
    else:  # De 8:00 PM a 6:00 AM
        return 24.5  # Temperatura nocturna

def update_custom_setpoint(value):
    global custom_setpoint
    custom_setpoint = value if value is not None else None
    ciclo()  # Re-trigger control cycle with updated setpoint

def solve_temp(setpoint):
   # initialize_pwm()  # Asegura que PWM est  inicializado antes de usarlo
    actual_temp = read_temperature()
    if actual_temp < setpoint - 1:  # Por debajo del rango
        power = 20
    elif actual_temp > setpoint + 1:  # Por encima del rango
        power = 100
    else:  # Dentro del rango
        power = 20

    # Aplicar potencia a los motores
    set_motor_power(pwm_motor_1, power)
    set_motor_power(pwm_motor_2, power)

def solve_humidity():
    if GPIO.input(PIN_INTERRUPT) == GPIO.HIGH:
        GPIO.output(PIN_OUTPUT, GPIO.HIGH)
        print("Interrupci n detectada, PIN 16 en ALTO")
    else:
        GPIO.output(PIN_OUTPUT, GPIO.LOW)

def ciclo():
    """
    Main control cycle for temperature and humidity.
    Dynamically adapts to the current setpoint.
    """
    setpoint = get_setpoint()  # Fetch dynamic or custom setpoint
    control_temperature(setpoint, pid_controller)  # PID temperature control
    solve_temp(setpoint)  # Adjust temperature based on the current setpoint
    solve_humidity()  # Handle humidity control

if __name__ == "__main__":
    try:
        initialize_pwm()  # Inicializa PWM antes de usarlo
        while True:
            ciclo()
            time.sleep(1)  # Evita que el ciclo consuma recursos en exceso
    except KeyboardInterrupt:
        print("Interrupci n detectada. Limpiando GPIO...")
```

C. Programa Interface

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Control de Motores y Temperatura</title>
    <style>
      body {
        font-family: Arial, sans-serif;
        background: linear-gradient(to right, #141e30, #243b55);
        color: white;
        text-align: center;
        margin: 0;
        padding: 0;
      }
      h1,
      h2 {
        margin-top: 20px;
      }
      .container {
        display: flex;
        flex-wrap: wrap;
        justify-content: center;
        margin-top: 20px;
      }
      .slider-container {
        margin: 20px;
        text-align: center;
      }
      .slider {
        width: 300px;
        margin: 10px 0;
      }
      .button {
        margin: 10px;
        padding: 15px 30px;
        font-size: 18px;
        cursor: pointer;
        border: none;
        border-radius: 5px;
        background-color: #4caf50;
        color: white;
        transition: transform 0.2s, background-color 0.2s;
      }
      .button:hover {
        background-color: #45a049;
        transform: scale(1.05);
      }
      .response {
        margin-top: 20px;
        padding: 10px;
        background-color: #282c34;
        border-radius: 5px;
        width: 80%;
        max-width: 500px;
        margin-left: auto;
        margin-right: auto;
      }
      .temperature-box {
        margin-top: 20px;
        padding: 20px;
        background-color: #333;
        border-radius: 10px;
        font-size: 24px;
        font-weight: bold;
      }
      .chart-container {
        width: 80%;
```

```
        max-width: 800px;
        margin: 20px auto;
      }
    </style>
    <script>
      function sendAction(action, value = null) {
        fetch("/", {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ action: action, value: value }),
        })
          .then((response) => response.json())
          .then((data) => {
            const responseDiv = document.getElementById("response");
            responseDiv.innerHTML = `<p>${data.message}</p>`;
            responseDiv.style.color =
              data.status === "success" ? "lightgreen" : "red";
          })
          .catch((error) => {
            const responseDiv = document.getElementById("response");
            responseDiv.innerHTML = `<p>Error de conexi n: ${error.message}</p>`;
            responseDiv.style.color = "red";
          });
      }

      function updateSliderValue(action, sliderId, labelId) {
        const slider = document.getElementById(sliderId);
        const value = slider.value;
        const label = document.getElementById(labelId);
        label.textContent = `${value}`; // Actualiza el texto sin duplicar "%"
        sendAction(action, parseInt(value)); // Env a el valor al servidor
      }

      function updateTemperature() {
        fetch("/", {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ action: "temperatura", value: 0 }),
        })
          .then((response) => response.json())
          .then((data) => {
            if (
              data.status === "success" &&
              data.message.startsWith("Temperatura promedio")
            ) {
              const tempBox = document.getElementById("temperature-box");
              tempBox.textContent = data.message;
            }
          })
          .catch((error) => {
            const tempBox = document.getElementById("temperature-box");
            tempBox.textContent = "Error obteniendo temperatura";
          });
      }

      function sendSetValue() {
        const value = document.getElementById("setpoint").value;
        sendAction("set_value", parseFloat(value));
      }
      function setFoco() {
        const focoInput = document.getElementById("foco-input").value;
        const potencia = parseFloat(focoInput);

        if (!isNaN(potencia) && potencia >= 0 && potencia <= 100) {
          sendAction("foco", potencia); // Env a el valor al servidor
        } else {
          alert("Por favor, introduce un valor v lido entre 0 y 100.");
        }
      }

```

```
      function updateCharts() {
        document.getElementById("temperatureChart").src =
          "/temperature-chart?" + new Date().getTime();
        document.getElementById("actionChart").src =
          "/action-chart?" + new Date().getTime();
      }

      window.onload = function () {
        updateCharts();
      };

      // Actualiza las gr ficas cada minuto
      setInterval(updateCharts, 60000);

      // Llama inicial para que la temperatura se muestre al cargar la p gina
      document.addEventListener("DOMContentLoaded", updateTemperature);
    </script>
  </head>
  <body>
    <h1>Panel de Control</h1>
    <h2>Temperatura Actual</h2>
    <div id="temperature-box" class="temperature-box">
      Obteniendo temperatura...
    </div>
    <h2>Control de Motores</h2>
    <div class="container">
      <div class="slider-container">
        <label for="motor1-slider">Motor Derecho</label>
        <input
          id="motor1-slider"
          class="slider"
          type="range"
          min="0"
          max="100"
          step="1"
          oninput="updateSliderValue('motor_derecho', 'motor1-slider', 'motor1-value')"
        />
        <p>Potencia: <span id="motor1-value">50</span>%</p>
      </div>
      <div class="slider-container">
        <label for="motor2-slider">Motor Izquierdo</label>
        <input
          id="motor2-slider"
          class="slider"
          type="range"
          min="0"
          max="100"
          step="1"
          oninput="updateSliderValue('motor_izquierdo', 'motor2-slider', 'motor2-value')
              "
        />
        <p>Potencia: <span id="motor2-value">50</span>%</p>
      </div>
    </div>
    <h2>Control de Ambos Motores</h2>
    <div class="container">
      <div class="slider-container">
        <label for="ambos-slider">Ambos Motores</label>
        <input
          id="ambos-slider"
          class="slider"
          type="range"
          min="0"
          max="100"
          step="1"
          oninput="updateSliderValue('ambos_motores', 'ambos-slider', 'ambos-value')"
        />
        <p>Potencia: <span id="ambos-value">70</span>%</p>
      </div>
    </div>
```

```
    <h2>Control del Foco</h2>
    <div class="container">
      <div class="input-container">
        <label for="foco-input">Foco (Potencia 0-100%)</label>
        <input
          type="number"
          id="foco-input"
          placeholder="0-100"
          min="0"
          max="100"
          style="
            padding: 10px;
            font-size: 16px;
            width: 150px;
            margin-right: 10px;
          "
        />
        <button class="button" onclick="setFoco()">Definir</button>
      </div>
    </div>
    <h2>Otros Controles</h2>
    <div class="container">
      <button class="button" onclick="sendAction('bomba', 1)">
        Encender Bomba
      </button>
      <button class="button" onclick="sendAction('bomba', 0)">
        Apagar Bomba
      </button>
      <button class="button" onclick="sendAction('set_pid', 30)">
        PID (30 C )
      </button>
    </div>
    <h2>Modo Predeterminado</h2>
    <button class="button" onclick="sendAction('predeterminado')">
      Activar Modo Predeterminado
    </button>
    <h2>Establecer Temperatura</h2>
    <div class="container">
      <input
        type="number"
        id="setpoint"
        placeholder="Setpoint  C "
        min="0"
        max="100"
        style="padding: 10px; font-size: 16px; width: 150px; margin-right: 10px"
      />
      <button class="button" onclick="sendSetValue()">
        Establecer Setpoint
      </button>
    </div>
    <div class="chart-container">
      <img
        id="temperatureChart"
        src="/temperature-chart"
        alt="Gr fica de Temperatura"
      />
    </div>
    <div class="chart-container">
      <img id="actionChart" src="/action-chart" alt="Gr fica de Acciones" />
    </div>
    <div id="response" class="response"></div>
  </body>
</html>
```

D. Programa Main

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import json
import signal
import sys
import threading
import time
from datetime import datetime, timedelta
from control import motor_derecho, motor_izquierdo, ambos_motore, bomba, potencia_foco,
    set_PID, temperatura, cleanup, predeterminado, set_value
import matplotlib.pyplot as plt

HOST = "0.0.0.0"
PORT = 8080

# Variable global para almacenar la temperatura
current_temperature = {"average": None}

# Funci n para actualizar la temperatura peri dicamente en un hilo
def update_temperature():
    """
    Hilo en segundo plano que actualiza la temperatura cada 5 segundos.
    """
    global current_temperature
    while True:
        try:
            temp = temperatura()  # Llama a la funci n para leer la temperatura
            if temp is not None:
                current_temperature["average"] = temp
                print(f"[Hilo Temperatura] Temperatura actualizada: {temp:.2f} C ")
            else:
                current_temperature["average"] = None
                print("[Hilo Temperatura] No se pudo obtener una temperatura v lida.")
        except Exception as e:
            print(f"[Hilo Temperatura] Error actualizando la temperatura: {e}")
            current_temperature["average"] = None  # Manejo expl cito de errores
        time.sleep(5)  # Espera 5 segundos antes de la siguiente lectura

def log_action(action, message):
    """
    Funci n para registrar las acciones en una bit cora.
    """
    with open('bitacora_acciones.txt', 'a') as log_file:
        timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
        log_file.write(f"{timestamp} - Acci n: {action} - {message}\n")

def read_log_file(file_path, time_limit):
    """
    Lee un archivo de bit cora y devuelve los registros dentro del l mite de tiempo
        especificado.
    """
    data = {"timestamps": [], "values": []}
    time_limit = datetime.now() - timedelta(minutes=time_limit)
    with open(file_path, 'r') as log_file:
        for line in log_file:
            timestamp_str, value = line.split(" - ")[0], line.split(" - ")[-1]
            timestamp = datetime.strptime(timestamp_str, "%Y-%m-%d %H:%M:%S")
            if timestamp >= time_limit:
                data["timestamps"].append(timestamp_str)
                data["values"].append(value.strip())
    return data

def generate_temperature_chart():
    """
    Genera una gr fica de temperatura de los  ltimos  5 minutos y la guarda como imagen
        .
    """
    data = read_log_file('BitacoraTemp.txt', 5)
    timestamps = data["timestamps"]
```

```
    temperatures = [float(value.split(": ")[1].replace(" C ", "")) for value in data["
        values"]]

    plt.figure(figsize=(10, 5))
    plt.plot(timestamps, temperatures, marker='o')
    plt.title('Temperatura de los  ltimos  5 minutos')
    plt.xlabel('Tiempo')
    plt.ylabel('Temperatura ( C )')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig('temperature_chart.png')
    plt.close()

def generate_action_chart():
    """
    Genera una gr fica de acciones de los  ltimos  5 minutos y la guarda como imagen.
    """
    data = read_log_file('bitacora_acciones.txt', 5)
    timestamps = data["timestamps"]
    actions = data["values"]

    plt.figure(figsize=(10, 5))
    plt.bar(timestamps, range(len(actions)), tick_label=actions)
    plt.title('Acciones de los  ltimos  5 minutos')
    plt.xlabel('Tiempo')
    plt.ylabel('Acciones')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig('action_chart.png')
    plt.close()

class RequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == "/":
            self.send_response(200)
            self.send_header("Content-type", "text/html")
            self.end_headers()
            with open("interface.html", "r") as file:
                self.wfile.write(file.read().encode("utf-8"))
        elif self.path == "/temperature-chart":
            self.send_response(200)
            self.send_header("Content-type", "image/png")
            self.end_headers()
            with open("temperature_chart.png", "rb") as file:
                self.wfile.write(file.read())
        elif self.path == "/action-chart":
            self.send_response(200)
            self.send_header("Content-type", "image/png")
            self.end_headers()
            with open("action_chart.png", "rb") as file:
                self.wfile.write(file.read())
        elif self.path == "/temperature-data":
            self.send_response(200)
            self.send_header("Content-type", "application/json")
            self.end_headers()
            data = read_log_file('BitacoraTemp.txt', 5)
            temperatures = [float(value.split(": ")[1].replace(" C ", "")) for value in
                data["values"]]
            response_data = {"timestamps": data["timestamps"], "temperatures":
                temperatures}
            self.wfile.write(json.dumps(response_data).encode("utf-8"))
        elif self.path == "/action-data":
            self.send_response(200)
            self.send_header("Content-type", "application/json")
            self.end_headers()
            data = read_log_file('bitacora_acciones.txt', 5)
            actions = [value for value in data["values"]]
            response_data = {"timestamps": data["timestamps"], "actions": actions}
            self.wfile.write(json.dumps(response_data).encode("utf-8"))
        else:
```

```
            self.send_error(404, "Archivo no encontrado.")

    def do_POST(self):
        content_length = int(self.headers.get('Content-Length', 0))
        post_data = self.rfile.read(content_length)
        try:
            data = json.loads(post_data.decode("utf-8"))
            action = data.get("action")
            value = data.get("value")

            response_data = {"status": "success"}

            # Manejo de acciones
            if action == "motor_derecho":
                motor_derecho(value)
                message = f"Motor derecho ajustado al {value}%"
                response_data["message"] = message
            elif action == "motor_izquierdo":
                motor_izquierdo(value)
                message = f"Motor izquierdo ajustado al {value}%"
                response_data["message"] = message
            elif action == "ambos_motores":
                ambos_motore(value)
                message = f"Ambos motores ajustados al {value}%"
                response_data["message"] = message
            elif action == "bomba":
                bomba(value)
                message = "Bomba encendida" if value == 1 else "Bomba apagada"
                response_data["message"] = message
            elif action == "foco":
                potencia_foco(value)
                message = f"Foco ajustado al {value}% de potencia"
                response_data["message"] = message
            elif action == "set_pid":
                set_PID(value)
                message = f"PID ajustado a {value} C "
                response_data["message"] = message
            elif action == "temperatura":
                # Devuelve la temperatura almacenada en la variable global
                temp = current_temperature["average"]
                message = f"Temperatura promedio: {temp:.2f} C " if temp is not None
                    else "Temperatura no disponible"
                response_data["message"] = message

            # Nuevas funcionalidades
            elif action == "predeterminado":
                predeterminado()
                message = "Modo predeterminado activado"
                response_data["message"] = message
            elif action == "set_value":
                temp = value
                set_value(temp)
                message = f"Setpoint configurado a {temp} C "
                response_data["message"] = message

            else:
                raise ValueError("Acci n no reconocida")

            # Registrar la acci n en la bit cora
            log_action(action, message)

            # Respuesta HTTP exitosa
            self.send_response(200)
            self.send_header("Content-type", "application/json")
            self.end_headers()
            self.wfile.write(json.dumps(response_data).encode("utf-8"))
        except Exception as e:
            # Respuesta en caso de error
            self.send_response(500)
            self.send_header("Content-type", "application/json")
```

```
            self.end_headers()
            self.wfile.write(json.dumps({"status": "error", "message": str(e)}).encode("
                utf-8"))

def cleanup_and_exit(server):
    """
    Funci n para cerrar el servidor y liberar recursos de forma segura.
    """
    print("\nCerrando servidor y limpiando recursos...")
    try:
        cleanup()  # Limpia los pines GPIO en el archivo control.py
    except Exception as e:
        print(f"Error durante la limpieza: {e}")
    finally:
        server.server_close()  # Cierra el servidor HTTP
        print("Servidor detenido y recursos limpiados.   Adis  !")
        sys.exit(0)

def signal_handler(sig, frame):
    """
    Captura se ales como CTRL+C y llama a la limpieza segura.
    """
    cleanup_and_exit(httpd)

if __name__ == "__main__":
    # Inicia el hilo para actualizar la temperatura
    temperature_thread = threading.Thread(target=update_temperature, daemon=True)
    temperature_thread.start()

    # Captura la se al SIGINT (CTRL+C) para limpiar y salir de forma segura
    signal.signal(signal.SIGINT, signal_handler)

    # Inicia el servidor HTTP
    httpd = HTTPServer((HOST, PORT), RequestHandler)
    print(f"Servidor iniciado en http://{HOST}:{PORT}")

    # Genera las gr ficas cada minuto
    def generate_charts_periodically():
        while True:
            generate_temperature_chart()
            generate_action_chart()
            time.sleep(60)

    chart_thread = threading.Thread(target=generate_charts_periodically, daemon=True)
    chart_thread.start()

    try:
        # Mantiene el servidor corriendo
        httpd.serve_forever()
    except KeyboardInterrupt:
        cleanup_and_exit(httpd)
    except Exception as e:
        print(f"Error inesperado: {e}")
        cleanup_and_exit(httpd)
```

E. Programa motor PWM

```python
import RPi.GPIO as GPIO

GPIO.setwarnings(False)  # Desactiva advertencias de GPIO
GPIO.cleanup()  # Limpia los recursos previos
GPIO.setmode(GPIO.BCM)

def setup_motor(pin, frequency=250):
    """
    Configura el pin como salida PWM para controlar el motor.

    Args:
        pin: Pin GPIO conectado al motor.
        frequency: Frecuencia del PWM (en Hz).

    Returns:
        pwm: Objeto PWM inicializado.
    """
    GPIO.setup(pin, GPIO.OUT)
    pwm = GPIO.PWM(pin, frequency)
    pwm.start(0)
    return pwm

def set_motor_power(pwm, power):
    """
    Ajusta la potencia del motor.

    Args:
        pwm: Objeto PWM configurado.
        power: Potencia en porcentaje (0-100).
    """
    if 0 <= power <= 100:
        pwm.ChangeDutyCycle(power)
    else:
        print("Error: La potencia debe estar entre 0 y 100.")

def cleanup():
    """
    Limpia los pines GPIO.
    """
    GPIO.cleanup()
```

F. Programa PID

```python
import smbus2
import struct
import time
from Temperature import read_temperature

SLAVE_ADDR = 0x0A
i2c = smbus2.SMBus(1)

class PIDController:
    def __init__(self, kp=3.0, ki=2.5, kd=1.5):
        self.Kp = kp
        self.Ki = ki
        self.Kd = kd
        self.integral = 0
        self.previous_error = 0
        self.last_time = time.time()

    def calculate(self, setpoint, actual_value):
        current_time = time.time()
        dt = current_time - self.last_time
        self.last_time = current_time

        error = setpoint - actual_value
        self.integral += error * dt
        derivative = (error - self.previous_error) / dt if dt > 0 else 0

        output = self.Kp * error + self.Ki * self.integral + self.Kd * derivative
        self.previous_error = error

        output = max(0, min(100, output))
        return round(output)

def write_power(pwr):
    try:
        data = struct.pack('<f', pwr)
        msg = smbus2.i2c_msg.write(SLAVE_ADDR, data)
        i2c.i2c_rdwr(msg)
    except Exception as e:
        print(f"Error en la escritura I2C: {e}")

def control_temperature(setpoint, pid_controller):
    try:
        actual_temp = read_temperature()
        if actual_temp is None:
            print("No se obtuvo una lectura v lida. Intentando de nuevo...")
            return None

        adjusted_power = pid_controller.calculate(setpoint, actual_temp)
        write_power(adjusted_power)

        print(f"Setpoint: {setpoint} C  | Actual: {actual_temp:.2f} C  | Potencia: {
            adjusted_power}%")
        return adjusted_power
    except Exception as e:
        print(f"Error en el control PID: {e}")
        return None
```

G. Programa Temperature

```python
import time

sensor_1 = '/sys/bus/w1/devices/28-0000096e37aa/w1_slave'
sensor_2 = '/sys/bus/w1/devices/28-80301e356461/w1_slave'
sensor_3 = '/sys/bus/w1/devices/28-1e051b356461/w1_slave'
sensor_4 = '/sys/bus/w1/devices/28-0000096d7bc7/w1_slave'

sensors = [sensor_1, sensor_2, sensor_3, sensor_4]

def sensor_temperature(sensor_file):
    try:
        with open(sensor_file, 'r') as file:
            lines = file.readlines()

        if lines[0].strip().endswith('YES'):
            temperature_data = lines[1].split("t=")
            if len(temperature_data) == 2:
                temperature_c = float(temperature_data[1]) / 1000.0
                return temperature_c
    except FileNotFoundError:
        return "Sensor no encontrado"
    except Exception as e:
        return f"Error: {e}"
    return "Lectura fallida"

def average(temperatures):
    sum = 0
    for i in temperatures:
        sum += i
    return sum / len(temperatures)

def log_temperature(temperature):
    with open('BitacoraTemp.txt', 'a') as log_file:
        timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
        log_file.write(f"{timestamp} - Temperatura: {temperature:.2f} C \n")

def read_temperature():
    temps = []
    for i in sensors:
        temp = sensor_temperature(i)
        if isinstance(temp, float):
            temps.append(temp)
    avg_temp = average(temps)
    print(temps)
    log_temperature(avg_temp)
    return avg_temp

print(read_temperature())
```