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, accesibles a través de un servidor web local, para facilitar el análisis y la toma de decisiones sobre el funcionamiento del invernadero.

2. Antecedentes

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.

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 .

Temperatura 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.

3. Materiales

4. Lectura Continua de Temperatura en Consola

A partir de la ruta obtenida durante la configuración del sistema de monitoreo de la temperatura, se abrió y leyó el archivo en Python, obteniendo el valor de lectura válido, es decir, "YES". Posteriormente, se buscó la cadena "t=", que indicaba dónde comenzaba la temperatura. Esta se leyó y multiplicó por 1000 para obtener su valor en grados Celsius.

```
1
2         with open(sensor_file, 'r') as file:
3             lines = file.readlines()
4
5         if lines[0].strip().endswith('YES'):
6             temperature_data = lines[1].split("t=")
7             if len(temperature_data) == 2:
8                 temperature_c = float(temperature_data[1]) / 1000.0
9                 return temperature_c
```

Código 1: Código del Experimento 1 desplegado en consola

Finalmente, para la escritura simplemente se imprimió en la línea de comando la temperatura leída con un time sleep de un segundo.

Este proceso se puede visualizar en Video 1

5. Despliegue de Apeliido en Display LCD

Se configuró la dirección del dispositivo de I2C del display para el reconocimiento en la Raspberry pi así como las variables para especificar el modo que se utiliza, datos o comandos, las dimenciones y la dirección de la primera y segunda línea del display.

Se definieron 4 funciones del display las cuales son:

1. lcd_init: Permite limpiar el display

```
1
2 def lcd_init():
3     # Inicializaci n del display
4     lcd_byte(0x33, LCD_CMD)  # Modo de 8 bits
5     lcd_byte(0x32, LCD_CMD)  # Modo de 4 bits
6     lcd_byte(0x28, LCD_CMD)  # Modo 2 l neas , 5x7 puntos
7     lcd_byte(0x0C, LCD_CMD)  # Encender display, sin cursor
8     lcd_byte(0x06, LCD_CMD)  # Incrementar cursor
9     lcd_byte(0x01, LCD_CMD)  # Limpiar display
10    time.sleep(0.005)  # Espera para que el display se estabilice
```

Código 2: Código del Experimento 2 limpiado de los valores LCD

2. lcd_byte: recibe los bits de dónde se desea escribir, primera o segunda línea y el modo, a partir de esto se separan los bytes en 4 bits, los de alto y los debajo, los envía y habilita el control de la pantalla LCD.

```
def lcd_byte(bits, mode):
    # Enviar byte de datos
    bits_high = mode | (bits & 0xF0) | LCD_BACKLIGHT
    bits_low = mode | ((bits << 4) & 0xF0) | LCD_BACKLIGHT

    bus.write_byte(LCD_ADDR, bits_high)
    lcd_toggle_enable(bits_high)

    bus.write_byte(LCD_ADDR, bits_low)
    lcd_toggle_enable(bits_low)
```

Código 3: Código del Experimento 2 Envio de valores

3. lcd_toggle_enable: Recibe como parámetro los bits que se van a habilitar y habilita la escritura de los mismos.

```
def lcd_toggle_enable(bits):
    # Activar y desactivar se al de enable
    time.sleep(0.0005)
    bus.write_byte(LCD_ADDR, (bits | ENABLE))
    time.sleep(0.0005)
    bus.write_byte(LCD_ADDR, (bits & ~ENABLE))
    time.sleep(0.0005)
```

Código 4: Código del Experimento 2 Habilitar bits

4. lcd_message: recibe una cadena la cuál se enviará con la función de lcd_byte para cada byte dentro de la cadena.

```
def lcd_message(message):
    # Enviar mensaje al LCD
    message = message.ljust(LCD_WIDTH, " ")
    for i in range(LCD_WIDTH):
        lcd_byte(ord(message[i]), LCD_CHR)
```

Código 5: Código del Experimento 2 Manejo de cadena

A partir del código descrito se mandó el mensaje con el apellido de uno de los integrantes.

Este proceso se puede visualizar en Video 2

6. Desplegado de la temperatura y apellido paterno en el display

A partir de las funciones para la escritura del display del experimento anterior, se escribió el apellido y la temperatura.

La temperatura se obtuvo con el código del experimento 1, donde se agregó un return para poder devolver una cadena con los datos, en lugar de imprimirla en la línea de comandos.

```

        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 f"{temperature_c:.2f} C"
```

Código 6: Código del Experimento 3 Desplegado de temperatura

Este proceso se puede visualizar en Video 3

7. Desplegado de la temperatura en grados Centígrados y Farenheit

A partir de las funciones del código de desplegado de texto en el display, se agregó código para realizar la conversión entre unidades, así como para escribir el resultado en el segundo renglón, en la otra esquina del display, con el fin de facilitar su comprensión y separación del otro dato.

```
            if len(temperature_data) == 2:
                temperature_c = float(temperature_data[1]) / 1000.0
                temperature_f = (temperature_c * 9/5) + 32
                return f"{temperature_c:.1f}C {temperature_f:.1f}F"
```

Código 7: Código del Experimento 4 Desplegado de temperatura Farenheit

Este proceso se puede visualizar en Video 4

8. Marquesina de apellidos con temperaturas

A partir de las funciones presentadas anteriormente, se realizó el desplegado correspondiente de la temperatura, tanto en Celsius como en Fahrenheit. Además, se agregó una nueva función que permitía el desplazamiento del texto en el primer renglón agregando espacios a la cadena para lograr el efecto de marquesina. Esto se consiguió a través del siguiente código.

```
def marquee_text(text, position):
    # Funci n para hacer un corrimiento de marquesina
    return text[position:] + " " + text[:position]
```

Código 8: Código del Experimento 5 Comportamiento marquesina

Esta función fue llamada dentro del main, cuidando la longitud de la cadena según la capacidad del display, de esta forma completando el efecto infinito de la marquesina.

```
    while True:
        # Actualizar el texto de la marquesina en la primera l nea
        marquee_message = marquee_text(last_names_str, position)
        lcd_message(marquee_message[:LCD_WIDTH], LCD_LINE_1)
        position = (position + 1) % len(last_names_str)

        # Lee y muestra la temperatura en la segunda l nea
        temperatura = read_temperature()
        lcd_message(temperatura, LCD_LINE_2)
```

Código 9: Código del Experimento 5 Ciclo principal

Este proceso se puede visualizar en Video 5

9. Servidor web de control del display

Se utilizaron las mismas funciones mencionadas anteriormente para el control del display LCD y el sensor de temperatura. Adicionalmente, se modificó la función main, que pasó a llamarse marquee_loop, la cual ahora llama a las funciones marquee_text y lcd_message, iniciando con el mismo comportamiento del experimento anterior. Esta vez, se añadió un sleep definido por la variable marquee_speed.

```
def marquee_loop():
    global position
    while True:
        marquee_message = marquee_text(last_names_str, position)
        lcd_message(marquee_message[:LCD_WIDTH], LCD_LINE_1)
```

```
        lcd_message(read_temperature(), LCD_LINE_2)

        position = (position + (1 if marquee_direction == 'left' else -1)) % len(
            last_names_str)
        time.sleep(marquee_speed)
```

Código 10: Código del Experimento 6 Marquee loop

Este código será llamado mediante un hilo mientras que el hilo principal ejecuta en e limpiado del display.

Para representar los nuevos comportamientos solicitados, se generaron las funciones update_marquee_speed, update_marquee_direction y update_temperature_unit. Asimismo, se actualizaron las funciones para el desplegado de texto, adaptándolas a cada caso mediante un if a partir de la variable global de las funciones update.

```
def update_marquee_speed(new_speed):
    global marquee_speed
    marquee_speed = max(0.1, min(new_speed, 1.0))

def update_marquee_direction(new_direction):
    global marquee_direction
    marquee_direction = new_direction if new_direction in ['left', 'right'] else 'left'

def update_temperature_unit(unit):
    global show_temperature_unit
    show_temperature_unit = unit if unit in ['C', 'F', 'both'] else 'both'
```

Código 11: Código del Experimento 2 limpiado de los valores LCD

Finalmente, se creó una página HTML y se desarrolló el servidor web que llama al código descrito anteriormente, de forma que se controla el desplegado mediante la página, como se ve en el video.

Este proceso se puede visualizar en Video 6

10. Graficas e históricos

Se realizó a partir del código del experimento anterior, modificando el código del servidor de forma que se agregó un archivo histórico llamado temperature_log.log, el cual contiene la información de los 10 últimos minutos. Se utilizó la librería Matplotlib para generar la gráfica, que se actualizará cada vez que haya una lectura por parte del sensor. A su vez, el servidor se actualiza cada 60 segundos para refrescar los resultados en la pantalla.

```
# Leer datos del archivo log
with open(log_file, 'r') as fp:
    for line in fp:
        timestamp, temp = line.split()
        timestamps.append(float(timestamp))
        temperatures.append(float(temp))

current_time = time.time()
time_intervals = [current_time - (i * 60) for i in range(10)]
temp_intervals = []

for target_time in time_intervals:
    valid_data = [(ts, temp) for ts, temp in zip(timestamps, temperatures) if abs(ts -
        target_time) < 60]
    if valid_data:
        closest_temp = min(valid_data, key=lambda x: abs(x[0] - target_time))[1]
        temp_intervals.append(closest_temp)
    else:
        temp_intervals.append(temp_intervals[-1] if temp_intervals else 0)

```

```
plt.plot(time_labels[::-1], temp_intervals[::-1], marker='o', color='#8B0000', linestyle
    ='-')
plt.scatter(time_labels, temp_intervals, color='lime', zorder=5)
for i, (label, temp) in enumerate(zip(time_labels[::-1], temp_intervals[::-1])):
    plt.text(i, temp + 0.2, f'{temp:.2f} C ', ha='center', va='bottom', fontsize=10,
        color='black')

plt.ylim(22, 30)
plt.ylabel('Temperatura ( C )')
plt.title('Temperatura en los  ltimos  10 minutos')
plt.xticks(rotation=45)
plt.grid(True)
plt.savefig(plot_file)
plt.close()
```

Código 12: Código del Experimento 7 Gráfica histórica

Este proceso se puede visualizar en Video 7

11. Conclusiones

Cruz Calderón Jorge Luis:

En la práctica se implementó de manera eficiente el monitoreo de temperatura en un display LCD usando una Raspberry Pi y un sensor DS18B20, mediante los protocolos 1-Wire e I2C. Cada parte del desarrollo se completó de manera correcta, desde la lectura continua de temperatura en consola hasta el control remoto vía una página web.

La estructura simplificada de estos buses facilitó la conexión y programación de los dispositivos, permitiendo un despliegue en tiempo real y opciones adicionales, como la conversión de escalas de temperatura y un desplazamiento dinámico de texto en el display. Esta práctica demostró la versatilidad y precisión del sistema embebido para monitoreo ambiental, validando el uso de estos protocolos en aplicaciones de bajo costo y alto rendimiento.

Frías Hernández Camille Emille Román:

El protocolo de escritura 1-Wire resulta extremadamente útil para ahorrar alambrado y, además, facilita enormemente la lectura de los valores al almacenarlos en un archivo que es fácilmente accesible, ya en mili-Centígrados, teniendo que hacer solo una conversión a Centígrados.

Finalmente, se muestra la utilidad del protocolo I2C, una vez más, al ahorrar el manejo de los bits necesarios para controlar el display, reduciéndolo a solamente dos alambres: SCL y SDA. Pese a no ser el protocolo más rápido, para usos sencillos como lo es la escritura, resulta útil, pues la latencia no es un factor relevante.

Sánchez Estrada Angel Isaac:

En esta práctica se realizó el despliegue de temperatura en una pantalla LCD de 16x2, utilizando una Raspberry Pi y un sensor de temperatura DS18B20, lo cual permitió visualizar en tiempo real los cambios de temperatura en grados Celsius. La implementación fue funcional y sin mayores complicaciones, pues la comunicación entre el sensor y la pantalla LCD se manejó mediante el protocolo 1-Wire y una interfaz I2C, lo que facilitó el uso de solo dos pines de comunicación. No obstante, el mayor reto se presentó al implementar la marquesina en la pantalla; debido a los tiempos de refresco de la LCD, la velocidad del desplazamiento del texto no resultó completamente perceptible, limitando la fluidez del movimiento.

Además, el sensor DS18B20 requiere un identificador único de 64 bits para transmitir datos correctamente, siendo fundamental este identificador para el funcionamiento adecuado del sistema. Aunque se había trabajado previamente en la implementación gráfica, fue necesario ajustar el código para capturar y mostrar los valores de temperatura de forma directa, sin conversión, lo cual limitó las lecturas a dos

valores específicos. Finalmente, esta práctica fue sencilla en términos de implementación, pero requirió una adaptación cuidadosa de los códigos previos y optimización en la visualización de datos en pantalla, considerando las limitaciones del hardware utilizado.