Control de Invernadero

Ambriz Zuloaga Brayan Arturo, Martínez Ramírez José Ángel,
Mendoza Flores Axel Fernando

28 de mayo de 2025

1. Objetivo

Diseñar e implementar un sistema de control de temperatura y humedad en un entorno cerrado (microinvernadero), utilizando plataformas Raspberry Pi 4B y RP2040, sensores DHT11 y una carga resistiva controlada por TRIAC mediante cruce por cero. El sistema empleará un algoritmo PID y permitirá el monitoreo y ajuste en tiempo real a través de una interfaz web, con el fin de mantener condiciones óptimas para el cultivo de albahaca.

2. Introducción

En los últimos años la tecnología nos ha ayudado a desarrollar sistemas automatizados para facilitar nuestro trabajo, estos sistemas se encuentran en todas las áreas que se puedan imaginar. Por mencionar el caso de la agricultura moderna, a la hora de optimizar recursos, aumentar la eficiencia de producción, entre otros factores. Dentro de este contexto, los invernaderos inteligentes representan una alternativa eficiente para el control preciso del ambiente de cultivo, ya que permiten regular diferentes aspectos, como son la temperatura, humedad, iluminación, etc.Este proyecto se enfoca en el diseño e implementación de un sistema automatizado de control de temperatura en lazo cerrado, con el fin de crear un entorno adecuado para el desarrollo de plantas aromáticas como la albahaca, utilizando herramientas de bajo costo como la Raspberry Pi 4, la Raspberry Pi Pico 2040 y el protocolo de comunicación I2C, integradas a través de un algoritmo PID y una interfaz web de monitoreo en tiempo real.[1]

2.1. Antecedentes

Este proyecto se enmarca en el desarrollo de un sistema automatizado de control de temperatura en lazo cerrado, diseñado específicamente para una pecera adaptada como microinvernadero para el cultivo de albahaca, una planta aromática de gran valor culinario, medicinal y agrícola. Para la implementación del sistema se utilizaron plataformas de bajo costo y alta flexibilidad como la Raspberry Pi 4, encargada del control central, y la Raspberry Pi Pico 2040, utilizada como esclavo I2C para capturar datos ambientales y controlar el actuador (foco calefactor)[2]. Esta arquitectura permite establecer una comunicación maestro-esclavo mediante el protocolo I2C, manteniendo la simplicidad y eficiencia en el sistema.[3]

2.2. Raspberry Pi 4B

Es una computadora de placa única (SBC, por sus siglas en inglés) que permite la ejecución de sistemas operativos completos como Raspberry Pi OS. Su capacidad de procesamiento y conectividad la hace ideal como unidad central de control (nodo maestro) en sistemas embebidos. En este proyecto se encarga del procesamiento del algoritmo PID, la comunicación con la Raspberry Pi Pico mediante el protocolo I^2C, y de servir una interfaz web mediante Flask para el monitoreo y control del invernadero.

2.3. Raspberry Pi Pico (RP2040)

Es una microcontroladora basada en el chip RP2040 de doble núcleo. Se utiliza como esclava I²C en el sistema, encargándose de la lectura del sensor DHT11 y el control del actuador (foco calefactor). Gracias a sus características de bajo consumo y alta versatilidad, es adecuada para tareas específicas de adquisición de datos y control de hardware, liberando carga al sistema maestro.

2.4. Sensor DHT11

Este sensor digital permite medir temperatura y humedad con una sola conexión. Entrega datos calibrados y es ideal para sistemas embebidos debido a su bajo costo. En el proyecto, el DHT11 se conecta a la Raspberry Pi Pico, que lee los datos y los transmite al maestro a través de I²C. Su utilización permite controlar el entorno del invernadero y activar los mecanismos de calefacción o riego según sea necesario.

2.5. Control PID

El control PID es una forma muy común de controlar sistemas de manera automática, especialmente cuando se busca mantener una variable, como la temperatura, dentro de ciertos límites. Este tipo de control se llama de "lazo cerrado" porque está siempre revisando el valor actual y lo compara con un valor deseado. A partir de esa diferencia (llamada error), el controlador decide cuánto debe actuar para corregirlo.

El nombre PID viene de los tres elementos que lo componen:

- **Proporcional (P)**: actúa en función del tamaño del error. Mientras más grande sea el error, más fuerte corrige.
- **Integral (I)**: suma los errores que se han ido acumulando con el tiempo, ayudando a eliminar errores constantes.
- **Derivativo (D)**: observa cómo cambia el error, anticipándose a lo que podría pasar, y ayuda a estabilizar el sistema.

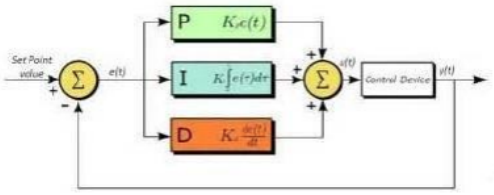

Figura 1: Representación gráfica de un controlador PID

En este proyecto de invernadero para albahaca, el PID fue programado en la Raspberry Pi 4, que actúa como la parte principal del sistema. Regularmente, la Pi recibe la temperatura medida por la Raspberry Pi Pico y la compara con la temperatura deseada que el usuario puede cambiar desde una página web. Si hay una diferencia entre el valor deseado y el actual, el PID calcula la energía necesaria para ajustar la temperatura y envía ese valor de potencia a la Pico usando la comunicación I²C. La Pico, a su vez, toma ese valor y ajusta la intensidad del foco que calienta el invernadero. Gracias a este sistema, se logró mantener la temperatura estable de forma automática y sin intervención manual. Además, se incluyeron límites al valor integral para evitar acumulaciones excesivas, y se ajustaron las constantes KP, KI y KD para obtener una respuesta estable, estas implementaciones mostraran una imagen como la siguiente.

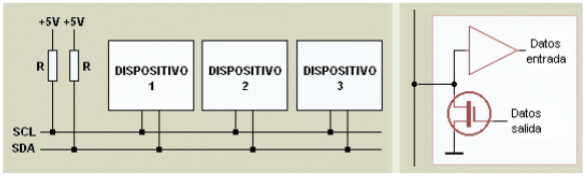

Figura 2: Diagrama del protocolo I2C

2.6. L298N

Es un puente H doble que permite controlar la dirección y velocidad de motores DC o bombas. Soporta hasta 46 V y 2 A por canal. En este proyecto, se utiliza para controlar la bomba de diafragma de 12 V, activando el sistema de riego cuando el sistema detecta niveles bajos de humedad, garantizando el riego adecuado de la albahaca.

2.7. Bus I2C

El bus I2C es un estándar ampliamente utilizado que permite la interconexión eficiente entre dispositivos con capacidad de procesamiento, como microcontroladores o, en este caso, las placas Raspberry Pi 4B y Raspberry Pi Pico 2040. Este protocolo fue desarrollado con el propósito de facilitar la transferencia de información entre módulos de manera síncrona y en serie [4]. La comunicación se lleva a cabo mediante dos líneas principales:

- **SCL**: Responsable de proporcionar los pulsos de reloj que sincronizan todos los dispositivos conectados al bus.
- **SDA**: Línea dedicada al intercambio de datos entre los dispositivos.

Para establecer la comunicación, todos los dispositivos deben estar conectados a las mismas líneas SDA y SCL, compartiendo así el canal de transmisión. Un aspecto importante de este protocolo es su arquitectura jerárquica, donde existen dispositivos maestros y esclavos. El maestro es quien inicia y controla la comunicación en el bus.

Cuando el maestro desea iniciar una comunicación, genera una secuencia especial denominada "start.º inicio. Esta es una de las dos condiciones especiales definidas en el protocolo I2C; la otra es la condición de "stop.º parada. Estas condiciones son únicas porque permiten que la línea SDA cambie de estado mientras la línea SCL se mantiene en nivel alto. Durante la transmisión de datos, la línea SDA debe permanecer estable mientras SCL está alto. Las secuencias de inicio y parada indican el inicio y el final de una transacción con los dispositivos esclavos conectados al bus.

3. Material

Se deberá contar con una Raspberry Pi con sistema operativo Raspbian e interprete Python. Ademas se usaran circuitos como detector de cruce por cero y un dimmer.

- Raspberry Pi 4B
- Micro controlador RP2040 con headers soldados y Micropython recargado
- 1 TRIAC BT138 o BT139
- 4 diodos 1N4007 o puente rectificador equivalente
- 1 optoacoplador MOC 3021
- 1 optoacoplador 4N25
- 1 foco incandescente 30 W (NO AHORRADOR NI LED)

- 1 resistencia de $68k$,1/4Watt
- 1 resistencia de 10k, 1/4Watt
- 2 resistencia de 4k7, 1/4Watt
- 1 resistencia de 1k, 1Watt
- 2 resistencia de 470, 1/4Watt
- 2 resistencia de 330, 1/4Watt
- 1 LED de 5mm
- 1 LED ultrabrillante de 5mm
- 1 protoboard o circuito impreso equivalente
- 1 fuente de alimentación regulada a 5V y al menos 2 amperios de salida
- Conectores varios
- Cable micro USB con soporte de datos
- Ventilador DC 12 V
- Módulo L298N
- Bomba de diafragma de 12 V
- Fuente de alimentacion regualda a 12 V y al menos 3 amperios de salida
- 1.5 m de manguera de 10 mm
- Pecera de 20 L
- Tabla perfocell
- Sensor de humedad-temperatura DHT11
- Tierra
- Planta de albahaca
- Botella de agua
- Cinchos de plastico
- Bolsa de plastico transparente
- Socket

4. Instrucciones

Instrucciones de configuración y puesta en marcha

A continuación se detallan los pasos necesarios para configurar y poner en funcionamiento todo el sistema, desde la Raspberry Pi Pico con MicroPython hasta la Raspberry Pi 4 con Python y las conexiones físicas. Estas instrucciones estan escritas con el fin de que cualquier persona interesada pueda replicar este proyecto sin complicaciones.

1. Instalar firmware de MicroPython en la Raspberry Pi Pico (RP2040)

Para comenzar a trabajar con el microcontrolador RP2040, uno de los primeros pasos fundamentales es instalar el firmware de MicroPython. Este entorno permite programar directamente, a continuación se describen los pasos para instalar MicroPython.

1. Ingresar al sitio oficial: https://micropython.org/download/rp2-pico/
2. Descargar el archivo con extensión `.uf2` más reciente para la Raspberry Pi Pico.
3. Con la Pico desconectada, mantener presionado el botón BOOTSEL y conectarla por USB a la computadora.
4. Se abrirá como si fuera una memoria USB. Copiar el archivo `.uf2` descargado directamente a la Pico.
5. Una vez copiado, la Pico se reiniciará automáticamente ya con MicroPython instalado.

Este proceso hace posible desarrollar programas con el IDE Thonny, gracias a MicroPython, es posible interactuar de forma sencilla con los pines del microcontrolador, accediendo a periféricos y ejecutando scripts de manera eficiente, lo cual facilita mucho el desarrollo de proyectos como el micro-invernadero automatizado.[5]

2. Configuración de la Raspberry Pi 4 (maestro)

Para poner en funcionamiento el sistema en una Raspberry Pi 4 sin interfaz gráfica, se debe realizar la configuración completamente desde la terminal. A continuación se explican los pasos necesarios para preparar el entorno:

1. **Actualizar el sistema operativo**:

   ```
   sudo apt update && sudo apt upgrade -y
   ```

2. **Instalar Python en la Raspberry Pi 4B**, ingrese el siguiente comando para instalar Python

   ```
   sudo apt install python3
   ```

   Cuando termine el proceso de instalación puede verificar si se instalo correctamente con el siguiente comando.

   ```
   python3 --version
   ```

3. **Instalar pip**, el gestor de paquetes para Python:

   ```
   sudo apt install python3-pip
   ```

4. **Instalar las librerías necesarias para el proyecto**:

   ```
   pip install flask smbus2
   ```

Una vez realizados estos pasos, la Raspberry Pi 4 estará lista para ejecutar el servidor Flask y comunicarse con la Raspberry Pi Pico mediante el bus I2C.

3. Alambrado del circuito de potencia en corriente alterna (AC)

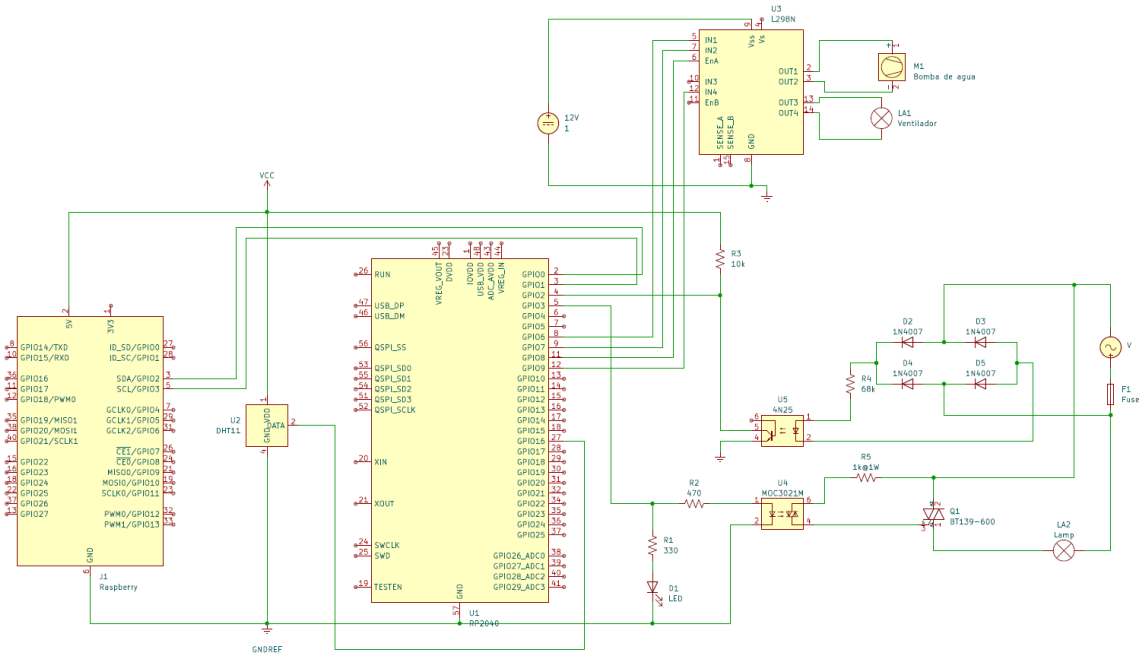

Figura 3: Diagrama de alambrado del circuito

4. Configuración del protocolo I2C entre Raspberry Pi 4 y Raspberry Pi Pico

La comunicación entre ambas placas se realiza usando el protocolo I2C. Para configurarlo se deben de seguir los siguiente pasos.

1. Conectar las líneas correctamente:
   - SDA (RPi 4 GPIO 2) → SDA (Pico GPIO 0)
   - SCL (RPi 4 GPIO 3) → SCL (Pico GPIO 1)
2. Hacer la conexion de VCC y GND de ambos dispositivos en una protoboard.
3. En la Raspberry Pi 4 se utiliza la librería `smbus2` para enviar y recibir datos desde el bus I2C.
4. En la Raspberry Pi Pico se implementó un esclavo I2C personalizado con el archivo `i2cslave.py`, que permite recibir y responder solicitudes directamente desde los registros internos del microcontrolador.
5. Probar la conexión con el siguiente comando en la RPi 4:

```
i2cdetect -y 1
```

Con los pasos anteriores estaremos un paso más adelante para poder ejecutar el el sistema completo del invernadero.

4.1. Implementación

Con la configuración previa solo faltaria ejecutar los archivos, los cuales se encuentran dentro de nuestro Git https://github.com/AngelMtz02/ProyectoFinal_FSEm, mismos archivos pueden ser copiados tanto en la Raspberry Pi 4 como en la Raspberry Pi Pico. A continuación, se describen los pasos para ejecutar correctamente el control del invernadero.

4.2. Agregar archivos a la Raspberry Pi Pico

A continuación describiremos de manera detallada los pasos a seguir para poder ejecutar todo el proyecto. Primero, se deberan de agregar los archivos que se encuentran en nuestro https://github.com/AngelMtz02/ProyectoFinal_FSEm. Con ayuda del IDE Thonny podremos guardar todos los archivos necesarios para convertir a la Pico en un esclavo I2C, lee datos del sensor DHT11 y controla actuadores como el foco, la bomba o el ventilador, ademas se guardara el archivo `main.py` para que se ejecute automaticamente el archivo y asi controlar las clases de los demas programas.

Advertencia: al momento de conectar la Pico por USB para cargar el código, es importante **desconectar temporalmente el pin VCC (3.3V o 5V)** si ya está alimentada desde la Raspberry Pi 4. Esto es para evitar posibles daños al puerto USB por doble alimentación (conflicto entre fuentes de poder).

4.3. Códigos de la RP2040

Aquí detallaremos a fondo la implementación de 3 de los 8 códigos que nos permitiera:

- Medir temperatura y humedad desde un sensor DHT.
- Controlar un ventilador DC con cambio de dirección y velocidad variable.
- Gestionar un dimmer que regula potencia.
- Comunicarse con un maestro a través del protocolo I2C, para recibir comandos y enviar datos ambientales.

slavecontroller.py

```
from machine import Timer
from dht_temp_hum import DHTSensor
from i2cslave import I2CSlave
from dimmer import Dimmer
import ustruct
import time

class Slave:
    def __init__(self):
        # Dispositivos
        self.i2c = I2CSlave(address=0x0A, sda=0, scl=1)
        self.sensor = DHTSensor(pin=16, name="Interior", num_readings=5)
        self.dimmer = Dimmer(zx_pin_num=2, triac_pin_num=3)
        self.timer = Timer()

        # Estado del dispositivo
        self.temperature = 0.0
        self.humidity = 0.0
        self.power = 0.0
        self.running = True

    def start(self):
        """Inicia todas las operaciones"""
        # Configurar timer para lecturas del sensor
```

```
        self.timer.init(mode=Timer.PERIODIC, period=2000,
            callback=self._update_sensor)
        print("Dispositivo esclavo iniciado")

    def stop(self):
        """Limpia recursos al terminar"""
        self.running = False
        self.timer.deinit()
        self.dimmer.set_power(0)  # Apagar al salir
        self.i2c.deInit()
        print("Dispositivo detenido")

    def _update_sensor(self, timer):
        """Actualiza las lecturas del sensor"""
        if self.sensor.measure():
            self.temperature, self.humidity = self.sensor.get_values()
            print(f"Medición: {self.temperature}°C, {self.humidity}%")

    def _handle_i2c_communication(self):
        """Maneja toda la comunicación I2C de forma no bloqueante"""
        # 1. Manejar lectura de potencia
        if self.i2c.waitForData(timeout=100):  # Timeout corto
            try:
                data = self.i2c.read()
                if len(data) == 4:
                    self.power = ustruct.unpack('<f', data)[0]
                    self.dimmer.set_power(self.power)
                    print(f"Potencia actualizada: {self.power}%")
            except Exception as e:
                print(f"Error al procesar potencia: {e}")

        # 2. Manejar solicitudes de temperatura
        if self.i2c.waitForRdReq(timeout=100):
            self.i2c.write(ustruct.pack('<ff', self.temperature, self.humidity))

    def run(self):
        """Bucle principal no bloqueante"""
        try:
            while self.running:
                self._handle_i2c_communication()
                time.sleep_ms(100)  # Pausa para evitar sobrecarga
        except KeyboardInterrupt:
            self.stop()
```

main.py

```
from machine import Timer
from dht_temp_hum import DHTSensor
from i2cslave import I2CSlave
from dimmer import Dimmer
import ustruct
import time
from slave_controller import Slave
```

```
class I2CDeviceController:
    def __init__(self):
        # Dispositivos
        self.i2c = I2CSlave(address=0x0A, sda=14, scl=15)
        self.sensor = DHTSensor(pin=16, name="Interior", num_readings=5)
        self.dimmer = Dimmer(zx_pin_num=2, triac_pin_num=3)
        self.timer = Timer()

        # Estado del dispositivo
        self.temperature = 0.0
        self.humidity = 0.0
        self.power = 0.0
        self.running = True

    def start(self):
        #nicia todas las operaciones
        # Configurar timer para lecturas del sensor
        self.timer.init(mode=Timer.PERIODIC, period=2000,
            callback=self._update_sensor)
        print("Dispositivo esclavo iniciado")

    def stop(self):
        #Limpia recursos al terminar
        self.running = False
        self.timer.deinit()
        self.dimmer.set_power(0)  # Apagar al salir
        self.i2c.deInit()
        print("Dispositivo detenido")

    def _update_sensor(self, timer):
        #Actualiza las lecturas del sensor
        if self.sensor.measure():
            self.temperature, self.humidity = self.sensor.get_values()
            print(f"Medición: {self.temperature}°C, {self.humidity}%")

    def _handle_i2c_communication(self):
        # 1. Manejar lectura de potencia
        if self.i2c.waitForData(timeout=0):
            try:
                data = self.i2c.read()
                if len(data) == 4:
                    self.power = ustruct.unpack('<f', data)[0]
                    self.dimmer.set_power(self.power)
                    print(f"Potencia actualizada: {self.power}%")
            except Exception as e:
                print(f"Error al procesar potencia: {e}")

        # 2. Manejar solicitudes de temperatura
        if self.i2c.waitForRdReq(timeout=0):
            self.i2c.write(ustruct.pack('<ff', self.temperature, self.humidity))

    def run(self):
        try:
            while self.running:
                self._handle_i2c_communication()
```

```
                #time.sleep_ms(100)
        except KeyboardInterrupt:
            self.stop()

def main():
    rp2040 = Slave()
    rp2040.start()
    rp2040.run()

if __name__ == '__main__':
    main()
```

dcfan.py

```
class DCFan:
    def __init__(self, pin1, pin2, enable_pin, min_duty=15000, max_duty=55535):
        self.pin1 = pin1
        self.pin2 = pin2
        self.enable_pin = enable_pin
        self.min_duty = min_duty
        self.max_duty = max_duty
        self.speed = 0

    def on(self, speed):
        self.speed = speed
        duty = self.duty_cycle(self.speed)
        print(f'Set speed: {speed}%, Duty cycle enviado: {duty}/65535')
        self.enable_pin.duty_u16(duty)
        self.pin1.value(0)
        self.pin2.value(1)


    def backwards(self, speed_percent):
        self.speed = speed_percent
        duty = self.duty_cycle(self.speed)
        self.enable_pin.duty_u16(duty)
        self.pin1.value(1)
        self.pin2.value(0)   # Corrijo la dirección aquí para backwards

    def stop(self):
        self.enable_pin.duty_u16(0)
        self.pin1.value(0)
        self.pin2.value(0)

    def duty_cycle(self, speed_percent):
        if speed_percent <= 0:
            return 0
        elif speed_percent >= 100:
            return 65535
        else:
            return int(self.min_duty + (self.max_duty - self.min_duty) *
                (speed_percent / 100))
```

4.4. Componentes Implementados

- **Sensor de temperatura y humedad**
- **Dimmer (Triac control)**
- **Comunicación I2C**
- **Ventilador DC**

Ejecución en Raspberry Pi4B

En la Raspberry Pi 4 se encuentra el código maestro del sistema, que incluye el controlador PID, la comunicación I2C y el servidor web desarrollado en Flask. Todo se ejecuta desde la terminal.

Activación del entorno virtual: Para mantener organizado el proyecto y sus dependencias, se creó un entorno virtual de Python. Al iniciar la sesión en la Raspberry Pi 4, antes de ejecutar el programa principal, se activó con:

```
source env/bin/activate
```

Este comando asegura que se usen las versiones correctas de las librerías instaladas, como `flask` y `smbus2`.

Ejecución del sistema: Desde la carpeta del proyecto se ejecutó el sistema con:

```
python3 greenhouse_raspberry.py
```

Este script lanza el servidor web en el puerto 5000, inicia el hilo de control PID, y se comunica por I2C con la Pico.

3. Acceso al sistema desde otro dispositivo

Desde cualquier otro dispositivo conectado a la misma red, se puede acceder al servidor abriendo un navegador web e ingresando la dirección IP de la Raspberry Pi 4, seguida del puerto 5000:

```
http://<ip-local>:5000
```

Por ejemplo:

```
http://192.168.1.150:5000
```

Ahí se presenta una interfaz visual con control gráfico para modificar la temperatura deseada, una gráfica en tiempo real, y los valores actuales de temperatura, humedad y potencia.

4. Fragmentos relevantes del código

A continuación se presentan algunos fragmentos clave que resumen el comportamiento del sistema:

- **Lógica del PID:**

```
error = desired_temperature - current_temperature
integral += error * dt
derivative = (error - previous_error) / dt
power = KP * error + KI * integral + KD * derivative
```

- **Comunicación I2C desde la Raspberry Pi 4:**

```
i2c.write_byte_data(SLAVE_ADDR, 0x01, int(power))
data = i2c.read_i2c_block_data(SLAVE_ADDR, 0x02, 4)
```

- **Servidor Flask que recibe nueva temperatura deseada:**

```
@app.route('/set-temp', methods=['POST'])
def set_temp():
    global desired_temperature
    desired_temperature = float(request.json['temp'])
    return '', 204
```

- **Interfaz web HTML integrada:**

```
<input type="number" id="tempInput" value="30" min="16" max="100">
<button onclick="updateTemp()">Actualizar</button>
```

Estos fragmentos reflejan los pilares del sistema: control automático, comunicación entre dispositivos, y supervisión gráfica del estado del invernadero.

5. Video del funcionamiento

El funcionamiento del proyecto está disponible en los siguientes links: Prueba Control de potencia y Prueba del sistema de irrigación

6. Repositorio del código

El código fuente del proyecto está disponible en el siguiente enlace: GitHub

7. Conclusiones

El desarrollo del sistema esclavo basado en la Raspberry Pi Pico (RP2040) para un invernadero ha permitido integrar exitosamente múltiples componentes críticos para la automatización del ambiente. A través de la implementación de distintos módulos de código, se logró establecer una arquitectura modular, eficiente y fácilmente escalable.

La clase principal Slave, que centraliza el control del sistema, permite la lectura periódica de temperatura y humedad mediante el sensor DHT, así como la recepción de comandos a través del protocolo I2C para ajustar la potencia del dimmer, encargado de controlar dispositivos como lámparas o calefactores. Esta estructura permite al microcontrolador operar como un dispositivo esclavo inteligente, capaz de reaccionar en tiempo real a las solicitudes del maestro.

Por otro lado, la implementación del módulo DCFan complementa el sistema al proporcionar control bidireccional y proporcional de un ventilador de corriente directa, lo cual es esencial para la regulación de la temperatura y ventilación del invernadero. Su diseño orientado a objetos permite una integración limpia y flexible con el resto del sistema.

En conjunto, este proyecto demuestra la capacidad de la RP2040 para actuar como un nodo esclavo robusto en aplicaciones de automatización ambiental, utilizando comunicación I2C, sensores digitales y control de potencia en corriente alterna y directa. La arquitectura modular adoptada facilita futuras mejoras, como la inclusión de sensores adicionales, almacenamiento local o comunicación con servicios en la nube.

Referencias

[1] N. Karle, A. Nandan y A. Yadav. «IoT Based Greenhouse Monitoring System Using Raspberry Pi». En: *ResearchGate* (2021). URL: https://www.researchgate.net/publication/354297402.

[2] E. Upton y G. Halfacree. *Raspberry Pi User Guide*. 4.ª ed. Wiley, 2019.

[3] M. A. Suárez Reyes y A. A. Valverde Tobar. *Diseño e implementación de un invernadero automatizado para el cultivo de hortalizas mediante tecnología IoT y energía solar*. 2023. URL: https://repositorio.upse.edu.ec/handle/46000/9082.

[4] Eduardo J Carletti. «Comunicación-bus i2c». En: *Robots Argentina* (2007).

[5] José Mauricio Matamoros de Maria y Campos. *Instalación de MicroPython en el microcontrolador RP2040*. Práctica 2 de Fundamentos de Sistemas Embebidos, Facultad de Ingeniería, UNAM. 2023.