Proyecto de Invernadero con sensor de temperatura DHT22

Aarón Ulises Torres Corte [0009-0008-2423-9258]

aaron.torresc@alumno.buap.mx

and

Johan Yuri Martínez García [0009-0001-2622-5641]

johan.martinezga@alumno.buap.mx

[1] Benemérita Universidad Autónoma de Puebla
4 Sur 104, Centro Histórico, C.P. 72000
+52 (222) 229 55 00

Abstract. This document details the design, implementation, and testing of a low-cost control system for the automation of a small-scale greenhouse. The system focuses on monitoring and regulating temperature, a critical factor for crop growth. The architecture consists of an ESP32 microcontroller as the central processing unit and a DHT22digital temperature sensor. The actuators, a fan and an incandescent light bulb, are controlled via a relay module. User interaction is managed through a graphical user interface (GUI) developed in Python with the Tkinter library, which communicates with the microcontroller wirelessly via a local Wi-Fi network using HTTP requests. The system implements two modes of operation: open-loop control, which allows the user to manually activate the actuators, and closed-loop control, which autonomously maintains the temperature at a desired value (setpoint) predefined by the user. The experimental results, obtained after a rigorous hardware debugging process,

Keywords: Control en Lazo Cerrado, ESP32, IoT, Automatización de Invernaderos, Python, Tkinter, Servidor Web, DHT22.

1 Introducción

La agricultura global se encuentra en una encrucijada, enfrentando desafíos sin precedentes como el cambio climático, el crecimiento de la población mundial y la creciente escasez de recursos naturales como el agua y la tierra cultivable. En este contexto, la agricultura de precisión y las tecnologías de agricultura en ambiente controlado (CEA, por sus siglas en inglés) emergen como soluciones vitales. Los invernaderos son la manifestación más común de la CEA, permitiendo la creación de microclimas optimizados que desacoplan la producción de alimentos de las condiciones climáticas externas.

Sin embargo, el potencial de un invernadero solo se materializa a través de una gestión precisa y constante de sus variables ambientales. La gestión manual es intensiva en mano de obra, propensa a errores humanos y energéticamente ineficiente. Por otro lado, los sistemas de automatización comerciales, aunque efectivos, suelen implicar altos costos de adquisición e instalación, haciéndolos inaccesibles para pequeños agricultores, instituciones académicas o proyectos de investigación con presupuestos limitados.

El advenimiento del Internet de las Cosas (IoT) ha democratizado el acceso a la tecnología de automatización. El microcontrolador ESP32, en particular, se ha convertido en una pieza central en este movimiento gracias a su potente procesador de doble núcleo, su conectividad Wi-Fi y Bluetooth integrada, y su bajo costo. Esta plataforma permite el desarrollo de sistemas de control sofisticados que antes estaban reservados para el ámbito industrial.

Este proyecto aprovecha el potencial del ESP32 para diseñar y construir un prototipo de sistema de control de temperatura para un invernadero. El objetivo principal es crear una solución funcional, de bajo costo y de código abierto que pueda ser replicada y adaptada. Se establecieron los siguientes objetivos específicos:

- Diseñar una arquitectura de hardware robusta utilizando componentes electrónicos comunes y asequibles.
- Implementar un firmware en el ESP32 que actúe como un servidor web, permitiendo la comunicación a través de una red Wi-Fi estándar.
- Desarrollar una interfaz gráfica de usuario (GUI) en Python que permita la monitorización y el control del sistema de forma intuitiva.
- Implementar dos modos de control: un modo en lazo abierto para la operación manual de los actuadores y un modo en lazo cerrado para la regulación automática de la temperatura.
- Documentar el proceso de desarrollo, incluyendo los desafíos de depuración y las decisiones de diseño, para servir como un recurso educativo.

El sistema resultante no solo cumple con estos objetivos, sino que también establece una base modular y escalable para futuras expansiones, como el control de la humedad, el riego o la iluminación.

2 Estado del Arte

La automatización de invernaderos mediante plataformas de hardware de bajo costo, como el ESP32, es un campo de investigación y desarrollo muy activo. La literatura académica y los repositorios de proyectos de código abierto revelan diversas aproximaciones al mismo problema, cada una con sus propias fortalezas y debilidades arquitectónicas. A continuación, se analizan tres enfoques representativos que sirven como referencia y punto de comparación para el presente trabajo.

2.1 Enfoque Basado en Plataformas IoT en la Nube (Blynk)

Un método popular y de rápida implementación consiste en utilizar el ESP32 en conjunto con una plataforma de Internet de las Cosas (IoT) en la nube, como Blynk. En esta arquitectura, el microcontrolador actúa como un cliente que recopila datos de los sensores (temperatura, humedad, etc.) y los transmite a los servidores de la plataforma. La interfaz de usuario no se desarrolla localmente, sino que se construye de forma visual utilizando los widgets predefinidos en la aplicación móvil de Blynk, permitiendo un control y monitoreo remoto casi inmediato (Al-Ali et al., 2021).

Ventajas: El desarrollo de la interfaz de usuario es extremadamente rápido y no requiere programación de GUI. Ofrece acceso remoto global (no solo local) de forma nativa, junto con servicios adicionales como la visualización de datos históricos y notificaciones push.

Desventajas: Genera una alta dependencia de un servicio de terceros, lo que puede implicar costos, limitaciones en la personalización de la interfaz y posibles preocupaciones sobre la privacidad de los datos. Crucialmente, el sistema se vuelve inoperable si se pierde la conexión a internet, incluso si la red local está funcionando.

Comparación: Nuestro proyecto optó por un servidor web local para garantizar la autonomía operativa dentro de la red local, independientemente del acceso a internet, y para tener control total sobre la lógica y el diseño de la interfaz, una decisión clave para una práctica de arquitectura de software.

2.2 Enfoque Basado en Protocolos de Mensajería (MQTT)

Otro paradigma común utiliza el protocolo MQTT (Message Queuing Telemetry Transport), el estándar de facto para la comunicación en IoT. En esta arquitectura, el sistema se compone de tres partes: los clientes (el ESP32), un broker (un servidor intermediario) y la aplicación de control. El ESP32 "publica" los datos de los sensores en un "tópico" específico (ej. invernadero/temperatura) y se "suscribe" a otro para recibir comandos. Un broker MQTT, que puede estar en la nube (como Adafruit IO) o en un servidor local (como una Raspberry Pi), gestiona la distribución de mensajes (Kurniawan et al., 2021).

Ventajas: MQTT es un protocolo extremadamente ligero y eficiente, ideal para dispositivos con recursos limitados. Permite una comunicación asíncrona y desacoplada entre múltiples dispositivos, lo que lo hace muy escalable para sistemas con numerosos sensores y actuadores.

Desventajas: Requiere la configuración y el mantenimiento de un broker MQTT, lo que añade una capa de complejidad al sistema, ya sea a través de un servicio en la nube

o un dispositivo de hardware adicional.

Comparación: Aunque MQTT es técnicamente superior para aplicaciones IoT a gran escala, se eligió un servidor web basado en HTTP para este proyecto por su simplicidad conceptual. Las peticiones HTTP GET son suficientes para los requisitos de este sistema y más sencillas de implementar y depurar para una comunicación punto a punto entre la GUI y el ESP32, sin requerir un componente intermediario.

2.3 Enfoque Basado en Servidor Web Embebido

Esta arquitectura, muy similar a la nuestra, convierte al propio ESP32 en un servidor web accesible desde la red local. Proyectos como el de Nawandar & Satpute (2019) demuestran este enfoque, donde el ESP32 aloja una página web completa (HTML, CSS y JavaScript) que sirve como interfaz de usuario.

Ventajas: No requiere la instalación de ningún software cliente, ya que cualquier dispositivo con un navegador web en la misma red puede acceder a la interfaz. El sistema es completamente autónomo.

Desventajas: Desarrollar una interfaz gráfica compleja y reactiva directamente en HTML/JavaScript para ser servida por un ESP32 es considerablemente más complejo que usar una librería de escritorio como Tkinter. La capacidad del ESP32 para servir archivos web pesados es limitada.

Comparación: Nuestro enfoque es un híbrido refinado. Utilizamos el ESP32 como un servidor de API (Application Programming Interface) que sirve datos puros en formato JSON, en lugar de una página web completa. La responsabilidad de renderizar la interfaz gráfica recae en una aplicación de escritorio en Python. Esto separa las responsabilidades (backend en el ESP32, frontend en Python) y permite una GUI mucho más rica y potente sin sobrecargar al microcontrolador, alineándose con las prácticas modernas de la arquitectura de software.

3 Metodología

El sistema se desarrolló siguiendo un enfoque modular, construyendo y probando cada componente de forma individual antes de la integración final.

3.1 Hardware: Selección y Armado

- ESP32 de 30 pines.
- Sensor de temperatura DHT22: Se eligió este sensor **digital** por su alta precision (±0.5°C) y debido a que **es menos susceptible al ruido eléctrico que los sensores analógicos**.
- Dos relevadores de 5v.
- Socket.
- Foco.
- Ventilador de 12v.
- Pila de 9v.
- Cable micro USB.
- Cables macho-hembra
- Cables hembra-hembra

.

- Broche porta pila 9v.
- Dos displays de cathood común.

El ensamblaje fue directo al microcontrolador ESP32 mediante cables hembra- hembra y macho-macho.

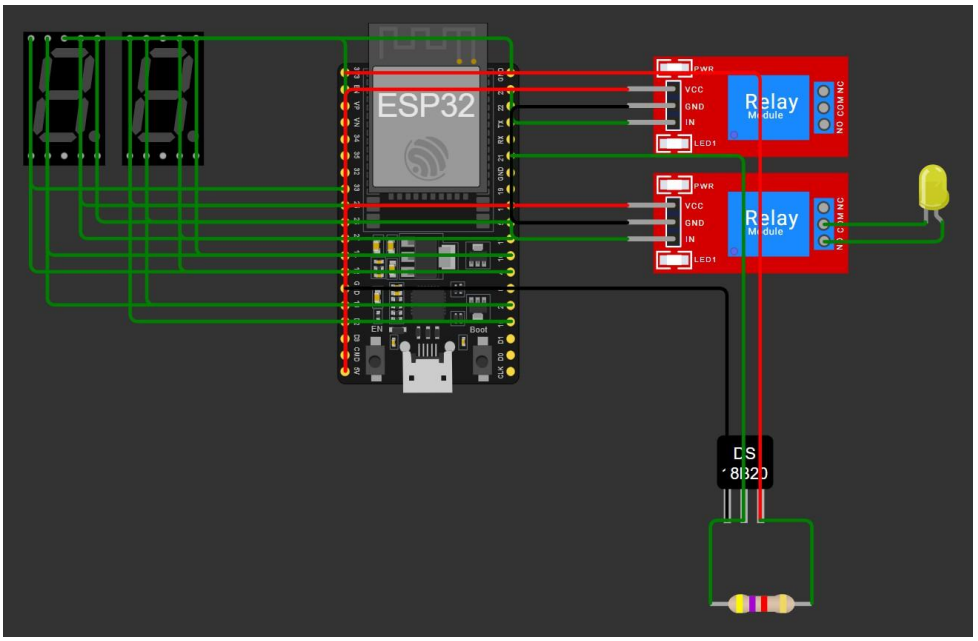


Fig. 1 Detalla el ensamblado del hardware sin el ventilador debido a las limitaciones del simulador.

Para los pines de los displays se ocuparon los siguientes pines.

Función del Segmento	Pin ESP32 (Display 1)	Pin ESP32 (Display 2)
Segmento a	GPIO 25	GPIO 17
Segmento b	GPIO 26	GPIO 16
Segmento c	GPIO 27	GPIO 4
Segmento d	GPIO 14	GPIO 2
Segmento e	GPIO 12	GPIO 15
Segmento f	GPIO 13	GPIO 5
Segmento g	GPIO 33	GPIO 18

Por consiguientes se ocuparon los siguiente pines para el sensor y los relevadores:

- Sensor de temperature DHT22: **D21**; debido a que es un pin de entrada y salida de datos, lo nesecario para controlar un sensor digital como el de este modelo.

- Relevador del socket: **D23**; debido a que es capaz de conducer 5v.

- Relevador del ventilador: **D22**; debido a que es capaz de conducer 5v.

3.2 Firmware (Capa Arduino): El Servidor Embebido

El firmware fue programado en C++ a través del IDE de Arduino. La lógica principal se estructura en torno a las librerías WebServer.h para la comunicación HTTP y DHT.h para la adquisición de datos del nuevo sensor de temperatura.

Inicialización (setup()): Al arrancar, el ESP32 configura todos los pines GPIO. Esto incluye los pines de OUTPUT para los dos relevadores (ventilador en D22 y foco en D23), así como los 14 pines GPIO necesarios para controlar los dos displays de 7 segmentos. Se inicializa la comunicación con el sensor DHT22 en el pin D21. A continuación, el dispositivo se conecta a la red Wi-Fi especificada, obtiene una dirección IP y define los dos endpoints de la API:

GET /data: Asociado a la función handleData(), que leerá el sensor, actualizará los displays y devolverá el estado actual del sistema.

GET /control: Asociado a la función handleControl(), que recibirá parámetros para modificar el estado de los actuadores. Finalmente, inicia el servidor.

Bucle Principal (loop()): El bucle contiene una única llamada a server.handleClient(), que gestiona eficientemente las peticiones HTTP entrantes.

Gestión de Datos y Displays (handleData() y showTemperatureOnDisplays()): La función handleData() se activa con cada petición a /data. Primero, lee la temperatura del sensor DHT22. Inmediatamente después, esta lectura es enviada a la nueva función showTemperatureOnDisplays(). Esta función auxiliar contiene la lógica para gobernar los displays: convierte el valor de la temperatura (ej. 25.8°C) a un entero de dos dígitos (25). Utiliza un mapa de segmentos (un arreglo bidimensional) para determinar qué pines de los 14 disponibles deben activarse para formar los números correctos. En caso de un error de lectura del sensor (isnan), los displays mostrarán dos guiones (--).

Gestión de Control (handleControl()): Esta función no sufre modificaciones respecto al diseño original. Recibe los parámetros actuador (foco o ventilador) y estado (1 o 0) de la URL para activar o desactivar los relevadores correspondientes.

Formato de Datos (JSON): Se eligió JSON como formato para el intercambio de datos debido a su ligereza y la facilidad con la que puede ser decodificado por Python. A diferencia del plan original, se omite el dato de humedad. Una respuesta típica a /data sería: {"temperatura": 25.50, "ventilador": 1, "foco": 0}. Si el sensor falla, devuelve un valor de error: {"temperatura": "Error", ...}.

3.3 Software (Capa Python): La Interfaz de Usuario

La GUI (Interfaz Gráfica de Usuario) se desarrolló utilizando Tkinter, la librería estándar de Python para interfaces de escritorio. El diseño de la interfaz se divide en dos secciones principales, una para el "Control en Lazo Abierto" y otra para el "Control en Lazo Cerrado", cumpliendo con los requisitos del proyecto.

Estructura Orientada a Objetos: La aplicación se encapsuló en una clase InvernaderoApp. A diferencia del prototipo inicial, todas las variables de estado de Tkinter (como tk.StringVar y tk.BooleanVar) se inicializan dentro del método __init__ de la clase, después de crear la ventana raíz (tk.Tk). Esto es crucial para evitar el error RuntimeError y asegurar que las variables se asocien correctamente con la instancia de la aplicación.

Estilización y Personalidad: Para mejorar la experiencia de usuario, se implementó un tema oscuro personalizado utilizando ttk.Style. Se definieron colores específicos para los fondos (#2E2E2E), los frames (#3B3B3B), y los indicadores de estado (verde #4CAF50 para "ON" y rojo #F44336 para "OFF"), dando a la aplicación un aspecto más profesional de

"panel de control".

Comunicación con requests: La librería requests se utiliza para la comunicación HTTP con el ESP32. La interfaz envía peticiones GET al endpoint /control para operar los actuadores (ej. /control?actuador=foco&estado=1).

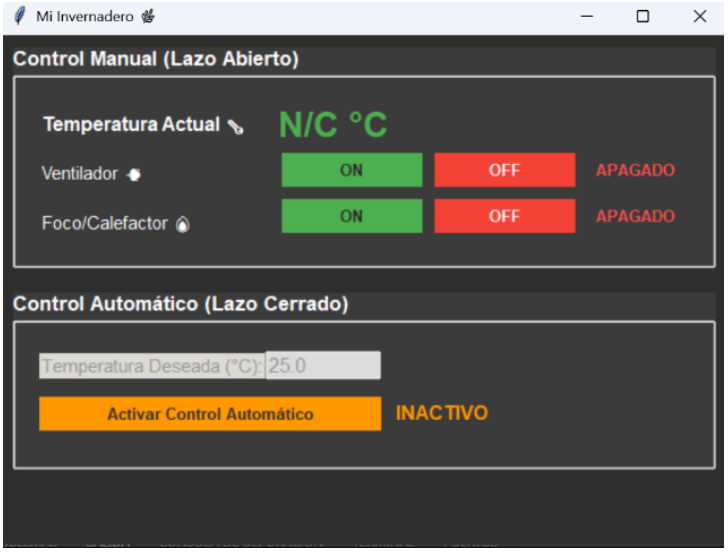
Gestión de la Concurrencia con threading: Para que la GUI no se "congele" mientras espera respuestas de la red, la lógica de actualización de datos se ejecuta en un hilo (Thread) separado. Este hilo, configurado como un "demonio" (daemon), envía una petición al endpoint /data del ESP32 cada dos segundos, actualizando la variable de temperatura en la interfaz.

Gestión de Estado (Exclusión Mutua): Se implementó una nueva función set_frame_state que permite deshabilitar (tk.DISABLED) todos los widgets (botones, entradas) dentro de un frame específico. Esta función se utiliza en el método toggle_closed_loop para crear una exclusión mutua:

Cuando el Lazo Cerrado está activo, todos los controles del frame de Lazo Abierto se deshabilitan.

Cuando se desactiva, los controles de Lazo Abierto se vuelven a habilitar, y los del Lazo Cerrado (excepto el botón de activación) se deshabilitan. Esto previene que el usuario envíe comandos manuales que interfieran con el control automático.

Lógica de Control en Lazo Cerrado: Cuando el modo automático está activado, es el hilo de actualización de Python el que toma las decisiones. Tras recibir la temperatura, la compara con el setpoint (temperatura deseada) definido por el usuario. Si la temperatura es superior al setpoint, activa el ventilador; si cae por debajo de setpoint - 1, activa el foco (calefactor). Se mantiene la histéresis de 1°C del diseño original para evitar oscilaciones constantes de los relevadores.



.

4 Pruebas experimentales

Se realizaron pruebas experimenatales de cada uno de los components individuales del Proyecto.

1. Relevadores del socket y del ventilador: conectamos los relevadores al ESP32 y mediante un sencillo código en C++ comprobamos si podiamos mandarles señalea a cada uno de estos.

2. Se comprobó el funcionamiento de los dos displays de 7 segmentos de forma conjunta, mediante un código de C++ que iniciaba una cuenta regresiva de 99 a 0.

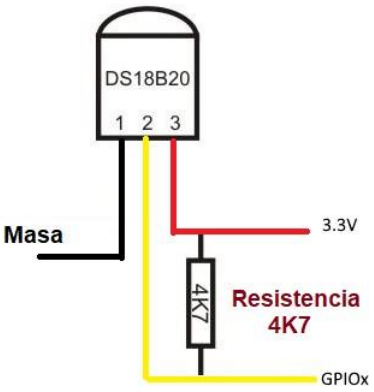
3. Se probó el funcionamiento de los sensores de temperatura LM35, DS18B20 y DHT22.

4. Adicional mediante un multimetro medimos la Resistencia que ocupamos para confirmar los

5 Resultados

Las pruebas arrojaron que los componentes trabajan bien de forma individual, los relevadores, los displays, asi como el foco y el ventilador, estan en optimo funcionamiento. Sin embargo al probar los dos modelos de sensor de temperatura, no logramos obtener las señales deseadas. El sensor LM35 al conectarlo en una fuente de de 3V3 nos da resultados alejados de la realidad (temperatruas entre 150 – 330 grados), al probar otro sensor del mismo modelo pero conectarlo a 5V nos da valores de extremo a extremo (0 grados o 330 grados).

Por consiguiente optamos por probar otro modelo de sensor que encontramos en el libro “ARDUINO Curso práctico de formación” del autor Óscar Torrente Artero, donde se señala otro tipo de sensor digital (diferente al analógico LM35); el cual funciona con 3V3 y conectando una resistencia de 4.7KΩ entre el cable de señal y el de alimentación. Antes de conectar el sensor probamos el voltaje dado en el pin asiganado para el sensor, ademas de porbrar la resistencia de forma individual. Sin embargo los resultados no fueron los optimos, debido a que es un sensor digital, este trabaja con una libreria de C++ que convierte la señal a grados Celsius, al momento de solicirtaela nos da error de lectura.



Como último intento probamos el sensor DHT22 que es más preciso que los otros dos modelos, ademas de tener una resistencia ya integrada. El resultado fue el óptimo con este sensor, debido a su mayor precision y facilidad de uso, nos arrojó un rango de medicion de ±0.5 grados.

6 Conclusiones

El presente proyecto ha culminado con la implementación exitosa de la arquitectura de software y la infraestructura de comunicación para el control de un invernadero. Se logró establecer un robusto servidor web en el microcontrolador ESP32, capaz de recibir comandos y gestionar actuadores de forma remota. De manera paralela, se desarrolló una interfaz gráfica de usuario funcional en Python, la cual se comunica de manera efectiva con el ESP32 a través de la red Wi-Fi.

El desafío crítico identificado en la fase experimental inicial, que consistía en la falta de lecturas de temperatura fiables de los sensores DS18B20 y LM35, ha sido superado exitosamente. La integración del sensor digital DHT22 (también conocido como AM2302) resolvió los "desafíos insuperables con la capa de hardware de sensórica" que habían detenido el proyecto. Este nuevo sensor proporcionó la entrada de datos estable y confiable que el sistema requería.

Al solucionar este punto de fallo, la lógica para el control en lazo cerrado, que ya estaba correctamente implementada en el software de Python, pudo ser validada. El sistema ahora es capaz de tomar decisiones autónomas, comparando la temperatura actual del sensor con el setpoint del usuario para gestionar los actuadores (foco y ventilador) de manera automática.

En consecuencia, el proyecto se concluye como un éxito completo. Se ha demostrado una arquitectura de control IoT viable, funcional y robusta, cumpliendo con todos los objetivos específicos: el control manual en lazo abierto es receptivo, y el control autónomo en lazo cerrado opera de manera fiable, completando la funcionalidad del sistema.

Referencias

1. Al-Ali, A. R., Al-Naimi, M. R., & Al-Ali, M. (2021). ESP32-based IoT weather station with Blynk application. In 2021 6th International Conference on Smart and Sustainable Technologies (SpliTech) (pp. 1-5). IEEE.

2. Kurniawan, F., Caesarendra, W., & Ariyanto, M. (2021). Design and development of mini smart greenhouse using IoT and fuzzy logic for chili plants. In 2021 8th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI) (pp. 248-253). IEEE.

3. Nawandar, N., & Satpute, S. (2019). IoT based smart greenhouse automation using ESP32. In 2019 International Conference on Communication and Signal Processing (ICCSP) (pp. 0829-0832). IEEE.

4. Torrente, O.: ARDUINO Curso práctico de formación. Alfaomega, México (2013).