Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
Laboratorio de Microcontroladores

Sistema de Control Inteligente

Junior Ruiz Sánchez - B97026
Mario Rocha Morales - B96561

II - Ciclo 2024
Profesor: Marco Villalta Fallas

Índice

Resumen

El proyecto **Sistema de Control Inteligente** consiste en el desarrollo de una solución integrada utilizando dos microcontroladores: la **Raspberry Pi Pico** y el **ESP32 WROOM (TTGO BOARD)**. La Raspberry Pi Pico utiliza el módulo ESP8266 para conectarse a internet y establecer un servidor local que maneja solicitudes mediante endpoints para realizar acciones como encender/apagar una luz, enviar un "magic packet" para "Wake-on-LAN", y recibir datos de un sensor de movimiento RCWL-0516 para activar la luz al detectar una persona. Por otro lado, el ESP32 gestiona comandos a través de un bot de Telegram para encender o apagar una cafetera, además de monitorear un sensor de gas MQ5 para detectar posibles fugas y enviar alertas mediante el bot. Finalmente, el ESP32 interactúa con la Raspberry Pi Pico mediante solicitudes locales, permitiendo así que el bot de Telegram controle tanto el ESP32 como la Raspberry Pi Pico, creando un sistema de control remoto eficiente y seguro.

Palabras clave: Microcontroladores, Raspberry Pi Pico, ESP32 WROOM, IoT, Telegram Bot, Control Remoto, Detección de Gas, Wake-on-LAN

1. Introducción

El desarrollo de sistemas de control inteligente ha revolucionado la manera en que interactuamos con nuestro entorno, permitiendo la automatización y monitoreo de dispositivos de manera remota y eficiente. En este contexto, el proyecto **Sistema de Control Inteligente** se enfoca en la integración de dos microcontroladores: la **Raspberry Pi Pico** y el **ESP32 WROOM (TTGO BOARD)**, con el propósito de crear una solución robusta que combina control remoto, monitoreo ambiental y comunicación en tiempo real.

La **Raspberry Pi Pico** se encarga de gestionar las solicitudes locales mediante el módulo WiFi ESP8266, estableciendo un servidor local que maneja diversos endpoints. Estos endpoints permiten realizar acciones como encender o apagar una luz, enviar un "magic packet" para activar la función "Wake-on-LAN", y recibir datos de un sensor de movimiento RCWL-0516 para activar la luz al detectar la presencia de una persona. Por otro lado, el **ESP32 WROOM** utiliza un bot de Telegram para recibir y procesar comandos que permiten encender o apagar una cafetera, así como monitorear un sensor de gas MQ5 para detectar posibles fugas y enviar alertas inmediatas a través del bot.

La interacción entre ambos microcontroladores se realiza mediante solicitudes HTTP locales, lo que permite que el ESP32 actúe como intermediario entre el bot de Telegram y la Raspberry Pi Pico. Esta arquitectura en cascada asegura una gestión centralizada y eficiente del sistema, optimizando tanto el control como la seguridad de los dispositivos conectados.

Durante el desarrollo de este proyecto, se adquirieron conocimientos profundos en la configuración y programación de microcontroladores, la integración de módulos de comunicación inalámbrica, y la implementación de sistemas de control y monitoreo basados en eventos. Además, se exploraron técnicas de manejo de interrupciones y temporizadores, así como la creación de interfaces de usuario a través de aplicaciones de mensajería instantánea.

2. Objetivos

2.1. Objetivo General

Desarrollar un **Sistema de Control Inteligente** que integre la Raspberry Pi Pico y el ESP32 WROOM para controlar y monitorear dispositivos domésticos de manera remota y eficiente, utilizando comunicación inalámbrica y automatización basada en eventos.

2.2. Objetivos Específicos

- Implementar un servidor local en la Raspberry Pi Pico mediante el módulo ESP8266 para gestionar solicitudes y controlar dispositivos como luces y la cafetera.
- Desarrollar un bot de Telegram en el ESP32 WROOM que permita recibir comandos para encender/apagar la cafetera y controlar otros dispositivos.
- Integrar sensores de movimiento y gas para automatizar el control de dispositivos y garantizar la seguridad mediante alertas en tiempo real.
- Establecer una comunicación eficiente entre la Raspberry Pi Pico y el ESP32 WROOM mediante solicitudes HTTP locales.

3. Alcances

3.1. Alcance del Proyecto

El **Sistema de Control Inteligente** abarca la integración de dos microcontroladores principales: la **Raspberry Pi Pico** y el **ESP32 WROOM (TTGO BOARD)**, para la gestión remota y automatizada de dispositivos domésticos. El sistema permite controlar luces y una cafetera mediante comandos de Telegram, así como monitorear la presencia de personas y detectar posibles fugas de gas, enviando alertas en tiempo real. Además, incorpora la funcionalidad de "Wake-on-LAN" para activar computadoras de manera remota, proporcionando una solución completa para la automatización y seguridad del hogar.

3.2. Limitaciones

- La dependencia de una red WiFi estable para el funcionamiento óptimo del sistema.
- La precisión de los sensores de movimiento y gas puede variar según las condiciones ambientales.
- La comunicación entre los microcontroladores está limitada a solicitudes HTTP locales, lo que requiere una configuración adecuada de la red.

4. Justificación

La automatización y el control remoto de dispositivos domésticos se han convertido en aspectos fundamentales para mejorar la comodidad, eficiencia energética y seguridad en los hogares modernos. El proyecto **Sistema de Control Inteligente** responde a esta necesidad al proporcionar una solución integrada que combina la capacidad de control remoto mediante Telegram con la automatización basada en eventos a través de sensores.

La elección de la **Raspberry Pi Pico** y el **ESP32 WROOM** se fundamenta en sus características técnicas avanzadas y su versatilidad en aplicaciones de Internet de las Cosas (IoT). La Raspberry Pi Pico, con su módulo ESP8266, permite establecer una conexión robusta a internet y gestionar múltiples solicitudes locales, mientras que el ESP32 WROOM ofrece capacidades de procesamiento y comunicación avanzadas, esenciales para la interacción con el usuario y el manejo de sensores críticos como el de gas.

Además, la implementación de sensores de movimiento y gas no solo automatiza el control de dispositivos, sino que también incrementa la seguridad del entorno, proporcionando alertas en tiempo real ante situaciones potencialmente peligrosas. La integración con Telegram facilita una interfaz de usuario accesible y eficiente, permitiendo el control y monitoreo desde cualquier lugar con conexión a internet.

Este proyecto no solo aporta valor práctico al demostrar la viabilidad de un sistema de control inteligente, sino que también contribuye al aprendizaje y desarrollo de habilidades en programación de microcontroladores, integración de sistemas IoT y manejo de comunicaciones inalámbricas.

5. Marco Teórico

5.1. Microcontroladores Utilizados

5.1.1. Raspberry Pi Pico

La **Raspberry Pi Pico** es una placa de microcontrolador basada en el chip RP2040, diseñado por la Fundación Raspberry Pi. Este microcontrolador cuenta con una arquitectura de doble núcleo ARM Cortex-M0+ a "133MHz", ofreciendo un equilibrio óptimo entre rendimiento y eficiencia energética. Entre sus características destacan:

- Memoria: 264KB de SRAM y 2MB de Flash.
- Entradas/Salidas: 26 pines GPIO programables.
- Conectividad: A través de módulos adicionales como el ESP8266 para comunicaciones inalámbricas.
- Lenguajes de Programación: Soporta C/C++ y MicroPython.

5.1.2. ESP32 WROOM (TTGO BOARD)

El **ESP32 WROOM** es un módulo de microcontrolador de doble núcleo basado en la arquitectura Tensilica Xtensa LX6, operando a "240MHz". Es conocido por sus capacidades avanzadas de conectividad y procesamiento, lo que lo hace ideal para aplicaciones IoT. Sus características principales incluyen:

- Memoria: 520KB de SRAM y 4MB de Flash.
- Conectividad: WiFi 802.11 b/g/n y Bluetooth 4.2 BR/EDR y BLE.
- Entradas/Salidas: Más de 30 pines GPIO con múltiples funciones.
- Periféricos: Soporta SPI, I2C, UART, ADC, DAC, PWM, entre otros.
- Lenguajes de Programación: Soporta Arduino IDE, Espressif IDF y MicroPython.

5.2. Comunicación UART y HTTP

5.2.1. UART (Universal Asynchronous Receiver/Transmitter)

La comunicación UART es un método asíncrono de transmisión de datos que permite la comunicación serial entre dispositivos electrónicos. En este proyecto, la UART se utiliza para conectar la Raspberry Pi Pico con el módulo ESP8266, facilitando el intercambio de datos y comandos necesarios para el control remoto y la monitoreo de dispositivos.

5.2.2. HTTP (HyperText Transfer Protocol)

El protocolo HTTP es la base de la comunicación en la web, permitiendo el intercambio de información entre clientes y servidores. En este sistema, se utiliza para que el ESP32 envíe solicitudes HTTP a la Raspberry Pi Pico, controlando así dispositivos como luces a través de endpoints definidos en el servidor local.

5.3. Bots de Telegram

5.3.1. Integración con Telegram

Telegram es una aplicación de mensajería instantánea que ofrece una API robusta para la creación de bots, permitiendo la automatización de tareas y la interacción con usuarios. En este proyecto, se utiliza un bot de Telegram para recibir comandos del usuario que luego son procesados por el ESP32 para controlar dispositivos conectados y enviar alertas sobre condiciones críticas como fugas de gas.

5.4. Sensores de Movimiento y Gas

5.4.1. Sensor de Movimiento RCWL-0516

El RCWL-0516 es un sensor de movimiento basado en radar Doppler, capaz de detectar la presencia de personas mediante cambios en las ondas de radio. Es altamente sensible y puede detectar movimientos a través de materiales como paredes y puertas, lo que lo hace ideal para aplicaciones de automatización de iluminación.

5.4.2. Sensor de Gas MQ5

El MQ5 es un sensor de gas que puede detectar la presencia de gases combustibles como propano, metano y gas natural. Su salida analógica varía en función de la concentración de gas en el ambiente, permitiendo la implementación de sistemas de alerta para prevenir posibles fugas y garantizar la seguridad del entorno.

6. Metodología

6.1. Descripción de la Aplicación

El **Sistema de Control Inteligente** está diseñado para gestionar y monitorear dispositivos domésticos de manera remota y automática. Utiliza la **Raspberry Pi Pico** para manejar solicitudes locales y controlar dispositivos como luces a través de un servidor local. El **ESP32 WROOM** se encarga de la interacción con el usuario mediante un bot de Telegram, permitiendo el control remoto de la cafetera y la recepción de alertas sobre posibles fugas de gas.

6.2. Lenguajes de Programación y Bibliotecas Utilizadas

- **MicroPython**: Utilizado en la Raspberry Pi Pico para la programación de scripts que gestionan el servidor local y el control de dispositivos conectados.
- **C++ (Arduino IDE)**: Empleado en el ESP32 WROOM para desarrollar el bot de Telegram y gestionar comuniciaciones con el sensor de gas y control de alimentación.
- **Bibliotecas:**
  - `CTBot`: Para la integración del bot de Telegram en el ESP32.
  - `WiFi.h`: Para la conexión a redes WiFi.
  - `HTTPClient.h`: Para realizar solicitudes HTTP desde el ESP32.
  - `esp8266_uart`: Módulo personalizado para manejar la comunicación UART entre la Raspberry Pi Pico y el ESP8266.

6.3. Proceso de Desarrollo

El desarrollo del proyecto siguió los siguientes pasos:

1. **Configuración de Hardware:** Conexión de la Raspberry Pi Pico con el módulo ESP8266, configuración de pines para sensores y relés, y montaje de dispositivos como luces y la cafetera.
2. **Programación de la Raspberry Pi Pico:** Desarrollo de scripts en MicroPython para establecer un servidor local, manejar solicitudes HTTP y controlar dispositivos conectados.
3. **Programación del ESP32 WROOM:** Implementación de un bot de Telegram en C++ utilizando el Arduino IDE, manejo de comandos de usuario y monitoreo de sensores.
4. **Integración de Sistemas:** Establecimiento de la comunicación UART entre la Raspberry Pi Pico y el ESP32, y configuración de endpoints para la interacción entre ambos microcontroladores.
5. **Pruebas y Validación:** Realización de pruebas funcionales para asegurar el correcto funcionamiento del sistema, incluyendo la detección de movimiento y fugas de gas, y la respuesta a comandos de Telegram.

7. Desarrollo y Análisis de Resultados

7.1. Implementación del Sistema

El **Sistema de Control Inteligente** se implementó integrando dos microcontroladores: la **Raspberry Pi Pico** y el **ESP32 WROOM (TTGO BOARD)**. A continuación, se describen las funcionalidades principales de cada componente y su interacción.

7.1.1. Raspberry Pi Pico con Módulo ESP8266

La Raspberry Pi Pico, equipada con el módulo ESP8266, actúa como el servidor local del sistema. Utiliza MicroPython para ejecutar scripts que establecen un servidor HTTP, manejando diversos endpoints que permiten controlar dispositivos conectados.

- **Control de Luces:** A través de endpoints específicos, se pueden encender o apagar luces conectadas al sistema.
- **Wake-on-LAN:** Mediante el envío de un "magic packet", se puede activar la función "Wake-on-LAN" para encender una computadora de manera remota.
- **Detección de Movimiento:** El sensor de movimiento RCWL-0516 detecta la presencia de personas y, al activarse, envía una señal para encender las luces automáticamente.

7.1.2. ESP32 WROOM con Bot de Telegram

El ESP32 WROOM gestiona la interacción con el usuario mediante un bot de Telegram, permitiendo el control remoto de la cafetera y la recepción de alertas sobre posibles fugas de gas.

- **Control de Cafetera:** El bot de Telegram permite encender o apagar la cafetera mediante comandos específicos.
- **Monitoreo de Gas:** El sensor MQ5 monitorea la concentración de gas en el ambiente. Si se detecta una fuga, el sistema envía alertas inmediatas a través del bot de Telegram.
- **Comunicación con Raspberry Pi Pico:** El ESP32 envía solicitudes HTTP a los endpoints de la Raspberry Pi Pico para realizar acciones como encender/apagar luces o enviar "magic packets".

7.2. Análisis del Código

A continuación, se presentan fragmentos descriptivos del código utilizados en ambos microcontroladores, junto con su explicación para comprender su funcionamiento dentro del sistema.

7.2.1. Código Principal de ESP32 ('ESP32_hostBot.ino')

El código en el ESP32 está diseñado para manejar la comunicación con Telegram, procesar comandos recibidos y controlar dispositivos conectados.

```
// Función para manejar comandos recibidos desde Telegram
void manejarComando(String comando) {
    if (comando == "cafe on") {
        digitalWrite(releCafe, HIGH);
```

```
        miBot.sendMessage(msg.sender.id, "Cafetera encendida.");
    }
    else if (comando == "cafe off") {
        digitalWrite(releCafe, LOW);
        miBot.sendMessage(msg.sender.id, "Cafetera apagada.");
    }
    // Manejo de otros comandos...
}
```

Descripción:

- **Inclusión de Bibliotecas:** Se incluyen las bibliotecas necesarias para la conexión WiFi, manejo de Telegram y solicitudes HTTP.
- **Definición de Pines:** Se definen los pines GPIO utilizados para controlar el relé y leer el sensor de gas.
- **Configuración Inicial:** En la función `setup()`, se inicializan las conexiones WiFi y Telegram, configurando los pines como entradas o salidas según corresponda.
- **Loop Principal:** En la función `loop()`, se verifica el estado de la cafetera y se manejan los mensajes de Telegram, además de monitorear continuamente el sensor de gas.
- **Manejo de Comandos:** La función `manejarComando()` procesa los comandos recibidos a través de Telegram, ejecutando acciones como encender/apagar la cafetera o controlar las luces.
- **Detección de Gas:** La función monitorea el valor leído del sensor de gas y envía alertas si se supera un umbral predefinido.

7.2.2. Descripción de los Endpoints

El sistema utiliza varios endpoints para manejar diferentes acciones mediante solicitudes HTTP. A continuación, se detallan los endpoints implementados y su funcionalidad:

- /**wol**: Este endpoint permite enviar un "magic packet" para activar la función "Wake-on-LAN" en una computadora remota. Al acceder a esta ruta, el sistema envía el paquete necesario para encender la computadora de manera remota. En el handler de este endpoint se llama a la función "connect", ya que esta reinicia el módulo WiFi. Esto es necesario ya que no se podía enviar el "magic packet" mientras el listener estaba activo.
- /**luzon**: Este endpoint se utiliza para encender las luces conectadas al sistema. Al recibir una solicitud en esta ruta, el servidor activa el relé correspondiente, lo que enciende las luces.
- /**luzoff**: Similar al anterior, este endpoint apaga las luces conectadas al sistema. Al acceder a esta ruta, el servidor desactiva el relé, apagando así las luces.
- /**goodn**: Este endpoint desactiva la detección de movimiento. Al recibir una solicitud en esta ruta, el sistema detiene el monitoreo del sensor de movimiento, evitando que las luces se activen automáticamente.
- /**goodm**: Este endpoint activa la detección de movimiento. Al acceder a esta ruta, el sistema reanuda el monitoreo del sensor de movimiento, permitiendo que las luces se activen al detectar presencia.

7.2.3. Comandos del Bot de Telegram

El bot de Telegram está diseñado para recibir y procesar comandos que permiten controlar dispositivos conectados y monitorear condiciones ambientales. A continuación, se describen los comandos implementados y su relación con los endpoints del sistema:

- "cafe on": Este comando enciende la cafetera conectada al sistema. Al recibir este comando, el ESP32 activa el relé correspondiente para encender la cafetera.
- "cafe off": Este comando apaga la cafetera. Al recibirlo, el ESP32 desactiva el relé, apagando la cafetera.
- "buenas noches": Este comando envía una solicitud al endpoint "/goodn" para desactivar la detección de movimiento, evitando que las luces se enciendan automáticamente.
- "buenos dias": Este comando envía una solicitud al endpoint "/goodm" para activar la detección de movimiento, permitiendo que las luces se enciendan al detectar presencia.
- "luz on": Este comando enciende las luces conectadas al sistema. Está relacionado con el endpoint "/luzon", que activa el relé para encender las luces.
- "luz off": Este comando apaga las luces. Está relacionado con el endpoint "/luzoff", que desactiva el relé para apagar las luces.
- "encender pc": Este comando envía una solicitud al endpoint "/wol" para enviar un "magic packet" y encender una computadora de manera remota.

7.2.4. Fragmentos de Código para Comunicación UART

A continuación, se presentan fragmentos descriptivos del código utilizado para manejar la comunicación UART entre la Raspberry Pi Pico y el módulo ESP8266, destacando la implementación de la clase `ESP8266ATWIFI` y su uso en la gestión de endpoints y comandos AT.

```
# Handler para el endpoint /wol
def handle_wol_request():
    wifi_manager.connect()
    wifi_manager.send_magic_packet(mac_address)
    wifi_manager.start_server()
```

Descripción: Handler de "/wol": Al recibir una solicitud en el endpoint "/wol", se llama a la función `handle_wol_request()`, que reinicia el módulo WiFi y envía el "magic packet" para encender la computadora remota.

7.2.5. Clase `ESP8266ATWIFI` en 'esp8266_uart.py'

La clase `ESP8266ATWIFI` maneja la configuración y comunicación con el módulo ESP8266 mediante comandos AT, facilitando la conexión WiFi, la gestión de endpoints y el envío de "magic packets" para Wake-on-LAN.

```
class ESP8266ATWIFI:
    def __init__(self, ssid, password, uart_num, tx_pin, rx_pin, baudrate=115200,
                 I_WANT_TO_DEPURATE_THIS_FS=True, default_not_found_message="Endpoint no
        self.ssid = ssid
        self.password = password
```

```
        self.uart = UART(uart_num, baudrate=baudrate,
                         tx=Pin(tx_pin), rx=Pin(rx_pin))
        self.endpoints = {}
        self.I_WANT_TO_DEPURATE_THIS_FS = I_WANT_TO_DEPURATE_THIS_FS
        self.default_not_found_message = default_not_found_message

    def connect(self, max_retries=3):
        self.debug_print("[WiFiManager] Reiniciando el módulo WiFi...")
        self.send_command('AT+RST', 2)
        time.sleep(2)
        self.uart.read()  # Limpia el buffer

        self.debug_print("[WiFiManager] Configurando modo cliente...")
        self.send_command('AT+CWMODE=1', 2)

        for attempt in range(max_retries):
            self.debug_print(
                f"[WiFiManager] Intento de conexión al WiFi ({attempt + 1}/{max_retries}
            response = self.send_command(
                f'AT+CWJAP="{self.ssid}","{self.password}"', 15)
            time.sleep(10)

            if response and (b"WIFI CONNECTED" in response or b"WIFI GOT IP" in response
                self.debug_print("[WiFiManager] Conexión exitosa.")
                break
            else:
                self.debug_print(
                    "[WiFiManager] Error al conectar, reintentando...")
        else:
            self.debug_print(
                "[WiFiManager] Error: No se pudo conectar al WiFi tras varios intentos."

        self.debug_print("[WiFiManager] Obteniendo información de red...")
        self.send_command('AT+CIFSR', 10)
        time.sleep(1)

    def send_magic_packet(self, mac_address):
        """Genera y envía un Magic Packet para Wake-on-LAN."""
        mac_bytes = binascii.unhexlify(
            mac_address.replace(":", "").replace("-", ""))
        if len(mac_bytes) != 6:
            raise ValueError("La dirección MAC debe tener 6 bytes.")

        magic_packet = b'\xFF' * 6 + mac_bytes * 16
        broadcast_ip = "255.255.255.255"
        port = 9

        self.send_command(f'AT+CIPSTART="UDP","{broadcast_ip}",{port}', 2)
        self.send_command(f'AT+CIPSEND={len(magic_packet)}')
        self.uart.write(magic_packet)
```

```
        time.sleep(1)
        self.send_command('AT+CIPCLOSE')

    # Métodos adicionales como send_command, await_response, etc.
```

Descripción:

- **Inicialización:** Configura la conexión UART con los parámetros especificados, incluyendo SSID, contraseña y pines de transmisión y recepción.
- **Método `connect()`:** Reinicia el módulo WiFi y establece la conexión con la red especificada, intentando múltiples veces si es necesario.
- **Método `send_magic_packet()`:** Genera y envía un "magic packet" a la dirección MAC especificada para activar la función "Wake-on-LAN".

7.3. Comandos AT Utilizados con ESP8266

En el desarrollo de este proyecto, se han utilizado diversos comandos AT para configurar y controlar el módulo ESP8266. A continuación, se presentan los comandos AT más relevantes utilizados, junto con una breve descripción de su funcionamiento:

- **AT+RST**: Reinicia el módulo ESP8266. Es esencial reiniciar el módulo antes de cualquier configuración para asegurar que esté en un estado limpio.
- **AT+CWMODE=1**: Configura el ESP8266 en modo cliente (Station). Este modo permite que el módulo se conecte a una red WiFi existente.
- **AT+CWJAP="SSID","PASSWORD"**: Establece la conexión del ESP8266 a una red WiFi específica utilizando el SSID y la contraseña proporcionados.
- **AT+CIFSR**: Obtiene la dirección IP asignada al ESP8266 después de la conexión exitosa a la red WiFi.
- **AT+CIPMUX=1**: Habilita el modo de múltiples conexiones, permitiendo que el ESP8266 maneje más de una conexión simultáneamente.
- **AT+CIPSERVER=1,80**: Inicia un servidor HTTP en el puerto 80, permitiendo que el ESP8266 escuche y responda a solicitudes HTTP entrantes.
- **AT+CIPSTART="UDP","255.255.255.255",9**: Establece una conexión UDP hacia la dirección de broadcast para enviar "magic packets" en el puerto 9, utilizado para Wake-on-LAN.
- **AT+CIPSEND=<length>**: Indica al ESP8266 que está listo para recibir datos a enviar a través de la conexión establecida. El parámetro `<length>` especifica la cantidad de bytes a enviar.
- **AT+CIPCLOSE**: Cierra la conexión actual, liberando recursos y permitiendo nuevas conexiones.

Descripción de Uso:

- Los comandos AT son enviados al ESP8266 a través de la interfaz UART desde la Raspberry Pi Pico o el ESP32 WROOM.

- Cada comando AT es seguido de una espera para recibir la respuesta del módulo, asegurando que la acción se ha completado correctamente antes de proceder con el siguiente comando.
- La implementación de estos comandos permite configurar el ESP8266 para conectarse a una red WiFi, iniciar un servidor HTTP para manejar solicitudes, y enviar "magic packets" para activar dispositivos remotos mediante Wake-on-LAN.

8. Conclusiones y Recomendaciones

8.1. Conclusiones

El desarrollo del proyecto **Sistema de Control Inteligente** permitió la creación de una solución integrada que combina la capacidad de control remoto mediante Telegram con la automatización basada en eventos a través de sensores. La Raspberry Pi Pico y el ESP32 WROOM demostraron ser componentes efectivos para gestionar y monitorear dispositivos domésticos, ofreciendo una interfaz de usuario accesible y una respuesta rápida ante condiciones críticas como fugas de gas.

Las pruebas realizadas confirmaron la funcionalidad correcta de las principales características del sistema, incluyendo el control de luces, la función de "Wake-on-LAN", y la detección de movimiento y gas. Además, la integración de la comunicación UART y la gestión de endpoints HTTP entre ambos microcontroladores resultaron eficientes, asegurando una interacción fluida y confiable entre los componentes del sistema.

8.2. Observaciones

Durante el desarrollo del proyecto, se identificaron algunas limitaciones relacionadas con la precisión de los sensores y la dependencia de una red WiFi estable para el funcionamiento óptimo del sistema. Asimismo, la gestión de múltiples comandos simultáneos en el bot de Telegram podría mejorarse para evitar posibles conflictos en la ejecución de acciones.

8.3. Mejoras deseadas

- **Optimización de Sensores:** Implementar calibraciones adicionales para los sensores de gas y movimiento para mejorar la precisión y reducir falsas detecciones.
- **Mejora en la Comunicación:** Considerar el uso de protocolos de comunicación más robustos o la implementación de redundancias para asegurar una conexión estable entre los microcontroladores.
- **Interfaz de Usuario:** Desarrollar una interfaz más avanzada en Telegram que permita una interacción más intuitiva y personalizada, incluyendo la visualización de estados en tiempo real.
- **Seguridad del Sistema:** Implementar medidas adicionales de seguridad para proteger los endpoints y garantizar que solo usuarios autorizados puedan enviar comandos al sistema.
- **Escalabilidad:** Diseñar el sistema de manera que permita la integración de nuevos dispositivos y funcionalidades en el futuro, ampliando así su aplicabilidad y utilidad.

9. Bibliografía

Referencias

[1] Raspberry Pi Foundation. (2021). *Raspberry Pi Pico*. https://www.raspberrypi.com/products/rp2040/specifications/

[2] Espressif Systems. (2020). *ESP32 Datasheet*. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[3] Muscat, J. (2020). *CTBot Library for Arduino and ESP32*. Arduino Documentation. https://docs.arduino.cc/libraries/ctbot/

[4] Mantech. (s.f.). *RCWL-0516 Microwave Radar Sensor Module Datasheet*. https://www.mantech.co.za/Datasheets/Products/RCWL-0516.pdf

[5] Hanwei Electronics Co., Ltd. (2020). *MQ-5 Gas Sensor Datasheet*. https://files.seeedstudio.com/wiki/Grove-Gas_Sensor-MQ5/res/MQ-5.pdf

[6] Embedded server with Wake on LAN function https://ieeexplore.ieee.org/abstract/document/5167657

[7] Python Software Foundation. (2021). *MicroPython Documentation*. https://docs.micropython.org/en/latest/

10. Anexos

10.1. Fragmentos de Código

10.1.1. Fragmento de Código para Manejo de Comandos en ESP32

A continuación, se presenta un fragmento descriptivo del código utilizado en el ESP32 para manejar comandos de Telegram y controlar dispositivos conectados.

```
// Función para manejar comandos recibidos desde Telegram
void manejarComando(String comando) {
    if (comando == "cafe on") {
        digitalWrite(releCafe, HIGH);
        miBot.sendMessage(msg.sender.id, "Cafetera encendida.");
    }
    else if (comando == "cafe off") {
        digitalWrite(releCafe, LOW);
        miBot.sendMessage(msg.sender.id, "Cafetera apagada.");
    }
    // Manejo de otros comandos...
}
```

10.1.2. Fragmento de Código para Comunicación UART

A continuación, se muestra un fragmento descriptivo del código utilizado para manejar la comunicación UART entre la Raspberry Pi Pico y el módulo ESP8266.

```
# Handler para el endpoint /wol
def handle_wol_request():
    wifi_manager.connect()
    wifi_manager.send_magic_packet(mac_address)
    wifi_manager.start_server()
```

10.1.3. Clase `ESP8266ATWIFI` en 'esp8266_uart.py'

La clase `ESP8266ATWIFI` maneja la configuración y comunicación con el módulo ESP8266 mediante comandos AT, facilitando la conexión WiFi, la gestión de endpoints y el envío de "magic packets" para Wake-on-LAN.

```
class ESP8266ATWIFI:
    def __init__(self, ssid, password, uart_num, tx_pin, rx_pin, baudrate=115200,
                 I_WANT_TO_DEPURATE_THIS_FS=True, default_not_found_message="Endpoint no
        self.ssid = ssid
        self.password = password
        self.uart = UART(uart_num, baudrate=baudrate,
                         tx=Pin(tx_pin), rx=Pin(rx_pin))
        self.endpoints = {}
        self.I_WANT_TO_DEPURATE_THIS_FS = I_WANT_TO_DEPURATE_THIS_FS
        self.default_not_found_message = default_not_found_message

    def connect(self, max_retries=3):
        self.debug_print("[WiFiManager] Reiniciando el módulo WiFi...")
```

```
        self.send_command('AT+RST', 2)
        time.sleep(2)
        self.uart.read()  # Limpia el buffer

        self.debug_print("[WiFiManager] Configurando modo cliente...")
        self.send_command('AT+CWMODE=1', 2)

        for attempt in range(max_retries):
            self.debug_print(
                f"[WiFiManager] Intento de conexión al WiFi ({attempt + 1}/{max_retries}
            response = self.send_command(
                f'AT+CWJAP="{self.ssid}","{self.password}"', 15)
            time.sleep(10)

            if response and (b"WIFI CONNECTED" in response or b"WIFI GOT IP" in response
                self.debug_print("[WiFiManager] Conexión exitosa.")
                break
            else:
                self.debug_print(
                    "[WiFiManager] Error al conectar, reintentando...")
        else:
            self.debug_print(
                "[WiFiManager] Error: No se pudo conectar al WiFi tras varios intentos."

        self.debug_print("[WiFiManager] Obteniendo información de red...")
        self.send_command('AT+CIFSR', 10)
        time.sleep(1)

    def send_magic_packet(self, mac_address):
        """Genera y envía un Magic Packet para Wake-on-LAN."""
        mac_bytes = binascii.unhexlify(
            mac_address.replace(":", "").replace("-", ""))
        if len(mac_bytes) != 6:
            raise ValueError("La dirección MAC debe tener 6 bytes.")

        magic_packet = b'\xFF' * 6 + mac_bytes * 16
        broadcast_ip = "255.255.255.255"
        port = 9

        self.send_command(f'AT+CIPSTART="UDP","{broadcast_ip}",{port}', 2)
        self.send_command(f'AT+CIPSEND={len(magic_packet)}')
        self.uart.write(magic_packet)
        time.sleep(1)
        self.send_command('AT+CIPCLOSE')

    # Métodos adicionales como send_command, await_response, etc.
```

Descripción:

- **Inicialización:** Configura la conexión UART con los parámetros especificados, incluyendo SSID, contraseña y pines de transmisión y recepción.

- **Método `connect()`:** Reinicia el módulo WiFi y establece la conexión con la red especificada, intentando múltiples veces si es necesario.
- **Método `send_magic_packet()`:** Genera y envía un "magic packet" a la dirección MAC especificada para activar la función "Wake-on-LAN".