Sistema de Riego Automatizado con Raspberry Pi, ESP32 y Comunicación Vía MQTT y HTTP

Colaboración
Arnulfo Gamaliel Hernández González; Daniela Lizbeth Hernández Sánchez; Oscar Omar García Vázquez; José Luis Fernández Jiménez, Tecnológico Nacional de México / Instituto Tecnológico Superior de Misantla

Fecha de recepción: 13 de febrero de 2025
Fecha de aceptación: 08 de agosto de 2025

RESUMEN: Este artículo presenta el desarrollo de un sistema de riego automatizado basado en tecnologías IoT, orientado a optimizar el consumo de agua en cultivos mediante una arquitectura de bajo costo y alta escalabilidad. El sistema utiliza una Raspberry Pi como servidor central ejecutando Flask en Python y una base de datos MariaDB para almacenar eventos y configuraciones. Además, implementa dos módulos ESP32: uno configurado como Access Point con portal cautivo para programar horarios de riego mediante HTTP, y otro como cliente MQTT encargado de activar o desactivar un relé que controla la bomba de agua.

El objetivo principal es proporcionar una solución adaptable que reduzca la necesidad de intervención humana y ofrezca herramientas para la gestión precisa del riego. El sistema permite el control tanto automatizado como manual, demostrando su aplicabilidad en diferentes tipos de cultivos. Los resultados muestran un funcionamiento estable en un entorno de prueba controlado, evidenciando el potencial del sistema para escalar mediante la integración de sensores ambientales o modelos de inteligencia artificial.

PALABRAS CLAVE: IoT, MQTT, riego auomatizado, Raspberry Pi, ESP32, agricultura inteligente.

ABSTRACT: This article presents the development of an automated irrigation system based on IoT technologies, designed to optimize water consumption in crops through a low-cost and highly scalable architecture. The system employs a Raspberry Pi as a central server running Flask in Python and a MariaDB database to store events and configurations. It also integrates two ESP32 modules: one configured as an Access Point with a captive portal for scheduling irrigation times via HTTP, and another operating as an MQTT client responsible for activating or deactivating a relay that controls the water pump.

The main objective is to provide an adaptable solution that reduces the need for human intervention and offers precise tools for irrigation management. The system supports both automated and manual control, demonstrating its applicability in various types of crops. The results show stable performance in a controlled test environment, highlighting the system's potential to scale through the integration of environmental sensors or artificial intelligence models.

KEYWORDS: IoT, MQTT, automated irrigation, Raspberry Pi, ESP32, smart agriculture.

INTRODUCCIÓN

La agricultura actual enfrenta limitaciones importantes relacionadas con la disponibilidad de agua y la necesidad de mejorar la eficiencia en los sistemas de riego, especialmente en cultivos de alta demanda hídrica [1]. Los métodos tradicionales suelen provocar un uso excesivo del recurso y carecen de mecanismos de control preciso, lo que afecta la sostenibilidad y la productividad. En este contexto, las tecnologías

de Internet de las Cosas (IoT) han emergido como una alternativa viable para automatizar procesos, monitorear variables ambientales y optimizar el consumo de agua mediante sistemas inteligentes conectados [2].

El uso de dispositivos IoT ha aumentado significativamente durante la última década, alcanzando los 16.7 mil millones de dispositivos en 2024, con proyecciones de llegar a 25.4 mil millones para 2030 [3]. Esta expansión tecnológica ha impulsado también el crecimiento del mercado de riego inteligente, que busca resolver las ineficiencias de los sistemas manuales mediante la integración de sensores, conectividad inalámbrica y automatización [4], [5]. En particular, la adopción de protocolos ligeros como MQTT y la disponibilidad de microcontroladores de bajo costo como el ESP32 han facilitado el desarrollo de soluciones accesibles para pequeños productores.

A pesar de los avances, muchos sistemas de riego automatizado requieren infraestructura compleja, conexión constante a internet o computadoras dedicadas, lo que limita su adopción en entornos rurales. Por ello, resulta relevante investigar alternativas basadas en arquitecturas locales, económicas y escalables, capaces de operar sin dependencia de servicios externos.

Hipótesis

La implementación de un sistema de riego automatizado basado en IoT, utilizando Raspberry Pi y ESP32 con protocolos HTTP y MQTT, permite mejorar la eficiencia y el control del riego al reducir la intervención manual y optimizar el uso del agua en comparación con métodos tradicionales.

Con base en lo anterior, el objetivo de este trabajo es diseñar e implementar un sistema de riego automatizado de bajo costo que permita programar, monitorear y controlar el suministro de agua mediante una arquitectura modular basada en Raspberry Pi, ESP32 y comunicación IoT, demostrando su funcionalidad y potencial de escalabilidad para futuras mejoras.

MATERIAL Y MÉTODOS

El desarrollo del sistema de riego automatizado se llevó a cabo en varias etapas, asegurando una integración eficiente de hardware y software. El sistema de riego se diseñó con los siguientes componentes:

ESP32 como Access Point para permitir la configuración del sistema mediante una interfaz web.

ESP32 como cliente MQTT encargado de recibir comandos desde la Raspberry Pi y activar la bomba de agua mediante un relé.

Raspberry Pi como servidor central responsable del almacenamiento de datos en MariaDB, la comunicación con los ESP32 y la gestión del riego.

Protocolo MQTT y HTTP para la comunicación eficiente entre dispositivos.

Metodología

La metodología empleada aborda tanto el desarrollo del hardware como del software, así como las pruebas realizadas en un entorno controlado.

Arquitectura del Sistema: El sistema está compuesto por:

- Raspberry Pi 4B como servidor central con:
  - Flask en Python (versión 3.10)
  - Base de datos MariaDB
  - Servicio MQTT (Mosquitto Broker)
- ESP32 Access Point programado en Arduino IDE con:
  - Biblioteca WiFi.h
  - Biblioteca WebServer.h
  - Portal cautivo
  - Envío de datos mediante solicitudes HTTP
- ESP32 cliente MQTT programado en Arduino IDE, utilizando:
  - WiFi.h
  - PubSubClient
  - Control de relé en pin digital
- Relé de 120V para activar/desactivar la bomba de agua.

Base de Datos: Se utilizó MariaDB con la siguiente estructura:

- tabla eventos_riego:

id_evento (int, PK)
fecha_inicio (timestamp)
fecha_fin (timestamp)
duracion_segundos (int)
estado_rele (tinyint)
modo (varchar – manual/automático)

Toda la información enviada desde los ESP32 se almacena para consulta posterior.

Desarrollo del Software

ESP32 Access Point

- Red WiFi: ESP32_AP
- IP estática: 192.168.4.1
- Formulario HTML para capturar: Fecha, Hora de Encendido, Hora de Apagado.

Los datos se envían en JSON vía HTTP hacia la Raspberry Pi.

Raspberry Pi – Servidor Flask: Endpoints principales:

- /encender – activa el relé y registra el evento.
- /apagar – desactiva la bomba y registra la finalización.
- /guardar – almacena datos provenientes del ESP32 AP.

ESP32 Cliente MQTT: Se suscribe al tópico riego/control:

- o Mensaje “ON” - activa relé
- o Mensaje “OFF” - desactiva relé

Entorno de Pruebas: El sistema fue evaluado en laboratorio con una bomba real de 120V, un depósito de 20 litros y ciclos cortos de riego. Se midió latencia, estabilidad del broker y funcionamiento continuo durante 48 horas.

RESULTADOS

Raspberry Pi como Servidor y Centro de Control Sistema de riego automatizado con Raspberry Pi

Este sistema de riego parte de la idea de eliminar la necesidad de contar con una computadora dedicada exclusivamente al control del riego en cultivos, reduciendo así los costos asociados a la adquisición y mantenimiento de un sistema de cómputo. Para lograrlo, se implementó una Raspberry Pi como servidor central, configurado con Flask en Python, el cual recibe solicitudes HTTP con información sobre la fecha, hora de encendido y apagado del riego. Estos datos son almacenados en una base de datos MariaDB, alojada en la misma Raspberry Pi.

El usuario interactúa con el sistema a través de su celular, conectándose a la red ESP_AP, generada por el dispositivo ESP32 Access Point. Una vez conectado, se abre automáticamente un sitio web con un formulario donde el usuario puede programar los horarios de riego. Posteriormente, el servidor monitorea la información ingresada y, cuando llega el momento de activar la bomba de agua, envía una solicitud HTTP a al dispositivo ESP32 cliente MQTT, indicando la orden de encendido o apagado.

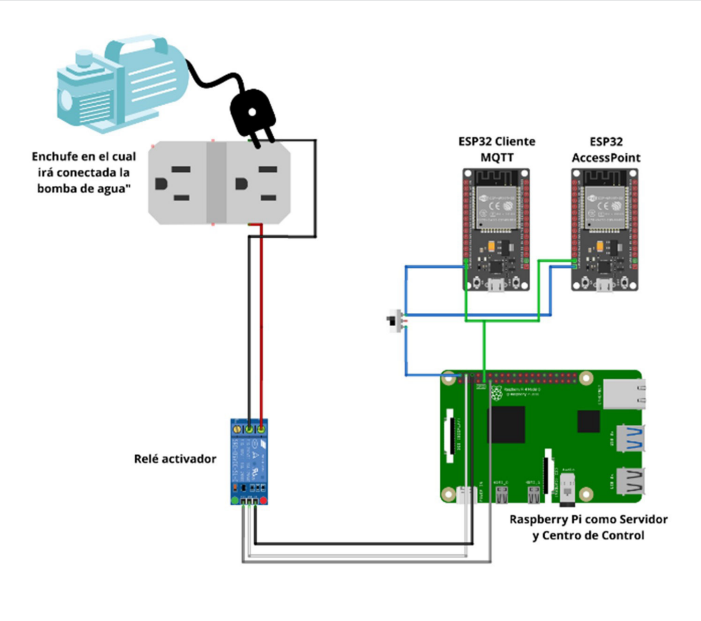


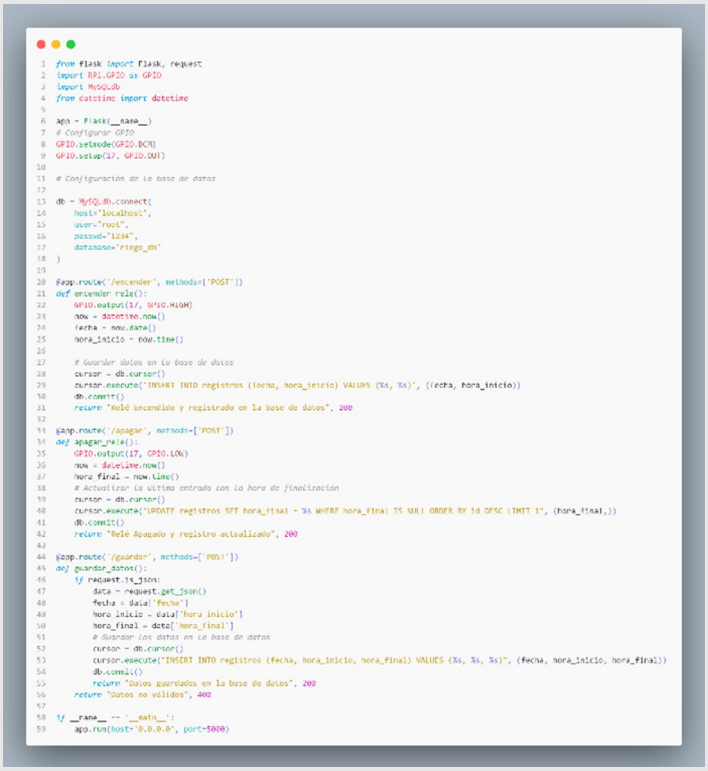
Figura 1. Iteraciones para el desarrollo de la aplicación web adaptativa.
Fuente: Elaboración propia

Al recibir la orden, el ESP32 cliente MQTT envía una confirmación mediante MQTT al servidor Flask en la Raspberry Pi, que actúa como centro de control quien finalmente, activa o desactiva el relé encargado de abrir o cerrar el circuito de 120V, permitiendo así el encendido o apagado automático de la bomba de agua en tiempo real. El servidor también ofrece la opción de activar o desactivar la bomba de agua de forma inmediata, sin necesidad de una programación previa.

Raspberry Pi como Servidor y Centro de Control

La Raspberry Pi actúa como servidor central, ejecutando una aplicación web desarrollada con Flask en Python, que recibe solicitudes HTTP desde un ESP32 Access Point y almacena la información en una base de datos MariaDB. El servidor está diseñado para gestionar el encendido y apagado de una bomba de agua mediante una Raspberry Pi y módulos ESP32, utilizando los protocolos HTTP y MQTT.

El sistema permite tanto la programación de riego como la activación manual en tiempo real. Dispone de tres endpoints principales: /encender, que activa el relé y registra el evento en la base de datos; /apagar, que desactiva la bomba y actualiza el tiempo de finalización; y /guardar, que almacena datos personalizados en formato JSON. A través de una interfaz web, los usuarios pueden configurar y monitorear el riego desde cualquier dispositivo conectado a la red local, optimizando el consumo de agua y automatizando su gestión sin necesidad de una computadora dedicada exclusivamente a esta tarea.



```python
from flask import Flask, request
import RPi.GPIO as GPIO
import MySQLdb
from datetime import datetime

app = Flask(__name__)
# Configurar GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)

# Configuración de la base de datos

db = MySQLdb.connect(
    host="localhost",
    user="root",
    passwd="1234",
    database="riego_db"
)

@app.route('/encender', methods=['POST'])
def encender_rele():
    GPIO.output(17, GPIO.HIGH)
    now = datetime.now()
    fecha = now.date()
    hora_inicio = now.time()

    # Guardar datos en la base de datos
    cursor = db.cursor()
    cursor.execute("INSERT INTO registros (fecha, hora_inicio) VALUES (%s, %s)", (fecha, hora_inicio))
    db.commit()
    return "Relé encendido y registrado en la base de datos", 200

@app.route('/apagar', methods=['POST'])
def apagar_rele():
    GPIO.output(17, GPIO.LOW)
    now = datetime.now()
    hora_final = now.time()
    # Actualizar la última entrada con la hora de finalización
    cursor = db.cursor()
    cursor.execute("UPDATE registros SET hora_final = %s WHERE hora_final IS NULL ORDER BY id DESC LIMIT 1", (hora_final,))
    db.commit()
    return "Relé Apagado y registro actualizado", 200

@app.route('/guardar', methods=['POST'])
def guardar_datos():
    if request.is_json:
        data = request.get_json()
        fecha = data['fecha']
        hora_inicio = data['hora_inicio']
        hora_final = data['hora_final']
        # Guardar los datos en la base de datos
        cursor = db.cursor()
        cursor.execute("INSERT INTO registros (fecha, hora_inicio, hora_final) VALUES (%s, %s, %s)", (fecha, hora_inicio, hora_final))
        db.commit()
        return "Datos guardados en la base de datos", 200
    return "Datos no válidos", 400

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Figura 2. Entorno de desarrollo.
Fuente: Elaboración propia.

ESP32 Access Point

El desarrollo de un portal cautivo con ESP32 para la recolección de datos mediante IoT consiste en la implementación de un sistema donde el ESP32 actúa como punto de acceso (AP) y servidor web para capturar datos de fecha y hora, los cuales son transmitidos a un servidor alojado en una Raspberry Pi. El ESP32 se configura en modo AP, creando una red Wi-Fi denominada “ESP32_AP” con una dirección IP estática (192.168.4.1) mediante la biblioteca WiFi.h. A través de un portal cautivo, los usuarios ingresan la información en un formulario HTML, cuyas peticiones HTTP son gestionadas con la biblioteca WebServer.h. Posteriormente, los datos son enviados en formato JSON a la Raspberry Pi mediante solicitudes HTTP. Los resultados muestran que este enfoque permite la captura y transmisión eficiente de datos en una red local sin necesidad de conexión a internet, aunque se identificaron mejoras potenciales en la gestión de errores de comunicación. En conclusión, la combinación del ESP32 y la Raspberry Pi ofrece una solución viable para la automatización y monitoreo en entornos IoT, facilitando la gestión de información en aplicaciones sin acceso a redes externas.

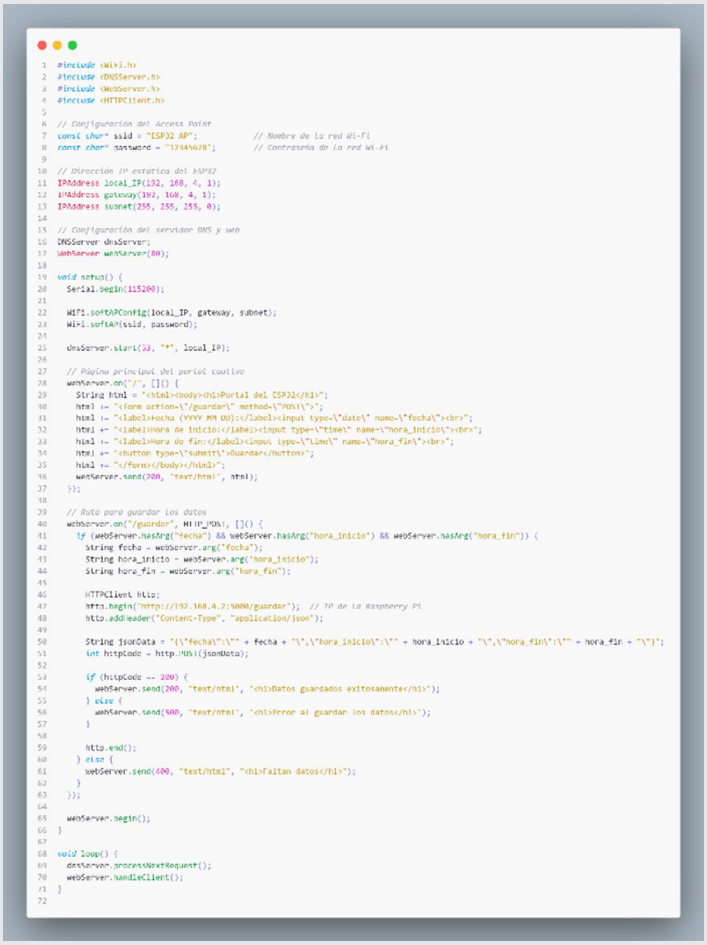


Figura 3. Validación de datos recibidos en la base de datos.
Fuente: Elaboración propia.

ESP32 cliente MQTT

el ESP32 se conecta a una red Wi-Fi para comunicarse con un servidor MQTT alojado en una Raspberry Pi. Utilizando la biblioteca WiFi.h, el ESP32 establece la conexión inalámbrica y, mediante la biblioteca PubSubClient, se configura como un cliente MQTT que se suscribe al tema “riego/control” para recibir comandos de activación o desactivación de un relé conectado al pin 2. Los mensajes en formato “ON” o “OFF” permiten encender o apagar el sistema de riego de manera remota y eficiente. Las pruebas realizadas demostraron una comunicación estable con baja latencia, lo que garantiza una respuesta rápida en escenarios agrícolas. En conclusión, la combinación del ESP32 y MQTT proporciona una solución escalable y eficiente para la automatización del riego, permitiendo la gestión remota de dispositivos y la optimización del uso del agua mediante la integración con una Raspberry Pi.

```
#include <WiFi.h>
#include <PubSubClient.h>

// Configuración Wi-Fi
const char* ssid = "ESP32_AP";
const char* password = "12345678";

// Configuración MQTT
const char* mqtt_server = "192.168.4.2";  // IP de la Raspberry Pi con el broker Mosquitto
const char* topic = "riego/control";      // Tema MQTT para encender/apagar el relé
WiFiClient espClient;
PubSubClient client(espClient);

// Configuración del pin del relé
#define RELE_PIN 2

void callback(char* topic, byte* payload, unsigned int length) {
  String message;
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  if (message == "ON") {
    digitalWrite(RELE_PIN, HIGH);
    Serial.println("Relé ENCENDIDO");
  } else if (message == "OFF") {
    digitalWrite(RELE_PIN, LOW);
    Serial.println("Relé APAGADO");
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(RELE_PIN, OUTPUT);
  digitalWrite(RELE_PIN, LOW);

  // Conectar a Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando a WiFi...");
  }
  Serial.println("Conectado a WiFi");

  // Conectar a MQTT
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);

  while (!client.connected()) {
    Serial.println("Conectando a MQTT...");
    if (client.connect("ESP32_Riego")) {
      Serial.println("Conectado a MQTT");
      client.subscribe(topic);
    } else {
      delay(5000);
    }
  }
}

void loop() {
  client.loop();
}
```

Figura 6. Vista del login.
Fuente: Elaboración propia.

Los dispositivos mantuvieron comunicación estable con latencias menores a 100 ms. El sistema ejecutó riegos automáticos exitosamente, almacenando eventos en MariaDB y permitiendo activación manual. La arquitectura mostró capacidad para escalar con sensores ambientales y módulos adicionales ESP32.

A diferencia de los sistemas tradicionales de riego automatizado, que suelen depender de una conexión permanente a internet, computadoras dedicadas o infraestructura costosa, la arquitectura propuesta opera de manera completamente local y autónoma. Esto elimina la necesidad de servicios externos y reduce significativamente los costos de implementación y mantenimiento. Además, el uso de dispositivos de bajo costo como la Raspberry Pi y el ESP32 permite desarrollar una solución accesible, modular y fácilmente replicable en diferentes contextos agrícolas, incluso en zonas rurales con conectividad limitada. Esta característica otorga una ventaja significativa frente a sistemas comerciales que requieren controladores especializados o plataformas en la nube para funcionar.

CONCLUSIONES

El sistema IoT implementado demostró ser eficiente, económico y flexible. Su arquitectura modular permite ampliaciones como sensores ambientales e inteligencia artificial. Este trabajo sienta las bases de un sistema avanzado de riego inteligente adaptable a cultivos en suelo, aeroponía e hidroponía.

AGRADECIMIENTOS

Agradecemos al Tecnológico Nacional de México (TecNM) por el apoyo financiero brindado al proyecto "Construcción de cultivo aeropónico con tratamiento de variables agroclimatológicas a través del IOT para favorecer la producción y consumo sostenible de Lactuca sativa como alternativa en la generación de autoempleo para familias de escasos recursos", el cual hizo posible el desarrollo de esta investigación.

BIBLIOGRAFÍA

[1] Food and Agriculture Organization of the United Nations (FAO), FAO Statistical Yearbook 2024. Rome: FAO, 2024. [Online]. Available: https://www.fao.org

[2] UNESCO, UN World Water Development Report 2024: Water for Prosperity and Peace. Paris: UNESCO, 2024. [Online]. Available: https://www.unesco.org

[3] IoT Analytics, "Global IoT market forecast: Number of connected IoT devices," IoT Analytics, Hamburg, Sep. 2024. [Online]. Available: https://iot-analytics.com

[4] Research and Markets, 5G IoT – Global Strategic Business Report 2024. Dublin: Research and Markets, 2025. [Online]. Available: https://www.globenewswire.com

[5] MarketsandMarkets, "Smart Irrigation Market by Component, System Type, Application, and Region – Global Forecast to 2029," MarketsandMarkets, 2024. [Online]. Available: https://www.marketsandmarkets.com

[6] Farmonaut, "Revolutionizing Water Management: Smart Irrigation Systems for Sustainable Agriculture," Farmonaut, Dec. 2024. [Online]. Available: https://farmonaut.com.

[7] La Trobe University, "AI supercharges smart irrigation for farmers," La Trobe University, Melbourne, 2024. [Online]. Available: https://www.latrobe.edu.au.