Comunicación con Alertas - Servidor de tierra y Barco

Raspberry Pi del barco: Añadimos la configuración para recibir y procesar los comandos enviados desde el servidor de tierra. El comportamiento del sistema es el siguiente.

1. **Recepción del comando**:
    - La Raspberry Pi está siempre en modo escucha, esperando mensajes del servidor de tierra en el tópico "***barco/stop***".
    - Cuando recibe el mensaje "STOP", ya sea desde la acción manual o automática, lo procesa inmediatamente.
2. **Ejecución de Apagado del programa**:
    - Al interpretar el mensaje "STOP", la Raspberry Pi termina su ejecución de publicar los datos de los sensores.

mqtt_pub_alertas.py

```
import paho.mqtt.client as mqtt
import gpiozero
import serial
import time
import sys, os

import psutil

# DEFINICION DE CONSTANTES
DISTANCIA = "Distancia:"
ACELERACION = "Aceleración (g) en X, Y, Z:"
GIROSCOPIO = "Giroscopio (grados/s) en X, Y, Z:"
COMPAS = "Compás en X, Y, Z:"
ANGULO_NORTE_X = "El ángulo en sentido horario entre el norte magnético y el eje X:"
ANGULO_NORTE_PROYECCION_X = "El ángulo en sentido horario entre el norte magnético y la proyección del eje X
positivo en el plano horizontal:"

ser = None
alerta = False

# CONFIGURACION DEL CLIENTE MQTT
#broker = "broker.emqx.io"
#broker = "127.0.0.1"
broker = "francasa.dyndns.org"
port = 1883
client = mqtt.Client("SensorPublisher")

# CONEXION AL BROKER MQTT
def connect_mqtt():
    client.connect(broker, port)
    client.on_message = on_message #Publicador de alertas
    client.subscribe("barco/stop")
    client.loop_start()

# RECEPCIÓN DE MENSAJES DE ALERTA:
def on_message(client, userdata, msg):
    global alerta
    print(f"Mensaje recibido en {msg.topic}: {msg.payload.decode()}")
    if msg.payload.decode() == "STOP":
        print("¡Programa detenido por el servidor!")
        alerta = True
        ser.close()
        client.loop_stop()
        client.disconnect()        # Cerrar puerto serial si está abierto
        sys.exit(0)

# LEER DATOS DE TEMPERATURA DE RASPI
def getRaspData():
    cpu = gpiozero.CPUTemperature()
    temp = cpu.temperature
    return temp

# INICIALIZAR LA CONEXION SERIAL
def init_serial_connection():
```

```
    global ser
    arduino_port = '/dev/ttyUSB0'
    baud_rate = 9600
    timeout_sec = 2

    try:
        ser = serial.Serial(arduino_port, baud_rate, timeout=timeout_sec)
        print("Conexión exitosa con el Arduino")
    except serial.SerialException:
        print("No se pudo conectar al Arduino. Verifica el puerto y el cable.")
        return None

# LEER DATOS DEL ARDUINO Y DEVOLVERLOS COMO UNA LISTA
def read_sensor_data():
    try:
        if ser.in_waiting > 0:
            line = ser.readline().decode('utf-8').strip()
            return line
    except Exception as e:
        print(f"Error al leer datos: {e}")
        return None

# PROCESAR LINEA POR LINEA
def procesar_linea(line):
    distance = None
    acceleration = None
    gyroscope = None
    compass = None
    heading = None
    tilt_heading = None

    if line is None:
        line = " "

    if DISTANCIA in line:
        distance = get_distance(line)
    elif ACELERACION in line:
        acceleration = get_acceleration(line)
    elif GIROSCOPIO in line:
        gyroscope = get_gyroscope(line)
    elif COMPAS in line:
        compass = get_compass(line)
    elif ANGULO_NORTE_X in line:
        heading = get_heading(line)
    elif ANGULO_NORTE_PROYECCION_X in line:
        tilt_heading = get_tilt_heading(line)

    return distance, acceleration, gyroscope, compass, heading, tilt_heading

# PUBLICACION DE CADA VALOR EN SU CORRESPONDIENTE TOPICO
def publicData(lineDistance, lineAcceleration, lineGyroscope, lineCompass, lineaHeading, lineTiltHeading,
start):
    if lineDistance is not None:
        print(f"Publicando distancia: {lineDistance}")
        client.publish("sensor/distance/MQTT", str(lineDistance))

    elif lineAcceleration is not None:
        print(f"Publicando aceleración: {lineAcceleration}")
        client.publish("sensor/acceleration/MQTT", str(lineAcceleration))

    elif lineGyroscope is not None:
        print(f"Publicando giroscopio: {lineGyroscope}")
        client.publish("sensor/gyroscope/MQTT", str(lineGyroscope))

    elif lineCompass is not None:
        print(f"Publicando compás: {lineCompass}")
        client.publish("sensor/compass/MQTT", str(lineCompass))

    elif lineaHeading is not None:
        print(f"Publicando ángulo (norte y eje X): {lineaHeading}")
        client.publish("sensor/heading/MQTT", str(lineaHeading))
```

```
    elif lineTiltHeading is not None:
        print(f"Publicando ángulo (norte y proyección eje X): {lineTiltHeading}")
        client.publish("sensor/tilt_heading/MQTT", str(lineTiltHeading))

        end = time.time()
        latencia = end - start
        print(f"Latencia: {latencia} secs")

# OBTENER DISTANCIA
def get_distance(line):
    if line and "Distancia:" in line:
        distance_str = line.split(":")[1].strip().replace("cm", "")
        return float(distance_str)
    return None

# OBTENER ACELERACION
def get_acceleration(line):
    if line and "Aceleración (g) en X, Y, Z:" in line:
        return line.split(":")[1].strip()
    return None

# OBTENER GIROSCOPIO
def get_gyroscope(line):
    if line and "Giroscopio (grados/s) en X, Y, Z:" in line:
        return line.split(":")[1].strip()
    return None

# OBTENER BRUJULA
def get_compass(line):
    if line and "Compás en X, Y, Z:" in line:
        return line.split(":")[1].strip()
    return None

# OBTENER HEADING
def get_heading(line):
    if line and "El ángulo en sentido horario entre el norte magnético y el eje X:" in line:
        return float(line.split(":")[1].strip())
    return None

# OBTENER TILTHEADING
def get_tilt_heading(line):
    if line and "El ángulo en sentido horario entre el norte magnético y la proyección del eje X positivo en el
plano horizontal:" in line:
        return float(line.split(":")[1].strip())
    return None

# MAIN
def main():
    global alerta
    connect_mqtt()
    init_serial_connection()

    try:
        while not alerta:
            temperatura = getRaspData()
            line = read_sensor_data()

            start = time.time()

            #before
            before = psutil.net_io_counters()

            client.publish("sensor/temperature/MQTT", str(temperatura))

            time.sleep(1)


            # OBTENER LOS DATOS Y PROCESARLOS
            lineDistance, lineAcceleration, lineGyroscope, lineCompass, lineaHeading, lineTiltHeading =
procesar_linea(line)
```

```
            # PUBLICACION DE DATOS
            publicData(lineDistance, lineAcceleration, lineGyroscope, lineCompass, lineaHeading,
lineTiltHeading, start)

            #after
            after =  psutil.net_io_counters()

            sent_bytes = after.bytes_sent - before.bytes_sent
            received_bytes = after.bytes_recv - before.bytes_recv
            #print(f"Sent: {sent_bytes} bytes, received {received_bytes} bytes")

    finally:
        ser.close()
        client.loop_stop()
        client.disconnect()        # Cerrar puerto serial si está abierto
        sys.exit(1)

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

Raspberry Pi del servidor de tierra: Añadimos dos funcionalidades principales para enviar un comando de apagado (stop) al barco. El comportamiento del sistema es el siguiente.

1. **Acción Manual**:
    - Añadimos un botón en el servidor de tierra.
    - Cuando se presiona este botón, se genera y envía un comando "STOP" hacia la Raspberry Pi del barco, mediante el tópico "***barco/stop***".
    - Esto nos permite tomar el control directo y apagar el programa en ejecución en la Raspberry Pi del barco en situaciones especificas.
2. **Acción automática**:
    - El servidor monitorea constantemente la temperatura.
    - Si la temperatura supera un umbral predefinido (en nuestro caso pusimos, 55°C), el sistema automáticamente genera un comando "STOP" y lo envíado al barco.
    - Esto nos asegura evitar sobrecalentamientos que podrían dañar los componentes electrónicos o causar fallas en el barco.

server_alertas.py

```
# LIBRERIAS
import cv2
import threading, sys, signal, base64, time
import numpy as np
import paho.mqtt.client as mqtt
import sqlite3
from sqlite3 import Error
from flask import Flask, request, jsonify, render_template, Response

# VARIABLES DEL SERVER FLASK
app = Flask(__name__)
server_running = True
flask_server = None

# VARIABLES GLOBALES
dataReceived = {}
dataServer = "Datos del servidor"
latest_frame = None
topicVideo = 'video/stream'
topicStop = "barco/stop"

# CONFIGURACIÓN DE BASE DE DATOS
DATABASE = "sensor_database.db"

#*************************************************************#
#********************** MySQL/MariaDB ************************#
#*************************************************************#
def create_connection():
    """Crear conexión con la base de datos SQLite"""
```

```
    conn = None
    try:
        conn = sqlite3.connect(DATABASE)
        print("Conexión a SQLite establecida")
    except Error as e:
        print(f"Error al conectar a SQLite: {e}")
    return conn

# CREAR TABLA SI ESTA NO EXISTE:
def create_table():
    """Crear las tablas necesarias"""
    conn = create_connection()
    if conn is not None:
        try:
            cursor = conn.cursor()
            cursor.execute("""CREATE TABLE IF NOT EXISTS sensor_data (
                                id INTEGER PRIMARY KEY AUTOINCREMENT,
                                timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
                                temperature TEXT,
                                distance TEXT,
                                acceleration TEXT,
                                gyroscope TEXT,
                                compass TEXT,
                                heading TEXT,
                                tilt_heading TEXT);""")
            conn.commit()
            print("Tabla sensor_data creada o ya existente")
        except Error as e:
            print(f"Error al crear la tabla: {e}")
        finally:
            conn.close()
    else:
        print("No se pudo establecer la conexión con la base de datos.")

# # FUNCIONES PARA INSERTAR CADA DATO INDIVIDUALMENTE
def insert_temperature(temperature):
    """Insertar temperatura en la base de datos"""
    conn = create_connection()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO sensor_data (temperature) VALUES (?)
            """, (temperature,))
            conn.commit()
            print("Temperatura insertada correctamente")
        except Error as e:
            print(f"Error al insertar temperatura: {e}")
        finally:
            conn.close()


def insert_distance(distance):
    """Insertar distancia en la base de datos"""
    conn = create_connection()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO sensor_data (distance) VALUES (?)
            """, (distance,))
            conn.commit()
            print("Distancia insertada correctamente")
        except Error as e:
            print(f"Error al insertar distancia: {e}")
        finally:
            conn.close()


import json
```

```
def insert_acceleration(acceleration):
    """Insertar aceleración en la base de datos"""
    conn = create_connection()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO sensor_data (acceleration) VALUES (?)
            """, (acceleration,))
            conn.commit()
            print("Aceleración insertada correctamente")
        except Error as e:
            print(f"Error al insertar aceleración: {e}")
        finally:
            conn.close()


def insert_gyroscope(gyroscope):
    """Insertar giroscopio en la base de datos"""
    conn = create_connection()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO sensor_data (gyroscope) VALUES (?)
            """, (gyroscope,))
            conn.commit()
            print("Giroscopio insertado correctamente")
        except Error as e:
            print(f"Error al insertar giroscopio: {e}")
        finally:
            conn.close()


def insert_compass(compass):
    """Insertar compás en la base de datos"""
    conn = create_connection()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO sensor_data (compass) VALUES (?)
            """, (compass,))
            conn.commit()
            print("Compás insertado correctamente")
        except Error as e:
            print(f"Error al insertar compás: {e}")
        finally:
            conn.close()


def insert_heading(heading):
    """Insertar heading en la base de datos"""
    conn = create_connection()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO sensor_data (heading) VALUES (?)
            """, (heading,))
            conn.commit()
            print("Heading insertado correctamente")
        except Error as e:
            print(f"Error al insertar heading: {e}")
        finally:
            conn.close()


def insert_tilt_heading(tilt_heading):
    """Insertar tilt heading en la base de datos"""
```

```
    conn = create_connection()
    if conn:
        try:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO sensor_data (tilt_heading) VALUES (?)
            """, (tilt_heading,))
            conn.commit()
            print("Tilt Heading insertado correctamente")
        except Error as e:
            print(f"Error al insertar tilt heading: {e}")
        finally:
            conn.close()


#*****************************************************************#
#************************** MQTT *********************************#
#*****************************************************************#
client = mqtt.Client()

# VARIABLES PARA ALMACENAR LOS DATOS DE LOS SENSORES
temp_sensor_data_MQTT = None
dist_sensor_data_MQTT = None
accel_sensor_data_MQTT = None
gyroscope_sensor_data_MQTT = None
compass_sensor_data_MQTT = None
heading_sensor_data_MQTT = None
tiltHeading_sensor_data_MQTT = None

# LISTA DE TOPICOS MQTT PARA LOS DATOS DE LOS SENSORES
TOPICS = {
    'sensor/temperature/MQTT': 'temp_sensor_data_MQTT',
    'sensor/distance/MQTT': 'dist_sensor_data_MQTT',
    'sensor/acceleration/MQTT': 'accel_sensor_data_MQTT',
    'sensor/gyroscope/MQTT': 'gyroscope_sensor_data_MQTT',
    'sensor/compass/MQTT': 'compass_sensor_data_MQTT',
    'sensor/heading/MQTT': 'heading_sensor_data_MQTT',
    'sensor/tilt_heading/MQTT': 'tiltHeading_sensor_data_MQTT',
}

# FUNCION PARA MANEJAR LA CONEXION A MQTT
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Conectado al broker MQTT")

        client.subscribe(topicVideo)

        for topic in TOPICS.keys():
            client.subscribe(topic)
            print(f"Suscrito al topico: {topic}")
    else:
        print(f"Error al conectarse al broker MQTT. Código de error: {rc}")

# FUNCION PARA PROCESAR LOS MENSAJES RECIBIDOS DE MQTT
def on_message(client, userdata, msg):
    global temp_sensor_data_MQTT, dist_sensor_data_MQTT, accel_sensor_data_MQTT
    global gyroscope_sensor_data_MQTT, compass_sensor_data_MQTT, heading_sensor_data_MQTT
    global tiltHeading_sensor_data_MQTT

    global latest_frame
    topic = msg.topic

    # VERIFICACION SI EL TOPICO ES DE VIDEO
    if topic == topicVideo:
        nparr = np.frombuffer(msg.payload, np.uint8)
        latest_frame = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
        print("Frame de video recibido a través de MQTT")

    else:
        # PROCESADO DE DATOS
        payload = msg.payload.decode('utf-8')
```

```
        # ASIGNAMOS EL DATO A LA VARIABLE CORRESPONDIENTE
        if topic == 'sensor/temperature/MQTT':
            temp_sensor_data_MQTT = payload
            print(f"Temperatura: {temp_sensor_data_MQTT}")
            insert_temperature(temp_sensor_data_MQTT)

            # ALARMA AUTOMATICA
            if float(temp_sensor_data_MQTT) > 55:
                print("Temperatura elevada.")
                client.publish(topicStop, "STOP")


        elif topic == 'sensor/distance/MQTT':
            dist_sensor_data_MQTT = payload
            insert_distance(dist_sensor_data_MQTT)
            print(f"Distancia: {dist_sensor_data_MQTT}")
        elif topic == 'sensor/acceleration/MQTT':
            accel_sensor_data_MQTT = payload
            insert_acceleration(accel_sensor_data_MQTT)
            print(f"Aceleración: {accel_sensor_data_MQTT}")
        elif topic == 'sensor/gyroscope/MQTT':
            gyroscope_sensor_data_MQTT = payload
            insert_gyroscope(gyroscope_sensor_data_MQTT)
            print(f"Giroscopio: {gyroscope_sensor_data_MQTT}")
        elif topic == 'sensor/compass/MQTT':
            compass_sensor_data_MQTT = payload
            insert_compass(compass_sensor_data_MQTT)
            print(f"Compás: {compass_sensor_data_MQTT}")
        elif topic == 'sensor/heading/MQTT':
            heading_sensor_data_MQTT = payload
            insert_heading(heading_sensor_data_MQTT)
            print(f"Heading: {heading_sensor_data_MQTT}")
        elif topic == 'sensor/tilt_heading/MQTT':
            tiltHeading_sensor_data_MQTT = payload
            insert_tilt_heading(tiltHeading_sensor_data_MQTT)
            print(f"Tilt Heading: {tiltHeading_sensor_data_MQTT}")

# FUNCION PARA INICIAR EL CLIENTE MQTT
def start_mqtt():
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect("francasa.dyndns.org", 1883)
    client.loop_start()

#*****************************************************************#
#*****************************************************************#
#*****************************************************************#

# ENVIO DEL VIDEO
def generate_video():
    global latest_frame
    while True:
        if latest_frame is not None:
            # Reduce la resolución de los fotogramas antes de transmitirlos
            img_resized = cv2.resize(latest_frame, (320, 240))
            ret, jpeg = cv2.imencode('.jpg', img_resized, [int(cv2.IMWRITE_JPEG_QUALITY), 70])
            if ret:
                frame = jpeg.tobytes()
                yield (b'--frame\r\n'
                       b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
        time.sleep(0.05)

# CAPTURA DEL BW DE VIDEO STREAMING
total_bytes_transmitted = 0
def monitor_bandwidth(interval=5):
    global total_bytes_transmitted
    while True:
        time.sleep(interval)
        # Convertir a bits por segundo
        bandwidth_bps = (total_bytes_transmitted * 8) / interval
```

```
        print(f"Ancho de banda promedio: {bandwidth_bps:.2f} bps")
        total_bytes_transmitted = 0

# Hilo para ejecutar la monitorizacion
bandwidth_thread = threading.Thread(target=monitor_bandwidth)
bandwidth_thread.start()

#*****************************************************************#
#************************** CTR+C ********************************#
#*****************************************************************#
def signal_handler(sig, frame):
    """Manejador de señal para SIGINT (Ctrl+C)."""
    print("Ctrl+C detectado. Cerrando suscriptores y servidor Flask...")

    if client is not None and client.is_connected():
        print("Cerrando cliente MQTT...")
        client.disconnect()
    if zenoh_session is not None:
        print("Cerrando sesión Zenoh...")
        zenoh_session.close()

    print("Servidor Flask cerrado correctamente.")
    sys.exit(0)

#*****************************************************************#
#************************** INICIO *******************************#
#*****************************************************************#
@app.route('/')
def dashboard():
    return render_template('dashboard_secciones.html')

#*****************************************************************#
#***************************** GET *******************************#
#*****************************************************************#
@app.route('/serverData', methods=['GET'])
def getServerData():
    response = {
        "message": "Datos recibidos exitosamente",
        "data": dataServer
    }
    return jsonify(response), 200

@app.route('/database')
def show_combined_data():
    """Obtener los 100 últimos datos independientes de cada sensor y combinarlos en columnas"""
    conn = create_connection()
    if conn:
        try:
            cursor = conn.cursor()

            # Consultas independientes para los 100 últimos datos de cada tipo
            query_temperature = "SELECT temperature FROM sensor_data WHERE temperature IS NOT NULL ORDER BY id
DESC LIMIT 100"
            query_distance = "SELECT distance FROM sensor_data WHERE distance IS NOT NULL ORDER BY id DESC
LIMIT 100"
            query_acceleration = "SELECT acceleration FROM sensor_data WHERE acceleration IS NOT NULL ORDER BY
id DESC LIMIT 100"
            query_gyroscope = "SELECT gyroscope FROM sensor_data WHERE gyroscope IS NOT NULL ORDER BY id DESC
LIMIT 100"
            query_compass = "SELECT compass FROM sensor_data WHERE compass IS NOT NULL ORDER BY id DESC LIMIT
100"
            query_heading = "SELECT heading FROM sensor_data WHERE heading IS NOT NULL ORDER BY id DESC LIMIT
100"
            query_tilt_heading = "SELECT tilt_heading FROM sensor_data WHERE tilt_heading IS NOT NULL ORDER BY
id DESC LIMIT 100"

            # Ejecución de consultas
            temperature_data = [row[0] for row in cursor.execute(query_temperature).fetchall()]
            distance_data = [row[0] for row in cursor.execute(query_distance).fetchall()]
            acceleration_data = [row[0] for row in cursor.execute(query_acceleration).fetchall()]
            gyroscope_data = [row[0] for row in cursor.execute(query_gyroscope).fetchall()]
```

```
            compass_data = [row[0] for row in cursor.execute(query_compass).fetchall()]
            heading_data = [row[0] for row in cursor.execute(query_heading).fetchall()]
            tilt_heading_data = [row[0] for row in cursor.execute(query_tilt_heading).fetchall()]

            # Combinar los datos en un diccionario para enviar a la plantilla
            combined_data = {
                "temperature": temperature_data,
                "distance": distance_data,
                "acceleration": acceleration_data,
                "gyroscope": gyroscope_data,
                "compass": compass_data,
                "heading": heading_data,
                "tilt_heading": tilt_heading_data
            }

            return render_template('combined_data.html', combined_data=combined_data, enumerate=enumerate)

        except Error as e:
            return jsonify({"error": f"Error al obtener datos combinados: {e}"}), 500
        finally:
            conn.close()
    else:
        return jsonify({"error": "No se pudo conectar a la base de datos"}), 500


@app.route('/database/<sensor_type>')
def show_sensor_data(sensor_type):
    """Obtener los datos con timestamp para un sensor específico"""
    conn = create_connection()
    if conn:
        try:
            cursor = conn.cursor()

            # Diccionario para mapear el sensor_type al campo de la base de datos
            sensor_mapping = {
                "temperature": "temperature",
                "distance": "distance",
                "acceleration": "acceleration",
                "gyroscope": "gyroscope",
                "compass": "compass",
                "heading": "heading",
                "tilt_heading": "tilt_heading",
            }

            # Verificar que el sensor solicitado existe
            if sensor_type not in sensor_mapping:
                return jsonify({"error": "Tipo de sensor no válido"}), 400

            # Consultar los datos para el sensor solicitado
            sensor_column = sensor_mapping[sensor_type]
            query = f"""
                SELECT timestamp, {sensor_column}
                FROM sensor_data
                WHERE {sensor_column} IS NOT NULL
                ORDER BY timestamp DESC
                LIMIT 100;
            """

            sensor_data = cursor.execute(query).fetchall()

            # Renderizar la plantilla
            return render_template(
                'sensor_data_individual.html',
                sensor_type=sensor_type,
                sensor_data=sensor_data
            )

        except Error as e:
            return jsonify({"error": f"Error al obtener datos: {e}"}), 500
        finally:
            conn.close()
```

```
    else:
        return jsonify({"error": "No se pudo conectar a la base de datos"}), 500



#*********************************************************************#
#*************** DATOS DE LOS SENSORES POR MQTT ******************#
#*********************************************************************#
@app.route('/mqttSensorData')
def get_mqtt_sensor_data():
    sensor_data = jsonify({
        "temperature": temp_sensor_data_MQTT,
        "distance": dist_sensor_data_MQTT,
        "acceleration": accel_sensor_data_MQTT,
        "gyroscope": gyroscope_sensor_data_MQTT,
        "compass": compass_sensor_data_MQTT,
        "heading": heading_sensor_data_MQTT,
        "tiltHeading": tiltHeading_sensor_data_MQTT
    })

    if all([temp_sensor_data_MQTT, dist_sensor_data_MQTT, accel_sensor_data_MQTT, gyroscope_sensor_data_MQTT,
            compass_sensor_data_MQTT, heading_sensor_data_MQTT, tiltHeading_sensor_data_MQTT]):
        return sensor_data
    else:
        return jsonify({"message": "No data received yet"}), 404

#*********************************************************************#
#************* DATOS DEL VIDEO ENVIADO POR MQTT ******************#
#*********************************************************************#
@app.route('/video_feed')
def video_feed():
    return Response(generate_video(), mimetype='multipart/x-mixed-replace; boundary=frame')

#*********************************************************************#
#******************** BOTON MANUAL AL BARCO **********************#
#*********************************************************************#
@app.route('/stop', methods=['POST'])
def stop():
    # PUBLICAMOS LA ALERTA
    client.publish(topicStop, "STOP")

    return '''
        <!DOCTYPE html>
        <html>
        <head>
            <title>Mensaje enviado</title>
        </head>
        <body style="text-align: center; font-family: Arial, sans-serif; margin-top: 50px;">
            <h1>¡Mensaje enviado al barco!</h1>
            <a href="/" style="text-decoration: none; font-size: 20px; color: blue;">Volver</a>
        </body>
        </html>
    '''

#*********************************************************************#
#***************************** MAIN ******************************#
#*********************************************************************#
if __name__ == '__main__':
    try:
        create_table()

        # Iniciar la simulación de datos aleatorios
        print("Iniciando simulación de datos aleatorios...")
        simulation_thread = threading.Thread(target=simulate_data, args=(5,), daemon=True)
        simulation_thread.start()

        start_mqtt()
        app.run(debug=False, host='0.0.0.0', port=8080)
    except KeyboardInterrupt:
        signal_handler(signal.SIGINT, None)
```

- **Dashboard Adicional**: Para mejorar la interacción con el sistema, hemos diseñado un archivo "**dashboard_secciones.html**", el cual nos proporciona una representación visual de los controles y estado del sistema.

dashboard_secciones.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>MQTT Dashboard</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            display: flex;
            flex-direction: column;
            justify-content: center;
            align-items: center;
            margin: 0;
            padding: 0;
            background-color: #f4f4f9;
        }
        #video-stream {
            margin: 10px;
            display: flex;
            justify-content: center;
            align-items: center;
            background-color: #000;
            padding: 10px;
            border: 2px solid #ddd;
            border-radius: 5px;
        }
        #sensor-data {
            margin: 10px;
            background: #fff;
            padding: 20px;
            border: 2px solid #ddd;
            border-radius: 5px;
            display: flex;
            flex-direction: column;
            align-items: center;
            width: 660px;
        }
        h1 {
            font-size: 1.5em;
            text-align: center;
            color: #333;
        }
        select {
            margin: 20px 0;
            padding: 10px;
            font-size: 16px;
            border-radius: 5px;
            border: 1px solid #ddd;
        }
        .alert-section {
            text-align: center;
            margin-top: 20px;
        }
        .alert-section h1 {
            color: red;
            font-size: 30px;
        }
        .red-button {
            width: 150px;
            height: 150px;
            background-color: red;
            color: white;
```

```
            font-size: 20px;
            font-weight: bold;
            border: none;
            border-radius: 50%;
            box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.3);
            cursor: pointer;
            transition: transform 0.2s;
        }
        .red-button:hover {
            transform: scale(1.1);
            box-shadow: 0px 6px 10px rgba(0, 0, 0, 0.5);
        }
        .red-button:active {
            transform: scale(0.9);
            box-shadow: 0px 2px 4px rgba(0, 0, 0, 0.2);
        }
    </style>
    <script>
        // Función para navegar a las rutas específicas de cada sensor o todos los datos juntos
        function navigateToSensorData() {
            const sensorType = document.getElementById("sensor-selector").value;
            if (sensorType) {
                window.location.href = `/database/${sensorType}`;
            }else{
                window.location.href = `/database`;
            }
        }
    </script>
</head>
<body>
    <!-- Video Stream -->
    <div id="video-stream">
        <img src="/video_feed" width="640" height="480" alt="Video Stream">
    </div>

    <!-- Sensor Data -->
    <div id="sensor-data">
        <h1>MQTT Sensor Data</h1>

        <!-- Desplegable para seleccionar sensor -->
        <select id="sensor-selector" onchange="navigateToSensorData()">
            <option value="" disabled selected>Selecciona un sensor</option>
            <option value="temperature">Temperatura</option>
            <option value="distance">Distancia</option>
            <option value="acceleration">Aceleración</option>
            <option value="gyroscope">Giroscopio</option>
            <option value="compass">Compás</option>
            <option value="heading">Heading</option>
            <option value="tilt_heading">Tilt Heading</option>
            <option value="">Histórico de Datos</option>
        </select>

        <!-- Gráfica -->
        <canvas id="sensorChart" width="640" height="480"></canvas>
    </div>

    <!-- Alert Section -->
    <div class="alert-section">
        <h1>¡NO PULSES EL BOTÓN!</h1>
        <form action="/stop" method="post">
            <button type="submit" class="red-button">NO</button>
        </form>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script>
        const ctx = document.getElementById('sensorChart').getContext('2d');
        const sensorChart = new Chart(ctx, {
            type: 'bar',
            data: {
                labels: [],
```

```
        datasets: [{
            label: 'Sensor Values',
            data: [],
            backgroundColor: [
                'rgba(255, 99, 132, 0.2)',
                'rgba(54, 162, 235, 0.2)',
                'rgba(255, 206, 86, 0.2)',
                'rgba(75, 192, 192, 0.2)',
                'rgba(153, 102, 255, 0.2)',
                'rgba(255, 159, 64, 0.2)'
            ],
            borderColor: [
                'rgba(255, 99, 132, 1)',
                'rgba(54, 162, 235, 1)',
                'rgba(255, 206, 86, 1)',
                'rgba(75, 192, 192, 1)',
                'rgba(153, 102, 255, 1)',
                'rgba(255, 159, 64, 1)'
            ],
            borderWidth: 1
        }]
    },
    options: {
        responsive: true,
        scales: {
            y: {
                beginAtZero: true
            }
        }
    }
});

// Función para actualizar el gráfico con nuevos datos
async function fetchSensorData() {
    try {
        const response = await fetch('/mqttSensorData');
        const data = await response.json();

        console.log('Datos recibidos:', data); // Mostrar datos en la consola

        const labels = [];
        const values = [];
        const backgroundColors = [];
        const borderColors = [];

        // Esquema de colores fijo para cada variable
        const colorMap = {
            temperature: 'rgba(255, 99, 132, 0.7)', // Rojo
            heading: 'rgba(54, 162, 235, 0.7)', // Azul
            tilt_heading: 'rgba(255, 255, 255, 0.7)', // Negro
            acceleration: 'rgba(255, 206, 86, 0.7)', // Amarillo
            compass: 'rgba(75, 192, 192, 0.7)', // Verde
            gyroscope: 'rgba(153, 102, 255, 0.7)', // Morado
            distance: 'rgba(201, 203, 207, 0.7)' // Gris
        };

        // Procesar los datos
        for (const [key, value] of Object.entries(data)) {
            if (key === "acceleration" || key === "compass" || key === "gyroscope") {
                const [x, y, z] = value.split(',').map(Number);

                labels.push(`${key.charAt(0).toUpperCase() + key.slice(1)} X`);
                labels.push(`${key.charAt(0).toUpperCase() + key.slice(1)} Y`);
                labels.push(`${key.charAt(0).toUpperCase() + key.slice(1)} Z`);

                values.push(x, y, z);

                const color = colorMap[key] || colorMap.default;
                backgroundColors.push(color, color, color);
                borderColors.push(color, color, color);
            } else {
```

```
                        labels.push(key);
                        values.push(value);

                        const color = colorMap[key] || colorMap.default;
                        backgroundColors.push(color);
                        borderColors.push(color);
                    }
                }

                // Actualizar el gráfico
                sensorChart.data.labels = labels;
                sensorChart.data.datasets[0].data = values;
                sensorChart.data.datasets[0].backgroundColor = backgroundColors;
                sensorChart.data.datasets[0].borderColor = borderColors;
                sensorChart.update();
            } catch (error) {
                console.error('Error fetching sensor data:', error);
            }
        }

        // Actualiza el gráfico cada 1 segundo
        setInterval(fetchSensorData, 1000);
        fetchSensorData();
    </script>
</body>
</html>
```

- **Histórico de datos común**: Para mejorar la interacción con el sistema, hemos diseñado un archivo "**combinated_data.html**", el cual nos permite ver un histórico de los 100 últimos registros de cada uno de cada uno de los datos de forma conjunta.

combined_data.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Histórico de Datos</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        table {
            border-collapse: collapse;
            width: 100%;
            margin-bottom: 20px;
        }
        th, td {
            border: 1px solid #ddd;
            text-align: center;
            padding: 8px;
        }
        th {
            background-color: #f4f4f4;
            font-weight: bold;
        }
        h1 {
            text-align: center;
            margin-top: 20px;
        }
        .back-button {
            display: block;
            width: 150px;
            margin: 20px auto;
            padding: 10px;
            text-align: center;
            text-decoration: none;
            font-size: 16px;
            font-weight: bold;
```

```
            background-color: #007BFF;
            color: white;
            border-radius: 5px;
            border: none;
            cursor: pointer;
            transition: background-color 0.3s;
            text-align: center;
        }
        .back-button:hover {
            background-color: #0056b3;
        }
    </style>
</head>
<body>
    <!-- Botón para volver a la página principal -->
    <a href="/" class="back-button">Volver a Inicio</a>

    <h1>Histórico de Datos</h1>
    <table>
        <thead>
            <tr>
                <th>#</th>
                <th>Temperatura</th>
                <th>Distancia</th>
                <th>Aceleración</th>
                <th>Giroscopio</th>
                <th>Compás</th>
                <th>Heading</th>
                <th>Tilt Heading</th>
            </tr>
        </thead>
        <tbody>
            {% for i in range(100) %}
                <tr>
                    <td>{{ i + 1 }}</td>
                    <td>{{ combined_data.temperature[i] if i < combined_data.temperature|length else '-' }}</td>
                    <td>{{ combined_data.distance[i] if i < combined_data.distance|length else '-' }}</td>
                    <td>{{ combined_data.acceleration[i] if i < combined_data.acceleration|length else '-' }}<
/td>
                    <td>{{ combined_data.gyroscope[i] if i < combined_data.gyroscope|length else '-' }}</td>
                    <td>{{ combined_data.compass[i] if i < combined_data.compass|length else '-' }}</td>
                    <td>{{ combined_data.heading[i] if i < combined_data.heading|length else '-' }}</td>
                    <td>{{ combined_data.tilt_heading[i] if i < combined_data.tilt_heading|length else '-' }}<
/td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
</body>
</html>
```

- **Histórico de datos individualizado:** Para mejorar la interacción con el sistema, hemos diseñado un archivo **sensor_data_individual.html**", el cual nos permite ver un histórico de los 100 últimos registros de cada uno de los datos.

sensor_data_individual.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Datos de {{ sensor_type.capitalize() }}</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        table {
            border-collapse: collapse;
            width: 50%;
```

```
            margin: 20px auto;
        }
        th, td {
            border: 1px solid #ddd;
            text-align: center;
            padding: 8px;
        }
        th {
            background-color: #f4f4f4;
            font-weight: bold;
        }
        h1 {
            text-align: center;
        }
        .back-button {
            display: block;
            width: 150px;
            margin: 20px auto;
            padding: 10px;
            text-align: center;
            text-decoration: none;
            font-size: 16px;
            font-weight: bold;
            background-color: #007BFF;
            color: white;
            border-radius: 5px;
            border: none;
            cursor: pointer;
            transition: background-color 0.3s;
        }
        .back-button:hover {
            background-color: #0056b3;
        }
    </style>
</head>
<body>
    <!-- Botón para volver a la página principal -->
    <a href="/" class="back-button">Volver a Inicio</a>

    <h1>Últimos 100 Datos de {{ sensor_type.capitalize() }}</h1>
    <table>
        <thead>
            <tr>
                <th>Timestamp</th>
                <th>{{ sensor_type.capitalize() }}</th>
            </tr>
        </thead>
        <tbody>
            {% for timestamp, value in sensor_data %}
                <tr>
                    <td>{{ timestamp }}</td>
                    <td>{{ value }}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
</body>
</html>
```