Comunicación BLE

Raspberry Servidor (Servidor de Tierra):

- Antes de iniciar la comunicación entre cliente y servidor. Ejecutamos el siguiente comando:

server

```
sudo rfcomm listen /dev/rfcomm0 1
```

- Este comando configura el adaptador Bluetooth en nuestro sistema para escuchar conexiones entrantes en un canal específico (en este caso, el canal 1) a través de RFCOMM, que es el protocolo de puerto serie de Bluetooth.

- En la raspberry que simula el servidor de tierra. Existirá el siguiente código que recibe los datos de los sensores por BLE:

ble_server.py

```
# LIBRERIAS
import threading
import sys
import signal
import time
import serial
from flask import Flask, jsonify, Response
import datetime, os
import json

# VARIABLES DEL SERVER FLASK
app = Flask(__name__)
server_running = True

# VARIABLES GLOBALES
total_sensor_bytes_received = 0
sensor_data = {
    "distance": None,
    "acceleration": None,
    "gyroscope": None,
    "compass": None,
    "angle_north_x": None,
    "angle_north_proj_x": None,
    "hora": None,
}

# ********************************************************************
# ************* CONFIGURACION PARA BLE ESCUCHAR LOS DATOS **************
# ********************************************************************
ble_port = '/dev/rfcomm0'
ble_baudrate = 9600
ble_connection = None

# Esperar hasta que el puerto esté disponible
def wait_for_ble_connection():
    global ble_connection
    while not os.path.exists(ble_port):
        print(f"Esperando a que {ble_port} esté disponible...")
        time.sleep(1)
    try:
        ble_connection = serial.Serial(ble_port, ble_baudrate, timeout=1)
        print(f"Conexión BLE establecida en {ble_port}")
    except serial.SerialException as e:
```

```
        print(f"Error al conectar con {ble_port}: {e}")
        ble_connection = None

# LECTURA DE DATOS DESDE BLE
def read_ble_data():
    while ble_connection and ble_connection.is_open:
        try:
            line = ble_connection.readline().decode('utf-8').strip()
            if line:
                print(f"Dato recibido por BLE: {line}")
                process_ble_data(line)
        except Exception as e:
            print(f"Error al leer desde BLE: {e}")

# Procesa cada línea recibida por BLE
def process_ble_data(line):
    global sensor_data, total_sensor_bytes_received
    try:
        total_sensor_bytes_received += len(line.encode('utf-8'))
        if "Distancia:" in line:
            sensor_data["distance"] = line.split(":")[1].strip()
        elif "Aceleración:" in line:
            sensor_data["acceleration"] = line.split(":")[1].strip()
        elif "Giroscopio:" in line:
            sensor_data["gyroscope"] = line.split(":")[1].strip()
        elif "Compás:" in line:
            sensor_data["compass"] = line.split(":")[1].strip()
        elif "Ángulo (norte y eje X):" in line:
            sensor_data["angle_north_x"] = line.split(":")[1].strip()
        elif "Ángulo (norte y proyección eje X):" in line:
            sensor_data["angle_north_proj_x"] = line.split(":")[1].strip()

            # Enviamos ACK cuando se reciba el último paquete
            send_acknowledgment("ACK")
    except Exception as e:
        print(f"Error procesando el dato: {e}")

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

# ************************************************************************
# ************* CONFIGURACION PARA BLE ENVIAR LOS DATOS ****************
# ************************************************************************
def send_acknowledgment(message):
    if ble_connection and ble_connection.is_open:
        try:
            ble_connection.write((message + '\n').encode('utf-8'))
            print(f"ACK enviado: {message}")
        except Exception as e:
            print(f"Error al enviar ACK: {e}")

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

# CAPTURA DEL BW DE LOS SENSORES
def monitor_sensor_bandwidth(interval=5):
    global total_sensor_bytes_received
    while True:
        time.sleep(interval)
        # Convertir a bits por segundo
        bandwidth_bps = (total_sensor_bytes_received * 8) / interval
        print("-----------------------------------------------")
        print(f"Ancho de banda promedio de sensores: {bandwidth_bps:.2f} bps")
        print("-----------------------------------------------")
        total_sensor_bytes_received = 0  # Reiniciar el contador

#*****************************************************************#
#************************** CTR+C ********************************#
#*****************************************************************#
```

```
def signal_handler(sig, frame):
    """Manejador de señal para SIGINT (Ctrl+C)."""
    print("Ctrl+C detectado. Cerrando conexión BLE y servidor Flask...")
    if ble_connection and ble_connection.is_open:
        ble_connection.close()
    print("Servidor Flask cerrado correctamente.")
    sys.exit(0)

signal.signal(signal.SIGINT, signal_handler)

#*****************************************************************#
#************************** FLASK ******************************#
#*****************************************************************#
@app.route('/')
def index():
    return '''
        <html>
            <head>
                <title>BLE Sensor Data</title>
            </head>
            <body>
                <h1>BLE Sensor Data</h1>
                <p>Consulta los datos de los sensores en <a href="/sensorData">/sensorData</a></p>
            </body>
        </html>
    '''

@app.route('/sensorData', methods=['GET'])
def get_sensor_data():
    if any(sensor_data.values()):
        cleaned_data = {
            k: (v.replace("\u00b0", "°") if isinstance(v, str) else v)
            for k, v in sensor_data.items()
        }
        return Response(json.dumps(cleaned_data, ensure_ascii=False),
                        content_type="application/json; charset=utf-8", status=200)
    else:
        return Response(json.dumps({"message": "No data received yet"}, ensure_ascii=False),
                        content_type="application/json; charset=utf-8", status=404)

#*****************************************************************#
#***************************** MAIN ******************************#
#*****************************************************************#
if __name__ == '__main__':
    wait_for_ble_connection()  # Esperar hasta que el puerto esté disponible
    ble_thread = threading.Thread(target=read_ble_data, daemon=True)
    ble_thread.start()  # Iniciar el hilo de lectura de datos

    # Hilo para monitorear el ancho de banda
    bandwidth_thread = threading.Thread(target=monitor_sensor_bandwidth, daemon=True)
    bandwidth_thread.start()


    app.run(debug=False, host='0.0.0.0', port=80)
```

Raspberry Cliente (Barco):

- En la raspberry que simula el barco, es decir, el cliente. Existirá el siguiente código que envía los datos de los sensores por BLE:

ble_sendata.py

```
import bluetooth
import gpiozero
import serial
import time
```

```
import datetime

import psutil

# Dirección MAC del servidor BLE (Raspberry Pi A)
server_mac_address = "B8:27:EB:7D:8C:DE"
port = 1

# 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
latencia_start = None

# ***********************************************************************
# ************* CONFIGURACION PARA BLE ENVIAR LOS DATOS ****************
# ***********************************************************************
# 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

# 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

# INICIALIZAR LA CONEXION SERIAL BLUETOOTH
def init_bluetooth_connection():
    global ser
    bluetooth_port = '/dev/rfcomm0'  # Ajusta este puerto según tu configuración
    baud_rate = 9600
    timeout_sec = 2

    try:
        ser = serial.Serial(bluetooth_port, baud_rate, timeout=timeout_sec)
        print("Conexión Bluetooth exitosa")
    except serial.SerialException:
        print("No se pudo conectar al dispositivo Bluetooth. Verifica el puerto y el emparejamiento.")
        return None

# LEER DATOS DEL BLUETOOTH 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
```

```
# Enviar datos por Bluetooth
def send_data_bluetooth(client_sock, temperatura, lineDistance, lineAcceleration, lineGyroscope, lineCompass,
lineaHeading, lineTiltHeading):
    global latencia_start

    try:
        #mensaje = f"Temperatura: {temperatura}°C\n"
        #client_sock.send(mensaje + "\n")

        if lineDistance is not None:
            mensaje = f"Distancia: {lineDistance} cm"
            print(f"Enviando por Bluetooth: {mensaje}")
            client_sock.send(mensaje + "\n")

        elif lineAcceleration is not None:
            mensaje = f"Aceleración: {lineAcceleration}"
            print(f"Enviando por Bluetooth: {mensaje}")
            client_sock.send(mensaje + "\n")

        elif lineGyroscope is not None:
            mensaje = f"Giroscopio: {lineGyroscope}"
            print(f"Enviando por Bluetooth: {mensaje}")
            client_sock.send(mensaje + "\n")

        elif lineCompass is not None:
            mensaje = f"Compás: {lineCompass}"
            print(f"Enviando por Bluetooth: {mensaje}")
            client_sock.send(mensaje + "\n")

        elif lineaHeading is not None:
            mensaje = f"Ángulo (norte y eje X): {lineaHeading}°"
            print(f"Enviando por Bluetooth: {mensaje}")
            client_sock.send(mensaje + "\n")

        elif lineTiltHeading is not None:
            mensaje = f"Ángulo (norte y proyección eje X): {lineTiltHeading}°"
            print(f"Enviando por Bluetooth: {mensaje}")
            latencia_start = time.time()
            client_sock.send(mensaje + "\n")

            if not receive_ack(client_sock):
                print("ACK no recibido para Ángulo (norte y proyección eje X)")

    except Exception as e:
        print(f"Error al enviar datos: {e}")

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

# **********************************************************************
# ************* CONFIGURACION PARA BLE ESCUCHAR LOS DATOS **************
# **********************************************************************
def receive_ack(client_sock):
    global latencia_start
    try:
        # Esperar un mensaje del cliente
        ack_message = client_sock.recv(1024).decode('utf-8').strip()
        if ack_message == "ACK":
            print("ACK recibido")
            end_time = time.time()
            latencia = end_time - latencia_start
            print("-------------------------------------------")
            print(f"Latencia calculada: {latencia:.3f} segundos")
            print("-------------------------------------------")
            return True
        else:
            print(f"Mensaje inesperado recibido: {ack_message}")
            return False
    except Exception as e:
        print(f"Error al recibir ACK: {e}")
```

```
        return False

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

#*******************************************************************#
#****************************** MAIN *******************************#
#*******************************************************************#
def main():
    #init_bluetooth_connection()
    init_serial_connection()

    # Crear socket Bluetooth
    client_sock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)

    try:
        # Conectar al servidor Bluetooth
        client_sock.connect((server_mac_address, port))
        print("Conectado a", server_mac_address)

        while True:

            temperatura = getRaspData()
            line = read_sensor_data()


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

            # Preparar mensaje para enviar
            send_data_bluetooth(client_sock, temperatura, lineDistance, lineAcceleration, lineGyroscope,
lineCompass, lineaHeading, lineTiltHeading)

    except OSError as e:
        print("Error:", e)
    finally:
        ser.close()
        client_sock.close()
        #client.loop_stop()

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