RTOS BASED ESP SENSOR DATA TRANSFER TO MQTT END POINT IN JASON FORMATE

- Sensors used are NEO-6E GPS Module and MPU6050
- The data return by the sensors are accelerometer x,y,z in m/s^2, Gyroscope x, y, z angle, Location, Speed, Temperature
- The Code is RTOS based it gets the sensor data and process are run and fetched concurrently
- The MPU should get data for every 5ms that is it gets data at rate of 200Hz
- The GPS data are fetched at 1HZ
- The Code is based on FreeRTOS
- The data is transmitted over WIFI Network to MQTT Broker

```
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <freertos/queue.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <TinyGPSPlus.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

// Wi-Fi and MQTT Configuration
const char* ssid = "WIFI_SSID"; // Replace with your Wi-Fi SSID
const char* password = "WIFI_PASSWORD"; // Replace with your Wi-Fi password
const char* mqtt_server = "broker.com"; // Public MQTT broker for testing
const char* mqtt_topic = "esp32/sensors_data";
WiFiClient espClient;
PubSubClient client(espClient);

// GPS Configuration
#define RXD2 16
#define TXD2 17
```

```
#define GPS_BAUD 9600
HardwareSerial gpsSerial(2);
TinyGPSPlus gps;

// MPU6050 Configuration
Adafruit_MPU6050 mpu;

// Data Structure for Sensor Data
struct SensorData {
  float accel_x, accel_y, accel_z; // m/s^2
  float gyro_x, gyro_y, gyro_z;   // rad/s
  float temperature;             // °C
  float lat, lng;                // degrees
  float speed;                   // km/h
  char time[20];                 // YYYY-MM-DDTHH:MM:SSZ
};

// FreeRTOS Queue
QueueHandle_t sensorQueue;

// Function Prototypes
void connectWiFi();
void connectMQTT();
void publishSensorData(SensorData data);

// Task Handles
TaskHandle_t mpuTaskHandle = NULL;
TaskHandle_t gpsTaskHandle = NULL;
TaskHandle_t mqttTaskHandle = NULL;
```

```
// MPU6050 Task: Read data at 200 Hz (5ms interval)
void mpuTask(void *pvParameters) {
  TickType_t xLastWakeTime = xTaskGetTickCount();
  const TickType_t xFrequency = pdMS_TO_TICKS(5); // 5ms = 200 Hz

  while (1) {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);

    SensorData data;
    data.accel_x = a.acceleration.x;
    data.accel_y = a.acceleration.y;
    data.accel_z = a.acceleration.z;
    data.gyro_x = g.gyro.x;
    data.gyro_y = g.gyro.y;
    data.gyro_z = g.gyro.z;
    data.temperature = temp.temperature;
    data.lat = 0.0; // Will be updated by GPS task
    data.lng = 0.0;
    data.speed = 0.0;
    strcpy(data.time, "1970-01-01T00:00:00Z");

    // Send to queue (non-blocking)
    xQueueSend(sensorQueue, &data, 0);

    // Maintain 200 Hz
    vTaskDelayUntil(&xLastWakeTime, xFrequency);
  }
```

```
}

// GPS Task: Read and parse GPS data every 1 second
void gpsTask(void *pvParameters) {
  TickType_t xLastWakeTime = xTaskGetTickCount();
  const TickType_t xFrequency = pdMS_TO_TICKS(1000); // 1 second

  while (1) {
    while (gpsSerial.available() > 0) {
      if (gps.encode(gpsSerial.read())) {
        if (gps.location.isValid() && gps.date.isValid() && gps.time.isValid()) {
          SensorData data;
          data.accel_x = 0.0; // Will be updated by MPU task
          data.accel_y = 0.0;
          data.accel_z = 0.0;
          data.gyro_x = 0.0;
          data.gyro_y = 0.0;
          data.gyro_z = 0.0;
          data.temperature = 0.0;
          data.lat = gps.location.lat();
          data.lng = gps.location.lng();
          data.speed = gps.speed.kmph();
          snprintf(data.time, sizeof(data.time), "%04d-%02d-%02dT%02d:%02d:%02dZ",
                   gps.date.year(), gps.date.month(), gps.date.day(),
                   gps.time.hour(), gps.time.minute(), gps.time.second());

          // Send to queue (non-blocking)
          xQueueSend(sensorQueue, &data, 0);
        }
      }
    }

    // Maintain 1 Hz
    vTaskDelayUntil(&xLastWakeTime, xFrequency);
  }
}

// MQTT Task: Aggregate data and publish to MQTT
void mqttTask(void *pvParameters) {
  SensorData mpuData = {0};
  SensorData gpsData = {0};
  bool hasMpuData = false;
  bool hasGpsData = false;

  while (1) {
    // Receive data from queue
    SensorData data;
    while (xQueueReceive(sensorQueue, &data, 0) == pdTRUE) {
      if (data.lat == 0.0 && data.lng == 0.0) {
        // MPU data
        mpuData = data;
        hasMpuData = true;
      } else {
        // GPS data
        gpsData = data;
        hasGpsData = true;
      }
    }
    // Publish if we have both MPU and GPS data
    if (hasMpuData && hasGpsData) {
      publishSensorData(gpsData); // Use GPS data as base, MPU data is already in queue
      hasMpuData = false;
      hasGpsData = false;
    }

    // Check MQTT connection
    if (!client.connected()) {
      connectMQTT();
    }
    client.loop();

    // Small delay to prevent task hogging
    vTaskDelay(pdMS_TO_TICKS(10));
  }
}

// Connect to Wi-Fi
void connectWiFi() {
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    vTaskDelay(pdMS_TO_TICKS(500));
    Serial.print(".");
  }
  Serial.println("\nWi-Fi connected");
}
```

```
// Connect to MQTT
void connectMQTT() {
  Serial.println("Connecting to MQTT...");
  client.setServer(mqtt_server, 1883);
  while (!client.connected()) {
    if (client.connect("ESP32Client")) {
      Serial.println("MQTT connected");
    } else {
      Serial.print("MQTT failed, rc=");
      Serial.print(client.state());
      vTaskDelay(pdMS_TO_TICKS(2000));
    }
  }
}

// Publish Sensor Data as JSON
void publishSensorData(SensorData data) {
  StaticJsonDocument<256> doc;
  JsonObject accel = doc.createNestedObject("accelerometer");
  accel["x"] = data.accel_x;
  accel["y"] = data.accel_y;
  accel["z"] = data.accel_z;
  JsonObject gyro = doc.createNestedObject("gyroscope");
  gyro["x"] = data.gyro_x;
  gyro["y"] = data.gyro_y;
  gyro["z"] = data.gyro_z;
  doc["temperature"] = data.temperature;
  JsonObject loc = doc.createNestedObject("location");
  loc["lat"] = data.lat;
  loc["lng"] = data.lng;
  doc["speed"] = data.speed;
  doc["time"] = data.time;

  char jsonBuffer[256];
  serializeJson(doc, jsonBuffer);
  if (client.publish(mqtt_topic, jsonBuffer)) {
    Serial.println("Published to MQTT:");
    Serial.println(jsonBuffer);
  } else {
    Serial.println("Failed to publish to MQTT");
  }
}

void setup() {
  // Initialize Serial
  Serial.begin(115200);

  // Initialize GPS
  gpsSerial.begin(GPS_BAUD, SERIAL_8N1, RXD2, TXD2);
  Serial.println("GPS Serial started");

  // Initialize MPU6050
  Wire.begin();
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050");
    while (1) vTaskDelay(pdMS_TO_TICKS(10));
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_94_HZ); // Higher bandwidth for 200 Hz
  Serial.println("MPU6050 initialized");

  // Connect to Wi-Fi and MQTT
  connectWiFi();
  connectMQTT();

  // Create FreeRTOS Queue
  sensorQueue = xQueueCreate(10, sizeof(SensorData));
  if (sensorQueue == NULL) {
    Serial.println("Failed to create queue");
    while (1) vTaskDelay(pdMS_TO_TICKS(10));
  }

  // Create FreeRTOS Tasks
  xTaskCreatePinnedToCore(
    mpuTask, "MPUTask", 4096, NULL, 2, &mpuTaskHandle, 1);
  xTaskCreatePinnedToCore(
    gpsTask, "GPSTask", 4096, NULL, 1, &gpsTaskHandle, 1);
  xTaskCreatePinnedToCore(
    mqttTask, "MQTTTask", 4096, NULL, 1, &mqttTaskHandle, 1);
}

void loop() {
  // Empty: FreeRTOS handles tasks
  vTaskDelay(pdMS_TO_TICKS(1000));
}
```