MQTT PUBLISHER USING ESP32 IN JASON FORMATE

- The code collects the sensor data from MPU6050 and GPS NEO-6E module and send to MQTT Broker in a Jason format over Wifi
- Unlike using RTOS for task Scheduling and Queue based communication here the task are carried out in a sequential manner using millis() based based function for task scheduling to meet the time constrain

```
#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
};

// Global Sensor Data
SensorData mpuData = {0};
SensorData gpsData = {0};
bool hasMpuData = false;
bool hasGpsData = false;

// Timing Variables
unsigned long lastMpuRead = 0;
unsigned long lastGpsRead = 0;
const unsigned long mpuInterval = 5; // 5ms = 200 Hz
const unsigned long gpsInterval = 1000; // 1 second

// Function Prototypes
void connectWiFi();
void connectMQTT();
void publishSensorData();
void readMPU();
void readGPS();

// Connect to Wi-Fi
void connectWiFi() {
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(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());
      delay(2000);
    }
  }
}

// Publish Sensor Data as JSON
void publishSensorData() {
  if (hasMpuData && hasGpsData) {
    StaticJsonDocument<256> doc;
    JsonObject accel = doc.createNestedObject("accelerometer");
    accel["x"] = mpuData.accel_x;
    accel["y"] = mpuData.accel_y;
    accel["z"] = mpuData.accel_z;
    JsonObject gyro = doc.createNestedObject("gyroscope");
    gyro["x"] = mpuData.gyro_x;
    gyro["y"] = mpuData.gyro_y;
    gyro["z"] = mpuData.gyro_z;
    doc["temperature"] = mpuData.temperature;
    JsonObject loc = doc.createNestedObject("location");
    loc["lat"] = gpsData.lat;
    loc["lng"] = gpsData.lng;
    doc["speed"] = gpsData.speed;
    doc["time"] = gpsData.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");
    }
    hasMpuData = false; // Reset flags after publishing
    hasGpsData = false;
  }
}

// Read MPU6050 Data
void readMPU() {
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  mpuData.accel_x = a.acceleration.x;
  mpuData.accel_y = a.acceleration.y;
  mpuData.accel_z = a.acceleration.z;
  mpuData.gyro_x = g.gyro.x;
  mpuData.gyro_y = g.gyro.y;
  mpuData.gyro_z = g.gyro.z;
  mpuData.temperature = temp.temperature;
  mpuData.lat = 0.0; // Not used
  mpuData.lng = 0.0;
  mpuData.speed = 0.0;
  strcpy(mpuData.time, "1970-01-01T00:00:00Z");
  hasMpuData = true;
}

// Read GPS Data
void readGPS() {
  while (gpsSerial.available() > 0) {
    if (gps.encode(gpsSerial.read())) {
      if (gps.location.isValid() && gps.date.isValid() && gps.time.isValid()) {
        gpsData.accel_x = 0.0; // Not used
        gpsData.accel_y = 0.0;
        gpsData.accel_z = 0.0;
        gpsData.gyro_x = 0.0;
        gpsData.gyro_y = 0.0;
        gpsData.gyro_z = 0.0;
        gpsData.temperature = 0.0;
        gpsData.lat = gps.location.lat();
        gpsData.lng = gps.location.lng();
        gpsData.speed = gps.speed.kmph();
        snprintf(gpsData.time, sizeof(gpsData.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());
        hasGpsData = true;
      }
    }
  }
}

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) delay(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();
}

void loop() {
  // Read MPU6050 at 200 Hz (every 5ms)
  unsigned long currentMillis = millis();
  if (currentMillis - lastMpuRead >= mpuInterval) {
    readMPU();
    lastMpuRead = currentMillis;
  }

  // Read GPS every 1 second
  if (currentMillis - lastGpsRead >= gpsInterval) {
    readGPS();
    lastGpsRead = currentMillis;
  }

  // Publish data if both MPU and GPS data are available
  publishSensorData();

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