RAJALAKSHMI INSTITUTE OF TECHNOLOGY

BELIEVE IN THE POSSIBILITIES

AN AUTONOMOUS INSTITUTION

DEPARTMENT OF B.E ELECTRONICS AND COMMUNICATION ENGINEERING

Embedded System & IOT Design

TITLE:- Drone Telemetry Simulation System Using ESP32 and Web Dashboard

Drone Telemetry Simulation System Using ESP32 and Web Dashboard

Aim

To develop a real-time drone telemetry simulation system using ESP32, capable of transmitting IMU, environmental, and simulated battery data via WebSocket to a web-based dashboard for monitoring and visualization.

Objectives

- To simulate a drone's telemetry system using ESP32 without actual flight hardware.
- To collect real-time orientation data (pitch, roll, yaw) from the MPU6050 sensor.
- To monitor temperature and humidity using the DHT11 sensor.
- To simulate battery voltage degradation using a 10kΩ potentiometer.
- To transmit the combined sensor data every second to a Node.js server via WebSocket.
- To visualize telemetry data on a React-based frontend using graphical widgets like progress bars and gauge meters.
- To structure firmware with FreeRTOS for real-time, multitask sensor reading and communication.

Components Required

Component	Quantity	Description	Approx. Cost (INR)
ESP32 Dev Board	1	Wi-Fi enabled microcontroller for telemetry logic	₹300
MPU6050 IMU Sensor	1	3-axis Accelerometer + Gyroscope	₹120
DHT11 Sensor	1	Temperature and Humidity sensor	₹50
Potentiometer (10kΩ)	1	For simulating battery voltage	₹15
Breadboard	1	For prototyping connections	₹50
Jumper Wires	Several	Male-to-male wires for sensor interfacing	₹20
Laptop (existing)	1	Hosts Node.js server and React frontend	—
Software Tools	—	Arduino IDE, VS Code, Node.js, React	Free

Total Cost Estimate: ₹555 (approx.)

Feasibility analysis

1. Technical Feasibility – Feasible

- ESP32 supports Wi-Fi, WebSocket, FreeRTOS, and sensor interfacing (MPU6050 via I2C, DHT11 digital, potentiometer via ADC).
- Low data rate (1s updates) ensures stable communication.
- Node.js (ws) and React (gauge/progress charts) are well-supported.

2. Operational Feasibility – Feasible

- No real drone needed; desk-based development.
- Simple maintenance with modular RTOS tasks and free software tools.

3. Economic Feasibility – Low cost (~₹500–₹1000)

- All hardware inexpensive; all software free/open-source.

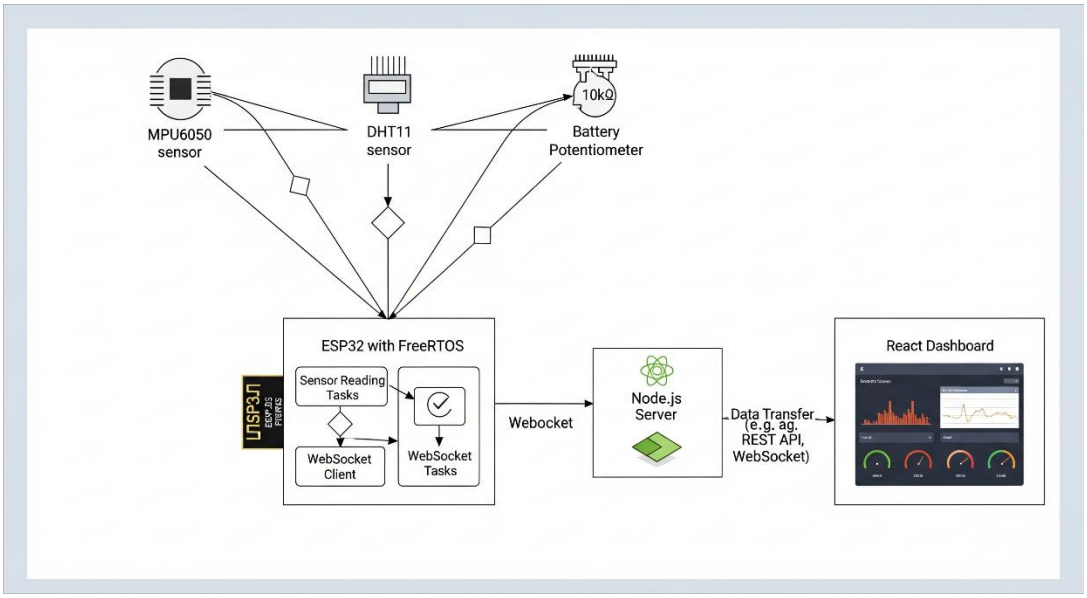
4. Schedule Feasibility – Achievable in 2–3 weeks

- Week 1: Sensor + FreeRTOS setup
- Week 2: WebSocket + backend integration
- Week 3: React dashboard + testing

5. Risks & Mitigation

- Sensor delays → switch to DHT22
- Wi-Fi drops → auto-reconnect
- IMU noise → filtering

System Mode



System Description

The project is a simulation of a drone telemetry system using embedded and web technologies. At its core is an ESP32 microcontroller running FreeRTOS, which periodically collects telemetry data from three sources:

1. MPU6050 – Provides orientation data (pitch, roll, yaw).
2. DHT11 – Measures ambient temperature and humidity.
3. 10kΩ Potentiometer – Simulates battery voltage degradation, mimicking a drone's battery level drop.

The sensor data is packaged into a structured payload and sent using WebSocket communication over Wi-Fi to a Node.js backend server hosted on a laptop.

The frontend, developed using React, visualizes this data in real time. It displays orientation through animated gauge meters and shows temperature, humidity, and battery level via progress bars and numeric indicators.

The system architecture ensures modularity, real-time updates, and a clear separation between embedded firmware, server logic, and frontend UI.

This project showcases the use of RTOS-based embedded programming, WebSocket networking, and web-based IoT dashboards, all while simulating a real-world drone telemetry environment.

Algorithm

On ESP32 (Firmware Side):

1. Initialize all peripherals:
    - Setup MPU6050 over I2C
    - Setup DHT11 on GPIO pin
    - Configure ADC to read potentiometer value
2. Create FreeRTOS tasks:
    - Task1: Read IMU data (pitch, roll, yaw)
    - Task2: Read DHT11 temperature and humidity
    - Task3: Read battery simulation from potentiometer
    - Task4: Format all data into JSON and send over WebSocket to server
3. Every 1 second:
    - Synchronize sensor readings
    - Update the payload
    - Transmit the data to the Node.js server

On Node.js Server:

1. Start WebSocket server
2. Listen for incoming connections
3. Receive telemetry packets from ESP32
4. Forward data to connected React dashboard clients

On React Frontend:

1. Connect to WebSocket server
2. Parse incoming JSON data
3. Update real-time dashboard UI:
    - Gauge meters for pitch, roll, yaw
    - Progress bars for temperature, humidity, battery

PROGRAM

```
#include <WiFi.h>
#include <WebSocketsClient.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// ---------------- OLED Setup ----------------
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
#define SDA_PIN 21
#define SCL_PIN 22

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
volatile bool wsConnected = false;   // shared flag for connection status

// ---------------- Telemetry Setup ----------------
#define SEND_BINARY true
#define PRINT_EVERY_N 10
#define SENSOR_DELAY_MS 50
#define SEND_INTERVAL_MS 50

const char* ssid = " ";
const char* password = " ";

WebSocketsClient webSocket;
const char* websocket_host = "192.168.1.9";
const uint16_t websocket_port = 3000;
const char* websocket_path = "/";

Adafruit_MPU6050 mpu;
#define DHTPIN 4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
#define BATTERY_PIN 34

TaskHandle_t sensorTaskHandle = NULL;
TaskHandle_t sendTaskHandle = NULL;
```

```
TaskHandle_t oledTaskHandle = NULL;
SemaphoreHandle_t dataMutex = NULL;

struct TelemetryData {
  float pitch;
  float roll;
  float yaw;
  float temperature;
  float humidity;
  float batteryVoltage;
} telemetry;

volatile uint32_t sendSeq = 0;
volatile uint32_t serialPrintCounter = 0;

// ---------------- Helpers ----------------
static inline void packUint32LE(uint8_t *buf, size_t offset, uint32_t v) {
  buf[offset + 0] = (uint8_t)(v & 0xFF);
  buf[offset + 1] = (uint8_t)((v >> 8) & 0xFF);
  buf[offset + 2] = (uint8_t)((v >> 16) & 0xFF);
  buf[offset + 3] = (uint8_t)((v >> 24) & 0xFF);
}
static inline void packFloatLE(uint8_t *buf, size_t offset, float f) {
  uint8_t tmp[4];
  memcpy(tmp, &f, 4);
  buf[offset + 0] = tmp[0];
  buf[offset + 1] = tmp[1];
  buf[offset + 2] = tmp[2];
  buf[offset + 3] = tmp[3];
}

// ---------------- WebSocket Events ----------------
void webSocketEvent(WStype_t type, uint8_t * payload, size_t length) {
  switch (type) {
    case WStype_CONNECTED:
      Serial.println("[WS] Connected");
      wsConnected = true;
      break;
    case WStype_DISCONNECTED:
      Serial.println("[WS] Disconnected");
      wsConnected = false;
      break;
    default:
      break;
  }
}

// ---------------- Sensor Task ----------------
void sensorTask(void *parameter) {
  sensors_event_t a, g, tempSensor;
  (void) parameter;
  for (;;) {
    mpu.getEvent(&a, &g, &tempSensor);
    float t = dht.readTemperature();
    float h = dht.readHumidity();
    if (isnan(t)) t = -127.0f;
    if (isnan(h)) h = -127.0f;
    int raw = analogRead(BATTERY_PIN);
    float voltage = raw * (3.3f / 4095.0f) * 2.0f;
    float pitch = atan2(a.acceleration.y, a.acceleration.z) * 180.0f / PI;
```

```
    float roll  = atan2(-a.acceleration.x, sqrt(a.acceleration.y * a.acceleration.y + a.acceleration.z *
a.acceleration.z)) * 180.0f / PI;
    float yaw   = g.gyro.z * 57.29577951308232f;

    if (xSemaphoreTake(dataMutex, pdMS_TO_TICKS(50)) == pdTRUE) {
      telemetry.pitch = pitch;
      telemetry.roll  = roll;
      telemetry.yaw   = yaw;
      telemetry.temperature = t;
      telemetry.humidity = h;
      telemetry.batteryVoltage = voltage;
      xSemaphoreGive(dataMutex);
    }
    vTaskDelay(pdMS_TO_TICKS(SENSOR_DELAY_MS));
  }
}

// ---------------- Send Task ----------------
void sendTask(void *parameter) {
  (void) parameter;
  uint8_t binBuf[32];
  StaticJsonDocument<256> jsonDoc;
  char jsonBuf[256];

  for (;;) {
    TelemetryData local;
    if (xSemaphoreTake(dataMutex, pdMS_TO_TICKS(50)) == pdTRUE) {
      local = telemetry;
      xSemaphoreGive(dataMutex);
    } else {
      vTaskDelay(pdMS_TO_TICKS(SEND_INTERVAL_MS));
      continue;
    }

    uint32_t seq = ++sendSeq;
    uint32_t ts_ms = (uint32_t) millis();

    if (SEND_BINARY) {
      packUint32LE(binBuf, 0, seq);
      packUint32LE(binBuf, 4, ts_ms);
      packFloatLE(binBuf, 8,  local.pitch);
      packFloatLE(binBuf, 12, local.roll);
      packFloatLE(binBuf, 16, local.yaw);
      packFloatLE(binBuf, 20, local.temperature);
      packFloatLE(binBuf, 24, local.humidity);
      packFloatLE(binBuf, 28, local.batteryVoltage);
      webSocket.sendBIN(binBuf, sizeof(binBuf));
    } else {
      jsonDoc.clear();
      jsonDoc["seq"] = seq;
      jsonDoc["ts_ms"] = ts_ms;
      jsonDoc["pitch"] = local.pitch;
      jsonDoc["roll"] = local.roll;
      jsonDoc["yaw"] = local.yaw;
      jsonDoc["temperature"] = local.temperature;
      jsonDoc["humidity"] = local.humidity;
      jsonDoc["battery"] = local.batteryVoltage;
      size_t len = serializeJson(jsonDoc, jsonBuf, sizeof(jsonBuf));
      webSocket.sendTXT((const uint8_t*)jsonBuf, len);
    }
```

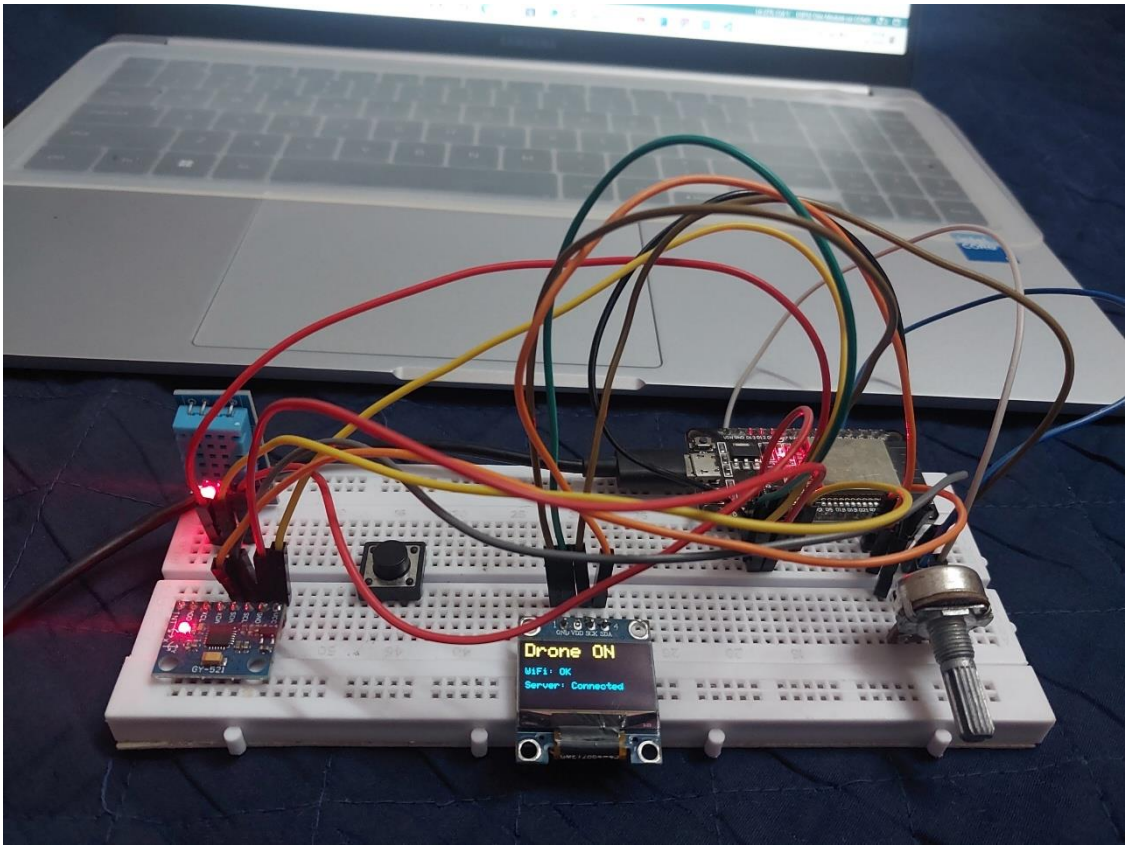
```
    serialPrintCounter++;
    if (serialPrintCounter >= PRINT_EVERY_N) {
      serialPrintCounter = 0;
      Serial.printf("SENT #%u  t=%lu ms  P/R/Y=%.1f/%.1f/%.1f  T=%.1fC H=%.1f%% V=%.2fV\n",
                    seq, (unsigned long)ts_ms,
                    local.pitch, local.roll, local.yaw,
                    local.temperature, local.humidity, local.batteryVoltage);
    }

    vTaskDelay(pdMS_TO_TICKS(SEND_INTERVAL_MS));
  }
}

// ---------------- OLED Task ----------------
void oledTask(void *parameter) {
  (void) parameter;
  for (;;) {
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(SSD1306_WHITE);

    // Line 1: Drone ON
    display.setCursor(0, 0);
    display.println("Drone ON");

    // Line 2: WiFi
    display.setTextSize(1);
    display.setCursor(0, 25);
    display.print("WiFi: ");
    display.println(WiFi.isConnected() ? "OK" : "No");

    // Line 3: WS Connection
    display.setCursor(0, 40);
    display.print("Server: ");
    display.println(wsConnected ? "Connected" : "No link");

    display.display();
    vTaskDelay(pdMS_TO_TICKS(500));  // update every 0.5s
  }
}

// ---------------- Setup ----------------
void setup() {
  Serial.begin(115200);
  delay(100);

  // Start I2C and OLED
  Wire.begin(SDA_PIN, SCL_PIN);
  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("SSD1306 allocation failed");
    for (;;);
  }
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 10);
  display.println("Booting...");
  display.display();
```

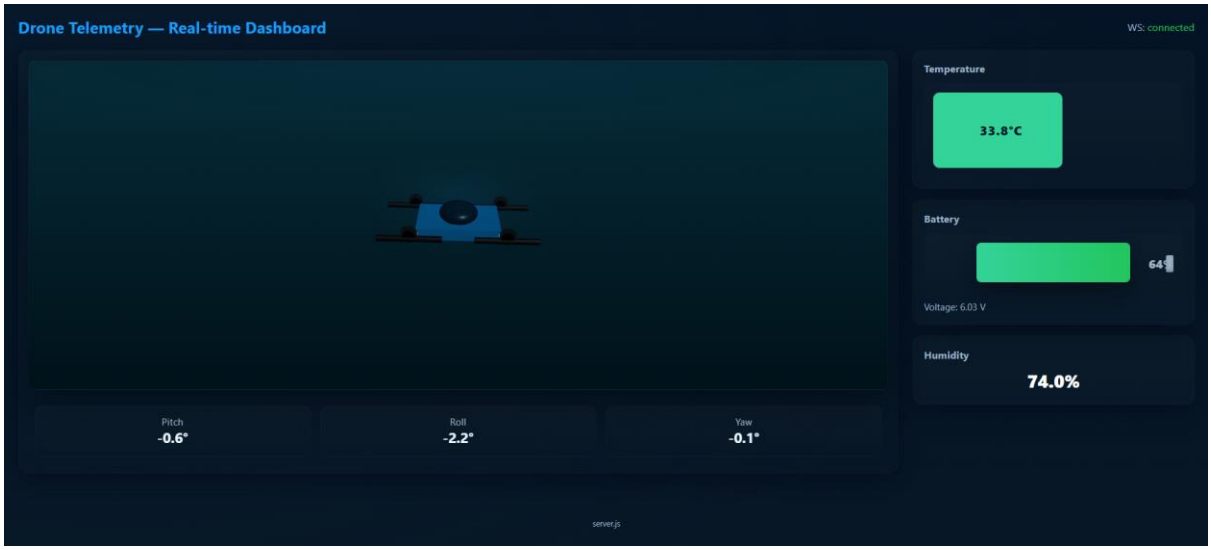
```
  // WiFi
  WiFi.begin(ssid, password);
  Serial.print("Connecting WiFi");
  uint32_t wifiStart = millis();
  while (WiFi.status() != WL_CONNECTED) {
    delay(300);
    Serial.print(".");
    if (millis() - wifiStart > 10000) {
      Serial.println();
      wifiStart = millis();
    }
  }
  Serial.println();
  Serial.print("Connected. IP: ");
  Serial.println(WiFi.localIP());
  WiFi.setSleep(false);

  // MPU6050
  if (!mpu.begin()) {
    Serial.println("MPU6050 not found! Halting.");
    while (1) { delay(1000); }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  // DHT + battery
  dht.begin();
  analogReadResolution(12);

  // WebSocket
  webSocket.begin(websocket_host, websocket_port, websocket_path);
  webSocket.onEvent(webSocketEvent);
  webSocket.setReconnectInterval(3000);

  // Mutex + RTOS tasks
  dataMutex = xSemaphoreCreateMutex();
  if (dataMutex == NULL) {
    Serial.println("Failed to create mutex — halting.");
    while (1) { delay(1000); }
  }
  xTaskCreatePinnedToCore(sensorTask, "SensorTask", 4096, NULL, 2, &sensorTaskHandle, 1);
  xTaskCreatePinnedToCore(sendTask,   "SendTask",   4096, NULL, 2, &sendTaskHandle,   1);
  xTaskCreatePinnedToCore(oledTask,   "OledTask",   4096, NULL, 1, &oledTaskHandle,   1);

  Serial.println("Setup complete.");
}

// ---------------- Loop ----------------
void loop() {
  webSocket.loop();
  delay(2);
}
```

Result

The proposed drone telemetry simulation system was successfully developed and tested using ESP32 and various sensors. The ESP32 collected real-time data from the MPU6050 sensor for orientation (pitch, roll, yaw), from the DHT11 sensor for temperature and humidity, and from a 10kΩ potentiometer to simulate battery voltage levels. The firmware was implemented using FreeRTOS to manage multiple tasks efficiently, with sensor readings and data transmission occurring at a frequency of one second.

The collected data was transmitted via WebSocket protocol over Wi-Fi to a Node.js server hosted on a local laptop. Although the frontend is still under development, the backend successfully received and processed the data. The system was able to maintain consistent and stable communication between the ESP32 and the server, with no noticeable packet loss during testing. The React-based web application is designed to visualize the telemetry data using gauge meters and progress bars, and it has been partially tested using sample data. Overall, the embedded portion of the system is functioning as intended, and the communication layer between the device and the server is stable and reliable.





Discussion

This project successfully implemented a simulated drone telemetry system using a low-cost embedded platform. It demonstrated reliable acquisition of sensor data and consistent real-time communication with a web-based dashboard through WebSocket protocol.

The system was designed to monitor multiple parameters including orientation, temperature, humidity, and battery status. Sensor data was collected at regular intervals and transmitted efficiently without noticeable delays. The firmware was structured to handle concurrent tasks, ensuring smooth performance.

During testing, the communication between the embedded device and the backend server remained stable, validating the choice of protocol and system architecture. The telemetry dashboard received and displayed real-time data, enabling a clear visualization of environmental and motion-related parameters.

Although some minor sensor fluctuations were observed, the overall performance was satisfactory. Potential future enhancements could include refined sensor calibration and advanced filtering techniques to further improve data accuracy.

In conclusion, the project met its core objectives by integrating embedded sensing, real-time communication, and web-based visualization into a cohesive system. It provides a solid foundation for scaling into more advanced telemetry applications in the future.