****

i4 Marine Technologies Pvt. Ltd.

**GLOF BOUY**



**DOCUMENT No. i4MT/GOLFBOUY/UM**

**Prepared and published by**

**i4 Marine Technologies Pvt. Ltd. Electronic Estate, Pune Satara Road**

**Pune-411 009**

**[ MAY 2025 ]**

**Project Title: Golf Buoy – Water Level Monitoring System for Glacial Lakes**

**Objective:**

The primary objective of this project is to monitor water level rise in glacier-fed lakes using a compact, floating buoy system. The buoy collects water depth and altitude data, determines GPS position and orientation, and sends it wirelessly via LoRa to a remote station for monitoring and analysis.

**Hardware Components**

| **Component** | **Purpose** |
| --- | --- |
| ESP32 (x2) | One for sensor data acquisition and transmission, one for data reception |
| Eagle i Flight Controller | Provides GPS and IMU heading data |
| GPS Module | Connected to flight controller for location data |
| Altimeter Sensor | Measures altitude of the buoy (for water level changes) |
| Echosounder | Measures water depth below the buoy |
| LoRa Module (SX1278 or similar) x2 | Wireless data transmission |
| Power Module | Monitors battery voltage/current |
| Buck Converter (24V to 5V) | Steps down voltage to power flight controller |
| 11.1V LiPo Battery | Primary power source |
| Switch | Manual ON/OFF control for the system |

**system Overview**

1. **Sensing Unit (on buoy)**:
   * ESP32 collects:
     + Altimeter & echosounder data via I2C/analog
     + IMU & GPS data from Eagle i FC via UART
     + Voltage data from the power module
   * Sends all data via LoRa
2. **Receiver Unit (on ground)**:
   * ESP32 + LoRa receives data
   * Displays real-time readings via Serial Monitor (Arduino IDE)

**Wiring Diagram & Explanation**

**Power Supply**

* **11.1V LiPo Battery** → [Switch] → [Buck Converter 24V to 5V]
* **Buck Converter Output (5V)** → Flight Controller (via power input port)
* **Flight Controller** → 5V regulated output to power **ESP32 TX unit**

**Power Distribution**

**[11.1V LiPo Battery]**

**↓**

**[Switch]**

**↓**

**[24V to 5V Buck Converter]**

**↓**

**[5V Output from Buck]**

**├──> Eagle i Flight Controller (Power Port)**

**└──> ESP32 (VIN pin)**

**Sensor Connections to ESP32 (Transmitting Unit on Buoy)**

**1. Echosounder (UART)**

| ESP32 Pin | Echosounder Pin | Note |
| --- | --- | --- |
| RX (GPIO16 or any) | TX | From sonar to ESP32 |
| TX (GPIO17 or any) | RX | From ESP32 to sonar |
| GND | GND | Shared ground |
| 5V or 3.3V | VCC | Depends on sonar model |

**2. Altimeter (I2C)**

| ESP32 Pin | Altimeter Pin |
| --- | --- |
| SDA (GPIO21) | SDA |
| SCL (GPIO22) | SCL |
| GND | GND |
| 3.3V or 5V | VCC (depends on sensor) |

3. **Power Module (I2C)**

| ESP32 Pin | Power Module Pin |
| --- | --- |
| SDA (shared with altimeter) | SDA |
| SCL (shared with altimeter) | SCL |
| GND | GND |
| VCC | 3.3V or 5V |

**UART Data from Flight Controller to ESP32**

* Use a UART port on the Eagle i flight controller to send GPS and heading data to the ESP32.

| **Eagle i FC UART Port** | **ESP32 Pin** |
| --- | --- |
| TX | RX |
| RX | TX |
| GND | GND |
| **(5V is already shared from buck)** |  |

**Set the correct baud rate in the ESP32 code to match the FC telemetry output (usually 115200).**

**LoRa Module Wiring (SPI)**

| **ESP32 Pin** | **LoRa Module** |
| --- | --- |
| GPIO18 | SCK |
| GPIO19 | MISO |
| GPIO23 | MOSI |
| GPIO5 | NSS ( |
| GPIO14 | RST |
| GPIO26 | DIO0 |
| GND | GND |
| 3.3V | VCC (use 3.3V only; LoRa is not 5V tolerant) |

**ESP32 (RX - Ground Receiver Side)**

| **Connection** | **Sensor/Module** | **Interface** | **Notes** |
| --- | --- | --- | --- |
|  | LoRa Module | SPI | Receives data from buoy |
| USB | Arduino IDE Serial Monitor | USB | Displays sensor data |

**Data Format (Example)**

"altitude": 256.8,

"depth": 3.4,

"voltage": 11.6,

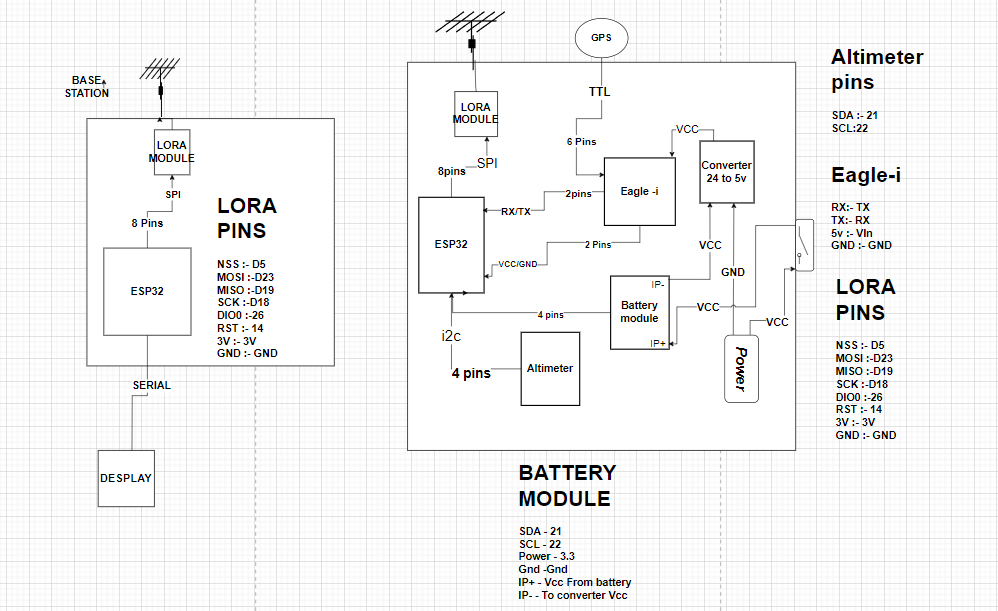
"heading": 74.5,

"gps": { "lat": 31.01234, "lon": 77.06543 }

**Key Notes**

* **ESP32 must have proper UART baud rate configured** to read data from Eagle i flight controller.
* **Altimeter and echosounder** should be tested and calibrated for accuracy before deployment.
* **Use appropriate waterproof casing** for buoy electronics.

**Connection Diagram**

****

**Code used to read data from sensors and send it through LORA**

**Transmitter code**

#include <HardwareSerial.h>

#include <SPI.h>

#include <LoRa.h>

#include <Wire.h>

#include <Adafruit\_BMP3XX.h>  // For BMP390

// MSP protocol commands

#define MSP\_GPS 106

#define MSP\_ATTITUDE 108

// Flight controller serial (UART1)

HardwareSerial FCSerial(1); // RX=GPIO16, TX=GPIO17

// LoRa pins and config

#define LORA\_CS   5

#define LORA\_RST  14

#define LORA\_IRQ  26

#define LORA\_SCK  18

#define LORA\_MISO 19

#define LORA\_MOSI 23

// I2C pins (optional if default SDA=21, SCL=22 are used)

#define I2C\_SDA 21

#define I2C\_SCL 22

// LoRa addresses

byte localAddress = 0xB0;

byte receiverAddress = 0xB1;

// Altimeter object

Adafruit\_BMP3XX bmp;

void setup() {

  Serial.begin(115200);

  FCSerial.begin(115200, SERIAL\_8N1, 16, 17);

  // Initialize I2C

  Wire.begin(I2C\_SDA, I2C\_SCL);

  // Initialize BMP390

  if (!bmp.begin\_I2C()) {

    Serial.println("BMP390 not detected!");

    while (1);

  }

  bmp.setTemperatureOversampling(BMP3\_OVERSAMPLING\_8X);

  bmp.setPressureOversampling(BMP3\_OVERSAMPLING\_4X);

  bmp.setIIRFilterCoeff(BMP3\_IIR\_FILTER\_COEFF\_3);

  bmp.setOutputDataRate(BMP3\_ODR\_50\_HZ);

  Serial.println("BMP390 initialized.");

  // Setup SPI and LoRa

  SPI.begin(LORA\_SCK, LORA\_MISO, LORA\_MOSI, LORA\_CS);

  LoRa.setPins(LORA\_CS, LORA\_RST, LORA\_IRQ);

  if (!LoRa.begin(433E6)) {

    Serial.println("LoRa init failed!");

    while (1);

  }

  Serial.println("LoRa initialized.");

}

void sendMSPRequest(uint8\_t cmd) {

  uint8\_t checksum = 0;

  FCSerial.write('$');

  FCSerial.write('M');

  FCSerial.write('<');

  FCSerial.write((uint8\_t)0);

  FCSerial.write(cmd);

  checksum ^= cmd;

  FCSerial.write(checksum);

}

bool readMSPResponse(uint8\_t\* buffer, uint8\_t expectedLength) {

  uint32\_t start = millis();

  int index = 0;

  while (millis() - start < 500) {

    if (FCSerial.available()) {

      uint8\_t b = FCSerial.read();

      if (index == 0 && b != '$') continue;

      buffer[index++] = b;

      if (index >= expectedLength) return true;

    }

  }

  return false;

}

void sendLoRaMessage(byte destAddress, const char\* message) {

  LoRa.beginPacket();

  LoRa.write(destAddress);   // Receiver address

  LoRa.write(localAddress);  // Sender address

  LoRa.print(message);       // Payload

  LoRa.endPacket();

}

void loop() {

  char loraMessage[100];

  // ---- Request GPS ----

  sendMSPRequest(MSP\_GPS);

  delay(10);

  uint8\_t gpsBuffer[23];

  if (readMSPResponse(gpsBuffer, sizeof(gpsBuffer))) {

    if (gpsBuffer[0] == '$' && gpsBuffer[1] == 'M' && gpsBuffer[2] == '>') {

      uint8\_t numSats = gpsBuffer[6];

      int32\_t lat = gpsBuffer[7] | (gpsBuffer[8] << 8) | (gpsBuffer[9] << 16) | (gpsBuffer[10] << 24);

      int32\_t lon = gpsBuffer[11] | (gpsBuffer[12] << 8) | (gpsBuffer[13] << 16) | (gpsBuffer[14] << 24);

      Serial.println("---- GPS ----");

      Serial.print("Satellites: "); Serial.println(numSats);

      Serial.print("Latitude: "); Serial.println(lat / 10000000.0, 7);

      Serial.print("Longitude: "); Serial.println(lon / 10000000.0, 7);

      snprintf(loraMessage, sizeof(loraMessage), "GPS:%d,%.7f,%.7f", numSats, lat / 10000000.0, lon / 10000000.0);

      sendLoRaMessage(receiverAddress, loraMessage);

    }

  } else {

    Serial.println("No GPS response.");

  }

  delay(50);

  // ---- Request IMU ----

  sendMSPRequest(MSP\_ATTITUDE);

  delay(10);

  uint8\_t imuBuffer[11];

  if (readMSPResponse(imuBuffer, sizeof(imuBuffer))) {

    if (imuBuffer[0] == '$' && imuBuffer[1] == 'M' && imuBuffer[2] == '>') {

      int16\_t roll  = imuBuffer[5] | (imuBuffer[6] << 8);

      int16\_t pitch = imuBuffer[7] | (imuBuffer[8] << 8);

      int16\_t yaw   = imuBuffer[9] | (imuBuffer[10] << 8);

      Serial.println("---- IMU ----");

      Serial.print("Roll: "); Serial.print(roll / 10.0); Serial.println("°");

      Serial.print("Pitch: "); Serial.print(pitch / 10.0); Serial.println("°");

      Serial.print("Yaw: "); Serial.print(yaw); Serial.println("°");

      snprintf(loraMessage, sizeof(loraMessage), "IMU:%.1f,%.1f,%d", roll / 10.0, pitch / 10.0, yaw);

      sendLoRaMessage(receiverAddress, loraMessage);

    }

  } else {

    Serial.println("No IMU response.");

  }

  delay(50);

  // ---- Read BMP390 Altimeter ----

  if (bmp.performReading()) {

    float temperature = bmp.temperature; // °C

    float pressure = bmp.pressure / 100.0; // hPa

    float altitude = bmp.readAltitude(1013.25); // m

    Serial.println("---- Altimeter (BMP390) ----");

    Serial.print("Temperature: "); Serial.print(temperature); Serial.println(" °C");

    Serial.print("Pressure: "); Serial.print(pressure); Serial.println(" hPa");

    Serial.print("Altitude: "); Serial.print(altitude); Serial.println(" m");

    snprintf(loraMessage, sizeof(loraMessage), "ALT:%.2f,%.2f,%.2f", temperature, pressure, altitude);

    sendLoRaMessage(receiverAddress, loraMessage);

  } else {

    Serial.println("BMP390 reading failed.");

  }

  delay(1000); // Loop delay

}

**Receiver code**

#include <SPI.h>

#include <LoRa.h>

// LoRa pins

#define LORA\_CS 5

#define LORA\_RST 14

#define LORA\_IRQ 26

#define LORA\_SCK 18

#define LORA\_MISO 19

#define LORA\_MOSI 23

// Receiver address

byte localAddress = 0xB1;

void setup() {

Serial.begin(115200);

while (!Serial);

SPI.begin(LORA\_SCK, LORA\_MISO, LORA\_MOSI, LORA\_CS);

LoRa.setPins(LORA\_CS, LORA\_RST, LORA\_IRQ);

if (!LoRa.begin(433E6)) {

Serial.println("LoRa init failed!");

while (1);

}

Serial.print("LoRa Receiver Address: 0x");

Serial.println(localAddress, HEX);

Serial.println();

}

void loop() {

int packetSize = LoRa.parsePacket();

if (packetSize) {

byte receiver = LoRa.read();

byte sender = LoRa.read();

String message = "";

while (LoRa.available()) {

message += (char)LoRa.read();

}

if (receiver == localAddress) {

if (message.startsWith("GPS:")) {

int sats;

float lat, lon;

sscanf(message.c\_str(), "GPS:%d,%f,%f", &sats, &lat, &lon);

Serial.println("---- GPS ----");

Serial.print("Satellites: "); Serial.println(sats);

Serial.print("Latitude: "); Serial.println(lat, 7);

Serial.print("Longitude: "); Serial.println(lon, 7);

}

else if (message.startsWith("IMU:")) {

float roll, pitch;

int yaw;

sscanf(message.c\_str(), "IMU:%f,%f,%d", &roll, &pitch, &yaw);

Serial.println("---- IMU ----");

Serial.print("Roll: "); Serial.print(roll); Serial.println("°");

Serial.print("Pitch: "); Serial.print(pitch); Serial.println("°");

Serial.print("Yaw: "); Serial.print(yaw); Serial.println("°");

}

else if (message.startsWith("ALT:")) {

float temp, pressure, alt;

sscanf(message.c\_str(), "ALT:%f,%f,%f", &temp, &pressure, &alt);

Serial.println("---- Altimeter (BMP390) ----");

Serial.print("Temperature: "); Serial.print(temp); Serial.println(" °C");

Serial.print("Pressure: "); Serial.print(pressure); Serial.println(" hPa");

Serial.print("Altitude: "); Serial.print(alt); Serial.println(" m");

}

Serial.println(); // Blank line between messages

}

}

}