FLIGHT TRACKER

Complete System Documentation

Arduino Nano 33 BLE · iOS SwiftUI App · RFM95 LoRa · GT-U7 GPS

Hardware wiring · BLE protocol · iOS setup · Troubleshooting
Version 7 · base_station_v7.ino

1 System Overview

The Flight Tracker system comprises three hardware units and one iOS app that work together to display the real-time position of an airborne flight logger, guide the operator back to the landing site using step estimates and compass directions, and monitor LoRa link quality — all without a cellular or Wi-Fi network.

1.1 Architecture

```
[Flight Logger]──LoRa 915 MHz──>[Arduino Nano 33 BLE]──BLE──>[iPhone]
                                  + GT-U7 GPS                 Flight Tracker App
                                  (Base Station)
```

1.2 Component Roles

Component	Role
Flight Logger (airborne)	Transmits 8-field CSV telemetry via LoRa at configured intervals
Arduino Nano 33 BLE (Base Station)	Receives LoRa packets, reads own GPS, computes navigation, transmits BLE
GT-U7 GPS Module	Provides base-station GPS coordinates
iPhone + Flight Tracker App	Displays live map, distance, bearing, step count, and direction

1.3 Data Flow

1. Flight logger transmits 8-field CSV over LoRa: LAT,LON,BARO_AGL,PITCH,ROLL,YAW,SPEED_KMH,SATELLITES
2. Base station receives the LoRa packet and reads its own GPS position.
3. Arduino computes: ground distance (Haversine), bearing (base → logger), walking steps, cardinal direction.
4. All values are packed into a single UTF-8 CSV BLE characteristic and notified every second.
5. iPhone app subscribes to the characteristic and parses the CSV — no reassembly needed.
6. App also uses CoreLocation for the iPhone's own position and independently calculates iPhone → logger navigation.

1.4 BLE Architecture — Why Custom Characteristic?

Previous versions used the Nordic UART Service (NUS), which requires a serial-terminal app to reassemble fragmented 20-byte chunks and does not work with standard iOS Core Bluetooth. Version 7 replaces this with a **single custom BLE characteristic** that delivers all data atomically.

Property	Value
Service UUID	A1B2C3D4-0000-1000-8000-00805F9B3401
Nav Characteristic UUID	A1B2C3D4-0000-1000-8000-00805F9B3402
Properties	Read + Notify
Max value size	128 bytes (UTF-8 CSV)
Update rate	1 Hz (when LoRa packet received and phone connected)

1.5 CSV Characteristic Format

Eleven comma-separated fields are transmitted in a single notification packet:

```
baseLat,baseLon,logLat,logLon,distMetres,bearingDeg,steps,cardinal,AGL,logHdg,RSSI
```

Example value:

```
37.774900,-122.419400,37.775900,-122.418800,134.5,47.2,176,NE,18.3,274.0,-95
```

Field	Index	Description	Unit
baseLat	0	Base station latitude	Degrees (6 d.p.)
baseLon	1	Base station longitude	Degrees (6 d.p.)
logLat	2	Logger latitude	Degrees (6 d.p.)
logLon	3	Logger longitude	Degrees (6 d.p.)
distMetres	4	Ground distance base→logger	Metres
bearingDeg	5	Bearing base→logger (0=N, CW)	Degrees
steps	6	Estimated walking steps	Count
cardinal	7	8-point cardinal direction	String: N/NE/E/SE/S/SW/W/NW
AGL	8	Barometric altitude above ground	Metres
logHdg	9	Logger magnetic heading	Degrees
RSSI	10	LoRa packet signal strength	dBm (negative integer)

2 Hardware Wiring & Configuration

2.1 Parts List

Part	Notes
Arduino Nano 33 BLE	Either variant works. NOT the Uno — must be the Nano 33 BLE for 3.3 V logic and built-in BLE
RFM95W LoRa Module	915 MHz for USA. Use 868 MHz for EU, 433 MHz for some Asian regions
GT-U7 GPS Module	UART output, 9600 baud, NMEA sentences. Usually ships with a ceramic patch antenna
915 MHz antenna	Whip or helical, soldered or connected to the RFM95 ANT pad
USB cable (Micro-USB)	To power and program the Nano 33 BLE from a computer
Breadboard or PCB	For prototyping connections

Important: The Nano 33 BLE operates at 3.3 V logic only. Do NOT connect any 5 V signals — they will permanently damage the nRF52840 SoC. All modules listed are 3.3 V compatible.

2.2 RFM95 LoRa Module → Nano 33 BLE

The RFM95 uses SPI. All connections are direct — no level shifter required because both modules operate at 3.3 V.

RFM95 Pin	Nano 33 BLE Pin	Notes
VCC	3.3V	Do NOT connect to 5V or VIN
GND	GND	
NSS (CS)	D10	SPI chip-select (active-low)
MOSI	D11	SPI data: Arduino → RFM95
MISO	D12	SPI data: RFM95 → Arduino
SCK	D13	SPI clock
RST	D9	Hardware reset (active-low)
DIO0	D2	Interrupt: RX_Done / TX_Done
DIO1	Not connected	Unused in receive-only mode
DIO2	Not connected	Unused in receive-only mode

RFM95 Pin	Nano 33 BLE Pin	Notes
ANT	915 MHz antenna	Solder whip directly to ANT pad

LoRa Settings — MUST Match Flight Logger v2

Parameter	Value
Frequency	915 MHz USA (change LORA_FREQUENCY for other regions)
Spreading Factor	SF9
Signal Bandwidth	125 kHz
Coding Rate	4/5
CRC	Enabled

2.3 GT-U7 GPS Module → Nano 33 BLE

The GPS connects to Hardware Serial1 (pins D0 and D1) for reliable, interrupt-free NMEA parsing.

GT-U7 Pin	Nano 33 BLE Pin	Notes
VCC	3.3V	
GND	GND	
TX (GPS out)	D0 (Serial1 RX)	GPS sends NMEA → Arduino reads
RX (GPS in)	D1 (Serial1 TX)	Optional: Arduino can send PMTK config commands

> **Tip:** Position the GPS antenna with a clear view of the sky. Cold-start fix time is 1–3 minutes outdoors. The red LED on the GT-U7 blinks once per second when a satellite fix is active.

2.4 LED Status — Nano 33 BLE Built-in RGB LED

The Nano 33 BLE has an RGB LED that is active-low (LOW = on, HIGH = off). The firmware uses it to indicate BLE status.

LED State	Meaning
Red (LEDR on)	Bluetooth advertising — no phone connected

LED State	Meaning
Green (LEDG on)	Bluetooth connected to iPhone

> **Warning:** LED_BUILTIN on the Nano 33 BLE maps to D13 internally, which is also the SPI SCK line used by LoRa. Always use LEDR/LEDG/LEDB constants — never LED_BUILTIN — to avoid corrupting SPI communication with the RFM95.

2.5 Quick-Reference Pin Table

Nano 33 BLE	RFM95 LoRa	GT-U7 GPS
3.3V	VCC	VCC
GND	GND	GND
D0 (RX1)	—	TX
D1 (TX1)	—	RX
D2	DIO0	—
D9	RST	—
D10	NSS (CS)	—
D11 (MOSI)	MOSI	—
D12 (MISO)	MISO	—
D13 (SCK)	SCK	—

2.6 Power Budget

Module	Current (approx.)
Nano 33 BLE	~13 mA idle, ~22 mA with BLE active
RFM95 (receive mode)	~10 mA
GT-U7 GPS	~25 mA acquiring, ~20 mA tracking
Total	~55–60 mA

A 1000 mAh LiPo battery provides roughly 16 hours of continuous operation.

3 Arduino Firmware (base_station_v7.ino)

3.1 Required Libraries

Install all three via Arduino IDE → Tools → Manage Libraries:

Library	Search Term	Author
ArduinoBLE	ArduinoBLE	Arduino
Arduino LoRa	LoRa	Sandeep Mistry
TinyGPS++	TinyGPSPlus	Mikal Hart

> **Warning:** Install 'Arduino-LoRa' by Sandeep Mistry. Do NOT install RadioHead — it uses a different API and will not compile with this sketch.

3.2 Board Selection

7. Open Arduino IDE → Tools → Board → Boards Manager.
8. Search for 'Nano 33 BLE' and install Arduino Mbed OS Nano Boards.
9. Select Tools → Board → Arduino Mbed OS Nano Boards → Arduino Nano 33 BLE.
10. Select the correct port under Tools → Port.
11. Click Upload (Ctrl+U / ⌘U).

3.3 Expected Serial Monitor Output at Startup

Set baud rate to 115200. You should see:

```
===== BASE STATION v7 (Nano 33 BLE) =====
GPS started on Serial1 (D0/D1) @ 9600 baud
LoRa OK: SF9, 125 kHz, 915 MHz — RX mode, no level shift needed
BLE OK: Advertising as 'FlightBase'
  Service UUID : A1B2C3D4-0000-1000-8000-00805F9B3401
  Nav Char UUID: A1B2C3D4-0000-1000-8000-00805F9B3402
```

3.4 How the Firmware Works

The main loop() performs four tasks on every iteration:

- BLE.poll() — services the BLE stack: processes connections, disconnections, reads, and notification acknowledgements. Must be called frequently.
- GPS feed — reads bytes from Serial1 (D0) and passes them to TinyGPS++. When a valid NMEA fix is decoded, baseLat and baseLon are updated.
- LoRa receive — calls LoRa.parsePacket(). If a packet is available, it reads the 8-field CSV from the flight logger, parses it, and logs full telemetry to USB Serial.
- BLE notify — once per second (when a packet has been received and a phone is subscribed), computes navigation and writes the CSV characteristic.

3.5 Navigation Calculations

Calculation	Formula	Output
Ground distance	Haversine formula on base and logger coordinates	Metres
Bearing	Forward azimuth (atan2 of cross-track components)	Degrees 0–360°
Steps	distanceMetres ÷ 0.762 m (average adult stride)	Integer count
Cardinal direction	8-sector lookup: N/NE/E/SE/S/SW/W/NW	String

3.6 Changing LoRa Frequency for Your Region

Region	Frequency	LORA_FREQUENCY value in sketch
USA, Canada	915 MHz	`915E6`
Europe	868 MHz	`868E6`
Asia (some)	433 MHz	`433E6`
Australia	915 MHz	`915E6`

Change the value in BOTH the base station sketch and the flight logger sketch.

4 iOS App (Flight Tracker)

4.1 File Overview

File	Role
FlightTrackerApp.swift	SwiftUI @main entry point — no changes needed
BLEManager.swift	Core Bluetooth: scanning, connecting, CSV parsing, published state
ContentView.swift	All UI: scan list, connecting screen, map dashboard, telemetry panel

4.2 Required Info.plist Keys

Add both keys to your Xcode project's Info.plist before running on a physical device. Without them, iOS will crash the app on first Bluetooth or Location access.

Key	Suggested Value
NSBluetoothAlwaysUsageDescription	Flight Tracker uses Bluetooth to connect to the FlightBase ground station.
NSLocationWhenInUseUsageDescription	Flight Tracker shows your location on the map to help you navigate to the logger.

4.3 App Screen Flow

12. ScanView — Bluetooth scans for peripherals advertising the Flight Navigation Service UUID. Lists found FlightBase devices with signal strength. User taps Connect.
13. ConnectingView — Spinner while: (a) central connects to peripheral, (b) discovers Flight Navigation Service, (c) subscribes to Nav characteristic, (d) reads initial value.
14. MapDashboardView — Full-screen MapKit map with three labelled pins, connection bar (RSSI + disconnect), and telemetry panel.

4.4 Map Pins

Pin	Symbol	Data Source
Blue — Base Station	antenna.radiowaves.left.and.right	BLE characteristic fields 0–1 (baseLat, baseLon)
Red — Flight Logger	airplane	BLE characteristic fields 2–3 (logLat, logLon)
Blue dot — iPhone	System user location dot	CoreLocation (CLLocationManager)

4.5 Telemetry Panel

The bottom panel shows two navigation sections:

- Base → Logger: distance, bearing, steps, and direction banner — values come directly from the BLE characteristic (computed on the Arduino).
- Phone → Logger: distance, bearing, steps, and direction banner — computed in the iOS app using CLLocation.distance(from:) and haversineForwardBearing(). Only shown when Location permission is granted.

4.6 Xcode Setup Steps

15. Create a new iOS App project in Xcode (SwiftUI, minimum target iOS 15).
16. Replace the generated files with the four .swift files provided.
17. Add Info.plist keys for Bluetooth and Location (see 4.2).
18. Connect your iPhone via USB cable.
19. Select your iPhone as the run destination (not Simulator — Core Bluetooth is disabled in Simulator).
20. Click Run (⌘R).
21. On first launch, approve both Bluetooth and Location permission dialogs.

5 BLE Communication Protocol

5.1 Connection Sequence

```
iPhone (Central)                    Arduino (Peripheral)
──────────────────────────────────────────────────────────
scanForPeripherals(services: [FlightServiceUUID])
                          <── Advertising 'FlightBase'
connect(peripheral)
                          ─── ACK ──>
discoverServices([FlightServiceUUID])
                          <── Service discovered
discoverCharacteristics([NavCharUUID])
                          <── Characteristic discovered
setNotifyValue(true, for: navChar)
                          ─── Subscription ACK ──>
readValue(for: navChar)           <- immediate read
                          <── Current CSV value
                          <── Notify: CSV (every 1 second)
```

5.2 MTU Negotiation

iOS 15+ automatically negotiates ATT MTU up to 251 bytes during connection. The Nav characteristic value is at most 128 bytes and always fits in a single ATT PDU — no fragmentation or reassembly is needed in the iOS app.

5.3 Reconnection Behaviour

Scenario	Behaviour
Unexpected disconnect (range loss, power loss)	App auto-rescans after 2 seconds
User-initiated disconnect (tap × button)	App returns cleanly to the scan list
Arduino loses BLE connection	Arduino resumes advertising automatically; LED turns red
App reopened while Arduino advertising	Scan list immediately shows FlightBase; user taps Connect

6 Verifying with nRF Connect

Use nRF Connect to confirm the Arduino is transmitting correctly before testing the iOS app.

22. Install nRF Connect for Mobile (Nordic Semiconductor — free, App Store).
23. Power on the base station. The RGB LED should be red (advertising).
24. Open nRF Connect → Scanner tab → tap Scan.
25. Tap Connect next to FlightBase.
26. Tap the Client tab → expand Unknown Service A1B2C3D4…3401.
27. Find characteristic A1B2C3D4…3402.
28. Tap the ⬇ (down arrow) button to enable notifications.
29. Tap the three-dot menu → Show as UTF-8. (Default is hex — this step is essential.)

You should see the CSV updating every second:

```
37.774900,-122.419400,37.775900,-122.418800,134.5,47.2,176,NE,18.3,274.0,-95
```

> **Note:** If the base station GPS is not yet locked, latitude/longitude fields will read 0.000000. Allow 1–3 minutes outdoors for GPS acquisition.

7 Troubleshooting Guide

7.1 BLE / Connectivity

Symptom	Likely Cause	Fix
FlightBase not in scan list	Bluetooth off on iPhone	Enable in Settings → Bluetooth
FlightBase not in scan list	BLE library failed to init	Open Serial Monitor — look for ERROR: BLE init failed
FlightBase not in scan list	UUID mismatch between sketch and app	Confirm A1B2C3D4-0000-1000-8000-00805F9B3401 in both files
Connects then immediately drops	Power supply instability	Ensure Nano is powered from USB or a stable regulated 3.3V source
'Navigation characteristic not found'	UUID mismatch on char	Confirm …3402 in both sketch and BLEManager.swift
nRF Connect shows no service	Old firmware still flashed	Re-upload base_station_v7.ino
App shows 'Bluetooth access denied'	Permission not granted	Settings → Privacy & Security → Bluetooth → Flight Tracker → Enable
Connection drops after ~30 s	iOS suspends background BLE	Add bluetooth-central to UIBackgroundModes in Info.plist for persistence

7.2 LoRa Reception

Symptom	Likely Cause	Fix
No packets received	Frequency mismatch	Verify LORA_FREQUENCY matches flight logger
No packets received	SF/BW mismatch	Confirm SF9, 125 kHz, CR 4/5 on both ends
No packets received	Missing antenna	Transmitting without antenna damages RFM95 — ensure antenna is connected
Packets received but garbled	CRC disabled	Ensure LoRa.enableCrc() is called on both devices
'WARN: Malformed packet'	Wrong CSV field count	Verify flight logger sends exactly 8 comma-separated fields
Very low RSSI (< −100 dBm)	Range or obstructions	SF9 at 915 MHz has ~5 km line-of-sight; move closer or raise antenna

7.3 GPS

Symptom	Likely Cause	Fix
baseLat always 0.0	No GPS fix yet	Wait 1–3 min outdoors; check Serial Monitor for GPS data
GPS fix lost indoors	Normal behaviour	GPS requires clear sky view — always use outdoors
GPS fix never achieved	Wrong baud rate	Confirm GT-U7 factory default is 9600 baud; check Serial1.begin(9600)
GPS TX on wrong pin	Wiring error	GT-U7 TX must connect to Nano D0 (Serial1 RX)

7.4 iOS App Display

Symptom	Likely Cause	Fix
'Waiting for flight logger data…' stuck	Logger not transmitting LoRa	Check logger is powered and transmitting
Map pins at 0,0 (Gulf of Guinea)	No GPS fix on base station	Allow GPS to acquire fix outdoors
Steps showing 0	Distance = 0 (no base GPS)	Wait for base station GPS lock
iPhone pin missing from map	Location permission denied	Settings → Privacy → Location Services → Flight Tracker → While Using
Phone→Logger section missing	Same as above	Grant Location permission
App crashes on launch	Missing Info.plist keys	Add both NSBluetooth and NSLocation usage description keys

7.5 Serial Monitor

Symptom	Likely Cause	Fix
Garbage characters	Wrong baud rate	Set Serial Monitor to 115200 baud
ERROR: LoRa init failed	SPI wiring incorrect	Check D10/11/12/13 + D9/D2 connections to RFM95
ERROR: BLE init failed	ArduinoBLE not installed	Install via Library Manager
No output at all	Wrong board selected	Confirm Arduino Nano 33 BLE in Tools → Board

Appendix A — Stride Length Customisation

The step estimate uses 0.762 m (2.5 ft) per step — a standard gait-analysis reference value. To calibrate for your own stride:

- Walk 10 steps on flat ground. Measure the total distance. Divide by 10 to get your stride length in metres.
- Update STEP_LENGTH_M in base_station_v7.ino and the 0.762 constant in ContentView.swift to match.

Location	Code to change
base_station_v7.ino line ~107	`#define STEP_LENGTH_M 0.762f`
ContentView.swift iPhoneSteps	`let steps = Int((d / 0.762).rounded())`

Appendix B — Enabling Background BLE

By default, iOS may suspend BLE scanning when the app moves to the background. To keep the connection alive:

30. In Xcode, select your target → Signing & Capabilities.
31. Add the Background Modes capability.
32. Check Uses Bluetooth LE accessories.

This adds bluetooth-central to UIBackgroundModes in Info.plist. The BLE connection will now persist even when the screen is locked or the app is backgrounded.

Appendix C — Glossary

Term	Definition
AGL	Above Ground Level — barometric altitude relative to launch elevation
ATT	Attribute Protocol — the base BLE data transport layer
Bearing	Compass direction from one point to another, measured clockwise from North (0–360°)
Cardinal	8-point compass direction: N, NE, E, SE, S, SW, W, NW
CSV	Comma-Separated Values — the text format used by the BLE characteristic
DIO0	Digital I/O pin 0 on RFM95 — signals RX_Done interrupt to the Arduino
Haversine	Mathematical formula that calculates great-circle distance between two GPS coordinates

Term	Definition
MTU	Maximum Transmission Unit — the largest payload that fits in one BLE packet
NMEA	Standard GPS sentence format output by the GT-U7 module
NUS	Nordic UART Service — the old BLE serial approach replaced in v7
RSSI	Received Signal Strength Indicator — LoRa link quality in dBm (more negative = weaker)
SPI	Serial Peripheral Interface — used to communicate between Arduino and RFM95
UUID	Universally Unique Identifier — identifies a BLE service or characteristic