Development Guide:
EKF Sensor Fusion with iPhone IMU/GPS for RoboMaster S1

Internship Project Report

August 11, 2025

Abstract

This guide explains how to integrate an iPhone 14 Plus (iOS 18.7) running SensorLog with a Raspberry Pi 4 (ROS Noetic) to stream IMU, GPS, magnetometer, and barometer data into a 15-state Extended Kalman Filter (EKF). It emphasizes methodology and sensor capabilities, network architecture, ROS bridging, EKF rationale, and detailed development steps, enabling deeper understanding before coding.

Contents

1 SensorLog Streams and Their Roles

SensorLog can export all built-in iPhone sensors at up to 100 Hz. Below is a compact table of the primary streams:

Streaming Modes:

- **TCP Server:** highest throughput, minimal delay. Client (Pi) connects to phone's IP:port.
- **HTTP POST:** easy through firewalls, limited to tens of Hz.
- **On-device CSV:** full logging for offline analysis, later retrieved via AirDrop or USB.

Sensor	Description & Usage	ROS Topic
Accelerometer (user)	3-axis linear acceleration, gravity removed (g). Used to predict velocity and position by integrating body-frame acceleration (rotated to NED).	`/ios/imu_accel`
Gyroscope	3-axis angular rate (rad/s). Drives attitude propagation (roll, pitch, yaw) in the filter.	`/ios/imu_gyro`
Magnetometer	3-axis magnetic field (µT). Provides absolute heading corrections for yaw.	`/ios/mag`
Barometer	Relative altitude change (m) and pressure (kPa). Supplies high-rate vertical position updates.	`/ios/baro`
GPS	Latitude, longitude, altitude, speed, course. Anchors absolute position and velocity (converted to NED frame).	`/ios/fix`
CoreMotion Attitude (opt.)	iOS-computed orientation quaternion. Useful for rapid prototyping but bypasses your own attitude fusion logic.	`/ios/imu_quat`

Table 1: Key SensorLog outputs mapped to ROS topics

2 15-State EKF Architecture

2.1 State Vector

We adopt the classical INS/GNSS error–state formulation:

$$\mathbf{x} = \left[p_N, p_E, p_D,\ v_N, v_E, v_D,\ \phi, \theta, \psi,\ b_{\omega_x}, b_{\omega_y}, b_{\omega_z},\ b_{a_x}, b_{a_y}, b_{a_z}\right]^{\mathsf{T}}.$$

Bias terms b_ω, b_a model gyro and accelerometer drift.

2.2 Process Model Rationale

- *Position*: double–integrate bias-corrected acceleration in body frame, apply rotation $R(\phi, \theta, \psi)$, add gravity.
- *Velocity*: single–integrate same acceleration.
- *Attitude*: integrate gyro minus bias to update Euler angles.
- *Biases*: model as random-walk processes to capture slow drift.

2.3 Measurement Updates

GPS supplies absolute position (p_N, p_E, p_D) and velocity; corrects both states at 1–5 Hz.

Magnetometer yields yaw ψ; corrects long-term heading drift at up to 25 Hz.

Barometer provides p_D directly; refines vertical position at 25 Hz.

3 Network & ROS Integration

Hardware Roles:

- **iPhone:** runs SensorLog in TCP mode on port 56204, streaming selected sensors.
- **Wi-Fi Router:** common 5 GHz SSID for phone and Pi.
- **Raspberry Pi 4:** runs ROS Noetic, bridge node, and EKF node.
- **Laptop:** SSH/ROS monitor; also connects to RoboMaster's own Wi-Fi for control via DJI SDK.

4 Detailed Development Methodology

4.1 1. Bridge Setup and Verification

First, install ROS Noetic on Ubuntu 22.04. Clone and build a SensorLog–to–ROS bridge (e.g. imu_from_ios_sensorlog). On the phone, enable only the six streams listed above. Launch the bridge and confirm that `/ios/imu_accel`, `/ios/imu_gyro`, `/ios/mag`, `/ios/baro`, and `/ios/fix` appear at the expected rates. Check timestamps for consistency.

4.2 2. EKF Node Configuration

Use the `robot_localization` package to prototype your EKF:

a. Write an `ekf.yaml` that subscribes to the five topics above.

b. Set up the 15-state mask so that position, velocity, attitude, and biases are all estimated.

c. Initialize process noise Q based on sensor datasheets (e.g. accelerometer noise density, gyro bias stability).

d. Initialize measurement noise R from SensorLog-reported accuracies (GPS HDOP, barometer resolution).

e. Launch and view the fused `/odometry/filtered` in RViz to verify the pose stream.

Iterate tuning live: increase measurement variances if the filter overreacts to noise; increase bias-process noise if slow drifts remain uncorrected.

4.3 3. Real-Time Validation

Record a rosbag of both raw and fused topics:

```
rosbag record -O run_demo /ios/* /odometry/filtered
```

Manually move the robot or carry the phone through a simple walk-through. Verify that the filtered pose follows true motion without diverging or oscillating. Adjust covariances until the estimate is both responsive and stable.

4.4 4. Offline Comparative Analysis

Extract CSV from the rosbag (or directly use SensorLog's CSV). In Python (pandas and matplotlib), compute trajectory errors and RMSE between:
1. EKF fused pose vs. raw GPS.
2. EKF fused pose vs. pure inertial dead-reckoning.
3. Alternative filters (UKF, RTS smoother) applied offline.

Alternatively, use MATLAB's Navigation Toolbox to prototype an UKF or fixed-lag smoother and compare performance metrics.

4.5 5. RoboMaster S1 Integration

Maintain the Pi's connection to the common router to keep the EKF running. From the laptop (on the S1's AP), send high-level velocity or waypoint commands via the DJI Python SDK. Use the EKF–estimated pose on the Pi (published on a ROS topic) to implement conditional behaviors (e.g. hold orientation until fused heading error $< 5°$, then advance). Perform incremental field tests in an open area, first verifying heading control, then position control.

5 Key References

- Särkkä, *Bayesian Filtering and Smoothing* — EKF theory and Jacobian derivations.
- FOI report: 15-state INS/GNSS EKF — full matrices F, H, Q, R.
- PX4 ECL EKF source code — practical error-state implementation.
- `robot_localization` ROS docs — EKF/UKF examples and parameter guides.
- imu_from_ios_sensorlog — SensorLog to ROS bridge.