Flight-Ready LiDAR-Inertial Odometry for Embedded Drone Platforms

Alvaro J. Gaona, David Perez-Saura, Francisco J. Anguita, and Pascual Campoy
Computer Vision and Aerial Robotics Group at Centre for Automation and Robotics C.A.R. (UPM-CSIC), Universidad Politécnica de Madrid (CVAR-UPM), Calle Jose Gutierrez Abascal 2, 28006 Madrid, Spain

Abstract

Open-source LiDAR–inertial odometry (LIO) systems have achieved remarkable benchmark accuracy, yet current state-of-the-art implementations are primarily optimized for evaluation performance rather than the requirements of real-time closed-loop aerial control. When deployed onboard UAVs, this can introduce limitations that degrade flight performance. In this work, we identify five architectural deficiencies in a representative tightly coupled IESKF-based LIO implementation: odometry publishing tied to the LiDAR rate (10 Hz instead of the IMU's 200 Hz), missing velocity outputs, execution bottlenecks that block IMU processing, mutex contention, and synchronization race conditions. We introduce corresponding modifications including IMU-rate forward propagation, direct body-frame velocity publishing, SLERP-based smoothing, dual-executor isolation, and explicit synchronization protection. The resulting system increases odometry output from 10 Hz to a stable 200 Hz, provides a complete Twist state at every IMU sample, and preserves continuity during transient LiDAR loss. Experiments on a Livox Mid-360 / Pixhawk 4 Mini autonomous UAV with motion-capture ground truth validate the approach. Since the underlying estimator (IESKF + ikd-Tree) remains unchanged, the proposed improvements can be directly applied to FAST-LIO2-derived implementations.

1 Introduction

Tightly-coupled LiDAR-inertial odometry has matured rapidly over the last five years, with implementations such as FAST-LIO2 [1], LIO-SAM [2], Point-LIO [3], and DLIO [4] routinely reporting centimeter-level accuracy on standard benchmark datasets. These implementations are open-source, well-documented, and widely cited; a roboticist building an aerial-mapping system has every reason to start from one of them.

A roboticist building an autonomous *flight* system, however, quickly discovers that benchmark accuracy and flight-readiness are not the same thing. A flight controller imposes requirements that benchmark evaluations do not exercise: pose feedback at hundreds of Hertz to keep the inner control loop stable, linear and angular velocity estimates at the same rate for damping and feedforward, low-noise signals that do not excite actuator oscillations, and uninterrupted continuity even during transient sensor degradation. An LIO implementation that meets none of these requirements out of the box is not unusable; it is simply not yet ready for flight.

Figure 1: Quadrotor platform used for the validation flights: Livox Mid-360 LiDAR on top, Pixhawk 4 Mini autopilot and Jetson Orin NX onboard computer below, reflective mocap markers on the prop guards. Flight footage: `https://vimeo.com/1192528650?share=copy&fl=sv&fe=ci`.

This paper documents the modifications required to take a representative open-source IESKF-based LIO implementation from benchmark-tuned to flight-ready.[1] The five contributions are:

1. A forward-propagation scheme that publishes pose and velocity at every IMU sample (200 Hz) using the most recent IESKF-corrected state as an anchor.

2. Body-frame linear and angular velocity derived directly from the filter state, populating the previously-empty field of the odometry message.

3. A first-order EMA filter on velocity and SLERP-based filtering on orientation, with cutoff frequencies chosen to remove integration noise without obscuring drone-bandwidth maneuvers.

[1] Source code: `https://github.com/alvgaona/fr-lio`.

4. A dual-executor architecture that isolates the IMU callback from scan-processing latency, eliminating rate-degradation under load.

5. An identification and fix of a race condition in the IMU/LiDAR synchronization function that emerges only under multi-threaded execution.

These modifications are purely architectural: the underlying IESKF state propagation, point-to-plane registration, and ikd-Tree map remain unchanged. They apply to any FAST-LIO2-derived implementation and to any IESKF-based system where the filter correction operates at a rate slower than the IMU.

The paper is organized as follows. Section 2 surveys representative open-source LIO implementations and characterizes the flight-readiness gap each leaves open. Section 3 reviews the publishing pipeline of the baseline implementation and identifies the limitations that motivate each modification. Section 4 presents the IMU-rate forward propagation and body-frame velocity derivation. Section 5 describes the dual-executor architecture, mutex-contention reduction, and synchronization race-condition fix. Section 6 covers EMA/SLERP filtering and chained-propagation fallback for LiDAR-interruption robustness. Section 7 reports flight-test validation on a Livox Mid-360 / Pixhawk 4 Mini platform with motion-capture ground truth. Section 8 concludes.

2 RELATED WORK

Modern LIO implementations have converged on two architectural patterns: tightly-coupled iterated error-state extended Kalman filters (IESKF) with incremental map structures, and tightly-coupled factor graphs with incremental smoothing back-ends. Both deliver centimeter-accurate trajectories on benchmark datasets [1, 2, 3, 4]; neither was designed with the requirements of an in-flight controller in mind. We briefly review the four most widely-used representatives and characterize the flight-readiness gap each leaves open.

FAST-LIO2 [1] is the canonical IESKF+*ikd*-Tree implementation. The state is propagated forward by IMU integration over each scan interval, then corrected by a single IESKF iteration loop with point-to-plane registration against the *ikd*-Tree map. Publishing happens once per scan, at the end of the IESKF correction. The default ROS 1/2 deployment uses a single-threaded executor, so all callbacks (IMU buffering, LiDAR buffering, IESKF processing, publishing) serialize on one thread. The Twist field of the published `Odometry` message is left at zero. None of these choices is wrong for offline mapping or benchmark evaluation; all four limit flight-readiness directly.

LIO-SAM [2] replaces the IESKF with a tightly-coupled factor graph optimized incrementally via iSAM2. It supports GPS and loop-closure factors, which makes it well-suited to outdoor mapping. Publishing rate is again tied to the keyframe selection (typically 1–10 Hz), and the published velocity is derived from preintegrated IMU factors only at keyframe boundaries. The factor-graph back-end is computationally heavier than an IESKF and the per-keyframe variability makes the publishing rate unpredictable, both of which compound the flight-control problem rather than relieve it.

Point-LIO [3] addresses high-bandwidth motion by treating each LiDAR point as an individual measurement update rather than batching points into accumulated scans. This raises the IESKF correction rate substantially (the authors report up to kilohertz output) and is well-suited to aggressive motions where intra-scan distortion would otherwise dominate. The output frequency is high, but the architecture inherits its FAST-LIO2's single-executor concurrency model; under sustained heavy load the same thread still serializes IMU, LiDAR, and publishing. Point-LIO closes one of the four flight-readiness gaps (rate) without addressing the others (velocity availability, executor isolation, race-condition robustness).

DLIO [4] is a recent lightweight LIO that adds continuous-time motion correction to deskew scans during integration, packaged with a clean ROS 2 implementation. The lightweight design makes it attractive for embedded deployment, and the ROS 2 native architecture supports multi-threaded executors in principle. However, the published implementation does not by default isolate the IMU callback to a dedicated executor, does not populate the Twist field with body-frame velocities, and inherits the same publishing-coupled-to-LiDAR pattern as FAST-LIO2. The architectural improvements are at the estimation layer, not at the deployment layer.

Summary. Across all four representative implementations, the flight-readiness gap manifests as the same four issues: publishing rate coupled to (or capped by) the LiDAR rate, missing velocity fields, single-threaded execution that blocks the IMU callback under load, and an absence of any mechanism to maintain odometry continuity during transient LiDAR loss. Table 1 summarizes which axes each implementation addresses out of the box. The modifications presented in this paper close all four gaps simultaneously and apply directly to any of these implementations, since they intervene at the publishing layer rather than at the estimation layer; the underlying choice of IESKF vs. factor graph, batched scans vs. per-point updates, or discrete vs. continuous-time motion correction is independent of the architectural fix. Visual-inertial state estimators such as VINS-Mono [5] and OpenVINS [6] already publish at IMU rate by propagating the corrected state between visual updates; our contribution can be read as adapting that publishing pattern to LIO.

3 BACKGROUND AND LIMITATIONS

The baseline implementation publishes odometry inside the LiDAR scan-processing loop. Each cycle accumulates a complete scan, executes IMU forward propagation across

	200 Hz rate	Velocity output	Isolated IMU thread	LiDAR-loss resilience
FAST-LIO2 [1]	no	no	no	no
LIO-SAM [2]	no	partial	no	no
Point-LIO [3]	yes	no	no	no
DLIO [4]	no	no	no	no
This work	**yes**	**yes**	**yes**	**yes**

Table 1: Flight-readiness axes addressed by representative open-source LIO implementations. "Velocity output" = body-frame linear and angular velocities published alongside the pose. "Isolated IMU thread" = IMU sampling runs on a dedicated thread, decoupled from scan-processing latency. "LiDAR-loss resilience" = mechanism to maintain odometry continuity when LiDAR scans stop arriving for a transient period.

the scan interval, performs IESKF iterations with point-to-plane registration, updates the ikd-Tree map, and publishes the resulting pose. The publishing rate is therefore tied to the LiDAR scan-arrival rate.

For the Livox Mid-360 sensor at 100 Hz, an accumulator collecting 10 packets per IESKF cycle yields a ∼10 Hz publishing rate. At a flight speed of 2 m/s, the controller receives a new pose every 0.2 m of travel and must extrapolate between updates—an extrapolation that is unsafe for tight closed-loop control. Three structural issues underlie this:

Publishing rate coupled to LiDAR rate. When scan processing exceeds the inter-scan period, for example during heavy ikd-Tree insertions, the publishing rate degrades below 10 Hz. The rate is therefore not a guarantee but a best-case.

Absence of velocities. The published odometry message carries only the pose; the velocity component is left at zero. The flight controller must numerically differentiate position to obtain velocity, amplifying high-frequency noise and introducing at least one sampling period of lag.

Single-threaded execution. The baseline serializes IMU buffering, LiDAR buffering, IESKF processing, and publishing on a single thread. While the IESKF processes a scan, the IMU samples are queued. When publishing happens at the end of scan processing this serialization is harmless; moving the publication to the IMU sample rate to achieve 200 Hz makes blocking the dominant bottleneck.

4 IMU-RATE FORWARD PROPAGATION

4.1 Operating Principle

The IESKF correction occurs at the accumulated LiDAR rate (∼10 Hz). Between corrections, each new IMU sample propagates the state forward using the inertial dynamics. The propagation does not replace the filter; it uses the most recent corrected state as an *anchor* and applies single-step integration with the current IMU measurement.

Let t_a denote the time of the last IESKF correction and $(\mathbf{R}_a, \mathbf{p}_a, \mathbf{v}_a, \mathbf{b}_{g,a}, \mathbf{b}_{a,a}, \mathbf{g}_a)$ the corrected anchor state. For an IMU sample at time $t > t_a$ with raw measurements $\boldsymbol{\omega}_m$ and $\mathbf{a}_m$, the bias-corrected angular velocity and body-frame acceleration are

$$\boldsymbol{\omega} = \boldsymbol{\omega}_m - \mathbf{b}_{g,a}, \quad \mathbf{a}_b = \mathbf{a}_m - \mathbf{b}_{a,a}, \tag{1}$$

and the world-frame acceleration is $\mathbf{a}_w = \mathbf{R}_a \mathbf{a}_b + \mathbf{g}_a$. With $\Delta t = t - t_a$, the propagated state is

$$\mathbf{p}(t) = \mathbf{p}_a + \mathbf{v}_a \Delta t + \tfrac{1}{2}\mathbf{a}_w \Delta t^2, \tag{2}$$

$$\mathbf{v}(t) = \mathbf{v}_a + \mathbf{a}_w \Delta t, \tag{3}$$

$$\mathbf{R}(t) = \mathbf{R}_a \,\mathrm{Exp}(\boldsymbol{\omega} \Delta t), \tag{4}$$

where $\mathrm{Exp} : \mathbb{R}^3 \to \mathrm{SO}(3)$ is the SO(3) exponential map. Equation (4) is the zeroth-order discretization, treating $\boldsymbol{\omega}$ as constant over Δt; this is appropriate at the 5 ms IMU period. The integration also assumes that $\mathbf{R}_a$, the biases, and gravity remain constant over Δt. For $\Delta t < 0.1$ s (the nominal 10 Hz IESKF rate), the resulting error is on the order of centimeters in position and fractions of a degree in orientation [7]—negligible relative to the controller's resolution.

4.2 Anchor Update

Each IESKF correction copies its output state to the propagation anchor under mutex protection. The anchor stores position, velocity, rotation, gyroscope and accelerometer biases, gravity, and the timestamp of the corresponding scan end. The next IMU sample resets Δt to a small value, so propagation never accumulates error beyond the inter-correction interval.

4.3 Body-Frame Velocities

The IESKF state vector includes velocity in the world frame, $\mathbf{v} \in \mathbb{R}^3$. The flight controller requires velocity in the body frame, obtained via the inverse rotation:

$$\mathbf{v}_b(t) = \mathbf{R}(t)^\top \mathbf{v}(t). \tag{5}$$

The angular velocity in the body frame is the bias-corrected gyroscope measurement, available directly from (1). Both quantities populate the previously-empty Twist field of the published `Odometry` message.

5 THREADING AND CONCURRENCY

5.1 Dual-Executor Isolation

To publish at 200 Hz, the IMU callback must execute without delay at every sample. A single-threaded executor cannot satisfy this when scan processing takes tens of milliseconds (typical during heavy ikd-Tree insertions): under load, the IMU callback is starved for the duration of every IESKF cycle and the effective publishing rate degrades to the LiDAR rate.

The fix uses two independent single-threaded executors. The IMU callback is registered to an exclusive callback group assigned to a dedicated executor that runs on its own thread;

the LiDAR callback, IESKF timer callback, and point-cloud publishers are assigned to the main executor. The IMU thread is therefore decoupled from scan-processing time and contends only for shared mutexes, not for execution slots.

5.2 Mutex Contention from In-Buffer Preprocessing

Even with separate executors, the IMU thread can be blocked if it requires a mutex held by the main thread. In the baseline, the LiDAR callback acquires the buffer mutex *before* performing scan preprocessing (point filtering, format conversion), an operation that takes several milliseconds.

The fix is to perform preprocessing outside the mutex-protected section. The mutex is acquired only for the buffer insertion of the preprocessed scan, an operation that takes microseconds. This reduces mutex hold time by approximately three orders of magnitude.

5.3 Race Condition in Synchronization

The packet synchronization function pairs LiDAR scans with IMU measurements to form IESKF inputs. In the baseline, this function accesses shared buffers without mutex protection. Under single-threaded execution this is safe; under the dual-executor architecture, concurrent writes from the IMU callback and reads from the synchronization function constitute a data race that occasionally produces malformed input pairs and silently degrades estimator accuracy.

The fix protects the synchronization function with the same mutex that guards the buffers. The mutex critical section is short (single-pass over both buffers), and the operation does not appear in any benchmark trace because it is masked by IESKF processing.

5.4 Path Message Serialization

A secondary bottleneck arises from publishing the `nav_msgs::Path` message at the IMU rate. `Path` accumulates the full trajectory; at 200 Hz, after 30 s of flight the message contains 6000 poses and serialization consumes significant CPU on the IMU thread. The fix throttles `Path` publication to 10 Hz; the message is visualization data, not control data, and the rate reduction has no functional effect.

6 FILTERING AND ROBUSTNESS

6.1 EMA on Velocity and SLERP on Orientation

Single-step integration of accelerometer measurements transfers high-frequency noise directly to the propagated velocity, producing typical fluctuations of $\sim$0.2 m/s (and several-m/s peak excursions on the vertical axis, see Figure 4) that a finely-tuned velocity controller would amplify into actuator oscillations. We apply a first-order exponential moving average (EMA) to the published body-frame velocities:

$$\bar{\mathbf{v}}_b[k] = \alpha \mathbf{v}_b[k] + (1-\alpha)\bar{\mathbf{v}}_b[k-1], \tag{6}$$

with the same form applied to the body-frame angular velocity. The cutoff frequency at sampling rate f_s is $f_c = (f_s/2\pi)\ln(1/(1-\alpha))$. For $f_s = 200$ Hz and $\alpha = 0.05$,

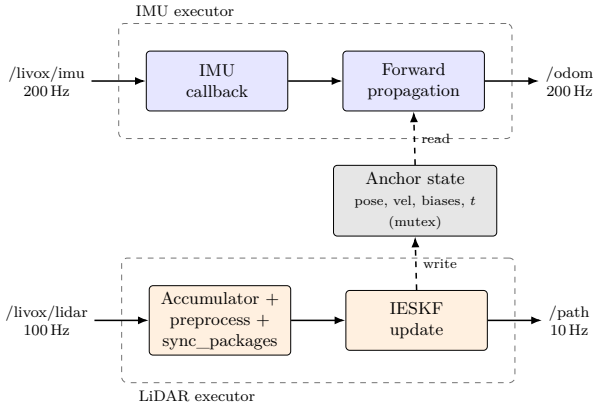

Figure 2: Dual-executor threading architecture. The IMU callback runs on its own single-threaded executor (top) and publishes the 200 Hz `/odom` stream by forward-propagating from the shared anchor state. LiDAR preprocessing and IESKF correction run on a second executor (bottom) and write the anchor on each successful update. The anchor state (pose, velocity, biases, timestamp) is mutex-protected; the critical section is short and the synchronization function acquires the same mutex.

$f_c \approx 1.6$ Hz, which removes accelerometer noise without significantly attenuating drone maneuvers under the precision-hover and indoor-trajectory regimes used in our validation (closed-loop bandwidth $\lesssim 1$ Hz). Aggressive flight regimes (racing-style trajectories, heavy payload swings) extend the controller bandwidth above this cutoff and require a larger α at the cost of higher noise pass-through.

Quaternion filtering cannot be performed component-wise. We use spherical linear interpolation (SLERP):

$$\bar{\mathbf{q}}[k] = \mathrm{SLERP}\left(\bar{\mathbf{q}}[k-1], \mathbf{q}[k], \alpha\right), \tag{7}$$

which respects the manifold structure of S^3 and reduces to weighted-and-normalized linear interpolation for small angular changes between consecutive samples.

Linear-velocity filtering is applied *after* the rotation to the body frame, not before; filtering in the world frame and then rotating injects gyroscope noise into the otherwise-clean filtered signal.

6.2 Chained Propagation under LiDAR Interruption

If LiDAR scans stop arriving (sensor obstruction, packet loss in the DDS network, transient driver failure), the anchor timestamp freezes and Δt grows without bound. For large Δt, the constant-rotation assumption of (2) breaks down and integration error becomes quadratic.

When Δt exceeds 0.5 s, the propagation is *chained*: the propagated state becomes the new anchor for subsequent IMU samples,

$$\mathbf{p}_a, \mathbf{v}_a, \mathbf{R}_a, t_a \leftarrow \mathbf{p}(t_k), \mathbf{v}(t_k), \mathbf{R}(t_k), t_k. \tag{8}$$

Biases and gravity are not updated during chaining, since the IESKF estimate remains the best available. Each subsequent integration operates on a small Δt (the IMU period, 5 ms), avoiding quadratic error accumulation.

6.3 *Anchor Timestamp Guard*

When IESKF processing of a scan takes longer than the inter-scan period, the IESKF may attempt to overwrite the anchor with a state whose timestamp is earlier than what chained propagation has already advanced to. This would produce a backward jump in the published odometry. The fix conditions the write of position, velocity, rotation, and timestamp on the new timestamp being later than the current anchor. Biases and gravity are always updated, since they are independent of the time index.

7 FLIGHT-TEST VALIDATION

We validate the modifications on the quadrotor shown in Figure 1: a Livox Mid-360 LiDAR, a Pixhawk 4 Mini autopilot, and a Jetson Orin NX running ROS 2 (Humble), with a takeoff weight of approximately 1.8 kg and a 4S 5000 mAh battery. Validation flights were conducted in the Drone Arena of the Centre for Automation and Robotics (CAR), Universidad Politécnica de Madrid, operated by the Computer Vision & Aerial Robotics (CVAR) Group. The arena is a $6 \times 6 \times 12$ m volume instrumented with 16 OptiTrack motion-capture cameras providing 6-DOF ground-truth pose at 100 Hz with submillimeter accuracy.

The flight stack is built on the Aerostack2 framework [8], which provides the high-level mission management, behaviour layer, and state-machine orchestration. The inner-loop position and attitude controller is a differential-flatness controller [9] that exploits the quadrotor's flatness property to compute attitude and thrust setpoints from desired position trajectories and their derivatives; this controller is the principal consumer of the high-rate odometry and velocity stream produced by the modifications described in this paper. Two autonomous indoor flight sequences are used in what follows. A 58 s sequence, replayed under identical inputs through both the baseline FAST-LIO and the modified implementation, supports the publishing-rate histogram and the trajectory-accuracy comparison (Figure 3, Table 2). A separate 60 s sequence supports the body-frame velocity tracking analysis (Figure 4); the same sequence, with a 1.5 s contiguous block of `/livox/lidar` messages removed offline before replay through both implementations, is used to exercise the chained-propagation fallback (Figure 5).

7.1 *Publishing Rate*

Figure 3 shows the inter-message interval distribution for the published odometry topic, comparing the baseline against the modified implementation. The modified implementation publishes at a mean interval of 5.00 ms ($\sigma = 0.90$ ms, $p_{99} = 6.95$ ms) — a steady 200 Hz — while the baseline distribution

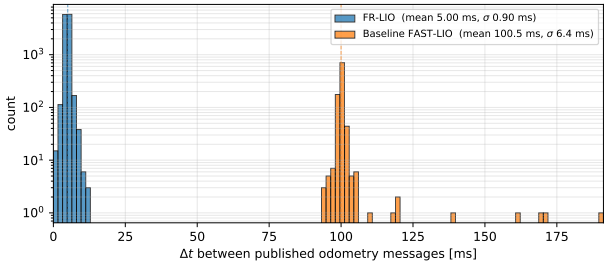

Figure 3: Inter-message interval distribution for the published odometry topic on a log-scale y-axis. The modified implementation (blue) concentrates at $\Delta t = 5$ ms ($f = 200$ Hz, $\sigma = 0.90$ ms), while baseline FAST-LIO (orange) publishes at the scan rate near $\Delta t = 100$ ms ($\sigma = 6.4$ ms).

sits at 100.5 ms ($\sigma = 6.4$ ms, $p_{99} = 107.8$ ms), reflecting its scan-rate publishing.

7.2 *Trajectory Accuracy*

Metric	Baseline	Modified
ATE RMSE (cm)	11.7	5.2
ATE median (cm)	4.7	3.8
ATE max (cm)	32.4	28.3
RPE 1 s (cm)	4.9	2.4

Table 2: Trajectory accuracy against motion-capture ground truth on a 58 s indoor flight. Each LIO trajectory is rigidly aligned to ground truth via least-squares SE(3) over the first 5 s of motion (motion onset detected at GT speed > 0.1 m/s sustained for 0.3 s).

Table 2 compares the published trajectory against motion-capture ground truth. The modified implementation matches the baseline on median ATE within 1 cm while approximately halving the RPE (1 s) drift, reflecting the tighter timing of the dual-executor architecture without altering the iterated Kalman update. The ATE max values (28–32 cm) for both implementations are isolated transients during rapid yaw segments where the IESKF registration is briefly mis-constrained by the Mid-360's reduced lateral overlap; the median and RMSE are the load-bearing comparison numbers.

7.3 *Velocity Tracking*

Figure 4 compares published body-frame velocities against ground-truth velocities derived from motion-capture position with a zero-phase Savitzky–Golay smoother (21-tap, order 3). Over the 60 s flight, the EMA-filtered output tracks ground truth with RMSE 0.05 m/s on the horizontal axes (v_x, v_y) and 0.10 m/s on v_z. The high-frequency content removed by the EMA, measured as the standard deviation of the unfiltered–filtered difference, is 5–11 cm/s per

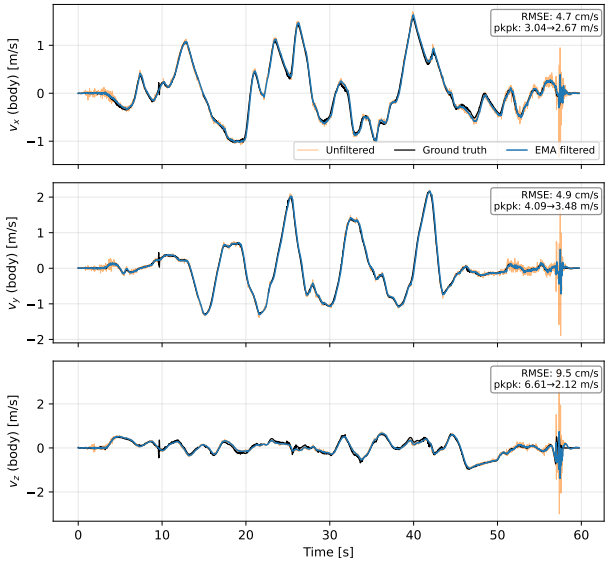

Figure 4: Body-frame linear-velocity tracking against motion-capture ground truth. EMA filter ($\alpha = 0.05$, $f_c \approx 1.6$ Hz) removes accelerometer noise without lagging the controller-bandwidth signal.

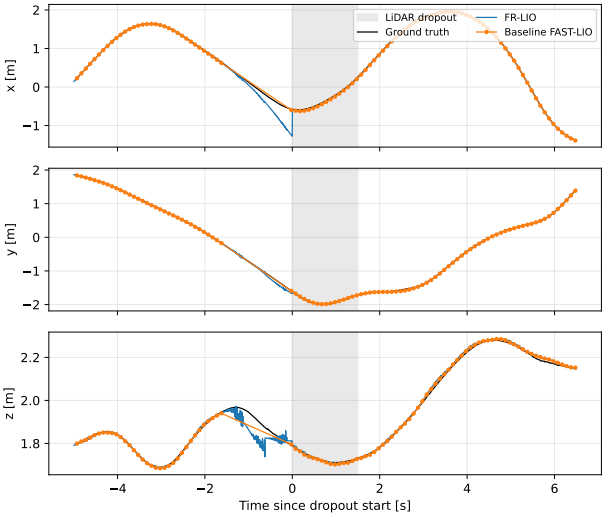

Figure 5: Position traces (x, y, z) before, during, and after a 1.5 s synthetic LiDAR dropout (gray band). The modified implementation (blue) continues publishing through the dropout via chained IMU propagation, accumulating ∼0.49 m of drift in body-frame position; baseline FAST-LIO (orange) freezes at the last EKF state until LiDAR resumes. Both implementations re-converge to ground truth (black) within one update after LiDAR returns.

axis. The largest reduction is on v_z, where the propagator's peak-to-peak noise drops from 6.6 m/s to 2.1 m/s, eliminating the high-bandwidth content that would otherwise excite the thrust loop. The magnitude of the unfiltered v_z noise reflects the operating environment of a body-mounted IMU on a quadrotor: motor harmonics in the 100–300 Hz range alias into the 200 Hz sample rate, and even a small misalignment of the estimated gravity vector projects onto v_z when integrated. The EMA at $f_c \approx 1.6$ Hz attenuates this content without touching the sub-Hz controller-bandwidth signal.

7.4 LiDAR-Interruption Resilience

To exercise the chained-propagation fallback, we apply the offline 1.5 s LiDAR-removal described above to the 60 s sequence used in Figure 4 and replay the resulting bag through both implementations. Figure 5 shows the resulting position traces. The modified implementation continues publishing throughout, accumulating approximately 49 cm of drift across the dropout. This drift reflects open-loop IMU integration at hover and converges back to ground truth on the next IESKF correction. The baseline implementation freezes at the last EKF state and resumes only when LiDAR returns. A controller running at 200 Hz under the modified implementation thus sees a smoothly drifting estimate rather than a discontinuity, which is the property that keeps the inner loop stable across the dropout.

8 Conclusion

We presented a set of architectural modifications that transform a benchmark-oriented tightly coupled LiDAR–inertial odometry implementation into a flight-ready system for embedded UAV platforms. Experimental results demonstrate that these changes substantially improve deployment characteristics without altering the underlying estimator architecture. As shown in Figure 3, the proposed system increases odometry output from the baseline scan-coupled rate of approximately 100 ms between updates (10 Hz) to a stable 5.0 ms interval (200 Hz), while significantly reducing timing variability. The modifications also preserve—and in some cases improve—estimation quality. Table 2 shows that trajectory accuracy remains comparable to the baseline in median ATE (3.8 cm vs. 4.7 cm), while reducing ATE RMSE from 11.7 cm to 5.2 cm and approximately halving the 1 s relative pose error (4.9 cm to 2.4 cm). Body-frame velocity estimation further achieves tracking errors below 0.1 m/s on all axes while attenuating high-frequency noise, particularly on the vertical channel where peak-to-peak fluctuations decrease from 6.6 m/s to 2.1 m/s (Figure 4).

Finally, Figure 5 demonstrates resilience to transient sensor failures: during a synthetic 1.5 s LiDAR interruption, the proposed approach continues publishing state estimates and limits drift accumulation to approximately 0.49 m, whereas the baseline freezes until measurements resume. Together, these results show that the proposed modifications improve timing, velocity availability, and robustness while preserving estimator accuracy. Because the IESKF + ikd-Tree estimation core remains unchanged, the approach can be directly

integrated into FAST-LIO2-derived systems and generalized to other LIO implementations where correction updates operate more slowly than IMU propagation.

ACKNOWLEDGMENT

This work has been supported by the project SHEREC "Safe Healthy and Environmental Ship Recycling", Referece: 101136056, funded by European Union under the Horizon Europe Program HORIZON-CL4-2023-HUMAN-01 CNECT.

REFERENCES

[1] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. FAST-LIO2: Fast direct LiDAR-inertial odometry. *IEEE Transactions on Robotics*, 38(4):2053–2073, August 2022.

[2] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Daniela Rus. LIO-SAM: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5135–5142, 2020.

[3] Dongjiao He, Wei Xu, Nan Chen, Fanze Kong, Chongjian Yuan, and Fu Zhang. Point-LIO: Robust high-bandwidth light detection and ranging inertial odometry. *Advanced Intelligent Systems*, 5(7):2200459, 2023.

[4] Kenny Chen, Ryan Nemiroff, and Brett T. Lopez. Direct LiDAR-inertial odometry: Lightweight LIO with continuous-time motion correction. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3983–3989, 2023.

[5] Tong Qin, Peiliang Li, and Shaojie Shen. VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[6] Patrick Geneva, Kevin Eckenhoff, Woosik Lee, Yulin Yang, and Guoquan Huang. OpenVINS: A research platform for visual-inertial estimation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4666–4672, 2020.

[7] Joan Solà. Quaternion kinematics for the error-state Kalman filter, 2017. arXiv:1711.02508.

[8] Miguel Fernandez-Cortizas, Martin Molina, Pedro Arias-Perez, Rafael Perez-Segui, David Perez-Saura, and Pascual Campoy. Aerostack2: A software framework for developing multi-robot aerial systems, 2024.

[9] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2520–2525, 2011.