Subscription charge failed. Please update your payment method.

Update

×

Thought for 30s >

Here’s the short version:

- **What node this is:** In the ROS 2 branch of LIO-SAM, `imuPreintegration.cpp` implements the **IMU pre-integration node**. It builds and repeatedly solves a small **GTSAM factor graph** that fuses raw IMU measurements with the most recent LiDAR odometry to (1) **estimate IMU bias**, (2) **propagate a high-rate (IMU-rate) odometry** estimate, and (3) **provide a good initial guess** for the downstream LiDAR optimization. The graph is **reset in a sliding-window fashion** to keep it real-time. GitHub
- **What it subscribes to / uses conceptually**
  - **Raw IMU**: accelerometer & gyro (hundreds of Hz). These get **pre-integrated** between LiDAR frames using the current bias estimate.
  - **LiDAR odometry** (lower-rate): when a new LiDAR pose arrives, the node adds an **IMU factor** and a **between-pose factor** to the graph, optimizes, and **updates the IMU bias**. (Exact topic names can vary with your params/launch.) A line-by-line walkthrough of this file confirms that pattern. limhyungtae.git... +1
- **What it publishes / provides**
  - **IMU-rate odometry (pose + vel)** in the LiDAR/base frame and a corresponding **TF** (often `odom → base_link`). This is the smooth, high-rate state propagated by IMU pre-integration, corrected whenever a LiDAR factor arrives. (The project README describes this factor graph as producing real-time odometry at IMU frequency.) GitHub
- **What the internal factor graph does**
  - Maintains a **tiny graph** with variables for pose, velocity, and **IMU bias**.
  - Adds a **prior** at startup, then repeatedly adds **pre-integrated IMU factors** between consecutive states; when a new LiDAR odom arrives it adds a **lidar-odom factor** (between-pose constraint), runs a quick **nonlinear optimization**, and **re-seeds** the pre-integrator with the refined bias/state.
  - Uses **failure detection / sanity checks** (e.g., abnormally large velocity) to **reset** pre-integration if the propagation goes off the rails—an implementation detail that shows up frequently in issues referencing this file. GitHub +1

Subscription charge failed. Please update your payment method.

fusion in a small, frequently reset graph lets the system output **low-latency motion** at IMU rate while still benefiting from LiDAR corrections at a lower rate. DSpace

If you want, I can also point out the specific callbacks and the main publish/subscribe topics in your checkout (they can differ slightly across forks/params), or walk through the pre-integration math it's using.

Sources

what if my lidar do a big mistake in rotation?

Excellent question — this is actually a **critical issue** in tightly-coupled LiDAR–IMU systems like **LIO-SAM**, especially in the context of the **IMU preintegration node** (`imuPreintegration.cpp`).

Let's unpack what happens step by step and what you can do about it:

🧭 1. What happens when LiDAR odometry gives a wrong rotation

In LIO-SAM, the **IMU preintegration node** assumes that the LiDAR odometry inputs (poses from `lio_sam/mapping`) are roughly correct and can be trusted as **keyframe anchors**.

When a new LiDAR pose arrives:

- The node **adds a factor** linking the IMU's predicted pose (from preintegration) to the incoming LiDAR pose.
- Then it **optimizes a small GTSAM factor graph** to estimate IMU biases and refine the pose/velocity.

If the LiDAR odometry has a **large rotational error**, the following chain of effects occurs:

🔄 a. The wrong LiDAR pose is added as a strong constraint

The factor graph assumes that this LiDAR rotation is true.

→ The optimizer tries to "explain" the rotation discrepancy as an IMU bias or integration

Subscription charge failed. Please update your payment method.

Now, every new IMU preintegration from that point forward is computed with a **wrong bias**, leading to a drift or even divergence in propagated poses.

🌀 c. You'll see sudden jumps or oscillations

The high-rate odometry (`/odometry/imu_incremental`) can jump violently or spin out, because the bias-corrected IMU propagation now believes the system rotated sharply or continuously.

⚠️ 2. How LIO-SAM mitigates it (but not perfectly)

LIO-SAM includes some safeguards, mainly inside `imuPreintegration.cpp` :

- **Failure detection / sanity check:**
  It monitors for unrealistic velocities, accelerations, or poses (for example, `imuIntegratorImu_` diverging).
  If detected, it **resets the preintegration** and discards corrupted state.
  → You'll see logs like "IMU preintegration reset due to large translation/rotation".
- **Graph reset after each LiDAR keyframe:**
  The small factor graph is **cleared** after every optimization.
  This limits how far a bad LiDAR rotation can propagate (it won't accumulate indefinitely).

However, these measures only **limit damage**, they don't **fix** the corrupted bias immediately — recovery may take several LiDAR frames or require a full reinitialization.

🛠️ 3. How to make it more robust

If you're experimenting or modifying LIO-SAM, here are practical steps:

Strategy	What it does	Implementation idea
Reject outlier LiDAR poses	Prevents bad constraints from entering the graph	Compare LiDAR incremental rotation with IMU-integrated rotation over the same time. If they differ by >X°, discard the LiDAR update.

Subscription charge failed. Please update your payment method.

	frame	
Temporal smoothing	Reduces shock from a single bad frame	Average several LiDAR frames or use IMU-only propagation until consistency is restored.
Bias reinitialization	Resets corrupted IMU bias estimates	Re-estimate bias from recent IMU-only window if inconsistency detected.

🔍 4. How to recognize it in logs or visualization

You'll typically see:

- Sudden discontinuity in `/lio_sam/mapping/odometry` or `/odom/imu_incremental`
- Unreasonable angular velocities or biases in `/imu/bias`
- A visible "flip" or "spin" in RViz trajectory, often followed by a reset message

---

✅ 5. Summary

Situation	**Effect**	**Recovery**
Small LiDAR rotation error	Slight bias drift, corrected next frame	Auto-corrects
Large LiDAR rotation jump	IMU bias corruption, pose jump	Needs bias reset or LiDAR rejection
Continuous wrong LiDAR orientation	IMU preintegration diverges	Full reinit required

---

Would you like me to show you **how to detect and reject LiDAR pose outliers** (e.g. code snippet you can add in `imuPreintegration.cpp`)? That's a common and effective improvement.

Subscription charge failed. Please update your payment method.

Excellent follow-up — this goes to the **core design philosophy** of systems like **LIO-SAM**.

Let's break it down clearly:

🔧 Two main categories of LiDAR–IMU fusion

1. Loosely coupled fusion

➤ Concept

The LiDAR and IMU **each estimate motion independently**, and their results are **fused afterward** (e.g., in an EKF or complementary filter).

They don't directly optimize over raw IMU and LiDAR features together.

🧩 Example architecture

1. **LiDAR odometry** produces a pose at ~10 Hz
   → $\mathbf{T}_{L_t}$
2. **IMU propagation** produces an independent high-rate pose
   → $\mathbf{T}_{I_t}$
3. A **filter (e.g., EKF)** fuses them:
   - LiDAR pose update is a measurement.
   - IMU integration provides prediction between updates.

✅ Pros

- **Simple & modular**: You can plug in any LiDAR odometry or IMU pipeline.
- **Fast & low computational load**.
- Easier to **debug** and maintain.

❌ Cons

- **No cross-sensor constraints** at the raw level → less accuracy.
- If LiDAR has a bad frame, EKF can't "re-explain" it via IMU constraints.
- IMU biases are estimated less accurately.
- Can exhibit **slower drift correction**.

📚 Examples

- LOAM + EKF,

Subscription charge failed. Please update your payment method.

2. Tightly coupled fusion

➤ Concept

Raw IMU and LiDAR data are **jointly optimized** in one factor graph (or filter).
Both sensors directly contribute constraints on the **same state variables** (pose, velocity, bias).

🧩 Example (LIO-SAM)

- The state $x = [R, p, v, b_g, b_a]$ represents rotation, position, velocity, gyro bias, and accel bias.
- **IMU factors** connect consecutive states via preintegration.
- **LiDAR factors** (pose constraints) are added whenever a LiDAR frame is processed.
- The graph optimizer (GTSAM) jointly adjusts all of these states.

✅ Pros

- **High accuracy**: IMU helps LiDAR during motion distortion; LiDAR helps constrain IMU drift.
- Naturally handles **IMU bias estimation**.
- Works well under **feature-poor or high-dynamic motion**.

❌ Cons

- **Heavier computation** (needs nonlinear optimization).
- More complex to implement and tune.
- Sensitive to **gross sensor errors** (as you saw — bad LiDAR rotations can hurt bias).

📚 Examples

- **LIO-SAM**, **LIO-Mapping**, **VINS-Fusion**, **LINS**, **FAST-LIO 2**, etc.

---

3. Hybrid / semi-tightly coupled

Some systems run **loosely coupled real-time odometry**, but periodically perform a **tightly coupled batch optimization** (e.g., loop closure refinement).
This gives a trade-off between robustness and computation.

Subscription charge failed. Please update your payment method.

Architecture	EKF / filter	Factor graph / nonlinear optimization
Bias Estimation	Approximate	Accurate
Accuracy	Moderate	High
Complexity	Low	High
Robustness to bad LiDAR	Higher (less sensitive)	Lower (can corrupt bias)
Real-time feasibility	Easy	Needs optimization tricks

Would you like me to sketch (with a simple diagram) how the **factor graph looks** in the tightly coupled case (like in LIO-SAM)? It helps visualize the IMU and LiDAR constraints clearly.

in generale in slam we cannot deduce the position of imu but we can deduce the rotation at high precision

✅ Excellent observation — and yes, you've touched on a **fundamental truth** about IMU-based estimation and SLAM.
Let's go step by step and clarify **why** this happens, what it means physically, and how SLAM systems use that property.

⚙️ 1. What an IMU actually measures

An **IMU** provides:

- **Gyroscope** → measures *angular velocity* ω

Subscription charge failed. Please update your payment method.

- Double-integrating accelerometer data gives **position** (translation).

But they behave *very differently* in practice.

🧭 2. Rotation estimation — *precise and drift-resistant*

- **Gyroscopes are very stable**: their bias drift is typically small and slow.
- Integrating angular velocity once → orientation error grows *linearly* with time.

Example:
If your gyro has 0.01°/s bias → after 10 s, orientation error ≈ 0.1°, which is very small.

So orientation from IMU is:

🔸 *High rate*, *smooth*, *low noise*, and *drift grows slowly.*

That's why in SLAM, IMU orientation is trusted strongly — it's great for **motion distortion correction**, **scan deskewing**, and **initial attitude** estimation.

🛰️ 3. Position estimation — *imprecise and drift-prone*

Accelerometers are much noisier and more bias-sensitive:

- A tiny constant bias (e.g. 0.01 m/s²) integrates to huge position errors:

$$\text{Error} = \frac{1}{2} b_a t^2$$

  → after 10 s, ≈ 0.5 m drift!
- Integration also requires **gravity compensation**, which depends on correct orientation — so any small rotation error pollutes the linear acceleration estimate.

So position from pure IMU integration quickly becomes unreliable.
In short:

🔸 *Translational drift grows quadratically* with time.
🔸 *Rotation error feeds into position drift.*

Subscription charge failed. Please update your payment method.

But SLAM systems exploit the **strong orientation estimate** to:

- **Deskew LiDAR scans** (remove motion distortion between points),
- **Predict pose between LiDAR frames**,
- **Constrain roll/pitch** when LiDAR features are weak,
- **Bridge** short gaps where LiDAR loses track.

Meanwhile, LiDAR (or vision) provides **drift-corrected translation** and long-term constraints.

So, typically:

- **IMU** → **fast, short-term orientation + relative motion**
- **LiDAR/vision** → **absolute position correction**

⚖️ 5. Key takeaway

Quantity	From IMU	Quality	Drift growth
Rotation	Integrate gyro	Accurate (low bias)	Linear
Position	Double integrate accel	Poor (bias sensitive)	Quadratic
Velocity	Single integrate accel	Moderate	Linear

👉 Therefore:

> In SLAM, **we rely on IMU for orientation** and short-term motion propagation, but **we must fuse it with LiDAR/vision** to constrain translation.

If you'd like, I can show how this principle appears *mathematically* in the **IMU preintegration equations** used in LIO-SAM — you'll see clearly why rotation is well-constrained but position drifts without external correction. Would you like that derivation?

Subscription charge failed. Please update your payment method.

awesome — here's a compact, math-first view of IMU preintegration and why rotation is tight while position drifts.

1) Continuous-time IMU model (body frame)

State $x = \{R,\, v,\, p,\, b_g,\, b_a\}$:

- $R \in SO(3)$ (world ← body)
- $v, p \in \mathbb{R}^3$
- gyro/accel biases b_g, b_a

Measurements:

$$\begin{aligned} \omega_m &= \omega + b_g + n_g, \\ a_m &= R^\top (a - g) + b_a + n_a, \end{aligned}$$

with white noises n_g, n_a and gravity g.

Dynamics:

$$\begin{aligned} \dot{R} &= R\,\widehat{\omega}, \\ \dot{v} &= a + g, \\ \dot{p} &= v, \\ \dot{b}_g &= n_{wg}, \quad \dot{b}_a = n_{wa}. \end{aligned}$$

2) Discrete propagation (between keyframes $i \to j$)

Using the IMU in $[t_i, t_j]$, we define **preintegrated** terms (Forster et al.) that remove explicit dependence on global R_i, p_i, v_i:

$$\boxed{\begin{aligned} \Delta\tilde{R}_{ij} &\approx \prod_{k=i}^{j-1} \exp\big((\omega_{m,k} - b_g)\,\Delta t\big), \\ \Delta\tilde{v}_{ij} &\approx \sum_{k=i}^{j-1} R_k\,(a_{m,k} - b_a)\,\Delta t, \\ \Delta\tilde{p}_{ij} &\approx \sum_{k=i}^{j-1} \Big[\Delta\tilde{v}_{ik}\,\Delta t + \tfrac{1}{2} R_k\,(a_{m,k} - b_a)\,\Delta t^2\Big]. \end{aligned}}$$

Subscription charge failed. Please update your payment method.

$$p_j \approx p_i + v_i\Delta t + \tfrac{1}{2}g\Delta t^2 + R_i\big(\Delta\tilde{p}_{ij} + \mathbf{J}^p_{bg}\delta b_g + \mathbf{J}^p_{ba}\delta b_a\big).$$

Here $\delta b_g, \delta b_a$ are first-order bias corrections and the $\mathbf{J}$ terms are precomputed Jacobians.

3) Factor residuals (used in LIO-SAM/GTSAM)

Given variables $(R_i, v_i, p_i, b_{g,i}, b_{a,i}, R_j, v_j, p_j, b_{g,j}, b_{a,j})$, the IMU factor penalizes:

$$\begin{aligned} r_R &= \backslash\mathrm{Log}\Big(\Delta\tilde{R}_{ij}^\top R_i^\top R_j\Big), \\ r_v &= R_i^\top(v_j - v_i - g\Delta t) - \Big(\Delta\tilde{v}_{ij} + \mathbf{J}^v_{bg}\delta b_g + \mathbf{J}^v_{ba}\delta b_a\Big), \\ r_p &= R_i^\top\Big(p_j - p_i - v_i\Delta t - \tfrac{1}{2}g\Delta t^2\Big) - \Big(\Delta\tilde{p}_{ij} + \mathbf{J}^p_{bg}\delta b_g + \mathbf{J}^p_{ba}\delta b_a\Big), \\ r_{bg} &= b_{g,j} - b_{g,i}, \qquad r_{ba} = b_{a,j} - b_{a,i}. \end{aligned}$$

Each residual is weighted by the **preintegrated covariance** Σ_{ij} accumulated from IMU noise.

4) Why rotation is precise but position drifts

4.1 Error-state intuition

Linearize with small errors $\delta\theta, \delta v, \delta p$. The continuous error covariances (rough scalings) evolve as:

- **Orientation** (gyro noise spectral density σ_g^2):

  $$\mathrm{Var}[\delta\theta](t) \;\sim\; \sigma_g^2\, t \quad\Rightarrow\quad \|\delta\theta\| \sim O(\sqrt{t}).$$

  Single integration of noise → **slow growth**.
- **Velocity** (accel noise σ_a^2 and orientation error coupling):

  $$\mathrm{Var}[\delta v](t) \;\sim\; \sigma_a^2\, t \;+\; \|g\|^2\,\mathrm{Var}[\delta\theta]\, t \;\Rightarrow\; \|\delta v\| \sim O(\sqrt{t}) + O(t).$$
- **Position** (double integration of accel noise + gravity misprojection):

  $$\mathrm{Var}[\delta p](t) \;\sim\; \sigma_a^2\, t^3 \;+\; \|g\|^2\,\mathrm{Var}[\delta\theta]\, t^3 \;\Rightarrow\; \|\delta p\| \sim O(t^{3/2}) + O(t^2).$$

Key point: position integrates acceleration **twice** and is also polluted by any tiny attitude error (since gravity must be subtracted in the right direction). Hence, translation uncertainty explodes much faster.

Subscription charge failed. Please update your payment method.

estimates.

- Accel bias b_a is harder: it couples with gravity and translation; you need **rich motion** (tilts/accelerations) or exteroceptive constraints (LiDAR/vision) to pin it down. Poor excitation ⇒ translation drifts.

5) What the graph “trusts” in practice

- The rotation residual r_R is driven mostly by integrated gyro with small bias; its covariance grows slowly ⇒ **tight rotational constraints**.
- The position residual r_p mixes gravity, bias, and double integration ⇒ **looser translational constraints** unless you fuse LiDAR/vision (as LIO-SAM does) to anchor translation and help estimate b_a.

6) Practical takeaways for LIO-style systems

- Use IMU to **deskew** and provide **high-rate attitude** — this is where it shines.
- Let LiDAR/vision (plus loop closure/GNSS if available) **constrain translation** and assist in estimating b_a.
- Ensure **motion excitation** (some tilting/turning) to keep b_g, b_a observable and reduce coupling with gravity.

If you want, I can also show a tiny pseudocode block for computing $(\Delta\tilde{R}, \Delta\tilde{v}, \Delta\tilde{p})$ and their Jacobians the “Forster way,” like what’s under the hood of `imuPreintegration.cpp`.