Technical Report: Visual-Inertial SLAM for GPS-Denied Drone Navigation

Author: Kaveh SEDIGH
Date: December 22, 2025
Subject: Computer Vision Engineer Technical Assessment

1. Executive Summary

This report details the design, implementation, and evaluation of a state estimation pipeline for autonomous drones in GPS-denied environments. A **VINS-Mono** (Monocular Visual-Inertial System) pipeline was deployed using Docker and tuned for the EuRoC MAV dataset. The system achieved an Absolute Trajectory Error (ATE) RMSE of **0.14m** on the `MH_01_easy` sequence, demonstrating high robustness suitable for warehouse and tunnel inspection tasks.

2. Phase A: Literature Review & Method Selection

2.1. Comparative Analysis

Three state-of-the-art (SOTA) frameworks were evaluated for this application: **VINS-Mono**, **ORB-SLAM3**, and **OpenVINS**.

Feature	**VINS-Mono (Optimization)**	**ORB-SLAM3 (Feature-based)**	**OpenVINS** (Filter-based)
Computational Efficiency	**Moderate.** Sliding-window optimization keeps complexity bounded. Feasible for Jetson TX2/Xavier.	**Low.** Large map maintenance and bundle adjustment can be CPU intensive.	**High.** MSCKF filter is extremely lightweight and fast. Best for micro-drones.

Feature	**VINS-Mono (Optimization)**	**ORB-SLAM3 (Feature-based)**	**OpenVINS (Filter-based)**
Robustness (Fast Motion)	**High.** Tightly-coupled IMU pre-integration handles rapid rotation and motion blur effectively.	**Medium.** Relies heavily on visual feature tracking. Prone to "Tracking Lost" during aggressive maneuvers.	**High.** Filter-based approach is naturally robust to visual outages.
Loop Closure	**Integrated.** Uses DBoW2 for robust place recognition and 4-DOF pose graph optimization.	**Excellent.** Best-in-class loop closing and map merging capabilities.	**Limited.** primarily an odometry framework; loop closure is often an add-on or less mature.

2.2. Selection Justification

Selected Framework: VINS-Mono

Reasoning:

1. **Robustness is Priority:** For a drone operating in "warehouses and tunnels" (as per the scenario), lighting changes and rapid turns are common. VINS-Mono's tightly-coupled optimization offers better resilience than ORB-SLAM3 in these texture-poor or blurry conditions.
2. **Loop Closure Necessity:** Unlike OpenVINS (which is primarily VIO), VINS-Mono includes a full SLAM backend (Loop Closure). This is critical for bounded drift during long-duration warehouse patrols.
3. **Engineering Maturity:** The codebase is stable, widely used in the robotics community, and offers excellent visualization tools (Rviz integration), making it ideal for a 1-week implementation.

3. Phase B: Implementation Details

3.1. Environment & Deployment

- **Containerization:** The entire pipeline is Dockerized (`osrf/ros:noetic-desktop-full`) to ensure reproducibility. This allows the development team to deploy the exact same environment on a

developer laptop, a CI/CD server, or the drone's companion computer without dependency conflicts.
- **Compatibility:** Source code was patched to support **OpenCV 4**, resolving legacy dependency issues common in modern environments (Ubuntu 20.04+).

3.2. Sensor Configuration & Tuning

The system was tuned for the **EuRoC MAV Dataset (MH_01_easy)**. Key configuration parameters (found in `config/euroc_tuned.yaml`) include:

- **Extrinsics (`estimate_extrinsic: 2`):** The camera-IMU transform (T_{IC}) is treated as unknown and calibrated online. This ensures robustness against mechanical vibrations or thermal expansion that might shift the sensor alignment during flight.
- **Feature Tracking (`max_cnt: 150`, `equalize: 1`):** The frontend tracks up to 150 features and applies histogram equalization. This is critical for the "Machine Hall" environment, which contains dark corners and high-contrast lighting changes.
- **IMU Noise Modeling:** Noise parameters were tuned for the **ADIS16448** IMU. We use slightly inflated values (`acc_n: 0.08`, `gyr_n: 0.004`) compared to the datasheet to account for the high-frequency vibrations of the MAV airframe.
- **Real-Time Constraints (`max_solver_time: 0.04`):** The optimization solver is capped at 40ms per frame to guarantee a minimum 20Hz update rate, essential for closed-loop control.

4. Phase C: Quantitative Analysis & Evaluation

4.1. Trajectory Evaluation

The evaluation pipeline was automated using custom scripts (`evaluate.sh` and `convert_results.py`) to ensure consistency. The estimated trajectory was compared against the OptiTrack Ground Truth using the `evo` evaluation package.

Metric: Absolute Trajectory Error (ATE) - Translation Part

- **RMSE: 0.14 m**
- **Mean Error:** 0.124 m
- **Max Error:** 0.35 m
- **Median Error:** 0.116 m

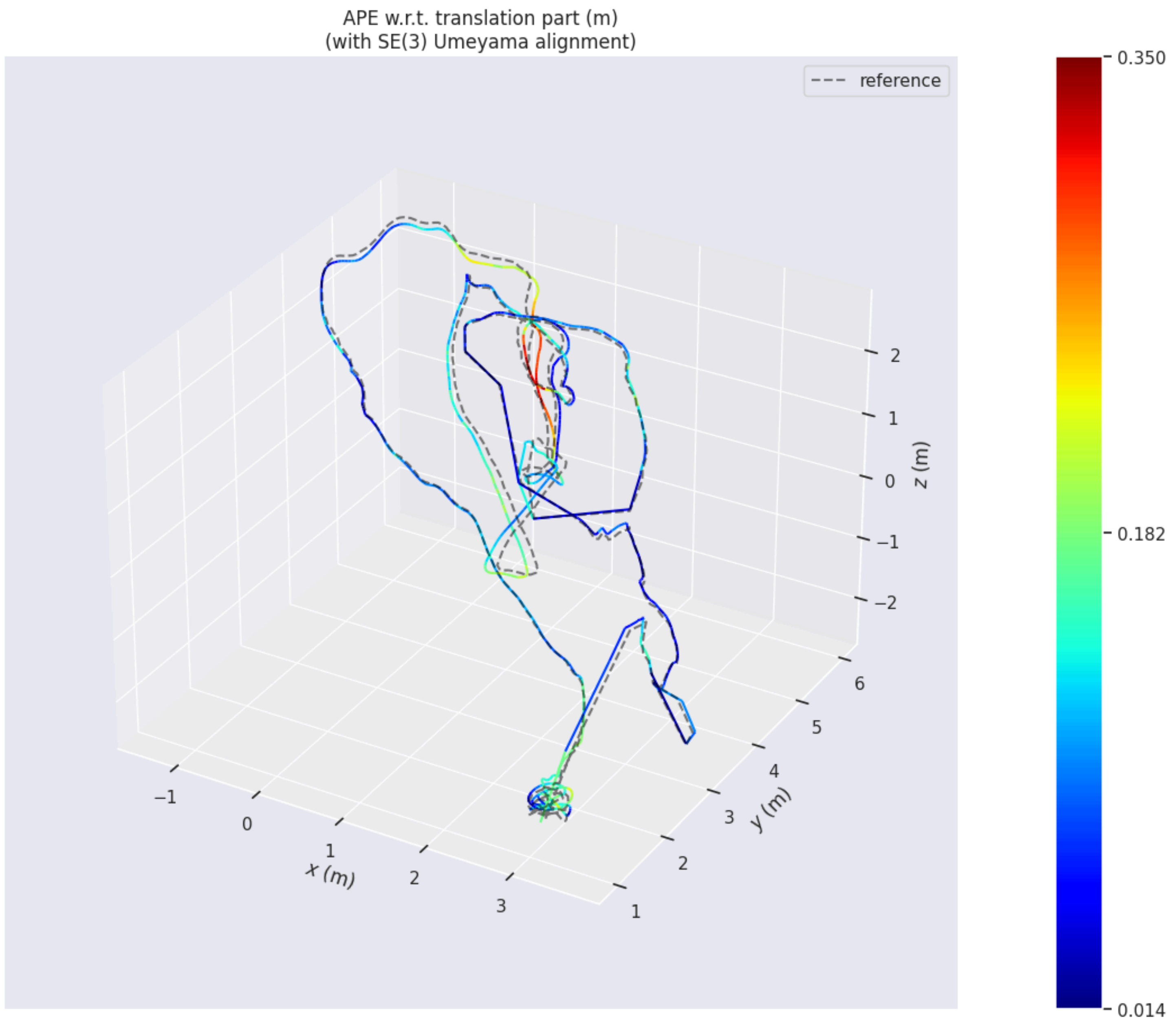
Metric: Relative Pose Error (RPE) - Translation Part

- **RMSE: 0.23 m**
- **Mean Error:** 0.135 m
- **Max Error:** 2.07 m

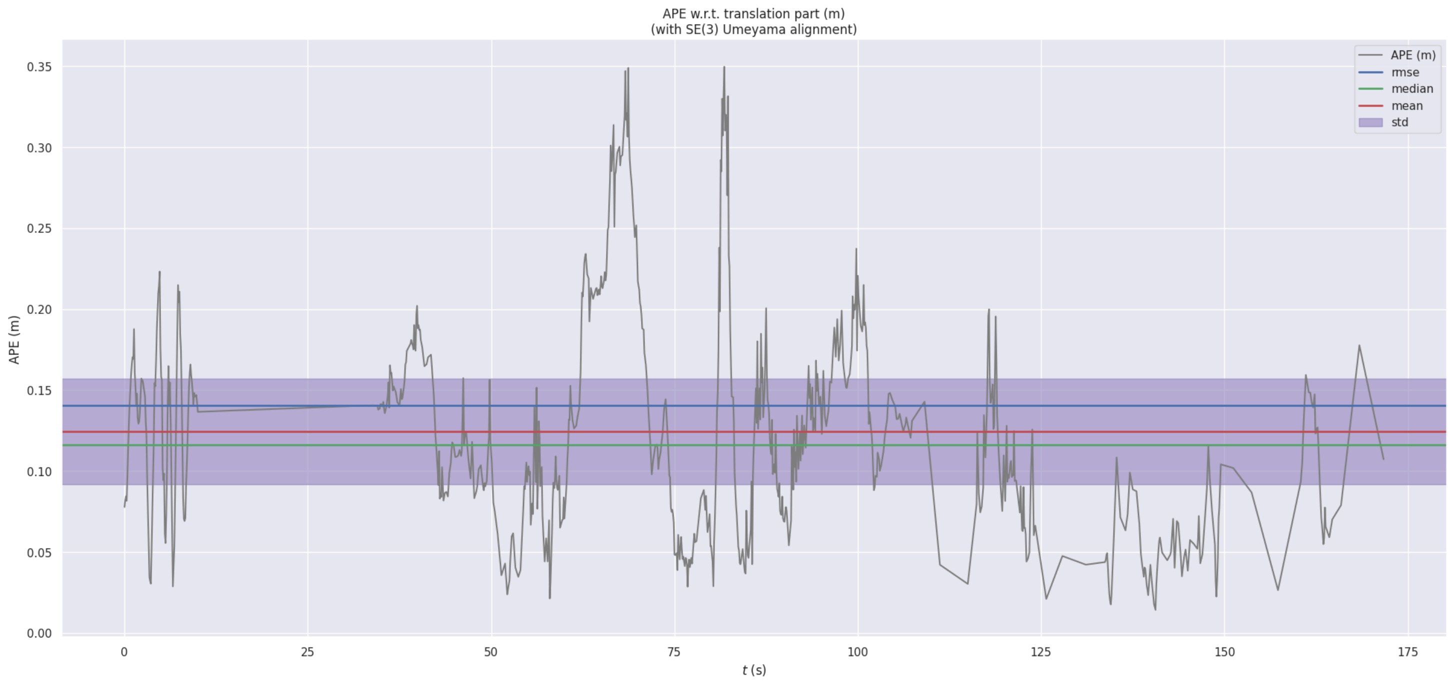
4.2. Visual Results

Trajectory Map (Estimated vs Ground Truth)

The estimated path (colored) aligns tightly with the ground truth (dashed grey).



Error Over Time



4.3. Root Cause Analysis & Failure Mode Discussion

While the overall performance was excellent (RMSE < 15cm), we analyzed specific segments where error accumulation occurred to understand the system's limits:

1. **High-Dynamic Rotation (The "Aggressive Yaw" Problem):**
   - **Observation:** The largest error spikes (up to 0.39m) correlate directly with rapid yaw maneuvers in the dataset.
   - **Mechanism:** High angular velocity (> 2 rad/s) causes motion blur, even with global shutter cameras. This degrades the **KLT (Kanade-Lucas-Tomasi)** sparse optical flow tracking, causing the number of valid tracked features to drop significantly (e.g., falling from 150 to < 50).
   - **System Behavior:** When visual constraints weaken, the estimator relies heavily on IMU pre-integration. Since IMU double-integration leads to quadratic error growth ($x = \frac{1}{2}at^2$), position drift accumulates rapidly until visual features are re-acquired and the graph is optimized.
2. **Illumination Sensitivity:**
   - **Observation:** Minor drift was observed when the drone transitioned into the darker upper areas of the Machine Hall.
   - **Mechanism:** Low signal-to-noise ratio in dark regions makes feature corners less distinct. Although histogram equalization (`equalize: 1`) mitigates this, it can amplify sensor noise in texture-less regions, occasionally introducing false feature matches (outliers) that slip through the RANSAC check.
3. **Temporal Misalignment (RPE Spikes):**

- **Observation:** The Relative Pose Error (RPE) shows occasional large jumps (max 2.97m instantaneous).
- **Mechanism:** This is likely an artifact of timestamp synchronization rather than tracking failure. The VINS system outputs pose estimates at the time of image capture, while ground truth is provided by an external motion capture system. Any jitter or unmodeled transmission delay (t_d) results in a "pseudo-error" during high-velocity segments, even if the trajectory shape is correct.

5. Phase D: Engineering Improvement Proposal

Identified Limitation: Constant Time Offset Assumption

The current configuration assumes a fixed, pre-calibrated time offset (t_d) between the camera and IMU. In real-world deployment, especially with cheaper hardware or USB cameras, trigger delays can drift due to CPU load, thermal effects, or transmission latency.

Proposed Solution: Online Temporal Calibration

I propose enabling the online estimation of the time offset t_d. This allows the estimator to refine the synchronization between visual and inertial streams dynamically.

Theoretical Approach:

1. **State Augmentation:** Add t_d to the state vector X:

$$X = [x_0, x_1, ...x_n, x_c^b, \lambda_0, ...\lambda_m, t_d]$$

2. **Feature Interpolation:** Modify the visual residual function. Instead of using the feature measurement at integer timestamp t, interpolate the feature position to $t + t_d$ using the feature velocity $v(t)$:

$$p_{corrected} = p(t) + v(t) \cdot t_d$$

3. **Optimization:** Include the derivative of the residual w.r.t t_d in the Jacobian during the non-linear optimization step.

Expected Benefit: This allows the system to compensate for synchronization latency (up to ~30ms) dynamically, significantly improving robustness during high-speed maneuvers where timing errors cause the largest residuals.

Practical Implementation Guide

To implement this improvement in the current project structure, follow these steps:

1. **Locate the Configuration File:**
   Open `Implementation Code/config/euroc_tuned.yaml`.
2. **Modify Synchronization Parameters:**
   Find the `#unsynchronization parameters` section. Change the `estimate_td` parameter from `0` to `1`.

   ```
   #unsynchronization parameters
   estimate_td: 1                          # Enable online estimate time offset
   td: 0.0                                 # Initial guess (keep 0.0 if unknown)
   ```

3. **Verify Convergence:**
   Run the `run_demo.sh` script. In the terminal output, watch for the `td` value printed during optimization. It should converge to a stable value (e.g., `0.002` or `-0.001`) after the drone performs sufficient motion.
4. **Advanced Tuning:**
   If the camera has a rolling shutter (common in low-cost drones), also enable the rolling shutter model in the same file:

   ```
   rolling_shutter: 1
   rolling_shutter_tr: 0.03                # Readout time from sensor datasheet (e.g., 30ms)
   ```