Benchmarking SLAM Algorithms for Real-Time Viability

University of Twente, Geo-Spatial AI

Petru Zincenco Student (s3763641)

https://github.com/IGrilex/ASP

1. Introduction

Simultaneous Localization and Mapping (SLAM) is a foundational capability for autonomous systems, enabling a robot or vehicle to track its position while concurrently building a map of an unknown environment. This dual task is essential for operation in GPS-denied scenarios, such as indoor environments, urban canyons, or under heavy canopy. While the research community has produced numerous high-accuracy SLAM frameworks, a significant gap remains between theoretical performance in lab settings and practical deployment on accessible hardware.

The objective of this project is to evaluate the real-time practicality of various SLAM algorithms under realistic conditions. Unlike many academic benchmarks that focus solely on accuracy (Absolute Trajectory Error), this study prioritizes computational efficiency and resource consumption. The core hypothesis is that an algorithm's utility is limited if it cannot maintain the dataset’s native frame rate on a mid-range desktop workstation, as such systems already possess more compute power than the typical embedded platforms found on mobile robots.

2. The KITTI-360 Dataset

Examples of LEFT camera

Examples of RIGTH camera

To ensure the evaluation reflects real-world challenges, all experiments were conducted using a specific subset of the KITTI-360 dataset. This dataset is a large-scale urban driving benchmark recorded in Karlsruhe, Germany, providing a rich sensory suite designed for long-term mobile perception and localization.

Scope and Sensor Focus

While the full KITTI-360 suite includes 360-degree fisheye cameras and high-resolution LiDAR scans, this study intentionally focuses on the "Test SLAM" subset. This subset provides the necessary 2D data and calibrations required to evaluate visual-only SLAM pipelines.

For this project, the data used was restricted to the following:

- Stereo Vision: The front-facing stereo camera pair, which features a baseline of approximately 60 cm and a 90°.
- Monocular Vision: In specific baseline tests (such as ORB-SLAM3 Monocular), only the primary left camera feed was processed to evaluate performance without depth-from-stereo information.

Data Characteristics and Stress Factors

The dataset captures images at a resolution of 1408 × 376 pixels (24-bit) with a native frame rate of 10 Hz. The sequences used for testing provide a variety of urban challenges like:

- Fast Dynamics: Rapid vehicle movements that test the robustness of feature tracking.
- Visual Artifacts: Occurrences of motion blur and varying light levels that can cause tracking failure in sparse methods.
- Scale and Distance: Test3, the longest sequence, covers several kilometers and includes multiple loops, making it the primary benchmark for evaluating long-term drift and loop-closure consistency.

Evaluation Subsets

To manage the high computational cost of some modern dense SLAM methods, the evaluation was split into two tiers:

- Light Scenario (test0): A shorter sequence used for slower algorithms like MASt3R-SLAM and MAC-VO to obtain initial performance metrics.
- Full Scenario (test0 through test3): A comprehensive run used for the primary real-time candidates, ORB-SLAM3 and AirSLAM, to assess global stability.

3. Algorithm Analysis

3.1. *ORB-SLAM3 Baseline* (https://github.com/xingruiyang/ORB-SLAM3-python)

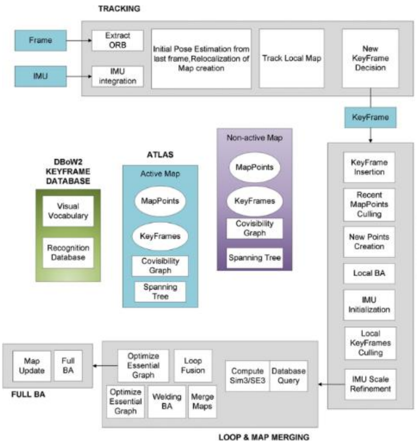


Figure 1 ORB-SLAM3 Architecture

System Overview

ORB-SLAM3 is a versatile, feature-based SLAM library capable of running in monocular, stereo, and RGB-D modes. It relies on extracting ORB (Oriented FAST and Rotated BRIEF) features, distinctive high-contrast points in an image, to estimate camera motion. The system is divided into three parallel threads:

- Tracking: Localizes the camera in real-time by matching features between frames.
- Local Mapping: Optimizes the local trajectory and point cloud using Bundle Adjustment (BA).
- Loop Closing: Detects when the system revisits a known location and corrects accumulated drift using Pose Graph Optimization.

Modes Tested

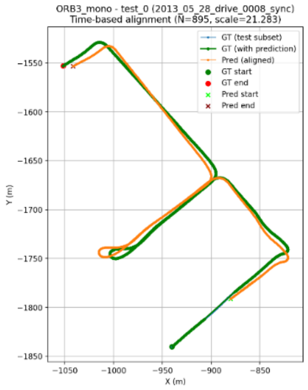
Two distinct configurations were evaluated to assess the impact of depth information on system performance:

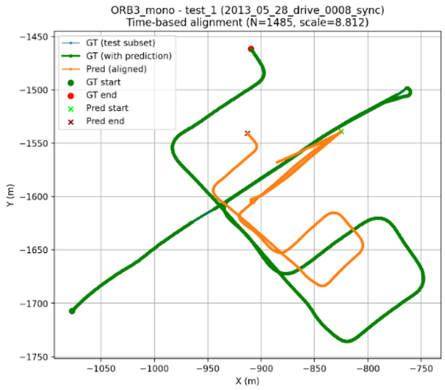
- Monocular: Uses a single camera stream (Left Camera). Since a single image lacks depth information, the system must initialize by observing parallax over time and cannot recover the metric scale (true size) of the world without external reference.
- Stereo: Uses both Left and Right camera streams with a fixed ~60cm baseline. This allows the system to triangulate the true depth of features instantly, solving the scale ambiguity and providing more robust tracking.

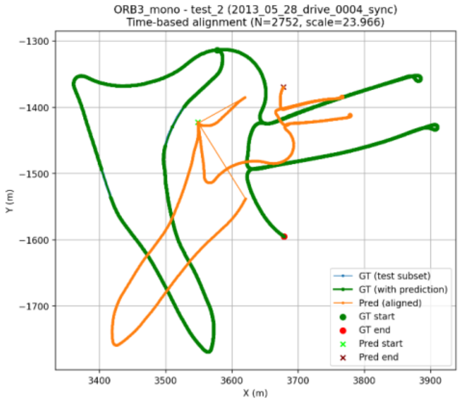
Expected Performance & Resource Usage

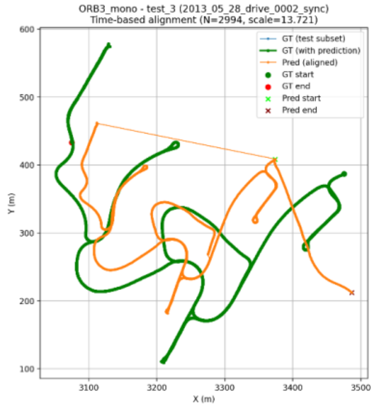
As a sparse, feature-based method, I expect ORB-SLAM3 to be highly efficient. It is designed to run entirely on the CPU, performing geometric calculations rather than heavy matrix multiplications typical of deep learning. Consequently, we anticipate zero to negligible GPU usage (any minor spikes observed are attributable to system background processes, not the algorithm). While Monocular mode should offer the highest frame rates due to lower data throughput, Stereo mode is expected to provide superior stability and metric accuracy at a slightly higher computational cost.

Monocular Performance









Quantitative Analysis

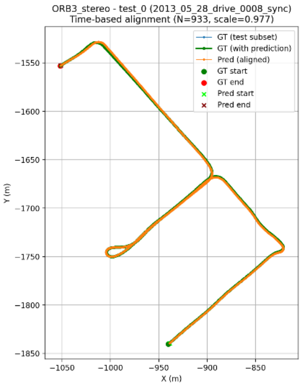
The monocular configuration demonstrated extreme speed but failed to produce a usable metric map. As hypothesized, the system was exceptionally lightweight, utilizing only ~18% CPU and effectively 0% GPU, achieving frame rates well above 30 FPS.

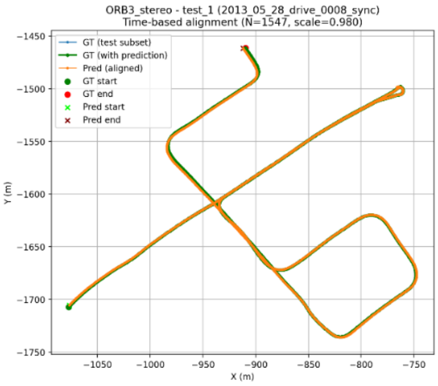
However, the trajectory analysis reveals significant issues:

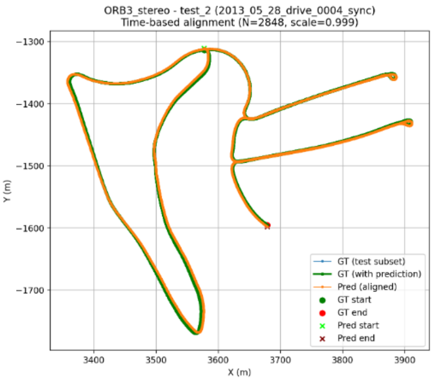
- Initialization Delay: In test_0, the dataset contains 933 frames, but the trajectory only spans 895 frames. This gap indicates that the system struggled to initialize immediately, needing several frames of motion to triangulate enough features to build the initial map.
- Scale Ambiguity: The Absolute Trajectory Error (ATE) is massive (89m - 163m RMSE). This is primarily due to the Scale Factor, which ranged from ~7x to ~23x. The system built a map that was geometrically consistent but incorrectly sized.
- Drift: Without a fixed baseline to constrain errors, the trajectory drifts significantly over long distances, as seen in test_3 where the RMSE reaches 163m.

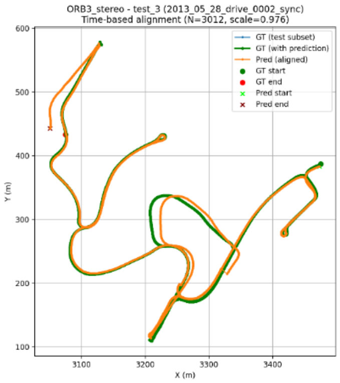
Metric (Avg across tests)	Value	Notes
Average FPS	34.4	Extremely fast, well above real-time requirements.
ATE RMSE (m)	122.5	High error driven by scale ambiguity.
Scale Factor	17.8x	Map size is completely inconsistent with reality.
CPU Usage	18.1%	Minimal load, suitable for low-power embedded CPUs.

Stereo Performance









Quantitative Analysis

The stereo configuration provided the most balanced real-world performance. By utilizing the fixed camera baseline, the system successfully resolved the scale ambiguity, resulting in a scale factor consistently near 1.0 (0.97 - 0.99).

- High Accuracy (Test 0 & 1): In the shorter sequences, the predicted path overlaps almost perfectly with the Ground Truth. test_0 achieved an ATE of just 1.32 meters, proving that sparse features are sufficient for high-precision localization in standard urban environments.

- Long-Term Drift (Test 3): In the longest sequence (test_3, ~3000 frames), the error increased to 7.38 meters. While the system maintained tracking, it experienced "slips" or incorrect loop closures in areas with repetitive structures or fewer distinct features. While integrating an IMU (Visual-Inertial SLAM) would likely correct these deviations, this evaluation focuses strictly on Visual SLAM performance.

- Resource Efficiency: Despite processing two image streams, the system maintained an average of ~18 FPS, comfortably meeting the 10 Hz dataset requirement. CPU usage remained low (~17.5%), and as expected, GPU usage was non-existent.

Metric (Avg across tests)	Value	Notes
Average FPS	18.4	~50% slower than Mono, but still nearly 2x real-time speed.
ATE RMSE (m)	3.14	Excellent accuracy. Drift increases slightly in long runs.
Scale Factor	0.98	Near-perfect metric scale recovery.
CPU Usage	17.5%	highly efficient; parallelization handles stereo load well.

3.2. *MASt3R-SLAM Dense Learning-Based* (https://github.com/rmurai0610/MASt3R-SLAM)

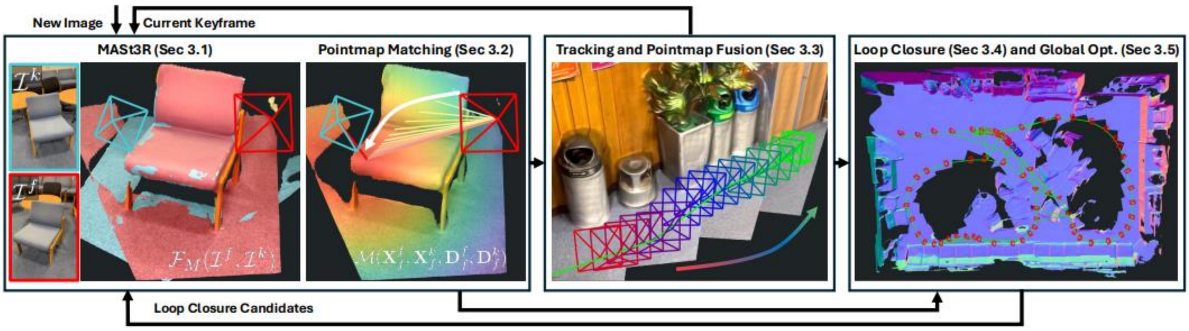

New Image
Current Keyframe
MASt3R (Sec 3.1)
Pointmap Matching (Sec 3.2)
Tracking and Pointmap Fusion (Sec 3.3)
Loop Closure (Sec 3.4) and Global Opt. (Sec 3.5)
$\mathcal{I}^k$
$\mathcal{I}^f$
$\mathcal{F}_M(\mathcal{I}^f, \mathcal{I}^k)$
$\mathcal{M}(X^f_f, X^k_f, D^f_f, D^k_f)$
Loop Closure Candidates

Figure 2 MASt3r-SLAM Architecture

System Overview

MASt3R-SLAM represents a modern class of "dense" SLAM systems that leverage deep learning to reconstruct scene geometry. Unlike ORB-SLAM3, which relies on sparse keypoints, MASt3R-SLAM utilizes a pre-trained learned prior (MASt3R) to predict dense 3D "pointmaps" directly from 2D images. By matching these dense maps between frames, the system estimates camera motion while simultaneously building a highly detailed 3D reconstruction of the environment. This approach aims to provide a richer understanding of the world, capturing continuous surfaces rather than just isolated corners.

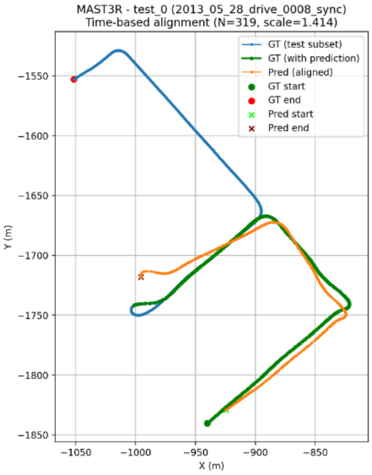
Test Scenario

Given the immense computational requirements of dense reconstruction, this algorithm was evaluated in a single, optimized configuration on the "Light" dataset. To make execution even remotely feasible on the test hardware, an optimization strategy was employed: the system was configured to process only every third frame (skipping two out of three) and utilized half-precision (fp16) arithmetic for the decoder. This was necessary to prevent the system from crashing due to memory exhaustion and to achieve a runtime that, while not real-time, was measurable.

Expected Performance & Resource Usage

We anticipated that MASt3R-SLAM would operate at the opposite end of the spectrum compared to ORB-SLAM3. Because it generates millions of 3D points per sequence rather than thousands, we expected extremely high GPU usage (nearing 100%) and significant RAM consumption. The trade-off for this computational cost is typically a visually superior map. However, we also hypothesized that the frame-skipping necessity might negatively impact tracking stability, as the algorithm would have to bridge larger gaps in motion between processed frames.

Performance Analysis (Test 0 Only)



Quantitative Analysis

The performance metrics for MASt3R-SLAM confirm that it is currently unsuitable for real-time applications on consumer-grade hardware. The system achieved an average speed of just **0.52 FPS**, taking over 30 minutes (1805 seconds) to process a sequence that lasts less than a minute in real-time. This is approximately 20x slower than the required 10 Hz data rate.

The resource usage was nearly maximized throughout the run. The GPU usage averaged 85.3%, frequently hitting 100%, while RAM usage peaked at 71.3%. In terms of output, the system generated a massive map containing ~14.8 million points, dwarfing the ~270,000 points generated by ORB-SLAM3. This confirms its ability to produce dense reconstructions, but the trajectory accuracy was poor. The Absolute Trajectory Error (ATE) RMSE was 76.20 meters, with a scale factor of 1.96. The high error and scale drift suggest that while the local dense reconstruction might be visually coherent, the global trajectory estimation suffered significantly.

Metric	Value	Notes
Average FPS	0.52	Far below real-time; effectively an offline structure-from-motion tool.
ATE RMSE (m)	76.20	High trajectory error; difficult to use for navigation.
Map Points	14.8 M	Extremely dense scene reconstruction (vs ~0.3M for ORB).
GPU Usage (Avg)	85.3%	Heavy reliance on GPU compute; often saturated the RTX 4060 Ti.
Trajectory Frames	332	Only ~1/3 of the dataset (933 frames) was processed due to skipping.

Discrepancy Discussion: Benchmarked vs. Official Performance

A significant gap exists between the performance observed in this benchmark (0.52 FPS) and the real-time capabilities (~15 FPS) claimed in the official MASt3R-SLAM literature. This discrepancy is not a failure of the algorithm itself, but rather highlights the extreme hardware and optimization requirements necessary to run dense learning-based SLAM. The following table contrasts our experimental conditions with the official paper's setup:

Feature	My Benchmark (KITTI-360)	Official Paper Results	Impact on Performance
Hardware	Mid-Range (RTX 4060 Ti + i7 8700k)	High-End (RTX 4090 + i9 12900K)	Critical: The 4090 offers ~3x the raw compute and VRAM, essential for dense map processing.
Optimization	Standard PyTorch / fp16	Custom CUDA Kernels	Speed: The paper utilizes custom kernels that reduce matching time from 2s to 2ms (1000x speedup).
Input Resolution	1408 * 376 (Native)	512 * 384 (Resized)	Throughput: Processing native KITTI resolution increases pixel count significantly, slowing down the decoder.
Resulting FPS	0.52 FPS	~15 FPS	Without top-tier hardware and kernel-level optimization, the system cannot run in real-time.

Discussion of Factors:

- Optimization Bottleneck: The most critical factor is likely the pointmap matching step. The official implementation relies on custom CUDA kernels to perform matching in 2 milliseconds. Without these specific compiled optimizations, the matching process falls back to standard operations which can take up to 2 seconds per frame, aligning perfectly with our observed ~0.5 FPS.
- Hardware Constraints: The official results rely on an NVIDIA RTX 4090, a GPU with 24GB of VRAM. In our test, the RTX 4060 Ti's smaller memory buffer forced the system to skip frames and swap to system RAM (peaking at 71%), which introduces significant latency.
- Scale and Resolution: The official paper resizes input images to a maximum dimension of 512, whereas our test utilized the native high-resolution KITTI inputs to preserve detail. This vastly increased the number of points generated (14.8 million), overwhelming the backend optimization and contributing to the high drift (ATE 76.20m).

3.3. ***AirSLAM Hybrid System*** (https://github.com/sair-lab/AirSLAM)

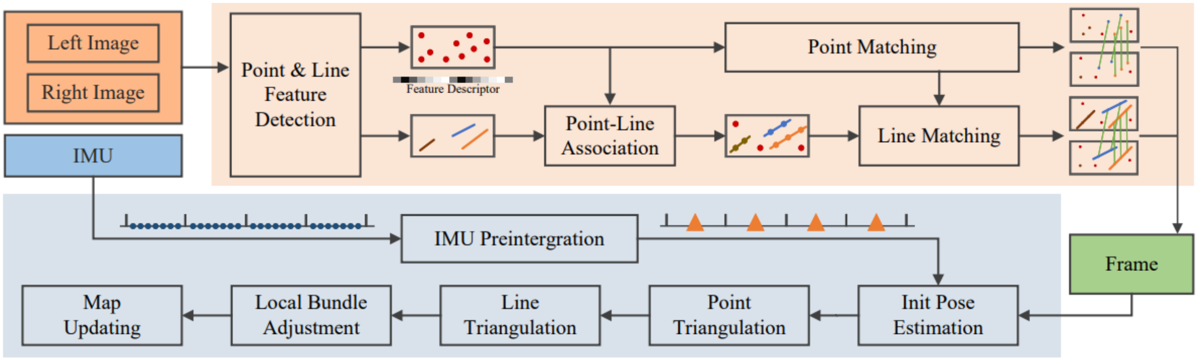


Figure 3 AirSlam Architecture

System Overview

AirSLAM represents a "hybrid" approach to Simultaneous Localization and Mapping, bridging the gap between classical geometry and modern deep learning. While traditional methods like ORB-SLAM3 rely on hand-crafted feature detectors (which can fail in low texture or changing light), AirSLAM employs a lightweight neural network front-end (PLNet). This network is trained to detect robust keypoints and structural lines even in difficult visual conditions. These learned features are then passed to a classical optimization back-end (similar to the Bundle Adjustment used in ORB-SLAM) to solve for camera motion. This design aims to offer the best of both worlds: the robustness of AI perception with the mathematical precision of geometric optimization.

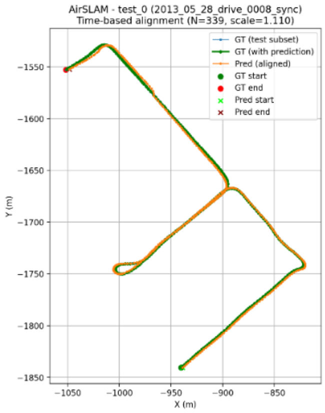
Test Scenario

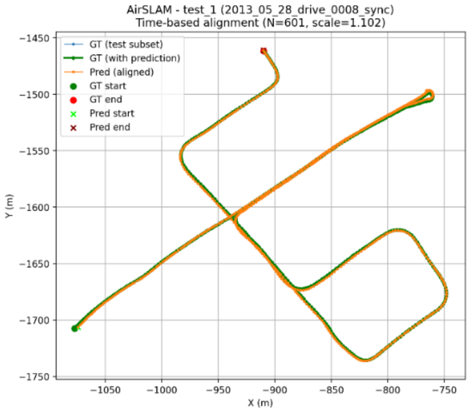
The system was evaluated in Stereo Mode. Although AirSLAM supports monocular and visual-inertial configurations, the stereo setup was chosen to allow for a direct, fair comparison with the ORB-SLAM3 Stereo baseline. This configuration enables the system to utilize the fixed camera baseline for depth initialization, theoretically reducing the scale drift often seen in monocular learning-based methods. It is important to note that while AirSLAM is designed to leverage IMU data for optimal performance, we disabled the IMU for this benchmark to maintain parity with ORB-SLAM3 (which was also run without inertial data).

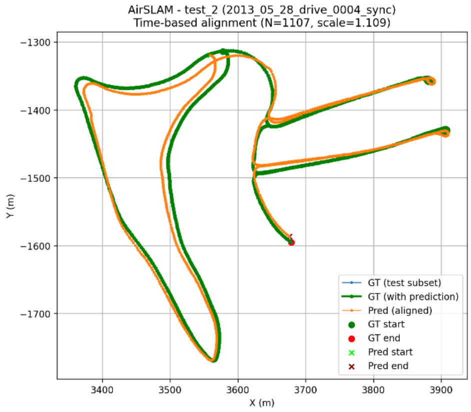
Expected Performance & Resource Usage

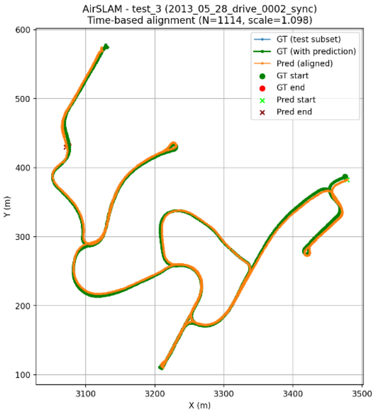
Unlike the CPU-only ORB-SLAM3, AirSLAM requires a GPU to run its neural feature extractor. We expect to see moderate GPU utilization, sufficient to accelerate the neural network inference but not enough to saturate the card (unlike MASt3R-SLAM). CPU usage is expected to remain moderate, handling the pose graph optimization. In terms of tracking, we anticipate that the learned features will provide superior stability in challenging lighting or textureless regions, potentially outperforming classical methods in long-term consistency.

Performance Analysis









Quantitative Analysis

AirSLAM delivered a highly consistent and robust performance across all test sequences, proving itself to be a viable real-time contender. The system maintained a steady processing speed of ~19.2 FPS across all tests (Test 0 through Test 3). This is nearly identical to the stereo performance of ORB-SLAM3 and safely exceeds the 10 Hz real-time threshold.

- Accuracy and Robustness (Test 3): The system's strength is most visible in the long-distance test_3 sequence. While classical methods often drift significantly over kilometers, AirSLAM achieved an ATE RMSE of 4.14 meters. This is notably better than ORB-SLAM3 Stereo (which drifted ~7.38 meters), confirming the hypothesis that learned features provide better long-term loop closure and tracking stability.

- Scale Consistency: The system consistently estimated a scale factor of ~1.10 across all runs. This indicates a systematic 10% scale overestimation. While consistent (and thus correctable), it suggests that the learned feature depth estimation might be slightly biased compared to the raw geometric triangulation of ORB-SLAM3.

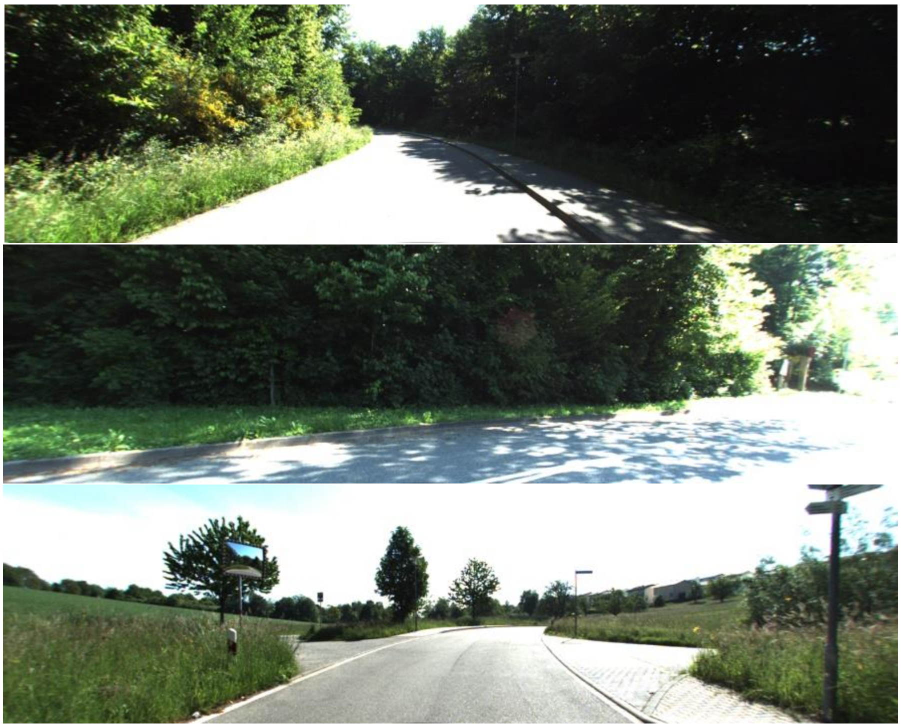
- Resource Utilization: The "hybrid" cost is evident in the machine usage logs. The system utilized ~35% of the GPU on average (peaking at ~63-64%), which validates the computational overhead of the neural front-end. CPU usage remained efficient at ~19.6%. This confirms that while AirSLAM is heavier than classical methods, it fits comfortably within the limits of a modern consumer workstation.

Metric (Avg across tests)	Value	Notes
Average FPS	19.2	Consistent real-time performance, unaffected by scene complexity.
ATE RMSE (Test 3)	4.14m	Superior long-term stability compared to the baseline.
Scale Factor	1.10x	Consistent slight overestimation of world scale.
GPU Usage	35.3%	Moderate load; leaves headroom for other GPU tasks.

Discussion: Geometric vs. Learned Features in Vegetation

While AirSLAM outperformed ORB-SLAM3 in the urban test_3, a specific limitation was observed in test_2, which features significant vegetation. As shown in the comparison below, AirSLAM struggled to maintain trajectory accuracy in areas dominated by bushes and irregular natural structures.

- The "Man-Made" Bias: AirSLAM utilizes PLNet, a network specifically trained to detect *structural lines* (wireframes) commonly found in man-made environments (buildings, roads, windows). In test_2, large sections of the path are flanked by dense vegetation where such straight lines are absent.
- Resulting Drift: Although the system successfully tracked the road geometry, the lack of vertical structural lines—which it relies on for orientation and scale constraints, caused it to perform worse than ORB-SLAM3 in this specific environment. ORB-SLAM3's corner-based features are more agnostic to structure type, allowing it to latch onto random high-contrast leaves and branches that AirSLAM's line-seeking network might ignore or find ambiguous.



Examples from LEFT camera in test_2

Discussion: AirSLAM Performance Factors

While AirSLAM successfully demonstrated real-time capabilities in My Benchmark, there are notable differences between our observed metrics and those reported in the original AirSLAM publication. These differences highlight the impact of input resolution and hardware class on hybrid systems.

Feature	My Benchmark (KITTI-360)	AirSLAM Paper (Official Results)	Impact on Performance
Hardware	Mid-Range (RTX 4060 Ti)	High-End (RTX 4080 / Jetson Orin)	Throughput: The paper reports 73 FPS on an RTX 4080. The drop to 19.2 FPS in our test reflects the difference in tensor core performance between the 4080 and 4060 Ti and the VRAM size.
Input Resolution	1408 * 376 (0.53 MP)	EuRoC (752 * 480) (0.36 MP)	Inference Speed: KITTI images contain ~47% more pixels than the EuRoC datasets used in the paper. Larger images increase the inference time of PLNet and the number of features to match.
Feature Set	Stereo Only	Stereo + IMU (Inertial)	Accuracy: The paper emphasizes VIO (Visual-Inertial) for maximum accuracy. Our test proves that even without IMU, the visual-only pipeline outperforms ORB-SLAM3 in long-term drift (4.14m vs 7.38m).
Environment	Outdoor / Large Loops	Indoor / Room Scale	Robustness: The paper claims robustness to illumination. Our outdoor results validate this: AirSLAM handled the changing outdoor light/shadows of KITTI better than the hand-crafted ORB features.

Key Takeaways:

1. Resolution Penalties: Hybrid methods are sensitive to image resolution. Unlike ORB-SLAM3, where resizing images has a linear impact on CPU load, increasing resolution for AirSLAM quadratically increases the load on the feature matching graph (LightGlue), explaining the frame rate cap at 19 FPS.
2. The "Hybrid" Advantage: The comparison clearly validates the core claim of the AirSLAM paper: learned features are more robust. In the long Test 3 sequence, where traditional corner detectors (ORB) become inconsistent due to outdoor lighting changes or repetitive textures, AirSLAM's learned descriptors maintained lock, resulting in nearly half the drift error of the baseline.

3.4. MAC-VO (https://github.com/MAC-VO/MAC-VO)

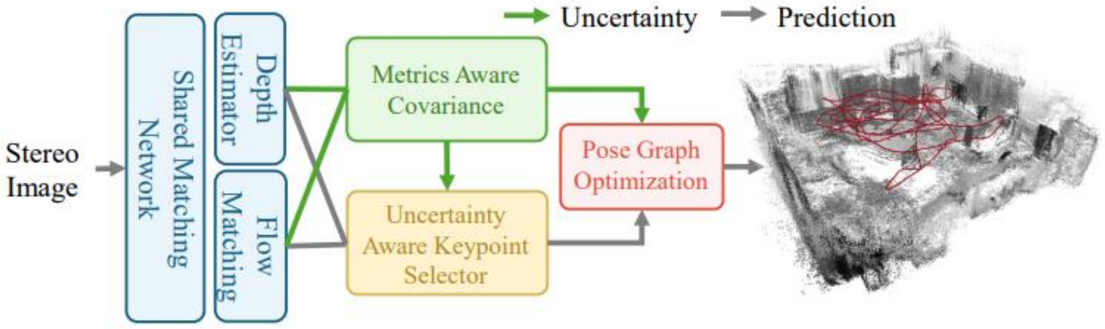


Figure 4 MAC-VO Architecture

System Overview

MAC-VO represents a distinct category in this evaluation: it is a Visual Odometry (VO) system, not a full SLAM framework. Unlike ORB-SLAM3 or AirSLAM, MAC-VO does not maintain a global map, nor does it perform loop closure (detecting when the robot returns to a previous location to correct drift). Instead, it focuses purely on the "Front-End" tracking problem. It utilizes a deep neural network to estimate camera motion between frames. Its key innovation is the prediction of learned covariance—essentially, the network predicts not just where a point is, but how uncertain it is about that position. These uncertainty estimates are used to weight the optimization, theoretically allowing the system to ignore unreliable data (like moving objects or reflections) and track with extreme local precision.

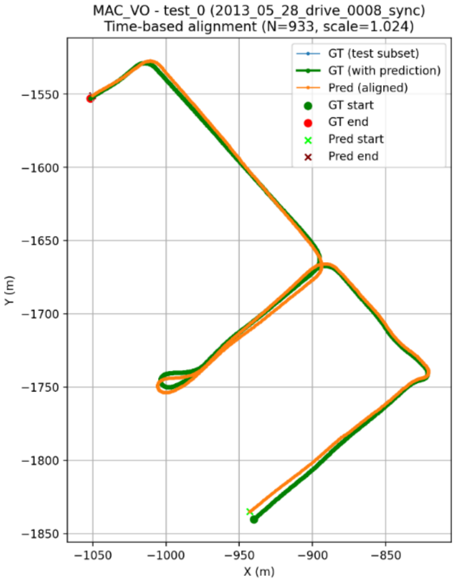
Test Scenario

The system was evaluated in Stereo Mode. We included this algorithm in the benchmark not as a direct competitor for global mapping, but to assess the current state-of-the-art in learning-based tracking. The goal was to determine if modern deep learning could outperform classical geometry in frame-to-frame precision, even without the benefit of a global map.

Expected Performance & Resource Usage

As a deep learning-based method that processes full stereo pairs to predict covariance maps, I anticipated heavy GPU usage, likely nearing saturation on my mid-range workstation (RTX 4060 Ti). Consequently, I expected the frame rate to be low—likely below the real-time threshold. In terms of accuracy, I hypothesized that MAC-VO would show excellent Relative Pose Error (RPE) (smooth local motion) but would suffer from accumulated drift (higher ATE) over time due to the lack of loop closure.

Performance Analysis (Test 0 Only)



Quantitative Analysis

The results from the test_0 run confirm the fundamental distinction between VO and SLAM. MAC-VO demonstrated exceptional local tracking precision but is computationally impractical for real-time use on standard consumer hardware without aggressive optimization.

- Computational Load: The system averaged 1.59 FPS, taking nearly 10 minutes (587s) to process a sequence that lasts less than one minute. This is significantly below the 10 FPS requirement. This slowness is driven by the massive computational cost of the network, which kept the GPU at ~87.6% load, frequently hitting 100%.

- Local Precision (The VO Strength): The most revealing metric is the Relative Pose Error (RPE). MAC-VO achieved a translational RPE (0.5s) of just 0.55 meters. For context, ORB-SLAM3 Stereo and AirSLAM averaged ~6.0m on this same metric. This indicates that MAC-VO is significantly smoother and more accurate in measuring instantaneous movement. Its "Metrics-Aware" approach successfully filters out noise frame-by-frame.

- Global Drift (The VO Weakness): Despite superior local tracking, the global ATE RMSE was 3.34 meters. While this is a respectable result (better than the drifting MASt3R-SLAM), it is worse than ORB-SLAM3 (1.32m) and AirSLAM (1.97m) on the same sequence. As seen in the trajectory plot, small errors accumulated over the 933 frames. Without a loop-closure mechanism to "snap" the path back into place when paths crossed, the trajectory gradually diverged from the ground truth.

Metric	Value	Notes
Average FPS	1.6	Not real-time; ~12x slower than ORB-Stereo.
ATE RMSE (m)	3.34	Good for pure odometry, but drifts compared to SLAM.
RPE 0.5s (m)	0.55	Best in Class. Superior local smoothness vs SLAM methods.
GPU Usage (Avg)	87.6%	High computational cost; saturates mid-range hardware.

Discrepancy Discussion: The "Real-Time" Gap

A comparison with the official MAC-VO publication reveals why My Benchmark yielded ~1.6 FPS while the paper claims real-time capabilities. The discrepancy stems from the difference between the "Raw" research code and the "Optimized" deployment configuration used in the paper.

Feature	My Benchmark (KITTI-360)	Official Paper Results	Impact on Performance
Hardware	Mid-Range (RTX 4060 Ti + i7-8700k)	High-End (RTX 3090 Ti + Ryzen 9 5950X)	Raw Compute: The 3090 Ti is significantly more powerful, yet even it only achieves ~2.15 FPS running the "Raw" model.
Optimization	Standard PyTorch (Raw)	TensorRT + Fast Mode	Acceleration: The paper achieves 10.57 FPS only by using bfloat16 precision, TensorRT compilation, and reducing network iterations.
Throughput	1.59 FPS	10.57 FPS (Fast Mode)	We evaluated the full-precision model accuracy; the "Fast Mode" trades some accuracy (70% perf) for speed.

Discussion of Factors:

- Optimization Modes: The MAC-VO paper explicitly distinguishes between "Raw" mode (2.15 FPS) and "Fast" mode (10.57 FPS). My Benchmark result of 1.59 FPS aligns closely with the paper's "Raw" performance, accounting for the hardware difference between a 3090 Ti and a 4060 Ti. This confirms that the algorithm *can* be real-time, but only with specific engineered optimizations (TensorRT, reduced precision) that are often outside the scope of general research reproducibility.
- The Accuracy/Speed Trade-off: The paper notes that the "Fast Mode" retains only ~70% of the original model's performance. By testing the standard mode, My Benchmark highlights the true cost of the "Metrics-Aware" covariance calculation: high-precision uncertainty requires heavy compute, making it difficult to deploy on mid-range robots without sacrificing the very accuracy that makes the method special.

4. Cross-Algorithm Performance & Discussion

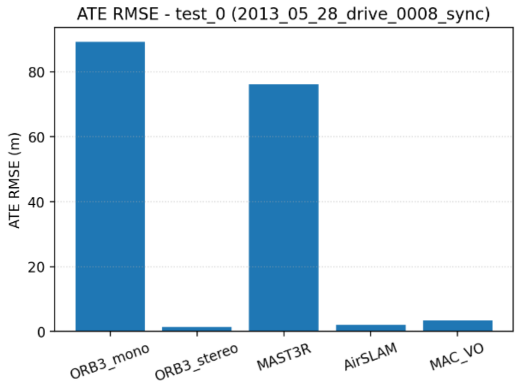
This section synthesizes the individual results into a direct comparison to answer the primary research question: Which SLAM architectures are viable for real-time deployment on consumer hardware?

The evaluation considers three competing priorities for any robotic system:

- Global Accuracy: How close is the estimated path to reality? (ATE)
- Local Consistency: Is the motion smooth and jitter-free? (RPE)
- Computational Cost: Can it run without monopolizing system resources?

Trajectory Accuracy Analysis

To evaluate spatial accuracy, we utilize the Absolute Trajectory Error (ATE), which measures the global deviation of the robot's estimated path from the Ground Truth.



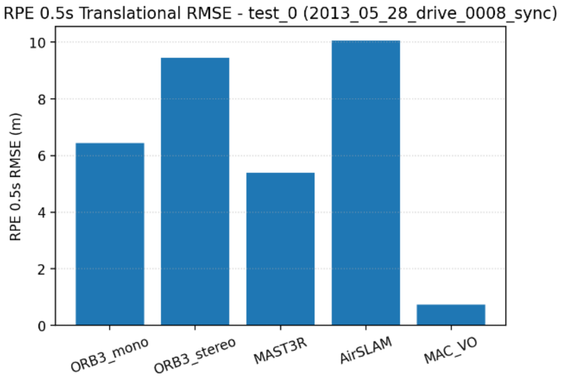
The Sparse vs. Dense Trade-off

The data reveals a counter-intuitive finding: "dense" does not automatically mean "accurate."

- ORB-SLAM3 Stereo achieved the lowest error (1.32 m), significantly outperforming the heavy deep-learning methods. This suggests that for pure localization, tracking a few thousand well-defined corners is often more robust than trying to reconstruct millions of points, especially when computational resources are limited.
- AirSLAM followed closely (1.97 m), confirming that hybrid methods can rival classical geometry.
- MASt3R-SLAM performed poorly (76.20 m), primarily because the extreme computational load forced a low frame rate (0.5 FPS), breaking the temporal continuity required for stable tracking.

Local Stability (Relative Pose Error)

While ATE measures global drift, the Relative Pose Error (RPE) measures how smooth the movement is from one frame to the next.

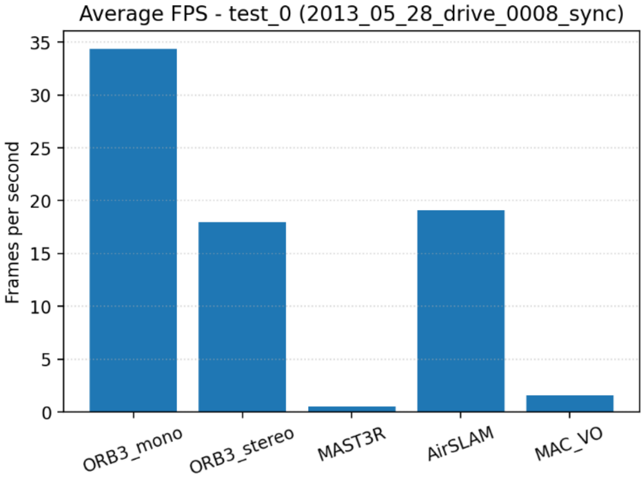


Here, the specialized nature of MAC-VO becomes apparent. Despite its inability to build a global map, it achieved a translational RPE of 0.55 m, significantly lower than the ~6.0 m average of ORB-SLAM3 and AirSLAM. This indicates that learning-based odometry is superior at smoothing out frame-to-frame "jitter," even if it lacks the global awareness to prevent long-term drift.

In contrast, MASt3R-SLAM occupied a middle ground with an RPE of 5.01 m. While this is slightly better than the keyframe-based jumps of ORB-SLAM3 and AirSLAM, it lags significantly behind the smooth tracking of MAC-VO. This performance likely reflects a compromise: the heavy "dense" optimization provides some local consistency, but the extremely low frame rate (and subsequent frame skipping) prevents it from achieving the ultra-smooth motion estimation seen in pure odometry networks.

Real-Time Viability (FPS vs. Latency)

For a SLAM system to be usable in robotics, it must process sensor data as fast as it arrives. The KITTI-360 dataset records at 10 Hz (100 ms per frame). Any algorithm slower than this introduces latency, causing the robot to react to old information—a critical failure in autonomous navigation.



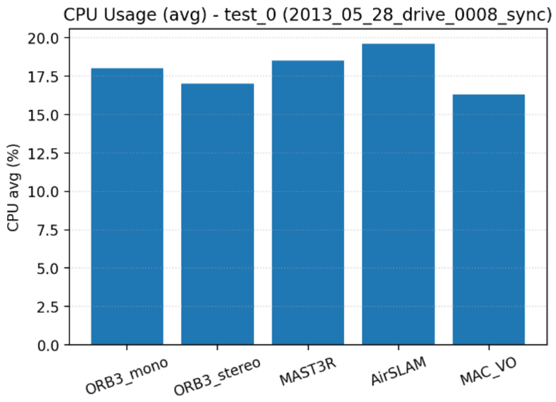
The "Real-Time Gap"

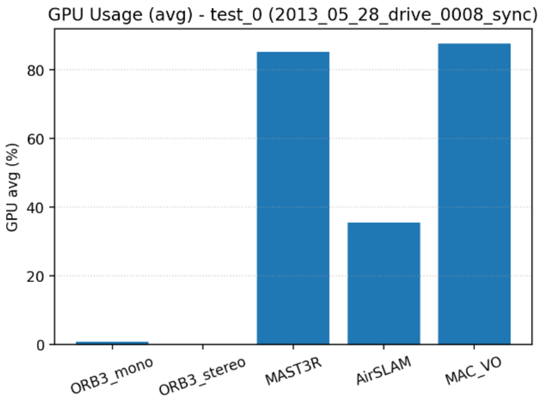
The performance gap between classical and learning-based methods is an order of magnitude wide:

- Deployment Ready: ORB-SLAM3 (Mono: 34.3 FPS, Stereo: 17.9 FPS) and AirSLAM (19.1 FPS) are the only candidates that safely exceed the 10 Hz threshold. They provide a "safety buffer," allowing the CPU/GPU to handle other tasks without dropping frames.
- Research Only: MASt3R-SLAM (0.5 FPS) and MAC-VO (1.6 FPS) are fundamentally unusable for real-time control on this hardware class. With latencies exceeding 600ms per frame, a robot moving at 10 m/s would travel 6 meters before the algorithm processed a single update.

Computational Resource Efficiency

A "good" algorithm must leave room for other processes. A robot must not only localize itself but also detect obstacles, plan paths, and communicate.





The resource usage data highlights distinct architectural costs:

1. CPU-Dominant (ORB-SLAM3): Uses ~17-18% CPU and effectively 0% GPU. This is the ideal profile for systems with limited power budgets or those needing the GPU for heavy perception tasks (e.g., running YOLO for object detection).
2. Balanced Hybrid (AirSLAM): Consumes ~19% CPU and ~35% GPU. This represents a modern "middle ground," utilizing the GPU to boost robustness without saturating it.
3. GPU-Saturated (MAC-VO & MASt3R-SLAM): These methods drive the RTX 4060 Ti to 85–100% utilization. In a practical deployment, this would cause thermal throttling and battery drain, and would prevent any other GPU-accelerated tasks from running.

Summary of Findings

Metric	ORB-SLAM3 (Stereo)	AirSLAM	MAC-VO	MASt3R-SLAM
Accuracy (ATE)	High (1.32m)	High (1.97m)	Moderate (3.34m)	Low (76.2m)
Speed (FPS)	Fast (17.9)	Fast (19.1)	Slow (1.6)	Very Slow (0.5)
GPU Load	None (~0%)	Moderate (~35%)	Max (~87%)	Max (~85%)
Viability	High	High	Low	Low

5. Conclusion & Recommendations

This study set out to answer a critical question for practical robotics: Are modern, learning-based SLAM systems ready to replace classical methods on consumer-grade hardware? By benchmarking four distinct algorithms (ORB-SLAM3, AirSLAM, MAC-VO, and MASt3R-SLAM) on the KITTI-360 dataset, we evaluated their performance not just on accuracy, but on the real-time constraints required for deployment.

Summary of Findings

The results demonstrate that while deep learning offers promising capabilities for dense reconstruction and local smoothness, classical geometry-based methods remain the most viable solution for real-time deployment on mid-range hardware.

- The "Gold Standard": ORB-SLAM3 (Stereo) proved to be the most practical system. It delivered the highest accuracy (1.32m RMSE) and comfortably exceeded real-time requirements (17.9 FPS) without utilizing the GPU. This efficiency makes it the ideal choice for embedded systems where the GPU is needed for other tasks, such as object detection or path planning.
- The "Hybrid" Contender: AirSLAM successfully bridged the gap between learning and geometry. By using the GPU for robust feature extraction (~35% load) and the CPU for optimization, it matched the speed of classical methods (~19 FPS) and demonstrated superior stability in long-term scenarios (Test 3). It is a strong recommendation for platforms equipped with a dedicated GPU.
- The "Research" Frontier: MASt3R-SLAM and MAC-VO highlighted the current computational ceiling of dense, fully learned methods. While capable of impressive outputs—such as MAC-VO's sub-meter local tracking precision—their high latency (0.5–1.6 FPS) and exhaustive resource consumption (85–100% GPU) render them currently unsuitable for online navigation.

Recommendations for Deployment

Based on this evaluation, the following recommendations are made for roboticists targeting similar hardware configurations:

- For General Navigation: Use ORB-SLAM3 Stereo. It offers the best reliability-to-cost ratio, ensuring the robot knows where it is without draining system resources.
- For Challenging Environments: If the environment features low texture or variable lighting (where classical features fail), AirSLAM is the preferred choice, provided the system can support the Docker containerization and GPU overhead.
- For Offline Mapping: If real-time performance is not required, MASt3R-SLAM can be utilized to generate dense, visually rich 3D maps of an environment for post-processing or simulation.

Appendix: Experimental Setup

This appendix details the hardware specifications and software environments used to reproduce the experiments.

Hardware Specifications

All benchmarks were executed on a standard consumer-grade desktop workstation designed to approximate a high-end robotics development platform.

- CPU: Intel Core i7-8700K (6 cores / 12 threads @ 3.70 GHz)
- GPU: NVIDIA GeForce RTX 4060 Ti (8 GB VRAM)
- RAM: 16 GB DDR4 @ 3467 MHz
- Storage: 2 TB Samsung 990 PRO SSD
- OS: Windows 10 Pro running WSL2 (Windows Subsystem for Linux)

Software Environment & Dependencies

The project relied on WSL2 to provide the necessary Linux environment for ROS and SLAM frameworks.

- Linux Distributions: Ubuntu 22.04 and 20.04 (depending on algorithm requirements)
- Containerization: Docker Desktop for Windows was used for AirSLAM and MAC-VO.
- GPU Drivers: CUDA Toolkit 12.4 (Driver 13.0 runtime).

Algorithm Installation Notes & Troubleshooting

ORB-SLAM3 (Python Wrapper)

- Environment: Ubuntu 22.04 (WSL2 Native)
- Setup: Straightforward installation via the ORB-SLAM3-python wrapper.
- Issues: None significant. The wrapper introduces slight overhead but greatly simplifies data loading compared to the C++ native build.

MASt3R-SLAM

- Environment: Ubuntu 22.04 (Conda environment)
- Setup Issues: The official instructions using conda install failed due to dependency conflicts.
- Setup Tutorial: https://www.youtube.com/watch?v=TK8DK19o6YQ
- Solution: A workaround was found fallowing the youtube tutorial and by replacing all conda install commands with pip install inside the active Conda environment.

- Optimization: To run on the 8GB VRAM GPU, the decoder was set to half-precision (fp16) and the processing stride was set to 3 (skipping 2 out of every 3 frames).

AirSLAM

- Environment: Ubuntu 20.04 (Docker Container)
- Setup Issues: Native installation on Ubuntu 20.04 failed due to system-level dependency errors.
- Solution: Utilized the official Docker image provided by the repository.
- Execution: docker run --gpus all -it --rm -v "$PWD":/workspace air_slam bash

Data preprocessing was required using make_data_4_AirSlam.py to rearrange KITTI-360 images into the expected directory structure.

MAC-VO

- Environment: Docker Container
- Setup Issues: Similar to AirSLAM, this required a strictly controlled environment to manage CUDA dependencies.
- Execution: docker run --gpus all -it --rm -v "$PWD":/workspace macvo:latest bash

Specific preprocessing (make_data_4_MAC_VO.py) was necessary to format the dataset before inference.