MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Revisiting Monocular SLAM with Spatio-Temporal Scene Modeling

Piedade, Valter; Manam, Lalit; Yamazaki, Masashi; Miraldo, Pedro

TR2026-056 May 16, 2026

Abstract

Visual SLAM is one of the most fundamental problems in computer vision, with direct applications to real-time localization tasks such as AR/VR, robotics, and 3D scene re- construction. Although significant progress has been made in both sparse and dense approaches, real-time monocular SLAM remains challenging—particularly in the uncalibrated setting, where existing methods are often inefficient and lack modularity. In this paper, we present a new visual SLAM pipeline, called SLAM-MER, which is implemented from scratch in C++ explicitly leveraging the spatio-temporal structure of the SLAM problem for improved localization, and has modular design to easily integrate off-the-shelf components. We introduce a temporal representation based on a buffer of recent keyframes that preserves short-term scene continuity. We complement this by incorporating a spatial representation based on a 3D cell-based scene model, enabling efficient retrieval of relevant 3D points from previously reconstructed regions. Leveraging recent feed-forward geometry estimators, our hybrid design combines sparse keypoint-based localization with a semi-dense anchor-point–driven spatial representation. This integration allows us to achieve real-time per- formance (exceeding 80 FPS) and a substantial efficiency improvement compared to existing uncalibrated monocular SLAM pipelines, while maintaining or improving localization accuracy.

IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2026

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Revisiting Monocular SLAM with Spatio-Temporal Scene Modeling

Valter Piedade[1,2], Lalit Manam[1], Masashi Yamazaki[3], Pedro Miraldo[1,✉]

[1]Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA
[2]Instituto Superior Técnico, Lisbon [3]Mitsubishi Electric Corporation, Tokyo

Abstract

Visual SLAM is one of the most fundamental problems in computer vision, with direct applications to real-time localization tasks such as AR/VR, robotics, and 3D scene reconstruction. Although significant progress has been made in both sparse and dense approaches, real-time monocular SLAM remains challenging—particularly in the uncalibrated setting, where existing methods are often inefficient and lack modularity. In this paper, we present a new visual SLAM pipeline, called SLAM-MER, which is implemented from scratch in C++ explicitly leveraging the spatio-temporal structure of the SLAM problem for improved localization, and has modular design to easily integrate off-the-shelf components. We introduce a temporal representation based on a buffer of recent keyframes that preserves short-term scene continuity. We complement this by incorporating a spatial representation based on a 3D cell-based scene model, enabling efficient retrieval of relevant 3D points from previously reconstructed regions. Leveraging recent feed-forward geometry estimators, our hybrid design combines sparse keypoint-based localization with a semi-dense anchor-point–driven spatial representation. This integration allows us to achieve real-time performance (exceeding 80 FPS) and a substantial efficiency improvement compared to existing uncalibrated monocular SLAM pipelines, while maintaining or improving localization accuracy. Project page: `merl.com/research/highlights/slam-mer`.

1. Introduction

Visual Simultaneous Localization and Mapping (SLAM) estimates a camera's 3D trajectory while simultaneously constructing a map of the environment. This problem has been studied for decades in both Computer Vision (CV) and robotics, with applications ranging from autonomous navigation to augmented reality. In robotics, multi-sensor variants that incorporate Inertial Measurement Unit (IMU) [5, 44, 53, 72] or Light Detection and Ranging (LiDAR) [23, 63, 72, 73] have improved accuracy and robustness. The CV community has focused on the more challenging monocular setting [14, 42, 45, 47, 54]. In this work, we aim to improve monocular visual SLAM by effectively leveraging real-time spatio-temporal cues of the scene; see Fig. **1**. Throughout this paper, we use Frames Per Second (FPS) as a metric for the real-time capability of our SLAM system, that is, the frequency with which incoming frames are processed by the SLAM pipeline.

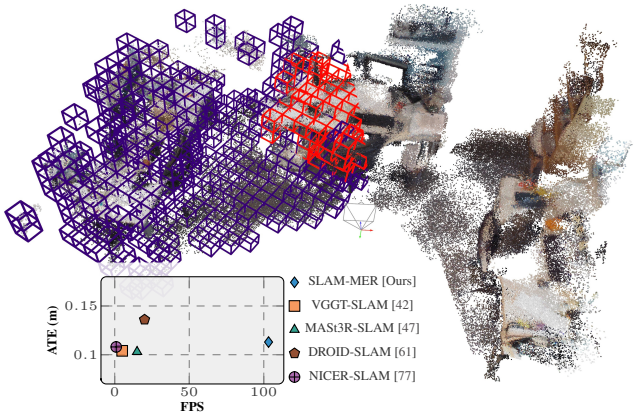


Figure 1. We propose a new pipeline for uncalibrated monocular SLAM, called **SLAM-MER**. Our pipeline uses a hybrid approach that combines sparse 3D points for real-time localization with feed-forward models for computing geometric priors. This figure shows our spatial modeling of the scene for the `office` sequence of 7-Scenes [57]. Over the semi-dense 3D reconstructed map, we show a cell-grid representation of the scene, where the red cells indicate those selected when spatially querying 3D points. For this sequence, our method runs at 100 FPS.

In its simplest form, SLAM consists of three components: map representation, camera localization, and map adjustment. Maps typically contain 3D points—either sparse keypoints or denser structures—and keyframes that capture distinct viewpoints of the scene. Localization estimates the camera pose of each frame using the 3D points from the map, while the adjustment stage maintains global consistency by optimizing a 3D covisibility graph at local

or global levels. A crucial additional component is loop detection. Because drift inevitably accumulates during localization and mapping, most systems run a parallel thread for Visual Place Recognition (VPR) and geometric verification to identify loop closures. In this paper, we present a new visual SLAM pipeline built from scratch in C++. While we rely on state-of-the-art components where appropriate, our primary technical contribution is improved camera localization through querying 3D points that effectively reflect the current camera pose.

Most localization approaches in SLAM, whether dense or sparse, rely on tracking 2D image keypoints from the latest keyframe to the current frame. This requires every tracked pixel to have a corresponding 3D map point. Although effective for gathering candidate points for pose estimation, this strategy fails to preserve temporal consistency in streaming scenarios–for instance, camera jitter may trigger unnecessary keyframe creation when 2D keypoints momentarily disappear. We address this with an explicit *temporal representation*: a temporal buffer of recent frames that maintains connections to previous keyframes. This allows 2D keypoints to be tracked for longer and reduces the size of both the covisibility graph and the overall map.

Although the temporal buffer reduces the size of the map, as shown in our ablations, it cannot fully prevent forgetting because it is ultimately bounded by its capacity. To revisit and exploit regions reconstructed earlier, we introduce a complementary *spatial representation*. Instead of relying solely on a sparse set of 3D points, we maintain a 3D volume of cells that captures a coarse spatial layout of the scene. For each incoming frame, we determine which cells are visible and query the 3D points contained within them. These 3D points are then combined with those retrieved from the temporal buffer to form the set of 3D points used for absolute pose estimation.

In the CV community, several directions have been explored for representing the 3D map in real-time SLAM. Recent feed-forward geometry estimators such as [37, 65, 67] have led to growing interest in dense representations for this problem. A key advantage of these models is that they can produce point clouds directly from input images without requiring camera calibration–a desirable property for uncalibrated monocular SLAM. However, dense representations are often too computationally heavy for real-time use, where processing rates above 25 FPS are needed. In this paper, we propose a hybrid approach that maintains a sparse, compact scene representation for real-time localization while using an anchor-point–based strategy to densify the map. By leveraging feed-forward single-image geometry estimators, our system naturally supports uncalibrated monocular SLAM. Our contributions are as follows.

- We introduce a **spatio-temporal querying** of 3D points for V-SLAM. (a) Spatial: using cell indexing to retrieve candidate map-points given a camera pose; (b) Temporal: retain map-points for a fixed time window.
- We argue that accurate localization can be achieved with **3D–2D correspondences**, without per-frame depth as required in recent 3D–3D matching pipelines (MASt3R-/VGGT-SLAM). This design preserves localization accuracy while improving efficiency by relying on lightweight keypoint tracking, with depth inferred only for keyframes.
- We present the Mitsubishi Electric Research framework for visual SLAM (**SLAM-MER**), **a new modular C++ framework** for real-time uncalibrated monocular SLAM, that fuses geometry principles with recent data-driven techniques.

Our framework allows modules to be easily swapped (*e.g.*, feed-forward depth estimator, VPR, feature detector); some examples are shown in the supplementary material. The framework will be publicly released to facilitate experimentation and development.

2. Related Work

The literature on visual SLAM is extensive, and a complete survey is beyond the scope of this paper. We structure our review into three parts: traditional sparse geometric SLAM methods; methods based on image rendering (like NeRFs [43], Gaussian splatting [29]); and recent approaches that use feed-forward predictors for geometry estimation. For a broader overview, we refer to [6, 49].

Traditional geometric SLAM: Traditional visual SLAM systems rely on four key geometric components: handcrafted feature extraction, multi-view geometry, robust 3D-to-2D pose estimation, and sparse map reconstruction. MonoSLAM [14] was the first real-time monocular system, jointly estimating camera pose and sparse 3D points via an Extended Kalman Filter. PTAM [30] introduced decoupled tracking and mapping with parallel bundle adjustment, forming the foundation of modern real-time SLAM. The ORB-SLAM family [5, 45, 46] further advanced these ideas using ORB features across monocular, stereo, and RGB-D setups. Concurrently, LSD-SLAM [21] proposed a direct, semi-dense approach based on photometric alignment. Classical SLAM has also been extended to wide field of view cameras [7], visual–inertial systems [3, 5, 38, 44, 53], and multimodal vision–LiDAR methods [23, 72]. While these systems are computationally efficient, dense 3D mapping remains difficult, and none achieves accurate real-time SLAM with uncalibrated intrinsics. In this paper, we build on the strengths of traditional geometric SLAM—namely, the sparse 3D map representations and keypoint-based localization for real-time performance.

SLAM using image rendering methods: Image rendering methods have recently enabled high-quality dense SLAM systems. Although originally developed for image synthe-

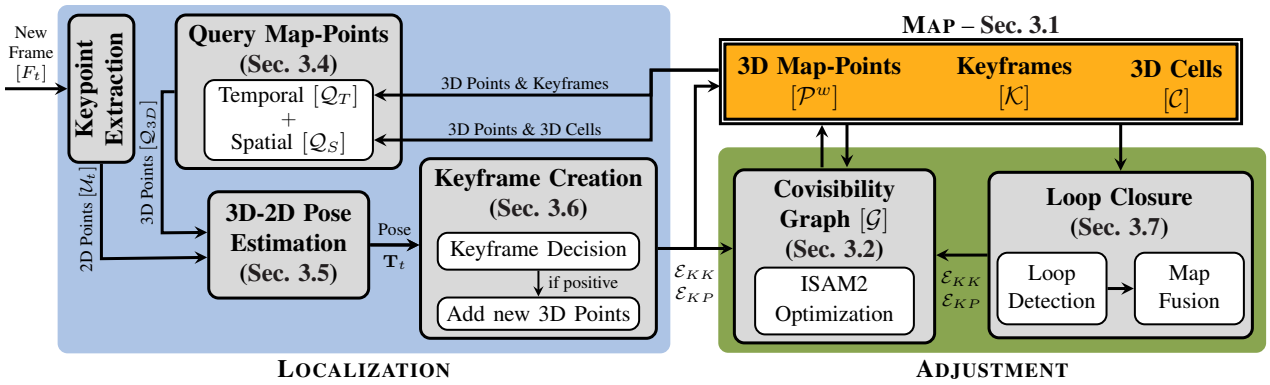


Figure 2. Our **SLAM-MER** pipeline. New frames are localized based on two-way correspondences between 2D image keypoints and 3D map-points from the current map estimate: (1) temporal ones from previous frames and (2) spatial ones directly from the map via 3D cells. When a keyframe is created, 3D map-points are obtained from monocular depth inference using, *e.g.*, MASt3R [47]. Each new keyframe and its 3D map-points add constraints to the covisibility graph, which is updated from the incremental solver in ISAM2 [28]. To correct drift in the trajectory, we have a loop closure functionality that only adds new constraints to the covisibility graph when a loop is detected, but does not perform any optimization. The map will reflect the loop constraints after the succeeding update step in the incremental solver.

sis, several works have adapted them for SLAM. iMAP [59] first demonstrated real-time tracking and mapping using a neural implicit scene representation with RGB-D input. NeRF-SLAM [54] extended this idea to monocular SLAM by incrementally training a neural radiance field for joint pose and map estimation. Other approaches, including iSDF [48], NICE-SLAM [76], and NICER-SLAM [77], further optimize neural radiance or signed-distance fields together with camera poses. Gaussian splatting-based SLAM has emerged as another alternative due to its real-time rendering capability. GS-SLAM [70] demonstrated dense visual SLAM, while Outdoor Gaussian Pointmap SLAM [10] and SEGS-SLAM [69] integrate Gaussian splatting [29] with SLAM front-ends to produce photorealistic reconstructions. While these methods achieve high-quality dense reconstructions, they remain computationally heavy and slower to converge than classical geometric pipelines. Aiming for real-time operation, we focus on a simpler 3D point-cloud representation in our pipeline.

Feed-forward depth prediction in SLAM: Recent feed-forward multi-view models such as DUSt3R [67], MASt3R [37], and VGGT [65] produce dense depth from uncalibrated RGB images. These provide strong reconstruction priors that inspired a new wave of SLAM systems. Other models, such as CUT3R [66], Pi3 [68], and TTT3R [8], improve geometric consistency and generalization to different scenes. Large-scale extensions such as VGGT-Long [16] and Long3r [9] address kilometer-scale drift via chunking, loop alignment, and global refinement. Building on these methods, learning-based SLAM approaches replace geometric triangulation with dense depth prediction or correspondence prediction [26, 37, 39, 65, 67, 71]. MASt3R-SLAM [47] leverages MASt3R's priors for dense real-time mapping, while VGGT-SLAM [42] uses VGGT predictions for submap alignment on the $\mathbb{SL}(4)$ manifold. DROID-SLAM [61], DPV-SLAM [40], GO-SLAM [74], DeepV2D [60], DeepFactors [13], TANDEM [31], and SPAN3R [64] combine learned depth or motion cues with a back-end optimization.

Feed-forward point-cloud prediction probably marks the most significant shift in visual SLAM after MonoSLAM, PTAM, and ORB-SLAM. It simplifies traditional SLAM pipelines, avoids several failure-prone multi-view geometry stages, and enables more flexible operation while yielding dense scene predictions. However, no feed-forward SLAM framework currently operates in real-time above 30 FPS. Our pipeline combines the strengths of feed-forward methods with geometric SLAM to achieve 80+ FPS.

3. SLAM-MER Pipeline

This section details the proposed **SLAM-MER** pipeline. It consists of a map representation structure, a covisibility graph, and two modules that interact with them: localization and adjustment. Figure **2** provides an overview of our proposed SLAM framework. Below, we first define the data structures and then describe the two modules.

3.1. Map Representation

The input is a stream of frames $\mathcal{F} = \{F_t\}_{t=\{1,\cdots,S\}}$. Each frame F_t consists of an RGB image obtained from an uncalibrated monocular camera along with a timestamp. We denote $\mathcal{U}_t$ as the set of 2D keypoints in the image and $\mathbf{T}_t \in \mathbb{SE}(3)$ for the absolute pose of the frame, which consists of a rotation and a translation (see [47, 62]).

Given an input stream $\mathcal{F}$, we create a map of the environment $\mathcal{M} = (\mathcal{K}, \mathcal{P}^w, \mathcal{C})$ in an online manner. It consists

of keyframes $\mathcal{K} \subset \mathcal{F}$, 3D map-points $\mathcal{P}^w$, and 3D cells $\mathcal{C}$. Keyframes and 3D map-points are the typical map representation in SLAM, with keyframes being the input frames used for constructing the map. Every keyframe $K_i \in \mathcal{K}$ is associated with a pose $\mathbf{T}_i^{kf} \in \mathrm{Sim}(3)$, a set of 2D image keypoints $\mathcal{U}_i^{kf}$ along with its corresponding local 3D points $\mathcal{P}_i^{kf}$ in the local coordinates of the keyframe. Every 3D map-point $P_j \in \mathcal{P}^w$ consists of its world coordinates and a descriptor (the descriptor of the most recent 2D keypoint). Every cell $C_i \in \mathcal{C}$ represents a volume in the 3D space grouping 3D map-points, *i.e.*, $C_i \subset \mathcal{P}^w$ such that $C_k \cap C_l = \phi$ for any $C_k, C_l \in \mathcal{C}$.

3.2. Covisibility Graph

The covisibility graph $\mathcal{G} = (\{\mathcal{K} \cup \mathcal{P}^w\}, \{\mathcal{E}_{KK} \cup \mathcal{E}_{KP}\})$ consists of the relations between keyframes and map-points. Every node $K_i \in \mathcal{K}$ represents a keyframe, and every node $P_j \in \mathcal{P}^w$ represents a 3D map-point. Every edge $(i_1, i_2) \in \mathcal{E}_{KK}$ represents a keyframe-keyframe relation in terms of the relative pose between them. Each edge $(i, j) \in \mathcal{E}_{KP}$ represents a tuple of constraints: (1) 3D-2D projection constraint between a 2D keypoint in the keyframe K_i and a 3D map-point P_j, and (2) 3D-3D Euclidean distance between a transformed local 3D point of keyframe K_i and P_j.

3.3. Pipeline Workflow

Our pipeline has two core modules running in parallel: localization and adjustment. This design choice aids in achieving real-time operation as we do not require separate computation during loop closure like other SLAM pipelines, *e.g.*, ORB-SLAM3 [5]–we do not explicitly separate adjustments into local and global ones.

Localization module: Given a current frame F_t, the localization module starts by extracting 2D keypoints $\mathcal{U}_t$ from the image using ALIKED [75] features. Then, we query 3D map-points $\mathcal{P}^w$ (Sec. **3.4**) to establish 3D-2D correspondences (1) temporally between the current frame and previous frames, and (2) spatially through the visible 3D map-points in the current frame. These two-way correspondences are used for estimating the absolute pose $\mathbf{T}_t$ of the current frame F_t in the world coordinates (Sec. **3.5**). Now, we decide if a keyframe needs to be created from the current frame based on certain criteria (Sec. **3.6**). For creating a keyframe K_i from frame F_t, we assign the keypoints $\mathcal{U}_t$ to $\mathcal{U}_i^{kf}$, and create local 3D points $\mathcal{P}_i^{kf}$ for every 2D keypoint in $\mathcal{U}_i^{kf}$. We get these local 3D points from depth inference methods. In this paper, we use MASt3R [37][1] but other methods like VGGT [65] or [50] (for distorted images) can be used. Unlike *e.g.*, MASt3R-SLAM [47] or VGGT-SLAM [42], which need to run inference for every frame, we only run it when a keyframe is created (and in parallel), making our pipeline more efficient compared to such dense approaches. Since the local 3D points $\mathcal{P}_i^{kf}$ obtained from depth inference are scale-free, we estimate a scale factor as follows: we check the 2D keypoints which have 3D-2D correspondences and get a scale factor that best fits the transformed local 3D points $\mathbf{T}_t\mathcal{P}_i^{kf}$ to the 3D map-points $\mathcal{P}^w$. Now, the keyframe K_i is added to the map with $\mathbf{T}_i^{kf}$ having the same rotation and translation as in $\mathbf{T}_t$. Then, we add newly observed local 3D points $\mathcal{P}_i^{kf}$ and 2D keypoints $\mathcal{U}_i^{kf}$ to the map. We update the covisibility graph $\mathcal{G}$ with the new keyframe $\mathcal{K}_i$, new 3D map-points, and the relations among them and with previous map elements. When a keyframe is not created with the current frame, none of these processes is performed, and we move to the next frame. We do not need to map new points $\mathcal{P}^w$, and therefore we do not need to run depth inference. We maintain a buffer with the last few frames that were localized. This, along with the two-way correspondences, results in a robust localization of the camera. To tackle extreme situations when the camera is lost, we develop a relocalization functionality (Sec. **3.8**).

We highlight that we do not require intrinsic camera parameters for monocular SLAM, as we infer the local 3D points from the image and compute the camera focal length during pose estimation (see Sec. **3.5**). The localization module is the bottleneck of the pipeline in terms of computational cost, and its accuracy is critical to ensure that the map is properly initialized to allow subsequent adjustments requiring only small corrections.

Adjustment module: The adjustment module operates in parallel with the localization module and interacts with both the map $\mathcal{M}$ and the covisibility graph $\mathcal{G}$. This module updates the map variables—namely the keyframe poses $\{\mathbf{T}_i^{kf}\}_{K_i \in \mathcal{K}}$ and the 3D map-points $\{P_j\}_{P_j \in \mathcal{P}^w}$—based on the connections defined in the covisibility graph $\mathcal{G}$. Unlike existing pipelines that only run optimization when a new keyframe is created, our module continuously monitors changes in $\mathcal{G}$ in a parallel thread and incrementally updates the variables using the ISAM2 solver [28] and the GTSAM framework [15]. ISAM2 provides an efficient factor-graph representation for updating the variables, and the covisibility graph supplies the necessary constraints to the solver. Another key advantage of ISAM2 is that it supports both local and global adjustments of $\mathcal{G}$ without requiring separate processing branches. This allows us to maintain a single unified graph structure throughout the optimization process. This module also includes the loop-closure functionality (Sec. **3.7**), which detects and validates loops between keyframes and then fuses the corresponding 3D map-points.

Modular pipeline: We note that any choice of 2D keypoint descriptor, depth inference model, or image-retrieval method can be easily integrated into our pipeline. The code

[1] We export the original model to ONNX and run inference in C++ using ONNXRuntime [18]. From MASt3R, we only use the predicted pointmap and respective confidences.

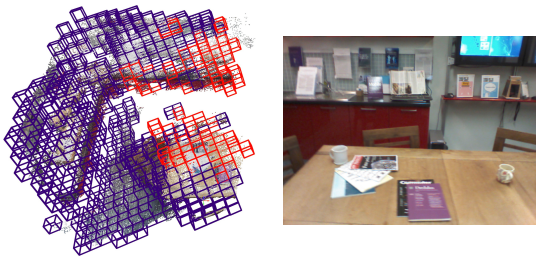


Figure 3. Spatial query of 3D map-points. The left image shows the cell-based representation, where red indicates the closest visible cells. The right image shows the corresponding camera view.

is fully modular, allowing components to be swapped with minimal effort. The pipeline can also be adapted to incorporate RGB-D inputs. This flexibility enables replacing off-the-shelf modules with stronger alternatives as they become available, ensuring long-term extensibility toward a robust SLAM system. See the supplementary materials for additional experiments.

3.4. Query Map-points

We query for 3D map-points to obtain 3D-2D correspondences between the current frame F_t keypoints and the 3D map-points $\mathcal{P}^w$. For this, we match the descriptors of the 2D keypoints $\mathcal{U}_t$ in the input image with the descriptors of 3D map-points $\mathcal{Q}_{3D} \subset \mathcal{P}^w$, which are selected with the temporal and spatial criteria stated next.

Temporal query ($\mathcal{Q}_T$): The localization module maintains a buffer $\mathcal{B} \subseteq (\{F_{t-N-1}, \cdots, F_{t-1}\})$ with N previous frames that were localized against the map. For every frame $F_l \in \mathcal{B}$, we know which keyframe $K_i \in \mathcal{K}$ has the most 3D map-points in common. This is known from 3D-2D inlier correspondences for the frame $F_l \in \mathcal{B}$ (obtained during the pose estimation of F_l). We collect all the 3D map-points from such keyframes K_i and construct a set $\mathcal{Q}_T$.

Spatial query ($\mathcal{Q}_S$): We use the pose $\mathbf{T}_k$ of the last frame F_k that was localized to obtain the visible 3D cells. The 3D cells are continuously asynchronously sorted based on the distance to the camera location (computed from $\mathbf{T}_k$) to improve efficiency. After sorting, for efficiency, we ensure that the picked-up cells are visible from the pose $\mathbf{T}_k$ by rasterizing the occupied ones back to the image. We only add to $\mathcal{Q}_S$ map-points from the closest 3D cells to the camera pose. In Fig. **3**, we show an example of a spatial query.

The final set of queried 3D map-points ($\mathcal{Q}_{3D}$) is given as $\mathcal{Q}_{3D} = \mathcal{Q}_T \cup \mathcal{Q}_S$. We do a nearest neighbour search using the descriptors in $\mathcal{Q}_{3D}$ and the descriptors of 2D keypoints $\mathcal{U}_t$ to obtain 3D-2D correspondences using FAISS [20, 27]. By combining both queries, we can obtain a good quantity of spatial and temporal 3D-2D correspondences, thus reducing the number of keyframes and improving efficiency. Moreover, the spatial query of 3D map-points can also close loops without relying on the loop closure module, as long as trajectory drift is low. If the camera pose $\mathbf{T}_k$ for frame F_k has a high drift, cells will fail to rasterize on the image, and will depend on the loop closure module to close loops.

3.5. Pose Estimation

Once we have 3D-2D correspondences by querying the 3D map-points, we localize the current frame F_t, *i.e.*, estimate its absolute pose $\mathbf{T}_t$. Since the camera calibration is unknown, we utilize the P4Pf solver introduced in [32]. The solver is run within a Random Sample Consensus (RANSAC) [11, 22, 51] loop for outlier robustness, and we perform a final pose refinement with inlier 3D-2D correspondences only. P4Pf gives us an estimate of the camera focal length, which we assume is the same in both axes. We also assume that the principal point is the center of the image[2]. Since P3P [19] is faster than P4Pf [32], we only use P4Pf until a certain number of frames are processed. After that, we assign the focal length as the median over all its past estimates and use it for upcoming frames, as if the camera is now calibrated. PoseLib [34] is used for both solvers.

3.6. Keyframe Creation

After estimating the pose $\mathbf{T}_t$ of the current frame F_t, we check whether a new keyframe should be created from the current frame F_t. New keyframes should be created when (1) the image can add a significant amount of new 3D map-points, and (2) not enough 3D map-points are being used for localization of the current frame. We use the following decision criteria for keyframe creation:

1. When inliers from pose estimation are low in number;
2. Check how well 2D keypoints with 3D-2D correspondences are spread across the image;
3. Look for a significant variation between the current and previously localized frame in terms of the number of 3D map-points tracked over all keyframes.

The first criterion is most commonly used for this task in SLAM. The second criterion ensures that the spread of the 2D image keypoints with 3D-2D correspondences (only the pose estimation inliers) is wide. This has two advantages: (1) It allows us to add new 3D map-points to increase the coverage of the 3D scene in the map, consequently avoiding the camera being lost, and (2) it reduces the chances of having an ill-conditioned problem for the pose estimation [25], thus increasing the pose accuracy. To measure the spread of 2D keypoints having 3D-2D correspondences, we compute a ratio between the contour areas of the convex hulls from all 2D keypoints with 3D-2D correspondences and all the detected 2D image keypoints.

[2]For strong lens distortion and shifted camera centers, P4Pf can be replaced by distortion-aware or shifted camera centers solvers [35, 36].

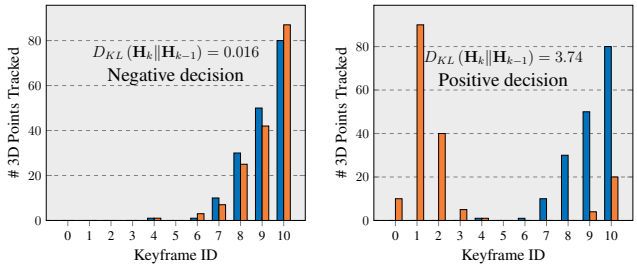


Figure 4. Keyframe decision criterion based on the distribution of points tracked per keyframe. When observing new parts of the scene, the histogram is skewed to the right since only the most recent keyframes have points in common with the current frame F_t. That leads to a small KL divergence score (D_{KL}) and no keyframe creation (left figure). When visiting a previously mapped position, previous keyframes start to observe current points, causing a drastic change in the histogram and a high D_{KL}. When such a change occurs, a keyframe is created (right figure).

To employ the third criterion, after pose estimation, we create a histogram $\mathbf{H}_k$ that counts the number of points in the current frame F_t viewed by the keyframes $\mathcal{K}$ in the map. Then we compare it with the histogram obtained for the previously localized frame, $\mathbf{H}_{k-1}$. We use Kullback–Leibler (KL) divergence [33] $D_{KL}(\mathbf{H}_k||\mathbf{H}_{k-1})$ to compare both histograms and decide if a keyframe should be created. Computing $D_{KL}(\mathbf{H}_k||\mathbf{H}_{k-1})$ is straightforward, see details in the supplementary material. We create a keyframe with the current frame F_t when there is a drastic change in the histogram, which indicates that we are getting 3D map-points from a previously known position (*i.e.*, forming a loop). Creating a keyframe adds new constraints to the covisibility graph, reflecting the loop formed. Figure **4** exemplifies the change in the histograms we look for. Note that this criterion is complementary to our pipeline because we are using spatial 3D map-point queries.

3.7. Loop Closure

In any online localization pipeline, drift in the camera pose inevitably occurs and must be corrected to obtain a correct map. Specific to our framework, drift can cause the proposed spatial query of 3D map-points to fail, as the pose of the current frame might point the camera at the wrong viewing direction or grossly misplace the camera location. To fix long-term trajectory drift, we have a loop closure functionality between keyframes, which has two main steps: loop detection and map fusion.

Loop detection: Loop detection has two phases, loop candidate selection and loop validation. To select loop candidates, instead of relying on the traditional bag-of-words approach like in ORB-SLAM3 [5], we use the state-of-the-art image retrieval technique MegaLoc [1][1] to generate image-level descriptors. For each keyframe, an image-level descriptor is stored in a database. When a new keyframe K_i is created, we select candidate keyframes $\mathcal{K}_i^{loop} \subset \mathcal{K}$ for loop closure that have a similar image-level descriptor to the current keyframe. Then, for every candidate keyframe $K_{cand} \in \mathcal{K}_i^{loop}$, we establish 3D-2D correspondences between the keyframe K_i and the 3D map-points $\mathcal{P}^w$ through K_{cand}. We check which of the 3D map-points $\mathcal{P}^w$ are visible to K_{cand}, and use the descriptors of these 3D map-points to match with the keypoint descriptors of K_i. Then, we geometrically validate the loop by absolute pose estimation using the 3D-2D correspondences.

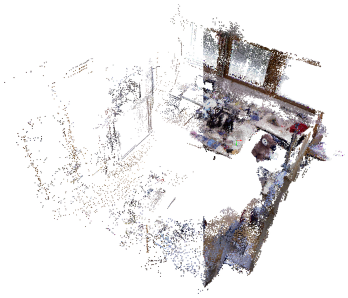


Figure 5. Reconstructed scene for the `room` sequence of TUM [58]. The left side of the figure shows the sparse 3D map-points used for localization. The right side displays the densified representation of the scene using the sparse map-points as anchors.

Map fusion: For every validated loop candidate, we add $\mathcal{E}_{KP}$ edges in the covisibility graph $\mathcal{G}$ from 3D-2D correspondences and an $\mathcal{E}_{KK}$ edge between K_{cand} and K_i. We also merge two 3D map-points in $\mathcal{P}^w$, if a keypoint from $\mathcal{U}_i^{kf}$ of keyframe K_i, having a 3D-2D correspondence obtained through K_{cand}, has a different 3D map-point associated with it at the time of keyframe creation. No explicit optimization step is run at the end of this stage, as the covisibility graph is continuously checking for newly added information and if it needs to run adjustments to the map.

3.8. Relocalization

It is crucial in SLAM to relocalize the camera once the camera is lost, *i.e.*, when it is unable to match keypoints with 3D query points, in its current frame F_t. Due to the temporal and spatial query of 3D map-points, our pipeline is robust to most short-duration camera occlusions. However, in extreme situations, *e.g.*, the kidnapped robot problem, a relocalization functionality is required. In our pipeline, we implemented a relocalization feature that is triggered when no frame in the buffer $\mathcal{B}$ has a reference keyframe in the map. In this mode, for every new frame, we follow the loop detection process: (1) create an image-level descriptor; (2) search for keyframe candidates with similar images; and (3) validate a keyframe candidate using pose estimation. For a valid keyframe candidate, a new keyframe is created with the current frame F_t, and the localization mode resumes.

3.9. Densified 3D Reconstruction

Although our map is composed of sparse 3D map-points, it can still yield a semi-dense 3D representation of the scene.

Table 1. Camera pose estimation results on TUM RGB-D and 7-Scenes datasets. Baseline Absolute Trajectory Error (ATE) (m) values are taken from [42]. For a fair FPS comparison with MASt3R-SLAM and VGGT-SLAM, we report their runtime measured on our machine. "Sparse Loc." indicates the use of sparse features for localization, "Dense map" denotes methods producing denser point clouds, and "Uncalib." specifies methods handling uncalibrated SLAM. The **best** and second-best results are highlighted separately for calibrated (in gray) and uncalibrated settings.

(a) TUM RGB-D dataset [58].

Method	Type			Sequence									Avg	FPS
	Sparse Loc.	Dense map	Uncalib.	360	desk	desk2	floor	plant	room	rpy	teddy	xyz		
ORB-SLAM3 [5]	✓	✗	✗	×	0.017	0.210	×	0.034	×	×	×	**0.009**	N/A	N/A
DPV-SLAM++ [40]	✓	✗	✗	0.132	0.018	0.029	0.050	0.022	0.096	0.032	0.098	0.010	0.054	~**30**
MASt3R-SLAM [47]	✗	✓	✗	**0.049**	**0.016**	**0.024**	**0.025**	**0.020**	**0.061**	**0.027**	**0.041**	**0.009**	**0.030**	11.9
DROID-SLAM [61]	✗	✓	✓	0.202	0.032	0.091	0.064	0.045	0.918	0.056	0.045	0.012	0.158	~20
MASt3R-SLAM [47]	✗	✓	✓	**0.070**	0.035	0.055	**0.056**	0.035	0.118	0.041	0.114	0.020	0.060	13.2
VGGT-SLAM (Sim(3)) [42]	✗	✓	✓	0.123	0.040	0.055	0.254	**0.022**	0.088	0.041	**0.032**	0.016	0.074	<5
VGGT-SLAM ($\mathbb{SL}(4)$) [42]	✗	✓	✓	0.071	**0.025**	0.040	0.141	0.023	0.102	**0.030**	0.034	0.014	**0.053**	<5
SLAM-MER [Ours]	✓	✓	✓	0.082	0.033	**0.036**	0.063	0.081	**0.083**	0.038	0.078	**0.010**	0.056	**86.6**

(b) 7-Scenes dataset [57]. The pumpkin sequence is omitted due to ground-truth inconsistencies and is explained in the supplemental material.

Method	Type			Sequence						Avg	FPS
	Sparse Loc.	Dence map	Uncalib.	chess	fire	heads	office	kitchen	stairs		
NICER-SLAM [77]	✗	✓	✗	**0.033**	0.069	0.042	0.108	**0.039**	0.108	0.067	<1
DROID-SLAM [61]	✗	✓	✗	0.036	0.027	0.025	**0.066**	0.040	0.026	**0.037**	~**20**
MASt3R-SLAM [47]	✗	✓	✗	0.053	**0.025**	**0.015**	0.097	0.041	**0.011**	0.040	14.4
DROID-SLAM [61]	✗	✓	✓	0.047	0.038	0.034	0.136	0.080	0.044	0.063	~20
MASt3R-SLAM [47]	✗	✓	✓	0.063	0.046	0.029	**0.103**	0.074	**0.032**	0.058	15.0
VGGT-SLAM (Sim(3)) [42]	✗	✓	✓	0.037	**0.026**	**0.018**	0.104	0.061	0.093	0.057	<5
VGGT-SLAM ($\mathbb{SL}(4)$) [42]	✗	✓	✓	**0.036**	0.028	**0.018**	**0.103**	**0.058**	0.093	**0.056**	<5
SLAM-MER [Ours]	✓	✓	✓	0.038	0.042	0.050	0.113	0.066	0.044	0.059	**103.2**

Densification is straightforward: for each keyframe K_i, we sample from the dense point cloud produced by the depth inference method, anchoring nearby points to their corresponding 3D map-points, and discarding those that lie too far away. Each map-point thus accumulates neighboring points from the keyframes that observe it, and because these are stored in local keyframe coordinates, they can be efficiently updated to world coordinates whenever the map-points or keyframes are adjusted. Consequently, map-points observed by more keyframes gather more neighbors, producing a compact yet denser version of the scene for localization, while regions not supported by map-points remain uncovered. In Fig. **5**, we show an example of 3D map-points and semi-dense point clouds that use map-points as anchors.

4. Experiments and Results

All experiments were conducted on a single computer equipped with an Intel Core i9-14900K CPU and an NVIDIA GeForce RTX 4090 GPU.

Metrics: We follow the evaluation protocol used in recent methods, namely MASt3R-SLAM [47] and VGGT-SLAM [42]. Performance is assessed in terms of trajectory accuracy and 3D reconstruction error. For pose accuracy, we report Root Mean Square Error (RMSE) of the ATE[3]. For 3D reconstruction quality, we report accuracy, completion, and Chamfer distance between the predicted reconstruction and the ground-truth scene, following [42, 47]. For efficiency, we measure frames-per-second (FPS), *i.e.*, the rate at which frames are processed.

Datasets: Experiments are conducted on the 7-Scenes [57] and TUM RGB-D [58] datasets, consistent with prior work [42]. Additional datasets and sequences are included in the supplementary material.

Baselines: We compare **SLAM-MER** against recent approaches spanning sparse *vs.* dense and calibrated *vs.* uncalibrated (see Tab. **1**). All accuracy metrics are taken from [42, 47]. For FPS, we evaluate the closest competitors—MASt3R-SLAM [47] and VGGT-SLAM [42]—on the same hardware. An *important note* about baselines is that our approach processes every frame of a 30 FPS video stream in real-time (no frame skipping), as shown below (Sec. **4.2**). In contrast, MASt3R-SLAM and VGGT-SLAM, among others, cannot process all frames in real-time and therefore drop two out of every three frames, as reported in their respective papers.

4.1. Ablation Study

Table **2** provides an ablation on the spatio-temporal queries.

[3] After Sim(3) Umeyama alignment [62] with timestamp synchronization. We use the evo evaluation toolkit [24].

Table 2. Ablation study on the temporal buffer size $|\mathcal{B}|$ and spatial query of 3D points (cell size $|\mathcal{C}|$). We measure ATE (m), number of keyframes ($|\mathcal{K}|$), number of 3D map-points ($|\mathcal{P}^w|$), and FPS.

Temporal $\|\mathcal{B}\|$	Spatial $\|\mathcal{C}\|$	Loop Closure	desk				room			
			ATE	$\|\mathcal{K}\|$	$\|\mathcal{P}^w\|$	FPS	ATE	$\|\mathcal{K}\|$	$\|\mathcal{P}^w\|$	FPS
1	–	✗	0.084	31	8659	168.9	0.304	88	23327	161.4
10	–	✗	0.073	31	7820	159.3	0.280	86	22731	144.9
100	–	✗	0.059	28	7551	114.2	0.191	79	20381	98.4
1000	–	✗	0.040	23	6445	81.8	0.151	67	17772	45.9
–	≈ 8 cm	✗	0.035	22	6342	40.3	0.103	77	20512	30.9
–	≈ 16 cm	✗	0.038	23	6521	72.8	0.241	76	20882	63.1
–	≈ 32 cm	✗	0.039	24	6675	93.8	0.269	78	21059	97.1
100	≈ 32 cm	✗	0.034	24	6217	83.8	0.163	74	20049	76.2
100	≈ 32 cm	✓	0.033	15	4268	93.6	0.094	73	18625	83.6

Temporal query: We first vary the buffer size $|\mathcal{B}|$ from 1 (equivalent to querying from the latest keyframe, as in prior work) to 10, 100 and 1000 frames (< 1 to ≈ 160 secs). As expected, a larger $|\mathcal{B}|$ increases accuracy by exposing more 3D points, but reduces the FPS. In `desk`, the largest gain appears at $|\mathcal{B}|= 1000$ since the entire sequence (614 frames) is being fitted. This does not occur in `room` (1363 frames). In general, $|\mathcal{B}|= 100$ offers the best accuracy–efficiency trade-off and is used in subsequent experiments.

Spatial query: Next, we test cell sizes of 8, 16 and 32 cm. Although ATE, keyframe count and point count change only marginally, FPS is strongly affected. Smaller cells require sorting and validating more cells per frame for a similar number of queried points, increasing cost; overly large cells return too many points, also slowing the pipeline (see supplementary material). For efficiency, we use 32cm.

Combined queries: Combining temporal and spatial queries improves ATE, particularly in the longer and more challenging `room` sequence, while preserving high FPS. Loop closure further reduces the ATE. In `desk`, drift is minimal, so spatial querying already retrieves long-term points, effectively acting as a loop closure. In `room`, drift prevents correct cell rasterization near the end of the sequence, so explicit loop closure is needed.

4.2. Camera Pose Estimation

Table **1** shows the performance of **SLAM-MER** compared to baselines. In general, the efficiency difference between **SLAM-MER** and all baselines is very noticeable. **SLAM-MER** is the only approach that performs localization using sparse 2D keypoints while simultaneously generating a semi-dense map. Compared to any uncalibrated approach, our results are on par with the most accurate ones, being much more time efficient than any prior method. Even compared to calibrated cases, we get a significant boost in efficiency. We are $3\times$ faster than DPV-SLAM++, which also utilizes sparse 2D keypoints but only produces a sparse point cloud, and not less than $5\times$ faster than the most efficient dense approach, DROID-SLAM.

Table 3. RMSE (m) dense reconstruction results on 7-Scenes [57]. Baseline values were taken from [42]. "@n" indicates a keyframe every n frames. **Best** and second-best results are highlighted.

Method	Type		7-Scenes		
	Sparse Loc.	Uncalib.	Acc. ↓	Comp. ↓	Chamfer ↓
DROID-SLAM [61]	✗	✗	0.141	0.048	0.094
MASt3R-SLAM [47]	✗	✗	0.089	0.085	0.087
Spann3R @20 [64]	✗	✗	0.069	0.047	0.058
Spann3R @2 [64]	✗	✗	0.124	**0.043**	0.084
MASt3R-SLAM [47]	✗	✓	0.068	0.045	0.056
VGGT-SLAM (Sim(3)) [42]	✗	✓	0.052	0.062	0.057
VGGT-SLAM ($\mathbb{SL}(4)$) [42]	✗	✓	0.052	0.058	**0.055**
SLAM-MER [Ours]	✓	✓	**0.034**	0.135	0.084

4.3. Densified 3D Reconstruction

Table **3** presents the 3D reconstruction evaluation on the 7-Scenes [57] dataset. It is important to note that our method generates a semi-dense point cloud that is anchored to the underlying 3D map-points. It means that if map-points and keyframes are updated with the adjustment module, the semi-dense points move accordingly. In contrast, prior methods typically stitch together independent dense point clouds using only the optimized camera poses and scales.

While this anchoring approach allows us to achieve high accuracy on the regions supported by map-points, it also means that semi-dense points in the areas with no supporting anchor points will not be generated. Consequently, although we expect strong performance in accuracy, **SLAM-MER** may not perform at par on completion and Chamfer distance, which reward full scene coverage.

From the results, we observe that **SLAM-MER** achieves a 35% improvement in accuracy over the best baseline.

5. Conclusion

We present **SLAM-MER**, a new real-time monocular visual SLAM pipeline that leverages spatio-temporal structure of the problem together with learning-based feed-forward point-cloud estimators. Our system explores spatio-temporal scene modeling for localization by combining (1) a buffer of previous frames for explicit short-term temporal continuity, and (2) a 3D cell-based spatial representation with rasterization. Our SLAM achieves real-time performance with a substantial increase in efficiency (low latency) over previous competitors while maintaining or improving localization accuracy. Our C++ implementation is highly modular, enabling easy integration of existing or future off-the-shelf components, and will be released to the community. In future, we plan to incorporate $\mathbb{SL}(4)$ pose representation [42], which looks promising for further improvements.

Acknowledgments

Valter was supported by Mitsubishi Electric Research Laboratories (MERL), LARSyS funding (DOI: 10.54499/LA/P/0083/2020), and Sustainable Stone by Portugal (co-financed by the Recovery and Resilience Plan of the European Union). Lalit and Pedro were exclusively supported by MERL. Masashi was exclusively supported by Mitsubishi Electric Corporation.

References

[1] Gabriele Berton and Carlo Masone. Megaloc: One retrieval to place them all. In *Conference on Computer Vision and Pattern Recognition Workshops*, pages 2886–2892, 2025. 6, 13, 14

[2] Mårten Björkman, Niklas Bergström, and Danica Kragic. Detecting, segmenting and tracking unknown objects using multi-label mrf inference. *Computer Vision and Image Understanding*, 118:111–127, 2014. 14

[3] Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. Robust visual inertial odometry using a direct EKF-based approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 298–304, 2015. 2

[4] G. Bradski. The OpenCV Library, 2000. 14, 16

[5] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM. *IEEE Transactions on Robotics (T-RO)*, 37(6):1874–1890, 2021. 1, 2, 4, 6, 7

[6] Luca Carlone, Ayoung Kim, Timothy Barfoot, Daniel Cremers, and Frank Dellaert. SLAM Handbook: From localization and mapping to spatial intelligence, 2025. 2

[7] David Caruso, Jakob Engel, and Daniel Cremers. Large-scale direct slam for omnidirectional cameras. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 141–148, 2015. 2

[8] Xingyu Chen, Yue Chen, Yuliang Xiu, Andreas Geiger, and Anpei Chen. TTT3R: 3D Reconstruction as test-time training. In *International Conference on Learning Representations (ICLR)*, 2026. 3

[9] Zhuoguang Chen, Minghui Qin, Tianyuan Yuan, Zhe Liu, and Hang Zhao. Long3R: Long sequence streaming 3D reconstruction. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5273–5284, 2025. 3

[10] Chong Cheng, Sicheng Yu, Zijian Wang, Yifan Zhou, and Hao Wang. Outdoor monocular slam with global scale-consistent 3D Gaussian pointmaps. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 26035–26044, 2025. 3

[11] Ondrej Chum and Jiri Matas. Locally optimized ransac. In *Joint Pattern Recognition Symposium*, 2003. 5

[12] Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, pages 1–2, 2004. 14

[13] Jan Czarnowski, Tristan Laidlow, Ronald Clark, and Andrew J. Davison. DeepFactors: Real-time probabilistic dense monocular SLAM. *IEEE Robotics and Automation Letters (RA-L)*, 5(2):721–728, 2020. 3

[14] A.J. Davison, I.D. Reid, N.D. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PMAI)*, 29(6):1052–1067, 2007. 1, 2

[15] Frank Dellaert and GTSAM Contributors. borglab/gtsam, 2022. 4

[16] Kai Deng, Zexin Ti, Jiawei Xu, Jian Yang, and Jin Xie. VGGT-Long: Chunk it, loop it, align it–pushing VGGT's limits on kilometer-scale long RGB sequences. *arXiv preprint arXiv:2507.16443*, 2025. 3

[17] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Conference on Computer Vision and Pattern Recognition Workshops*, pages 224–236, 2018. 13

[18] ONNX Runtime developers. Onnx runtime. `https://onnxruntime.ai/`, 2021. 4, 14

[19] Yaqing Ding, Jian Yang, Viktor Larsson, Carl Olsson, and Kalle Åström. Revisiting the P3P problem. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4872–4880, 2023. 5

[20] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *IEEE Transactions on Big Data*, 2025. 5

[21] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, pages 834–849, 2014. 2

[22] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 5

[23] Johannes Graeter, Alexander Wilczynski, and Martin Lauer. Limo: Lidar-monocular visual odometry. In *IEEE/RSJ International Conference on Intelligent*

Robots and Systems (IROS), pages 7872–7879, 2018. 1, 2

[24] Michael Grupp. evo: Python package for the evaluation of odometry and slam. `https://github.com/MichaelGrupp/evo`, 2017. 7

[25] Sebastian Haner and Anders Heyden. Covariance propagation and next best view planning for 3D reconstruction. In *European Conference on Computer Vision (ECCV)*, pages 545–556, 2012. 5

[26] Wonbong Jang, Philippe Weinzaepfel, Vincent Leroy, Lourdes Agapito, and Jerome Revaud. Pow3R: Empowering unconstrained 3D reconstruction with camera and scene priors. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1071–1081, 2025. 3

[27] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019. 5

[28] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research (IJRR)*, 31(2):216–235, 2012. 3, 4

[29] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):139–1, 2023. 2, 3

[30] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 225–234, 2007. 2

[31] Lukas Koestler, Nan Yang, Niclas Zeller, and Daniel Cremers. Tandem: Tracking and dense mapping in real-time using deep multi-view stereo. In *Conference on Robot Learning*, pages 34–45, 2022. 3

[32] Zuzana Kukelova, Jan Heller, and Andrew Fitzgibbon. Efficient intersection of three quadrics and applications in computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1799–1808, 2016. 5

[33] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951. 6

[34] Viktor Larsson and contributors. PoseLib - Minimal Solvers for Camera Pose Estimation, 2020. 5, 13

[35] Viktor Larsson, Zuzana Kukelova, and Yinqiang Zheng. Making minimal solvers for absolute pose estimation compact and robust. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2316–2324, 2017. 5

[36] Viktor Larsson, Torsten Sattler, Zuzana Kukelova, and Marc Pollefeys. Revisiting radial distortion absolute pose. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1062–1071, 2019. 5

[37] Vincent Leroy, Yohann Cabon, and Jérôme Revaud. Grounding image matching in 3D with MASt3R. In *European Conference on Computer Vision (ECCV)*, pages 71–91, 2024. 2, 3, 4, 14, 15

[38] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research (IJRR)*, 34(3):314–334, 2015. 2

[39] Zhengqi Li, Richard Tucker, Forrester Cole, Qianqian Wang, Linyi Jin, Vickie Ye, Angjoo Kanazawa, Aleksander Holynski, and Noah Snavely. MegaSaM: Accurate, fast and robust structure and motion from casual dynamic videos. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10486–10496, 2025. 3

[40] Lahav Lipson, Zachary Teed, and Jia Deng. Deep patch visual SLAM. In *European Conference on Computer Vision (ECCV)*, pages 424–440, 2024. 3, 7

[41] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004. 13

[42] Dominic Maggio, Hyungtae Lim, and Luca Carlone. Vggt-slam: Dense rgb slam optimized on the sl (4) manifold. *Advances in Neural Information Processing Systems (NeurIPS)*, 39, 2025. 1, 3, 4, 7, 8, 16, 17

[43] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, pages 405–421, 2020. 2

[44] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3565–3572, 2007. 1, 2

[45] Raul Mur-Artal and Juan D Tardós. ORB-SLAM2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics (T-RO)*, 33(5):1255–1262, 2017. 1, 2

[46] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. ORB-SLAM: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics (T-RO)*, 31(5):1147–1163, 2015. 2

[47] Riku Murai, Eric Dexheimer, and Andrew J. Davison. Mast3r-slam: Real-time dense slam with 3d reconstruction priors. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16695–16705, 2025. 1, 3, 4, 7, 8, 16, 17

[48] Joseph Ortiz, Alexander Clegg, Jing Dong, Edgar Sucar, David Novotny, Michael Zollhoefer, and Mustafa

Mukadam. iSDF: Real-time neural signed distance fields for robot perception. In *Robotics: Science and Systems (RSS)*, 2022. 3

[49] Martin Oswald, Matteo Poggi, Fabio Tosi, Youmin Zhang, Yiyi Liao, Vladimir Yugay, and Yue Li. NeuSLAM: Neural implicit representation for simultaneous localization and mapping. `https://sites.google.com/view/neuslam`, 2025. Accessed: 2025-11-05. 2

[50] Luigi Piccinelli, Christos Sakaridis, Mattia Segu, Yung-Hsu Yang, Siyuan Li, Wim Abbeloos, and Luc Van Gool. Unik3d: Universal camera monocular 3d estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1028–1039, 2025. 4

[51] Valter Piedade and Pedro Miraldo. Bansac: A dynamic bayesian network for adaptive sample consensus. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3738–3747, 2023. 5

[52] S.M. Pizer, R.E. Johnston, J.P. Ericksen, B.C. Yankaskas, and K.E. Muller. Contrast-limited adaptive histogram equalization: speed and effectiveness. In *Conference on Visualization in Biomedical Computing*, pages 337–345, 1990. 16

[53] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics (T-RO)*, 34 (4):1004–1020, 2018. 1, 2

[54] Antoni Rosinol, John J Leonard, and Luca Carlone. NeRF-SLAM: Real-time dense monocular slam with neural radiance fields. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3437–3444, 2023. 1, 3

[55] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2564–2571, 2011. 13

[56] Thomas Schops, Torsten Sattler, and Marc Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 134–144, 2019. 15, 16

[57] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2930–2937, 2013. 1, 7, 8, 15, 18

[58] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 573–580, 2012. 6, 7, 14, 15, 16, 17

[59] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J Davison. iMAP: Implicit mapping and positioning in real-time. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6229–6238, 2021. 3

[60] Zachary Teed and Jia Deng. DeepV2D: Video to depth with differentiable structure from motion. In *International Conference on Learning Representations (ICLR)*, 2020. 3

[61] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 16558–16569, 2021. 1, 3, 7, 8

[62] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PMAI)*, 13(4):376–380, 1991. 3, 7

[63] Ignacio Vizzo, Tiziano Guadagnino, Benedikt Mersch, Louis Wiesmann, Jens Behley, and Cyrill Stachniss. Kiss-icp: In defense of point-to-point icp–simple, accurate, and robust registration if done the right way. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):1029–1036, 2023. 1

[64] Hengyi Wang and Lourdes Agapito. 3D reconstruction with spatial memory. In *International Conference on 3D Vision*, pages 78–89, 2025. 3, 8

[65] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. VGGT: Visual geometry grounded transformer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5294–5306, 2025. 2, 3, 4, 14, 15

[66] Qianqian Wang, Yifei Zhang, Aleksander Holynski, Alexei A Efros, and Angjoo Kanazawa. Continuous 3D perception model with persistent state. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10510–10522, 2025. 3

[67] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. DUSt3R: Geometric 3D vision made easy. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20697–20709, 2024. 2, 3, 14, 15

[68] Yifan Wang, Jianjun Zhou, Haoyi Zhu, Wenzheng Chang, Yang Zhou, Zizun Li, Junyi Chen, Jiangmiao Pang, Chunhua Shen, and Tong He. π^3: Permutation-equivariant visual geometry learning. *arXiv preprint arXiv:2507.13347*, 2025. 3

[69] Tianci Wen, Zhiang Liu, and Yongchun Fang. Segs-slam: Structure-enhanced 3D Gaussian splatting slam with appearance embedding. In *IEEE/CVF Interna-*

tional Conference on Computer Vision (ICCV), pages 28103–28113, 2025. 3

[70] Chi Yan, Delin Qu, Dan Xu, Bin Zhao, Zhigang Wang, Dong Wang, and Xuelong Li. GS-SLAM: Dense visual SLAM with 3D Gaussian splatting. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19595–19604, 2024. 3

[71] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:21875–21911, 2024. 3

[72] Ji Zhang and Sanjiv Singh. Laser–visual–inertial odometry and mapping with high robustness and low drift. *Journal of Field Robotics*, 35(8):1242–1264, 2018. 1, 2

[73] Ji Zhang, Sanjiv Singh, et al. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems (RSS)*, pages 1–9, 2014. 1

[74] Youmin Zhang, Fabio Tosi, Stefano Mattoccia, and Matteo Poggi. GO-SLAM: Global optimization for consistent 3D instant reconstruction. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3727–3737, 2023. 3

[75] Xiaoming Zhao, Xingming Wu, Weihai Chen, Peter CY Chen, Qingsong Xu, and Zhengguo Li. ALIKED: A lighter keypoint and descriptor extraction network via deformable transformation. *IEEE Transactions on Instrumentation and Measurement*, 72:1–16, 2023. 4, 13, 14

[76] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R Oswald, and Marc Pollefeys. NICE-SLAM: Neural implicit scalable encoding for slam. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12786–12796, 2022. 3

[77] Zihan Zhu, Songyou Peng, Viktor Larsson, Zhaopeng Cui, Martin R. Oswald, Andreas Geiger, and Marc Pollefeys. NICER-SLAM: Neural implicit scene encoding for RGB SLAM. In *International Conference on 3D Vision*, pages 42–52, 2024. 1, 3, 7

Revisiting Monocular SLAM with Spatio-Temporal Scene Modeling

Supplementary Material

*These supplementary materials present: (1) details on the Kullback–Leibler divergence score used for keyframe creation; (2) implementation details; (3) experiments demonstrating the modularity of the proposed SLAM framework (***SLAM-MER***); (4) results on the ETH3D-SLAM dataset and further performance analysis; (5) ablation studies; (6) an explanation for the omission of the* `pumpkin` *sequence from 7-Scenes.*

A. Note on using KL Divergence

For the third rule for keyframe creation (Sec. 3.6 in the main paper), we use the Kullback–Leibler (KL) divergence, denoted $D_{KL}(\mathbf{H}_k||\mathbf{H}_{k-1})$, to compare histograms ($\mathbf{H}_k$) of counts of 3D map-points, seen by each keyframe, between two consecutive frames. With the two histograms of consecutive frames, $\mathbf{H}_k$ and $\mathbf{H}_{k-1}$, we add a small value $\epsilon = 10^{-12}$ to each bin in the histogram (to avoid zero divisions), normalize each one so that all bin values add to 1, and compute $D_{KL}(\mathbf{H}_k||\mathbf{H}_{k-1})$ as follows

$$D_{KL}(\mathbf{H}_k||\mathbf{H}_{k-1}) = \sum_i \mathbf{H}_k(i) \cdot \log\left(\frac{\mathbf{H}_k(i)}{\mathbf{H}_{k-1}(i)}\right). \quad \text{(A.1)}$$

The higher the value of $D_{KL}(\mathbf{H}_k||\mathbf{H}_{k-1})$, the more different the histograms are. We use this score to make a keyframe creation decision by identifying when the current frame suddenly observes a previously mapped area. This typically happens when the camera completes a loop. Fig. 4 of the main paper exemplifies the histogram behaviour we look for, in our third rule for keyframe creation.

B. Implementation Details

This section provides implementation details on the **SLAM-MER** pipeline, namely the settings used to obatin the experimental results reported in Sec. 4 of the main paper. The settings are as follows.

- **Keypoint extraction:** Maximum of 700 feature points using ALIKED [75];
- **Query map-points:** Map points are queried from the temporal query, whose buffer has a size of 100 frames, and from the spatial query. Each spatial query retieves a maximum of 200 cells;
- **3D-2D pose estimation:** Pose estimation is based on PoseLib [34] using default settings for RANSAC and final refinement. The estimation is considered valid if it has a minimum of 40 inliers and 20% inlier ratio;
- **Keyframe creation:** To create a keyframe, we set (1) the minimum number of inliers from pose estimation to 100, (2) the point spread score threshold to 0.35 and (3) the KL divergence threshold to 1. For every validated keyframe, we run MASt3R to obtain depth and filter 3D points with a confidence lower than 0.25;
- **Loop closure and relocalization:** We use MegaLoc [1] as the VPR approach to obtain the top 20 loop closure candidates, which are validated with inlier 3D-2D coorespondences in pose estimation.

C. Pipeline Modularity

In Sec. 3 of the main paper, we described our pipeline. Here, we showcase its modularity by utilizing different off-the-shelf methods for (1) keypoint extraction, (2) feed-forward geometry estimation, and (3) loop detection, and also discuss the impact of their specific choices.

C.1. Keypoint Extraction

Image keypoint extraction is the first step in the localization module (Sec. 3.3 in the main paper) of our pipeline when a new frame is received. Several keypoint extractors are available, typically categorized as either hand-crafted (*e.g.*, SIFT[41], ORB [55]) or learning-based (*e.g.*, SuperPoint [17], ALIKED [75]) approaches. Our modular pipeline can incorporate any keypoint extraction method; however, the choice of extractor can affect the accuracy and efficiency of the SLAM system, particularly within the localization module.

Table C.4. Performance comparison between two alternative methods for keypoints extraction: CudaSift and ALIKED. Results obtained for the 360 sequence of the TUM RGB-D dataset [58].

Method	Keypoint Extraction		Pose Estimation	
	# Keypoints	Time [ms]	% Inliers	Time [ms]
CudaSift [2]	418.2	2.02	54	1.84
ALIKED [75]	488.3	7.73	76	1.12

Table **C.4** presents an evaluation of the localization module—specifically the keypoint extraction and pose estimation steps—using the ALIKED and CudaSift [2] keypoint extractors. The results report the average (1) number of extracted keypoints, (2) keypoint extraction time, (3) percentage of pose estimation inliers, and (4) pose estimation runtime for the 360 sequence of the TUM RGB-D dataset [58]. For the keypoint extraction step, we set the maximum number of keypoints to 500. We observe that ALIKED reaches close to this target, whereas CudaSift produces considerably fewer keypoints. ALIKED also leads to more pose estimation inliers, indicating that keypoints are tracked more consistently. A higher number of inliers not only speeds up pose estimation but also improves point tracking against the map, thereby enhancing localization and reducing the need to create new keyframes. Beyond quantitative results, we also observe more stable keypoint tracking across consecutive frames when using ALIKED compared to CudaSift. The only downside of ALIKED (implemented in libTorch C++) compared to CudaSift (implemented in CUDA C++) is computational time, which is roughly $3\times$ longer, but still allows real-time capabilities.

C.2. Feed-Forward Depth Inference

During keyframe creation in the localization module (Sec. 3.3 in the main paper), our pipeline takes input from a monocular camera and uses a feed-forward depth inference method to obtain a point cloud. Several approaches have been proposed recently; here, we evaluate MASt3R [37], DUSt3R [67], and VGGT [65]. For each method, we exported the original model—together with the weights provided by the authors—to ONNX and ran inference in C++ using ONNXRuntime [18]. Similar to the keypoint extraction step, our pipeline can incorporate any feed-forward depth inference method; however, this choice has a significant impact on the system. Figure **C.6** presents results obtained using the three methods on two sequences.

We observe that MASt3R provides the overall best reconstruction results. We note that we use the same pipeline parameters for all reported experiments. The local 3D points produced by different feed-forward depth inference methods are scale-free—*i.e.*, they are not metric—as discussed in Sec. 3.3 of the main paper. Because different methods produce 3D points at different scales, our parameters could, in principle, be tuned to better match each approach. In this work, we chose to fine-tune all parameters for MASt3R because it is less computationally expensive than VGGT (as detailed below). After tuning for MASt3R, we fixed all parameters for the remainder of the experiments, including those shown here with alternative depth inference models. Our goal in this paper is to demonstrate the benefits of various design choices in our pipeline for real-time operation with accuracy comparable to current state-of-the-art methods. Additional parameter tuning for other depth inference approaches would not change the overall conclusions of the paper.

Regarding efficiency, there are notable differences among the three methods. MASt3R requires 112 ms for inference, while DUSt3R requires 98 ms; both use two images. For VGGT, inference takes 209 ms with one image, 481 ms with two images, 780 ms with three images, and 990 ms with four images. For VGGT, we use up to four images for inference.

Table C.5. Demonstration of the modularity of **SLAM-MER** for the VPR approach using MegaLoc and BoW. We report the average time to create an image descriptor, the loop detection average time, and the ATE and number of successfully closed loops (# Loops) for two sequences.

Method	Descriptor Creation [ms]	Detection Time [ms]	360		room	
			ATE	# Loops	ATE	# Loops
MegaLoc [1]	9	0.05	0.107	10	0.095	5
Bag-of-words [12]	2.5	0.10	0.097	7	0.115	3

C.3. Loop Detection

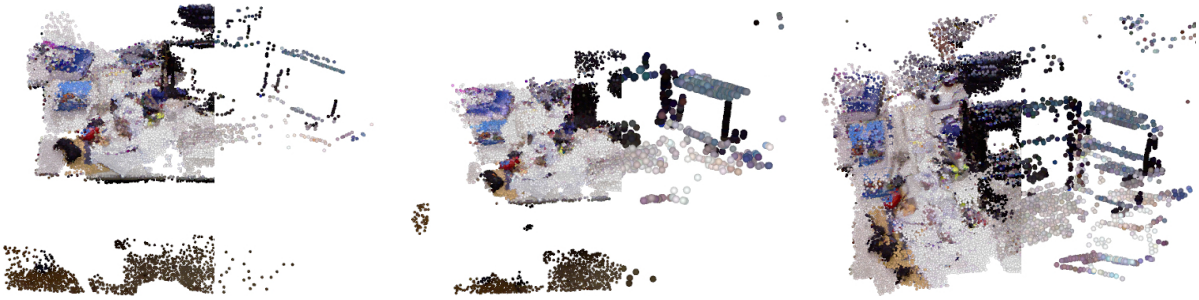
Next, we demonstrate the interchangeability of the loop detection approach in our pipeline (Sec. 3.7 of the main paper). The loop detection module uses a VPR method to identify the keyframes most similar to the current one. The module receives either a keyframe image or the keypoints detected in that image and outputs a list of the K-most similar keyframes, which we refer to as candidate keyframes. The better the VPR method performs, the more accurate the candidate keyframes will be, allowing more loops to be closed and resulting in a more accurate map.

We integrated the state-of-the-art MegaLoc [1] and the traditional Bag-of-Words (BoW) approach [12] into our pipeline. For MegaLoc, we exported the original model—together with the weights provided by the authors—to ONNX and ran inference in C++ using ONNXRuntime [18]. For BoW, we used the implementation available in OpenCV [4]. The BoW vocabulary was trained using the 360 sequence from the TUM RGB-D dataset [58]. Results from both approaches are reported in Table **C.5**.

Both BoW and MegaLoc require training. MegaLoc was

MASt3R DUSt3R VGGT

(a) `kitchen`, 7-Scenes dataset [57]



MASt3R DUSt3R VGGT

(b) `xyz`, TUM RGB-D dataset [58]

Figure C.6. Showcasing the modularity of our pipeline for the feed-forward depth inference. Our pipeline only runs it to generate local 3D points for every keyframe. This figure shows our hybrid reconstruction results using MASt3R [37], DUSt3R [67], and VGGT [65] for the `kitchen` and `xyz` sequences. In each image, we show the semi-dense 3D reconstruction (left) and the sparse map points used for localization (right). While any approach can be used, MASt3R gives the best results among the three.

trained on a wide range of datasets, making it generalizable across different scenes and ready to use off the shelf. BoW, on the other hand, requires a vocabulary to be built; this vocabulary is then used to create the image-level descriptor. We created the BoW vocabulary using the `360` sequence from TUM. We observe that BoW is faster than MegaLoc when creating the image-level descriptor. However, MegaLoc is faster when querying the database for similar descriptors and can close more loops overall. Since the loop closure module runs in parallel, efficiency becomes a concern only if one method is significantly slower than the other, which is not the case here. We use MegaLoc as the default, as it generalizes well across scenes and closes more loops. Additional VPR methods can be easily integrated into our pipeline.

C.4. Input Sensor

Another flexibility provided by our pipeline is the ability to use different types of vision sensors—either sensors that output depth directly, such as RGB-D cameras, or sensors that can infer depth indirectly, such as stereo RGB systems. Using sensors that provide depth significantly reduces the computational burden of our pipeline, since no depth estimation is required when a new keyframe is created. In this case, the depth is obtained simply by reading the sensor data.

Figure **D.7** shows results using our framework—without any modifications (including the parameters)—running visual SLAM with RGB-D sensors, directly using the raw depth information from the TUM RGB-D and 7-Scenes datasets (both of which provide RGB-D data). We highlight that our pipeline creates a set of local 3D points for each keyframe with the depth information provided by the sensor, and does not directly consider them as the map-points. This inherently allows some noise tolerance across different sensors, which is corrected by the adjustment module of our pipeline when we create new 3D map-points from these local 3D points. Our pipeline can work with any sensor that provides depth information along with an RGB image, without any change in the core modules, demonstrating the modularity across multiple sensors.

D. Additional Results

Sec. 4 of the main paper provides results with the 7-Scenes [57] and TUM RGB-D [58] datasets. This section complements it by providing (1) results for the ETH3D-SLAM [56] dataset in Sec. **D.1**, (2) a runtime breakdown of each module of the SLAM pipeline in Sec. **D.2**, and (3) a memory footprint analysis in Sec. **D.3**.

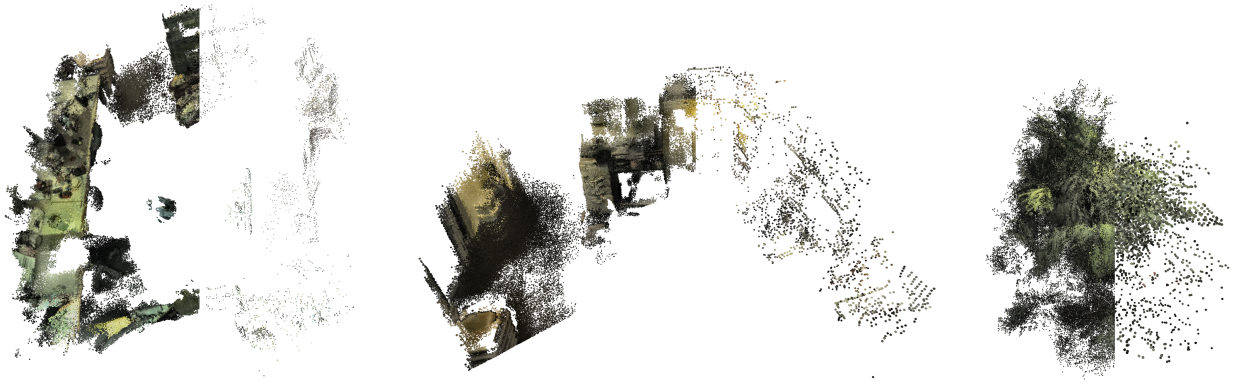


Figure C.8. Visualization of point clouds of ETH3D-SLAM [56] sequences, obtained with our pipeline. Each image shows the semi-dense 3D reconstruction (left) and the sparse 3D map-points used for localization (right).

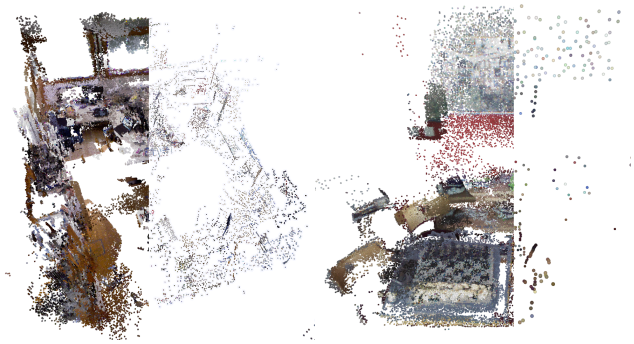


Figure D.7. Examples from the TUM and 7-Scenes sequences (`room` and `chess`), using RGB-D depth instead of depth predicted by a feed-forward model.

D.1. ETH3D-SLAM dataset

This subsection provides results on one additional dataset, ETH3D-SLAM [56]. We note that the ETH3D-SLAM dataset contains sequences that were not taken under ambient light conditions; all sequences were obtained in dark environments with minimal lighting and low texture, more suitable for RGB-D methods. Since our pipeline relies on keypoint extractors, which often fail to provide a good descriptor under these conditions, we perform the Contrast Limited Adaptive Histogram Equalization (CLAHE) [52] on the input images, as an additional preprocessing step only on this dataset to obtain reliable keypoint descriptors. We use OpenCV's implementation [4] for our experiments. Still, many sequences are very challenging for keypoint-based methods, with very aggressive changes in lighting conditions–the matching between descriptors from CudaSift and ALIKED fails, which means that the entire SLAM tends to fails. In these scenarios, a 3D-to-3D registration approach for camera localization, such as MASt3R-SLAM or VGGT-SLAM, is more appropriate, although it is significantly slower.

In Fig. **C.8**, we visualize point clouds (both sparse and dense) on sequences of the ETH3D-SLAM dataset. It can be seen that our pipeline provides good-quality point clouds even in these datasets with challenging lighting conditions.

Table D.6. Runtime breakdown per module of the SLAM pipeline in TUM RGB-D [58]. *w/o keyframe creation; **detecting and closing all loops per keyframe.

Module	Avg. Runtime [ms]	
	`desk`	`room`
Localization*	11.8	11.9
Keyframe creation	80.0	79.4
Covisibility graph	3.04	4.50
Loop closure**	14.1	20.9

D.2. Runtime Breakdown

Table **D.6** shows a runtime breakdown per module of the SLAM pipeline in two TUM RGB-D [58] sequences. The runtimes for Localization, Keyframe creation, and Covisibility graph are similar for both sequences. Localization does not include the runtime for creating keyframes, since it runs as a parallel process, and Localization does not wait for it to end. We only observe a change in runtime for Loop closure, which is expected since `room` is a larger sequence with a long range loop unlike `desk`.

D.3. Memory Footprint

Table **D.7** provides a memory footprint comparison betwen MASt3R-SLAM [47], VGGT-SLAM [42] and **SLAM-MER** in terms of the size of the map each approach builds. Since MASt3R-SLAM and VGGT-SLAM do not have the same concept of map-points (dense approaches) and do

Table D.7. Memory footprint comparison agaisnt MASt3R-SLAM [47] and VGGT-SLAM [42] in TUM RGB-D [58].

Map Components	MASt3R-SLAM		VGGT-SLAM		SLAM-MER [Ours]	
	desk	room	desk	room	desk	room
# Map-points	N/A	N/A	N/A	N/A	$8k$	$31k$
# Keyframes	12	53	57	197	16	74
# Cells	N/A	N/A	N/A	N/A	90	773

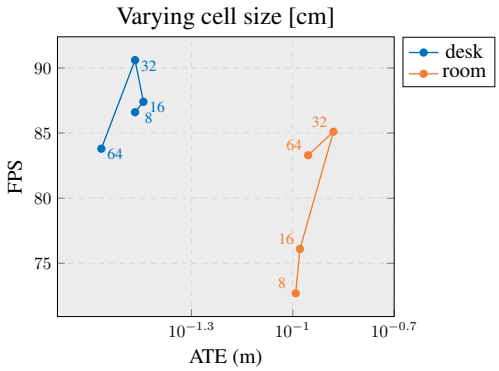


Figure E.9. Ablation study on the size of the cells (annotated next to each marker) used for spatially querying 3D points.

not create cells, the main comparison aspect is the number of created keyframes. **SLAM-MER** creates much fewer keyframes than VGGT-SLAM and slightly more than MASt3R-SLAM.

To manage memory footprint as map size increases, we (1) delete duplicated map points after merging observations in loop closure and (2) ignore empty cells for spatio-temporal querying of 3D points. We do not cull keyframes since they are created only when new map regions are visited. Moreover, culling is not required because our system avoids creating keyframes in previously mapped areas due to the spatio-temporal localization design and keyframe creation criteria.

E. Ablation Studies

In Sec. 4.1 of the main paper, we presented an ablation study on the temporal buffer size and on the cell dimensions used to query 3D points. Here, we provide three additional ablation studies. The first, in Sec. **E.1**, analyzes the performance of our **SLAM-MER** pipeline for different cell dimensions. The second, in Sec. **E.2**, analyzes the keyframe creation criteria. The final ablation, in Sec. **E.3** study analyzes dense reconstruction results for different numbers of anchor points and points per anchor.

E.1. Cell Dimensions

In the ablation study presented in Sec. 4.1 of the main paper, which analyzes the effect of cell size on the spatial querying of 3D points, we observed that increasing the cell size from 8 to 32 cm slightly increases the trajectory error but significantly improves the FPS. Based on those results and our goal of maximizing efficiency, we set the cell size to 32 cm for all of our experiments.

Although the cell-related ablation in the main paper focuses solely on spatial querying, here we provide additional results in which the cell size is varied while also utilizing the temporal query ($|\mathcal{B}|= 100$) and loop closure. Results are shown in Fig. **E.9**. When cell size inceases beyond 32 cm, we observe a decrease in FPS. This occurs because larger cells cause more 3D map-points to be selected during the spatial query–including points that are not in the camera's current field of view. This increases the time required to establish 3D–2D correspondences and raises the likelihood of outliers, which in turn degrades the performance of camera pose estimation.

Furthermore, we rasterize per cell to determine whether a cell lies within the camera's view. With larger cells, we may encounter situations where many visible points are discarded due to occlusions affecting entire cells.

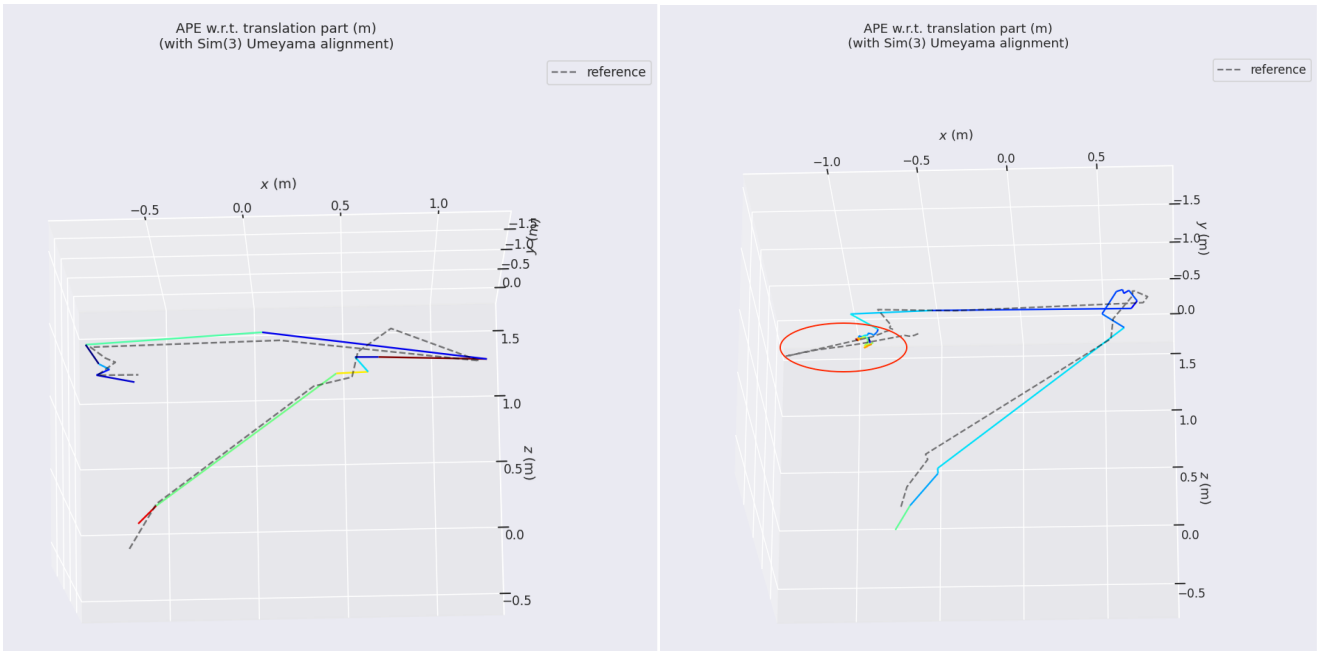
Table E.8. Ablation study on keyframe creation criteria in TUM RGB-D [58].

# Tracked Map-Points	Points Spread	D_{KL} Score	desk				room			
			ATE	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{P}^w\rvert$	FPS	ATE	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{P}^w\rvert$	FPS
✓	✗	✗	0.041	17	4796	92.1	0.119	63	16764	75.2
✓	✓	✗	0.039	18	4871	92.4	0.107	63	16642	78.6
✓	✗	✓	0.039	18	4973	93.3	0.112	66	16927	78.8
✓	✓	✓	0.038	19	5195	89.3	0.096	68	17377	78.6

E.2. Keyframe Creation

Sec. 3.6 of the main paper detailed the three keyframe creation rules we utilize in our pipeline. To summarize, the first rule ensures we create a keyframe when the number of inliers from pose estimation is low, *i.e.*, the current frame does not track enough 3D map-points. We refer to this criterion as the number of tracked map-points (# tracked map-points). The second rule checks how well the tracked map-points are spread across the image. When map-points are not well spread, we create a new keyframe. The final rule compares the distribution of the tracked map-points across keyframes. Keyframes are created when there is a sudden variation in consecutive frames of the keyframes that observe the tracked map-points. To measure this sudden variation, we compute the Kullback–Leibler divergence D_{KL} of the histogram of the number of tracked 3D points per keyframe.

Table **E.8** compares the performance of combinations of the keyframe creation rules. The rule based on the number of tracked map-points is always switched on for the experiments, because the lack of it results in pose estimation being more prone to failure due to less coverage of 3D map-points in different scene parts. Thus, we can easily lose the camera



(a) Trajectory obtained using MASt3R-SLAM

(b) Trajectory obtained using SLAM-MER

Figure E.10. Issue in the ground truth trajectory (dashed line) of the `pumpkin` sequence from 7-Scenes. The red ellipse highlights the part of the trajectory that does not match the actual camera motion–as seen from the input frames. We note that the ground truth trajectory shown here only consists of the frames used for evaluation.

Table E.9. Ablation study on semi-dense reconstruction results. We vary the number of anchor points (*i.e.*, map points) and the number of nearby points sampled per anchor on the 7-Scenes dataset [57]. Our default setting is highlighted.

# Anchors	# Points Per Anchor	Acc. ↓	Comp. ↓	Chamfer ↓
$4k$	20	0.034	0.135	0.084
$4k$	5	0.034	0.145	0.090
$4k$	60	0.033	0.097	0.065
$8k$	20	0.036	0.132	0.084

pose, triggering the re-localization operation mode more often. The remaining two rules are switched on and off in turn. We observe that using the point spread and the KL divergence score slightly increases the number of keyframes in both sequences we tested on. This is exactly the behaviour we expected from these criteria, *i.e.*, create keyframes not only when needed because there are few points but also based on the (1) spread of the 2D locations of tracked map-points, and (2) distribution per keyframe of the points being tracked (D_{KL} score). Besides creating more keyframes, using these criteria reduces trajectory error, and FPS differences are not significant.

E.3. Dense Reconstruction

Sec. 3.9 of the main paper details how we obtain a semi-dense scene representation from sparse map points. The main paper also shows that the sparse map points lead to a good reconstruction accuracy but lack scene completion since we rely on the map points for anchoring the semi-dense representation. In this experiment, we enforce the creation of additional map points indirectly by extracting more features from the input images, and vary the number of points sampled per anchor point. Results are shown in Tab. **E.9**. We can observe that (1) increasing the number of anchor points only leads to a marginal variation in completion and accuracy results, and (2) the number of points per anchor higly affects completion. Additionaly, increasing the number of points per anchor has a negligible effect on FPS of SLAM since they are sampled during keyframe creation and updated in the Adjustment module, which runs in parallel to the SLAM main thread.

F. Discussion on 7-Scenes: `pumpkin` sequence

In Tab. 2b of the main paper, we mentioned that we omitted the `pumpkin` sequence from the 7-Scenes dataset for evaluation. During the experiments on the `pumpkin` sequence, we noticed that a part of the ground truth trajectory seemed grossly incorrect–please see Fig. **E.10**. In the final part of the sequence, the camera undergoes an almost pure rotation, which is consistent with the trajectory recovered by both our method Fig. **E.10b** and MASt3R-SLAM Fig. **E.10a**. However, the reference ground-truth trajectory from the dataset

shows a large jump of almost 1 meter in the negative x direction, which is not consistent with the camera motion visible in the images. This is evident from the ground truth trajectory shown in the red ellipse in Fig. **E.10b**.

This issue appears to be related to drift at specific timestamps–note that this is an unrealistically large motion for a single keyframe, based on the dataset creation strategy publicly available. We believe MASt3R-SLAM does not show this artifact because, unlike our pipeline that takes all frames as input, MASt3R-SLAM skips frames to maintain an input rate of around 10 FPS.