# Deep Patch Visual SLAM

Lahav Lipson, Zachary Teed, and Jia Deng

Princeton University

**Abstract.** Recent work in Visual Odometry and SLAM has shown the effectiveness of using deep network backbones. Despite excellent accuracy, such approaches are often expensive to run or do not generalize well zero-shot. To address this problem, we introduce Deep Patch Visual-SLAM, a new system for monocular visual SLAM based on the DPVO visual odometry system. We introduce two loop closure mechanisms which significantly improve the accuracy with minimal runtime and memory overhead. On real-world datasets, DPV-SLAM runs at 1x-3x real-time framerates. We achieve comparable accuracy to DROID-SLAM on EuRoC and TartanAir while running twice as fast using a third of the VRAM. We also outperform DROID-SLAM by large margins on KITTI. As DPV-SLAM is an extension to DPVO, its code can be found in the same repository: `https://github.com/princeton-vl/DPVO`

**Keywords:** SLAM · Visual Odometry · Monocular

## 1 Introduction

Visual Simultaneous Localization and Mapping (SLAM) aims to estimate camera motion and produce a 3D environment map from a video stream. Most approaches treat SLAM as an optimization problem which seeks to align the unknown poses and depth to the visual measurements. Indirect approaches to SLAM estimate matches between frames as a pre-processing step and then minimize the reprojection error during bundle adjustment [2, 16, 17]. DROID-SLAM [31], a recently-proposed indirect method for SLAM with exceptional accuracy, alternates between predicting optical flow residuals and optimizing the camera pose and depth estimates. DROID-SLAM uses a deep-network backbone based on RAFT [29], and generalizes well to multiple datasets without ever having seen real-world data during training. Several other approaches have since been based on this design [31, 41, 45].

Despite DROID-SLAM's success, it requires a GPU with 24GB of memory to run on all datasets. This makes deploying DROID-SLAM on portable hardware challenging, which is required for many applications of SLAM. DROID-SLAM is also slower than several classical approaches, and requires a high-end GPU in order to run in real-time. To address these issues, a sparse analog of DROID-SLAM, Deep Patch Visual Odometry (DPVO [31]), was later introduced. DPVO only tracks a small number of keypoints (64-96) per-frame, as opposed to the dense correspondence that DROID-SLAM used.

DPVO is efficient, but it has several weaknesses. First, it is limited to a visual odometry system due to the high storage cost imposed by its use of high-resolution feature maps. Second, DPVO suffers from severe scale-drift outdoors.

To address these challenges, we propose DPV-SLAM, a new system for monocular visual SLAM with improved efficiency and accuracy. DPV-SLAM is based on the DPVO visual odometry system; our primary contributions are two separate SLAM loop closure mechanisms which improve accuracy across several domains, with only small penalties to inference speed and memory usage.

- **DPV-SLAM is general and robust.** We evaluate our approach on EuRoC, KITTI, TUM-RGBD and TartanAir. Our method performs well in all settings, suffering from 0 catastrophic failures. We compare DPV-SLAM to other methods which report results on both indoor/outdoor without re-training, and show that our method has low average error in all settings.
- **DPV-SLAM is fast.** DPV-SLAM runs 2.5x faster than DROID-SLAM on EuRoC and 2.3x faster on KITTI. Compared to the base DPVO system, we incur only a small reduction in speed (e.g., 60→50 FPS on [1]) and increase in cost (4G→5G GPU memory).
- **DPV-SLAM is accurate.** We perform similarly to DROID-SLAM on EuRoC and TartanAir. Compared to DPVO, we achieve 4.5x lower error on EuRoC (0.105→0.023). On KITTI, we outperform DROID-SLAM and DPVO by significant margins.

To construct DPV-SLAM, we introduce an efficient mechanism for proximity-based loop closure and an image-retrieval backend based on classical descriptors. Our proximity backend addresses a challenge with building SLAM systems on deep networks, which is their inability to run the backend and frontend in parallel. For example, previous systems [30, 45] run their respective backends in separate processes which compete for GPU resources if run on the same device. Without a second GPU, their odometry systems must pause periodically and wait for the backend to finish executing.

Our proximity backend runs on a single GPU, at low memory cost and low latency. We optimize and update a scene graph with both odometry and loop closure factors. To enable efficient global optimization, we contribute a CUDA-accelerated block-sparse implementation of bundle adjustment which is compatible with DPVO's patch graph scene representation. Our proximity-based loop closure runs considerably faster DROID-SLAM's backend on EuRoC [1] (0.1-0.18s vs 0.5-5s). We also extend DPV-SLAM to utilize a classical loop closure mechanism, which employs image retrieval and pose graph optimization to correct for scale drift.

## 2 Related Work

**Zero-Shot Cross-Domain Generalization** is a longstanding problem in visual SLAM. The challenge is to develop a system which avoids catastrophic failures in different domains, without requiring re-training. Classical approaches

to SLAM [2,16,17] are prone to catastrophic failures during fast camera motion, and generally underperform deep approaches on indoor datasets [5, 10]. Several works have proposed learning a generalized system for SLAM by using a deep-network backbone trained entirely on synthetic data. TartanVO [35] trained on TartanAir [36] and showed strong performance on both indoor and outdoor settings without fine tuning. DROID-SLAM [30] and DPVO [31] followed a similar approach, but used a differentiable bundle adjustment layer in order to learn outlier rejection by supervising on the predicted camera poses.

Our approach is also trained only on synthetic data, however we demonstrate better generalization and/or runtime. Many works in VO/SLAM focus on in-domain accuracy, i.e., approaches trained or developed with a particular test setting taken into special consideration [14, 22, 34, 37, 39, 40, 42–44]. This setting is orthogonal to ours; we do not claim to outperform VO/SLAM systems specialized for autonomous driving applications on the KITTI dataset, for example.

**Monocular SLAM** is especially challenging due to the ambiguity of scale in monocular video. Several VO/SLAM works remove the scale ambiguity altogether by relying on stereo video, inertial measurements, or depth [8, 20, 21]. In contrast, our method operates on monocular video. We focus our evaluation on methods which do the same. Monocular SLAM is important due to the wide availability of monocular video, and because they can be easily adapted to use additional sensors, whereas the reverse is not always true.

**Neural SLAM and rendering-focused SLAM**: Several recent approaches use Gaussian-splatting and/or NeRFs [12, 13, 27, 47]. These rendering-based approaches are primarily designed for high-quality reconstruction/rendering, with tracking being a secondary focus usually only evaluated with smooth/slow camera motion on [25]/ [4]. In contrast, we focus on tracking accuracy in hard settings, similar to [2, 5, 16, 17, 30, 31]. [45] is an exception, which we compare to.

**Loop Closure** enables VO/SLAM methods to correct drift by adding factors between temporally-distant pose variables. Campos et al. [2] categorized the types of loop closure as *mid-term* and *long-term* data-association based on their approach to detecting loops and optimizing the scene graph. *mid-term* loop closure uses the current estimate of poses and depth to detect loops, and updates the poses/depth using bundle adjustment. *long-term* data association uses visual place recognition to detect loops and updates the poses using pose graph optimization. DROID-SLAM [30] uses mid-term. LDSO [10] uses long-term. VO systems use neither (by definition). ORB-SLAM [2] uses both. DPV-SLAM uses mid-term, or optionally both.

**Deep Patch Visual Odometry (DPVO)** was proposed as a faster alternative to the visual odometry from DROID, based on two insights. The first is that virtually every approach to SLAM provides some mechanism to trade-off accuracy for speed and/or memory (to an extent). For example, one can increase the number of keypoints, RANSAC iterations, the optimization window size, the image resolution, the number of keyframes produced, or the connectivity of the factor graph. For SLAM methods with deep-network backbones, one can also increase the feature dimension, add more layers, or use quantization / mixed-precision.

The second insight is that, by predicting sparse optical flow as opposed to dense, the resulting memory/runtime savings are sufficient to offset the initial accuracy loss by *spending* them in other aspects of the design. This allows DPVO to achieve similar accuracy to DROID-SLAM's frontend, with much lower cost and faster inference. The drawback is that the DPVO design is more challenging to adapt to a full SLAM system due to the large per-frame storage requirement. DPVO also suffers from the same performance issues as DROID-SLAM on outdoor datasets.

## 3 Approach

### 3.1 Goals for a general SLAM system

In this work, we aim to construct a general approach to monocular visual-SLAM which is maximally useful for estimating camera-motion across domains. To this end, we clearly define our goals:

- **Zero-shot generalization**. We aim for DPV-SLAM to generalize across domains (outdoor and indoor), without requiring re-training or fine-tuning on data from the testing domain. Recent works based on deep-network backbones [30,35] have shown strong zero-shot generalization by training on synthetic data [36]. DPV-SLAM does this as well, but generalizes better. Likewise, classical approaches to VO/SLAM [2, 5, 10, 16, 17] are also considered "general" due to their use of classical feature-descriptors/detectors [19,23] instead of learned ones. In contrast, many recent works in VO/SLAM [14, 22, 34,37,39,40,42–44] do not demonstrate the ability to generalize between domains (outdoor/indoor) without re-training/tuning on domain-specific data.
- **Fast (real-time or better) inference.** We aim for DPV-SLAM to be fast. A significant application of SLAM is to provide real-time feedback to autonomous systems. For these applications, it is critical that the method can process images at a rate equal to or better than the camera hz. Many classical approaches [2, 5, 10, 16, 17], as well as some deep ones [30, 41, 45], slow down significantly during fast motion due to more frequent keyframing.
- **Low-cost.** We aim for DPV-SLAM to be low-cost. Many applications for visual SLAM require deploying the algorithm on portable hardware. The resources on such hardware are often limited. To this end, we show that DPV-SLAM uses minimal GPU memory.

### 3.2 DPVO Preliminaries

Our system is based on Deep Patch Visual Odometry (DPVO) [31]. DPVO is a sparse analog of the visual odometry frontend of DROID-SLAM which achieves similar accuracy with much lower latency and memory. In this section, we will discuss the details of DPVO that are relevant to our contribution. For more details, we refer the readers to the original DPVO paper.

**DPVO Overview:** Given an input video stream, DPVO seeks to estimate the 2D motion of selected keypoints across time by predicting optical flow and updating the depth and camera poses using bundle adjustment. DPVO only supports visual odometry, so it operates on a sliding window of frames and removes keyframes and features once they fall outside of the optimization window.

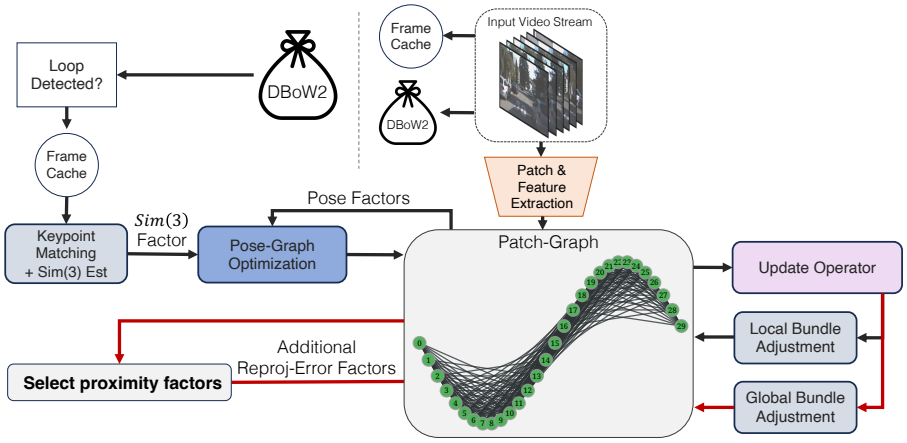



**Fig. 1:** Overview of DPV-SLAM. Our system maintains the odometry system from DPVO [31] as a frontend, and introduces efficient loop closure mechanisms. Like DPVO, our system utilizes a patch graph scene representation, which alternates between predicting sparse optical flow residuals and optimizing the camera poses and depth using bundle adjustment. DPV-SLAM detects previously visited locations and attempts to correct the accumulated drift via loop closure. The proximity loop closure detects loops using the pre-estimated geometry and uses global bundle adjustment. The second uses image retrieval and pose graph optimization.

**The patch graph:** DPVO uses a scene representation known as a patch graph, in which each frame $i$ contains a set of $p \times p$ patches $\mathbf{P}_{ik}$

$$\mathbf{P}_{ik} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{1} \\ \mathbf{d} \end{pmatrix} \qquad \mathbf{x}, \mathbf{y}, \mathbf{d} \in \mathbb{R}^{1 \times p^2} \tag{1}$$

where $\mathbf{d}$ is the inverse depth estimate. We denote the number of patches for frame $i$ as $K_i$, and $[K_i] := \{1, ..., K_i\}$. The patch graph is a bipartite graph, in which directed edges connect patches to frames. The scene representation is used by DPVO and DPV-SLAM. We visualize an example of our backend patch graph in Fig. 3.

Given the current poses, inverse depths and camera intrinsics, we can reproject any patch to any other frame. We denote the set of edges as $F$, the global

camera pose for frame $i$ as $G_i$, and the 3D→2D pinhole-projection function as $\pi(\cdot)$. We represent edges from $\mathbf{P}_{ik}$ to frame $j$ as $(i, k, j)$. The reprojection of $\mathbf{P}_{ik}$ to frame $j$ is denoted as

$$\mathbf{P}'_{ikj} = \pi[G_j^{-1} \cdot G_i \cdot \pi^{-1}(\mathbf{P}_{ik})] \tag{2}$$

The explicit objective of DPVO is to predict residual updates $\Delta_{ikj}$ to $\mathbf{P}'_{ikj}$ for all edges in order to improve the visual alignment. The resulting $\mathcal{I}_{ikj} := (\mathbf{P}'_{ikj} + \Delta_{ikj})$ represents the model's *ideal* reprojection of $\mathbf{P}_{ik}$ into frame $j$. After predicting $\Delta_{ikj}$, DPVO solves an optimization problem (BA) over the patch graph to align the actual patch reprojections to the ideal reprojections.

$$\underset{G,\mathbf{d}}{\arg\min} \sum_i \sum_{k \in [K_i]} \sum_{j:(i,k,j)\in F} \left\| \pi[G_j^{-1} \cdot G_i \cdot \pi^{-1}(\mathbf{P}_{ik})] - \mathcal{I}_{ikj} \right\|^2_{\Sigma_{ikj}} \tag{3}$$

Note that in eq. 3, $\mathcal{I}_{ikj}$ is treated as a constant. In addition to $\Delta_{ij}$, DPVO also predicts a confidence estimate $w_{ikj} \in \mathbb{R}^2$ for each edge. Eq. 3 minimizes the Mahalanobis distance, in which the error terms are weighted by the predicted confidences: $\Sigma_{ikj} = diag(w_{ikj})$.

**Patch extraction:** DPVO selects keypoints randomly, as opposed to the usual strategy of using a detector or image-gradients. This counter-intuitive strategy works sufficiently well [31], and is trivial to implement. The patch features are cropped around the chosen 2D keypoints from dense $({}^{H}/_{4} \times {}^{W}/_{4})$ feature maps predicted by a residual network learned end-to-end with the full model. DPVO extracts both $1\times1$ context features, and $p\times p$ correlation features. The correlation features are used to evaluate the visual alignment of the current pose and depth estimates, whereas the context features are provided as-is to the update operator.

**Update Operator:** The update operator of DPVO is the recurrent module used to predict $\Delta_{ikj}$ and $w_{ikj}$ for all edges in the patch graph. As an RNN, it also maintains a running hidden state $h_{ikj} \in \mathbb{R}^{384}$ for every edge $(i, k, j)$. The architecture includes several fully-connected gated residual units. As input, the update operator accepts the previous hidden state $h_{ikj}$, correlation features $\mathbf{C}_{ikj}$, and the context features for $\mathbf{P}_{ik}$.

**Patch correlation:** Correlation features $\mathbf{C}_{ikj}$ are computed for each edge in order to evaluate the visual alignment produced by the current depth and pose estimates. To compute $\mathbf{C}$, we use eq. 2 to reproject $\mathbf{P}_{ik}$ into frame $j$. Let $\mathbf{g}(u, v)$ represent the $p\times p$ patch correlation features indexed at $(u, v)$ and $\mathbf{P}'(u, v)$ denote $\mathbf{P}'_{ijk}$ indexed at $(u, v)$. $f(\cdot)$ denotes bilinear sampling, $\langle\cdot\rangle$ a dot product, and $\Delta_{\alpha\beta}$ a $7 \times 7$ grid centered at 0, indexed at $(\alpha, \beta)$. Each value in $\mathbf{C} \in \mathbb{R}^{p\times p\times 7\times 7}$ is computed as:

$$\mathbf{C}(u, v, \alpha, \beta) = \langle \mathbf{g}(u, v),\ \mathbf{f}(\mathbf{P}'(u, v) + \Delta_{\alpha\beta}) \rangle \tag{4}$$

$\mathbf{C}$ is recomputed and flattened before each invocation of the update operator. Colloquially, eq. 4 computes a dot product between correlation features for all pairs of grid points around either end of the flow vector induced from the poses

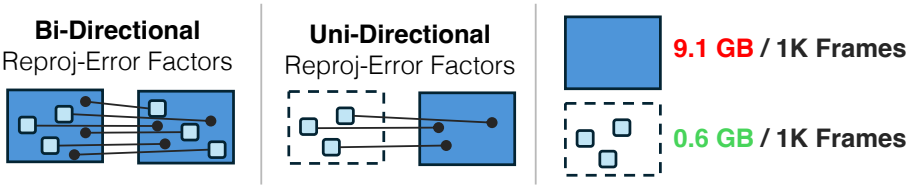



**Fig. 2:** Our optimization layer is composed of directed reprojection-error factors which constrain the depth of the outgoing frame, and both camera poses. The key insight is that, for each directed edge in the patch graph, the outgoing frame incurs a tiny storage cost and the incoming frame incurs a large cost, yet this factor sufficiently constrains both camera poses in the optimization. This is a special property of DPVO's patch representation, since outgoing flow vectors only require patch features. We leverage this fact to introduce long-range edges for loop closure with minimal cost.

and $\mathbf{P}_k$. The key insight is that eq. 4 requires storing the full correlation feature map for frame $j$ in memory, since $\mathbf{P}'_{ikj}$ is unbounded. This is an expensive requirement, which is addressed in our proximity backend.

### 3.3 Proximity Loop Closure

We contribute a loop closure mechanism to DPVO which detects loops via the camera's proximity to previously visited locations. This backend seeks to improve global consistency by periodically inserting long-range edges into the patch graph, updating their optical flow, and performing global bundle adjustment to update all camera poses and depth. Previous deep SLAM systems [30,45] require two GPUs in order to consistently average real-time inference. This is because CUDA operations typically utilize all available cores on their host device, and therefore must run sequentially, even when they are being called from separate processes. All Pytorch/CUDA operations in DPV-SLAM run in a single process on a single device, which is both simpler and computationally cheaper.

**Constructing the Patch Graph:** DPVO, like DROID-SLAM [30], stores dense feature maps in memory for updating the optical flow predictions. The larger the optimization window, the more feature maps must be kept, increasing cost. This problem is exacerbated in DPVO, where the feature maps are twice the spatial resolution ($1/4$ vs $1/8$). Note that this problem does not affect keypoint-based indirect SLAM methods like ORB-SLAM [2], only those based on RAFT [29].

We observe that, for each directed edge in the patch graph, the correlation operator consumes dense features only for the destination frame and only small patch features for the source frame. Meanwhile, the associated reprojection-error factor in the optimization is able to influence the camera poses for *both* frames. This means, for each edge in the patch graph, we can arbitrarily flip its direction to influence which of the two frames hosts the patch features and which the dense feature map, without significantly affecting the optimization result.

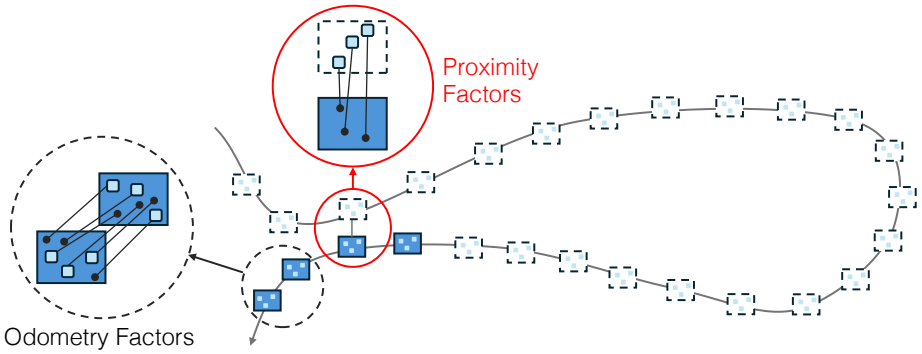



**Fig. 3:** The patch graph for our DPV-SLAM. We introduce directed factors from old frames to recent frames in the odometry frontend. These factors are chosen based on the frontend's *proximity* to previously visited locations. The construction of this patch graph only requires storing a finite number of dense feature maps, keeping the memory consumption small.

We exploit this fact to minimize the number of deep features stored in memory. Specifically, we permanently store only the patch features for all previous time-steps and create uni-directional proximity factors connecting these patches to recently observed frames in the frontend. Consequently, we incur only a minor storage overhead from the patch features ($\approx$ 0.6Gb / 1K frames). We depict this globally-connected patch graph in Fig. 3.

**Efficient Global Optimization:** We mix both odometry and loop-closure factors in the same optimization. This mandates a bundle adjustment implementation which can handle global optimization without severely impacting the odometry component. DPVO's pre-existing bundle adjustment is GPU-accelerated, but is still inefficient for large, sparse optimization problems. This is precisely the challenge we face when introducing a small number of long-range proximity factors into the optimization. A viable solution is to leverage the sparse structure of the hessian by implementing bundle adjustment in CUDA with block sparse representations. While prior work has implemented such a system for dense, uniformly sized depth maps [30], such a system has not been built for sparse, varying sized patch graphs. We contribute our implementation and use it to perform efficient global optimization.

### 3.4 Classical Loop Closure

Separately from our proximity loop closure, we also support a more traditional SLAM backend which uses classical image retrieval and pose graph optimization. This loop closure is especially important for correcting large amounts of drift. We refer to the variant of our model with both proximity and classical loop closure as DPV-SLAM++.

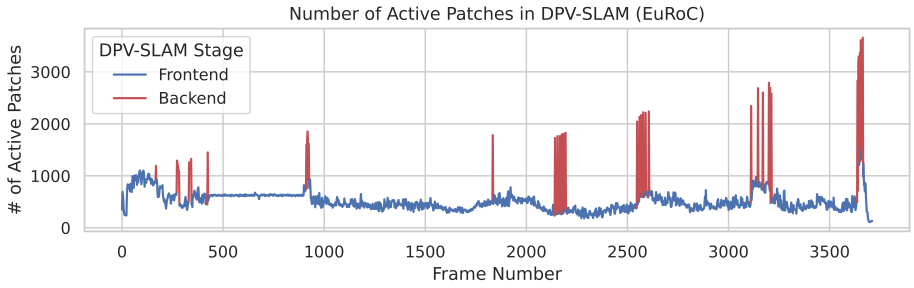



**Fig. 4:** We visualize the number of patches participating the optimization, over the coarse of a video. During invocations of our proximity backend, we perform global bundle adjustment which updates a significant portion of patch depths. Here, we only consider patches with at least one high-confidence outgoing edge ($w > 0.5$).

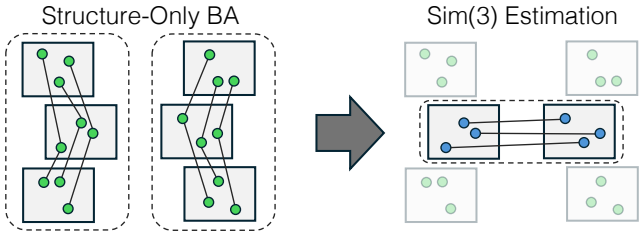



**Fig. 5:** Drift estimation. After identifying candidate image pair for loop closure using image retrieval, we seek to estimate the accumulated drift as a relative 7DOF transformation. Using off-the-shelf detectors and matchers, we estimate 2D correspondence from each retrieved image to its two temporal neighbors and perform structure-only bundle adjustment to triangulate their depth. Finally, we match between the resulting 3D keypoints and estimate a 7DOF point-cloud alignment with RANSAC+Umeyama [33].

**Detecting Long-term Loops:** We identify candidate image pairs using dBoW2 [9] for retrieval, which requires extracting ORB [19] features for each frame. The process of extracting features, indexing and searching the DBoW model takes less time than the forward pass of DPVO and happens concurrently in a separate process, thereby incurring virtually 0 runtime overhead. Following prior work [2], we look for multiple consecutive defections to increase precision.

**Estimating Drift:** We week to estimate the accumulated drift $\Delta S_{jk}^{loop} \in Sim(3)$ between retrieved image pair $(j, k)$. This is often done by matching between their previously-mapped keypoints. However, the keypoints of DPVO [31], our frontend, cannot directly be used for matching since DPVO does not use a repeatable-keypoint detector. This decision has been justified in several prior works [5, 10, 31] which showed that such detectors are not ideal for tracking small-motions. See Fig. 6 for examples.

We instead leverage off-the-shelf keypoint detectors and matchers [15,18,32], only during loop closure, to estimate 2D correspondence from each retrieved image to its temporal neighbors and perform structure-only bundle adjustment to triangulate their depth. We then match the 3D keypoints between the retrieved image pair and align their two point clouds using RANSAC+Umeyama [33]. We depict this process in Fig. 5.

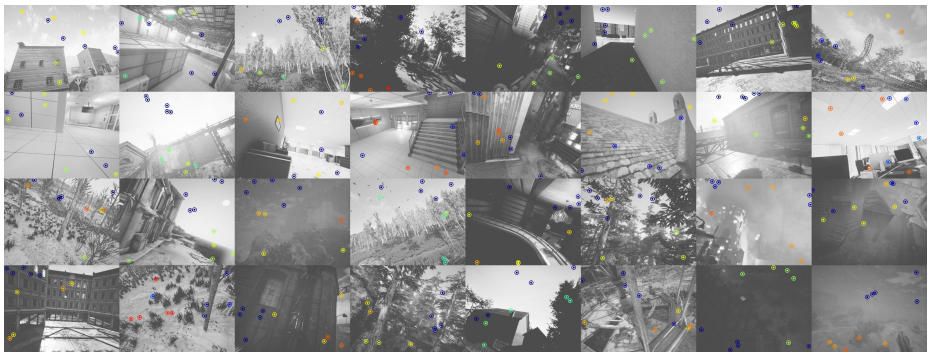


**Fig. 6:** Visualization of the 5 most confident (red/orange) and 5 least confident (blue) patch centroids on TartanAir [36]. 96 keypoints are chosen randomly in each image. Since only the edges have associated weights, we compute the "confidence" of each keypoint as the maximum predicted weight over all of its outgoing edges. In contrast to the usual expectation that the most salient features are the easiest to track, we observe that DPV-SLAM frequently selects points on near-featureless regions.

**Optimization:** The final step estimates an absolute similarity for all keyframes using a simplified version of the algorithm from [24]. We represent the estimated pose of each keyframe $i$ as an absolute similarity $S_i \in Sim(3)$ by converting the current global pose estimates into similarities with scale 1. These are the free variables of the optimization, while terms with the $\Delta$ suffix are constants. We add an error term to our optimization objective between each keyframe and its successor, defined in the tangent space of $Sim(3)$:

$$r_i = log_{Sim(3)}\left(\Delta S^{-1}_{(i,i+1)} \cdot S_i^{-1} \cdot S_{i+1}\right) \tag{5}$$

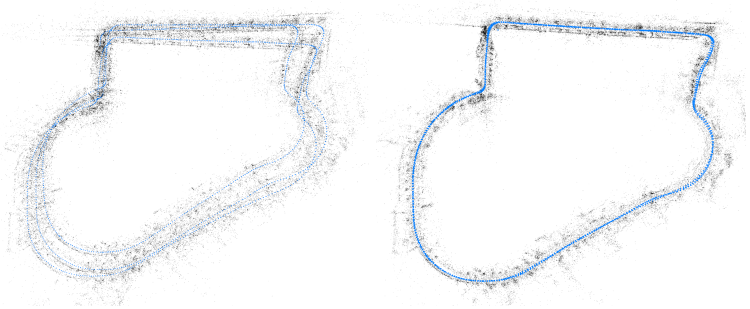
and error terms for the estimated loop connections:

$$r_{jk} = log_{Sim(3)}\left(\Delta S^{loop}_{jk} \cdot S_j^{-1} \cdot S_k\right) \tag{6}$$

We then optimize the following objective using the Levengberg-Marquardt algorithm:

$$\underset{S_1,\dots S_N}{\arg\min} \left( \sum_{i}^{N} \|r_i\|_2^2 + \sum_{(j,k)}^{L} \|r_{jk}\|_2^2 \right) \tag{7}$$

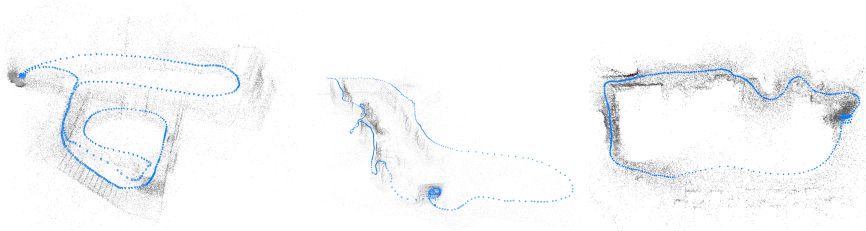
where $N$ is the total number of keyframes and $L$ is the list of detected loops. For each absolute similarity $S_i = (t_i, R_i, s_i)$, the global poses and inverse depths



**(a)** DPVO [31] **(b)** DPVO with classical-backend

**Fig. 7:** Predictions of the DPVO base model with and without classical loop closure (see Sec. 3.4) on the "Business Campus" sequence of the 4Seasons dataset [38]. Our approach leads to more accurate outdoor trajectories.

are updated as $G_i \leftarrow (t_i, R_i)$ and $d_i \leftarrow d_i / s_i$. The pose graph optimization is performed on the CPU in parallel to the main process, thereby incurring negligible runtime overhead.



**(a)** Sequence 41 **(b)** Sequence 20 **(c)** Sequence 43

**Fig. 8:** Qualitative visualization on TUM-Mono [6]

## 4 Experiments

We evaluate DPV-SLAM on four datasets: KITTI [11], TUM-RGBD [26], EuRoC-MAV [1], and the TartanAir [36] test set from the ECCV 2020 SLAM competition. We compare DPV-SLAM to methods which report results on both indoor and outdoor settings, and that do not require re-training their model per-domain. § denotes values which we measured using their open source code, since they were not reported in the original paper. For GO-SLAM, timings were measured using tracking/mono mode. All timing experiments were performed on an RTX-3090.

**TUM-RGBD [26]** We evaluate monocular SLAM on the entirety of the Freiburg 1 set of the TUM-RGBD benchmark in Tab. 1. The purpose is to show that our

| | 360 | desk | desk2 | floor | plant | room | rpy | teddy | xyz | Avg | FPS | VRAM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ORB-SLAM2 [17] | X | 0.071 | X | 0.023 | X | X | X | X | 0.010 | - | | |
| ORB-SLAM3 [2] | X | 0.017 | 0.210 | X | 0.034 | X | X | X | **0.009** | - | | |
| DeepTAM [46] | **0.111** | 0.053 | 0.103 | 0.206 | 0.064 | 0.239 | 0.093 | 0.144 | 0.036 | 0.116 | | |
| TartanVO [35] | 0.178 | 0.125 | 0.122 | 0.349 | 0.297 | 0.333 | 0.049 | 0.339 | 0.062 | 0.206 | | |
| DeepV2D [28] | 0.243 | 0.166 | 0.379 | 1.653 | 0.203 | 0.246 | 0.105 | 0.316 | 0.064 | 0.375 | | |
| DeepV2D [TartanAir] | 0.182 | 0.652 | 0.633 | 0.579 | 0.582 | 0.776 | 0.053 | 0.602 | 0.150 | 0.468 | | |
| DeepFactors [3] | 0.159 | 0.170 | 0.253 | 0.169 | 0.305 | 0.364 | 0.043 | 0.601 | 0.035 | 0.233 | | |
| DeFlowSLAM [41] | 0.159 | **0.016** | 0.030 | 0.169 | 0.048 | 0.538 | **0.021** | **0.039** | **0.009** | 0.114 | | |
| GO-SLAM [45] | 0.089 | **0.016** | **0.028** | 0.025 | 0.026 | 0.052 | 0.019 | 0.048 | 0.010 | **0.035** | 6.4[§] | 7.2G[§] |
| DROID-SLAM [30] | **0.111** | 0.018 | 0.042 | 0.021 | **0.016** | **0.049** | 0.026 | 0.048 | 0.012 | 0.038 | **30** | 8.5G[§] |
| DPV-SLAM | 0.112 | 0.018 | 0.029 | 0.057 | 0.021 | 0.330 | 0.030 | 0.084 | 0.010 | 0.076 | **30** | **4.0G** |
| DPV-SLAM++ | 0.132 | 0.018 | 0.029 | 0.050 | 0.022 | 0.096 | 0.032 | 0.098 | 0.010 | 0.054 | **30** | 6.0G |

**Table 1:** ATE on the TUM-RGBD benchmark. Bolded indicates the best result. We report runtimes and memory for methods whose average error is similar to ours. We perform similarly to other methods based on DROID-SLAM [30,41,45] (0.054-0.076 vs 0.035-0.114), however these approaches do not perform well (or don't report results) outdoors (See Tab. 2), and are more expensive (4.0-6.0G vs 7.2-8.5G). DPV-SLAM++ performs well in both indoors and outdoors. Our objective is a SLAM system which performs well in all settings.

approach performs well both indoors and outdoors. This benchmark evaluates handheld camera motion, and is especially challenging due to rolling shutter effects and motion blur. Video is recorded at 30FPS.

| Sequence | 06 | | 07 | | 09 | | 10 | | Avg | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $t_{rel}$ | $r_{rel}$ | $t_{rel}$ | $r_{rel}$ | $t_{rel}$ | $r_{rel}$ | $t_{rel}$ | $r_{rel}$ | $t_{rel}$ | $r_{rel}$ |
| TartanVO [35] | **4.72** | 2.95 | 4.32 | 3.41 | **6.0** | 3.11 | 6.89 | 2.73 | **5.48** | 3.05 |
| DPV-SLAM++ | 4.95 | **0.16** | **1.29** | **0.24** | 17.69 | **0.23** | **6.32** | **0.23** | 7.56 | **0.22** |

**(a)** Comparison with TartanVO

| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | Avg | FPS ↑ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ORB-SLAM2 [17] | 8.27 | X | **26.86** | **1.21** | 0.77 | 7.91 | 12.54 | 3.44 | **46.81** | 76.54 | **6.61** | - | 34 |
| ORB-SLAM3 [2] | **6.77** | X | 30.500 | 1.036 | 0.930 | 5.542 | 16.605 | 9.700 | 60.687 | 7.899 | 8.650 | - | 34 |
| LDSO [10] | 9.32 | **11.68** | 31.98 | 2.85 | 1.22 | **5.1** | 13.55 | 2.96 | 129.02 | **21.64** | 17.36 | **22.42** | **49** |
| DROID-VO [30] | 98.43 | 84.2 | 108.8 | 2.58 | 0.93 | 59.27 | 64.4 | 24.2 | 64.55 | 71.8 | 16.91 | 54.19 | 17 |
| DPVO [31] | 113.21 | 12.69 | 123.4 | 2.09 | **0.68** | 58.96 | 54.78 | 19.26 | 115.9 | 75.1 | 13.63 | 53.61 | 48 |
| DROID-SLAM [30] | 92.1 | 344.6 | X | 2.38 | 1.00 | 118.5 | 62.47 | 21.78 | 161.6 | X | 118.7 | - | 17 |
| DPV-SLAM | 112.8 | 11.50 | 123.53 | 2.50 | 0.81 | 57.80 | 54.86 | 18.77 | 110.49 | 76.66 | 13.65 | 53.03 | 39 |
| DPV-SLAM++ | 8.30 | 11.86 | 39.64 | 2.50 | 0.78 | 5.74 | **11.6** | **1.52** | 110.9 | 76.7 | 13.7 | 25.76 | 39 |

**(b)** Comparison with other SLAM approaches using ATE[m].

**Table 2:** Monocular SLAM on the KITTI [11] training set. TartanVO only reports results for sequences 6,7,9,10 using the $t_{rel}/r_{rel}$ metrics. Compared to other general approaches to SLAM, our method performs well, while running at 39 FPS. [41, 45] do not report results on KITTI. This table shows the challenge of generalizing across domains: DROID-SLAM performs exceptionally well on indoor datasets, but fails on KITTI. In turn, classical approaches perform well on KITTI, but fail on TUM (Tab. 1). DPVO-SLAM++ performs well on both.

| | MH01 | MH02 | MH03 | MH04 | MH05 | V101 | V102 | V103 | V201 | V202 | V203 | Avg | FPS | VRAM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DeepFactors [3] | 1.587 | 1.479 | 3.139 | 5.331 | 4.002 | 1.520 | 0.679 | 0.900 | 0.876 | 1.905 | 1.021 | 2.040 | | |
| DeepV2D [28]† | 0.739 | 1.144 | 0.752 | 1.492 | 1.567 | 0.981 | 0.801 | 1.570 | 0.290 | 2.202 | 2.743 | 1.298 | | |
| DeepV2D [TartanAir]† | 1.614 | 1.492 | 1.635 | 1.775 | 1.013 | 0.717 | 0.695 | 1.483 | 0.839 | 1.052 | 0.591 | 1.173 | | |
| TartanVO[1] [35]† | 0.639 | 0.325 | 0.550 | 1.153 | 1.021 | 0.447 | 0.389 | 0.622 | 0.433 | 0.749 | 1.152 | 0.680 | | |
| ORB-SLAM [16] | 0.071 | 0.067 | 0.071 | 0.082 | 0.060 | **0.015** | 0.020 | X | 0.021 | 0.018 | X | - | | |
| DSO [10]† | 0.046 | 0.046 | 0.172 | 3.810 | 0.110 | 0.089 | 0.107 | 0.903 | 0.044 | 0.132 | 1.152 | 0.601 | | |
| LDSO [10] | 0.046 | 0.035 | 0.175 | 1.954 | 0.385 | 0.093 | 0.085 | - | 0.043 | 0.405 | - | - | | |
| SVO [7]† | 0.100 | 0.120 | 0.410 | 0.430 | 0.300 | 0.070 | 0.210 | X | 0.110 | 0.110 | 1.080 | - | | |
| ORB-SLAM3 [2] | 0.016 | 0.027 | 0.028 | 0.138 | 0.072 | 0.033 | 0.015 | 0.033 | 0.023 | 0.029 | X | - | | |
| DPVO [31]† | 0.087 | 0.055 | 0.158 | 0.137 | 0.114 | 0.050 | 0.140 | 0.086 | 0.057 | 0.049 | 0.211 | 0.105 | | |
| GO-SLAM [45] | 0.016 | 0.014 | 0.023 | 0.045 | 0.045 | 0.037 | 0.011 | 0.023 | 0.016 | **0.010** | 0.022 | 0.024 | 6§ | 14G§ |
| DROID-SLAM [30] | **0.013** | **0.014** | 0.022 | 0.043 | 0.043 | 0.037 | 0.012 | 0.020 | **0.017** | 0.013 | **0.014** | **0.022** | 20 | 20G§ |
| DPV-SLAM | **0.013** | 0.016 | 0.022 | 0.043 | **0.041** | 0.035 | **0.008** | **0.015** | 0.019 | 0.011 | 0.033 | 0.023 | **50** | **5.0Gb** |
| DPV-SLAM++ | **0.013** | 0.016 | **0.021** | **0.041** | **0.041** | 0.035 | 0.010 | **0.015** | 0.021 | 0.011 | 0.023 | 0.023 | **50** | **7.0Gb** |

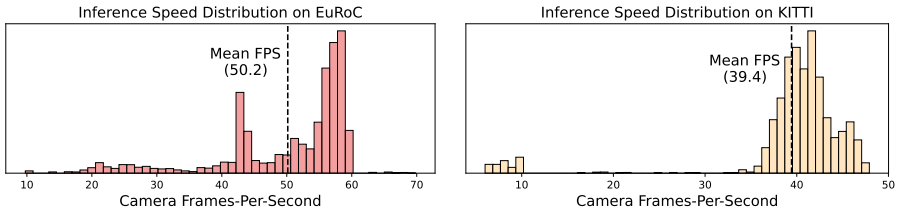
**Table 3:** Monocular SLAM on the EuRoC datasets, ATE[m]. † denotes visual odometry methods. We report runtimes and memory for methods whose average error is similar to ours. We perform marginally worse than DROID-SLAM (0.023 vs 0.022), but use significantly less GPU memory (5G vs 20G) and run 2.5x faster (50 FPS vs 20 FPS).

Compared to other deep SLAM systems [30, 45], our method performs similarly (0.054-0.076 vs 0.38-0.114), however these approaches do not perform well (or don't report results) in outdoor settings (See Tab. 2). Classical approaches [2, 17] generally do not perform well on this dataset. In contrast, they perform well on KITTI while previous deep SLAM methods do not. Our method performs well on both datasets.

**KITTI [11]** We evaluate monocular SLAM on sequences 00-10 from the KITTI training set in Tab. 2. The purpose is to show that our approach performs well both indoors and outdoors. Video is recorded at 10FPS. The KITTI dataset includes long outdoor driving sequences with several loops. In order to correct scale drift, monocular methods must implement some form of loop closure. However, the loop-closure approach used in DROID-SLAM is insufficient; in fact, it causes catastrophic failures on this dataset. Our approach achieves the second-lowest average error among all methods in Tab. 2, while averaging 39FPS. Note that the 1st place method is not especially accurate in indoor settings (see Tab. 3).

**EuRoC-MAV [1]** We evaluate monocular SLAM on the Machine-Hall and Vicon 1 & 2 sequences from the EuRoC MAV dataset. Video is recorded at 20 FPS. This benchmark contains long sequences with motion blur, over/under-exposed images, and rapid camera movement. On EuRoC, our method performs similarly to DROID-SLAM (0.023 vs 0.022 ATE), while running 2.5x faster (50 FPS vs 20 FPS) using a quarter of the memory (5.0G vs 20G). Compared to the base DPVO system, we achieve 4.5x lower error (0.105→0.023), with only a small reduction in speed (60→50-FPS) and increase in cost (4G→5G).

**TartanAir [36]** We evaluate monocular SLAM on the TartanAir test set from the ECCV 2020 SLAM competition. Compared to existing approaches, DPV-SLAM outperforms DROID-SLAM by a sizeable margin (0.16 vs 0.24). Our method runs at 27 FPS, while DROID-SLAM runs at 8FPS.




**(a)** Latency on EuRoC [1] **(b)** Latency on KITTI [11]

**Fig. 9:** The distribution of inference speed. We run DPV-SLAM on EuRoC and KITTI, and sample the current FPS uniformly over the runtime. In both cases, there are two modes, the larger being representative of the odometry runtime and the smaller being representative of the loop closure. On EuRoC, the loop closure causes the runtime to drop from 60 to 42 FPS. On KITTI, the infrequent loop closure causes the speed to very briefly dip below real-time (5-10 FPS) before returning to 40-FPS. The average speed on EuRoC and KITTI are 50-FPS and 39-FPS, or 2.5x and 3.9x real-time, respectively.

| Monocular | MH000 | MH001 | MH002 | MH003 | MH004 | MH005 | MH006 | MH007 | Avg |
|---|---|---|---|---|---|---|---|---|---|
| ORB-SLAM [16] | 1.30 | **0.04** | 2.37 | 2.45 | X | X | 21.47 | 2.73 | - |
| DeepV2D [28] | 6.15 | 2.12 | 4.54 | 3.89 | 2.71 | 11.55 | 5.53 | 3.76 | 5.03 |
| TartanVO [35] | 4.88 | 0.26 | 2.00 | 0.94 | 1.07 | 3.19 | 1.00 | 2.04 | 1.92 |
| DPVO [31] | 0.21 | **0.04** | 0.04 | 0.08 | 0.58 | **0.17** | 0.11 | 0.15 | 0.17 |
| DeFlowSLAM [41] | 0.63 | 0.06 | **0.02** | **0.01** | 2.80 | 0.20 | 0.31 | 0.45 | 0.56 |
| DROID-SLAM [30] | **0.08** | 0.05 | 0.04 | 0.02 | **0.01** | 1.31 | 0.30 | **0.07** | 0.24 |
| DPV-SLAM | 0.23 | **0.04** | 0.04 | 0.04 | 0.54 | **0.17** | **0.08** | 0.13 | **0.16** |
| DPV-SLAM++ | 0.21 | **0.04** | 0.04 | 0.04 | 0.92 | **0.17** | 0.11 | 0.13 | 0.21 |

**Table 4:** Results on the TartanAir monocular benchmark (ATE[m]). We outperform existing approaches.

## 5 Limitations

DPV-SLAM requires a GPU to run, which imposes additional hardware constraints. Additionally, the global bundle adjustment layer scales poorly to very large scenes, which is why we limit its range to 1000 frames. This bottleneck is primarily caused by the Cholesky decomposition.

The image retrieval is also susceptible to the occasional false-positive detection, leading to catastrophic failures. The classical loop closure also incurs an additional 2Gb of GPU memory due to the invocation of the U-Net keypoint detector [32].

## 6 Conclusion

We introduce DPV-SLAM, a system for monocular visual SLAM. DPV-SLAM generalizes well to different domains, and is efficient. We evaluate on EuRoC, TartanAir, TUM-RGBD and KITTI. We show that our approach is robust across domains, and is comparable to, or better than the SOTA on several real-world datasets, while running much faster and using less compute. This work was partially supported by the National Science Foundation.

## References


1. Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M.W., Siegwart, R.: The euroc micro aerial vehicle datasets. The International Journal of Robotics Research **35**(10), 1157–1163 (2016)
2. Campos, C., Elvira, R., Rodríguez, J.J.G., Montiel, J.M., Tardós, J.D.: Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. IEEE Transactions on Robotics **37**(6), 1874–1890 (2021)
3. Czarnowski, J., Laidlow, T., Clark, R., Davison, A.J.: Deepfactors: Real-time probabilistic dense monocular slam. IEEE Robotics and Automation Letters **5**(2), 721–728 (2020)
4. Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nießner, M.: Scannet: Richly-annotated 3d reconstructions of indoor scenes. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 5828–5839 (2017)
5. Engel, J., Koltun, V., Cremers, D.: Direct sparse odometry. IEEE transactions on pattern analysis and machine intelligence **40**(3), 611–625 (2017)
6. Engel, J., Usenko, V., Cremers, D.: A photometrically calibrated benchmark for monocular visual odometry. arXiv preprint arXiv:1607.02555 (2016)
7. Forster, C., Pizzoli, M., Scaramuzza, D.: Svo: Fast semi-direct monocular visual odometry. In: 2014 IEEE international conference on robotics and automation (ICRA). pp. 15–22. IEEE (2014)
8. Fu, T., Su, S., Wang, C.: islam: Imperative slam. arXiv preprint arXiv:2306.07894 (2023)
9. Gálvez-López, D., Tardos, J.D.: Bags of binary words for fast place recognition in image sequences. IEEE Transactions on Robotics **28**(5), 1188–1197 (2012)
10. Gao, X., Wang, R., Demmel, N., Cremers, D.: Ldso: Direct sparse odometry with loop closure. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 2198–2204. IEEE (2018)
11. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: 2012 IEEE conference on computer vision and pattern recognition. pp. 3354–3361. IEEE (2012)
12. Keetha, N., Karhade, J., Jatavallabhula, K.M., Yang, G., Scherer, S., Ramanan, D., Luiten, J.: Splatam: Splat, track map 3d gaussians for dense rgb-d slam. arXiv preprint (2023)
13. Li, H., Gu, X., Yuan, W., Yang, L., Dong, Z., Tan, P.: Dense rgb slam with neural implicit maps. arXiv preprint arXiv:2301.08930 (2023)
14. Li, R., Wang, S., Long, Z., Gu, D.: Undeepvo: Monocular visual odometry through unsupervised deep learning. In: 2018 IEEE international conference on robotics and automation (ICRA). pp. 7286–7291. IEEE (2018)
15. Lindenberger, P., Sarlin, P.E., Pollefeys, M.: Lightglue: Local feature matching at light speed. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 17627–17638 (2023)
16. Mur-Artal, R., Montiel, J.M.M., Tardos, J.D.: Orb-slam: a versatile and accurate monocular slam system. IEEE transactions on robotics **31**(5), 1147–1163 (2015)
17. Mur-Artal, R., Tardós, J.D.: Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. IEEE transactions on robotics **33**(5), 1255–1262 (2017)
18. Riba, E., Mishkin, D., Ponsa, D., Rublee, E., Bradski, G.: Kornia: an open source differentiable computer vision library for pytorch. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 3674–3683 (2020)

19. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: Orb: An efficient alternative to sift or surf. In: 2011 International conference on computer vision. pp. 2564–2571. Ieee (2011)
20. Schneider, T., Dymczyk, M., Fehr, M., Egger, K., Lynen, S., Gilitschenski, I., Siegwart, R.: maplab: An open framework for research in visual-inertial mapping and localization. IEEE Robotics and Automation Letters **3**(3), 1418–1425 (2018)
21. Schops, T., Sattler, T., Pollefeys, M.: Bad slam: Bundle adjusted direct rgb-d slam. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 134–144 (2019)
22. Shen, S., Cai, Y., Wang, W., Scherer, S.: Dytanvo: Joint refinement of visual odometry and motion segmentation in dynamic environments. In: 2023 IEEE International Conference on Robotics and Automation (ICRA). pp. 4048–4055. IEEE (2023)
23. Shi, J., et al.: Good features to track. In: 1994 Proceedings of IEEE conference on computer vision and pattern recognition. pp. 593–600. IEEE (1994)
24. Strasdat, H., Montiel, J., Davison, A.J.: Scale drift-aware large scale monocular slam. Robotics: science and Systems VI **2**(3), 7 (2010)
25. Straub, J., Whelan, T., Ma, L., Chen, Y., Wijmans, E., Green, S., Engel, J.J., Mur-Artal, R., Ren, C., Verma, S., et al.: The replica dataset: A digital replica of indoor spaces. arXiv preprint arXiv:1906.05797 (2019)
26. Sturm, J., Engelhard, N., Endres, F., Burgard, W., Cremers, D.: A benchmark for the evaluation of rgb-d slam systems. In: 2012 IEEE/RSJ international conference on intelligent robots and systems. pp. 573–580. IEEE (2012)
27. Sucar, E., Liu, S., Ortiz, J., Davison, A.J.: imap: Implicit mapping and positioning in real-time. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 6229–6238 (2021)
28. Teed, Z., Deng, J.: Deepv2d: Video to depth with differentiable structure from motion. arXiv preprint arXiv:1812.04605 (2018)
29. Teed, Z., Deng, J.: Raft: Recurrent all-pairs field transforms for optical flow. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16. pp. 402–419. Springer (2020)
30. Teed, Z., Deng, J.: Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. Advances in neural information processing systems **34**, 16558–16569 (2021)
31. Teed, Z., Lipson, L., Deng, J.: Deep patch visual odometry. Advances in Neural Information Processing Systems (2023)
32. Tyszkiewicz, M., Fua, P., Trulls, E.: Disk: Learning local features with policy gradient. Advances in Neural Information Processing Systems **33**, 14254–14265 (2020)
33. Umeyama, S.: Least-squares estimation of transformation parameters between two point patterns. IEEE Transactions on Pattern Analysis & Machine Intelligence **13**(04), 376–380 (1991)
34. Wang, S., Clark, R., Wen, H., Trigoni, N.: Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In: 2017 IEEE international conference on robotics and automation (ICRA). pp. 2043–2050. IEEE (2017)
35. Wang, W., Hu, Y., Scherer, S.: Tartanvo: A generalizable learning-based vo. In: Conference on Robot Learning. pp. 1761–1772. PMLR (2021)
36. Wang, W., Zhu, D., Wang, X., Hu, Y., Qiu, Y., Wang, C., Hu, Y., Kapoor, A., Scherer, S.: Tartanair: A dataset to push the limits of visual slam. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 4909–4916. IEEE (2020)

37. Wang, X., Maturana, D., Yang, S., Wang, W., Chen, Q., Scherer, S.: Improving learning-based ego-motion estimation with homomorphism-based losses and drift correction. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 970–976. IEEE (2019)
38. Wenzel, P., Wang, R., Yang, N., Cheng, Q., Khan, Q., von Stumberg, L., Zeller, N., Cremers, D.: 4seasons: A cross-season dataset for multi-weather slam in autonomous driving. In: Pattern Recognition: 42nd DAGM German Conference, DAGM GCPR 2020, Tübingen, Germany, September 28–October 1, 2020, Proceedings 42. pp. 404–417. Springer (2021)
39. Xu, S., Xiong, H., Wu, Q., Wang, Z.: Attention-based long-term modeling for deep visual odometry. In: 2021 Digital Image Computing: Techniques and Applications (DICTA). pp. 1–8. IEEE (2021)
40. Yang, N., Stumberg, L.v., Wang, R., Cremers, D.: D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 1281–1292 (2020)
41. Ye, W., Yu, X., Lan, X., Ming, Y., Li, J., Bao, H., Cui, Z., Zhang, G.: Deflowslam: Self-supervised scene motion decomposition for dynamic dense slam (2023)
42. Yin, Z., Shi, J.: Geonet: Unsupervised learning of dense depth, optical flow and camera pose. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1983–1992 (2018)
43. Zhan, H., Weerasekera, C.S., Bian, J.W., Garg, R., Reid, I.: Df-vo: What should be learnt for visual odometry? arXiv preprint arXiv:2103.00933 (2021)
44. Zhang, J., Henein, M., Mahony, R., Ila, V.: Vdo-slam: a visual dynamic object-aware slam system. arXiv preprint arXiv:2005.11052 (2020)
45. Zhang, Y., Tosi, F., Mattoccia, S., Poggi, M.: Go-slam: Global optimization for consistent 3d instant reconstruction. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 3727–3737 (2023)
46. Zhou, H., Ummenhofer, B., Brox, T.: Deeptam: Deep tracking and mapping. In: Proceedings of the European conference on computer vision (ECCV). pp. 822–838 (2018)
47. Zhu, Z., Peng, S., Larsson, V., Xu, W., Bao, H., Cui, Z., Oswald, M.R., Pollefeys, M.: Nice-slam: Neural implicit scalable encoding for slam. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12786–12796 (2022)