DEEP PATCH VISUAL ODOMETRY

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose Deep Patch Visual Odometry (DPVO), a new deep learning system for monocular Visual Odometry (VO). DPVO is accurate and robust while running at 2x-5x real-time speeds on a single RTX-3090 GPU using only 4GB of memory. We perform evaluation on standard benchmarks and outperform all prior work (classical or learned) in both accuracy and speed.

1 INTRODUCTION

Visual Odometry (VO) is the task of estimating a robot's position and orientation from visual measurements. In this work, we focus on most challenging case—monocular VO—where the only input is a monocular video stream. The goal of the system is to estimate the 6-DOF pose of the camera at every frame while simultaneously building a map of the environment.

VO is closely related to Simultaneous Localization and Mapping (SLAM). Like VO, SLAM systems aim to estimate camera pose and map the environment but also incorporate techniques for global corrections—such as loop closure and relocalization (Cadena et al., 2016). SLAM systems typically include a VO frontend which tracks incoming frames and performs local optimization. We observe that a significant portion of failures in SLAM systems occur in the frontend and hence focus on this aspect of the problem. However, we demonstrate that even without loop closure and global optimization, our approach is still accurate enough to outperform full SLAM systems.

Prior work typically treats VO as an optimization problem solving for a 3D model of the scene which best explains the visual measurements (Cadena et al., 2016). *Indirect* approaches first detect and match keypoints between frames, then solve for poses and 3D points which minimize the reprojection distance (Mur-Artal et al., 2015; Campos et al., 2021; Leutenegger et al., 2013). *Direct* approaches, on the other hand, operate directly on pixel intensities, attempting to solve for poses and depths which align the images (Engel et al., 2014; Forster et al., 2014; Engel et al., 2017). The main issue with prior systems, both direct and indirect, is the lack of robustness. Failure cases are too frequent for many important applications such as autonomous vehicles. These failure cases typically stem from moving objects, lost feature tracks, and poor convergence.

Several deep learning approaches have been introduced to address the robustness issue. The main promise of deep learning is more reliable feature matching. DROID-SLAM (Teed & Deng, 2021) and VOLODOR (Min et al., 2020; Min & Dunn, 2021) use neural networks to estimate dense flow fields which are subsequently used to optimize depth and camera pose. Other approaches have used neural networks to match and verify a sparse set of keypoints (DeTone et al., 2018; Sarlin et al., 2020; Sun et al., 2021; Truong et al., 2021; Choy et al., 2016). Methods such as BANet (Tang & Tan, 2018) and DeepFactors (Czarnowski et al., 2020) have used neural networks to parameterize the space of admissible depth maps. However, many of these systems come with a large computational cost which makes them impractical for real use cases. Furthermore, current deep VO systems are typically not as accurate as classical systems when evaluated on datasets on which they weren't trained (Teed & Deng, 2021).

We introduce DPVO, a novel patch-based deep VO system which overcomes these limitations. The central piece of our approach is a deep patch representation (Fig. 1). We use a neural network to extract a collection of patches from incoming frames. A recurrent neural network is then used to track each patch through time—alternating patch trajectory updates with a differentiable bundle adjustment layer. We train our entire system end-to-end on synthetic data but demonstrate strong generalization on real video.

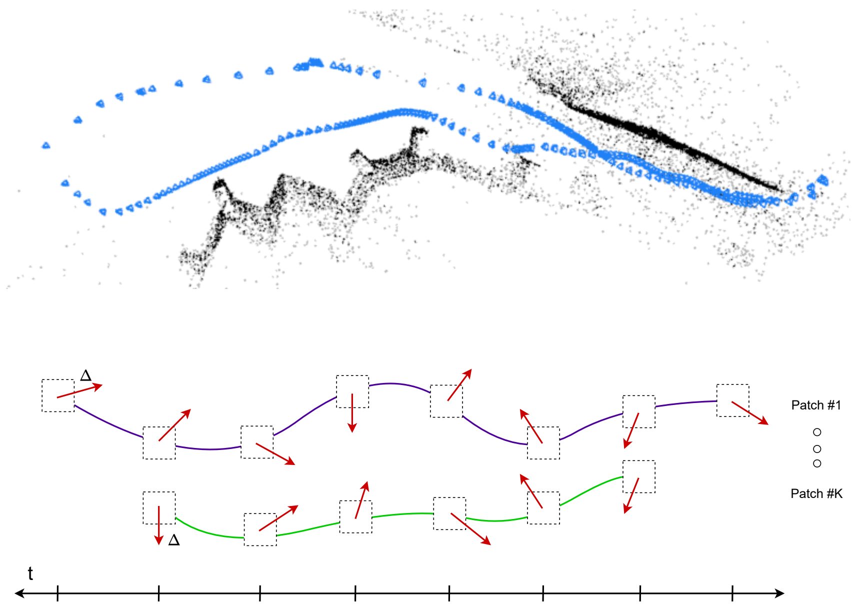


Figure 1: Deep Patch Visual Odometry (DPVO). Camera poses and a sparse 3D reconstruction (top) are obtained by iterative revisions of patch trajectories through time.

Compared to prior deep-learning systems, the novelty of our approach lies in the tight integration of three key ingredients in a single architecture: (1) patch-based correspondence, (2) recurrent iterative updates, and (3) differentiable bundle adjustment. Patch-based correspondence improves efficiency and robustness over dense flow. Recurrent iterative updates and differentiable bundle adjustment allow end-to-end learning of reliable feature matching.

DPVO is accurate, efficient, and simple to implement. On a modern graphics card (RTX-3090), it runs 2x real-time using only 4GB of memory. We also provide a model which runs at 100fps on the EuRoC dataset (Burri et al., 2016) while still outperforming prior work. Runtime is constant for each frame and does not depend on the degree of camera motion. The implementation of the system is exceedingly simple with a minimal code base. New network architectures can be easily swapped in without any necessary change to the underlying VO implementation or logic. We hope that DPVO can serve as a test bed for future development of deep VO and SLAM systems. In the Appendix, we provide an anonymized version of our code and a video of our method running.

2 Related Work

Visual Odometry (VO) systems aim to estimate robot state (position and orientation) from a video. Overtime, a VO system will accumulate drift, and modern SLAM methods incorporate techniques to identify previously mapped landmarks to correct drift (i.e loop closure). VO can be considered a subproblem of SLAM with loop closures disabled (Cadena et al., 2016).

Many different modalities of VO have been explored by past work, including visual-inertial odometry (VIO) (Von Stumberg et al., 2018; Forster et al., 2015) and stereo VO (Wang et al., 2017; Engel et al., 2014). Here, we focus on the monocular case, where the only input is a monocular video stream. Early works approached the problem using filtering and maximum-likelihood methods (Davison, 2003; Mourikis et al., 2007). Modern methods almost universally perform Maximum a Posteriori (MAP) estimation over factor graphs with Gaussian noise; in which case, the MAP estimate can be found by solving a non-linear least-squares optimization problem (Dellaert et al., 2017). This problem has lead to the development of many libraries for optimizing non-linear least-squares problems (Agarwal et al., 2022; Grisetti et al., 2011; Dellaert, 2012).

Among VO systems, our method borrows many core ideas of Direct Sparse Odometry (DSO) (Engel et al., 2017), a classical system based on least-squares optimization. Namely, we adopt a similar

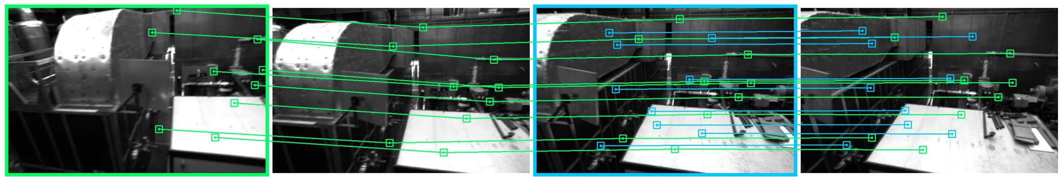


Figure 2: A subset of the patch trajectories predicted by our method. Patches extracted from the green keyframe are tracked through subsequent frames. When a new keyframe is added (blue), additional patches are extracted and tracked. Our method produces confidence values which weight their respective contribution to the bundle adjustment.

patch representation and reproject patches between frames to construct the objective function. Unlike DSO, the residuals are not based on intensity differences but instead predicted by a neural network which can pass information between patches and across patch lifetimes. Outlier rejection is automatically handled by the network, making our system more robust than classical systems like DSO and ORB-SLAM (Engel et al., 2017; Mur-Artal et al., 2015). One important component of classical systems is the careful selection of which image regions to use. We find, surprisingly, that our system works well on a small number (64 per frame) of *randomly* sampled image patches.

With regards to deep SLAM systems, our method is closely related to DROID-SLAM (Teed & Deng, 2021) but uses a different underlying representation. DROID-SLAM is an end-to-end deep SLAM system which shows good performance compared to both classical and deep baselines. Like our method, it works by iterating between motion updates and bundle adjustment. However, it estimates dense motion fields between selected pairs of frames which has a high computational cost and large memory footprint. While it is capable of real-time inference *on average*, its speed varies depending on the amount of motion in the video. Our method selects sparse patches from the video stream, with a constant runtime per frame and 5x faster inference than DROID-SLAM.

Prior works such as BA-Net (Tang & Tan, 2018) and Lindenberger et al. (2021) also have embedded bundle adjustment layers in end-to-end differentiable network architectures. However, BA-Net does not use patch-based correspondence. Lindenberger et al. (2021) and Dusmanu et al. (2020) have proposed neural networks which sit atop COLMAP (Schonberger & Frahm, 2016) and perform subpixel-level refinement; these approaches are not, however, able to perform 3D reconstruction on their own and are subject to failure cases in the underlying SfM system.

3 Approach

Our network is trained and evaluated in an online setting. New frames are added one by one and optimization is performed in a local window of keyframes. Our approach has two main modules: the *patch extractor* (3.1) and the *update operator* (3.2). The *patch extractor* extracts a sparse collection of image patches from incoming frames. The *update operator* attempts to track these patches through time using a recurrent neural network alternating iterative updates with bundle adjustment.

Preliminaries: The scene is represented as a collection of N camera poses $\mathbf{T} \in \mathbb{SE}(3)^N$ and a set of M square image patches $\mathbf{P}$. Using $\mathbf{d}$ to represent inverse depth and $(\mathbf{x}, \mathbf{y})$ pixel coordinates, we represent each patch as the $4 \times p^2$ homogeneous array

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

where p is the width of the patch. We assume a constant depth for the full patch, meaning that it forms a fronto-parallel plane in the frame from which it was extracted. Letting j denote the index of the source frame from which patch $\mathbf{P}_i$ was extracted, we reproject the patch onto another frame k

$$\mathbf{P}_i' \sim \mathbf{K}\mathbf{T}_k{\mathbf{T}_j}^{-1}\mathbf{K}^{-1}\mathbf{P}_i. \tag{2}$$

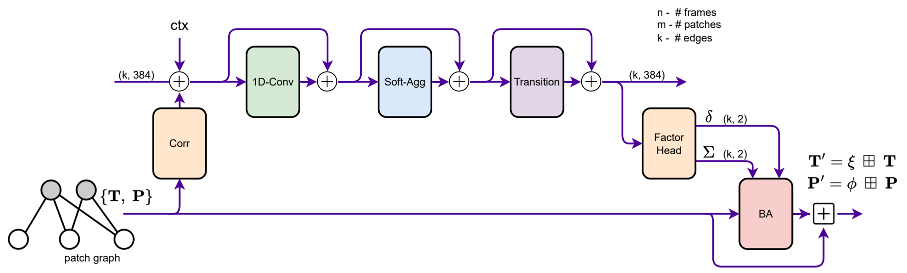


Figure 3: Schematic of the *update operator*. Correlation features are extracted from edges in the patch graph and injected into the hidden state alongside context features. We apply convolution, message passing and a transition block. The factor head produces trajectory revisions which are used by the bundle adjustment layer to update the camera poses and the depth of patches.

taking $\mathbf{K}$ to be the 4×4 calibration matrix

$$\mathbf{K} = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

The pixel coordinates $\mathbf{x}' = (x', y')$ can be recovered by dividing by the third element. For the rest of the paper, we use the shorthand $\mathbf{x}_{ij} = \omega_{ij}(\mathbf{T}, \mathbf{P})$ to denote the reprojection of patch i onto frame j in terms of pixel coordinates.

Patch Graph: We use a bipartite *patch graph* data structure to represent the patch lifetimes. Edges in the graph connect patches with frames. By default, the graph is constructed by adding an edge between each patch and every frame within distance r from the index of the frame it was extracted from. The reprojections of a patch onto all of its connected frames in the pose graph form the *trajectory* of the patch. We provide an example trajectory in Fig. 2.

3.1 Feature and Patch Extraction

We use a pair of residual networks to extract features from the input images. One network extracts *matching* features while the other extracts *context* features. The first layer of each network is a 7×7 convolution with stride 2 followed by two residual blocks at 1/2 resolution (dimension 64) and 2 residual blocks at 1/4 resolution (dimension 128), such that the final feature map is one-quarter the input resolution. The architectures of the matching and context networks are identical with the exception that the matching network uses instance normalization and the context network uses no normalization. We construct a two-level feature pyramid by applying average pooling to the matching features with a 4×4 filter with stride 4.

We store the matching features for each frame. We additionally extract patches from both the matching and context feature maps. Patch centroids are randomly sampled, then we use bilinear interpolation for feature retrieval. Unlike DROID-SLAM, we never explicitly build correlation volumes. We instead store both frame and patch feature maps such that correlation features can be computed on-the-fly.

3.2 Update Operator

The purpose of the *update operator* is to update both poses and patches. This is done by performing revisions to patch trajectories as shown in Fig. 1. We provide a schematic overview of the operator in Fig. 3 and detail the individual components below. Each "+" operation in the diagram is a residual connection followed by layer normalization. The update operator acts on the patch graph, and each edge in the patch graph is augmented with a hidden state (dimension 384). When a new edge is added, the hidden state is initialized with zeros.

Correlation: For each each edge (i, j) in the patch graph, we compute correlation features. We first use Eqn. 2 to reproject patch i in frame j: $\mathbf{x}_{ij} = \omega_{ij}(\mathbf{T}, \mathbf{P})$. Given patch features $\mathbf{g} \in \mathbb{R}^{p \times p \times D}$

and frame features $\mathbf{f} \in \mathbb{R}^{H \times W \times D}$, for each pixel (u, v) in patch i, we compute its correlation with a grid of pixels centered at the the reprojection of pixel (u, v) in frame j, using the inner product:

$$\mathbf{C}_{\mu\nu\lambda\rho} = \langle \mathbf{g}_{\mu\nu},\ \mathbf{f}(\mathbf{x}_{ij} + \Delta_{\lambda\rho}) \rangle \tag{4}$$

where we take Δ to be a 7×7 integer grid centered at 0 indexed by λ and ρ, and $\mathbf{f}(\cdot)$ denotes bilinear sampling. We compute correlation features for both levels in the pyramid and concatenate the results. This operation is implemented as an optimized CUDA layer which leverages the regular grid structure of the interpolation step. This implementation is identical to the alternative correlation implementation used by RAFT (Teed & Deng, 2020) and is equivalent to indexing correlation volumes due to the linearity of the inner product and interpolation.

***1D Temporal Convolution**:* We apply a 1D-Convolution in the temporal dimension to each patch trajectory. Since trajectories vary in length and keyframes are actively added and removed, it is not straightforward to implement convolution as a batched operation. Instead, for each edge (i, j) we index the features of its neighbors at $(i, j-1)$ and $(i, j+1)$, concatenate, then apply a linear projection. The temporal convolution allows the network to propagate information along each patch trajectory and model appearance changes of the patch through time.

***SoftMax Aggregation**:* We use global message passing layers to propagate information between edges in the patch graph. This operation has appeared before in the context of graph neural networks. Given edge e and denoting its neighbors as $N(e)$ we define the channel-wise aggregation function

$$\psi \left(\left[\sum_{x \in N(e)} \sigma(x) \cdot \phi(x) \right] \bigg/ \sum_{x \in N(e)} \sigma(x) \right) \tag{5}$$

where ψ and ϕ are linear layers and σ is a linear layer followed by a sigmoid activation. We perform two instantations of soft aggregation: (1) patch aggregation where edges are neighbors if they connect to the same patch (2) frame aggregation where edges are neighbors if they connect to both the same source and destination frames.

***Transition**:* The *transition* block in Fig. 3 is simply two residual units with Layer Normalization and ReLU non-linearities.

***Factor Head**:* The *factor head* consists of 2 MLPs with one hidden unit each. For each edge (i, j) in the pose graph, the first MLP predicts a trajectory update $\delta_{ij} \in \mathbb{R}^2$: a 2D flow vector indicating how the reprojection of the patch center should be updated in 2D; the second MLP predicts a confidence weight map $\Sigma_{ij} \in \mathbb{R}^2$ which is bounded to $(0, 1)$ using a sigmoid activation.

***Differentiable Bundle Adjustment**:* This layer in Fig. 3 operates globally on the patch graph and outputs updates to depth and camera poses. The predicted factors (δ, Σ) are used to define an optimization objective:

$$\sum_{(i,j) \in \mathcal{E}} \|\hat{\omega}_{ij}(\mathbf{T}, \mathbf{P}) - [\hat{\mathbf{x}}_{ij} + \delta_{ij}]\|^2_{\Sigma_{ij}} \tag{6}$$

where $\|\cdot\||_\Sigma$ is the Mahalanobis distance and $\hat{\mathbf{x}}_{ij}$ denotes the center of patch $\mathbf{x}_{ij}$. We apply two Gauss-Newton iterations to the linearized objective, optimizing the camera poses as well as the inverse depth component of the patch while keeping the pixel coordinates constant. This optimization seeks to refine the camera poses and depth such that the induced trajectory updates agree with the predicted trajectory updates. Similar to DROID-SLAM (Teed & Deng, 2021), we apply the Schur complement trick for efficient decomposition and backpropagate gradients through the Gauss-Newton iterations.

3.3 Training and Supervision

DPVO is implemented using PyTorch. We train our network on the TartanAir dataset. On each training sequence, we precompute optical flow magnitude between all pairs of frames using the ground truth poses and depth. During training, we sample trajectories where frame-to-frame optical flow magnitude is between 16px and 72px. This ensures that training instances are generally difficult but not impossible.

We apply supervision to poses and induced optical flow (i.e. trajectory updates), supervising each intermediate output of the update operator and detach the poses and patches from the gradient tape prior to each update.

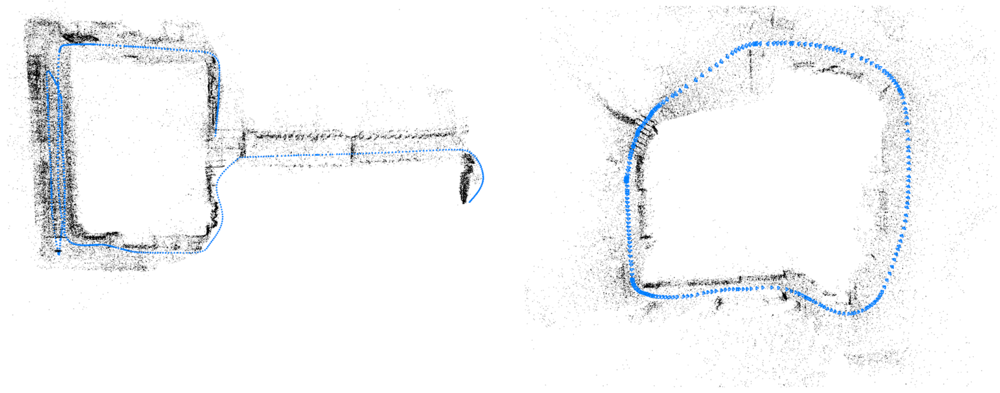


Figure 4: Example reconstructions: TartanAir (left) and ETH3D (right)

Pose Supervision: We scale the predicted trajectory to match the groundtruth using the Umeyama alignment algorithm (Umeyama, 1991). Then for every pair of poses (i, j), we supervise on the error

$$\sum_{(i,j)\ i \neq j} \|[(\mathbf{G}_i^{-1}\mathbf{G}_j)^{-1}(\mathbf{T}_i^{-1}\mathbf{T}_j)]\| \tag{7}$$

where $\mathbf{G}$ is the ground truth and $\mathbf{T}$ are the predicted poses.

Flow Supervision: We additionally supervise on the distance between the induced optical flow and the ground truth optical flow between each patch and the frames within two timestamps of its source frame. Each patch induces a $p \times p$ flow field. We take the minimum of all $p \times p$ errors.

The final loss is the weighted combination

$$\mathcal{L} = 10\mathcal{L}_{pose} + 0.1\mathcal{L}_{flow}. \tag{8}$$

Training Details: We train for a total of 240k iterations on a single RTX-3090 GPU with a batch size of 1. Training takes 3.5 days. We use the AdamW optimizer and start with an initial learning rate of 8e-5 which is decayed linearly during training. We apply standard augmentation techniques such as resizing and color jitter.

We train on sequences of length 15. The first 8 frames are used for initialization while the next 7 frames are added one at a time. We unroll 18 iterations of the update operator during training. For the first 1000 training steps, we fix poses with the ground truth and only ask the network to estimate the depth of the patches. Afterwards, the network is required to estimate both poses and depth.

4 VO System

In this section, we cover several key implementation details necessary for turning our network into a complete visual odometry system. The logic of the system is primarily implemented in Python with bottleneck operations such as bundle adjustment and visualization implemented in C++ and CUDA. Compared to other VO system, DPVO is exceedingly simple and requires minimal design choices.

Initialization: We use 8 frames for initialization. We add new patches and frames until 8 frames are accumulated and then run 12 iterations of our update operator. There needs to be some camera motion for initialization; hence, we only accumulate frames with an average flow magnitude of at least 8 pixels from the prior frame.

Expansion: When a new frame is added we extract features and patches. The pose of the new frame is initialized using a constant velocity motion model. The depth of the patch is initialized as the median depth of all the patches extracted from the previous 3 frames.

We connect each patch to every frame within distance r from the frame index where the patch was extracted. This means that when a new patch is added, we add edges between that patch and the previous r keyframes. When a new frame is added, we add edges between each patch extracted in

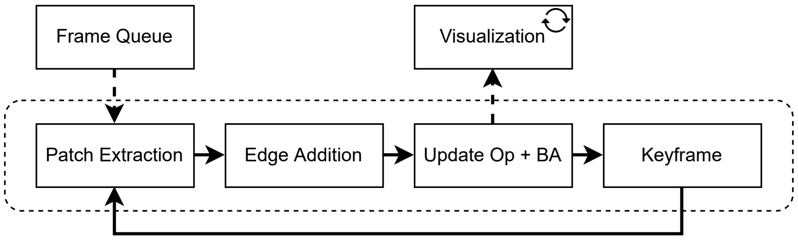


Figure 5: Overview of the VO System.

the last r keyframes with the new frame. This strategy means that the patch graph will always have a maximum size ensuring worst case constant runtime.

Optimization: Following the addition of edges we run one iteration of the update operator followed by two bundle adjustment iterations. We fix the poses of all but the last 10 keyframes. The inverse depths of all patches are free parameters. The patches are removed from optimization once they fall outside the optimization window.

Keyframing: The most recent 3 frames are always taken to be keyframes. After each update, we compute the optical flow magnitude between keyframe $t-5$ and $t-3$. If this is less than 64px, we remove the keyframe at $t-4$. When a keyframe is removed, we store the relative pose between its neighbors such that the full pose trajectory can be recovered for evaluation.

Visualization: Reconstructions are visualized interactively using a separate visualization thread. Our visualizer is implemented using the Pangolin library[1]. It directly reads from PyTorch tensors avoiding all unnecessary memory copies from CPU to GPU. This means that the visualizer has very little overhead–only slowing the full system down by approximately 10%. In the Appendix we provide a live video of our system reconstructing an iPhone video.

5 EXPERIMENTS

We evaluate DPVO on the TartanAir (Wang et al., 2020b) and EuRoC (Burri et al., 2016) benchmarks. On each dataset, we run multiple trials each with a different set of patches and report the median results obtained. Example reconstructions on the TartanAir and ETH3D datasets are shown in Fig. 4.

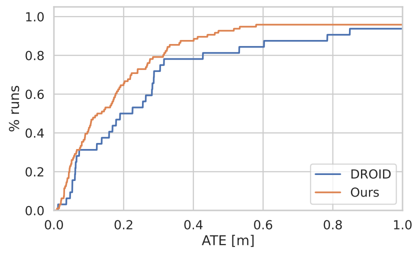
5.1 TARTAN AIR (WANG ET AL., 2020B)

Validation Split: We use the same 32-sequence validation split as DROID-SLAM and report aggregated results in Fig. 6a and compare with DROID-SLAM and ORB-SLAM3. We run our method 3 times on each sequence and aggregate the results. In the $[0, 1]$m error window, we get an AUC of 0.80 compared to 0.71 for DROID-SLAM.

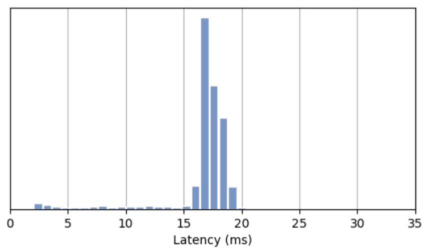
Test Split: In Tab. 1 we report results on the test-split used in the ECCV 2020 SLAM competition compared to state-of-the-art methods. Classical methods such as DSO and ORB-SLAM fail on more than 80% of the sequences, hence we use COLMAP as a classical baseline as it was used in the two winning solutions of the ECCV SLAM competition. For our method, we run five times on each sequence and report the median results.

We benchmark two versions of DROID-SLAM: the full version and a version where loop closure and global bundle adjustment is disabled (DROID-VO). DROID-VO is more comparable to our method as we only perform local optimization. We outperform both methods with an error 40% lower than DROID-SLAM and 64% lower than DROID-VO. COLMAP takes 2 days to complete the 16 sequences and typically produces broken reconstructions which lead to large errors in evaluation.

[1]`https://github.com/stevenlovegrove/Pangolin`



(a) Results on the TartanAir (Wang et al., 2020b) validation split. Our method gets an AUC of 0.80 compared to 0.71 for DROID-SLAM while running 4x faster.



(b) Runtime distribution on EuRoC. Our system averages 60fps with each new frame taking ∼17ms to process. The distribution is very centralized and never drops below 50fps.

	ME 000	ME 001	ME 002	ME 003	ME 004	ME 005	ME 006	ME 007	MH 000	MH 001	MH 002	MH 003	MH 004	MH 005	MH 006	MH 007	Avg
ORB-SLAM3 (Campos et al., 2021)	13.61	16.86	20.57	16.00	22.27	9.28	21.61	7.74	15.44	2.92	13.51	8.18	2.59	21.91	11.70	25.88	14.38
DSO (Engel et al., 2017)	9.65	3.84	12.20	8.17	9.27	2.94	8.15	5.43	9.92	0.35	7.96	3.46	-	12.58	8.42	7.50	7.32
COLMAP (Schonberger & Frahm, 2016)	15.20	5.58	10.86	3.93	2.62	14.78	7.00	18.47	12.26	13.45	13.45	20.95	24.97	16.79	7.01	7.97	12.50
DROID (Teed & Deng, 2021)	0.17	**0.06**	0.36	0.87	1.14	**0.13**	1.13	**0.06**	**0.08**	0.05	**0.04**	**0.02**	**0.01**	0.68	0.30	**0.07**	0.33
DROID-VO	0.22	0.15	0.24	1.27	1.04	0.14	1.32	0.77	0.32	0.13	0.08	0.09	1.52	0.69	0.39	0.97	0.58
Ours-60fps	**0.16**	0.11	**0.11**	**0.66**	**0.31**	0.14	**0.30**	0.13	0.21	**0.04**	**0.04**	0.08	0.58	**0.17**	**0.11**	0.15	**0.21**

Table 1: Results on the TartanAir monocular test split. Results are reported as ATE with scale alignment. For our method, we report the median of 5 runs.

5.2 EuRoC MAV (Burri et al., 2016)

In Tab. 2 we benchmark on the EuRoC (Burri et al., 2016) dataset and compare to other visual odometry methods including SVO (Forster et al., 2014), DSO (Engel et al., 2017) and the visual odometry version of DROID-SLAM (Teed & Deng, 2020). Video from the EuRoC benchmark is recorded at 20fps. Like DROID-SLAM, we skip every other frame, doubling the effective frame rate of the system. We benchmark two configuration settings: Ours-60fps uses 96 patches per image and a 10 frame optimization window and Ours-100fps uses 48 patches and a 7 frame optimization window. These runtimes are near-constant since our method uses a constant number of FLOPS per-frame, unlike prior methods which slow down and make more keyframes during fast motion. Following prior work, we run our system five times on each sequence, randomly sampling a different collection of patches, and report the median result. Our 3x real-time system outperforms prior work on the majority of the EuRoC sequences. The average error is 43% lower than DROID-VO (Teed & Deng, 2021). Even the 100fps system outperforms DROID-VO on most video.

5.3 ICL-NUIM (Handa et al., 2014)

In Tab. 3 we benchmark on the ICL-NUIM (Handa et al., 2014) dataset and compare to other visual odometry and SLAM methods including SVO (Forster et al., 2014), DSO (Engel et al., 2017) and DROID-SLAM (Teed & Deng, 2021). Video from the ICL-NUIM benchmark is recorded at 30fps. We evaluate the same 60fps and 100fps configurations of our method as on EuRoC MAV dataset in Sec. 5.2. We also run our system five times on each sequence, randomly sampling a different collection of patches, and report the median result. Our 3x real-time system outperforms prior work on the majority of the ICL-NUIM sequences. The average error of our 100fps model is 51% lower than DROID-SLAM (Teed & Deng, 2021) and 32% lower than SVO (Forster et al., 2014). Our 100fps and 60fps systems perform similarly.

5.4 Ablations

We perform ablation experiments on the TartanAir validation split and show results in Fig. 7. We use the same parameter settings in all experiments with augmentation disabled. We run each ablation experiment three times on the validation split and aggregate the results.

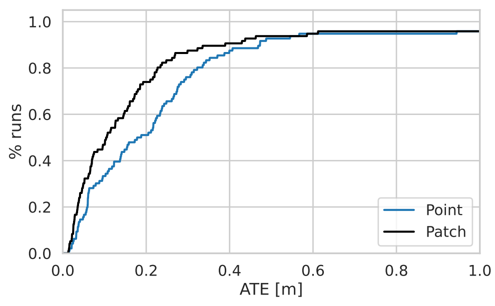
***Point vs Patch Features**:* In Fig. 7 (left), we demonstrate the importance of the patch over simply using point features (i.e 1x1 patches). The patch features encode local context which is lacking with

	MH01	MH02	MH03	MH04	MH05	V101	V102	V103	V201	V202	V203	Avg
TartanVO (Wang et al., 2020a)	0.639	0.325	0.550	1.153	1.021	0.447	0.389	0.622	0.433	0.749	1.152	0.680
SVO (Forster et al., 2014)	0.100	0.120	0.410	0.430	0.300	0.070	0.210	-	0.110	0.110	1.080	0.294
DSO (Engel et al., 2017)	**0.046**	**0.046**	0.172	3.810	**0.110**	0.089	**0.107**	0.903	**0.044**	0.132	1.152	0.601
DROID-VO (Teed & Deng, 2021)	0.163	0.121	0.242	0.399	0.270	0.103	0.165	0.158	0.102	0.115	**0.204**	0.186
Ours-60fps	0.087	0.055	**0.158**	**0.137**	0.114	**0.050**	0.140	**0.086**	0.057	**0.049**	0.211	**0.105**
Ours-100fps	0.101	0.067	0.177	0.181	0.123	0.053	0.158	0.095	0.095	0.063	0.310	0.129

Table 2: Monocular SLAM on the EuRoC datasets, ATE[m] compared to other visual odometry methods.

	Living Room 0	Living Room 1	Living Room 2	Living Room 3	Office Room 0	Office Room 1	Office Room 2	Office Room 3	Avg
ORB-SLAM2 (Mur-Artal & Tardós, 2017)	0.461	0.172	0.806	N/A	0.589	0.813	1.000	N/A	N/A
DROID (Teed & Deng, 2021)	0.008	0.027	0.039	0.012	**0.065**	0.025	0.858	0.481	0.189
DROID-VO (Teed & Deng, 2021)	0.010	0.123	0.072	0.032	0.095	0.041	0.842	0.504	0.215
SVO (Forster et al., 2014)	0.02	0.07	0.09	0.07	0.34	0.28	0.14	**0.08**	0.136
DSO (Engel et al., 2017)	0.01	0.02	0.06	0.03	0.21	0.83	0.36	0.64	0.270
DSO (Realtime) (Engel et al., 2017)	0.02	0.03	0.33	0.06	0.29	0.64	0.23	0.46	0.258
Ours-60fps	**0.006**	**0.006**	0.023	**0.010**	0.067	**0.012**	**0.017**	0.635	0.097
Ours-100fps	0.008	0.007	**0.021**	**0.010**	0.071	0.015	0.018	0.593	**0.093**

Table 3: Results on the ICL-NUIM dataset benchmark. Results are reported as ATE with scale alignment. For our method, we report the median of 5 runs. SVO and DSO results are from Forster et al. (2016). ORB-SLAM2 results are from Duan et al. (2021) (only partial results were provided). Despite our best efforts, we were unsuccessful in running monocular ORB-SLAM3: The official software would not produce any keyframes while running.



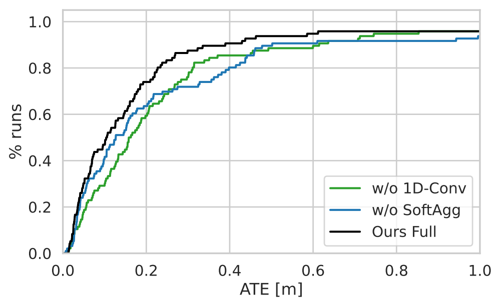


Figure 7: Ablation experiments. (Left) We show the importance of using patches over point features. (Right) Removal of different components of the update operator degrades accuracy.

point features. The additional information stored in the correlation features allows for more precise tracking.

Update Operator: In Fig. 7 (right), we test the effect of removing various components from the update operator. Both removing 1D-Convolution and Softmax-Aggregation degrade performance on the validation set.

Timing: The default configuration of our system runs at 60fps. We plot the distribution of runtimes over multiple frames and runs in Fig. 6b. The design of our system leads to very little variance in latency.

6 Conclusion

DPVO is a new deep visual odometry system built using a sparse patch representation. It is accurate and efficient, capable of running at 2x-5x real-time frame rates with minimal memory requirements.

7 Reproducibility Statement

In the Appendix, we provide a link to an anonymized repository of our code which reproduces our results.

References

Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. Ceres Solver, 3 2022. URL `https://github.com/ceres-solver/ceres-solver`.

Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 32(6):1309–1332, 2016.

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*, 37(6):1874–1890, 2021.

Christopher B Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal correspondence network. *Advances in neural information processing systems*, 29, 2016.

Jan Czarnowski, Tristan Laidlow, Ronald Clark, and Andrew J Davison. Deepfactors: Real-time probabilistic dense monocular slam. *IEEE Robotics and Automation Letters*, 5(2):721–728, 2020.

Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. In *Computer Vision, IEEE International Conference on*, volume 3, pp. 1403–1403. IEEE Computer Society, 2003.

Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

Frank Dellaert, Michael Kaess, et al. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139, 2017.

Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 224–236, 2018.

ZhiMin Duan, YingWen Chen, HuJie Yu, BoWen Hu, and Chen Chen. Rgb-fusion: Monocular 3d reconstruction with learned depth prediction. *Displays*, 70:102100, 2021.

Mihai Dusmanu, Johannes L Schönberger, and Marc Pollefeys. Multi-view optimization of local feature geometry. In *European Conference on Computer Vision*, pp. 670–686. Springer, 2020.

Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pp. 834–849. Springer, 2014.

Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):611–625, 2017.

Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE international conference on robotics and automation (ICRA)*, pp. 15–22. IEEE, 2014.

Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. Georgia Institute of Technology, 2015.

Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2016.

Giorgio Grisetti, Rainer Kümmerle, Hauke Strasdat, and Kurt Konolige. g2o: A general framework for (hyper) graph optimization. In *Proceedings of the IEEE international conference on robotics and automation (ICRA), Shanghai, China*, pp. 9–13, 2011.

A. Handa, T. Whelan, J.B. McDonald, and A.J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *IEEE Intl. Conf. on Robotics and Automation, ICRA*, Hong Kong, China, May 2014.

Stefan Leutenegger, Paul Furgale, Vincent Rabaud, Margarita Chli, Kurt Konolige, and Roland Siegwart. Keyframe-based visual-inertial slam using nonlinear optimization. *Proceedings of Robotis Science and Systems (RSS) 2013*, 2013.

Philipp Lindenberger, Paul-Edouard Sarlin, Viktor Larsson, and Marc Pollefeys. Pixel-perfect structure-from-motion with featuremetric refinement. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5987–5997, 2021.

Zhixiang Min and Enrique Dunn. Voldor+ slam: For the times when feature-based or direct methods are not good enough. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13813–13819. IEEE, 2021.

Zhixiang Min, Yiding Yang, and Enrique Dunn. Voldor: Visual odometry from log-logistic dense optical flow residuals. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4898–4909, 2020.

Anastasios I Mourikis, Stergios I Roumeliotis, et al. A multi-state constraint kalman filter for vision-aided inertial navigation. In *ICRA*, volume 2, pp. 6, 2007.

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*, 33(5):1255–1262, 2017.

Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4938–4947, 2020.

Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4104–4113, 2016.

Jiaming Sun, Zehong Shen, Yuang Wang, Hujun Bao, and Xiaowei Zhou. Loftr: Detector-free local feature matching with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8922–8931, 2021.

Chengzhou Tang and Ping Tan. Ba-net: Dense bundle adjustment network. *arXiv preprint arXiv:1806.04807*, 2018.

Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European conference on computer vision*, pp. 402–419. Springer, 2020.

Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in Neural Information Processing Systems*, 34:16558–16569, 2021.

Prune Truong, Martin Danelljan, Luc Van Gool, and Radu Timofte. Learning accurate dense correspondences and when to trust them. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5714–5724, 2021.

Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(04):376–380, 1991.

Lukas Von Stumberg, Vladyslav Usenko, and Daniel Cremers. Direct sparse visual-inertial odometry using dynamic marginalization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2510–2517. IEEE, 2018.

Rui Wang, Martin Schworer, and Daniel Cremers. Stereo dso: Large-scale direct sparse visual odometry with stereo cameras. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3903–3911, 2017.

Wenshan Wang, Yaoyu Hu, and Sebastian Scherer. Tartanvo: A generalizable learning-based vo. *arXiv preprint arXiv:2011.00359*, 2020a.

Wenshan Wang, Delong Zhu, Xiangwei Wang, Yaoyu Hu, Yuheng Qiu, Chen Wang, Yafei Hu, Ashish Kapoor, and Sebastian Scherer. 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, 2020b.

A APPENDIX

A.1 ANONYMIZED CODE

We provide an anonymized version of our code which reproduces our results:

`https://github.com/a46068902/DPVO_ICLR`

It includes interactive visualizations.

A.2 VIDEO

We also provide a video of our method running on video from an iPhone:

`https://www.youtube.com/watch?v=fqch28-jEvY`

A.3 CONFIDENCE WEIGHTS

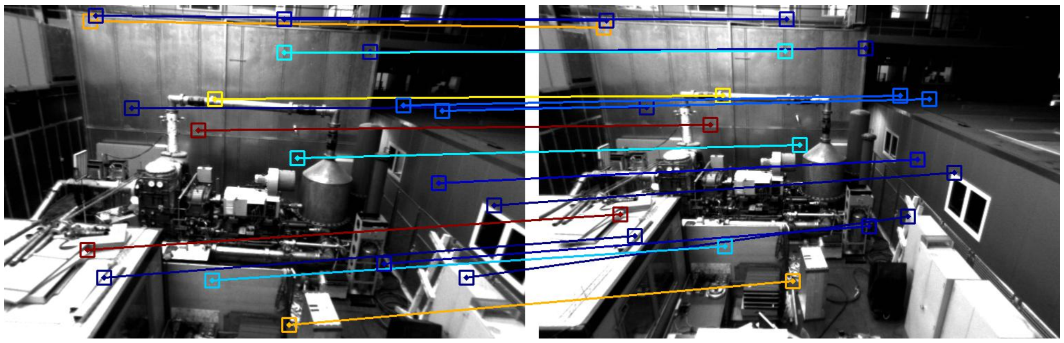


Figure 8: Factor confidence weights. Our method predicts a confidence weight $w \in \mathbb{R}^2$ bounded to $(0, 1)$ for each factor in the factor graph. Cold colors (blue) represent high confidence factors while hot colors (red) represent low confidence factors.