From Variance to Veracity: Unbundling and Mitigating Gradient Variance in Differentiable Bundle Adjustment Layers

Swaminathan Gurumurthy[1], Karnik Ram[1→2], Bingqing Chen[3],
Zachary Manchester[1], Zico Kolter[1,3]
[1]Carnegie Mellon University [2]TU Munich
[3]Bosch Center for Artificial Intelligence

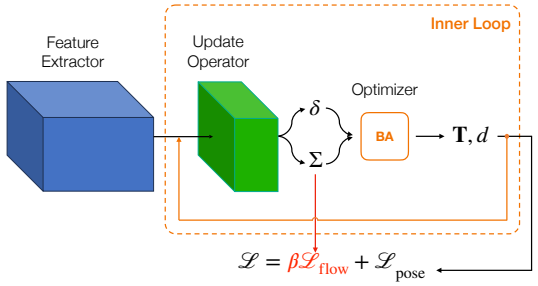
Abstract

Various pose estimation and tracking problems in robotics can be decomposed into a correspondence estimation problem (often computed using a deep network) followed by a weighted least squares optimization problem to solve for the poses. Recent work has shown that coupling the two problems by iteratively refining one conditioned on the other's output yields SOTA results across domains. However, training these models has proved challenging, requiring a litany of tricks to stabilize and speed up training. In this work, we take the visual odometry problem as an example and identify three plausible causes: (1) flow loss interference, (2) linearization errors in the bundle adjustment (BA) layer, and (3) dependence of weight gradients on the BA residual. We show how these issues result in noisy and higher variance gradients, potentially leading to a slow down in training and instabilities. We then propose a simple, yet effective solution to reduce the gradient variance by using the weights predicted by the network in the inner optimization loop to weight the correspondence objective in the training problem. This helps the training objective 'focus' on the more important points, thereby reducing the variance and mitigating the influence of outliers. We show that the resulting method leads to faster training and can be more flexibly trained in varying training setups without sacrificing performance. In particular we show 2–2.5× training speedups over a baseline visual odometry model we modify.

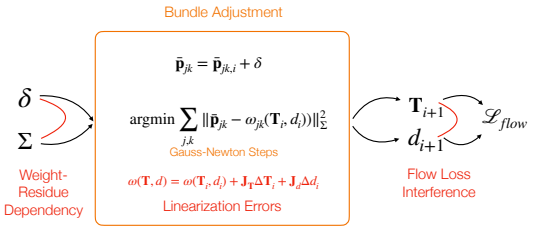
1. Introduction

Ego and exo pose estimation are essential for agents to safely interact with the physical world. These tasks have a long history of being tackled using geometry-based optimizaton [23, 24, 36, 42], and in the last decade, using deep networks to directly map inputs to poses [43, 57, 59, 62]. However, both these classes of approaches have shown brittleness — not being robust to outliers in the data or having poor accuracy in unseen scenes.

More recently, approaches that combine the best of both worlds in *learning to optimize* have demonstrated substantially better performance than previous methods [37, 54, 55]. These approaches combine a learned iterative update operator that mimics an optimization algorithm with im-



(a) SOTA pose estimation methods [37, 54, 55] tightly-couple learned front-ends with traditional BA optimizers. However, they can be slow to converge.



(b) We identify three factors that lead to high variance in gradients during their training.

Figure 1. We propose a simple, yet effective solution to stabilize and speed-up the training of SOTA pose estimation methods. (b) We first analyze the causes for their instability related to variance in their gradients, and (a) then mitigate them by using weights from the inner-loop optimization to weigh the correspondence outer objective, which leads to improved performance.

Corresponding author: gauthamsindia95@gmail.com
Code: https://github.com/swami1995/V2V

plicit layers that enforce known geometric constraints on the outputs. This general architecture has appeared across many tasks even beyond pose estimation [1, 2, 9, 27], where in each case an accurate and robust task-specific optimization solver is learned. In [37, 54, 55], for the task of pose estimation, a recurrent network that iteratively updates pose and depth is learned through a differentiable weighted bundle adjustment (BA) layer that constrains the updates. Feature correspondences are also iteratively refined together with the poses, thereby dynamically removing outliers and leading to better accuracy.

Although these methods achieve state-of-the-art (SOTA) results, they take exceedingly long to train. [54] mention that DROID-SLAM takes 1.5 weeks to train with 4x RTX 3090, while [55] mention that DPVO takes 3.5 days to train on a RTX 3090. Likewise, in our experiments, training the object pose estimation method from [37] took 1 week with 2x RTX 6000 for the smallest dataset reported in their paper.

In this paper, we first investigate the reasons for the slow training convergence speeds of these methods, using *deep patch visual odometry* (DPVO) [55] as an example problem setting for this analysis. We find that the bundle adjustment layer and the associated losses used in this setting lead to a high variance in the gradients. We identify three reasons contributing to the high variance. First, improper credit assignment arising from the specific choice of flow loss used which leads to interference between the gradients of outlier and inlier points. Second, improper credit assignment arising from the linearization issues in the bundle adjustment layer. And lastly, the dependence of the weight gradients on the residual of the BA objective resulting in the outliers dominating those gradients. We show how each of these problems lead to an increase in the gradient variance.

Next, we leverage the analysis to propose a surprisingly simple solution to reduce the variance in gradients by weighting the flow loss according to the 'importance' of the points for the problem, resulting in significant improvements in training speed and stability while achieving better pose estimation accuracy. We also experiment with other variance reduction techniques and demonstrate the superior performance of our proposed solution (Appendix Sec. 8.7). Using DPVO as an example, we demonstrate 2-2.5x speedups with these simple modifications. Furthermore, we show that the modifications also make the training less sensitive to specific training setups. As a result, we are able to train in a non-streaming setting, while reaching similar accuracies in the streaming setting, thereby leading to a further 1.2-1.5x speedup in training. Lastly, we apply the modifications to DROID-SLAM [54] with little hyperparameter tuning to show that the proposed modifications transfer to a completely new pipeline providing similar speedups and stable training. Furthermore, we show that our best models achieve about 50% improvement on the TartanAir validation set and a 24% improvement on the TartanAir test set. To summarize, the contributions of this paper are as follows:

- We identify three candidate reasons for high variance in the gradients when differentiating through the BA problems for Visual Odometry (VO) and SLAM and show how they are all affected by the presence of outliers.
- Using DPVO [55] as an example VO pipeline, we propose a simple modification to the loss function that reduces the variance in the gradients by mitigating the effect of outliers on the objective.
- We show that the above modification results in significant speedups and improvements in accuracy of the model on the TartanAir [58] validation and test splits used in the CVPR 2020 challenge. Further, we show that the modifications can be applied out-of-the-box to other settings/methods that use differentiable BA layers, such as DROID-SLAM and the non-streaming version of DPVO to obtain similar benefits.

2. Related Work

Pose Estimation using Deep Learning. A large body of works have tackled pose estimation, and we describe a few representative works that use deep learning here. For a broader overview, we refer the reader to [13, 17, 25]. [57, 59, 62] proposed deep networks to directly estimate ego pose between pairs of frames. [34, 40, 48, 53] integrate learned representations (features or depth) into traditional ego-pose estimation pipelines. [19, 52, 60] imposed geometric constraints on ego-pose network outputs via differentiable optimization layers. Similar approaches have been proposed for the task of multi-object pose estimation where 2D-3D correspondences are directly regressed [43, 45] and then passed through a differentiable PnP solver [16] for pose estimation. Overall, these works showed that deep learning could be applied to these tasks but fell short in accuracy and generalization.

Optimization-inspired iterative refinement methods have been applied to ego-pose [18, 32, 56, 61] and exo-pose estimation [31, 35] where the network iteratively refines its pose estimates as an update operator in order to satisfy geometric constraints. More recently, methods that iteratively refine poses and correspondences in a tightly-coupled manner have been proposed [37, 54, 55]. In these works, a network predicts patch correspondences [55] or dense flow [37, 54] which are then updated together with poses and depths in an alternating manner where one feeds into the other through differentiable geometric operations. In addition to correspondences, these methods also predict weights for the correspondences which have been shown to be important for pose estimation accuracy in many independent works [11, 33, 41, 46]. Overall, these iterative methods have achieved impressive performance in terms of accuracy

and generalization, but they still need large GPU memories [55] and their training times are prohibitively long which has limited their adoption for research.

Challenges with Implicit Optimization Layers. With the advent of implicit layers, it is possible to incorporate an optimization problem as a differentiable layer [3, 4, 44], which captures complex behaviours in a neural network. The BA layer [52] used in this work is an instance of such layers. In the forward pass, an implicit optimization layer solves a regular optimization problem given the current estimate of problem parameters. In the backward pass, one differentiates through the KKT conditions of the optimization problem to update the problem parameters.

While these implicit optimization layers boast expressive representational power, there exist challenges with such layers. Firstly, these problems naturally take on a bilevel structure, where the inner optimization learns the problem parameters and the outer problem optimizes for the decision variables given the current estimation of problem parameters. As a result, these problems are inherently hard to solve[4, 5, 29], as their easiest instantiation, e.g., linear programs for both inner and outer problems, can be non-convex [8]. While the convergence issues may be alleviated by techniques such as using good initialization [4] or robust solvers, there does not exist a general solution to the authors' best knowledge. Secondly, a range of numerical issues can arise from implicit optimization layers. The gradients derived from KKT conditions are only valid at fixed points of the problem. In practice, the solver may need to run long enough to reach a fixed point or a fixed point may not exist at all [5, 22]. The problem may be ill-conditioned due to reasons such as stiffness or discontinuities from physical systems [51] or compounding of gradients in chaotic systems [39]. A number of problem-specific solutions have been proposed [29, 30, 51] to these problems. For example, [6, 51] use zeroth-order methods to deal with non-smoothness and non-convexity in the problem. [29, 30] use interior point relaxations to smooth the discontinuities. Similarly, [10, 22] use penalty-based relaxations to handle the discontinuities. It's also common to regularize the inner problem during the backward pass to deal with ill-conditioning [5, 28]. However, given the vastness of the problems, we are of the opinion that this is still a broadly under-studied area.

3. Background

In this section, we review the approach of DPVO [55] for iterative ego-pose estimation, which serves as an example setting for all our analysis and experiments.

Feature Extraction. A scene, as observed from an input video, is represented as a set of camera poses $\mathbf{T}_j \in \mathbb{SE}(3)$ and square image patches $\mathbf{P}_k$. Patches are created by randomly sampling 2D locations in the image and extracting $p \times p$ feature maps centered at these coordinates $\mathbf{p}_k$. A bipartite patch-frame graph is constructed by placing an edge between every patch k and each frame j within distance r of the patch source frame. The reprojections of a patch in all of its connected frames form the trajectory of the patch.

Update Operator. The update operator iteratively updates the optical flow of each patch over its trajectory. The operator updates the embedding of each edge (k, j) of the patch graph via temporal convolutions and message passing. These updated embeddings are used by two MLPs to predict flow revisions $\delta_{jk} \in \mathbb{R}^2$ and confidence weights for each patch $\Sigma_{jk} \in \mathbb{R}^2$ between $[0, 1]$. The flow revisions are used to update the reprojected patch coordinates $\bar{\mathbf{p}}_{jk} := \bar{\mathbf{p}}_{jk} + \delta_{jk}$, which are passed to a differentiable BA layer along with their confidence weights Σ_{jk}.

Differentiable Bundle Adjustment. The bundle adjustment (BA) layer solves for the updated poses and depths that are geometrically consistent with the predicted flow revisions. The BA layer operates on a window of the patch graph to update the camera poses and patch depths, while keeping the revised patch coordinates $\bar{\mathbf{p}}_{jk}$ fixed. The BA objective is as follows:

$$\min_{\mathbf{T}_{ij}, d_k} \sum_{(k,j)} ||\bar{\mathbf{p}}_{jk} - \Pi(\mathbf{T}_{ij}, \Pi^{-1}(\mathbf{p}_k, d_k))||^2_{\Sigma_{jk}} \tag{1}$$

where Π denotes the projection operation, d_k denotes the depth of the k^{th} patch in the source frame i, and $\mathbf{T}_{ij}$ is the relative pose $\mathbf{T}_i \mathbf{T}_j^{-1}$. This objective is optimized using two Gauss-Newton iterations. The optimized poses and depths are then passed back to the update operator to revise the patch coordinates, and so on in an alternating manner.

Training Loss. The network is supervised using a flow loss and pose loss computed on the intermediate outputs of the BA layer. The flow loss computes the distance between the ground truth patch coordinates and estimated patch coordinates over all the patches and frames:

$$\mathcal{L}_{\text{flow}} = \sum_{j,k} \|\mathbf{p}^*_{jk} - \hat{\mathbf{p}}_{jk}\|_2 \tag{2}$$

where $\hat{\mathbf{p}}_{jk} = \Pi(\mathbf{T}_{ij}, \Pi^{-1}(\mathbf{p}_k, d_k))$ and $\mathbf{p}^*_{jk}$ is the corresponding reprojection of patch k in frame j using the ground truth pose and depth. Note that this loss amounts to a difference in the patch coordinates and not in the flows as the source patch coordinates in each flow term cancel out.

The pose loss is the error between the ground truth poses $\mathbf{G}$ and estimated poses $\mathbf{T}$ for every pair of frames (i, j):

$$\mathcal{L}_{\text{pose}} = \sum_{(i,j)} \|\text{Log}_{\mathbb{SE}(3)}[(\mathbf{G}_i.\mathbf{G}_j^{-1})^{-1}.(\mathbf{T}_i.\mathbf{T}_j^{-1})]\|_2 \tag{3}$$

The total loss is a weighted combination of the flow loss and pose loss,

$$\mathcal{L} = 10\mathcal{L}_{\text{flow}} + 0.1\mathcal{L}_{\text{pose}} \tag{4}$$

The original DPVO model is trained on random sequences of 15 frames, where the first 8 frames are used together for initialization and the subsequent frames are added one at a time. Their model is trained for 240K iterations using 19GB of GPU memory which takes 3.5 days on an RTX 3090. A total of 18 iterations of the update operator is applied on each sequence, where the first 8 iterations are applied during initialization as a batch-optimization, and the subsequent iterations are for every new, added frame. In our paper, we refer to these update iterations as the 'inner-loop optimization', this mode of training as the 'streaming' setting, and training models in our experiments to only batch-optimize the first 8 frames as the 'non-streaming' setting.

4. Factors Affecting Training Convergence

In this section, we identify three possible causes for slow training convergence. We show how each of these result in noisier/higher variance gradients during training, and consequently result in instabilities and slowdowns.

4.1. Flow loss interference

The flow loss defined in Eq. 2 operates on the reprojected patch coordinates $\hat{\mathbf{p}}_{jk}$ which are computed using the optimized poses $\mathbf{T}_i, \mathbf{T}_j$ and depth d_k outputs from the BA layer. Thus, the gradient of the loss with respect to d_k (and similarly for poses $\mathbf{T}_i, \mathbf{T}_j$) can be written as follows,

$$\nabla_{d_k}\mathcal{L}_{\text{flow}} \propto \sum_j \nabla_{d_k}\Pi(\mathbf{T}_{ij}, (\mathbf{p}_k, d_k)). \tag{5a}$$

$$\nabla_{T_i}\mathcal{L}_{\text{flow}} \propto \sum_{k,j} \nabla_{T_i}\Pi(\mathbf{T}_{ij}, (\mathbf{p}_k, d_k)). \tag{5b}$$

Thus, the gradients with respect to each reprojected patch $\hat{\mathbf{p}}_{jk}$ gets aggregated in the computation graph at the corresponding depth d_k (likewise for poses $\mathbf{T}_i, \mathbf{T}_j$) at the output of the BA layer. This becomes problematic when a significant fraction of the projections are noisy/outliers, as the noisy/outlier gradients would dominate the inlier gradients in the sum in Eq. 5a, leading to more noise in the total gradient estimate.

Since these gradients are also backpropagated through the BA layer, it results in noisy gradient estimates for the network parameters as well. Specifically, in the BA layer, each d_i/$\mathbf{T}_i$ are again a function of all the predicted flows and weights associated with that point/frame. Thus, the same noisy gradient computed at d_i/$\mathbf{T}_i$, gets backpropagated to all the associated points. This leads to the gradient estimates being noisy even at the 'good' predictions by the network.

4.2. Linearization errors in BA gradient

Given gradient estimates at the output poses and depth of the BA layer $\nabla_d\mathcal{L}, \nabla_{\mathbf{T}}\mathcal{L}$, the gradients with respect to its input flows and weights are computed as follows:

$$\begin{aligned}\nabla_\delta\mathcal{L} = &-(\nabla_{\mathbf{T}}\mathcal{L})^T(\mathbf{J}_{\mathbf{T}}^T\Sigma\mathbf{J}_{\mathbf{T}})^{-1}\mathbf{J}_{\mathbf{T}}^T\Sigma \\ &-(\nabla_d\mathcal{L})^T(\mathbf{J}_d^T\Sigma\mathbf{J}_d)^{-1}\mathbf{J}_d^T\Sigma\end{aligned} \tag{6a}$$

$$\begin{aligned}\nabla_\Sigma\mathcal{L} = &-(\nabla_{\mathbf{T}}\mathcal{L})^T(\mathbf{J}_{\mathbf{T}}^T\Sigma\mathbf{J}_{\mathbf{T}})^{-1}\mathbf{J}_{\mathbf{T}}^T\text{diag}(\mathbf{r}) \\ &-(\nabla_d\mathcal{L})^T(\mathbf{J}_d^T\Sigma\mathbf{J}_d)^{-1}\mathbf{J}_d^T\text{diag}(\mathbf{r})\end{aligned} \tag{6b}$$

where, $\mathbf{r} = (\bar{\mathbf{p}}_{kj} - \hat{\mathbf{p}}_{kj})$ is the bundle adjustment residual, $\mathbf{J}_d$ and $\mathbf{J}_{\mathbf{T}}$ are the jacobians of the projection $\Pi(\mathbf{T}_{ij}, \Pi^{-1}(\mathbf{p}_k, d_k))$ with respect to depth d and pose $\mathbf{T}$ respectively. This expression can be derived by applying the implicit function theorem (Theorem 1B.1) [21], on the BA problem as shown in Appendix Sec. 8.3.

Since the projection is non-linear containing multiple multiplicative operations, we observe that the Jacobians $\mathbf{J}_d$ and $\mathbf{J}_{\mathbf{T}}$ themselves are a function of d and $\mathbf{T}$. Thus, a high variance in the initialized d or $\mathbf{T}$ naturally lead to a high variance in the Jacobians, thereby leading to a high variance in the corresponding gradients ∇_Σ and ∇_δ, which are then backpropagated through the network. In our setup, d is initialized to random values and $\mathbf{T}$ is initialized to identity. Thus, the variance from linearization is primarily contributed by the linearization around the current d.

The use of a weighted objective in the BA problem partially mitigates this issue by masking out the gradients on the flows corresponding to the outlier points (which contribute the most to this high variance). However, the high variance remains problematic especially in the initial iterations of training (when the weight estimates themselves are not very accurate) and in the initial iterations of the inner-loop optimization when a large fraction of the depth and pose estimates are inaccurate.

4.3. Dependence of weight gradients on the BA residual

In the previous section, we discussed the effect of outliers on the BA linearization and consequently on the gradients. However, outliers in the BA problem contribute to an increase in gradient variance in a more straightforward way. Specifically, they have a direct effect on the gradient of the weights, as can be seen from the expression in Eq. 6b. The expression shows the direct dependence of the weight gradients on the residual, $\mathbf{r} = (\bar{\mathbf{p}}_{jk} - \hat{\mathbf{p}}_{jk})$, of the BA problem. Thus, the presence of high residual points in the optimization problem result in high variance in the weight gradients.

In fact, the presence of outliers also biases the weight gradients towards highly positive values as the training objective tries to reduce the influence of the outliers. This

consequently leads to a collapse in the weight distribution. However, we observe that a straightforward fix used by prior work[54, 55], i.e, clipping the magnitude of gradient passing through the weights easily mitigates this bias. We discuss more details on this effect with a simple illustrative example in Appendix Sec. 8.4.

To summarize, the above section highlights various aspects of the existing setup that contribute to noisy/high variance gradients. The noise and high variance in gradient estimates leads to ineffective parameter updates, thereby leading to training instabilities and slowdown. Furthermore, it's also important to note that the aforementioned effects exacerbate each other. For example, worse weight estimates result in bad BA outputs, which in turn contribute to worsening the flow loss interference and BA linearization errors, which further leads to noisier gradients thereby slowing down weight/flow updates, thus repeating the vicious cycle. By the same argument, mitigating either of these effects can also provide significant improvements on other problems!

5. A very simple solution: Weighted flow loss

We start with observing that all three problems mentioned in the previous section get exacerbated by the presence of outliers or computing gradients through outliers. So the natural question is if there exist obvious solutions to mask out the outliers in the outer training problem.

One of the tricks used by [54, 55] already partially accounts for this in the pose loss, i.e, they do not include the pose loss for the first couple of inner-loop iterations, thereby mitigating some of the issues discussed in Sec. 4.2. This simple modification in [54, 55] seems to provide a significant boost in training speeds as we show in our ablation experiments in Appendix Sec. 8.6.

Similar heuristics for the flow loss are harder to find as the depth/flow estimates of a significant fraction of points are bad even at the latter inner-loop iterations. Conventionally, SLAM and visual odometry problems define heuristic kernels on the flow residuals [7, 15] depending on the expected distribution of residuals/errors to trade-off between robustness and accuracy. Unfortunately, coming up with a similar simple/consistent heuristic to define 'outliers' in the outer training problem is more challenging as the errors and distribution of errors vary across examples, training iterations and inner optimization iterations. This requires a heuristic that adapts to the specific example, training convergence, and inner-loop optimization iteration.

Conveniently, we find that the weights learnt by the inner update operator for the bundle adjustment problem satisfy all these properties as they adapt online with the changing distribution of errors/residuals. Moreover, empirically we observe that the network learns a reasonable weight distribution very early on in training, while adapting the weight distribution rapidly to any changes in flows. Thus, we observe that using these weights to weight the flow loss works surprisingly well. The resulting flow loss is as follows.

$$\mathcal{L}_{\text{flow}} = \sum_{j,k} \|\mathbf{p}^*_{jk} - \hat{\mathbf{p}}_{jk}\|_{\Sigma^{\perp}_{jk}} \tag{7}$$

where $\perp$ denotes the stop gradient operator to prevent the objective from directly driving the weights to zero (We provide more discussion on what factors prevent these weights from collapsing to zero in Appendix Sec. 8.5). The main difference between this and Eq. 2 is that each residual in this objective is weighted by the weights Σ_{jk} predicted by the network for the inner BA problem. Intuitively, this objective incentivizes the network to focus on the points which are important for the inner optimization problem at that optimizer step / training iteration for that particular example.

Although the modification seems trivial and obvious in hindsight, we observe that it is significantly more effective than various other (more complicated) variance reduction approaches we tried (studied in Appendix Sec. 8.7). This apparent simplicity and effectiveness underscore the value of the proposed modifications!

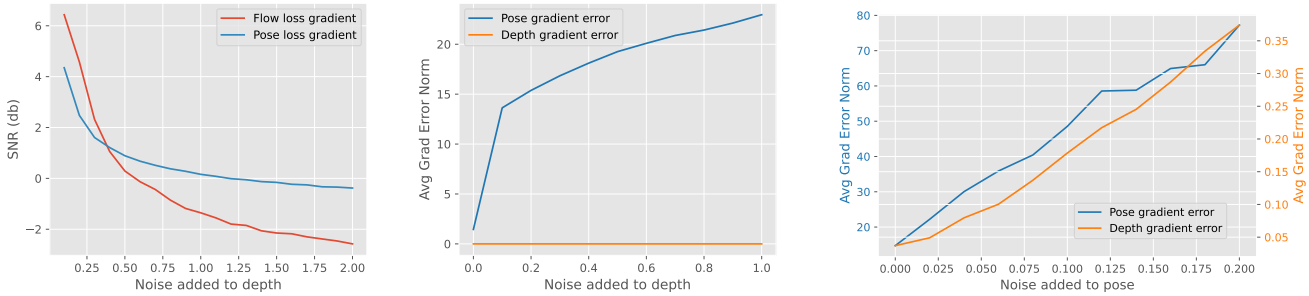
Balancing loss gradients. The introduction of the weighted flow loss changes the gradient contribution from the flow loss throughout training as the weight distribution changes. Thus, instead of using fixed coefficients to trade-off between pose and flow loss as in Eq. 4, we periodically (every 50 training iterations) update the flow loss coefficient β to ensure the gradient contributions of the pose and flow loss remain roughly equal throughout training. Given the infrequency in these updates, they barely affect the training speed and hence are cheap to compute amortized over the entire training run.

$$\beta = \frac{\|\nabla_\theta \mathcal{L}_{\text{pose}}\|_2}{\|\nabla_\theta \mathcal{L}_{\text{flow}}\|_2} \tag{8}$$

$$\mathcal{L} = \mathcal{L}_{\text{pose}} + \beta \mathcal{L}_{\text{flow}} \tag{9}$$

6. Results and Analysis

We analyze the effect of the factors discussed in Sec. 4 on the original DPVO model on the TartanAir [58] dataset. We then analyze a version trained with our proposed weighted flow objective. We show that the weighted objective helps increase the signal to noise ratio in the gradients throughout training and show the improvements in performance as a result. We also evaluate the pose estimation performance of this version on the TartanAir [58], EuRoC [12], and TUM-RGBD [50] benchmarks. We use the average absolute trajectory error (ATE) after $\text{Sim}(3)$ alignment of the trajectories, as the evaluation metric for pose estimation.



(a) Impact of depth noise on linearization. (b) Impact of depth noise on flow loss gradients. (c) Impact of pose noise on flow loss gradients.

Figure 2. (a) We compute the signal-to-noise ratio (SNR) in the loss gradients as we artificially add depth noise while linearizing the BA problem for gradient computation. We observe that the SNR in the flow loss deteriorates rapidly indicating its sensitivity to linearization errors. (b) We artificially add noise to a subset of depths right before the flow loss computation. We show the average gradient errors on all the pose and 'clean' depth variables as a result of the added noise. We see a monotonic increase in gradient error in pose gradients as we increase the noise added showing the impact 'outliers' have on the gradients of even the 'inlier' variables. (c) Similar to (b), here we add noise to the the first frame's pose and show the gradient errors on the rest of the frames and depths.

6.1. Analyzing factors affecting gradient variance

To understand the impact of linearization on the gradient variance (Sec. 4.2), we analyze the impact of adding noise to the depth used to compute the Jacobians in the BA problem. We leave the rest of the forward and backward pass unaltered and only add noise to the depth while computing the linearization for the backward pass in the BA problem. This helps us isolate the effects of linearization on the gradient computations. Specifically, Fig. 2a shows the signal-to-noise ratio (SNR) of the flow and pose loss gradients with respect to δ with increasing levels of noise. The SNR is computed assuming the no-depth-noise gradient as the true signal and treating any deviations from it as noise. The SNR computation details are provided in Appendix Sec. 8.8. This yields two interesting observations. First, the SNR deteriorates rapidly in the beginning indicating that the gradients are indeed sensitive to the noise in the iterates used for linearization. Second, the SNR in the flow loss gradients is high initially, but deteriorates rapidly compared to the pose loss gradients with increasing noise. This highlights the need to make flow loss robust to noisy points.

To analyze the effect of flow loss on the gradient noise (Sec. 4.1), we introduce noise on a few depth points or a single frame pose right before computing the flow loss and study the effect of the noise on the gradients of all the other points/poses. Fig. 2c shows a monotonic increase in gradient errors on the depths as well as all poses as we increase the noise added to the first pose. Likewise, Fig. 2b shows the monotonic increase in gradient errors of all poses as we add increasing amounts of noise to all depths on the first frame. This shows how outliers with increasingly large errors can have an increasingly adverse effect on the gradients of the non-outlier points/frames as well. The gradient errors are computed as the average L2 norm of the deviation in gradient from the no-noise gradients.

We analyze the weight residual dependence (Sec. 4.3) and the resulting variance / bias in Appendix Sec. 8.4, as its connections to the use of weighted loss are less direct.

6.2. Effect of the weighted flow loss on training

To understand the effect of the weighted flow loss on the variance of the gradients, we study the SNR of the gradients on the flow network parameters. Fig. 3 shows the SNR with the flow loss and the weighted flow loss at different points during training. The SNR computation details are provided in Appendix Sec. 8.8. The plots clearly demonstrate that the usage of weighted flow loss results in a boost in SNR throughout training. The boost is especially prominent in the initial stages of training, when the impact of outliers and noise in the pose/depth estimates are most significant. This clearly shows the promise of using the weighted flow loss instead of the regular flow loss for training.

We retrain DPVO with our modified weighted flow loss on the TartanAir dataset and show its validation error performance across training iterations. We observe in Fig. 4 that the average ATE of our method drops rapidly compared to the original. While our model takes only 80K iterations to reach an average error of ~ 0.2 m, the original model reaches the same performance at 180K iterations. In fact, while that's the peak performance reached by the base model, our model continues to improve and reaches a final convergence error of ~ 0.10 m, achieving half the base model's convergence error on the validation set. We also observe that, unlike the original model, the errors don't fluctuate rapidly over epochs and is more stable.

Further, the reduced variance in gradients allows us to train in other setups as well. For example, Fig. 5 shows the ATE of our model against the base model trained in the non-streaming setting, i.e. using just 8 frame initialization sequences instead of 15 frame sequences. This allows the

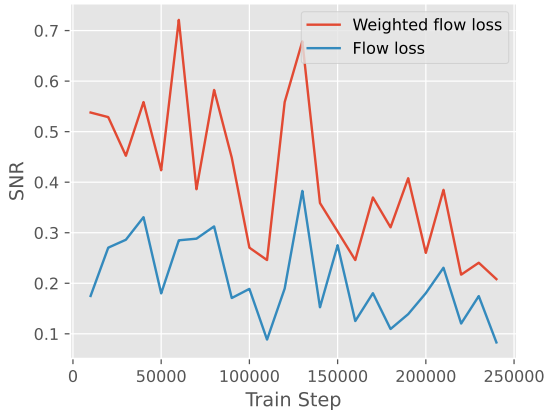


Figure 3. We compute the signal-to-noise ratio in the gradients of the flow loss and the weighted flow loss w.r.t flow network parameters at different training iterations of the base model. Specifically, we use the last linear layer's weights of the flow computation head of the network. We find that the weighted flow loss gradients have a higher SNR throughout the training. This is especially true in the initial iterations of training when the outlier count is very high.

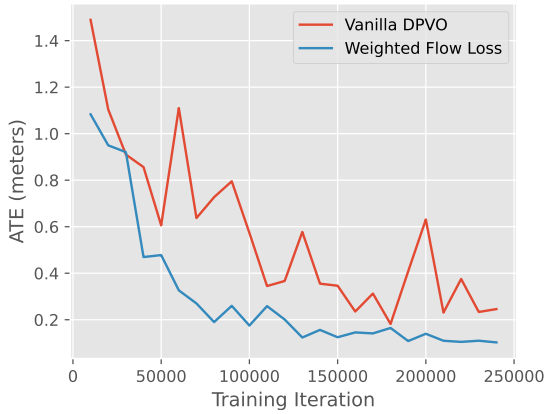


Figure 4. We observe that DPVO when trained with our weighted flow loss achieves much faster training, reaching ~ 0.2 m accuracy in only 80K iterations, and is much more stable. We report the median ATE across three trials on the validation split of TartanAir.

models to be trained faster (with per iteration cost of 0.6s vs 1.6s for the streaming version on an RTX A6000 GPU) and with lower GPU memory (7.2GB as opposed to 19.2GB GPU memory). Note that the evaluations are still done as earlier, i.e, by rolling out the model on the full validation sequences in the streaming setting. Yet, we observe that despite being trained to only batch-optimize over 8 frames, it generalizes to the streaming setting with our modified models obtaining a peak performance of $\sim$ 0.2m pose errors in 180K iterations (i.e, 2.7$\times$ faster than the base streaming model). Furthermore, we also test our modifications on DROID-SLAM, a completely different pipeline that also uses Bundle Adjustment layers with little to no hyperparameter tuning. We present the results in Appendix Sec. 8.2.

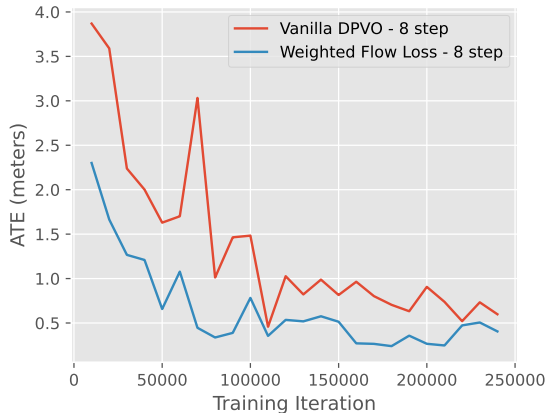


Figure 5. We retrain DPVO with and without the modified flow loss in the non-streaming batch setting and evaluate both models on validation sequences from TartanAir in the streaming setting. We observe that, beyond training faster and being more stable, the modified version generalizes better than the original model. This allows the model to be trained on shorter sequences without suffering high performance drops, thanks to the reduced gradient variance. We report the median ATE across three trials.

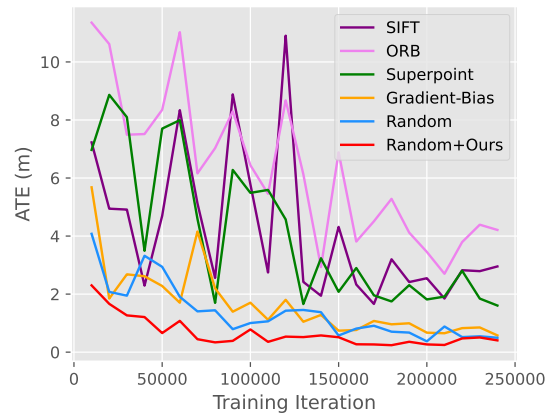


Figure 6. We retrain the original DPVO model with standard feature detection methods and observe that our method of random sampling with the modified flow objective has much improved training convergence. We report the median ATE across three trials on the validation split of TartanAir.

Again, we observe that the modifications result in significant speedups and stability during training suggesting that the methods and analysis discussed in this paper applicable broadly to approaches using differentiable BA layers.

Finally, to evaluate the ability of our modified model to weight patches effectively, we compare against other standard methods for selecting patches. Specifically, we retrain the original DPVO model with patches selected using SIFT [38], ORB[47], Superpoint[20], and naive gradient based sampling instead of the default random sampling. As shown in Fig. 6, we observe that random sampling along with the weights learned by our network is much more stable and accurate than other patch selection methods.

	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* [14]	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
COLMAP* [49]	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
DSO [24]	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
DROID-SLAM* [54]	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
DPVO	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
Ours	**0.08**	**0.05**	0.16	**0.30**	**0.27**	**0.08**	**0.20**	0.10	0.18	**0.03**	**0.03**	**0.02**	0.58	0.30	**0.08**	**0.05**	**0.16**

Table 1. ATE [m] results on the TartanAir [58] test split compared to other SLAM methods. For our method and DPVO, we report the median of 5 runs. (*) indicates the method used global loop closure optimization.

	MH01	MH02	MH03	MH04	MH05	V101	V102	V103	V201	V202	V203	Avg
TartanVO [59]	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 [26]	0.100	0.120	0.410	0.430	0.300	0.070	0.210	-	0.110	0.110	1.080	0.294
DSO [24]	**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 [54]	0.163	0.121	0.242	0.399	0.270	0.103	0.165	0.158	0.102	0.115	**0.204**	0.186
DPVO	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	**0.081**	0.067	0.171	0.179	0.115	**0.046**	0.160	0.097	0.056	0.059	0.252	0.117

Table 2. ATE [m] results on the EuRoC dataset [12] compared to other visual odometry methods. For our method and DPVO, we report the median of 5 runs. The performance of our model is similar to DPVO.

	360	desk	desk2	floor	plant	room	rpy	teddy	xyz	Avg
ORB-SLAM3 [14]	x	**0.017**	0.210	x	0.034	x	x	x	**0.009**	-
DSO [24]	0.173	0.567	0.916	0.080	0.121	0.379	0.058	x	0.036	-
DSO-Realtime [24]	0.172	0.718	0.728	0.068	0.167	0.767	x	x	0.031	-
DROID-VO [54]	0.161	0.028	0.099	**0.033**	**0.028**	**0.327**	**0.028**	0.169	0.013	0.098
DPVO	**0.135**	0.038	0.048	0.040	0.036	0.394	0.034	0.064	0.012	**0.089**
Ours	0.145	0.026	**0.044**	0.064	0.031	0.434	0.045	**0.046**	0.012	0.094

Table 3. ATE [m] results on the freiburg1 set of TUM-RGBD [50]. We evaluate *monocular* visual odometry, and is identical to the evaluation setting in DPVO [55]. For all methods, we report the median of 5 runs. (x) indicates that the method failed to track. The performance of our model is similar to DPVO.

6.3. Test results for pose estimation

We report pose estimation results on the TartanAir [58] test-split from the CVPR 2020 SLAM competition in Tab. 1, and compare to results from other baseline methods as reported in DPVO [55]. Traditional optimization-based approaches such as ORB-SLAM3 [14], COLMAP [49], DSO [24] fail to track accurately and have absolute trajectory errors (ATE) in the order of meters. Iterative learning-based DROID-SLAM [54] and its variant without global loop-closure correction (DROID-VO) show reasonable performance, but DPVO is able to show much better accuracy by only tracking a sparse number of patches instead of dense flow. Our modified version, is able to show even better accuracy with a 24% lower error on average. Morever, we observe that our model outperforms DPVO on all but two sequences in the dataset. Using the same models trained on the TartanAir train set, we also report the results on the EuRoC [12] and the TUM-RGBD [50] benchmark datasets in Tab. 2 and Tab. 3. Here, we obtain similar performance to DPVO. This suggests that, although the weighted flow loss helps improve the model accuracy on similar datasets, it doesn't resolve generalization issues related to domain shift from the TartanAir dataset to the real world.

7. Conclusions and Future work

In this paper, we analyze the high variance in gradients during the training of pose estimation pipelines that use differentiable bundle adjustment layers. We identify three plausible causes for the high variance and show how they lead to slower training and instability. We then propose a simple solution for these problems involving a weighted correspondence loss. We implement this on a SOTA VO pipeline and demonstrate improved training stability and a 2.5x training speedup. We also observe a 24% accuracy improvement on the TartanAir test split and similar accuracy as the vanilla model on other benchmarks. Unsurprisingly, the modifications don't automatically improve the model's ability to tackle distribution shift. We also observe that the depth accuracy for low-weight points, which might be important for dense SLAM approaches, deteriorates.

We see our work as an initial attempt at understanding the numerical issues stemming from the usage of bundle adjustment layers and optimization layers more broadly within deep learning pipelines. There are likely more factors contributing to issues like slower training, instability and generalization. We believe this broader area is relatively under-studied and requires more research to fully leverage the structure found in various real world problems.

References

[1] Jonas Adler and Ozan Öktem. Solving ill-posed inverse problems using iterative deep neural networks. *Inverse Problems*, 2017. 2

[2] Jonas Adler and Ozan Öktem. Learned primal-dual reconstruction. *IEEE transactions on medical imaging*, 2018. 2

[3] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019. 3

[4] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017. 3

[5] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018. 3

[6] Rika Antonova, Jingyun Yang, Krishna Murthy Jatavallabhula, and Jeannette Bohg. Rethinking optimization with differentiable simulation from a global perspective. In *Conference on Robot Learning*, pages 276–286. PMLR, 2023. 3

[7] Jonathan T. Barron. A general and adaptive robust loss function. In *CVPR*, 2019. 5

[8] Yasmine Beck and Martin Schmidt. A gentle and incomplete introduction to bilevel optimization. 2021. 3

[9] Mohak Bhardwaj, Byron Boots, and Mustafa Mukadam. Differentiable gaussian process motion planning. In *2020 IEEE international conference on robotics and automation (ICRA)*, 2020. 2

[10] Bibit Bianchini, Mathew Halm, and Michael Posa. Simultaneous learning of contact and continuous dynamics. *arXiv preprint arXiv:2310.12054*, 2023. 3

[11] Keenan Burnett, David J Yoon, Angela P Schoellig, and Timothy D Barfoot. Radar odometry combining probabilistic estimation and unsupervised feature learning. In *Robotics: Science and Systems*, 2021. 2

[12] 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*, 2016. 5, 8

[13] 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*, 2016. 2

[14] 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*, 2021. 8

[15] Nived Chebrolu, Thomas Läbe, Olga Vysotska, Jens Behley, and Cyrill Stachniss. Adaptive robust kernels for non-linear least squares problems. *IEEE Robotics and Automation Letters*, 2021. 5

[16] Bo Chen, Alvaro Parra, Jiewei Cao, Nan Li, and Tat-Jun Chin. End-to-end learnable geometric vision by backpropagating pnp optimization. In *CVPR*, 2020. 2

[17] Changhao Chen, Bing Wang, Chris Xiaoxuan Lu, Niki Trigoni, and Andrew Markham. A survey on deep learning for localization and mapping: Towards the age of spatial machine intelligence, 2020. 2

[18] Ronald Clark, Michael Bloesch, Jan Czarnowski, Stefan Leutenegger, and Andrew J Davison. Learning to solve nonlinear least squares for monocular stereo. In *ECCV*, 2018. 2

[19] Jan Czarnowski, Tristan Laidlow, Ronald Clark, and Andrew J Davison. Deepfactors: Real-time probabilistic dense monocular slam. *IEEE Robotics and Automation Letters*, 2020. 2

[20] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *CVPR Workshops*, 2018. 7

[21] Asen L Dontchev, R Tyrrell Rockafellar, and R Tyrrell Rockafellar. *Implicit functions and solution mappings: A view from variational analysis*. Springer, 2009. 4

[22] Priya L. Donti, David Rolnick, and J Zico Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021. 3

[23] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *ECCV*, 2014. 1

[24] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 2017. 1, 8

[25] Zhaoxin Fan, Yazhi Zhu, Yulin He, Qi Sun, Hongyan Liu, and Jun He. Deep learning on monocular object pose detection and tracking: A comprehensive overview. *ACM Computing Surveys*, 2022. 2

[26] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014. 8

[27] Lanke Frank Tarimo Fu and Maurice Fallon. Batch differentiable pose refinement for in-the-wild camera/lidar extrinsic calibration. In *Conference on Robot Learning*, 2023. 2

[28] Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. Jfb: Jacobian-free backpropagation for implicit networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6648–6656, 2022. 3

[29] Taylor A Howell, Simon Le Cleac'h, J Zico Kolter, Mac Schwager, and Zachary Manchester. Dojo: A differentiable simulator for robotics. *arXiv preprint arXiv:2203.00806*, 9, 2022. 3

[30] Taylor A Howell, Kevin Tracy, Simon Le Cleac'h, and Zachary Manchester. Calipso: A differentiable solver for trajectory optimization with conic and complementarity constraints. In *The International Symposium of Robotics Research*, pages 504–521. Springer, 2022. 3

[31] Shun Iwase, Xingyu Liu, Rawal Khirodkar, Rio Yokota, and Kris M Kitani. Repose: Fast 6d object pose refinement via deep texture rendering. In *ICCV*, 2021. 2

[32] Krishna Murthy Jatavallabhula, Ganesh Iyer, and Liam Paull. slam: Dense slam meets automatic differentiation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020. 2

[33] Y. Kanazawa and K. Kanatani. Do we really have to consider covariance matrices for image features? In *ICCV*, 2001. 2

[34] 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*, 2022. 2

[35] Yann Labbé, Justin Carpentier, Mathieu Aubry, and Josef Sivic. Cosypose: Consistent multi-view multi-object 6d pose estimation. In *ECCV*, 2020. 2

[36] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Ep n p: An accurate o (n) solution to the p n p problem. *IJCV*, 2009. 1

[37] Lahav Lipson, Zachary Teed, Ankit Goyal, and Jia Deng. Coupled iterative refinement for 6d multi-object pose estimation. In *CVPR*, 2022. 1, 2

[38] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 2004. 7

[39] Luke Metz, C Daniel Freeman, Samuel S Schoenholz, and Tal Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021. 3

[40] 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*, 2020. 2

[41] Dominik Muhle, Lukas Koestler, Krishna Murthy Jatavallabhula, and Daniel Cremers. Learning correspondence uncertainty via differentiable nonlinear least squares. In *CVPR*, 2023. 2

[42] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 2015. 1

[43] Kiru Park, Timothy Patten, and Markus Vincze. Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation. In *ICCV*, 2019. 1, 2

[44] Luis Pineda, Taosha Fan, Maurizio Monge, Shobha Venkataraman, Paloma Sodhi, Ricky TQ Chen, Joseph Ortiz, Daniel DeTone, Austin Wang, Stuart Anderson, Jing Dong, Brandon Amos, and Mustafa Mukadam. Theseus: A Library for Differentiable Nonlinear Optimization. *Advances in Neural Information Processing Systems*, 2022. 3

[45] Mahdi Rad and Vincent Lepetit. Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In *ICCV*, 2017. 2

[46] René Ranftl and Vladlen Koltun. Deep fundamental matrix estimation. In *ECCV*, 2018. 2

[47] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *ICCV*, 2011. 7

[48] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *CVPR*, 2020. 2

[49] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 8

[50] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, 2012. 5, 8

[51] Hyung Ju Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022. 3

[52] Chengzhou Tang and Ping Tan. BA-net: Dense bundle adjustment networks. In *ICLR*, 2019. 2, 3

[53] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *CVPR*, 2017. 2

[54] Zachary Teed and Jia Deng. DROID-SLAM: Deep visual SLAM for monocular, stereo, and RGB-d cameras. In *Advances in Neural Information Processing Systems*, 2021. 1, 2, 5, 8, 11, 12, 13

[55] Zachary Teed, Lahav Lipson, and Jia Deng. Deep patch visual odometry. *Advances in Neural Information Processing Systems*, 2023. 1, 2, 3, 5, 8, 12, 13

[56] Benjamin Ummenhofer, Huizhong Zhou, Jonas Uhrig, Nikolaus Mayer, Eddy Ilg, Alexey Dosovitskiy, and Thomas Brox. Demon: Depth and motion network for learning monocular stereo. In *CVPR*, 2017. 2

[57] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *IEEE Int. Conf. Robotics and Automation*, 2017. 1, 2

[58] 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)*, 2020. 2, 5, 8

[59] Wenshan Wang, Yaoyu Hu, and Sebastian Scherer. Tartanvo: A generalizable learning-based vo. In *Conference on Robot Learning*, 2021. 1, 2, 8

[60] Nan Yang, Lukas von Stumberg, Rui Wang, and Daniel Cremers. D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry. In *CVPR*, 2020. 2

[61] Huizhong Zhou, Benjamin Ummenhofer, and Thomas Brox. Deeptam: Deep tracking and mapping. In *ECCV*, 2018. 2

[62] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, 2017. 1, 2