SLIM: Self-Supervised LiDAR Scene Flow and Motion Segmentation Supplementary Material

1. Kabsch Algorithm

The goal of the Kabsch algorithm is to provide a rotation as the solution of bringing two lists of points as close together as possible. In this paper, we only need a specialized form

$$T_r = \underset{T \in \mathbb{R}^{4\times4}}{\operatorname{argmin}} \sum_{\mathbf{p}_i \in \mathcal{P}_t} w_i \left\| (T - I_4)\mathbf{p}_i - \mathbf{f}_i \right\|^2 \tag{1}$$

to solve with

$$T_r = \begin{pmatrix} R_r & \mathbf{t}_r \\ \mathbf{0} & 1 \end{pmatrix} \quad . \tag{2}$$

The solution for the translation offset is simply the mean flow

$$\mathbf{t}_r = \frac{1}{W} \sum_i w_i \mathbf{f}_i \tag{3}$$

with $W = \sum_i w_i$ being the sum of the weights. Substituting this back into the original optimization we get the normal form for the Kabsch algorithm

$$R_r = \underset{R \in \mathbb{R}^{3\times3}}{\operatorname{argmin}} \sum_{\mathbf{p}_i \in \mathcal{P}_t} w_i \left\| R\mathbf{p}_i - (\mathbf{p}_i + \mathbf{f}_i - \mathbf{t}_r) \right\|^2 \quad . \tag{4}$$

The Kabsch algorithm defines the solution via the cross-covariance matrix $H \in \mathbb{R}^{3\times3}$

$$H^{(ij)} = \frac{1}{W} \sum_k w_k p_k^{(i)} \left(p_k^{(j)} + f_k^{(j)} - t_r^{(j)} \right) \tag{5}$$

where we use $\mathbf{x}^T = (x^{(1)}, x^{(2)}, x^{(3)})$ as component notation. The solution is then given as

$$R_r = V \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{pmatrix} U^T \tag{6}$$

using the singular value decomposition (SVD) $H = U\Sigma V^T$ and the correction value $d = \operatorname{sign}(\det(VU^T))$ for ensuring a proper rotation.

As most deep learning frameworks provide SVDs as differentiable operators, and with H being differentiable w.r.t. the flow and the weights, the resulting transform T_r is also differentiable w.r.t. those components.

As the above equations make clear, the global scaling of the weights does not matter for the result, agreeing with our intuition. In fact, the division by the sum of weights in (5) is not even necessary, as the SVD stores any global scale in the unused diagonal matrix Σ.

2. Computation of Confidence Weights

Using the confidence logits we need to compute a set of weights for the above Kabsch algorithm. In our initial experiments, we first tried a softmax activation, however, we found it to be quite unstable and generalizes badly to unseen inputs. We attribute this behavior to the fact that a few outliers in the logit distribution contain almost all the weight after activation. Activating each logit independently with the sigmoid function alleviates the problem as all high logit values get mapped to roughly the same weight of 1. We now have to deal only with a numerical issue when all logits are quite low, e.g. below -100 for 32-bit floating-point-precision. Then the sigmoid function maps all values to exactly 0, leaving the Kabsch algorithm with nothing to optimize. We avoid this problem by normalizing the sum beforehand which theoretically does not change the result, however, we provide a fused computation for the sigmoid activation and normalization such that we have a numerically stable set of weights.

Defining $m_i = [\sigma(l_{cls,i}) \geq p_{stat}]$ as the mask for static points our goal is to compute

$$w_i = \frac{m_i \sigma(l_{wgt,i})}{\sum_{j|m_j=1} \sigma(l_{wgt,j})} \tag{7}$$

with

$$\sigma(x) = \frac{1}{1+e^{-x}} = e^x \frac{1}{1+e^x} = e^x \sigma(-x) \quad . \tag{8}$$

From the identity above it is easy to check that

$$\sigma(x) = e^{\min(x,0)} \sigma(|x|) = \begin{cases} e^0 \sigma(x) & \text{for } x \geq 0 \\ e^x \sigma(-x) & \text{for } x < 0 \end{cases} \tag{9}$$

holds. Using this in (7) with $l^-_{wgt,i} = \min(l_{wgt,i}, 0)$ we have

$$w_i = \frac{m_i \sigma(|l_{wgt,i}|) \exp(l^-_{wgt,i})}{\sum_{j|m_j=1} \sigma(|l_{wgt,j}|) \exp(l^-_{wgt,j})} \quad . \tag{10}$$

In the above equation, the sigmoid activations are now numerically stable between 0.5 and 1. The well-known trick of shifting all exponents about $s = \max_{i|m_i=1} l^-_{wgt,i}$ in the above expression makes the whole expression numerically stable

$$w_i = \frac{m_i \sigma(|l_{wgt,i}|) \exp(l^-_{wgt,i} - s)}{\sum_{j|m_j=1} \sigma(|l_{wgt,j}|) \exp(l^-_{wgt,j} - s)} \tag{11}$$

as the largest term of the sum in the denominator is now at least as large as $\sigma(|s|)e^0 \geq 0.5$ and therefore the denominator is well above 0.

3. Moving Classification Threshold

The result of any classification task depends strongly on the chosen threshold based on the predicted scores. Fig. 1 shows how differently trained networks generate different distributions for the classification scores and how some metrics vary when changing the threshold p_{thresh} for that score $\sigma(l_{cls,i})$. We are interested in optimizing the scene flow for the stationary and the moving points, therefore, we want to select the optimal threshold for the AEE 50-50 metric. As the classification is already used during training as a mask to improve the results of the resulting rigid transform T_r we need an online mechanism selecting this threshold. As the scores are bounded between 0 and 1 we do this by keeping a moving average of the AEE 50-50 metric m_k for $R = 100,000$ different thresholds p_k, resulting in different possible aggregated flows $\mathbf{f}^{(k)}_{agg,i}$.

For the supervised training, the update rule is

$$m'_k = m_k \alpha^{N_t} + (1-\alpha^{N_t}) \frac{1}{\sum_i c_i} \sum_{i=1}^{N_t} c_i |\mathbf{f}_{gt,i} - \mathbf{f}^{(k)}_{agg,i}| \tag{12}$$

with N_t being the number of points in the currently processed point cloud, α being the decay factor of the exponential moving average per point, and c_i being a pointwise weighting factor which is 1 for stationary points and $\frac{N_{mov}}{N_{stat}}$ for moving points. The last ratio is measured across the whole training set and the weights, therefore, account for the fact that we are interested in an optimal AEE 50-50 compared to optimizing AEE. The update rule also accounts for the current size of the point cloud and gives more weight to large point clouds as it would happen if we would update for every point individually. As we initialize the moving metrics with 0, we compute an unbiased estimate through also tracking a bias counter $b'_k = b_k \alpha^{N_t} + (1 - \alpha^{N_t})$ and simply dividing the above computed metric by it: $\hat{m}_k = m_k / b_k$.

For the exponential moving average to "forget" after every 5,000 training iterations, we set α to $1 - \frac{1}{5000\langle N_t \rangle}$, where $\langle N_t \rangle$ is the average point cloud size of the used dataset. Based on the unbiased estimates $\hat{m}_k$ we simply look for the best, therefore, smallest value and choose the corresponding threshold p_k as the optimal threshold $p_{thresh} = \text{argmin}_{p_k} \hat{m}_k$. The whole update step can be easily vectorized and implemented efficiently using standard deep learning framework components including cumulative sums and sorting of the classification scores and is implemented similar to computing efficiently precision-recall-curves.

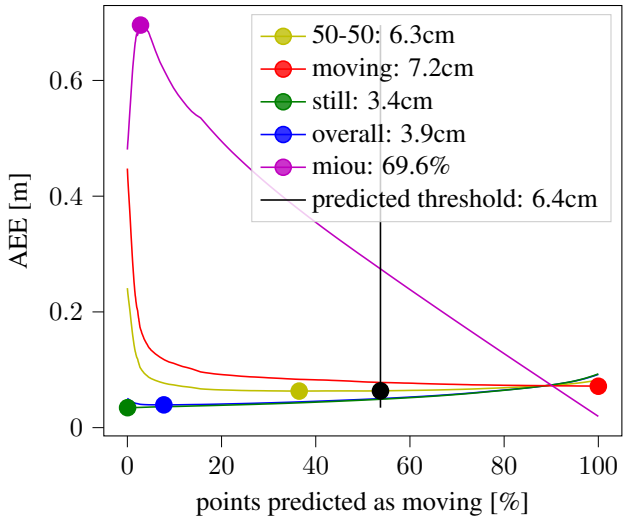

Figure 1: Different metrics across changing thresholds for supervised training on nuScenes: The vertical line represents the moving threshold computed online during training. The yellow curve represents AEE 50-50 on the test set. Selecting all points as moving gives the best result for the moving AEE but the worst for stationary AEE. The opposite is true when classifying all points as stationary and using the improved rigid motion result. However, the moving AEE increases much more dramatically if points are classified wrongly as static and the optimal threshold is therefore around 40% moving points. Our moving average predicts just about over 50% moving, however, the AEE 50-50 is very flat in this area, and our achieved AEE 50-50 of 6.4cm is just 1mm lower than the optimal threshold.

In the case of self-supervised trainings, we do not use the weights c_i as they already make use of labeled quantities, namely if a point is moving or stationary. We also can not use the ground truth flow and therefore replace the endpoint error term through the k-NN error term $e_i(\mathbf{f}^{(k)}_{aggr,i})$ defined in the paper in (4):

$$m'_k = m_k \alpha^{N_t} + (1 - \alpha^{N_t}) \sum_{i=1}^{N_t} |e_i(\mathbf{f}^{(k)}_{aggr,i})| \tag{13}$$

Losing the controlled weighting and accurate error estimation negatively influences moving averages and the accuracy, however, we found it to work reasonably well.

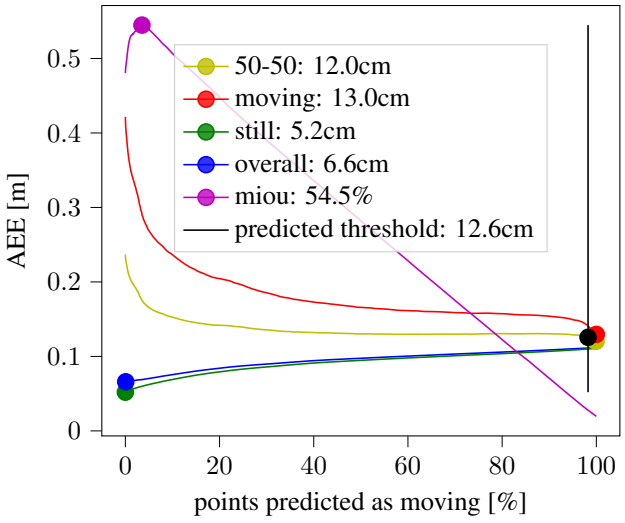

Figure 2: Different metrics across changing thresholds for self-supervised training on nuScenes: The vertical line represents the moving threshold computed online during training using the kNN residuals. The fact that stable self-supervised training criteria are more challenging to formulate for Scene Flow, leads to a larger gap between the optimal and learned threshold for self-supervised training.

Fig. 2 shows the same information as Fig. 1 only for a self-supervised training corresponding to the third row of Table 7 of the paper. Where in the supervised case the AEE 50-50 score from our moving threshold computation is just 1mm short compared to the optimal AEE 50-50, it is in the self-supervised case already more than 0.6cm worse than the optimal threshold could have been.

4. Robust Outlier Metric

The outlier metric used by previous works [4, 1, 6, 7] defines a point as an outlier if its predicted flow vector has an endpoint error >30cm or a relative error >10%. However, we believe this metric has several drawbacks:

- The name and intuition about this metric are misleading. A predicted flow of 1.2cm length for a ground truth flow of only 1.0cm would be already considered an outlier where most people would classify this as normal noise.
- It also contradicts the accuracy metrics. The case mentioned above would not only be classified as an outlier but also as strict accurate (<5cm) and relaxed accurate (<10cm).
- The definition itself is not minimal. The case where the endpoint error is greater than 30cm but smaller than 10% basically does not exist, as this would require a ground truth flow larger than 3m, or accounting for 10Hz sample frequency 30m/s=108km/h. This rarely occurs in the city-based datasets and therefore the 30cm condition is superfluous.

Table 1: Self-supervised training & evaluation on nuScenes - comparison of metrics ROutl and Outl

	Moving		Static	
	ROutl↓	Outl↓	ROutl↓	Outl↓
Zero	0.5783	1.0000	0.6838	1.0000
PointPWCNet (PPWC)	0.3848	0.9395	0.1800	0.8508
PoseFlowNet (PF)	0.9364	1.0000	**0.0035**	1.0000
Ours	**0.1309**	**0.8125**	0.0630	**0.5286**

To avoid these drawbacks, we introduce a robust outlier metric ROutl which we report instead for the two newly investigated datasets, nuScenes and CARLA. It is defined as the relative portion of points that have a predicted endpoint error >30cm and relative error >30%. We believe it to not suffer from any of the drawbacks mentioned above:

- Requiring both the relative and absolute conditions to hold removes counting the small noise on an almost 0 ground truth flow to not be counted as an outlier. Also requiring a 30% error instead of 10% separates it well from the relaxed accuracy, following more closely the intuition of the name outlier.
- A flow classified as a robust outlier can now never be at the same time (relaxed) accurate as those definitions are mutually exclusive.
- The definition is now minimal. A prediction for a small flow ground truth flow of up to 1m with 30% < 30cm needs to meet the absolute condition to be counted as an outlier, where a prediction of a large flow ground truth of above 1m or 36km/h needs to meet the relative condition.

Also, the robust outliers are counted as outliers by the old metric, so the robust outlier metric always reports a lower number. Table 1 shows a comparison of the two metrics and how they behave in relation to each other. Especially for moving points, the Outl metric is very close to 100% for all methods and thus does not convey a lot of information. ROut discriminates better between the methods.

5. Classification

We evaluate the classification using the following metrics (where TP: true positive, TN: true negative, FP: false positive, FN: false negative):

- accuracy: $\text{acc} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$

Table 2: Classification results for self-supervised and supervised training for different training datasets, evaluated on KITTI-SF. Acc, mIoU & sens are the resulting values that are obtained when the threshold is optimized w.r.t. AEE 50-50 (in the self-supervised case via kNN residuals as proxy). Opt acc & opt mIoU are scores that could achieved if the threshold were selected w.r.t. optimal classification, irregardless of Scene Flow estimation performance, a tradeoff that is also discussed in Fig. 1

	Train Dataset	acc	mIoU	sens	opt acc	opt mIoU
Self-	KITTI-RL	**0.601**	**0.429**	0.835	0.747	0.581
Super-	nuScenes	0.491	0.308	0.906	**0.835**	**0.699**
vised	CARLA	0.443	0.246	**0.991**	0.765	0.613
Super-	nuScenes	**0.861**	**0.754**	**0.942**	**0.925**	**0.855**
vised	CARLA	0.610	0.416	0.418	0.650	0.434

- sensitivity: sens $= \frac{\text{TP}}{\text{TP+FN}}$
- mean intersection over union: mIoU $= 0.5(\frac{\text{TP}}{\text{TP+FP+FN}} + \frac{\text{TN}}{\text{TN+FP+FN}})$

On KITTI Stereo Flow (KITTI-SF), a mIoU of 34% can be achieved by guessing that a point is moving with a probability of $p_{\text{mov,guess}} = 0.31$. This can be shown using the knowledge that roughly 40% of points in the dataset are moving ($p_{\text{gt}} = 0.4$), and from the optimality criteria for the binary classification: $\frac{\partial \text{mIoU}}{\partial p_{\text{mov,guess}}} = 0$ and $\frac{\partial (\text{IoU}_{\text{mov}} + \text{IoU}_{\text{still}})}{\partial p_{\text{mov,guess}}} = 0$. As already briefly mentioned in the paper, Table 2 shows that our method achieves a mIoU score of 42.9% on KITTI-SF when trained self-supervised on KITTI-RL. The results show, that like for Scene Flow estimation, the domain gap from KITTI-RL to KITTI-SF seems to be the smallest and leads to the best performance achieved through self-supervision. The domain gap from simulated CARLA data to KITTI-SF is obviously larger than from nuScenes to KITTI-SF, which can explain the performance gap between the two datasets when evaluating on KITTI-SF for both the self-supervised and the supervised case. A few example images for motion segmentation by the network trained using self-supervision can be seen in Fig. 3.

Once again, when comparing optimal scores and those optimized for AEE 50-50 in Table 2, it becomes clear that there is a tradeoff between optimal performance on the Scene Flow task (AEE 50-50) and optimal performance in terms of classification, as discussed in Section 3.

6. Architecture & Parameter Details

6.1. Training

We use the RMSProp optimizer with 0.0001 for the learning rate, with warm-up and decay. For the first 2k steps of the training, the learning rate is exponentially increased, starting from 0.0 to 0.0001. After that, exponential decay is applied with a ratio of 0.5 every 60k steps. While we chose the batch size of 1 during training, we use a batch of point clouds to perform "forward" and "backward" predictions.

6.2. PillarFeatureNet

The Pillar Feature Net (PFN) is used to create the Bird's-Eye-View (BEV) feature maps consumed $\mathcal{I}_t, \mathcal{I}_{t+1}$ by the RAFT backbone. We use the default configuration of the PFN introduced in [3] almost every aspect, with minor deviations described in the following: In their original paper, Lang et al. used batch normalization, but due to the small batch size in our experiment, we swap regular batch normalization for batch renormalization [2]. This is then followed by ReLU activation, like in the original paper. In our experiments, the spatial BEV dimensions are $H = 640, W = 640$ and $C = 64$ feature channels are used: $\mathcal{I}_t, \mathcal{I}_{t+1} \in \mathbb{R}^{H \times W \times C}$, but any reasonable combination of spatial and feature dimensions can work. We do not use the reflectivity/intensity channel of the point clouds, as the usage would break generalization when training on KITTI-RL and evaluating on KITTI-SF, where the network has to generalize from LiDAR point cloud data to stereo camera point cloud data.

6.3. RAFT

We use the parameters and configuration of the model called RAFT-S introduced in [5] whenever possible, for example, like in the original paper, the output of the context network (see [5]) has 160 channels and is divided into the initial hidden state h_0 with 96 channels and the encoded scene context with 64 channels. Like in the original RAFT-S as introduced by Teed et al. [5], we use 4 correlation levels with a correlation lookup radius of 3 each. Also, like in the original paper, the ConvGRU in the Update Block estimates flow at 1/8th of the original input resolution: $\mathcal{F} \in \mathbb{R}^{H/8 \times W/8 \times 2}$. As an extension to the original RAFT-S, we introduce logits $\mathcal{L}_{cls}$ and confidence weights $\mathcal{L}_{wgt}$, which are handled equivalently to the flow map, in the case of the logits with an additional encoder & decoder. The encoders in the Update Block use 2D convolutional layers with 64 and 32 channels each with ReLU activation, as depicted in Fig. 4. The Flow-Head (like in the RAFT-S paper version) and additional Classification-Head each use two convolutions, 128 filters each with ReLU activation Unlike in the original paper, no mask for upsampling the predictions to full resolution is learned, instead bilinear upsampling is used. For training and inference, six iterations are used in the update block to refine the result.

7. Transferability of the SLIM Loss Framework onto PointPWC

Our ablation study shows an improvement of our RAFT network over the other baseline methods, without applying

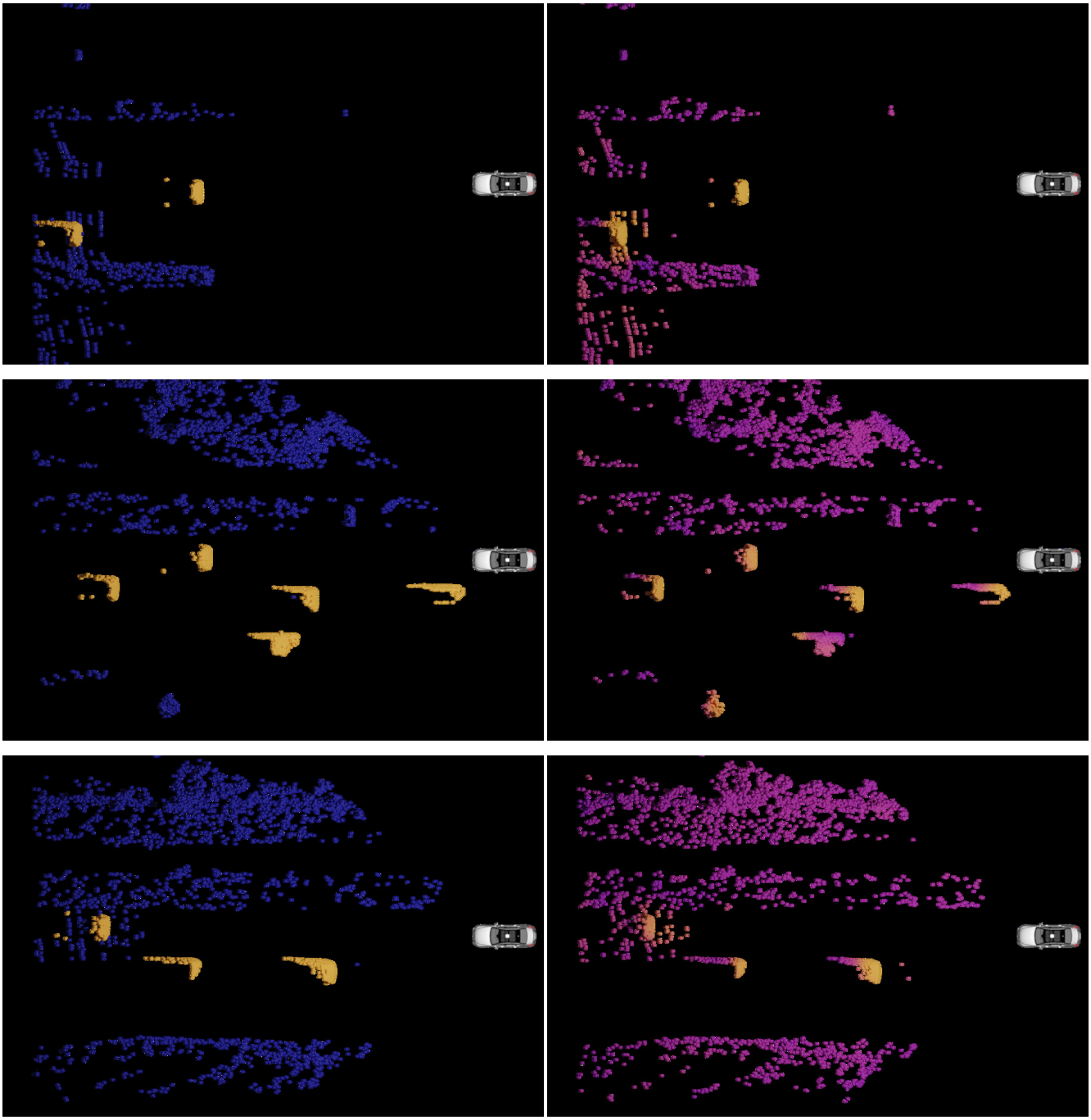

Figure 3: Motion segmentation by our self-supervised network trained on KITTI-RL evaluated on KITTI-SF. Left: Ground truth motion segmentation, Right: Predicted motion segmentation by our network, brighter colors indicate higher dynamicness.

our novel loss framework (Table 7, row 1). In addition, it demonstrates that motion segmentation and Kabsch regularization for rigid static flow boost SLIM's performance when trained self-supervised (row 2+3).

To investigate the relationship between the network architecture and the performance boost provided by our loss framework we present results of our SLIM loss applied to another model.

PoseFlowNet [6] already has a mechanism to deal with the rigid scene motion so our loss cannot be directly applied

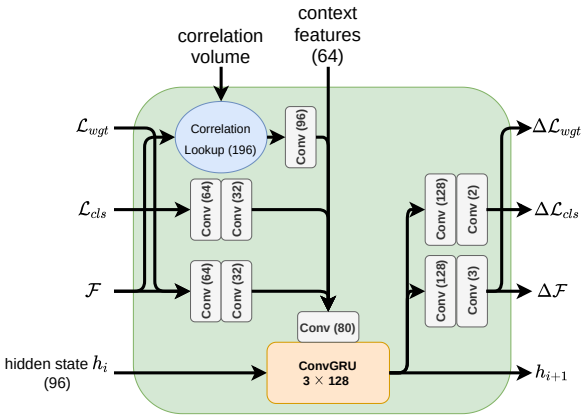

Figure 4: Update Block annotated with the number of channels for tensors and filters for convolutions.

onto that architecture. However, PointPWC has a suitable architecture to test the transferability of our proposed losses. We extend the PPWC architecture with the necessary logits to apply our loss framework (classification & confidence), as well as the necessary Kabsch regularization (from here on out referred to as PPWC-MOD), and apply the SLIM loss framework to each of the four pyramid levels. As the application of SLIM's losses requires cyclic consistency as a self-supervised training signal, we additionally extend the training regime of PointPWC to do forward and backward scene flow estimation by adding an inference step for the time-reversed input point clouds.

Table 3 shows that the above-introduced modifications result in a worse performance than the baseline PPWC (row 1,4). Further investigations into this matter reveal that the cyclic training routine does not harmonize well with PPWC. Therefore, we present the results of two more training configurations: The PPWC trained on cyclic samples (without any other modification) (row 2) and the PPWC-MOD without cyclic sampling and training (row 3). These experiments show that SLIM produces slightly better results compared to the corresponding baselines for some metrics, but not for all. In the case of non-cyclic training, it is also shown, that there are only marginal differences. We conclude that SLIM's performance boost also heavily depends on the chosen network architecture.

8. Runtimes

The runtimes of SLIM and the baselines are reported in Table 4. Please note that SLIM does not make use of the optimizations reported in [5].

9. Failure Cases

Our network predicts quite accurate static flow and thus has no problem distinguishing cars as moving even when having the same speed as the sensor as long as there are some static objects present. Most scenes in the NuScenes dataset have more than 90% of static points. Even for scenes with the lowest amount of static points of around 20% the flow estimates for the dominant dynamic objects is correct and only a few static elements fail, see Figure 5.

Table 3: Ablation study on nuScenes for PPWC[7]. Flow directions during training: f: forward, r: reverse. f+r: training using both forward and reverse time. f&r: training using both forward and reverse time with explicit cyclic consistency between forward and reverse predictions.
*: SLIM loss without cyclic consistency

network arch.	loss framework	flow dirs training	Mov. AEE	Stat. AEE	50-50 AEE
PPWC	orig	f	**0.3539**	0.1974	0.2756
PPWC	orig	f+r	0.6432	0.5457	0.5945
PPWC-MOD	SLIM*	f	0.3837	**0.1475**	**0.2656**
PPWC-MOD	SLIM	f&r	0.4005	0.2284	0.3145

Table 4: Runtimes as reported by the authors of each method. The runtime for RAFT-S has been measured using an implementation optimized for half-precision, SLIM (TensorFlow, 6 RAFT stages, unoptimized) takes 443 ms on a single Tesla V100 GPU.

Method	Runtime
PPWC [7]	117 ms
PF [6]	491 ms
RAFT-S [5]	50 ms
SLIM	443 ms

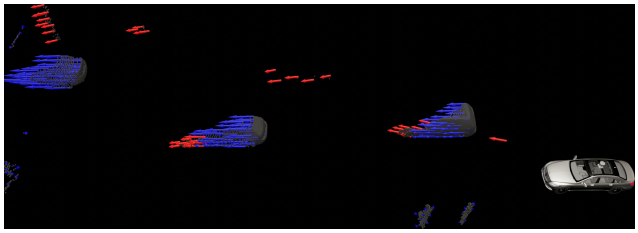

Figure 5: Very dynamic scene leads to flow outliers on the static pole in the background (top left). (red: outlier, blue: accurate)

References

[1] Xiuye Gu, Yijie Wang, Chongruo Wu, Yong Jae Lee, and Panqu Wang. Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*,

pages 3254–3263. Computer Vision Foundation / IEEE, 2019. 3

[2] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1945–1953, 2017. 4

[3] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 12697–12705. Computer Vision Foundation / IEEE, 2019. 4

[4] Xingyu Liu, Charles R. Qi, and Leonidas J. Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 529–537. Computer Vision Foundation / IEEE, 2019. 3

[5] Zachary Teed and Jia Deng. RAFT: recurrent all-pairs field transforms for optical flow. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part II*, volume 12347 of *Lecture Notes in Computer Science*, pages 402–419. Springer, 2020. 4, 6

[6] Ivan Tishchenko, Sandro Lombardi, Martin R. Oswald, and Marc Pollefeys. Self-supervised learning of non-rigid residual flow and ego-motion. In Vitomir Struc and Francisco Gómez Fernández, editors, *8th International Conference on 3D Vision, 3DV 2020, Virtual Event, Japan, November 25-28, 2020*, pages 150–159. IEEE, 2020. 3, 5, 6

[7] Wenxuan Wu, Zhiyuan Wang, Zhuwen Li, Wei Liu, and Fuxin Li. Pointpwc-net: Cost volume on point clouds for (self-)supervised scene flow estimation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part V*, volume 12350 of *Lecture Notes in Computer Science*, pages 88–107. Springer, 2020. 3, 6