SLIM: Self-Supervised LiDAR Scene Flow and Motion Segmentation

Stefan Andreas Baur[*1,2], David Josef Emmerichs[*1,5], Frank Moosmann[1], Peter Pinggera[1],
Björn Ommer[4,5] and Andreas Geiger[2,3]
[1]Mercedes-Benz AG, Stuttgart [2]MPI-IS, Tübingen [3]University of Tübingen
[4]Ludwig Maximilian University of Munich [5]Heidelberg University
firstname[_middlename].lastname@daimler.com
ommer@uni-heidelberg.de andreas.geiger@tue.mpg.de

Abstract

Recently, several frameworks for self-supervised learning of 3D scene flow on point clouds have emerged. Scene flow inherently separates every scene into multiple moving agents and a large class of points following a single rigid sensor motion. However, existing methods do not leverage this property of the data in their self-supervised training routines which could improve and stabilize flow predictions. Based on the discrepancy between a robust rigid ego-motion estimate and a raw flow prediction, we generate a self-supervised motion segmentation signal. The predicted motion segmentation, in turn, is used by our algorithm to attend to stationary points for aggregation of motion information in static parts of the scene. We learn our model end-to-end by backpropagating gradients through Kabsch's algorithm and demonstrate that this leads to accurate ego-motion which in turn improves the scene flow estimate. Using our method, we show state-of-the-art results across multiple scene flow metrics for different real-world datasets, showcasing the robustness and generalizability of this approach. We further analyze the performance gain when performing joint motion segmentation and scene flow in an ablation study. We also present a novel network architecture for 3D LiDAR scene flow which is capable of handling an order of magnitude more points during training than previously possible.

1. Introduction

Reasoning about the motion of dynamic objects in a scene is crucial for robotic applications such as autonomous driving. One representation for such motion is scene flow, a set of displacement vectors between two consecutive time frames, forming a 3D vector field $\mathcal{F}_{t\rightarrow t+1}$. Its application allows to transform points $\mathcal{P}_t$ recorded in the first frame into the second frame via $\mathbf{p}_i^{(t+1)} = \mathbf{p}_i^{(t)} + \mathbf{f}_i$. Another representation is the binary classification of points into either *moving* or *stationary*, known as motion segmentation.

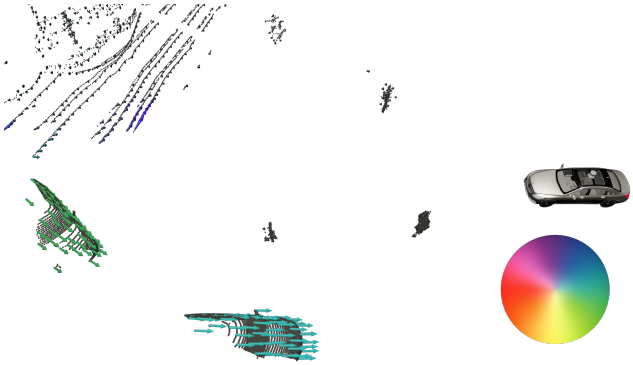

Figure 1: **Scene Flow Prediction by SLIM.** Flow vectors are colored by their direction and magnitude as illustrated with the color wheel. Our method correctly annotates the background as static and produces consistent results for the two vehicles, matching the ground truth scene flow.

This paper tackles the problem of simultaneous LiDAR scene flow estimation and motion segmentation using a deep network. Existing works treat these two problems independently. Most of them rely on large amounts of labeled data to obtain state-of-the-art results (see Section 2.2). Self-supervised learning attempts to alleviate this problem, but existing scene flow networks using weak or self-supervision have critical drawbacks: Currently, all of them require downsampling of point clouds from around 128,000 points for typical LiDAR frames to no more than 8,192 points during training or they train and evaluate on datasets with 1-to-1 point correspondences, see Section 2.4.

Our approach trains well even on datasets with no 1-to-1 correspondences and can handle one to two orders of magnitude more points per point cloud than previous approaches for self-supervised scene flow. We build on top of RAFT [39], the state-of-the-art for optical flow, and extend

[*] Joint first authors with equal contribution.

the network to perform iterative scene flow estimation and motion segmentation on PointPillar-based Bird's-Eye-View (BEV) feature representations of point clouds [20]. This architecture realizes an excellent trade-off between generalizability, accuracy, and computational efforts. Using the consistency of motion for the stationary background, we are able to extract training targets for self-supervised motion segmentation. No additional data or labels besides raw, consecutively recorded point clouds are required for training, the cues for self-supervised classification come from the typical motion profiles of these classes alone.

Our main **contributions** are:

- Our method is the first point-cloud-based scene flow estimation method that can classify points as "moving" and "stationary" based on self-supervised training.
- Our method significantly outperforms previous methods in point-cloud-based scene flow estimation, especially in terms of generalization to previously unseen data, which we demonstrate on multiple datasets, in both the self-supervised and fully supervised setting.
- Our novel network architecture can handle significantly more points than current weakly or self-supervised approaches.

2. Related Work

The term *scene flow* was first introduced by Vedula et al. [47]. Traditionally, it is estimated using stereo cameras [14, 25, 44, 16, 29, 6, 30, 51, 25], and has successfully been applied to other domains such as RGB-D sensors [36].

2.1. Motion Segmentation

While there exists a group of methods layering optical flow into consistent groups, they cannot be easily transferred to point clouds as they make heavy use of color information and brightness constancy assumptions [38, 54, 37]. Research for scene flow and motion segmentation on range images started with hand-crafted features and matching, see [28]. In this work, however, we focus on scene flow estimation in the point cloud domain, which provides its own challenges such as sparse data as well as the absence of well-established representations. Within recent years, the focus shifted towards learning-based methods, as discussed next.

2.2. Supervised Point-Cloud-Based Scene Flow

For many years, research in the area of point-cloud-based scene flow estimation focused on supervised learning methods. Most approaches relied on synthetic datasets [12, 21, 22, 50, 52, 33, 45] or non-public datasets [46] for training. Behl et al. [2] started to use the real-world Kitti Tracking dataset, but it is only densely annotated in the camera field of view (FoV). Gojcic et al.[11] lessened the requirement on annotated data by using weak supervision through odometry and foreground-background segmentation to train their network for simultaneous motion segmentation and scene flow estimation, enabling them to use the SemanticKitti [3] dataset to train their approach.

2.3. Self-Supervised Learning

The dependence on costly annotated data can be overcome by self-supervised learning. It has been successfully applied to train networks on images to predict optical flow, stereo flow, depth (or a combination thereof) [48, 1, 15, 56, 58, 19, 7, 10, 34, 49, 23], and motion segmentation [43, 4] using geometric and temporal/cyclic consistency losses. Self-supervised learning on point clouds, however, is a young field where in most cases self-supervision has also been used to pre-train a network on a proxy-task and to reuse or finetune the weights for the actual task using supervision [13, 35, 55, 32, 40, 57].

2.4. Self-Supervised Scene Flow on Point Clouds

Recently, there has been some research into self-supervised scene flow estimation on point clouds, but all of these works have various shortcomings.

Point Cloud Size: Many architectures are based on graphs [42, 31], PointNet/FlowNet3D [27] or layouts with coarse global all-to-all correlations combined with finer local correlations [18]. In order to cope with exploding memory requirements during training, they rely on random downsampling of their input point clouds, in many cases also during inference. Mittal et al. [27] subsample to 2,048 points, others [53, 18, 31, 42] downsample to 8,192 points. These methods perform well on point cloud data gathered from stereo cameras or RGB-D data sources with little sparsity, such as KITTI Stereo Flow (KITTI-SF) [26] or FlyingThings3D (FT3D) [24]. In comparison, LiDAR sensors cover much larger areas than stereo rigs but at a much lower point density. Regular LiDAR sensors measure upwards of 100k 3D points in each frame. We are able to process much more points than previous works. This gives our architecture an advantage over existing methods, as it can leverage arbitrary point densities during both training and inference.

Correspondence Assumption: Another serious drawback in current methods was discovered by Zuanazzi et al. [59]: They note that [53] and [27] train (and evaluate) on point clouds with a 1-to-1 correspondence $\mathbf{p}_i^{(t+1)} = \mathbf{p}_i^{(t)} + \mathbf{f}_i$, the KITTI-SF and FT3D dataset. Furthermore, they show in experiments heavily degrading performance when points are shuffled and sampled in an effort to destroy the correspondences prior to training/inference. Real-world measurements from LiDARs do not abide by the correspondence assumption, as the scene is sampled for each timestep, and a bijective mapping between point clouds does not exist.

Robustness to Outliers: Since flow vectors on static points

all derive from the same sensor motion, Tishchenko et al. [42] introduced a two-stage-network: The first stage, a pose regression network, estimates ego-motion and transforms the source point cloud. The second stage subsequently estimates flow. Although realistic data recordings contain much more static than moving points, which should lead to good performance, they suffer from high outlier rates on real data - even on static points. We demonstrate in our experiments a much better outlier resistance predicting dense scene flow for all points by regressing the sensor's ego-motion using the fully differentiable Kabsch algorithm [17].

2.5. Self-Supervised Motion Seg. on Point Clouds

To the best of our knowledge, there exists only one previous network that learns motion segmentation on point clouds using self-supervision: Thomas et al. [41] use multiple recordings of the same environment to train a network to identify permanently static, slow, and fast-moving objects. Our method does not require multiple recording passes and can train on pairs of point clouds directly.

3. Method

3D scene flow estimation uses two consecutive input point clouds $\mathcal{P}_t \in \mathbb{R}^{N\times 3}$, $\mathcal{P}_{t+1} \in \mathbb{R}^{M\times 3}$ to predict for each point in the first point cloud a 3D displacement vector, representing the motion of each point w.r.t. the sensor frame.

3.1. Network Architecture

We make use of an efficient network, specialized on LiDAR data from the perspective of an autonomous vehicle (AV), sacrificing the ability to generalize to more freely moving scenes and objects as seen in FT3D for example. Our network consists of three components, detailed in Fig. 2: A point cloud encoder, a flow backbone, and an output decoder explained below. We also introduce the used loss functions and their purposes.

Point Cloud Encoder: The input point clouds $\mathcal{P}_t, \mathcal{P}_{t+1}$ are separately encoded into a BEV pseudo-image using the Pillar Feature Net (PFN) introduced in [20] (shared weights), the resulting values $\mathcal{I}_t, \mathcal{I}_{t+1} \in \mathbb{R}^{H\times W\times C}$ are then processed by the backbone. To ensure comparability with other methods [53, 42, 21, 52, 12, 11], we use for all datasets the same BEV extent covering the square from $-35m \leq x, y \leq 35m$ around the ego vehicle, with x, y being the horizontal axes. We use a resolution of $H = W = 640$ which corresponds to a pillar size of roughly 11cm.

Flow Backbone: Our backbone is heavily inspired by RAFT [39] which was developed to predict dense optical flow on images. Its core component is an update block acting recursively on a hidden state and a flow prediction, producing a more refined and accurate flow with every iteration. Therefor, a correlation volume is constructed from independently encoded input images, using the previous flow prediction to look up correlation values and thus guide the flow to a more accurately matching pixel region. Even though RAFT was designed for dense optical flow, we show that it is also very suitable and generalizes well in the sparsely populated BEV domain, which consists of more scattered and smaller regions and very independent motion patterns (moving traffic participants) than regular images.

We adapt RAFT [39] to handle flow prediction and iteratively update two additional logits, as shown in Fig. 2. The first logit map $\mathcal{L}_{cls}$ is used as an output signal to classify the points as stationary or moving w.r.t. the world frame. The accuracy of predicted flow can vary greatly in a scene, as featureless surfaces are not suitable for flow estimation. The second logit $\mathcal{L}_{wgt}$ is used to overcome this problem, allowing the network to signal its confidence in a flow estimate. Both logits are used by the output decoder to aggregate and improve accuracy on static and dynamic scene elements.

$\mathcal{L}_{cls}$ is handled similarly to the flow stream, but the task of confidence weighting is more closely tied to the flow prediction and thus the data streams are therefore coupled during the processing of information. Except for this minor change we retain the general framework of RAFT, including zeroing the gradient not only on the input flow but also on the input logits of each update block.

Output Decoder: Our backbone produces a 2D BEV flow map $\mathcal{F} \in \mathbb{R}^{H\times W\times 2}$ together with two logit maps, $\mathcal{L}_{cls}, \mathcal{L}_{wgt}$. Additionally, our model keeps track of a global classification threshold p_{stat} through moving averages, discussed in more detail in Section 3.2.

Firstly, the output decoder uses these BEV maps to annotate each point of the input point cloud $\mathcal{P}_t$ with a flow vector $\mathbf{f}_i$ and two logits $l_{cls,i}, l_{wgt,i}$ according to the values of its corresponding pillar cell. By doing this we assume that all points in a pillar behave very similar. We believe this to be true for almost all LiDAR point clouds measured outdoors, as all moving traffic participants need to occupy some ground. Additionally, most LiDAR systems are built in a way that their beams are not upward-facing. Note that although our network architecture is specialized in this way, our loss framework is suited for any 3D scene flow prediction and does not require the assumption of a 2D flow. In order to regularize and improve the flow prediction on the static scene, the output decoder aggregates the points classified as still into a single coherent rigid-motion odometry transform $T_r \in \mathbb{R}^{4\times 4}$. This aggregated transform should be minimizing the squared error when comparing its resulting flow with the flow prediction from the network:

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

Defining the odometry transform $O_{t\to t+1} = E_{E^t}^{(t+1)} \in$

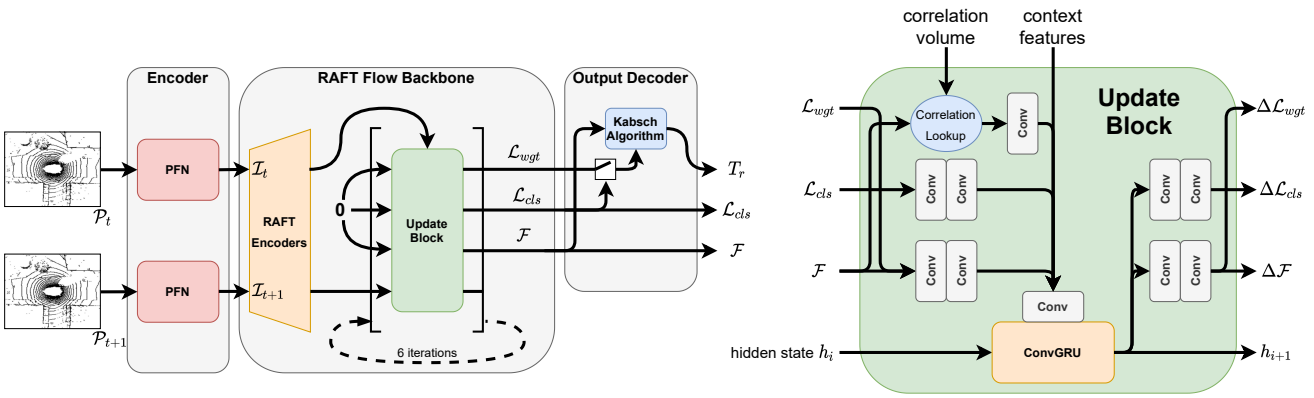

Figure 2: Overview of our network architecture. The Convolutional Gated Recurrent Unit (ConvGRU) iteratively predicts updates for the flow and logits, using correlation lookups guided by the predicted flow.

$\mathbb{R}^{4\times4}$ as the ego vehicle position at time $t+1$ measured in the ego vehicle frame from time step t, we know that all stationary points should have a flow of $(O_{t\to t+1}^{-1} - I_4)\mathbf{p}_i$. We therefore expect the computed transform $T_r^{-1} \approx O_{t\to t+1}$ to be a good approximation for the odometry of the vehicle.

We use Kabsch's algorithm [17] to compute T_r using singular value decomposition which is differentiable. The weights w_i for each point determine how much influence each flow vector prediction has on the final result T_r. We first apply a sigmoid activation to the confidence logits and then mask them based on the classification logits. Afterwards, we normalize all weights to sum to 1 to ensure numerical stability. See the supplementary material for details on the fused computation.

$$w_i = \frac{\sigma(l_{wgt,i})[\sigma(l_{cls,i}) \geq p_{stat}]}{\sum_j \sigma(l_{wgt,j})[\sigma(l_{cls,j}) \geq p_{stat}]} \tag{2}$$

The confidence logits only receive gradient updates through the computation of T_r and are therefore trained end-to-end without further supervision.

For the purpose of inference, the final aggregated flow map of our network is

$$\mathbf{f}_{agg,i} = \begin{cases} (T_r - I_4)\mathbf{p}_i & \text{if } l_{cls,i} \geq p_{stat} \\ \mathbf{f}_i & \text{if } l_{cls,i} < p_{stat} \end{cases} \tag{3}$$

or in words: Points predicted to be stationary use the flow from the aggregated rigid ego-motion transform, all other points use the raw flow prediction from the network. However, we make use of the aggregated transform T_r and the classification logits $\mathcal{L}_{cls}$ in the loss. Especially $\mathcal{L}_{cls}$ does not have any gradients yet through the aggregated flow or transform and is only trained through special loss terms introduced in the next section.

3.2. Loss

We have different self-supervised components based on the two input point clouds, and the network's predicted flow, logits, and the aggregated transform.

k-NN Loss: First and foremost we use a k-NN loss as a self-supervision signal for our flow estimation as is done by earlier works for self-supervised scene flow [27, 53, 42]. Given a pointwise flow $\mathbf{f}_i$ and a nearest neighbor function $NN_{\mathcal{P}_{t+1}}(\mathbf{p}_i) = \text{argmin}_{\mathbf{p}_j \in \mathcal{P}_{t+1}} |\mathbf{p}_j - \mathbf{p}_i|$ we find for every flow-corrected point in $\mathcal{P}_t$ an NN-based error distance:

$$e_i = |NN_{\mathcal{P}_{t+1}}(\mathbf{p}_i + \mathbf{f}_i) - (\mathbf{p}_i + \mathbf{f}_i)| \tag{4}$$

As we work with noisy data and especially non-bijective point clouds there are greater errors observed in distant points. To mitigate this, we define an outlier percentage $p_{out} << 1$ and ignore all errors lying in this top-percentile. Additionally, we apply this loss not to the aggregated flow $\mathcal{F}_{agg}$ but to the raw flow prediction $\mathcal{F}$ and the rigid flow $\mathbf{f}_{r,i} = (T_r - I_4)\mathbf{p}_i$ separately, resulting in a total NN loss

$$L_{nn} = \frac{1}{|\mathcal{P}_t|} \sum_i e_i(\mathbf{f}_i) + e_i(\mathbf{f}_{r,i}) \quad . \tag{5}$$

We found that this stabilizes the training in the early phases and does not hurt the performance. Applying the loss only to the aggregated flow has the problem that early during the training the smooth and small rigid flow is more accurate and thus dominates over the raw flow leading the network into a local optimum where the classification predicts only stationary points. This then leads to the raw flow getting masked out from the aggregated flow, consequently receiving no gradient updates, and thus performing badly. Of course, depending on training settings and network initialization, this behavior could also occur in reverse. Thus we train both flow fields every time at every point.

Rigid cycle consistency: Inspired by recent successful applications of cycle consistency losses [42, 27] we apply our network not only on the input point clouds $(\mathcal{P}_t, \mathcal{P}_{t+1})$ but also on the reversed order $(\mathcal{P}_{t+1}, \mathcal{P}_t)$, giving us access to the prediction of the inverse rigid motion T_r'. For accurate predictions, these two rigid transforms are the exact inverses to each other. In order to measure the error, we apply them to the input point cloud as a way of weighting the different

error aspects of the transforms.

$$L_{rcc} = \frac{1}{|\mathcal{P}_t|} \sum_{\mathbf{p}_i \in \mathcal{P}_t} |(T_r' T_r - I_4)\mathbf{p}_i| \tag{6}$$

Artificial Labels: The classification logits $\mathcal{L}_{cls}$ are trained using a self-supervised loss based on the NN errors e_i. Additionally, we compute these errors not only for the raw flow prediction $\mathbf{f}_i$ but also for the rigid counterpart $\mathbf{f}_{r,i}$ in order to compare those $e_{r,i}$ to the errors for the raw flow. We use the standard binary cross-entropy loss based on artificial labels depending on which NN error is smaller.

$$L_{al} = -\sum_{\mathbf{p}_i \in \mathcal{P}_t} [e_i < e_{r,i}] \log \sigma(l_{cls,i}) + [e_i \geq e_{r,i}] \log(1 - \sigma(l_{cls,i})) \tag{7}$$

Supervised loss: On datasets where ground truth flow labels $\mathcal{F}_{gt}$ and vehicle odometry $O_{t \to t+1}$ are available, it is possible to train the network using supervised losses:

$$L_{flow} = \frac{1}{|\mathcal{P}_t|} \sum_{\mathbf{p}_i \in \mathcal{P}_t} |\mathbf{f}_{gt,i} - \mathbf{f}_i| \tag{8}$$

Additionally, we supervise the classification logits by specifying a ground truth rigid class target through a threshold m_{thresh} to the non-rigid flow part

$$r_i = \left[|\mathbf{f}_{gt,i} - (O_{t \to t+1}^{-1} - I_4)\mathbf{p}_i| \leq m_{thresh} \right]. \tag{9}$$

The resulting cross-entropy-based classification loss is

$$L_{cls} = -\frac{1}{|\mathcal{P}_t|} \sum_{\mathbf{p}_i \in \mathcal{P}_t} r_i \log \sigma(l_{cls,i}) + (1 - r_i) \log(1 - \sigma(l_{cls,i})). \tag{10}$$

The last supervised loss is used to increase the performance of the rigid transform mostly by providing a signal to the confidence weights used by the computation of the transform.

$$L_r = \frac{1}{|\mathcal{P}_t|} \sum_{\mathbf{p}_i \in \mathcal{P}_t} |(T_r - O_{t \to t+1}^{-1})\mathbf{p}_i| \tag{11}$$

Classification Threshold: We observe in our experiments that our scores and metrics depend on a well-calibrated threshold p_{stat} for switching between the case of moving or stationary points. As our training pipeline already requires this threshold for the weights in the computation of T_r we require an online mechanism to adjust this threshold, instead of chosing it during post-processing. Therefore, we track moving averages of the metric $m(p)$ we want to optimize for a fixed set of possible thresholds $\{p_0, p_1, ..., p_N\}$. We can then define the threshold $p_{stat} = p_i$ with

$$i = \underset{i=0,...,N}{\operatorname{argmin}}\ m(p_i) \tag{12}$$

as the point where our metric is currently optimal. It is important to note, that in the case of self-supervised trainings we only make use of the k-NN errors e_i and $e_{r,i}$ which do not require any ground truth labels but also only approximate the metric we want to optimize. In the supervised setting, on the other hand, we use the ground truth flow to compute the moving averages of the metric as accurately as possible. For all our experiments we optimize the threshold according to the AEE 50-50 metric which we introduce in the following section. Please refer to the supplementary material for more details.

4. Evaluation

To verify our claims, we first report results for different experimental setups and compare our method against two other baselines using in total 4 different datasets. In a final set of experiments, we highlight the importance of the different components of our method through an ablation study.

4.1. Experimental Setup

We applied only minor modifications to the PointPillars (PP) PFN encoder and the RAFT backbone (called RAFT-S in [39]). Details on our architecture can be found in the supplementary. We trained all our experiments for 100k iterations with a batch size of 1 and a learning rate of 10^{-4} using the RMSProp optimizer. In order to stay consistent with previous evaluation protocols [27, 42, 53] we also remove all points up to 30cm above the ground from the point cloud prior to training and inference.

When training self-supervised we set the weights of the supervised losses to 0 and weighted the rest of the losses with $\lambda_{nn} = 2, \lambda_{rcc} = 1, \lambda_{al} = 0.1$. In the case of supervised trainings we only used the supervised losses with $\lambda_{flow} = 10, \lambda_{cls} = 1, \lambda_r = 10$.

Evaluation Metrics: To evaluate the performance, we use the established metrics and an extra outlier metric:

AEE average endpoint error (EE) across all points
AccS point ratio where EE < 0.05 or relative error < 0.05
AccR point ratio where EE < 0.1 or relative error < 0.1
Outl point ratio where EE > 0.3 or relative error > 0.1
ROutl point ratio where EE > 0.3 and relative error > 0.3

To ensure comparability with previous works we report Outl as a metric on KITTI-SF, however, we found it has several drawbacks, the most notable being that often the sum of Outl and AccR is larger than 100%, which is counter-intuitive for an outlier metric. On the other datasets, we exclusively report the robust variant ROutl defined above. A more detailed discussion and direct comparison of these two metrics can be found in the supplementary material.

As the other datasets we investigate contain a larger imbalance of moving scene parts w.r.t. stationary ones, we report scores separately for those two classes. Ground truth odometry is used to compute the non-rigid flow component $\mathbf{f}_{nr,gt,i} = \mathbf{f}_{gt,i} - (O_{t \to t+1}^{-1} - I_4)\mathbf{p}_i$. Points with a non-rigid flow larger than $m_{thresh} = 5cm$ (corresponds to $1.8 \frac{\text{km}}{\text{h}}$)

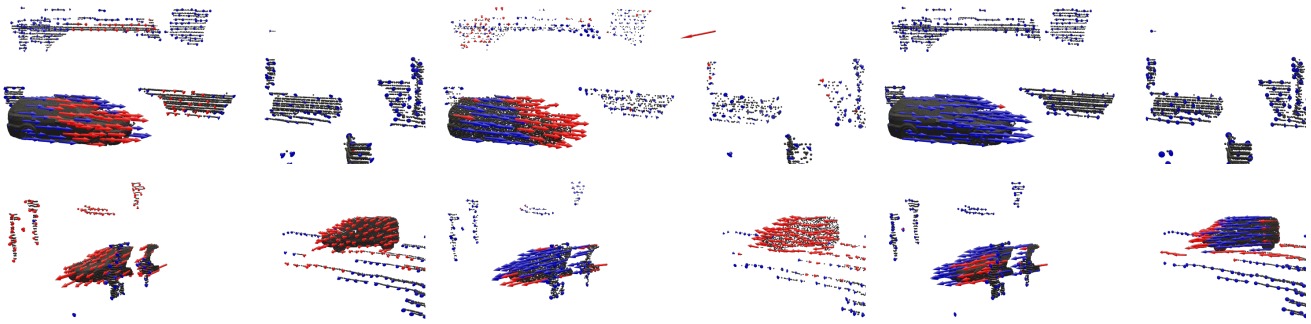

Figure 3: Qualitative comparison of methods on a KITTI-SF scene. Accurately estimated flow according to AccR is colored blue, inaccurate predictions red. From left to right: PointPWCNet (PPWC), PoseFlowNet (PF), Ours

are labeled as dynamic, the rest as stationary. We introduce AEE 50-50 as the average between the AEE measured separately on stationary and on moving points.

Baselines: We selected two methods as baselines which provide recent state-of-the-art results for self- as well as supervised training of scene flow: The first baseline we compare ourselves against, the PointPWCNet (PPWC) [53], uses a backbone similar to RAFT [39] in that it also relies on correlation volumes inside its architecture. The second baseline is taken from [42] and learns to predict the ego-motion component of the flow separately from the non-rigid flow, similar to our approach in the way that we also consider the ego-motion as a special case in our architecture. We refer to this method as PoseFlowNet (PF) in the following.

Datasets: To obtain representative results, we use three very different outdoor traffic scene datasets which only have in common that they observe the scene through a LiDAR from the position of an (autonomous) vehicle taking part in the traffic. Additionally, we use FT3D [24] for the baselines as training data.

FlyingThings3D: Both baseline methods make extensive use of the FT3D [24] dataset, consisting of randomly flying objects. The point clouds and the flow are 1-to-1 corresponding and have significantly more points than their counterparts from the other datasets. This dataset is not suited for our method as we make the assumptions that the flow is mainly in the horizontal x, y-plane and that points inside the same pillar have a similar movement pattern. Both of these assumptions do not hold for this dataset, therefore we do not use it to train our method.

nuScenes: nuScenes [5] is an autonomous driving (AD) dataset recorded using a Velodyne VLP32 LiDAR sensor, featuring densely annotated object boxes in space and time, as well as a pointwise semantic annotation. We annotated the point clouds with ground truth information using the recorded vehicle odometry for static points and the tracking information (bounding boxes) of moving objects.

CARLA: Using the CARLA Simulator [8], we collected an artificial dataset with 142k consecutive point clouds with ground truth flow annotation. Town07 is used exclusively for evaluation, where we recorded 1,900 different scenes.

KITTI: The KITTI-SF [26] training dataset is used by many works as the main benchmark for 3D scene flow [21, 53, 42, 18]. For these 142 samples, stereo ground truth and flow annotations are given, allowing conversion of the dataset to a point cloud representation with scene flow labels. In contrast to the nuScenes and CARLA dataset, it consists of pairs of 1-to-1 corresponding point clouds and flow, which differs from real-world point clouds measured at different time steps. We also compare our method against the baselines on this dataset. For training, we use the whole collection of unlabeled raw LiDAR point clouds published with the KITTI [9] dataset (abbreviated with KITTI-RL).

4.2. Impact of Point Cloud Density

Our network architecture is able to process a full scene from KITTI-RL as input, comprising roughly 64,000 points. While in theory the baselines can run inference on a similar size, during training it is infeasible to run them with more than the proposed 8,192 points as the memory requirements and the training time increase dramatically.

Therefore, increased point cloud density does not only provide more information but also represents a domain shift for the networks during inference. Our first set of experiments shows which type of evaluation produces better results for the baselines and our method and determines the evaluation mode we use in the following experiments.

Table 1 shows the results of these experiments. We train the baselines in a self-supervised fashion on FT3D, as it gives them the best scores when evaluating on KITTI-SF. For our network, we train self-supervised on KITTI-RL, as it, in turn, produces the best result on KITTI-SF.

We conclude that both baselines suffer from the domain shift due to different point densities, and do not benefit from the increased resolution. Therefore, we evaluate the baselines always on the downsampled point clouds and our

Table 1: Influence of point cloud density. Our method (*) is trained on KITTI-RL.

	Eval Data	#points	AEE↓	AccS↑	AccR↑	Outl↓
PPWC	FT3D	8192	**0.1610**	**0.2035**	**0.5809**	**0.7811**
		65536	0.1654	0.1999	0.5783	0.7876
	KITTI-SF	8192	**0.3466**	**0.0820**	**0.3145**	**0.8233**
		all	0.3648	0.0726	0.2974	0.8579
PF	FT3D	8192	**0.1726**	**0.0601**	**0.2336**	**0.9832**
		65536	0.1755	0.0584	0.2126	**0.9832**
	KITTI-SF	8192	**0.3009**	0.0986	**0.2296**	**0.9738**
		all	0.3256	**0.1104**	0.2058	0.9778
Ours*	KITTI-SF	8192	0.1207	0.5178	0.7956	0.4024
		all	**0.0668**	**0.7695**	**0.9342**	**0.2488**

Table 2: Self-supervised training with domain shift, evaluation on KITTI-SF

	Train Data	AEE↓	AccS↑	AccR↑	Outl↓
PPWC	FT3D	*0.3569*	0.0870	0.3202	0.8204
	nuScenes	0.7708	0.0897	0.2078	0.9193
	CARLA	0.8131	0.0094	0.0663	0.9928
	KITTI-RL	0.3712	*0.1992*	*0.4092*	*0.7406*
PF	FT3D	*0.3009*	0.0986	0.2296	0.9738
	nuScenes	0.4628	0.0000	0.1356	1.0000
	CARLA	0.3955	*0.1586*	*0.3866*	*0.9565*
	KITTI-RL	0.4448	0.0526	0.2310	0.9885
Ours	nuScenes	0.1013	0.7156	0.8739	0.3046
	CARLA	0.1454	0.5106	0.7899	0.3900
	KITTI-RL	**0.0668**	**0.7695**	**0.9342**	**0.2488**

method on the full point clouds.

4.3. Training Dataset and Domain Shift

The baselines have been evaluated on KITTI-SF with the networks trained on FT3D. We find that our method performs quite differently on KITTI-SF based on which dataset is used during training. Therefore, Table 2 showcases the validation performance of self-supervised models on KITTI-SF for different training datasets.

Our method gives the best results for KITTI-RL. This makes sense because except for the small camera FoV in KITTI-SF the geometric properties and their distribution match. The baselines give better results when trained on FT3D than on any other dataset. However, out of the other three LiDAR datasets, KITTI-RL seems to be the best choice. FT3D working better than KITTI-RL is also not surprising, as it resembles more the camera FoV as found in KITTI-SF. Qualitative results are shown in Fig. 3.

Table 3: Self-supervised training & evaluation on nuScenes

	Moving			Stat.	50-50
	AEE↓	AccR↑	ROutl↓	AEE↓	AEE↓
Zero	0.6381	0.1632	0.5783	0.5248	0.5814
PPWC	0.3539	0.2543	0.3848	0.1974	0.2756
PF	0.7399	0.0000	0.9364	**0.0570**	0.3985
Ours	**0.1050**	**0.7365**	**0.0240**	0.0925	**0.0987**

Table 4: Self-supervised training & evaluation on CARLA

	Moving			Stat.	50-50
	AEE↓	AccR↑	ROutl↓	AEE↓	AEE↓
Zero	0.4049	0.1805	0.4753	0.4752	0.4401
PPWC	0.3811	0.0948	0.5057	0.2577	0.3194
PF	0.5711	0.0085	0.9153	0.1093	0.3402
Ours	**0.0809**	**0.8351**	**0.0528**	**0.0853**	**0.0831**

4.4. Evaluation on LiDAR Data

For an evaluation closer to the real-world we perform self-supervised experiments on the two annotated datasets without point correspondences, CARLA, and nuScenes. Both datasets exhibit a much larger imbalance of moving points (non-rigid flow $> m_{thresh}$) compared to static points. Where in KITTI-SF around 40% of points are moving, only ≈5% move in nuScenes and CARLA. We, therefore, compute the AEE and Acc scores separately and report also the newly introduced AEE 50-50 for easier comparison of methods. Table 3 shows the results for the nuScenes dataset and Table 4 the results for CARLA, where we trained our method as well as the baselines on the respective datasets.

As one can see, both baseline methods perform much worse than our method. However, comparing their results to the ones they achieved on KITTI-SF suggests that the methods themselves have no real problem with missing correspondences when trained on more realistic data.

4.5. Supervised Flow Prediction

Finally, we present the best results for each method on the different datasets. All results are achieved using supervised training except in the case of our method and the KITTI-SF dataset. Here, our method performs best when trained self-supervised on KITTI-RL compared to supervised training on any of the other two annotated datasets, demonstrating the capabilities of our self-supervised model.

Table 5 and Table 6 show that our method greatly outperforms the baselines w.r.t. almost all metrics. The domain gap between KITTI-RL and KITTI-SF is the smallest of all, thus our network achieves the best result on KITTI-SF using self-supervision. The other methods are not able to profit from the small domain gap to the same extent.

In case of nuScenes and CARLA , the baselines strug-

Table 5: Best results on KITTI-SF for all methods. Note that PPWC and PF are trained using full supervision while our method uses only self-supervision.

Method	Train Data	AEE↓	AccS↑	AccR↑	Outl↓
PPWC	FT3D	0.2578	0.1803	0.4838	0.7176
PF	FT3D	0.1743	0.2716	0.5830	0.8729
Ours	KITTI-RL	**0.0668**	**0.7695**	**0.9342**	**0.2488**

Table 6: Best supervised results

	Method	Moving AEE↓	Moving AccR↑	Moving ROutl↓	Stat. AEE↓	50-50 AEE↓
CARLA	PPWC	0.2205	0.2456	0.4257	0.2120	0.2163
	PF	0.8143	0.0000	0.9024	0.1357	0.4750
	Ours	**0.04311**	**0.9547**	**0.0043**	**0.0466**	**0.0449**
nuScenes	PPWC	0.2341	0.4949	0.2034	0.0845	0.1593
	PF	0.7934	0.0000	1.0000	**0.0415**	0.4175
	Ours	**0.0702**	**0.8921**	**0.0170**	0.0499	**0.0600**

gle with the imbalance of moving points. Especially PF achieves promising results for the stationary points even outperforming our method on nuScenes. However, it completely ignores the moving objects. In contrast, our method uses a moving classification threshold and therefore gracefully adapts to datasets with changing amounts of moving points. We conclude that our method robustly handles different datasets with a consistent flow AEE 50-50 of roughly 5cm whereas the baselines perform drastically different and worse on the more realistic datasets.

4.6. Ablation Study

We now perform a series of experiments to ablate the importance of individual components, starting with our most basic setup: A PP encoder [20] with a RAFT backbone as a strong baseline, see Table 7. We first modify the RAFT backbone to update and track the flow map together with classification logits, which we use to apply an aggregation step to the masked ego-motion flow map in order to regularize the motion field for improved accuracy on the static parts of the scene. Then, another map of logits for producing weights used by the aggregation step is introduced, in theory, allowing the network to select regions with high flow certainty and resulting in even stronger performance on static scene elements. For the supervised training we observe a steady improvement in performance. However, in the self-supervised case, we observe that the confidence weights do not yield further improvements compared to pure aggregation. We attribute this to the fact that during self-supervised training, the moving thresholds are updated only through the very noisy kNN residuals. This is insufficient to guide the moving threshold to an optimal calibration. For a more detailed analysis, we refer to the supplementary material.

Table 7: Ablation study on nuScenes

	logit introd.	ego aggr.	conf. weights	Mov. AEE	Stat. AEE	50-50 AEE
Self-Supervised				0.1214	0.1889	0.1551
	✓	✓		**0.1050**	**0.0925**	**0.0987**
	✓	✓	✓	0.1413	0.1100	0.1256
Supervised				0.1273	0.1530	0.1401
	✓	✓		0.0933	**0.0474**	0.0704
	✓	✓	✓	**0.0702**	0.0499	**0.0600**

4.7. Motion Segmentation

As demonstrated in Table 7, the motion segmentation improves scene flow performance. Qualitative analysis of the logits in Fig. 4 reveals that the network indeed learns to pick out moving objects from the point clouds. With self-supervised training on KITTI-RL, our model achieves a mIoU score of 59.5% with a sensitivity of 73.1% on KITTI-SF. A more detailed discussion of motion segmentation can be found in the supplementary material.

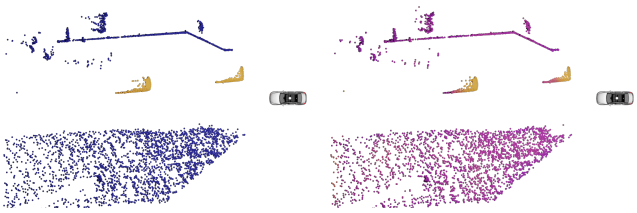

Figure 4: Left: Ground truth motion segmentation, right: Predicted dynamicness, higher moving probability brighter

5. Conclusion

We introduced SLIM, a new method for LiDAR scene flow estimation which improves over the state-of-the-art through several key contributions: We presented a new network design, allowing us to train and evaluate on larger point clouds, hence removing the need for downsampling. Our model classifies stationary and moving parts of the scene . We demonstrated that the resulting aggregation of static flow leads to a significant improvement of the rigid flow field, in turn enabling better self-supervised classification into stationary and moving points. In future work, we plan to explore better regularization strategies and more self-supervisory signals to further robustify our method.

6. Acknowledgements

Andreas Geiger was supported by the DFG EXC number 2064/1 - project number 390727645. Björn Ommer was supported by the German federal ministry BMWi within the project “KI Absicherung - Safe AI for automated driving”.

References

[1] Tali Basha, Yael Moses, and Nahum Kiryati. Multi-view scene flow estimation: A view centered variational approach. *Int. J. Comput. Vis.*, 101(1):6–21, 2013.

[2] Aseem Behl, Despoina Paschalidou, Simon Donné, and Andreas Geiger. Pointflownet: Learning representations for rigid motion estimation from point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 7962–7971. Computer Vision Foundation / IEEE, 2019.

[3] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jürgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 9296–9306. IEEE, 2019.

[4] Pia Bideau, Rakesh R. Menon, and Erik G. Learned-Miller. Moa-net: Self-supervised motion segmentation. In Laura Leal-Taixé and Stefan Roth, editors, *Computer Vision - ECCV 2018 Workshops - Munich, Germany, September 8-14, 2018, Proceedings, Part VI*, volume 11134 of *Lecture Notes in Computer Science*, pages 715–730. Springer, 2018.

[5] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 11618–11628. IEEE, 2020.

[6] Rodrigo L. Carceroni and Kiriakos N. Kutulakos. Multi-view scene capture by surfel sampling: From video streams to non-rigid 3d motion, shape and reflectance. *Int. J. Comput. Vis.*, 49(2-3):175–214, 2002.

[7] Yuhua Chen, Cordelia Schmid, and Cristian Sminchisescu. Self-supervised learning with geometric constraints in monocular video: Connecting flow, depth, and camera. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 7062–7071. IEEE, 2019.

[8] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 2017.

[9] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *Int. J. Robotics Res.*, 32(11):1231–1237, 2013.

[10] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow. Digging into self-supervised monocular depth estimation. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 3827–3837. IEEE, 2019.

[11] Zan Gojcic, Or Litany, Andreas Wieser, Leonidas J. Guibas, and Tolga Birdal. Weakly supervised learning of rigid 3d scene flow. *CoRR*, abs/2102.08945, 2021.

[12] 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.

[13] Kaveh Hassani and Mike Haley. Unsupervised multi-task feature learning on point clouds. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 8159–8170. IEEE, 2019.

[14] Frédéric Huguet and Frederic Devernay. A variational method for scene flow estimation from stereo sequences. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pages 1–7. IEEE Computer Society, 2007.

[15] Junhwa Hur and Stefan Roth. Self-supervised monocular scene flow estimation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 7394–7403. IEEE, 2020.

[16] Mariano Jaimez, Mohamed Souiai, Javier González Jiménez, and Daniel Cremers. A primal-dual framework for real-time dense RGB-D scene flow. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 98–104. IEEE, 2015.

[17] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Theoretical and General Crystallography*, 32(5):4, 1976.

[18] Yair Kittenplon, Yonina C. Eldar, and Dan Raviv. Flowstep3d: Model unrolling for self-supervised scene flow estimation. *CoRR*, abs/2011.10147, 2020.

[19] Hsueh-Ying Lai, Yi-Hsuan Tsai, and Wei-Chen Chiu. Bridging stereo matching and optical flow via spatiotemporal correspondence. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 1890–1899. Computer Vision Foundation / IEEE, 2019.

[20] 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.

[21] 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.

[22] Xingyu Liu, Mengyuan Yan, and Jeannette Bohg. Meteornet: Deep learning on dynamic 3d point cloud sequences. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 9245–9254. IEEE, 2019.

[23] Chenxu Luo, Zhenheng Yang, Peng Wang, Yang Wang, Wei Xu, Ram Nevatia, and Alan L. Yuille. Every pixel counts ++: Joint learning of geometry and motion with 3d holistic understanding. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(10):2624–2641, 2020.

[24] Nikolaus Mayer, Eddy Ilg, Philip Häusser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 4040–4048. IEEE Computer Society, 2016.

[25] Moritz Menze, Christian Heipke, and Andreas Geiger. Joint 3d estimation of vehicles and scene flow. In *Proc. of the ISPRS Workshop on Image Sequence Analysis (ISA)*, 2015.

[26] Moritz Menze, Christian Heipke, and Andreas Geiger. Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 2018.

[27] Himangi Mittal, Brian Okorn, and David Held. Just go with the flow: Self-supervised scene flow estimation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 11174–11182. IEEE, 2020.

[28] Frank Moosmann and Thierry Fraichard. Motion estimation from range images in dynamic outdoor scenes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 142–147, Anchorage, Alaska, USA, May 2010.

[29] Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 343–352. IEEE Computer Society, 2015.

[30] Jean-Philippe Pons, Renaud Keriven, and Olivier D. Faugeras. Modelling dynamic scenes by registering multi-view image sequences. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, pages 822–827. IEEE Computer Society, 2005.

[31] Jhony Kaesemodel Pontes, James Hays, and Simon Lucey. Scene flow from point clouds with or without learning. In Vitomir Struc and Francisco Gómez Fernández, editors, *8th International Conference on 3D Vision, 3DV 2020, Virtual Event, Japan, November 25-28, 2020*, pages 261–270. IEEE, 2020.

[32] Omid Poursaeed, Tianxing Jiang, Han Qiao, Nayun Xu, and Vladimir G. Kim. Self-supervised learning of point clouds via orientation estimation. In Vitomir Struc and Francisco Gómez Fernández, editors, *8th International Conference on 3D Vision, 3DV 2020, Virtual Event, Japan, November 25-28, 2020*, pages 1018–1028. IEEE, 2020.

[33] Gilles Puy, Alexandre Boulch, and Renaud Marlet. FLOT: scene flow on point clouds guided by optimal transport. 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 XXVIII*, volume 12373 of *Lecture Notes in Computer Science*, pages 527–544. Springer, 2020.

[34] Anurag Ranjan, Varun Jampani, Lukas Balles, Kihwan Kim, Deqing Sun, Jonas Wulff, and Michael J. Black. Competitive collaboration: Joint unsupervised learning of depth, camera motion, optical flow and motion segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 12240–12249. Computer Vision Foundation / IEEE, 2019.

[35] Jonathan Sauder and Bjarne Sievers. Context prediction for unsupervised deep learning on point clouds. *CoRR*, abs/1901.08396, 2019.

[36] Lin Shao, Parth Shah, Vikranth Dwaracherla, and Jeannette Bohg. Motion-based object segmentation based on dense RGB-D scene flow. *IEEE Robotics Autom. Lett.*, 3(4):3797–3804, 2018.

[37] Deqing Sun, Erik B. Sudderth, and H. Pfister. Layered rgbd scene flow estimation. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 548–556, 2015.

[38] E. Sun, D. et al.and Sudderth and M. J. Black. Layered segmentation and optical flow estimation over time. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1768–1775. IEEE, 2012.

[39] 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.

[40] Ali K. Thabet, Humam Alwassel, and Bernard Ghanem. Self-supervised learning of local features in 3d point clouds. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*, pages 4048–4052. IEEE, 2020.

[41] Hugues Thomas, Ben Agro, Mona Gridseth, Jian Zhang, and Timothy D. Barfoot. Self-supervised learning of lidar segmentation for autonomous indoor navigation. *CoRR*, abs/2012.05897, 2020.

[42] 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.

[43] Hsiao-Yu Tung, Hsiao-Wei Tung, Ersin Yumer, and Katerina Fragkiadaki. Self-supervised learning of motion capture. 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 5236–5246, 2017.

[44] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3d scene flow estimation with a piecewise rigid scene model. *Int. J. Comput. Vis.*, 115(1):1–28, 2015.

[45] Guangming Wang, Xinrui Wu, Zhe Liu, and Hesheng Wang. Hierarchical attention learning of scene flow in 3d point clouds. *CoRR*, abs/2010.05762, 2020.

[46] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 2589–2597. IEEE Computer Society, 2018.

[47] Xiaoying Wang. Three-dimensional flow optimization algorithm in complex scene based on differential equation. *Concurr. Comput. Pract. Exp.*, 30(22), 2018.

[48] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2794–2802. IEEE Computer Society, 2015.

[49] Xiaolong Wang, Allan Jabri, and Alexei A. Efros. Learning correspondence from the cycle-consistency of time. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 2566–2576. Computer Vision Foundation / IEEE, 2019.

[50] Zirui Wang, Shuda Li, Henry Howard-Jenkins, Victor Adrian Prisacariu, and Min Chen. Flownet3d++: Geometric losses for deep scene flow estimation. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2020, Snowmass Village, CO, USA, March 1-5, 2020*, pages 91–98. IEEE, 2020.

[51] Andreas Wedel, Clemens Rabe, Tobi Vaudrey, Thomas Brox, Uwe Franke, and Daniel Cremers. Efficient dense scene flow from sparse or dense stereo data. In David A. Forsyth, Philip H. S. Torr, and Andrew Zisserman, editors, *Computer Vision - ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part I*, volume 5302 of *Lecture Notes in Computer Science*, pages 739–751. Springer, 2008.

[52] Yi Wei, Ziyi Wang, Yongming Rao, Jiwen Lu, and Jie Zhou. PV-RAFT: point-voxel correlation fields for scene flow estimation of point clouds. *CoRR*, abs/2012.00987, 2020.

[53] 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.

[54] Jonas Wulff and Michael J. Black. Modeling blurred video with layers. In *Computer Vision – ECCV 2014*, volume 8694 of *Lecture Notes in Computer Science*, pages 236–252. Springer International Publishing, Sept. 2014.

[55] Saining Xie, Jiatao Gu, Demi Guo, Charles R. Qi, Leonidas J. Guibas, and Or Litany. Pointcontrast: Unsupervised pre-training for 3d point cloud understanding. 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 III*, volume 12348 of *Lecture Notes in Computer Science*, pages 574–591. Springer, 2020.

[56] Zhichao Yin and Jianping Shi. Geonet: Unsupervised learning of dense depth, optical flow and camera pose. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 1983–1992. IEEE Computer Society, 2018.

[57] Zaiwei Zhang, Rohit Girdhar, Armand Joulin, and Ishan Misra. Self-supervised pretraining of 3d features on any point-cloud. *CoRR*, abs/2101.02691, 2021.

[58] Yuliang Zou, Zelun Luo, and Jia-Bin Huang. Df-net: Unsupervised joint learning of depth and flow using cross-task consistency. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part V*, volume 11209 of *Lecture Notes in Computer Science*, pages 38–55. Springer, 2018.

[59] Victor Zuanazzi, Joris van Vugt, Olaf Booij, and Pascal Mettes. Adversarial self-supervised scene flow estimation. In Vitomir Struc and Francisco Gómez Fernández, editors, *8th International Conference on 3D Vision, 3DV 2020, Virtual Event, Japan, November 25-28, 2020*, pages 1049–1058. IEEE, 2020.