VoteFlow: Enforcing Local Rigidity in Self-Supervised Scene Flow

Lin, Y.; Wang, S.; Nan, L.; Kooij, J.F.P.; Caesar, Holger

DOI
10.1109/CVPR52734.2025.01599

Publication date
2025

Document Version
Final published version

Published in
Proceedings of the 2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2025)

2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) | 979-8-3315-4364-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/CVPR52734.2025.01599

VoteFlow: Enforcing Local Rigidity in Self-Supervised Scene Flow

Yancong Lin*,1,2, Shiming Wang*,1, Liangliang Nan[1], Julian Kooij[1] and Holger Caesar[1]
[1]TU Delft [2]ETH Zurich

Abstract

Scene flow estimation aims to recover per-point motion from two adjacent LiDAR scans. However, in real-world applications such as autonomous driving, points rarely move independently of others, especially for nearby points belonging to the same object, which often share the same motion. Incorporating this locally rigid motion constraint has been a key challenge in self-supervised scene flow estimation, which is often addressed by post-processing or appending extra regularization. While these approaches are able to improve the rigidity of predicted flows, they lack an architectural inductive bias for local rigidity within the model structure, leading to suboptimal learning efficiency and inferior performance. In contrast, we enforce local rigidity with a lightweight add-on module in neural network design, enabling end-to-end learning. We design a discretized voting space that accommodates all possible translations and then identify the one shared by nearby points by differentiable voting. Additionally, to ensure computational efficiency, we operate on pillars rather than points and learn representative features for voting per pillar. We plug the Voting Module into popular model designs and evaluate its benefit on Argoverse 2 and Waymo datasets. We outperform baseline works with only marginal compute overhead. Code is available at https://github.com/tudelft-iv/VoteFlow.

1. Introduction

Motion perception is essential for autonomous vehicles operating in dynamic environments. A crucial task in this domain, known as scene flow estimation, involves detecting per-point motion across consecutive LiDAR scans collected within short temporal intervals, e.g., 0.1 seconds given a 10Hz LiDAR scanner [3, 7, 29, 40]. Scene flow estimation has been the cornerstone in self-supervised scene understanding, which offers a way to interpret dynamic scenes without relying on extensively labeled data [11, 22, 34, 45, 49]. For example, [22] uses scene flow to associate moving objects over time and calculate object bounding boxes without any annotation cost. The generated object proposals provide valuable pseudo labels for subsequent model training. This strategy, also used in [6, 34, 45], leverages the low-cost nature of data collection to scale efficiently and thus generates large quantities of pseudo labels. Therefore, there is a perceivable demand for a robust scene flow estimator in autonomous driving.

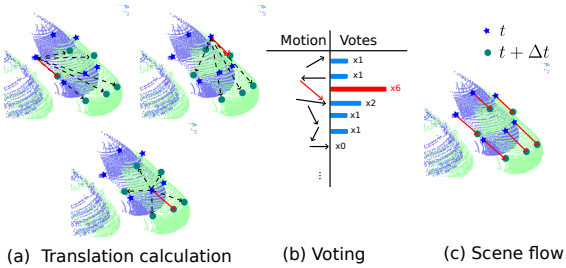

Figure 1. **Voting for identifying shared motion.** We take inspiration from motion rigidity and identify a motion shared by the majority via voting. We design a discrete voting space that encapsulates all possible translations that might occur within time Δt. For a given star ★ at time t, we calculate the displacements between itself and its neighbors ● at time $t + \Delta t$ and cast votes, defined by the cosine feature similarity between ★ and ●, to corresponding bins in the voting space. We accumulate votes from multiple spatially nearby points ★ at time t. The voting result indicates the likelihood of a motion shared by nearby points. Our differentiable voting is a light-weight add-on module compatible with popular model designs in scene flow estimation. In practice, voting takes pillars as input rather than individual points, thus reducing the computation substantially.

Scene flow methods typically assume *motion rigidity*, i.e. nearby points on rigid objects share the same motion. State-of-the-art methods exploit this assumption for self-supervised training through extra loss functions or regularizers [43, 51], or hard-coded post-processing [5]. However, these models lack the ability to encode motion rigidity by design. ICP-Flow* [25] enforces motion rigidity by using Iterative Closest Point (ICP) to align pre-clustered points. However, this can produce substantial errors if the given

*Equal contribution. Author ordering determined by coin flip.

*Winner of the Argoverse Scene Flow Challenge (unsupervised track) at CVPR 2024.

2575-7075/25/$31.00 ©2025 IEEE
DOI 10.1109/CVPR52734.2025.01599

clusters are over- or under-segmented (e.g., one cluster containing multiple close-by objects).

To overcome these limitations, we propose *VoteFlow*, a novel self-supervised scene flow estimation method that integrates feature learning and motion rigidity as an architectural inductive bias within its network design. The model extends the design of [15, 41, 50, 51], which has good generalization and inference speed, with a new differentiable Voting Module that identifies shared motion among neighboring points. In particular, the module efficiently identifies translation-dominated motion, which often characterizes object motion over short intervals in autonomous driving, by locally collecting votes among nearby LiDAR features across possible translational directions, as shown in Fig. 1. Our experiments show that VoteFlow outperforms the state of the art across most metrics not only on the Argoverse 2 benchmark but also on the Waymo Open dataset *without* retraining for that dataset. To summarize, our contributions are as follows:

- We introduce VoteFlow - a self-supervised scene flow estimator that encodes locally rigid motion by design.
- The core design of VoteFlow is a differentiable Voting Module - a light-weight add-on module that enables translation voting and end-to-end feature learning.
- VoteFlow outperforms state-of-the-art methods on the Argoverse 2 dataset with a considerably low inference latency. It also excels on the Waymo Open dataset in cross-dataset tests, outperforming baselines optimized on this particular dataset.

2. Related work

In this section we discuss prior works on scene flow estimation, approaches to enforce motion rigidity, and other usages of voting schemes in point cloud processing.

2.1. Scene flow estimation

Numerous works have emerged in scene flow estimation from large-scale point clouds in autonomous driving scenarios. Early works on scene flow, such as [3, 13, 14, 29], are supervised mainly by learning from annotated data, either from fully labeled real-world datasets [4, 12, 39] or synthetic datasets [8]. However, annotating large-scale datasets is costly, thus demanding the need for unsupervised alternatives. Later works remedy the need for labels by exploiting cycle consistency [2, 33, 48] between the forward and backward flows. Recently, test-time optimization techniques have been prevalent [5, 23, 24]. This family of works builds on top of multi-layer perceptrons (MLPs) and optimizes the cycle loss during inference. While the accuracy of these methods is outstanding, the inference time remains extended (up to several minutes [5, 23]) due to repeated optimization passes, making these methods impractical for online applications. To remedy the costly test-time inference, [41, 51] propose feed-forward neural networks as an alternative to test-time optimization, allowing for better generalization and faster inference. Particularly, [41] is capable of real-time inference. Our work also employs fast feed-forward networks.

2.2. Motion rigidity in scene flow estimation

A common assumption of scene flow estimation methods is that objects exhibit non-deformable motion. This rigid motion assumption can be exploited in several ways. One approach is to explicitly identify individual rigid objects and estimate a single rigid transformation for each such object. For instance, ICP-Flow [25] clusters points into objects during a pre-processing step to enforce per-cluster rigid motion. Clustering can also be applied in post-processing [5] to enforce rigid motion. However, clustering can produce substantial errors if objects are over- or under-segmented (e.g., one cluster containing multiple close-by objects), and clustering parameters are not end-to-end optimizable and therefore require extensive parameter tuning. Self-supervised training can also enforce consistent motion in a local neighborhood through extra loss functions or regularizers [43, 51]. However, such models rely on training the network to incorporate the rigidity assumption in the network weights, which is less data-efficient and harder to train due to potentially conflicting loss terms. PointPWC [48] improves locality and rigidity by constructing cost volumes and aggregating features from nearby points. However, its design is resource-intensive and does not explicitly enforce rigidity. Our model differs from previous works by explicitly encoding motion rigidity as an architectural inductive bias in the network design.

2.3. Voting in point cloud processing

Hough Voting [10] is a classic image processing technique initially designed for extracting geometric primitives from images, such as lines and circles. Later Hough Voting has been extended to find arbitrary shapes [1, 27, 28]. Hough Voting also has been widely used in processing point clouds, for segmentation [31], detection [21], tracking [32] and 6D pose estimation [38]. ICP-Flow [25] is a highly relevant work that extends the usage of majority voting to scene flow, where voting is used to localize the most dominant translation and to initialize the Iterative Closest Point algorithm. Our work also leverages majority voting for scene flow estimation but uses learned rather than hand-crafted features. Recently, there has been a line of works that encodes Hough Voting in deep learning, thus allowing for voting by learned features [20, 26, 35, 42, 47]. Our work shares the same spirit in combining voting with learned features.

3. Methodology

This section describes our proposed method, VoteFlow, and its novel Voting Module for efficiently identifying shared translations across local regions in LiDAR feature maps.

3.1. Problem statement

Scene flow estimation aims to recover a flow field (or equivalently point-wise translations) $\mathbf{F}^t \in \mathbb{R}^{3\times L} = \{\mathbf{f}_l \in \mathbb{R}^3\}_{l=1}^{L}$ from a pair of consecutive LiDAR scans $\mathbf{X}^t$ and $\mathbf{X}^{t+\Delta t}$, captured by an autonomous vehicle at time t and $t+\Delta t$, such that $\mathbf{X}^t + \mathbf{F}^t \approx \mathbf{X}^{t+\Delta t}$. $\mathbf{X} \in \mathbb{R}^{3\times L} = \{\mathbf{x}_l \in \mathbb{R}^3\}_{l=1}^{L}$ denotes a point cloud with L points. It is worth noting that $\mathbf{X}^t$ and $\mathbf{F}^t$ are of the same size, while $\mathbf{X}^t$ and $\mathbf{X}^{t+\Delta t}$ may differ in size. Following common practice in scene flow estimation [41, 46, 51], we also assume the ego motion is available, *i.e.,* the input points $\mathbf{X}^t$ and $\mathbf{X}^{t+\Delta t}$ have already been compensated by ego motion. Our goal is to develop a neural network that takes $\mathbf{X}^t$ and $\mathbf{X}^{t+\Delta t}$ as input and predicts $\mathbf{F}^t$, without relying on supervision.

3.2. Overview of VoteFlow

Fig. 2 shows an overview of VoteFlow. Given two consecutive LiDAR scans as input, the model first applies a Pillar Feature Net [19] ("pillarization") to convert both scans into a bird-eye-view pseudo image, where each grid cell ("pillar") represents a 2D location around the ego-vehicle and has an associated embedding. The model concatenates both pseudo images and learns features via a U-Net [37] backbone. Subsequently, the learned features go through our novel Voting Module, which creates a voting space for each *non-empty* pillar and applies convolution within the voting space. Later, the model retrieves points-wise features from the pseudo images, the fused features, and the voting features, appended by per-point offset with respect to the pillar center. The decoder converts point-wise features into point-wise scene flow.

We note that pillar representations of point clouds in autonomous driving are often sparse. A statistical analysis of the Argoverse 2 dataset [46] shows that more than 90% percent of pillars are empty[*]. Our proposed voting scheme exploits the sparsity of these feature maps.

The following subsections elaborate on each component.

3.2.1. Pillarization and backbone design

The model follows previous work [15, 41] and adopts the same setup during pillarization and backbone feature extraction. To be specific, we set the pillar size to be $\delta_y \times \delta_x$, resulting in a pseudo image of spatial size $H \times W$ for both LiDAR scans $\mathbf{X}^t$ and $\mathbf{X}^{t+\Delta t}$. The backbone takes as input the concatenation of both pseudo images $\mathbf{I}^t$ and $\mathbf{I}^{t+\Delta t}$ along the channel dimension and generates a fused feature map $\mathbf{G}$ of spatial size $H \times W$ using a U-Net with skip connections.

[*]As a common practice in scene flow estimation, ground points are removed from point clouds, as they provide little cue to predict motion [23, 41], which increases the proportion of empty pillars substantially.

3.2.2. Voting Module

To exploit motion rigidity in the network design, we observe that neighboring pillars on the same object are expected to share the same translation. Thus, we seek to identify the dominant translation from a number of nearby pillars. Our novel Voting Module creates a discretized voting space $\mathbf{V}$ for each non-empty pillar at time t that covers all possible translations that may occur within Δt. It then identifies M neighboring pillars at time t, and each of the M neighboring pillars casts votes for possible translations in $\mathbf{V}$ by identifying all neighboring pillars within a predefined radius at time $t+\Delta t$. After aggregating evidence on consistent motion by all neighboring pillars, a higher vote in $\mathbf{V}$ indicates more evidence for a particular translation. We elaborate on this concept as follows.

The module takes as input a set of indices of non-empty *source* pillars $\mathbf{P}^t$ at time t, the non-empty *target* pillar indices $\mathbf{P}^{t+\Delta t}$ at the next time instance, and the pseudo images ($\mathbf{I}^t$ and $\mathbf{I}^{t+\Delta t}$) from the Pillar Feature Net. It will output $\mathbf{H}$, which contains a *voting feature* for each pillar in $\mathbf{P}^t$.

For each pillar $\mathbf{p}_k^t \in \mathbf{P}^t$ the module selects the M spatially closest pillars $\{\mathbf{p}_{k,m}^t\}_{m=1}^{M}$ from the same time step (this set includes the pillar k itself). For each neighbor pillar $\mathbf{p}_{k,m}^t$ at time t a set of N spatially nearby pillars $\{\mathbf{p}_{k,m,n}^{t+\Delta t}\}_{n=1}^{N}$ at time $t+\Delta t$ has been selected from the available pillar indices $\mathbf{P}^{t+\Delta t}$ using the *ball query* function [36].

Note that each pair $(\mathbf{p}_{k,m}^t, \mathbf{p}_{k,m,n}^{t+\Delta t})$ represents a possible (discretized) 2D translation $\overrightarrow{T}_{k,m,n}$ of the m-th neighborhood pillar of k. We use the cosine similarity of their corresponding pseudo images $\mathbf{I}^t(\mathbf{p}_{k,m}^t)$ and $\mathbf{I}^{t+\Delta t}(\mathbf{p}_{k,m,n}^{t+\Delta t})$, as a 'voting score' $s_{k,m,n} \in [-1, +1]$ for the translation $\overrightarrow{T}_{k,m,n}$, such that a high score provides evidence in favor of this translation. Next, an empty voting space $\mathbf{V}_k^t$ is initialized for pillar k. With the range of allowed translation within Δt seconds set to (x_{min}, x_{max}) and (y_{min}, y_{max}), $\mathbf{V}_k^t$ is simply a discrete grid of size $H_v \times W_v$, where $H_v = \frac{y_{max}-y_{min}}{\delta_y}$ and $W_v = \frac{x_{max}-x_{min}}{\delta_x}$. All $M \times N$ votes are collected by summing their scores in the correct bins,

$$\mathbf{V}_k^t(\overrightarrow{T}_{k,m,n}) \leftarrow \mathbf{V}_k^t(\overrightarrow{T}_{k,m,n}) + s_{k,m,n} \quad \forall m, n. \tag{1}$$

Finally, all votes can be summarized. Simply taking the *argmax* on $\mathbf{V}_k^t$ has several downsides, however. It would only identify a single dominant direction, which can be erroneous during early training, and it could only produce a coarse discretized motion. Instead, our Voting Module summarizes the vote as a continuous feature vector such that the later decoder can access all voting information. This is achieved by applying two convolutional layers (with ReLUs) to each $\mathbf{V}_k^t$ and flattening the result. The voting fea-

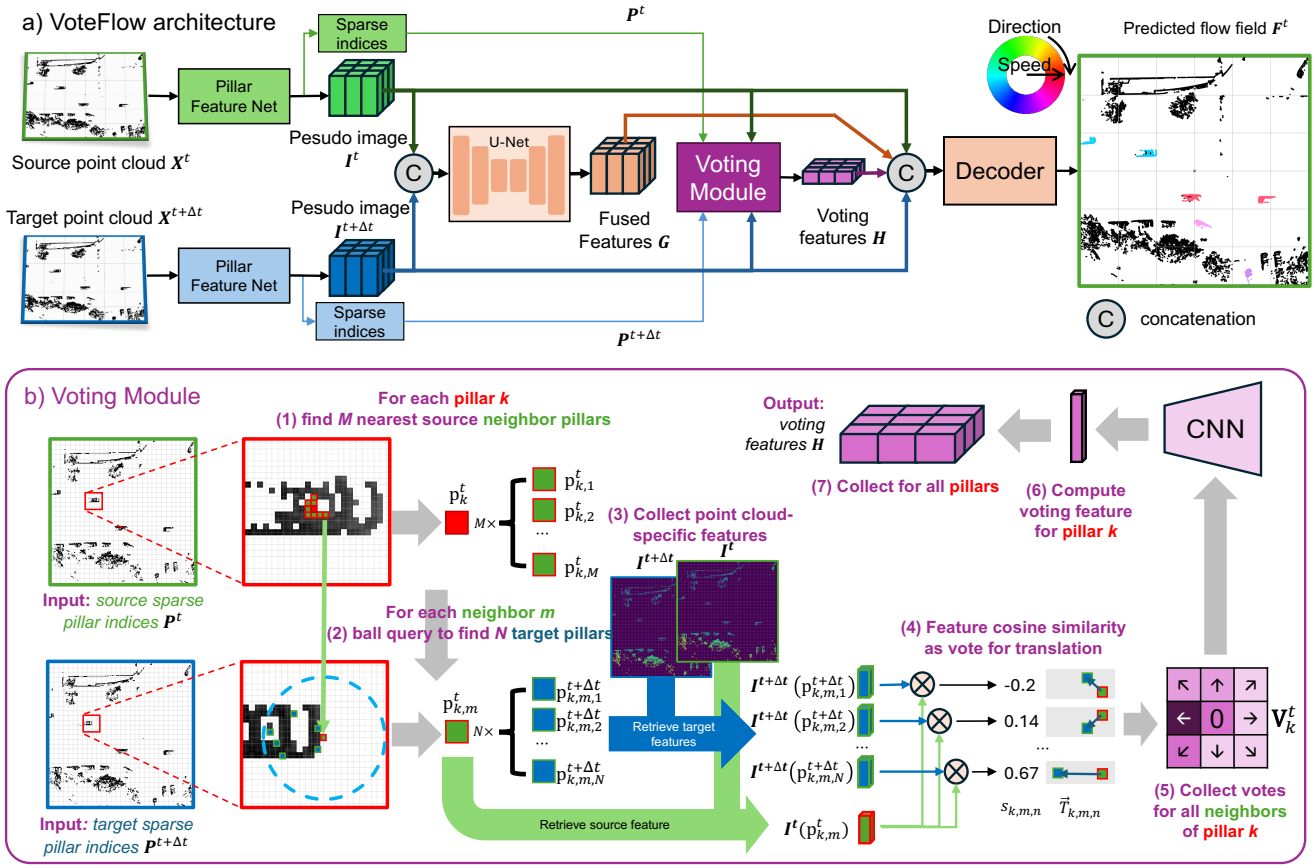

Figure 2. **The overall architecture of VoteFlow.** VoteFlow introduces a new end-to-end optimizable Voting Module that matches features in the local region around each pillar. Such matches are used to vote for a translation of the pillar. A CNN module summarizes the resulting votes as a 'voting feature'. The voting features are the output of the module and passed on to the decoder.

tures for all non-empty pillars are collected as $\mathbf{H}$ (the decoder will not need to lookup features for empty pillars).

3.2.3. Decoder

The decoder transforms the learned features into flow predictions. Similar to baseline works [15, 41], the decoder retrieves point-wise features from pseudo images $\mathbf{I}^t$ and $\mathbf{I}^{t+\Delta t}$, the fused features $\mathbf{G}$, but now also the voting features $\mathbf{H}$ by reusing the point-to-pillar indices. Additionally, the decoder also appends point-wise offsets (with respect to the pillar center) to point-wise features. The point-wise features go through 4 layers of fully connected layers (with ReLUs) and become per-point scene flow prediction. Notably, the decoder differs from SeFlow [51], as it uses fully connected layers rather than GRU layers. This is to reduce the computational cost during both training and inference.

3.3. Loss functions

We train VoteFlow in a self-supervised manner, adopting the loss functions from SeFlow [51]. The first loss is the common bidirectional Chamfer loss [23] $\mathcal{L}_{chamfer}$ that minimizes the distance between $\hat{\mathbf{X}}^t$ and $\hat{\mathbf{X}}^{t+\Delta t}$, where $\hat{\mathbf{X}}^t = \mathbf{X}^t + \mathbf{F}^t$. Let $\mathcal{D}(\mathbf{x}, \mathbf{Y}) = \min_{\mathbf{y}_k \in \mathbf{Y}} \|\mathbf{x} - \mathbf{y}_k\|$, then

$$\mathcal{L}_{chamfer} = \frac{1}{|\hat{\mathbf{X}}^t|} \sum_{\mathbf{x} \in \hat{\mathbf{X}}^t} \mathcal{D}(\mathbf{x}, \mathbf{X}^{t+\Delta t}) + \frac{1}{|\mathbf{X}^{t+\Delta t}|} \sum_{\mathbf{x} \in \mathbf{X}^{t+\Delta t}} \mathcal{D}(\mathbf{x}, \hat{\mathbf{X}}^t).$$

The other loss functions are: $\mathcal{L}_{dynamic}$, $\mathcal{L}_{static}$, and $\mathcal{L}_{cluster}$. $\mathcal{L}_{dynamic}$ is the same as $\mathcal{L}_{chamfer}$ but only applies to dynamic points, which are predefined by an offline method DUFOMap [9]. The purpose is to handle class imbalance as dynamic points are the minority of the points. $\mathcal{L}_{static}$ is only imposed on static points, which encourages static points to have zero flows. $\mathcal{L}_{cluster}$ is only on pre-clustered points computed by HDBSCAN [30] and encourages the points from the same cluster to have the same flow predictions. The total loss function is defined as $\mathcal{L}_{total} = \mathcal{L}_{chamfer} + \mathcal{L}_{dynamic} + \mathcal{L}_{static} + \mathcal{L}_{cluster}$. We refer the readers to baseline work [51] for details.

3.4. Implementation details

We follow previous works [41, 51] and Argoverse 2 official evaluation protocol to configure the hyperparameters. All point clouds are cropped within 102.4×102.4 meters and compensated by ego motion. Ground points have also been removed using rasterized HD maps. We set pillar sizes δ_x and δ_y to be 0.2 meters and maximal translations x_{max} and y_{max} to be 2 meters, resulting in a 20×20 voting space $\mathbf{V}$ per pillar where H_v and W_v are equal to 20. To conduct voting, we select $M = 8$ nearest neighbors for a given pillar at t and sample $N = 128$ neighboring pillars at $t + \Delta t$. The same setup also applies to the Waymo Open Dataset.

During training, we train VoteFlow for 12 epochs using Adam [18] optimizer and set the learning rate to the $2 \times e^{-4}$, which is later decreased by 10 after 6 epochs.

4. Experiments

We validate our approach on both the Argoverse 2 [46] and Waymo [39] datasets, which are the most commonly used datasets for scene flow in autonomous driving. This section describes the datasets, evaluation metrics and baselines, followed by evaluations and discussions.

4.1. Datasets

The **Argoverse 2** dataset [46] contains LiDAR scans captured by two roof-mounted 32-beam LiDARs, with a 0.1 second interval between consecutive scans. All LiDAR scans are compensated by ego motion. The ground points within the dataset are removed according to a rasterized height map. We conduct evaluations on the official test split. The Argoverse 2 2024 Scene Flow Challenge [16] has provided many baseline results on Argoverse 2.

We also evaluate on the **Waymo Open** [39] benchmark similar to [41, 51]. It contains 798 training and 202 validation scenes, with a LiDAR frequency of 10 Hz. Ground points are removed like Argoverse 2, and LiDAR scans are provided with ego-motion compensation.

4.2. Evaluation

We adopt the Bucketed Normalized EPE metric for evaluation on Argoverse 2 [16], which directly measures performance disparities across semantic classes and speed profiles, allowing us to normalize comparisons between classes moving at different speeds.

EPE stands for endpoint error between a predicted flow and ground truth flow. There are four classes in total: Car, Pedestrian, Other Vehicles, and Wheeled VRU. For each class, we compute the static EPE (in meters), which is the average endpoint error of all static points (with a speed lower than $0.4m/s$) and the dynamic normalized EPE (in ratio, defined as the mean of normalized endpoint errors over a set of predefined speed profiles, e.g., $0.4 - 0.8m/s$, $0.8 - 1.2m/s$, etc. Normalizing the endpoint error by speed ensures that high-speed objects (e.g., cars) and low-speed objects (e.g., pedestrians) are fairly evaluated. For example, a $0.5m/s$ error on a car moving $20m/s$ is negligible ($< 2.5\%$), while a $0.5m/s$ error on a pedestrian moving $0.5m/s$ fails to depict the pedestrian's motion.

On the Waymo Open dataset, we adopt the three-way EPE (in meters) evaluated on foreground static, foreground dynamic, and background, since Bucketed Normalized EPE is not yet available on the Waymo Open Dataset [16].

4.3. Baselines

We compare VoteFlow against five prominent baselines including NSFP [23], FastNSF [24], ZeroFlow [41], ICP-Flow [25] and SeFlow [51]. NSFP [23] and FastNSF [24] use test-time optimization. ZeroFlow [41] uses pseudo labels produced by NSFP [23] for training and exhibits strong scalability when ample data is available. ICP-Flow [25] incorporates motion rigidity explicitly into the design. However, it lacks the ability to learn strong features from data. SeFlow [51] is a recent work from ECCV'24 that achieves top performance on the leaderboard, thus providing a good reference for evaluating our model. Although VoteFlow focuses on self-supervised learning, we also include several fully supervised baselines, namely Flow4D [17], DeFlow [50], FastFlow3D [15] and TrackFlow [16].

4.4. Comparison on Argoverse 2

Tab. 1 shows the comparison on the Argoverse 2 test split, including both supervised and self-supervised approaches. We directly take the results from the Argoverse 2 Scene Flow Challenge Leaderboard*. We focus primarily on the dynamic errors over different categories, represented by normalized EPE (in ratio), since static errors, indicated by EPE (in meters) are negligible.

On average, we improve over the previous best, SeFlow, by 2.0%pt (percentage points) in dynamic normalized EPE averaged over all four categories, indicating the effectiveness of the motion rigidity prior in VoteFlow. Among all four categories, VoteFlow outperforms SeFlow on Car, Other Vehicles, Pedestrian, and Wheeled VRU by a margin of 1.2%pt, 0.4%pt, 4.6%pt, and 1.8%pt, respectively. An illustrative example to demonstrate the performance gap is that, given a car moving at 20 m/s, we reduce the EPE error by approximately 2.4 cm, as the normalized EPE is equivalent to the estimated speed error from EPE divided by ground truth speed. ICP-Flow is a strong baseline that achieves the best results on Car. However, its performance lags behind by a large margin on Wheeled VRU, thus leading to an inferior overall result.

Compared to supervised models, there is still a performance gap of 11.5%pt between our VoteFlow and the best-

*https://www.argoverse.org/sceneflow.html

Method	Labels	Bucketed Normalized EPE ↓						3-way EPE ↓ (in meters)			
		Dynamic (normalized EPE)					Static (EPE, in meters)	Avg.	FD	BS	FS
		Mean	Car	O. V.	Pd.	W. V	Mean				
FastFlow3D [15]	✓	0.532	0.243	0.391	0.982	0.514	0.018	0.062	0.156	0.005	0.024
DeFlow [50]	✓	0.276	0.113	0.228	0.496	0.266	0.022	0.034	0.073	0.004	0.025
TrackFlow [16]	✓	0.269	0.182	0.305	0.358	0.230	0.045	0.047	0.103	0.002	0.037
Flow4D [17]	✓	0.174	0.096	0.167	0.278	0.155	0.012	0.025	0.057	0.003	0.015
FastNSF [24]		0.383	0.269	0.413	0.500	0.325	0.074	0.112	0.163	0.091	0.081
NSFP [23]		0.422	0.251	0.331	0.723	0.383	0.028	0.061	0.116	0.034	0.032
ZeroFlow [41]		0.594	0.327	0.476	0.966	0.608	0.020	-	-	-	-
ICP-Flow [25]		0.331	**0.195**	0.331	0.435	0.363	0.027	0.065	0.137	0.025	0.033
SeFlow [51]		0.309	0.214	0.292	0.463	0.267	**0.014**	0.049	0.121	**0.006**	0.022
VoteFlow (ours)		**0.289**	0.202	**0.288**	**0.417**	**0.249**	**0.014**	**0.046**	**0.114**	**0.006**	**0.018**

Table 1. **Comparison on Argoverse 2 test split.** We compare all models using the Bucketed Normalized EPE, allowing for fine-grained analysis on individual classes, including Car, Other Vehicles (O. V.), Pedestrian (Pd), and Wheeled VRU (W. V.). The dynamic normalized EPE is a *ratio* as the endpoint error has been normalized by speed, while the other EPE metrics are in meters [16]. All results are from the Argoverse 2 Scene Flow Challenge Leaderboard. Our VoteFlow performs the best among all self-supervised models on mean dynamic normalized EPE. On individual classes, VoteFlow achieves the best result on Pedestrian and Wheeled VRU and performs competitively on Car and Other Vehicles, with only marginal gaps to the best. The inference time of VoteFlow is approximately 25.6 ms per sample on an A100 GPU.

Method	Labels	Same-domain training	EPE ↓ (in meters)		
			FD	FS	BS
FastFlow3D [15]	✓	✓	0.195	0.025	0.015
DeFlow [50]	✓	✓	0.098	0.026	0.010
FastNSF [24]		✓	0.301	0.015	0.040
NSFP [23]		✓	0.171	0.108	0.022
ZeroFlow [41]		✓	0.216	0.015	0.024
SeFlow [51]		✓	0.151	0.018	**0.011**
VoteFlow (ours)		✓	**0.117**	0.015	0.016
SeFlow [51]			0.155	0.018	0.013
VoteFlow (ours)			0.142	**0.014**	0.012

Table 2. **Comparison on Waymo Open validation split.** We report the EPE results on foreground dynamic (FD), foreground static (FS), and background static (BS). Although *without* training on Waymo, our model exhibits the best result across self-supervised models, indicating the generalization ability of VoteFlow across datasets.

performing Flow4D. Notably, Flow4D differs from others in using multiple LiDAR consecutive scans during training, indicating that exploring temporal information is a highly effective approach to further enhance the performance of scene flow estimation.

4.5. Comparison on Waymo Open

We also compare various models on the Waymo Open dataset, as shown in Tab. 2. The evaluation metric is EPE (in meters), evaluated on Foreground Dynamic (FD), Foreground Static (FS), and Background Static (BS). When training and valuation are both on Waymo, our model outperforms SeFlow by a margin of approximately 3 cm on FD. Notably, our VoteFlow exhibits the best performance among all self-supervised models on FD and FS even if we load the pretrained checkpoint on the Argoverse 2 dataset and directly test its performance on Waymo. VoteFlow excels *without* time-consuming and data-demanding training on the Waymo Open dataset, indicating its strong robustness across datasets.

4.6. Analysis on voting

We analyze the voting space qualitatively in Fig. 4 to validate its role in scene flow prediction. The middle column shows the voting space for a given pillar, indicated by a red triangle marker (▲) in the ground truth plot. We calculate a translation vector (indicated by the red arrow →) that points from the plot center (indicated by ×) to the $argmax$ bin (indicated by ■), i.e., the bin with the maximal vote. The center of voting space indicates zero translations and each bin represents a 0.2 × 0.2 meters region. As a reference, we also provide the ground truth flow (in digits) at the bottom of the ground truth figures. Overall, the $argmax$ bin in the voting space has a similar translation as the ground truth flow in both direction and displacement.

4.7. Ablation study

This section studies the impact of the key hyperparameters and the choice of decoder in model design.

4.7.1. Impact of M and N

M and N are two key hyperparameters in our Voting Module. Ideally, the M neighbors represent points from the same rigid body. We deploy k-NN search [36] to localize the nearest neighbors at time t. In contrast, N defines the

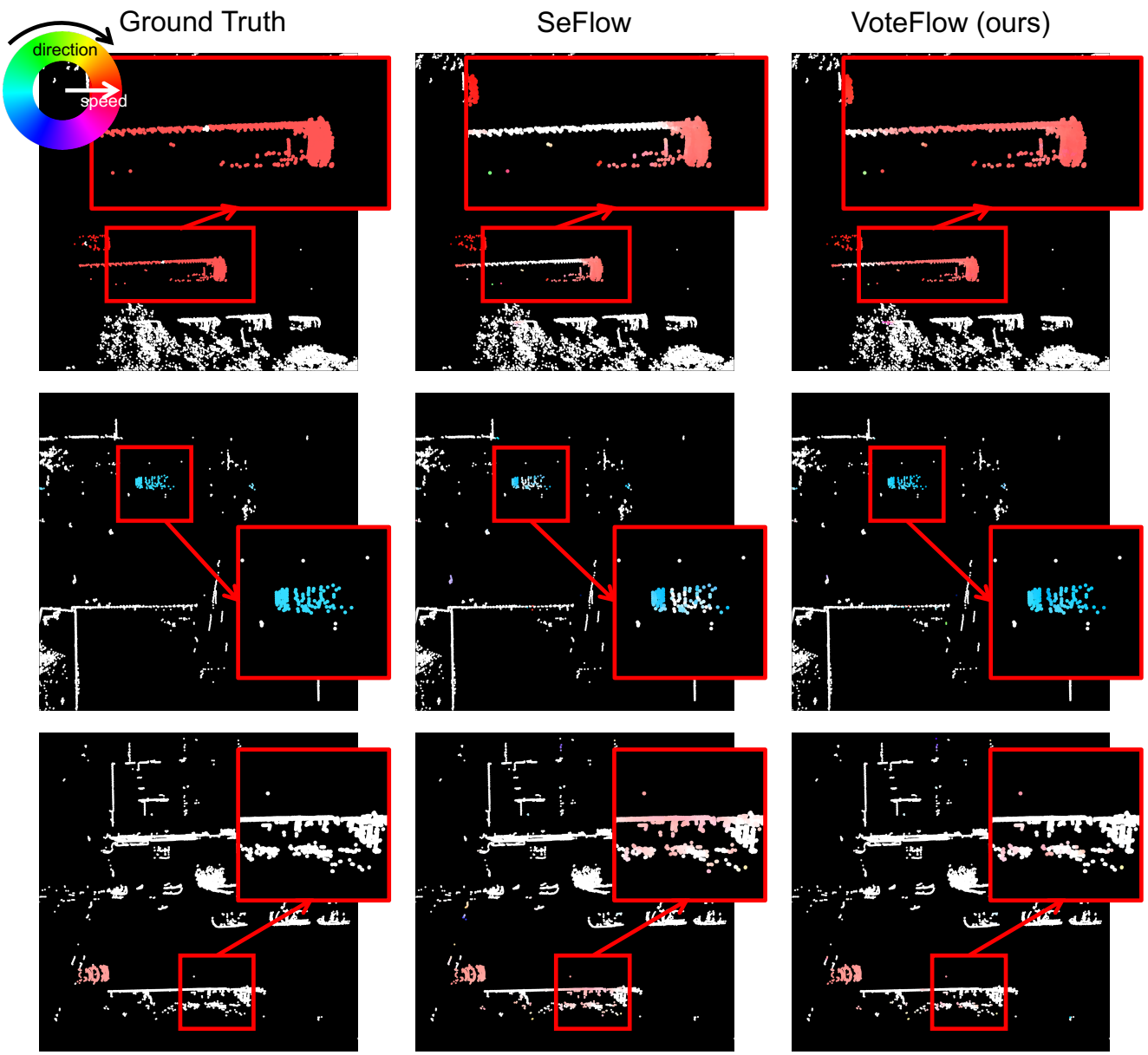

Figure 3. **Qualitative results on Argoverse 2 validation set.** Colors indicate directions and saturation of the color indicates the scale of the flow estimation. Thanks to the local rigidity prior, our VoteFlow predicts more consistent and coherent flow over objects compared to our baseline.

number of potential correspondences at $t + \Delta t$, which is expected to cover the entire search area. We employ the *ball query* function [36] to search potential neighbors within a specified radius. Our experiments in Tab. 3 showcase that changing M has a marginal impact on the performance. Raising N does not bring performance gain on Cars and Other Vehicles but benefits the result on small-scale objects, such as Pedestrians and Wheeled VRUs. This enhancement is likely due to the increased coverage of pillars that may contain an object at time $t + \Delta t$.

4.7.2. Choice of Decoder

Tab. 4 compares different decoder choices. SeFlow [51] employs a decoder with multiple GRU layers and iteratively refines features. We take the same design and insert our Voting Module. Compared to the SeFlow, VoteFlow with GRU decoder yields a substantially better result in the category Pedestrians but worse in Other Vehicles. Additionally, using GRU decoders slows down training. We further test a simple setup in our model, which consists of four fully

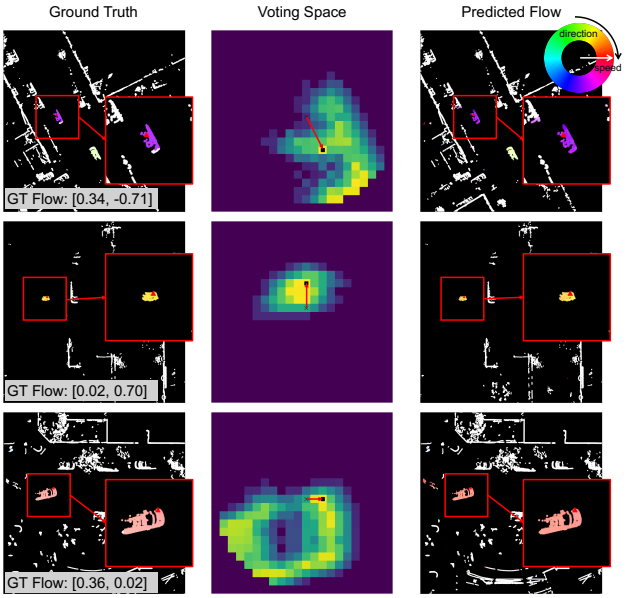

Figure 4. **Visualization of the voting space.** We select a pillar representing a moving object and plot its voting space, where the center indicates zero translations and boundaries indicate minimal and maximal translations along both dimensions. As shown in the voting space, the red arrow aligns with the ground truth flow (up to quantization errors). Overall, the voting space depicts the heatmap of the object's motion. For example, in the top row, the object is expected to move towards the bottom right, and the predicted heat map also has high responses along the same direction.

M	N	Latency (ms)	Bucket Normalized EPE (↓)				
			Dynamic (normalized EPE)				
			Mean	Car	O. V.	Pd.	W. V
4	128	27.3±5.0	0.337	0.223	0.355	0.434	0.338
8	128	25.6±5.2	**0.335**	0.222	0.347	0.424	0.347
16	128	27.2±5.2	0.337	0.223	**0.341**	0.444	0.338
32	128	28.7±5.2	0.336	0.222	0.350	0.434	0.337
64	128	81.9±17.3	0.339	**0.221**	0.352	0.442	0.341
8	64	**25.3±4.3**	0.343	**0.221**	0.362	0.449	0.341
8	256	27.4±5.4	0.337	0.222	0.370	**0.422**	**0.334**

Table 3. **Ablation study on M and N on Argoverse 2 val split.** We empirically test the influence of M and N on VoteFlow. Latency is measured on a single A100 GPU.

connected layers, and achieve significantly improved performance. Hence, our model uses the MLP decoder due to its computation efficiency and outstanding performance.

4.8. Qualitative evaluation

In Fig. 3, we show qualitative comparisons against SeFlow [51], the best-performing self-supervised baseline. Colors indicate directions and color saturation indicates the scale of the flow estimation. In the first row, we show the scene flow of a long bus, highlighted in the red box. Our method generates consistent predictions across nearly the entire vehicle, revealed by uniform coloring and coherent saturation. In contrast, predictions from SeFlow focus primarily on the front section, indicating a lack of rigid prior. Nevertheless, our method still encounters difficulties in yielding completely accurate predictions for this larger object. Similarly, in the second row, the SeFlow predicted flow on the object instance appears to be inconsistently colored compared to the ground truth, whereas our model achieves more uniform coloring overall. In the third row, SeFlow generates false positive predictions (FP) over a static object, implying that motion rigidity was not captured for the entire object. Despite a few false negatives, VoteFlow's predictions remain largely consistent with the ground truth. Qualitative results show that the architectural inductive bias in our VoteFlow improves the motion rigidity in scene flow prediction.

Decoder	# Params (M)	Bucketed Normalized EPE (↓)				
		Dynamic (normalized EPE)				
		Mean	Car	O. V.	Pd.	W. V
SeFlow [51]	0.077	0.369	0.234	**0.342**	0.541	0.358
VoteFlow (GRU decoder [50])	0.100	0.354	**0.221**	0.374	0.475	**0.344**
VoteFlow (MLP decoder)	**0.087**	**0.335**	0.222	0.347	**0.424**	0.347

Table 4. **Ablation Study on different decoder choices on Argoverse 2 val split.**

5. Conclusions

Our novel method, VoteFlow, is a state-of-the-art self-supervised scene flow estimation method. By using a novel Voting Module, we match pillar features across local spatial regions of subsequent LiDAR scans. This presents a new approach to incorporate motion rigidity inductive bias into scene flow estimation, distinct from loss-based and clustering-based approaches found in prior work. Our extensive experiments show that VoteFlow is competitive or outperforms other self-supervised models on Argoverse 2 and Waymo, and also when testing on Waymo *even without training on Waymo*.

Limitations and future work. In our work, the size of the voting space V, $H_v \times W_v$, depends on the magnitude of potential translation x_{max} and y_{max}. As Δt increases, the maximum possible translation grows, expanding the voting space and increasing the computational burden. In future work, we aim to extend the voting strategy to incorporate rotations alongside translations and enhance the voting module's efficiency, particularly for longer time intervals.

Furthermore, fusing multi-modal information of an autonomous driving car for joint scene flow and optical flow estimation is also of our interest [44].

Acknowledgement. Y. Lin was supported by NWO NGF-AiNed XS (file number: NGF.1609.23.015). S. Wang was supported by the 3D Urban Understanding (3DUU) Lab funded by the TU Delft AI Initiative.

References

[1] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *PR*, 1981. 2

[2] Stefan Andreas Baur, David Josef Emmerichs, Frank Moosmann, Peter Pinggera, Björn Ommer, and Andreas Geiger. Slim: Self-supervised lidar scene flow and motion segmentation. In *ICCV*, 2021. 2

[3] Aseem Behl, Despoina Paschalidou, Simon Donné, and Andreas Geiger. Pointflownet: Learning representations for rigid motion estimation from point clouds. In *CVPR*, 2019. 1, 2

[4] 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 *CVPR*, 2020. 2

[5] Nathaniel Chodosh, Deva Ramanan, and Simon Lucey. Re-evaluating lidar scene flow for autonomous driving. In *WACV*, 2024. 1, 2

[6] Minh-Quan Dao, Holger Caesar, Julie Stephany Berrio, Mao Shan, Stewart Worrall, Vincent Frémont, and Ezio Malis. Label-efficient 3d object detection for road-side units. In *IV*, 2024. 1

[7] Ayush Dewan, Tim Caselitz, Gian Diego Tipaldi, and Wolfram Burgard. Rigid scene flow for 3d lidar scans. In *IROS*, 2016. 1

[8] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, 2015. 2

[9] Daniel Duberg, Qingwen Zhang, MingKai Jia, and Patric Jensfelt. DUFOMap: Efficient dynamic awareness mapping. *RA-L*, 2024. 4

[10] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 1972. 2

[11] Emeç Erçelik, Ekim Yurtsever, Mingyu Liu, Zhijie Yang, Hanzhen Zhang, Pınar Topçam, Maximilian Listl, Yılmaz Kaan Caylı, and Alois Knoll. 3d object detection with a self-supervised lidar scene flow backbone. In *ECCV*, 2022. 1

[12] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *IJRR*, 2013. 2

[13] Shengyu Huang, Zan Gojcic, Jiahui Huang, Andreas Wieser, and Konrad Schindler. Dynamic 3d scene analysis by point cloud accumulation. In *ECCV*, 2022. 2

[14] Zhao Jin, Yinjie Lei, Naveed Akhtar, Haifeng Li, and Munawar Hayat. Deformation and correspondence aware unsupervised synthetic-to-real scene flow estimation for point clouds. In *CVPR*, 2022. 2

[15] Philipp Jund, Chris Sweeney, Nichola Abdo, Zhifeng Chen, and Jonathon Shlens. Scalable scene flow from point clouds in the real world. *RA-L*, 2021. 2, 3, 4, 5, 6

[16] Ishan Khatri, Kyle Vedder, Neehar Peri, Deva Ramanan, and James Hays. I can't believe it's not scene flow! In *ECCV*, 2025. 5, 6

[17] Jaeyeul Kim, Jungwan Woo, Ukcheol Shin, Jean Oh, and Sunghoon Im. Flow4d: Leveraging 4d voxel network for lidar scene flow estimation. *RA-L*, 2024. 5, 6

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 5

[19] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, 2019. 3

[20] Junha Lee, Seungwook Kim, Minsu Cho, and Jaesik Park. Deep hough voting for robust global registration. In *ICCV*, 2021. 2

[21] Alain Lehmann, Bastian Leibe, and Luc Van Gool. Fast prism: Branch and bound hough transform for object class detection. *IJCV*, 2011. 2

[22] Ted Lentsch, Holger Caesar, and Dariu M Gavrila. UNION: Unsupervised 3d object detection using object appearance-based pseudo-classes. *NeurIPS*, 2024. 1

[23] Xueqian Li, Jhony Kaesemodel Pontes, and Simon Lucey. Neural scene flow prior. *NeurIPS*, 2021. 2, 3, 4, 5, 6

[24] Xueqian Li, Jianqiao Zheng, Francesco Ferroni, Jhony Kaesemodel Pontes, and Simon Lucey. Fast neural scene flow. In *ICCV*, 2023. 2, 5, 6

[25] Yancong Lin and Holger Caesar. ICP-Flow: Lidar scene flow estimation with icp. In *CVPR*, 2024. 1, 2, 5, 6

[26] Yancong Lin, Silvia L Pintea, and Jan C van Gemert. Deep hough-transform line priors. 2020. 2

[27] Yancong Lin, Silvia-Laura Pintea, and Jan van Gemert. Nerd++: Improved 3d-mirror symmetry learning from a single image. *BMVC*, 2022. 2

[28] Yancong Lin, Ruben Wiersma, , Silvia L Pintea, Klaus Hildebrandt, Elmar Eisemann, and Jan C van Gemert. Deep vanishing point detection: Geometric priors make dataset variations vanish. 2022. 2

[29] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *CVPR*, 2019. 1, 2

[30] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2017. 4

[31] Fausto Milletari, Seyed-Ahmad Ahmadi, Christine Kroll, Christoph Hennersperger, Federico Tombari, Amit Shah, Annika Plate, Kai Boetzel, and Nassir Navab. Robust segmentation of various anatomies in 3d ultrasound using hough forests and learned data representations. In *MICCAI*, 2015. 2

[32] Fausto Milletari, Wadim Kehl, Federico Tombari, Slobodan Ilic, Seyed-Ahmad Ahmadi, Nassir Navab, et al. Universal hough dictionaries for object tracking. In *BMVC*, 2015. 2

[33] Himangi Mittal, Brian Okorn, and David Held. Just go with the flow: Self-supervised scene flow estimation. In *CVPR*, 2020. 2

[34] Mahyar Najibi, Jingwei Ji, Yin Zhou, Charles R Qi, Xinchen Yan, Scott Ettinger, and Dragomir Anguelov. Motion inspired unsupervised perception and prediction in autonomous driving. In *ECCV*, 2022. 1

[35] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *ICCV*, 2019. 2

[36] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020. 3, 6, 7

[37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 3

[38] Min Sun, Gary Bradski, Bing-Xin Xu, and Silvio Savarese. Depth-encoded hough voting for joint object detection and shape recovery. In *ECCV*, 2010. 2

[39] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*, 2020. 2, 5

[40] Arash K Ushani, Ryan W Wolcott, Jeffrey M Walls, and Ryan M Eustice. A learning approach for real-time temporal scene flow estimation from lidar data. In *ICRA*, 2017. 1

[41] Kyle Vedder, Neehar Peri, Nathaniel Chodosh, Ishan Khatri, Eric Eaton, Dinesh Jayaraman, Yang Liu, Deva Ramanan, and James Hays. Zeroflow: Fast zero label scene flow via distillation. In *ICLR*, 2024. 2, 3, 4, 5, 6

[42] Alexander Velizhev, Roman Shapovalov, and Konrad Schindler. Implicit shape models for object detection in 3d point clouds. *ISPRS Annals*, 2012. 2

[43] Kavisha Vidanapathirana, Shin-Fang Chng, Xueqian Li, and Simon Lucey. Multi-body neural scene flow. In *3DV*, 2024. 1, 2

[44] Shiming Wang, Holger Caesar, Liangliang Nan, and Julian FP Kooij. Unibev: Multi-modal 3d object detection with uniform bev encoders for robustness against missing sensor modalities. In *IV*, 2024. 8

[45] Yuqi Wang, Yuntao Chen, and Zhao-Xiang Zhang. 4d unsupervised object discovery. *NeuIPS*, 2022. 1

[46] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, Deva Ramanan, Peter Carr, and James Hays. Argoverse 2: Next generation datasets for self-driving perception and forecasting. In *NeurIPS*, 2021. 3, 5

[47] Oliver J Woodford, Minh-Tri Pham, Atsuto Maki, Frank Perbet, and Björn Stenger. Demisting the hough transform for 3d shape recognition and registration. *IJCV*, 2014. 2

[48] Wenxuan Wu, Zhi Yuan Wang, Zhuwen Li, Wei Liu, and Li Fuxin. Pointpwc-net: Cost volume on point clouds for (self-) supervised scene flow estimation. In *ECCV*, 2020. 2

[49] Guangyao Zhai, Xin Kong, Jinhao Cui, Yong Liu, and Zhen Yang. Flowmot: 3d multi-object tracking by scene flow association. *arXiv preprint arXiv:2012.07541*, 2020. 1

[50] Qingwen Zhang, Yi Yang, Heng Fang, Ruoyu Geng, and Patric Jensfelt. Deflow: Decoder of scene flow network in autonomous driving. In *ICRA*, 2024. 2, 5, 6, 8

[51] Qingwen Zhang, Yi Yang, Peizheng Li, Olov Andersson, and Patric Jensfelt. Seflow: A self-supervised scene flow method in autonomous driving. In *ECCV*, 2025. 1, 2, 3, 4, 5, 6, 7, 8