Hidden Gems: 4D Radar Scene Flow Learning Using Cross-Modal Supervision - Supplementary Material

Fangqiang Ding[1] Andras Palffy[2] Dariu M. Gavrila[2] Chris Xiaoxuan Lu[1,*]
[1]University of Edinburgh
[2]Delft University of Technology
{fding, xiaoxuan.lu}@ed.ac.uk {a.palffy, d.m.gavrila}@tudelft.nl

This supplementary document is organized as follows:

- Sec. A illustrates more details about the dataset and the splits we used in our experiments.
- Sec. B lists all symbols used in our approach together with their meanings and dimensions.
- Sec. C introduces more details about our multi-task model architecture.
- Sec. D presents more implementation details on retrieving cross-modal supervision and losses.
- Sec. E provides more detailed experimental results and analysis of our approach.
- Sec. F shows more qualitative scene flow, motion segmentation and ego-motion estimation results.

Besides, we also provide supplementary **demo** videos at https://youtu.be/PjKgznDizhI.

A. Dataset Details

Dataset separation. As discussed in the main text, the test set annotations of the official *View-of-Delft* (VoD) dataset [14] are withheld for benchmarking. Our task, however, needs to compute custom scene flow metrics with ground truth labels, which demands manually annotated 3D bounding boxes. Thus, for our experiments, we split the VoD dataset [14] ourselves. In these new splits, we kept the original *Val* set unchanged and divided a part of the original training set into a new *Test* set so that we can generate ground truth scene flow using object annotations for evaluation. The remaining sequences from the original training set are used for our training. To avoid wasting the data from the original testing set, we add its frames to our training data and form a new *Train* set. Note that we remove all annotations from the frames in this *Train* set and use it for self-supervised or cross-modal supervised learning methods. Concrete details of our splits can be found in Tab. A.

Data preprocessing. Given sequences of radar point clouds, we first filter out the radar points outside the Field-of-View (FoV) of the camera as only objects partially or fully within the camera FoV were annotated in the dataset originally. Then, we set a z-axis (up-down direction) range [-3m, 3m] and filter points outside of this range as these very low or high points are often ghost targets. During training, we randomly sample 256 radar points for each point cloud to facilitate fast mini-batch learning. During inference, however, we keep the original number of points because our task is to estimate the flow vector of every radar point in the source frame. Finally, for each sequence, we form $L_{seq} - 1$ scene flow input samples by combining pairs of consecutive radar point clouds, where L_{seq} is the length (*i.e.* number of frames) of the sequence.

	Val	*Test*	*Train*
Number of frames	1,296	2,724	4,662
Number of sequences	4	7	13
Is it annotated?	True	True	False

Table A. Details of our new splits for the VoD dataset [14].

Ground truth labelling. As mentioned in the main text, we only annotate ground truth scene flow and motion segmentation labels for the samples from *Val* and *Test* sets. Following [1, 6], we annotate ground truth scene flow by using object annotations (*i.e.*, bounding boxes and track IDs) and ground truth radar ego-motion (calculated from the RTK-GPS/IMU based odometer). For points belonging to the static background, we label their flow vectors with the ground truth radar ego-motion. For foreground objects, we track the ID of each annotated bounding box across consecutive point clouds and compute their rigid transformation *w.r.t* the radar coordinate frame. Each foreground point is assigned a ground truth flow vector, which is summarized by this corresponding rigid transformation. Then, for each foreground point belonging to an object, we assign a ground truth flow vector derived from the rigid transformation of the object. To obtain the ground truth motion segmentation, we first compute the non-rigid flow vector $\mathbf{f}_i^{nr}$ for each point by compensating the ground truth rigid ego-motion as $[\mathbf{f}_i^{nr}\ 1]^\top = [\mathbf{f}_i^{gt}\ 1]^\top - (\mathbf{T} - \mathbf{I}_4)[\mathbf{c}_i\ 1]^\top$. Then, points with $|\mathbf{f}_i^{nr}| > \zeta_{mov} = 5$cm are labelled as moving, while the rest

*Corresponding author: Chris Xiaoxuan Lu (xiaoxuan.lu@ed.ac.uk)

are labelled as static.

B. Notation

For the convenience of reference, we summarize all symbols used in our approach together with their meanings and dimensions in Tab. B.

C. Model Architecture Details

Here, we introduce more details about our module design and layer hyperparameters of our model architecture.

Backbone. The input to our backbone is two consecutive 4D radar point clouds, $\mathbf{P}^s \in \mathbb{R}^{N\times(3+C)}$ and $\mathbf{P}^t \in \mathbb{R}^{M\times(3+C)}$. As the foremost step, we use the *set conv* layer [12] to extract local features at different scales. Detailed layer parameters of this multi-scale *set conv* layer are as follows:

syntax: $SC([radii], [nsamples], [dimension])$

where $[radii]$ denotes the grouping radii for multiple scales, $[nsamples]$ denotes the number of local sampled points, and $[dimension]$ are the latent feature sizes.

$$SC([2.0, 4.0, 8.0, 16.0], [4, 8, 16, 32], [[32, 32, 64], [32, 32, 64], [32, 32, 64], [32, 32, 64]]) \rightarrow MLP(256 \rightarrow 256 \rightarrow 256 \rightarrow 256)$$

Note that we do not downsample our radar point clouds because they are already very sparse. After concatenating the global feature vector to point-wise features, we obtain the local-global features for each individual input point cloud: $g_\theta(\mathbf{P}^s) \in \mathbb{R}^{N\times512}$ and $g_\theta(\mathbf{P}^t) \in \mathbb{R}^{M\times512}$.

To propagate the features from the target point cloud to the source, we adopt the *cost volume* layer in [22] for feature correlation. The costs are aggregated in a robust patch-to-patch manner and a MLP was used to learn point-to-point cost dynamically:

$$MLP(512 \times 2 + 3 \rightarrow 512 \rightarrow 512 \rightarrow 512)$$

In patch-to-patch cost aggregation, the number of neighbour points is set to 8. As a results, we can obtain the correlated features $h_\theta(g_\theta(\mathbf{P}^s), g_\theta(\mathbf{P}^t)) \in \mathbb{R}^{N\times512}$.

The flow embedding $FE \in \mathbb{R}^{N\times(512+512+C)}$ can be generated by concatenating the correlated features, the local-global features and the raw input features of $\mathbf{P}^s$. We further feed this flow embedding into another multi-scale *set conv* layer to get $\mathbf{L} \in \mathbb{R}^{N\times256}$:

$$SC([2.0, 4.0, 8.0, 16.0], [4, 8, 16, 32], [[512, 256, 64], [512, 256, 64], [512, 256, 64], [512, 256, 64]]) \rightarrow MLP(256 \rightarrow 256 \rightarrow 256 \rightarrow 256)$$

Another max-pooling operation follows to restore the global feature vector and concatenates it to each point. Finally, we obtain our base backbone features $\mathbf{E} \in \mathbb{R}^{N\times512}$.

Initial flow and motion segmentation head. As the base backbone feature $\mathbf{E}$ includes complementary features extracted from two input point clouds, we simply use the MLP to implement our initial flow and motion segmentation heads for decoding. The output of the initial flow are 3D flow vectors $\hat{\mathbf{F}}^{init} \in \mathbb{R}^{N\times3}$:

$$MLP(512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 3)$$

The motion segmentation head estimates per-point moving probabilities $\hat{\mathbf{S}} \in \mathbb{R}^N$:

$$MLP(512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 1).$$

Ego-motion head. The purpose of our ego-motion head is to derive a rigid transformation $\hat{\mathbf{T}}$ that can summarize the scene flow rigid component induced by the radar ego-motion. To that end, we leverage the differentiable weighted Kabsch algorithm [7] that solves the ego-motion estimation problem in a close form through:

$$\hat{\mathbf{T}}^* = \underset{\hat{\mathbf{T}}\in\mathbb{R}^{4\times4}}{\arg\min} \sum_{i=1}^{N} w_i ||(\hat{\mathbf{R}}\mathbf{c}_i^s + \hat{\mathbf{t}}) - \mathbf{c}_i^w||_2^2, \tag{1}$$

where $\mathbf{c}_i^w = \mathbf{c}_i^s + \hat{\mathbf{f}}_i^{init}$ is the point coordinate warped by the initial scene flow. The sum of all weights is normalized as $\sum_{i=1}^N w_i = 1$, and

$$\hat{\mathbf{T}} = \begin{bmatrix} \hat{\mathbf{R}} & \hat{\mathbf{t}} \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}, \tag{2}$$

where $\mathbf{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ is the translation vector. In our method, we compute the weights from the estimated moving probabilities $\hat{\mathbf{S}}$ via:

$$w_i = \frac{1 - \hat{s}_i}{\sum_{i=1}^N (1 - \hat{s}_i)}. \tag{3}$$

To solve Eq. (1), the first step is to compute the centred point coordinates for $\mathbf{C}^s = \{\mathbf{c}_i^s\}_{i=1}^N$ and $\mathbf{C}^w = \{\mathbf{c}_i^w\}_{i=1}^N$. To do that, the weighted centroid of $\mathbf{C}^s$ and $\mathbf{C}^w$ can be derived through:

$$\bar{\mathbf{c}}^s = \sum_{i=1}^{N} w_i \mathbf{c}_i^s, \quad \bar{\mathbf{c}}^w = \sum_{i=1}^{N} w_i \mathbf{c}_i^w \tag{4}$$

We subtract the centroid coordinates from $\mathbf{C}^s$ and $\mathbf{C}^w$ and obtain the centered point coordinates $\tilde{\mathbf{C}}^s \in \mathbb{R}^{N\times3}$ and $\tilde{\mathbf{C}}^w \in \mathbb{R}^{N\times3}$. A weighted covariance matrix $\mathbf{H}_w \in \mathbb{R}^{3\times3}$ can then be formulated as:

$$\mathbf{H}_w = \tilde{\mathbf{C}}^{s\top} diag(w_1, w_2, ..., w_N) \tilde{\mathbf{C}}^w. \tag{5}$$

Symbol	Meaning	Dimension	Symbol	Meaning	Dimension
$\mathbf{P}^s$	Input source point cloud	$\mathbb{R}^{N\times(3+C)}$	$\mathbf{P}^t$	Input target point cloud	$\mathbb{R}^{M\times(3+C)}$
$\mathbf{p}_i^s, \mathbf{p}_i^t$	i-th point of $\mathbf{P}^s$, $\mathbf{P}^t$	$\mathbb{R}^{3+C}$	$\mathbf{c}_i^s, \mathbf{c}_i^t$	3D coordinate of point $\mathbf{p}_i^s, \mathbf{p}_i^t$	$\mathbb{R}^3$
$\mathbf{x}_i^s, \mathbf{x}_i^t$	Associated raw features of point $\mathbf{p}_i^s, \mathbf{p}_i^t$	$\mathbb{R}^C$	N	Number of points in the source point cloud	$\mathbb{R}$
M	Number of points in the target point cloud	$\mathbb{R}$	C	Number of channels of the associated raw features	$\mathbb{R}$
$\mathbf{F}$	Scene flow between two input point clouds	$\mathbb{R}^{N\times3}$	$\mathbf{f}_i$	Scene flow vector of the point $\mathbf{p}_i^s$	$\mathbb{R}^3$
$\mathbf{c}_i'$	3D coordinate of point $\mathbf{p}_i^s$ after warped by $\mathbf{f}_i$	$\mathbb{R}^3$	$\mathcal{L}$	Total loss value	$\mathbb{R}$
$\mathcal{L}_{ego}$	Ego-motion loss value	$\mathbb{R}$	$\mathcal{L}_{seg}$	Motion segmentation loss value	$\mathbb{R}$
$\mathcal{L}_{scene}$	Scene flow loss value	$\mathbb{R}$	$\mathbf{E}$	Base backbone features	$\mathbb{R}^{N\times C_e}$
C_e	Number of channels of the backbone features	$\mathbb{R}$	$\hat{\mathbf{F}}^{init}$	Initial scene flow estimation	$\mathbb{R}^{N\times3}$
$\hat{\mathbf{f}}_i^{init}$	Initial scene flow vector estimation of point $\mathbf{p}_i^s$	$\mathbb{R}^3$	$\hat{\mathbf{S}}$	Estimated moving probabilities	$\mathbb{R}^N$
$\hat{s}_i$	Estimated moving probability of point $\mathbf{p}_i^s$	$\mathbb{R}$	$\hat{\mathbf{T}}$	Estimated rigid transformation	$\mathbb{R}^{4\times4}$
$\mathbf{I}_4$	Identity matrix	$\mathbb{R}^{4\times4}$	$\hat{\mathbf{F}}$	Final scene flow estimation	$\mathbb{R}^{N\times3}$
$\hat{\mathbf{f}}_i$	Final scene flow vector estimation of point $\mathbf{p}_i^s$	$\mathbb{R}^3$	T	The length of mini-clips used for temporal update	$\mathbb{R}$
$\mathbf{O}$	Ground truth ego-motion transformation	$\mathbb{R}^{4\times4}$	$\mathbf{T}$	Ground truth rigid transformation	$\mathbb{R}^{4\times4}$
$\mathbf{F}^r$	Ground truth rigid flow components	$\mathbb{R}^{N\times3}$	$\mathbf{f}_i^r$	Ground truth rigid flow component of point $\mathbf{p}_i^s$	$\mathbb{R}^3$
v_i	Radar RRV measurement of point $\mathbf{p}_i^s$	$\mathbb{R}$	v_i^r	RRV component ascribed to the radar ego-motion	$\mathbb{R}$
$\mathbf{u}_i$	The unit radial vector of the point $\mathbf{p}_i^s$	$\mathbb{R}^3$	Δt	Time duration between two frames	$\mathbb{R}$
Δv_i	Absolute radial velocity of the point $\mathbf{p}_i^s$	$\mathbb{R}$	$\mathbf{S}^v$	Pseudo motion segmentation labels from RRV	$\mathbb{R}^N$
s_i^v	Pseudo motion segmentation label for $\mathbf{p}_i^s$ from RRV	$\mathbb{R}$	$\mathbf{S}^{fg}$	Pseudo foreground segmentation labels from LiDAR	$\mathbb{R}^N$
$\mathbf{F}^{fg}$	Pseudo scene flow labels from LiDAR	$\mathbb{R}^{N\times3}$	$\mathbf{f}_i^{fg}$	Pseudo scene flow label for $\mathbf{p}_i^s$ from LiDAR	$\mathbb{R}^3$
$\mathbf{S}^l$	Pseudo motion segmentation labels from LiDAR	$\mathbb{R}^N$	s_i^l	Pseudo motion segmentation label for $\mathbf{p}_i^s$ from LiDAR	$\mathbb{R}$
$\mathbf{S}$	Final pseudo motion segmentation labels	$\mathbb{R}^N$	s_i	Final motion segmentation label for $\mathbf{p}_i^s$	$\mathbb{R}$
$\mathcal{L}_{mot}$	MOT-based scene flow loss value	$\mathbb{R}$	$\mathcal{L}_{opt}$	Optical flow-based scene flow loss value	$\mathbb{R}$
$\mathcal{L}_{self}$	Self-supervised scene flow loss value	$\mathbb{R}$	$\mathbf{W}$	Pseudo optical flow labels	$\mathbb{R}^{N\times2}$
$\mathbf{w}_i$	Pseudo optical flow vector label for point $\mathbf{p}_i^s$	$\mathbb{R}^2$	$\mathbf{m}_i$	Corresponding image pixel of point $\mathbf{p}_i^s$	$\mathbb{R}^2$
λ_{opt}	Weighting parameter for $\mathcal{L}_{opt}$	$\mathbb{R}$	θ	Sensor calibration parameters	-
$\mathcal{D}(\cdot,\cdot,\cdot)$	Point-to-ray distance computing function	-	$\|\|\cdot\|\|_2$	L_2-norm, also known as Euclidean Distance	-

Table B. A list of all used symbols with their meanings and dimension in our approach.

The optimal rotation matrix $\hat{\mathbf{R}}^*$ is solved by:

$$\hat{\mathbf{R}}^* = \mathbf{V}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & d\end{bmatrix}\mathbf{U}, \tag{6}$$

where the singular value decomposition (SVD) is used in $\mathbf{H}_w = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$ and $d = sign(det(\mathbf{V}\mathbf{U}^\top))$ is the correction value to avoid special reflection cases. As the final step, we compute the optimal translation vector as:

$$\hat{\mathbf{t}}^* = \bar{\mathbf{c}}^w - \hat{\mathbf{R}}^*\bar{\mathbf{c}}^s. \tag{7}$$

Temporal update module. This module is embedded in our backbone module and its use in our method is optional. If enabled, it can propagate information from previous frames to the current frame. As a reminder, before we generate our base backbone feature $\mathbf{E}$, we get the local features $\mathbf{L} \in \mathbb{R}^{N\times256}$ with another multi-scale *set conv* layer. The global feature vector is obtained by $\mathbf{G} \in \mathbb{R}^{256} = MAX(\mathbf{L})$, where MAX is the max-pooling operation along the channel axis. Our temporal update module operates on $\mathbf{G}$ and applies a GRU network [4] to update it as a hidden state temporally, as seen in Fig. A.

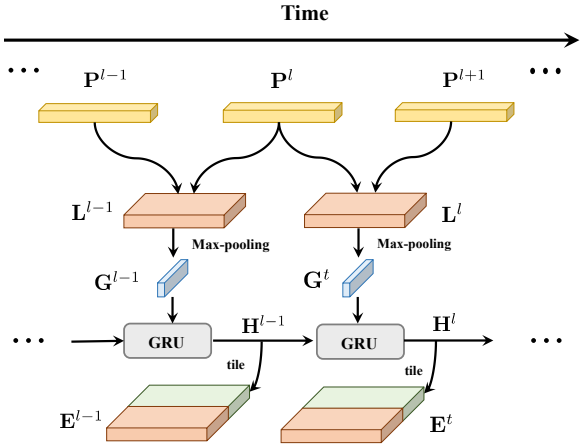

Figure A. Temporal update module. We concatenate the updated hidden state $\mathbf{H}$ to each point feature when activating this module, while we use the global feature vector $\mathbf{G}$ directly when the updating module is disabled. Note that the superscript (*e.g.*, l) denotes the temporal order index and we discard it in other places.

D. Cross-Modal Supervision Details

In this section, we provide some implementation details on cross-modal supervision retrieving and loss formulation.

Strategy for generate pseudo label $\mathbf{S}^v$. As introduced in the main text, we propose to generate a pseudo motion segmentation label $\mathbf{S}^v$ with the odometry information and radar RRV measurements. As a reminder, the pseudo label $\mathbf{S}^v$ is created by thresholding the ego-motion compensated RRV Δv_i, *i.e.*, the absolute radial velocity. Next, we will discuss our specific thresholding strategy.

Given per-point Δv_i, it is intuitive to generate a motion segmentation mask by labelling points with larger values than a fixed threshold as moving points. However, this straightforward strategy suffers from the temporal offset between the well-curated keyframes and original radar point clouds. Even though the coordinates of radar points are transformed to the timestamp of keyframes using high-frequency odometry information, their RRV measurements still correspond to the original timestamps. As a result, there is often a non-trivial global bias in Δv_i, which disturbs the identification of real moving points given a fixed threshold. For example, almost all points could be classified as moving when a large bias exists. To mitigate this issue, we propose to *normalize* all Δv_i by subtracting their mean value μ and then classify points by thresholding. The pseudo motion segmentation label $\mathbf{S}^v = \{s_i^v \in \{0, 1\}\}_{i=1}^N$ can then be created with a fixed threshold η_v, where 0 represents stationary points. We determine the value of η_v via grid searching on the *Val* set, as seen in Fig. B.

To validate if our bias-aware strategy helps to improve the quality of pseudo motion segmentation labels $\mathbf{S}^v$, we compare this advanced strategy to the direct (*i.e.* not bias-aware) one. As the pseudo motion segmentation label generation is affected by the threshold η_v, we iteratively change its value and observe how the mIoU between the ground truth and pseudo motion segmentation labels varies with the threshold. Fig. B shows the comparison between direct thresholding and bias-aware thresholding on the *Val* set. Without considering the global bias of residuals, direct thresholding can only give a maximum mIoU of 48.4%, while our proposed bias-aware thresholding produces a mIoU of 52%[1]. The results demonstrate the effectiveness of our advanced strategy to tackle the temporal misalignment of ego-motion and RRV measurements. Moreover, as shown in the top of Fig. B, the mIoU increases to near 50% only when the threshold is large enough ($>$ 1.0m/s), which further proves the presence of a global bias in Δv_i. For the best quality of pseudo label $\mathbf{S}^v$, we set $\eta_v = 0.3$ in all our experiments according to the results on the *Val* set exhibited in Fig. B.

LiDAR multi-object tracking. As another active ranging sensor, LiDAR can detect targets and measure their 3D positions by emitting pulsed lasers [11]. Compared with 4D radar, it can restore the geometric structure well with denser

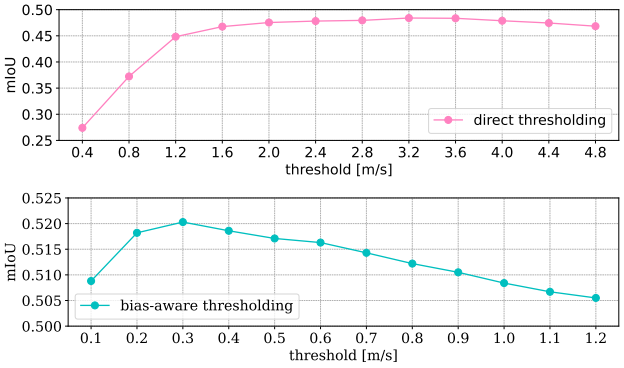

Figure B. The impact of threshold η_v when applying direct and bias-aware thresholding to generate pseudo motion segmentation labels $\mathbf{S}^v$ with the ego-motion and RRV measurements. Here, mIoU is computed between the ground truth motion segmentation labels and the generated pseudo one on the *Val* set.

point data under satisfactory weather conditions. Some recent works [20, 21, 23] perform 3D multi-object tracking (MOT) on LiDAR point clouds in a *tracking-by-detection* paradigm and show prominent results. Inspired by the progress, we propose to extract supervision signals from LiDAR by running 3D MOT algorithms on LiDAR point clouds. The method we employ is called AB3DMOT [20], which first detects 3D bounding boxes with a pretrained PointRCNN [15] model and tracks objects based on a 3D Kalman filter [8]. To implement this method, we first use the OpenPCDet[2] library and adopt the pretrained PointRCNN model to perform the object detection stage. The x, y, z range is set to [(0, 70.4), (-40, 40), (-3, 1)] meters respectively and the number of points is set as 4096 following the original paper [15]. Similar to the authors of the VoD dataset [14], we only detect three classes of object, *i.e.* *pedestrian, cyclist* and *car*, as foreground objects. For the tracking part, we use the official code[3] and take the detection results as input. Specifically, we set the minimal birth length Bir_{min} as 4 and the maximum death length Age_{max} as 8. The centre distance threshold for object association is 2 meters. As a result, the 3D MOT algorithm can give us a set of estimated bounding boxes and their track IDs for each frame.

Optical flow loss $\mathcal{L}_{opt}$. Different from LiDAR or radar point clouds, RGB images contain explicit object appearance and texture information, which provide rich semantics cues for inferring pixel-level 2D motions between frames, *i.e.*, optical flow estimation. As a 2D perspective projection of 3D scene flow, optical flow describes the motion of points on the image plane. Thus it can also be used to partially supervise the scene flow estimation of points, whose perspective projection is within the camera field-of-

[1]Note that tangentially moving targets are not distinguished in the pseudo label $\mathbf{S}^v$ because radar only measures the relative radial velocity.

[2]https://github.com/open-mmlab/OpenPCDet

[3]https://github.com/xinshuoweng/AB3DMOT

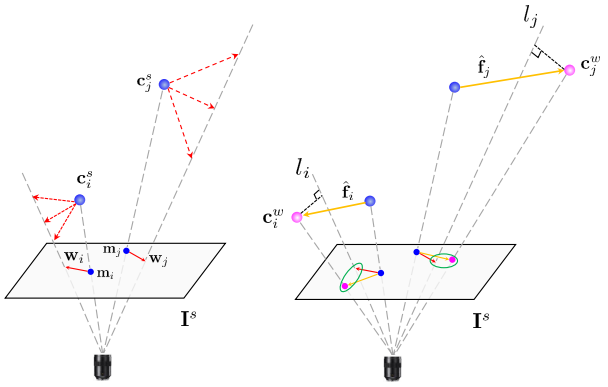

Figure C. Left figure: Blue denotes two points (*i.e.*, $\mathbf{c}_i^s$, $\mathbf{c}_j^s$) from the source point cloud $\mathbf{P}^s$ and their perspective projection $\mathbf{m}_i, \mathbf{m}_j$ on the corresponding image $\mathbf{I}^s$. Red arrows denote pseudo optical flow labels $\mathbf{w}_i, \mathbf{w}_j$ and their possible 3D projections (dashing line). Right figure: Yellow denotes the predicted scene flow vectors $\hat{\mathbf{f}}_i, \hat{\mathbf{f}}_j$ and their corresponding optical flow after perspective projection. Magenta denotes warped points $\mathbf{c}_i^w, \mathbf{c}_j^w$ and their perspective projection on image $\mathbf{I_s}$. l_i and l_j denote the corresponding rays of pixels warped by pseudo optical flow labels.

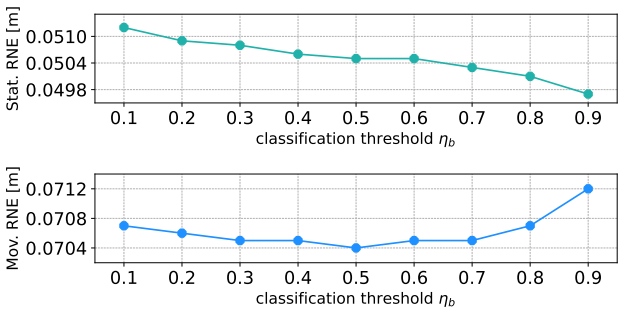

Figure D. Impact of the classification threshold η_b on the *Val* set. The value is changed from 0.1 to 0.9 by a step of 0.1.

T	EPE [m]↓	AccS↑	AccR↑	RNE [m]↓
1	0.132	0.263	0.552	0.053
3	0.128	0.280	0.560	0.051
5	**0.122**	**0.295**	**0.579**	**0.049**
7	0.131	0.264	0.560	0.053
10	0.129	0.260	0.551	0.052
15	0.142	0.203	0.488	0.057

Table C. Impact of the mini-clip length for temporal feature update. The results shown above are obtained by evaluating our CMFlow (T) method on the *Val* set. The best T (**bold**) are used for experiments on the *Test* set.

view (FoV). Thanks to the advance in data-driven optical flow estimation [16, 17], we can use a pretrained model to infer pseudo optical flow labels efficiently. For one pair of $\mathbf{P}^s$ and $\mathbf{P}^t$, we can input their synchronized monocular images $\mathbf{I}^s$ and $\mathbf{I}^t$ into the network and obtain an optical flow map. We adopt the RAFT-S [17] pretrained model[4] to estimate our optical flow. The number of iterations is set as 12 in practice. After drawing per-point optical flow vector $\mathbf{W} = \{\mathbf{w}_i \in \mathbb{R}^2\}_{i=1}^N$, we formulate our optical flow loss to constrain our scene flow prediction in the perspective view. In the main text, due to the depth-unawareness during perspective projection, we propose to minimize the point-to-ray distance instead of the flow divergence in pixel scale. Our motivation is further illustrated in Fig. C, where a larger distance between the warped point $\mathbf{c}_j^w$ and the expected ray l_j results in a smaller pixel-scale loss after perspective projection for a farther point $\mathbf{c}_j^s$. In practice, we discard point-to-ray distances lower than 0.25m when computing the loss to mitigate the impact of optical flow estimation errors.

Self-supervised loss. Motivated by self-supervised scene flow works [1, 5, 10, 13, 22], we also utilize self-supervised loss functions to complement our cross-modal supervised learning. Specifically, we leverage three self-supervised loss functions from [5] and keep their default hyperparameter settings. The first one is called soft Chamfer loss, which constrains scene flow by pulling mutual pseudo correspondences between the target point cloud $\mathbf{P}^t$ and the warped source point cloud $\mathbf{P}^w$ close to each other. In practice, the noise from outliers is mitigated and small errors led by low-resolution are discarded, to make the loss *soft* to intractable radar point clouds. The second loss is spatial smoothness loss, which is used for local consistency in scene flow estimation. In particular, a distance-based weight is computed at first and then the spatial smoothness of the predicted scene flow is enforced accordingly. The third loss is called radial displacement loss, where the radial projections of estimated scene flow are constrained by the RRV measurements.

[4] https://github.com/princeton-vl/RAFT

E. More Experimental Analysis

Impact of the classification threshold η_b. As an important hyperparameter, η_b is used to threshold the estimated moving probabilities $\hat{\mathbf{S}}$ to generate the binary motion segmentation mask during inference, which further determines the flow vectors of which points are refined with the ego-motion estimation. To select its optimal value, we run a series of evaluations on the *Val* set with the trained CMFlow model and show the results in Fig. D. As we increase the threshold value, the performance on Stat. RNE gets improved continuously, however the optimal threshold for Mov. RNE is 0.5. Considering that the scene flow accuracy on both moving and static points is crucial, we aim to seek a trade-off between Stat. RNE and Mov. RNE. We thus set the $\eta_b = 0.5$ as the optimal value for our experiments.

Impact of the mini-clip length T. When activating our

Method	EPE [m]↓	AccS↑	AccR↑	RNE [m]↓
FlowNet3D [12]	0.201	0.169	0.379	0.081
PointPWC-Net [22]	0.196	0.177	0.391	0.079
FlowStep3D [9]	0.286	0.061	0.185	0.115
PV-RAFT [19]	**0.126**	**0.258**	**0.587**	**0.051**
3DFlow [18]	0.277	0.104	0.294	0.111
Bi-PointFlowNet [3]	0.242	0.164	0.350	0.097

Table D. Comparison between fully-supervised radar scene flow methods on the *Test* set. All methods are trained using the same annotated training samples with ground truth scene flow.

temporal update module in the backbone, we can update the global feature vector temporally to propagate information from previous frames. However, training with long sequences in a brute-force way will lead to non-negligible over-fitting. To mitigate the issue, we split long training sequences into many mini-clips with a length of T and train mini-batches of them. During inference, the hidden state is re-initialized after T frames to emulate the training conditions. Here, we exhibit how to select the optimal T value in our experiments, as seen in Tab. C. As we increase the value of T, the performance on the *Val* set improves because temporal information from more previous frames can be used for the current frame. However, when the value of T exceeds 5, the performance starts to degrade and becomes even worse than the one without temporal update when $T = 15$. We attribute this to two reasons. First, as we discuss above, longer clips can exacerbate the over-fitting issue. Second, using long-term information from earlier frames will introduce more noise and disturb the current feature extraction.

Performance of fully-supervised methods. In our main experiments, we are interested in if our performance could catch up or surpass that of fully-supervised methods when more unannotated data is available for training. As a reminder, we select the state-of-the-art fully-supervised method, PV-RAFT [19], for comparison. Here for completeness, we also evaluate another five fully-supervised methods and compare them together with PV-RAFT [19] in Tab. D. Note that we use all available annotated samples in *Train* set and label their ground truth scene flow with object annotations (c.f. Sec. A) for fully-supervised training. Under the same training setting, PV-RAFT [19] achieves the best results among fully-supervised methods. This further proves its state-of-the-art performance and supports our choice of PV-RAFT as the compared fully-supervised method when more unannotated training data is available.

Impact of self-supervised loss. As a complement to our cross-modal supervision, the self-supervised loss can exploit the inherent spatio-temporal relationship and constraints in the input data to bootstrap scene flow learning. Here, we analyse its impact by ablating it from our method.

Method	EPE [m]↓	AccS↑	AccR↑	RNE [m]↓
w.o. $\mathcal{L}_{self}$	0.152	0.221	0.494	0.061
w. $\mathcal{L}_{self}$	**0.141**	**0.233**	**0.499**	**0.057**
Gain	-0.011	+0.012	+0.005	-0.004

Table E. Analysis of the impact of self-supervise loss. For the top row, we ablate the self-supervised loss during training.

	L (seg)	L (flow)	EPE [m]↓	AccS↑	AccR↑	RNE [m]↓
(a)			0.159	0.216	0.458	0.064
(b)	✓		0.156	0.221	0.467	0.063
(c)		✓	0.152	0.217	0.477	0.061
(d)	✓	✓	**0.141**	**0.223**	**0.499**	**0.057**

Table F. Ablation study for LiDAR modality. L (seg) denotes that the LiDAR supervision is used for generating pseudo motion segmentation labels, while L (flow) denotes that the LiDAR supervision is used to formulate the scene flow loss.

The results are shown in Tab. E. With the self-supervised loss, the performance of CMFlow improves in all metrics. This demonstrates that combining self-supervised and cross-modal supervision provides complementary learning cues and leads to more accurate scene flow estimation.

Impact of LiDAR modality. In our cross-modal supervised learning pipeline, LiDAR supervision is used in two ways to support our model training. One is to generate a pseudo motion segmentation label $\mathbf{S}^l$, which is further used to obtain a more reliable label $\mathbf{S}$ after fusing with $\mathbf{S}^v$. Another is to provide the pseudo scene flow label $\mathbf{F}^{fg}$ used to formulate the loss $\mathcal{L}_{mot}$ that supervises the final scene flow. Here, we analyze the impact of LiDAR modality in these two ways. As seen in Tab. F, the LiDAR modality contributes to our improvement gain in both aspects. First (c.f. row (b)), using the pseudo label $\mathbf{S}^l$ can effectively complement the label $\mathbf{S}^v$ where tangentially moving targets are not distinguished. Second (c.f. row (c)), the scene flow loss $\mathcal{L}_{mot}$ can constrain the scene flow vectors of identified moving points in $\mathbf{S}^l$. When using the LiDAR modality in both ways, an even larger improvement can be achieved. This demonstrates that each supervision component in our framework is significant and correlates to each other compactly.

Runtime efficiency. We evaluate the computational efficiency of CMFlow on a PC with a single RTX 3090 GPU. To emulate real-world processing, one sample is fed into our model per time. Our method contains 4.23 million model parameters and performs one inference step in 0.069 seconds on average (∼14 Hz). The maximum allocated GPU memory is 162 MB during inference. It can be seen that our method can achieve satisfactory real-time performance while having relatively low GPU memory consumption.

F. More Qualitative Results

Scene flow estimation. More qualitative scene flow results are shown in Fig. E. It can be seen that CMFlow can accurately estimate scene flow vectors in diverse driving scenarios. After warping the source point cloud with the estimated scene flow, both static background and multiple dynamic objects can be aligned well between two frames.

Motion segmentation. Additional qualitative results for the motion segmentation task can be seen in Fig. F. Without any ground truth labels used for training, our method results in accurate motion segmentation in dynamic environments. Different moving objects (*e.g.*, car, cyclist, pedestrian) can be segmented from the static background well.

Ego-motion estimation. We show more qualitative odometry results in Fig. G. As a byproduct of our method, the estimated rigid ego-motion transformation can effectively support the odometry task in complex urban driving scenarios. Compared to the baseline ICP [2] that directly solves the transformation with all points, CMFlow can distinguish and mitigate the impact of moving points, and thus has better odometry performance in dynamic environments.

References

[1] 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 *CVPR*, pages 13126–13136, 2021. 1, 5

[2] P.J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *PAMI*, 14(2):239–256, 1992. 7

[3] Wencan Cheng and Jong Hwan Ko. Bi-PointFlowNet: Bidirectional Learning for Point Cloud Based Scene Flow Estimation. In *ECCV*, pages 108–124, 2022. 6

[4] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *SSST*, pages 103–111, 2014. 3

[5] Fangqiang Ding, Zhijun Pan, Yimin Deng, Jianning Deng, and Chris Xiaoxuan Lu. Self-Supervised Scene Flow Estimation With 4-D Automotive Radar. *RA-L*, pages 1–8, 2022. 5

[6] Philipp Jund, Chris Sweeney, Nichola Abdo, Zhifeng Chen, and Jonathon Shlens. Scalable scene flow from point clouds in the real world. *RA-L*, 7(2):1589–1596, 2021. 1

[7] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallogr. A*, 32(5):922–923, 1976. 2

[8] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *J. Basic Eng.*, 82(1):35, 1960. 4

[9] Yair Kittenplon, Yonina C Eldar, and Dan Raviv. FlowStep3D: Model Unrolling for Self-Supervised Scene Flow Estimation. In *CVPR*, pages 4114–4123, 2021. 6

[10] Ruibo Li, Guosheng Lin, and Lihua Xie. Self-Point-Flow: Self-Supervised Scene Flow Estimation from Point Clouds with Optimal Transport and Random Walk. In *CVPR*, pages 15577–15586, 2021. 5

[11] You Li and Javier Ibanez-Guzman. Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. *IEEE SPM*, 37(4):50–61, 2020. 4

[12] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. FlowNet3D: Learning Scene Flow in 3D Point Clouds. In *CVPR*, pages 529–537, 2019. 2, 6

[13] Himangi Mittal, Brian Okorn, and David Held. Just go with the flow: Self-supervised scene flow estimation. In *CVPR*, pages 11177–11185, 2020. 5

[14] Andras Palffy, Ewoud Pool, Srimannarayana Baratam, Julian FP Kooij, and Dariu M Gavrila. Multi-class Road User Detection with 3+ 1D Radar in the View-of-Delft Dataset. *RA-L*, 7(2):4961–4968, 2022. 1, 4

[15] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud. In *CVPR*, pages 770–779, 2019. 4

[16] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *CVPR*, pages 8934–8943, 2018. 5

[17] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *ECCV*, pages 402–419, 2020. 5

[18] Guangming Wang, Yunzhe Hu, Zhe Liu, Yiyang Zhou, Masayoshi Tomizuka, Wei Zhan, and Hesheng Wang. What Matters for 3D Scene Flow Network. In *ECCV*, pages 38–55, 2022. 6

[19] Yi Wei, Ziyi Wang, Yongming Rao, Jiwen Lu, and Jie Zhou. PV-RAFT: point-voxel correlation fields for scene flow estimation of point clouds. In *CVPR*, pages 6954–6963, 2021. 6

[20] Xinshuo Weng, Jianren Wang, David Held, and Kris Kitani. 3D Multi-Object Tracking: A Baseline and New Evaluation Metrics. In *IROS*, pages 10359–10366, 2020. 4

[21] Xinshuo Weng, Yongxin Wang, Yunze Man, and Kris M Kitani. GNN3DMOT: Graph Neural Network for 3D Multi-Object Tracking With 2D-3D Multi-Feature Learning. In *CVPR*, pages 6499–6508, 2020. 4

[22] 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*, pages 88–107, 2020. 2, 5, 6

[23] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *CVPR*, pages 11784–11793, 2021. 4

Figure E. Qualitative scene flow results in seven *Test* scenes. From left to right: 1) radar points from the source frame projected to the corresponding RGB image (points are coloured by distance from the sensor), 2) two input radar point clouds, the source one (pink) and the target one (green), 3) the source point cloud warped by our predicted scene flow and the target radar point cloud, 4) the source point cloud warped by ground truth scene flow and the target one. We mark regions of interest in amber and apply the zooming-in operation for them.

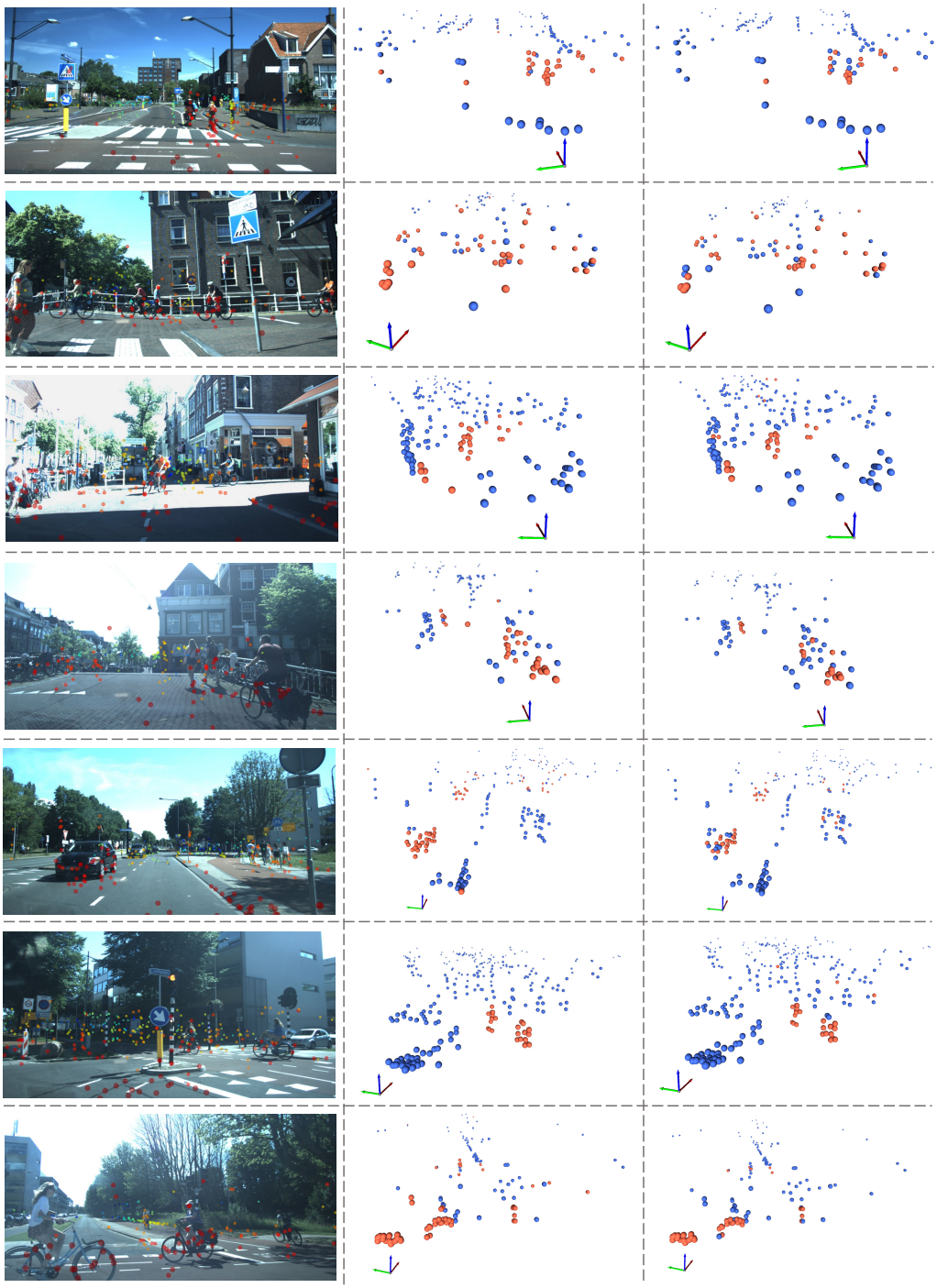

Figure F. Visualization of motion segmentation. The left column shows the corresponding image with radar points (coloured by range) projected onto it. In the middle and right columns, moving points are shown in orange while static points are shown in blue.

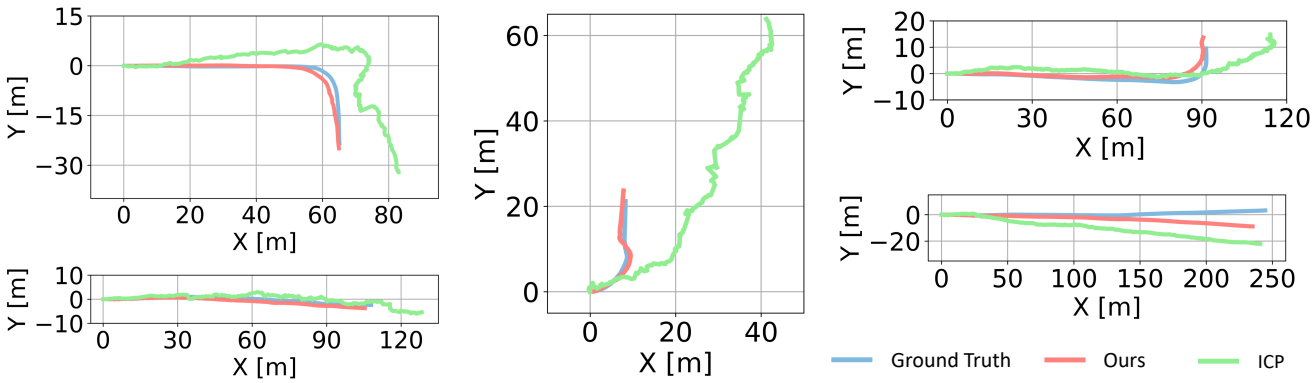

Figure G. Qualitative odometry results in five *Test* sequences. To plot the ego-vehicle trajectory, inter-frame ego-motion transformations are accumulated temporally. Please see the supplementary video for dynamic trajectory results.