Rigid Scene Flow Estimation and Prediction on Temporal LiDAR for Autonomous Driving

David Deng

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-228
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-228.html

September 19, 2022

Rigid Scene Flow Estimation and Prediction on Temporal LiDAR for Autonomous Driving

by

David Deng

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Enginnering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Avideh Zakhor, Chair
Professor Angjoo Kanazawa

Summer 2022

The dissertation of David Deng, titled Rigid Scene Flow Estimation and Prediction on Temporal LiDAR for Autonomous Driving, is approved:

Chair	Andrea Bajcsy	Date	8/29/2022
		Date	9/13/2022
		Date	

University of California, Berkeley

Rigid Scene Flow Estimation and Prediction on Temporal LiDAR for Autonomous Driving

Abstract

Rigid Scene Flow Estimation and Prediction on Temporal LiDAR for Autonomous Driving

by

David Deng

Master of Science in Electrical Enginnering and Computer Science

University of California, Berkeley

Professor Avideh Zakhor, Chair

With the prevalence of LiDAR and depth sensors, 3D point clouds have been in increasingly prevalent form of visual data, especially in the field of autonomous driving. As such, there is a need for algorithms that can process raw, unlabelled sequences of point clouds. In this report, we present two such methods for processing temporal LiDAR data: 1. a method for multi-body rigid scene flow and segmentation, and 2. a novel neural network for 3D point cloud prediction. Our prediction network is based on FlowNet3D and trained to minimize the Chamfer Distance (CD) and Earth Mover's Distance (EMD) to the next point cloud. Compared to directly using state of the art existing methods such as FlowNet3D, our proposed architectures achieve CD and EMD nearly an order of magnitude lower on the nuScenes dataset. In addition, we show that our predictions generate reasonable scene flow approximations without using any labelled supervision. Our second work exploits the multi-body rigidity present in dynamic scenes encountered in autonomous driving by parameterizing scene flow as the composition a global ego-motion and a set of bounding boxes associated with their own rigid motions. We construct a novel loss function and differentiable bounding box formulation to optimize these parameters. Our approach achieves state of the art accuracy on the KITTI Scene Flow benchmark, outperforming all previous approaches without using any annotated labels. Additionally, we demonstrate the effectiveness of our approach on motion segmentation and ego-motion estimation and produce visualizations of our predictions to corroborate our results.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank Professor Zakhor for her guidance and direction these past three years. Additionally, I would like to thank Scott McCrae, Ajay Gopi, Wendi Li, and Jordan Grelling for their contributions to various parts of these projects. Lastly, I would like to thank all my friends and family for their support and encouragement throughout my research endeavors.

Chapter 1

Introduction

Understanding the 3D structure and dynamics of a scene is an essential problem in autonomous driving. Consequently, LiDAR and point cloud processing have become a critical component of autonomous driving systems. In recent years, there has been extensive research towards developing algorithms that process individual frames of LiDAR, but only recently has more work been done towards processing sequences of LiDAR and understanding them temporally. Additionally, the increasing prevalence of LiDAR and depth sensors means an increase in the abundance of raw, unlabelled point cloud data and the need for algorithms to process them. While supervised learning for 3D point clouds have been studied extensively, researchers have only just begun to explore self-supervised point cloud processing. In this report, we present two methods for processing unlabelled, temporal LiDAR data: 1. a novel method for multi-body rigid scene flow between two successive LiDAR frames, and 2. a neural network architecture for 3D point cloud prediction.

1.1 Rigid Scene Flow Optimization

3D scene flow is a low-level motion representation, characterized as a 3D motion field over all points in the scene. With the increasing prevalence of LiDAR and depth sensors, scene flow estimation from point clouds has become an increasingly important problem. Advances in deep feature learning on point sets [50, 51, 10, 77, 30] have recently given rise to deep learning approaches to scene flow estimation [35, 66, 17, 69, 76, 16, 70, 46, 3, 45]. However, because real-world scene flow annotations are prohibitively expensive to acquire, many of these approaches depend on labels generated from synthetic datasets [42] and often do not generalize well to real world data, especially LiDAR scans. Additionally, most of these works predict scene flow fail to exploit the rigidity present in most scenes *i.e.* the dynamics of most scenes can be decomposed into the motion of multiple rigidly moving objects such as cars and cyclists. They predict unconstrained, pointwise motion vectors, resulting in inaccurate and physically inconsistent predictions.

In light of these shortcomings, our first project is a novel *object-level* scene flow estima-

tion approach that jointly optimizes a global ego-motion and a set of bounding boxes with their own rigid motions, without using any annotated labels. To optimize this new scene flow parameterization using gradient methods, we develop a novel loss function and a differentiable bounding box formulation. We demonstrate the effectiveness of our approach in several autonomous driving settings. In particular, our method outperforms the state of the art on the KITTI Scene Flow dataset by a large margin, achieving 2x lower end-point-error than the current state of the art.

In summary, our main contributions are that:

- We propose a novel objective function to optimize object-level rigid scene flow without any annotated labels.
- We develop a differentiable 3D bounding box formulation to optimize our scene flow parameters.
- Our method produces physically plausible and interpretable scene flow by constraining the predicted motion to be rigid.
- Our approach accurately detects moving objects at an instance level without labeled supervision.
- Our approach significantly outperforms the state of the art approaches on the KITTI Scene Flow benchmark.

1.2 Point Cloud Prediction

Several autonomous driving companies such as nuTonomy, Waymo, and Lyft have recently released large scale datasets [8, 56, 21] with high resolution LiDAR point clouds and numerous annotations. Perhaps more interestingly, they provide high frequency temporal LiDAR frames, or LiDAR "videos", opening doors to the use of temporal information in LiDAR point cloud analysis. Our first project aims to take advantage of the temporal data offered by the newly released datasets and tackles the point cloud prediction task using deep learning methods, that is, given a sequence of past point cloud frames, to predict future frames. While this task can easily be applied to other contexts, we implement and evaluate our models specifically in the context of autonomous driving and LiDAR. Much of the autonomous driving problem has to do with helping cars anticipate events and avoid collisions. To this end, predicted point clouds could be used to enhance object tracking pipelines, or to generate preliminary region proposals for future detections. Furthermore, no data labelling is required for this task; the supervision signal is the next frame of the LiDAR sequence, so our networks can be trained in a self-supervised fashion.

Extracting temporal information from point clouds is a difficult task and little work has been done in this area. Conventional architectures for sequential data such as LSTMs cannot be applied directly to point clouds due to their irregular structure. Voxelizing the

point cloud could work around this, but it reduces the resolution of the point cloud and introduces quantization artifacts into the prediction. We explore a number of end to end architectures that operate directly on point clouds to extract temporal features and predict future frames. In particular, we design a general architecture framework and implement and evaluate several of its variations. Through experiments we evaluate the performance and complexity of our models and assess the tradeoffs of various architectural design choices.

The outline of this report is as follows: in Chapter 2 we discuss relevant prior work for these two projects, in Chapter 3 and 4 we discuss the methods, experiments, results, and visualizations of our scene flow and prediction approach, and in Chapter 5 we summarize our findings and discuss the implications of our work.

Chapter 2

Related Work

2.1 Deep Learning on Point Sets

2.1.1 Voxel Based Approaches

Early attempts at learning on point clouds involved converting them to voxels and using CNN backbones to extract features. Works such as [41, 77] are able to achieve reasonable results on classification and detection. Voxel based approaches continue to be popular in the autonomous driving domain, with architectures such as [30], [71], [74] achieving strong accuracy with real time inference. Additionally, sparse convolution libraries such as Minkowski Engine [10] that allow for efficient processing of voxelized point clouds have recently become popular, general-purpose feature extractors.

2.1.2 Non-Voxel Based Approaches

In 2017, Qi et al. proposed PointNet [50], the first deep learning architecture that operated directly on point clouds by utilizing symmetric functions such as max pooling to make the network invariant to input permutations. Later works [51, 57, 34] would take advantage of the local structures in point clouds to learn local feature descriptors. Dynamic Graph CNN (DGCNN) [65] is a special case among these in that while most architectures group points and learn features based on Euclidean distance, DGCNN dynamically groups them based on their distance in the feature space, allowing greater expressive power and more useful local features. We take advantage of these properties in our proposed architectures for point cloud prediction.

2.1.3 Deep Temporal Learning on Point Sets

2.1.3.1 Temporal Learning with Neural Networks

Recurrent networks are well studied and known for their application on sequential data. In the past decade they have been applied to the task of video prediction [39], which is conceptually similar to our task. However, recently feedforward networks have been shown to perform just as well, if not better than recurrent networks on sequential data [2]. They do not suffer from vanishing gradients and other training instabilities common in recurrent networks, tend to be faster and more parallelizable, and typically require less memory during training. Similar to recurrent networks, feedforward models have also been applied to temporal video processing and prediction tasks [75]. Inspired by this, we choose to use feedforward networks for our point cloud prediction architectures.

2.1.3.2 Deep Temporal Learning on Point Sets

[52] is among the first papers applying deep learning to temporal point cloud, doing so in the context of object detection for LiDAR in autonomous driving. The first general framework for deep, temporal point cloud processing is [36], which introduced a PointNet++ style architecture to group points temporally as well. [12] and [13] extend [20] and [60] to the point cloud domain. Lastly, [68] is the first work tackling the point cloud prediction task applying it to trajectory forecasting in autonomous driving.

2.2 Scene Flow Estimation

2.2.1 Supervised Scene Flow

The term scene flow was first introduced in [61]. Traditional methods primarily predicted scene flow from stereo and RGB-D [22, 67, 27, 18, 53, 63, 64, 62], although some used LiDAR [59]. Following the rise of deep learning, neural networks became a prevalent tool for scene flow estimation on images [26, 73, 72, 23, 24]. [35] pioneered deep scene flow estimation on point clouds by combining the FlowNet architecture [11, 25] with advances in deep point cloud feature learning [50, 51]. They trained their model in a supervised manner on synthetic data [42] and generalized it to real world data [15]. Subsequent works on supervised scene flow estimation from point clouds followed this training framework. [17] and [69] developed faster and more accurate network architectures using sparse permutohedral lattices and cost volumes respectively. [76] predicted scene flow as a global ego-motion and a set of residual flow vectors. [49] framed scene flow estimation as a correspondence problem by borrowing ideas from optimal transport. [32] utilized high-order CRFs to constrain the scene flow output to be locally smooth and rigid. Lastly, [70] used a recurrent network to iteratively refine scene flow predictions.

2.2.2 Self-Supervised Scene Flow

Because of the domain shift from synthetic to real world datasets, as well as the difficulty of acquiring scene flow annotations for real LiDAR scans, recent works on scene flow estimation from point clouds have explored self-supervision. [45] was the first to do so, utilizing a nearest neighbor and cycle-consistency loss. [55] and [48] both developed loss functions that use a combination of Chamfer Distance, and smoothness and shape constraints. [70, 76, 3] all use loss functions similar to these, but [70] uses a recurrent network architecture, while [76, 3] predict a global ego-motion in addition to flow vectors. None of these approaches exploit the multi-body rigid nature of dynamic scenes.

2.2.2.1 Scene Flow Optimization

Rather than of predicting scene flow in real time, recently a few works have used offline optimization approaches to estimate scene flow. In particular, [48] directly optimizes its self-supervised loss function over each point cloud at inference time, and [33] trains a neural network over a single pair of point clouds to predict scene flow, using the network as an implicit smoothness regularizer. Our scene flow approach to be described in Section 3 falls within this category, but unlike the previous works, we optimize scene flow at the object level.

2.2.3 Object Scene Flow

While multi-body rigidity is a commonly used prior in scene flow estimation, most existing works require some form of annotation. [47, 54, 40, 72] predicted object-level scene flow from stereo, images, or RGB-D. [54, 40, 72] required annotated segmentation labels, and [47] focused on object detection. [46] was the first work to predict object level scene flow from point clouds, but it required scene flow, object detection, and ego-motion labels. More recently, [16] proposed a weakly supervised approach to object-level scene flow estimation that only requires ego-motion and foreground/background labels. All of the aforementioned approaches except [47] require annotated labels at training time. Our scene flow approach is the first to leverage the rigidity of dynamic scenes in scene flow estimation while also requiring no labelled supervision.

Chapter 3

Rigid Scene Flow

In this chapter we detail our work on optimizing rigid scene flow from liDAR without labels. In section 3.1 we discuss the details of our approach, particularly our scene flow parameterization, optimization objective, and inference. In section 4.2 we show and discuss our results and visualizations and conduct an ablation study to validate our design choices.

3.1 Method

The objective of 3D scene flow estimation is, given a pair of point clouds $P_1 \in R^{3\times N_1}, P_2 \in R^{3\times N_2}$, to predict the scene flow between them, defined as a set of vectors $F \in R^{3\times N_1}$ that indicate the motion of the points in P_1 to their corresponding location in P_2. Note that there may not be direct correspondences between the two point clouds.

3.1.1 Motion Representation

Previous works usually predict scene flow directly, parameterized as a set of pointwise motion vectors. However, this parameterization is highly unconstrained and fails to exploit the underlying rigidity present in most scenes. We instead parameterize scene flow as the composition of a global ego-motion and a set of bounding boxes containing moving objects, each with their own rigid motions. Specifically, our objective is to compute a global ego-motion $\mathbf{T}_{ego} = \{\mathbf{R}_{ego} \in SO(3), \mathbf{t}_{ego} \in R^3\} \in SE(3)$; a set of k bounding boxes $B = \{\mathbf{b}_i\}^k$ parameterized as $\mathbf{b} = (c, x, y, z, w, l, h, \theta)$ where c is a confidence score indicating whether or not the box contains a moving object, x, y, z is the center of the bounding box, w, l, h are the box's dimensions, and θ is a heading angle; and a set of k rigid transformations each with the form $\mathbf{T}_i = \{\mathbf{R}_i \in SO(3), \mathbf{t}_i \in R^3\} \in SE(3)$, one for each bounding box. Because we focus specifically on autonomous driving settings, we assume the boxes have zero pitch and roll. Additionally, in practice we find that confining $\mathbf{T}_i$ to $SE(2)$ on the ground plane produces more consistently accurate scene flow predictions.

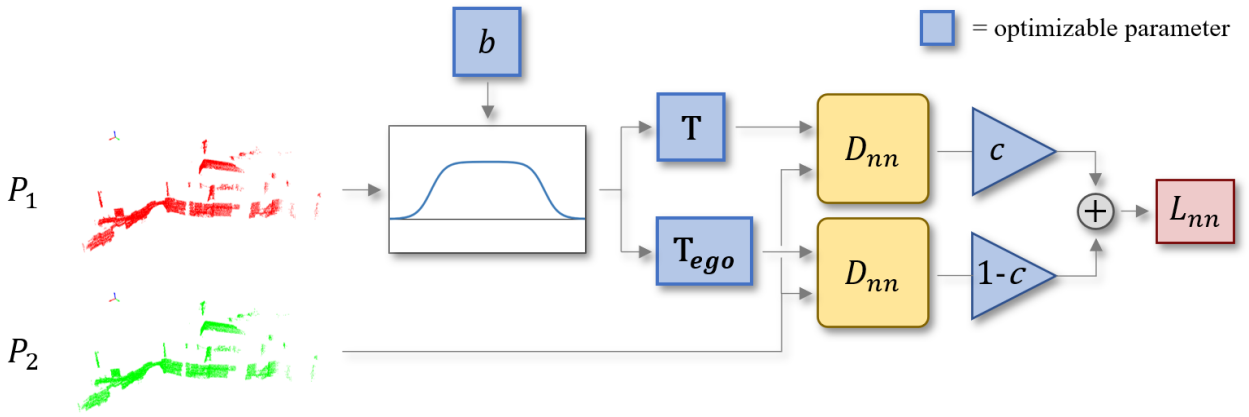
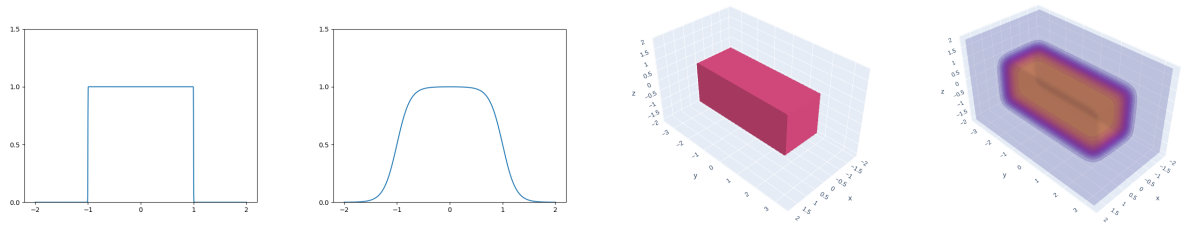


Figure 3.1: Overview of our loss function for a single bounding box. Terms in blue blocks are optimizable scene flow parameters b: bounding box parameters, $\mathbf{T}$: bounding box's rigid transformation, $\mathbf{T}_{ego}$: ego-motion transformation, c: box's confidence score, and the plot refers to our differentiable bounding box approximation. From P_1, we differentiably select the points inside the bounding box and transform them using $\mathbf{T}$ and $\mathbf{T}_{ego}$. Then we compute the nearest neighbor distance between the two transformed point sets and P_2. Lastly, we weight the two nearest neighbor distance by c and $1-c$ respectively and sum them to compute the loss.

We choose to parameterize objects as bounding boxes rather than using segmentation masks as in previous works because, similar to pointwise flow vectors, segmentation masks are highly unconstrained and can represent incoherent objects *e.g.* an object with one point on one side of the scene and another point completely detached from it on the other. On the other hand, bounding boxes constrain objects to physically plausible point sets. In our context of autonomous driving datasets, one can typically convert between the two representations, as objects occupy their own exclusive region in 3D space and do not penetrate each other as they often do in image pixels.

3.1.2 Overview

An overview of our proposed objective function is shown in Figure 3.1. To compute our scene flow parameters without any labelled supervision, we optimize them over a novel loss function that reflects the nearest neighbor distance (NND) from P_1 to P_2 after applying our scene flow parameters. Given a bounding box b_i, we select the points inside it using a differentiable bounding box approximation. We then transform these points using $\mathbf{T}_i$ and $\mathbf{T}_{ego}$ and compute the NND between the two transformed point sets and P_2. We call these terms the foreground and background loss, respectively. Finally, we multiply the foreground and background loss by c_i and $1-c_i$ respectively and add these together to compute the loss total for b_i. The final loss is the sum of all per-box losses. To compute scene flow parameters, we simply minimize this loss function using gradient-based optimization. As a clarifying note, we do not use any neural networks in this approach and are only interested



(a) (b) (c) (d)

Figure 3.2: Visualizations of non-differentiable vs differentiable bounding boxes on 1 and 3 dimensions: (a) non-differentiable 1D bounding line (b) differentiable 1D bounding line (c) non-differentiable 3D bounding box (d) differentiable 3D bounding box.

in directly optimizing the scene flow parameters. In the following sections, we explain each of the components of this loss function in detail.

3.1.3 Differentiable Bounding Boxes

Our proposed loss function takes the NND of transformed points in P_1 to P_2, where pointwise NND is defined as:

$$D_{nn}(P_1, P_2)[i] = \min_{p_2 \in P_2} ||p_2 - P_1[i]||_2^2 \tag{3.1}$$

However, the NND to P_2 of points in P_1 inside a bounding box b_i, written as $NND(b_i(P_1), P_2)$, is a step function with respect to b_i and therefore non-differentiable. In order to optimize b_i over the NND, we propose a novel, differentiable bounding box approximation.

Fundamentally, a bounding box is a membership function in 3D space. Each point along the function is assigned either a 1 or 0 depending on whether it is inside or outside the box. In the simplified case of a 1 dimensional bounding box or "bounding line", the membership function would be the difference of two shifted unit step functions, as shown in Figure 3.2a. To make this function differentiable, we can replace the step functions with sigmoid approximations, shown in Figure 3.2b, resulting in a 1D differentiable bounding box:

$$box_{1D}(x) = \frac{1}{1 + e^{k(x + \frac{w}{2})}} - \frac{1}{1 + e^{k(x - \frac{w}{2})}} \tag{3.2}$$

where w is the width of the box and k is a parameter controlling the sharpness of the sigmoid slope.

To generalize to 3D, we take the product of 1D bounding lines along each of the three axes of the bounding box, resulting in a bounding box membership density field shown in Figure 3.2d. To compute the membership weights of each point, we first transform them into the local coordinate frame of the box. Given the center of the bounding box x, y, z and the heading angle θ, the transformed points can be computed as:

$$\mathbf{p}_{box} = \mathbf{R}_{box}(\mathbf{p} - \mathbf{t}_{box}) \tag{3.3}$$

where $\mathbf{p}$ is the input point, $\mathbf{p}_{box}$ is the transformed point, $\mathbf{t}_{box}$ is the center of the bounding box, and $\mathbf{R}_{box}$ is the following rotation matrix parameterized by the yaw angle of the box, θ:

$$\mathbf{R}_{box} = \begin{bmatrix} -sin(\theta) & cos(\theta) & 0 \\ cos(\theta) & sin(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

The membership weights can then be written as:

$$box_{3D}(x, y, z) = \prod_{d \in \{x_b, y_b, z_b\}} \left(\frac{1}{1 + e^{k(d + \frac{w_d}{2})}} - \frac{1}{1 + e^{k(d - \frac{w_d}{2})}}\right) \tag{3.5}$$

where x_b, y_b, z_b are x, y, z transformed into the local coordinate frame of the bounding box, and w_d is the width of the box along dimension d.

3.1.4 Loss Function

Our proposed loss function is the sum of independently computed per-box losses, so for simplicity we will focus on the loss of a single box b, written as:

$$\begin{aligned} L_{nn} &= c \sum_{i=1}^{N_1} \hat{w}_i * (D_{nn}(\mathbf{R}\mathbf{p}_i + \mathbf{t}, P_2) + \epsilon) \\ &+ (1 - c) \sum_{i=1}^{N_2} \hat{w}_i * D_{nn}(\mathbf{R}_{ego}\mathbf{p}_i + \mathbf{t}_{ego}, P_2) \\ &= cL_{fg} + (1 - c)L_{bg} \end{aligned} \tag{3.6}$$

where c is the confidence score of the box. $\hat{w}_i$ is the i$^{\text{th}}$ element of $\hat{\mathbf{w}}$, a normalized vector of weights produced by the smooth bounding box approximation for b computed using Equation 3.5 over P_1, $\mathbf{p}_i$ is the i$^{\text{th}}$ point in P_1, $\mathbf{R}$ and $\mathbf{t}$ are b's associated rotation and translation, N_1 and N_2 indicated the number of points in P_1 and P_2, and ϵ is a constant penalty, to be described shortly. Lastly, L_{fg} and L_{bg} denote foreground and background loss, respectively.

Intuitively, this loss function minimizes the NND to P_2 after applying the scene flow parameters to to P_1. The inner product of the pointwise NND and $\hat{\mathbf{w}}$ results in a differentiable approximation of the NND of points inside b; both the foreground and background loss can be written in this form. Therefore, the foreground loss describes the NND of points inside the box under the assumption that they belong to a dynamic rigid object. Similarly, the background loss, describes the NND under the assumption that they belong to a static object. We refer to these two assumptions as the foreground and background hypotheses.

In practice, it is unknown whether a box contains a moving or static object, so to model this uncertainty, we write the loss as the weighted sum of the NND under the two hypotheses, weighted by c and $1 - c$, where $c \in (0, 1)$. If the foreground loss is lower than the background

loss, the box likely contains a dynamic object, and to minimize the overall loss, c will tend towards 1. Likewise, if the background loss is lower than the foreground loss, the box probably belongs to the static background and c will tend towards 0. Conversely, if the confidence is high, the loss from this box will primarily propagate gradients towards optimizing $\mathbf{T}$, and if it is low, the loss will contribute to optimizing $\mathbf{T}_{ego}$. During inference, we can threshold c to determine which boxes contain moving objects.

Since the ego-motion is shared among all bounding boxes, it is more constrained than the per-box rigid motions. When a bounding box contains a static part of the scene, we often find that the foreground rigid motion is nearly identical to the ego-motion, but converges to a barely smaller loss. This results in many static background objects having a high confidence. To address this, we add a small constant penalty ϵ to the foreground term. This value can be interpreted as the minimum distance an object needs to traverse to be classified as a moving object.

3.1.5 Auxillary Terms

In addition to the main loss function described above, we also incorporate a few auxillary terms. Since we narrow our use case specifically to autonomous driving datasets, we broadly assume that the detected moving objects are cars, and construct certain auxillary terms based on this assumption.

Box Dimension Regularization We apply a small penalty L_{shape} to the bounding box shape that constrains it to be about the size of an average car:

$$L_{shape} = \Delta_w^2 + \Delta_l^2 + \Delta_h^2 \tag{3.7}$$

where $\Delta_w, \Delta_l, \Delta_h$ are the underlying parameters to the width, length, and height of the boxes, described in Section 3.1.7.

Heading Term Because cars face the direction they are moving, we apply a consistency loss $L_{heading}$ that forces the heading of bounding boxes to point in the same direction as their motion.

$$L_{heading} = ||\theta_{xy} - \mathbf{t}_{xy}||_2^2 \tag{3.8}$$

where θ_{xy} is a 2D vector parameterizing the heading, as described in Section 3.1.7, and $\mathbf{t}_{xy}$ are the x and y components of $\mathbf{t}$ on the ground plane.

Angle Term Moving cars do not experience large rotations, so we apply a small penalty L_{angle} on the magnitude of the rotation angle θ due to $\mathbf{R}$.

$$L_{angle} = \theta^2 \tag{3.9}$$

Mass Term We also apply a mass term L_{mass} that encourages the bounding boxes to maximize the number of points inside them. If the box contains a moving object, this term encourages them to converge around it entirely. If it does not contain a moving object, it is

still helpful for the boxes to contain many background points in order to make the ego-motion estimation more robust.

$$L_{mass} = -\sum_{i=1}^{N_1} w_i \tag{3.10}$$

where w_i is the i$^{\text{th}}$ element of $\mathbf{w}$, the unnormalized vector of membership weights from the differentiable bounding box.

Combining these terms, our final per-box loss is

$$L = L_{nn} + \lambda_{shape}L_{shape} + \lambda_{heading}L_{heading} + \lambda_{angle}L_{angle} + \lambda_{mass}L_{mass} \tag{3.11}$$

where each λ is a hyperparameter that controls the influence of the corresponding loss.

3.1.6 Inference

3.1.6.1 Scene Flow

To select the boxes containing moving objects, we first filter out any bounding boxes that contain fewer than n_{min} points, where n_{min} is a dataset dependant threshold. Then we apply non-maximum-suppression and keep boxes with a score of 0.85 or above. We classify these boxes as dynamic. With the dynamic boxes, we segment P_1 by assigning each point to the most confident box it is in, as there may still be some overlap between boxes. Finally, we apply each box's rigid transformation to its points, and the ego-motion to the remaining background points to compute the scene flow as

$$\mathbf{f}_i = \mathbf{R}_i\mathbf{p}_i + \mathbf{t}_i - \mathbf{p}_i \tag{3.12}$$

where $\mathbf{p}_i$ is the i$^{\text{th}}$ point in P_1, $\mathbf{f}_i$ is its scene flow prediction, and $\mathbf{R}_i$ and $\mathbf{t}_i$ are its associated rotation and translation.

3.1.6.2 Motion Segmentation

We found that our described approach performs well on scene flow metrics, but that this performance was not initially reflected in the motion segmentation metrics. LiDAR scans often contain regions that are very sparse and therefore have poor correspondences. In these regions, our method sometimes predicts a dynamic object with a minute motion. To address this, we filter out any positive predictions that move less than 0.2 meters, corresponding to about 4.5 miles per hour, which we assume to be the minimum speed of a dynamic car or cyclist). Additionally, we found that our method would predict false positives at the boundaries of the scene or in occluded regions, where objects fade in and out of the scene. To address this, we introduce a cycle consistency check. Specifically, for every bounding box b_i, we optimize an additional $SE(3)$ transform T_i^i and confidence parameter c_i^i minimizing the loss function, but this time using the NND from P_2 to P_1 and computing the differentiable

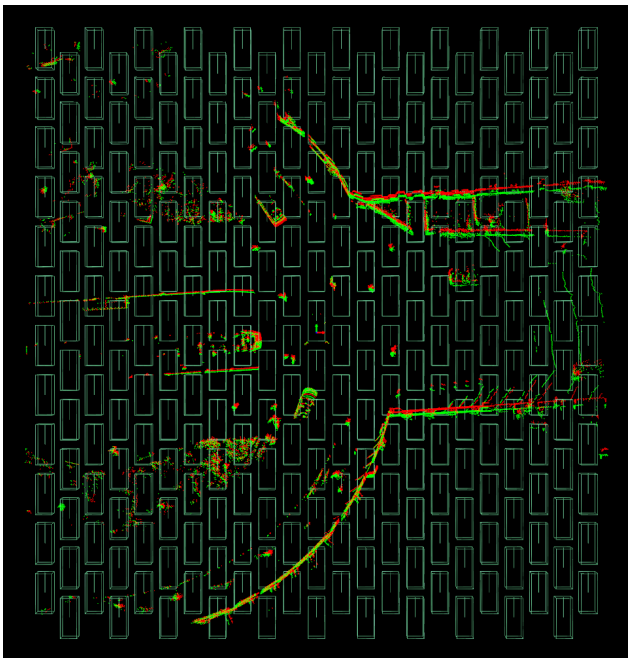


Figure 3.3: Initial bounding boxes during optimization.

box weights using b_i transformed according to its forward transform T_i. During inference, we only keep boxes where both confidence scores are above the threshold. Additionally, we assert that the two rigid transforms must be similar to each other. We measure their similarity by transforming the corners of the box by $T_i^i T_i$ and computing the average displacement of the corners, removing any predictions where the average displacement exceeds 0.2 m. We found this check mitigates false positives stemming from the aforementioned reasons.

3.1.7 Implementation Details

Our code is implemented using PyTorch for automatic differentiation. We use Adam [28] with a learning rate of 0.015 over 500 epochs to optimize the parameters. Optimizing 36 bounding boxes over a single pair of point clouds with 40,000 points each takes about 1.7 minutes and uses 1.4 GB on a Nvidia Titan RTX GPU.

Before optimization, the bounding boxes are initialized to a grid of anchor boxes distributed in a grid over the xy ground plane. In the stereo setting, the grid cells are 4 m wide and 8 m long, while for LiDAR they are instead 6 m long. Every other column is shifted forward by half the length of the grid cell, forming a diamond grid pattern. Following [30], we initialize template anchor boxes to $2 \times 34.9 \times 31.9$ m with a heading angle of 0 and vertical displacement of -1 m, as shown in Figure 3.3. The rotations and translations are initialized to the identity and zero respectively.

Some of our scene flow parameters cannot be optimized directly and need to be computed

Table 3.1: Hyperparameters

Dataset	k	ϵ(m)	λ_{shape}	$\lambda_{heading}$	λ_{angle}	λ_{mass}	n_{min}
StereoKITTI	8	0.01	8	1000	0.01	0.001	500
StereoKITTI Downsampled	8	0.02	8	1000	0.01	0.002	80
LidarKITTI	8	0.03	8	1000	0.01	0.002	50
nuScenes	8	0.04	8	1000	0.25	0.01	20
SemanticKITTI	8	0.05	8	1000	0.25	0.002	50

from latent parameters. Inspired by [31], we represent rotations as 3×3 matrices and project them onto $SO(3)$ via SVD. The confidences are computed by applying a sigmoid function to latent logits. The latent variables for the bounding box dimensions are exponentiated and then multiplied by the default anchor box dimensions to compute the true dimensions of the bounding box. Lastly, the heading is parameterized by a 2D vector and converted to an angle via $atan2$.

When computing the NND, in practice, we concatenate point cloud normals to the xyz coordinates and compute the NND in 6 dimensions to draw better correspondences. Point cloud normals are computed using [58], which finds the main principal vector for a local region around each point. In our case, we use the 30 nearest neighbors.

To improve efficiency, when computing the differentiable bounding box weights, we only keep points with weight above $1e-6$. Additionally, we only compute the loss on boxes that contain points with weight above this threshold i.e. they are not empty.

In Table 3.1, we list the hyperparameters we use on our various datasets. See section 3.2.1 for details on the datasets. Hyperparameters are primarily adjusted to account for the sparsity of point clouds in the dataset. Sparser point clouds require larger ϵ and λ_{mass} but smaller n_{min}.

3.2 Results

We evaluate our method both quantitatively and visually on various datasets for scene flow estimation, moving object detection, and ego-motion estimation and compare them with the current state of the art. Additionally, we explore the effects of our various design choices with an ablation study.

3.2.1 Datasets

KITTI Scene Flow [43, 44, 14] We evaluate our scene flow predictions on the KITTI Scene Flow Dataset, a real world autonomous driving dataset containing 142 pairs of point clouds annotated with scene flow vectors. Pedestrians and cyclists are removed, so the

Table 3.2: Dataset Details. N refers to the number of points in a point cloud.

Dataset	median N	flow	ego-motion	segmentation	correspondences
StereoKITTI	25218.5	✓	✓	✓	✓
LidarKITTI	4388.5	✓	✓	✓	
SemanticKITTI	67532.5		✓	✓	
nuScenes	6727.5	✓	✓	✓	

only moving objects are cars. The dataset has two settings: StereoKITTI and LidarKITTI. StereoKITTI is the traditional setting in which the point clouds are generated from stereo disparity maps and therefore have correspondences. Most of the existing methods evaluate on this setting. LidarKITTI is a more challenging setting in which the point clouds are captured by a Velodyne 64-beam LiDAR. They are sparser and do not have direct correspondences. The ground truth scene flow vectors are assigned by projecting the LiDAR points onto the StereoKITTI disparity map and using the associated scene flow annotations. Both datasets also contain segmentation masks, which we use to evaluate our motion segmentation accuracy.

For a fair comparison, we adopt the common data preprocessing step introduced by [17] of removing ground points via naive thresholding at 1.4 m below the sensor, and cropping any points further than 35 m from the sensor. We use this step on all our datasets and experiments. Due to memory constraints during training, most prior learning-based approaches also downsample their point clouds to 8,192 points. [3] found that increasing the point density for these approaches during evaluation actually decreases performance due to the domain shift. Our efficient implementation enables us to use the entire point cloud, so on StereoKITTI, we report our performance using both 8,192 points, and all points.

On LidarKITTI, we optimize our scene flow parameters over the entire LiDAR scan and evaluate it on the points in front of the vehicle with scene flow annotations. We found that only optimizing over points in front of the vehicle in LiDAR scans results in false positive detections at the point cloud boundary due to cropping. This problem is somewhat mitigated when using the entire 360°point cloud.

nuScenes [8] nuScenes is a more challenging dataset than KITTI, consisting of sparser LiDAR sweeps and more complex driving scenarios. We use a subset of nuScenes released by [33], consisting of 310 point cloud pairs with ground points removed using RANSAC. The ground truth flow vectors are drawn using track annotations.

SemanticKITTI [4] We evaluate our segmentation and ego-motion predictions on SemanticKITTI, a large scale autonomous driving dataset curated from the KITTI odometry dataset. It consists of 21 sequences of LiDAR frames annotated with instance segmentation labels and ego-motions. Both vehicles and pedestrians are present. Due to the size of the dataset, we only evaluate on a random subset of 500 point cloud pairs.

RawKITTI [15] RawKITTI consists of the approximately 38,000 raw LiDAR scans from the KITTI dataset. [3] uses this for self-supervised training.

FlyingThings3D (FT3D) [42] FlyingThings3D is a large synthetic dataset consisting of stereo pairs with scene flow annotations generated from CAD models moving randomly in space. While our approach does not make use of FT3D, previous supervised approaches use it for training.

3.2.2 Evaluation Metrics

To evaluate scene flow, we use the standard metric of 3D end-point error metric (EPE3D), which is the average $l2$ distance between the predicted and ground truth scene flow. We also adopt the following metrics from [35, 17]: strict accuracy (Acc3DS): the percentage of points with EPE3D < 0.05 m or relative error $< 5\%$; relaxed accuracy (Acc3DR): the percentage of points with EPE3D < 0.1 m or relative error $< 10\%$; and Outliers: the percentage of points with EPE3D > 0.3 m or relative error $> 10\%$.

To evaluate motion segmentation, we group all moving points as a single class and report the intersection-over-union (IoU) on this class, the mean intersection-over-union (mIoU), and the segmentation accuracy, each defined as

$$IoU = \frac{TP}{TP + FP + FN} \tag{3.13}$$

$$mIoU = 0.5(\frac{TP}{TP + FP + FN} + \frac{TN}{TN + FP + FN}) \tag{3.14}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.15}$$

We only consider moving vehicles and cyclists, as pedestrians move too slowly and non-rigidly for our method to detect. For ego-motion estimation, we report the average rotation and translation errors of our predictions in degrees and meters, as well as the rotation and translation accuracy, defined as the percentage of scenes where the rotation error $< 0.5°$ and the translation error < 0.1 m.

3.2.3 Baselines

We compare our work against the following state of the art approaches in both supervised and self-supervised scene flow estimation: FlowNet3D [35], HPLFlownet [17], PointPWCNet [69], FLOT [49], EgoFlow [76], FlowStep3D [70], HCRF-Flow [32], WeaklyRigidFlow [16], SLIM [3], and NSFP [33]. Additionally, we directly optimize scene flow vectors over two representative state of the art self-supervised loss functions: (i) the Chamfer Distance, used in [48, 70] (ii) the self-supervised loss from [69], which adds a smoothness and laplacian term to the Chamfer Distance. We compare motion segmentation results against LiDAR MOS [9], a state-of-the-art supervised approach, and SLIM [3], a state-of-the-art self-supervised

Table 3.3: Evaluation on KITTI Scene Flow.

Method	Supervision/ Approach	Training Data	EPE3D ↓	Acc3DS ↑	Acc3DR ↑	Outliers ↓
StereoKITTI						
FlowNet3D [35]	Full	FT3D	0.177	0.374	0.668	0.527
HPLFlowNet [17]	Full	FT3D	0.117	0.478	0.778	0.410
PointPWCNet [69]	Full	FT3D	0.069	0.728	0.888	0.265
FLOT [49]	Full	FT3D	0.056	0.755	0.908	0.242
EgoFlow [76]	Full	FT3D	0.069	0.670	0.879	0.404
FlowStep3D [70]	Full	FT3D	0.055	0.805	0.925	0.149
HCRF-Flow [32]	Full	FT3D	0.053	0.863	0.944	0.180
WeaklyRigidFlow [16]	Full	FT3D	0.042	0.849	0.959	0.208
PointPWCNet [69]	Self	FT3D	0.255	0.238	0.496	0.686
EgoFlow [76]	Self	FT3D	0.415	0.221	0.372	0.810
FlowStep3D [70]	Self	FT3D	0.102	0.708	0.839	0.246
SLIM [3]	Self	RawKITTI	0.1207	0.5178	0.7956	0.4024
SLIM*[3]	Self	RawKITTI	0.0668	0.7695	0.9342	0.2488
Chamfer*	Optimization	-	0.991	0.056	0.071	0.942
PointPWCNet [69]	Optimization	-	0.657	0.357	0.405	0.72
NSFP [33]	Optimization	-	0.036	0.912	0.961	0.154
NSFP*[33]	Optimization	-	0.034	0.914	0.962	0.151
Ours	Optimization	-	0.035	0.932	0.971	0.146
Ours*	Optimization	-	**0.017**	**0.973**	**0.989**	**0.096**
LidarKITTI						
PointPWCNet [69]	Full	FT3D	0.390	0.387	0.550	0.653
FLOT [49]	Full	FT3D	0.653	0.155	0.313	0.837
WeaklyRigidFlow [16]	Weak	SemKITTI	0.094	0.784	0.885	0.314
Chamfer*	Optimization	-	0.944	0.022	0.057	0.992
PointPWCNet [69]	Optimization	-	0.734	0.248	0.347	0.845
NSFP*[33]	Optimization	-	0.142	0.688	0.826	0.385
Ours*	Optimization	-	**0.085**	**0.883**	**0.929**	**0.239**
nuScenes						
Chamfer*	Optimization	-	0.879	0.035	0.082	0.976
PointPWCNet*[69]	Optimization	-	0.615	0.199	0.328	0.86
NSFP *[33]	Optimization	-	0.177	0.374	0.668	0.527
Ours *	Optimization	-	**0.107**	**0.717**	**0.862**	**0.321**

* methods that use the entire point cloud. All other methods downsample to 8,192 points.

approach. Both methods predict binary motion segmentation masks, classifying each point as static or dynamic. LiDAR MOS preprocesses its point clouds differently and does not exclude pedestrians in its evaluation. For ego-motion estimation, we use ICP [7] as a baseline.

Table 3.4: Ego-Motion Estimation Evaluation on SemanticKITTI.

Method	Rotation Error (∘) ↓	Translation Error (m) ↓	Rotation Accuracy ↑	Translation Accuracy ↑
ICP [7]	0.244	0.122	0.906	0.878
Ours	**0.235**	**0.107**	**0.916**	**0.94**

3.2.4 Scene Flow Evaluation

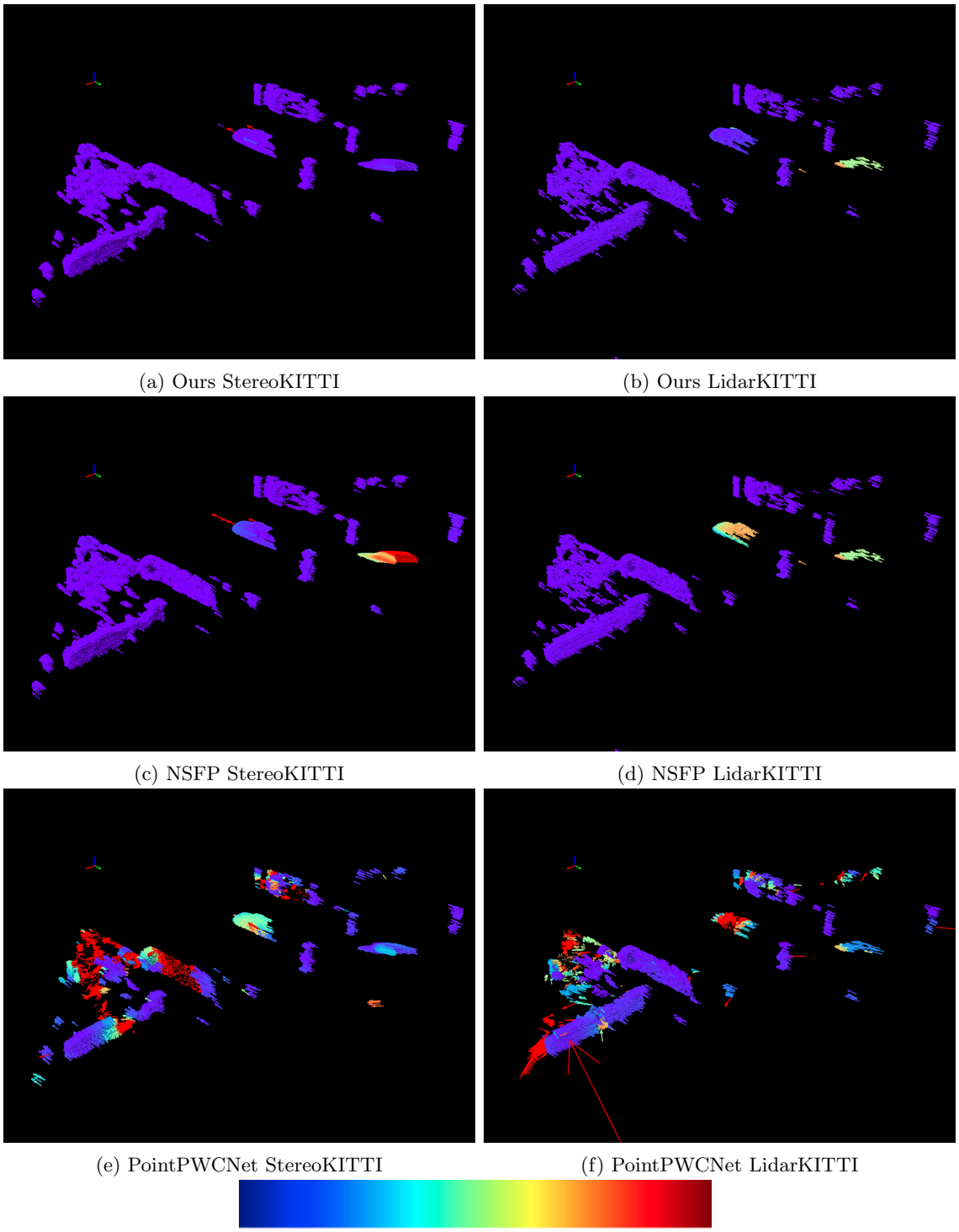
We report our results on StereoKITTI, LidarKITTI, and nuScenes in Table 3.3, as well as the baselines. Without relying on any annotated data, our method significantly outperforms the state of the art supervised and self-supervised baselines in all metrics on both datasets. In particular, when utilizing all points on StereoKITTI, our approach achieves a 2x lower EPE3D than the previous state of the art supervised approach and 4x lower EPE3D than the state of the art self-supervised approach. Even when using 8,192 points, our approach significantly outperforms all baselines on all metrics. Similarly, in the LidarKITTI setting, we outperform the current state-of-the-art supervised and weakly-supervised approaches by a substantial margin. From Table 3.3, it is clear that the supervised approaches are unable to generalize to LiDAR data. Lastly, despite being a more challenging dataset, our method performs well on nuScenes, achieving an average error of 0.107 m, significantly outperforming [33].

3.2.5 Ego-Motion Evaluation

Our ego-motion results on SemanticKITTI are shown in Table 3.4. We outperform ICP on all metrics, achieving an average of 0.235° rotation error and 0.107 m translation error. As a note, we observed that ICP performs unusually well on SemanticKITTI due to the already close alignment of the LiDAR scans, as well as the absence of moving objects in several of the scenes.

3.2.6 Motion Segmentation Evaluation

Our method achieves 92.9% accuracy and 86.6% mIoU on StereoKITTI, significantly outperforming SLIM's 60.1% accuracy and 42.9% mIoU. On SemanticKITTI, our method achieves 34.5% IoU, reliably detecting large, fast-moving objects. LiDAR MOS, achieves 56% IoU, performing significantly better. However, it requires ground-truth segmentation masks, while our approach does not require any labels.



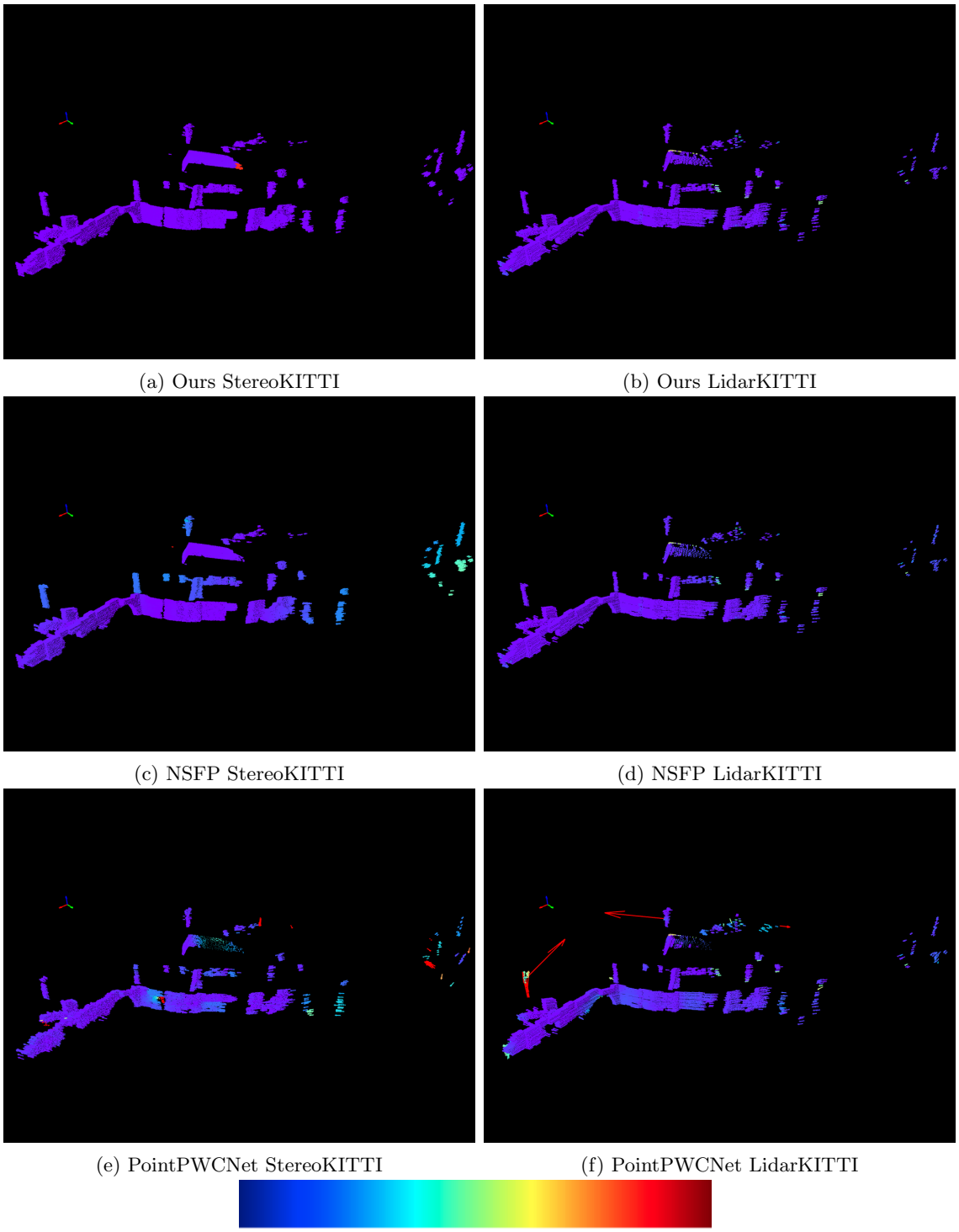
(a) Ours StereoKITTI (b) Ours LidarKITTI

(c) NSFP StereoKITTI (d) NSFP LidarKITTI

(e) PointPWCNet StereoKITTI (f) PointPWCNet LidarKITTI

(g) Error Colorbar

Figure 3.4: Visualization of scene flow predictions for our method, NSFP, and the PointPWCNet loss function under direct optimization on a scene in KITTI. Color indicates the EPE of the prediction, with red indicating high error and purple indicating low error. For StereoKITTI, the colorscale ranges from 0-0.5 m error, while for LidarKITTI, it ranges from 0-1 m. In this scene, the ego vehicle is moving forward as two cars approach from the opposite direction. Our approach is able to accurately predict the flow on both cars in the stereo setting, and the closer one in the LiDAR setting. NSFP struggles on moving objects, while PointPWC predicts locally smooth, but incoherent flow.



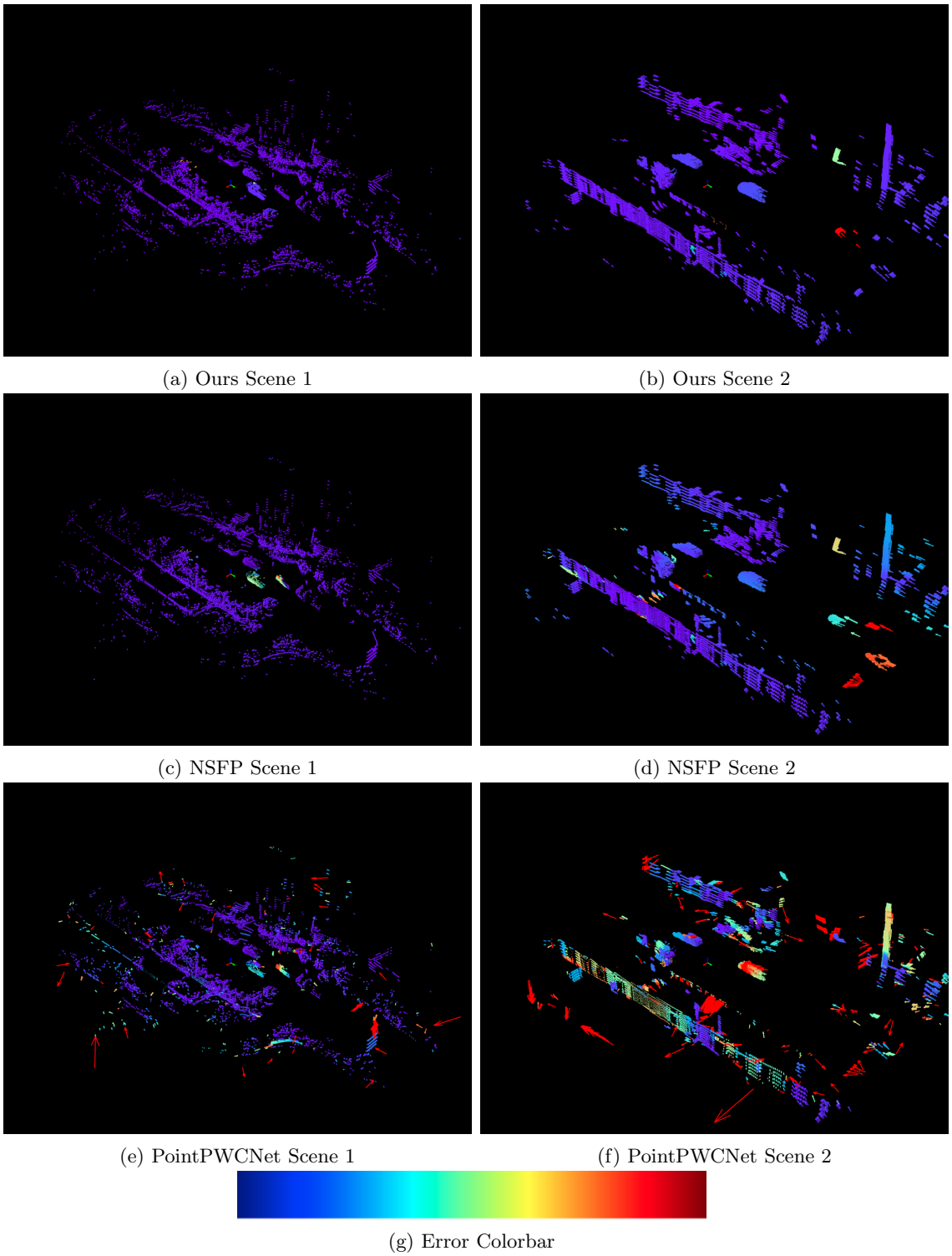
(a) Ours StereoKITTI (b) Ours LidarKITTI

(c) NSFP StereoKITTI (d) NSFP LidarKITTI

(e) PointPWCNet StereoKITTI (f) PointPWCNet LidarKITTI

(g) Error Colorbar

Figure 3.5: Visualization of scene flow predictions for our method, NSFP, and the PointPWCNet loss function under direct optimization on another scene in KITTI. Color indicates the EPE of the prediction, with red indicating high error and purple indicating low error. For StereoKITTI, the colorscale ranges from 0-0.5 m error, while for LidarKITTI, it ranges from 0-1 m. In this scene, a van is driving ahead of the moving ego vehicle. Our method and NSFP are able to accurately predict the flow on this scene in both settings, although our method is slightly more accurate. PointPWC also generally performs well but struggles in the sparser regions of the point cloud.



(a) Ours Scene 1

(b) Ours Scene 2

(c) NSFP Scene 1

(d) NSFP Scene 2

(e) PointPWCNet Scene 1

(f) PointPWCNet Scene 2

(g) Error Colorbar

Figure 3.6: Visualization of scene flow predictions for our method, NSFP, and the PointPWCNet loss function under direct optimization on two scenes in nuScenes. Color indicates the EPE of the prediction, with red indicating 1 m and purple indicating 0 m EPE. In scene 1, the ego vehicle is driving forward along with two cars ahead of it and one behind it. Our method is able to predict the motion of all cars but the one behind, due to the sparsity of points on that car. NSFP struggles with two of the moving cars and also falsely predicts the motion of a parked car. PointPWC's prediction exhibits a lot of artifacts, especially at the boundary of the scene. In scene 2, the ego vehicle approaches an intersection as another car drives close behind. In the other lane, three cars move in the opposite direction. Another car moves along the perpendicular street of the intersection, totalling five dynamic vehicles in this scene.

Table 3.5: Ablation study over various loss terms and design choices on StereoKITTI. ∇b refers to whether we use differentiable or non-differentiable bounding boxes.

∇b	L_{shape}	L_{mass}	$L_{heading}$	L_{angle}	EPE3D ↓	Acc3DS ↑	Acc3DR ↑	Outliers ↓
	✓		✓	✓	0.284	0.74	0.759	0.319
✓		✓	✓	✓	0.218	0.473	0.678	0.457
✓	✓		✓	✓	0.401	0.66	0.663	0.406
✓	✓	✓		✓	0.024	0.959	0.984	0.116
✓	✓	✓	✓		**0.017**	**0.974**	**0.989**	**0.096**
✓	✓	✓	✓	✓	**0.017**	0.973	**0.989**	**0.096**

3.2.7 Visualizations

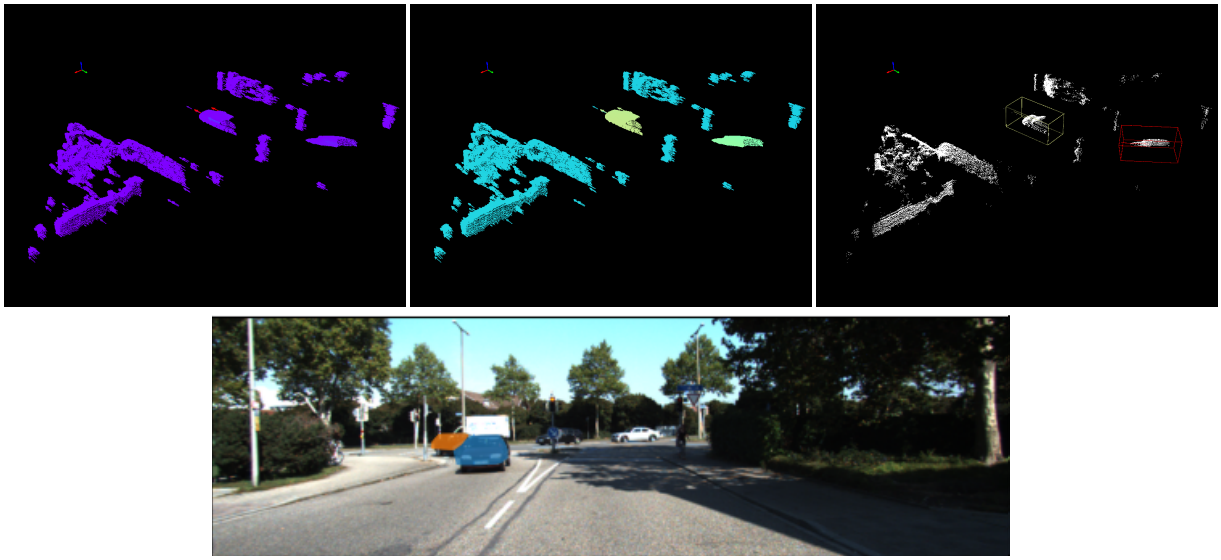
We qualitatively compare our method against NSFP [33] and the PointPWCNet [69] loss function, two state-of-the-art optimization methods, by visualizing the predictions on two scenes from KITTI in Figures 3.4 and 3.5 and one scene from nuScenes in Figure 3.6. On all scenes we find that our method produces the most accurate predictions. In particular, NSFP struggles to accurately predict the flow for dynamic objects, as shown in 3.4. Meanwhile, PointPWCNet can typically generate locally smooth predictions, but they are very noisy and fail to exhibit object level coherence. This is especially apparent in Figure 3.4 as well, where the predicted motion seems to change in different regions of the moving cars and there are a lot of noisy predictions near the boundaries of the scene. Despite the fact that the primary supervisory signal for all three approaches is the NND, our approach achieves more accurate predictions by directly constraining the scene flow to be rigid and physically consistent.

For a more exhaustive visualization of just our method, we also display our predictions over six scenes from StereoKITTI and LidarKITTI, visualized in 4 different ways, shown in Figures 3.7, 3.8, 3.9, 3.10, 3.11, 3.12. Our method is able to predict accurate scene flow and segmentation masks on all scenes using both stereo and LiDAR, although the stereo predictions are slightly more accurate. This is evidenced in that while the LiDAR predictions successfully detect most moving objects, the stereo predictions detect all of them, illustrated in Figures 3.7 and 3.11. As a note, in Scenes 4 and 6, there are moving bikers and motorcyclists. These points are cropped out in StereoKITTI, but with LidarKITTI, we utilize the entire point cloud, so our method detects these moving objects. They are shown as empty boxes in the third visualization on Scenes 4 and 6.

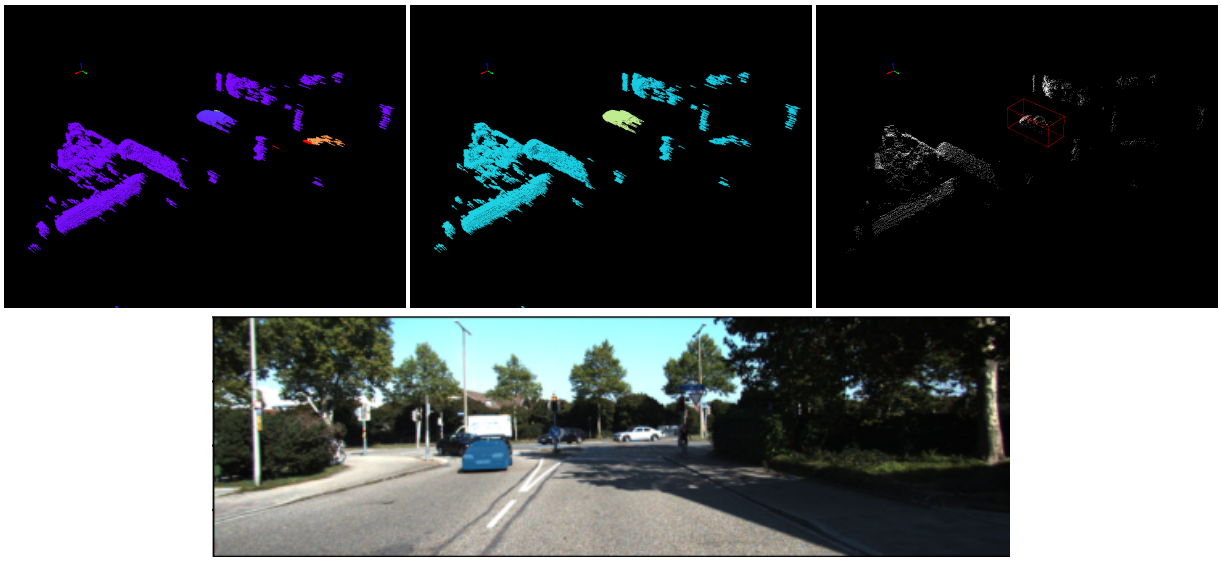
3.2.8 Ablation Study

To evaluate our design choices, we conduct an ablation study, shown in Table 3.5. We find that the differentiable bounding boxes, shape regularization, and mass term contribute significantly to our performance. Without differentiable bounding boxes, the shape and position of the boxes are not updated. Without the shape regularizer, the mass term causes bounding boxes to grow too large. Without the mass term, the boxes tend to favor empty

space over regions with points. The heading and angle terms also provide slight increases in accuracy. On StereoKITTI, we actually find that the angle term slightly decreases the accuracy, but generally, and especially in LiDAR settings, it improves optimization stability.

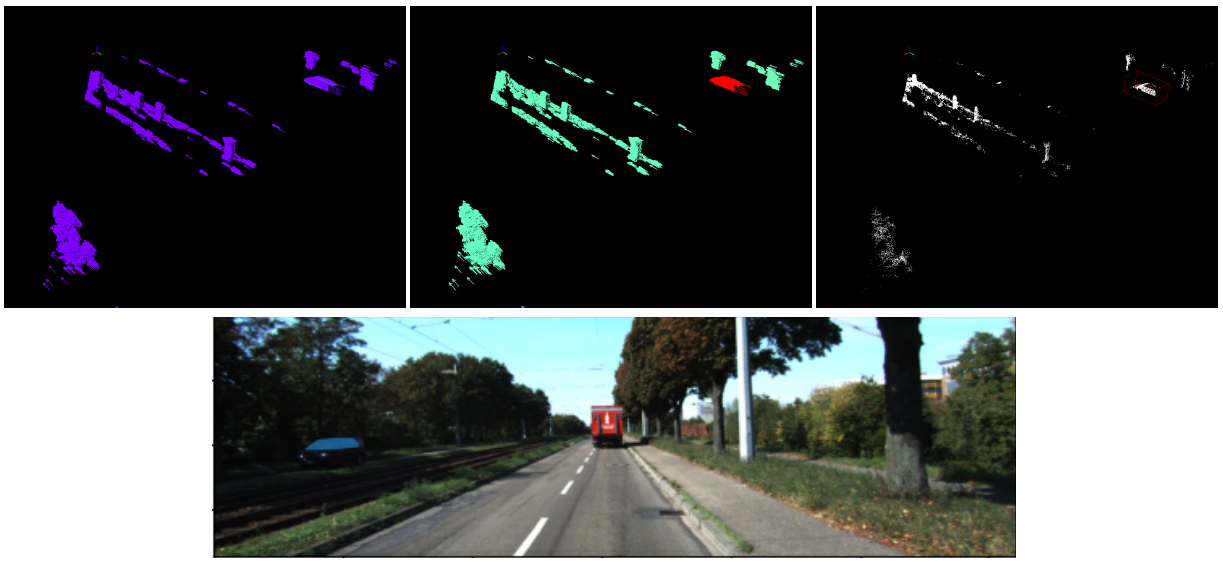


(a) StereoKITTI

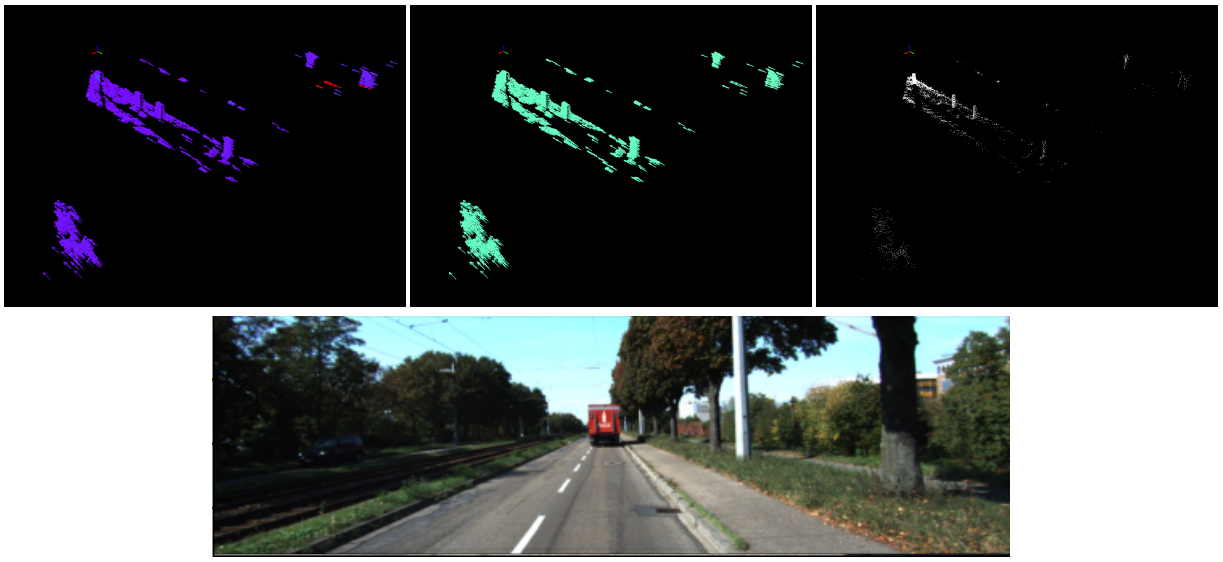


(b) LidarKITTI

Figure 3.7: Scene 1 Visualizations. (a) shows StereoKITTI predictions and (b) show LidarKITTI predictions on the same scene. The left visual in the first row of each subfigure shows the 3D end-point-error of our predictions, similar to Figures 3.4, 3.5, 3.6. The color can be interpreted using the same colorbar as the comparative visualizations, but with purple corresponding to 0 m and red corresponding to 0.75 m. The middle visual shows the magnitude of the predicted scene flow vectors using the same colorbar, with purple corresponding to 0 m and red corresponding to 2.5 m. The right visual shows the predicted bounding boxes using arbitrary colors. Lastly, the bottom visual projects a convex hull of the segmented points onto the image plane for each detected moving object, performing moving object instance segmentation on images. These colors are also arbitrary. This figure displays the same scene as 3.4.

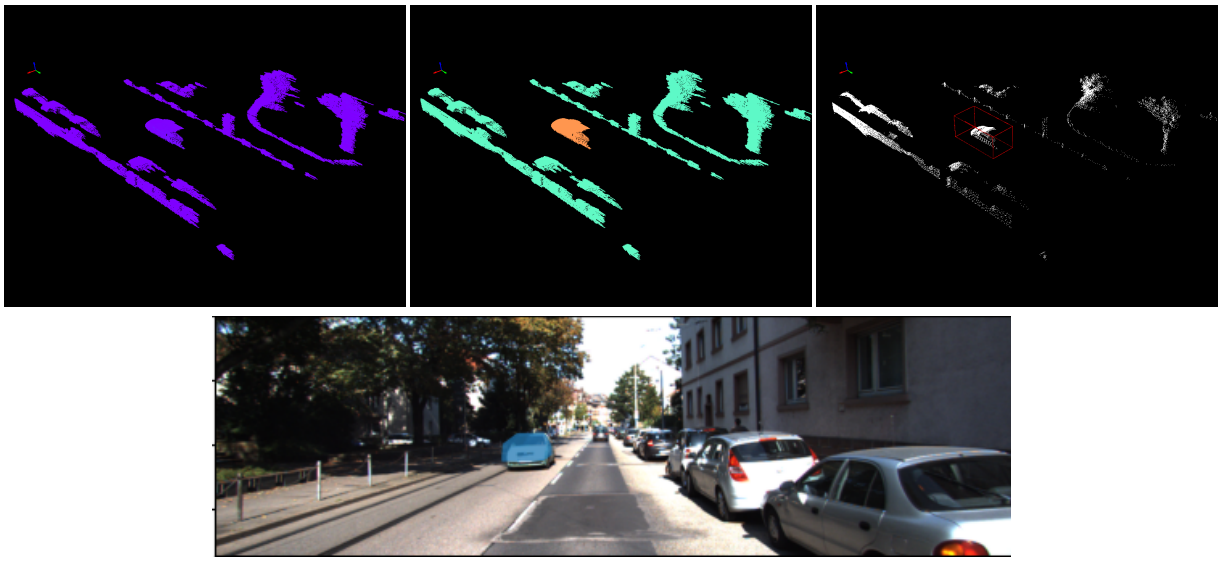


(a) StereoKITTI

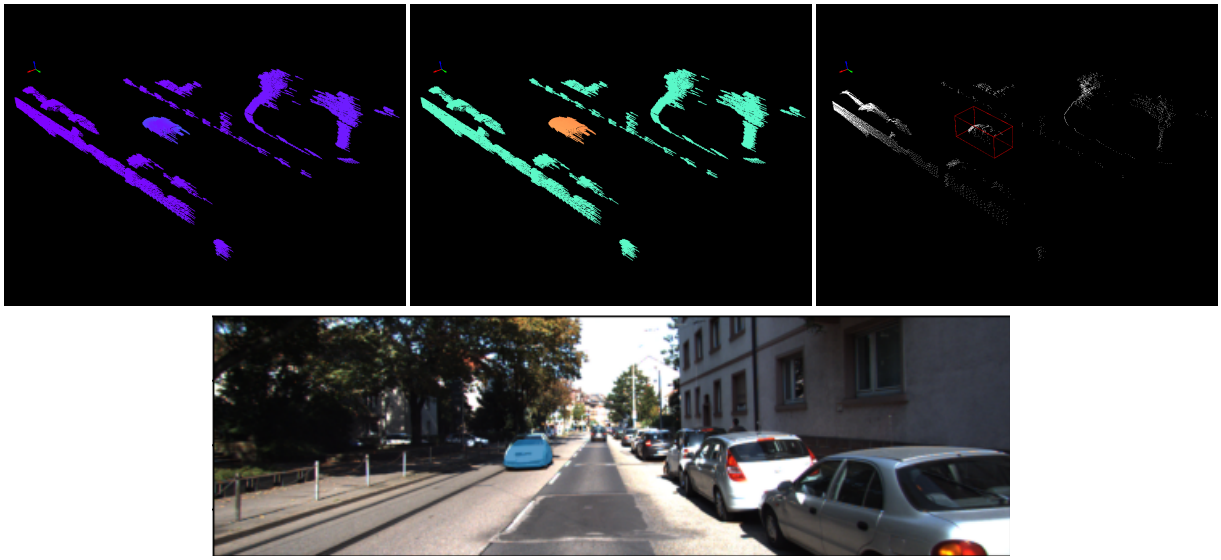


(b) LidarKITTI

Figure 3.8: Scene 2 Visualizations. Same as Figure 3.7. In this scene, the ego vehicle is moving fast on a main street as an oncoming car approaches on the other side of the road. Our method is able to identify the moving car in the stereo setting, but in the LiDAR setting, the points on the car are extremely sparse, making it difficult for our method to identify it.

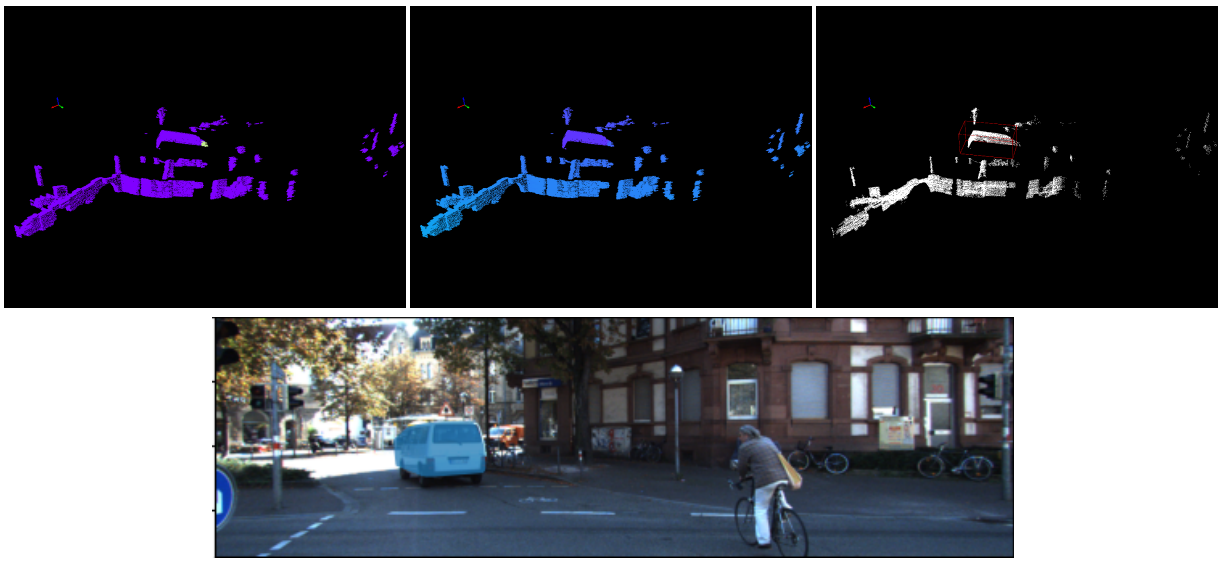


(a) StereoKITTI

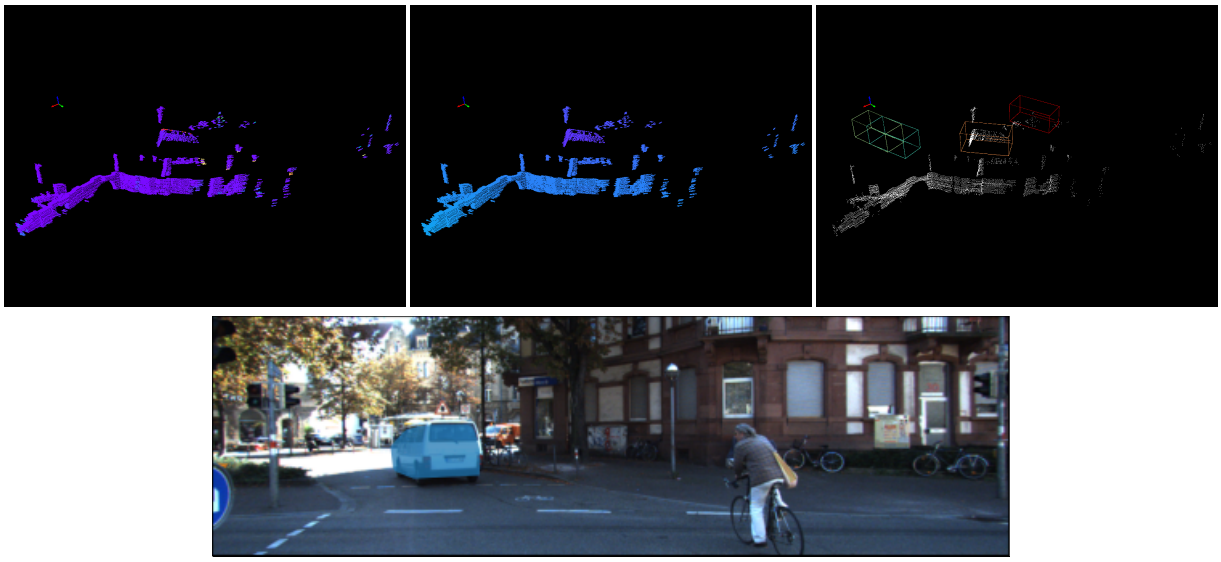


(b) LidarKITTI

Figure 3.9: Scene 3 Visualizations. Same as Figure 3.7. In this scene, the ego vehicle drives forward slowly on a narrow street as another car approaches in the opposite lane. Our method is able to accurately predict the flow on the dynamic car in both settings.

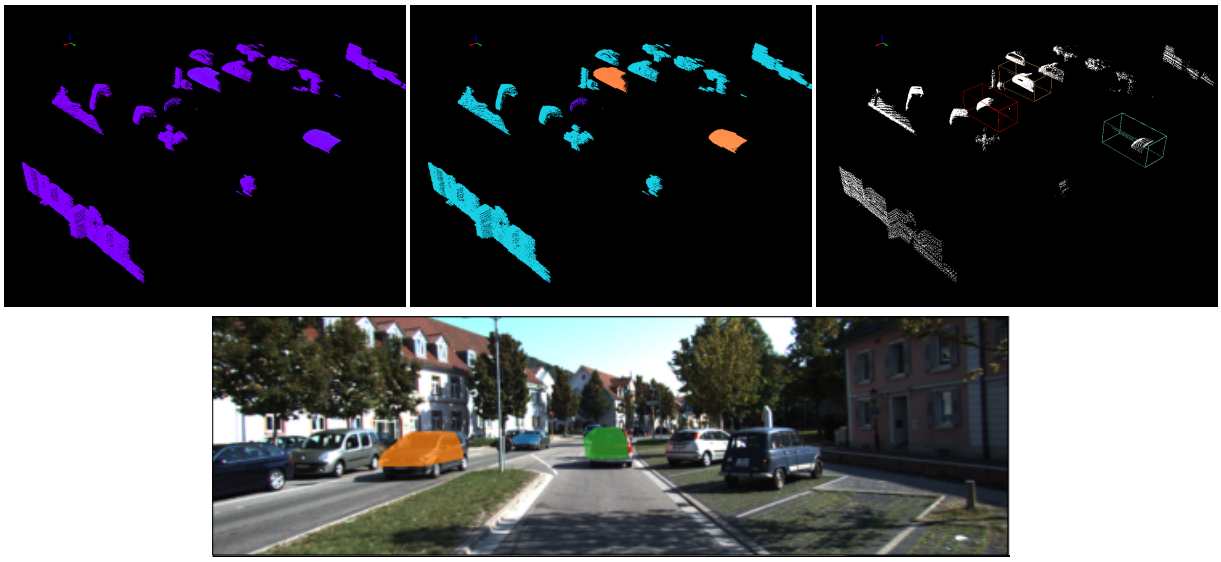


(a) StereoKITTI

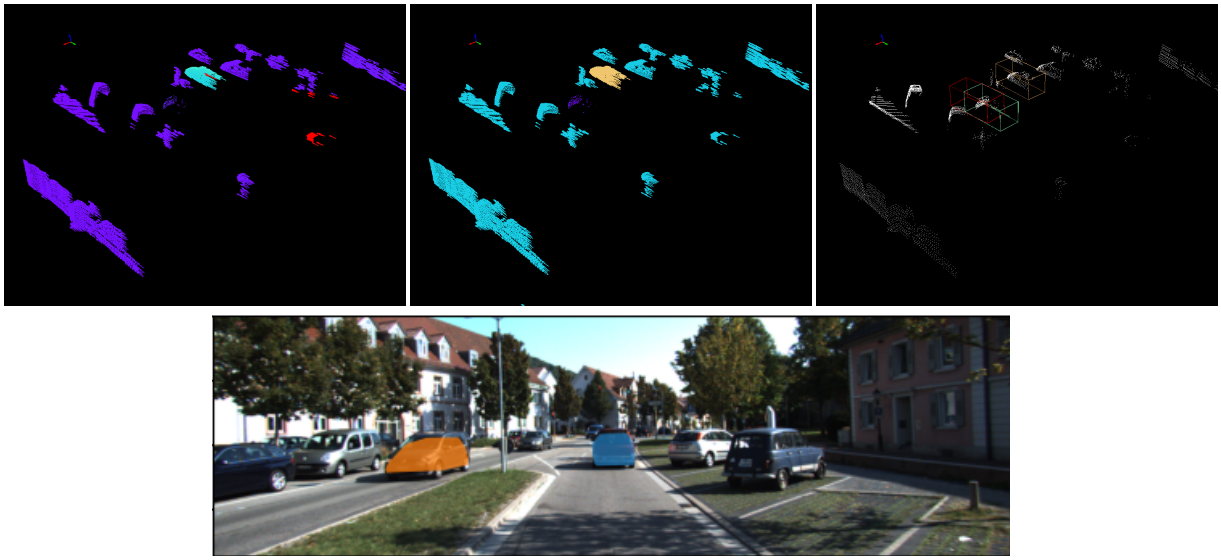


(b) LidarKITTI

Figure 3.10: Scene 4 Visualizations. Same as Figure 3.7. This figure displays the same scene as 3.5. Note that there is a biker in the scene. These points are cropped out in StereoKITTI as they don't posses ground truth scene flow annotations, but with LidarKITTI, we utilize the entire point cloud, so our method is able to detect the biker, as shown by the empty green box.

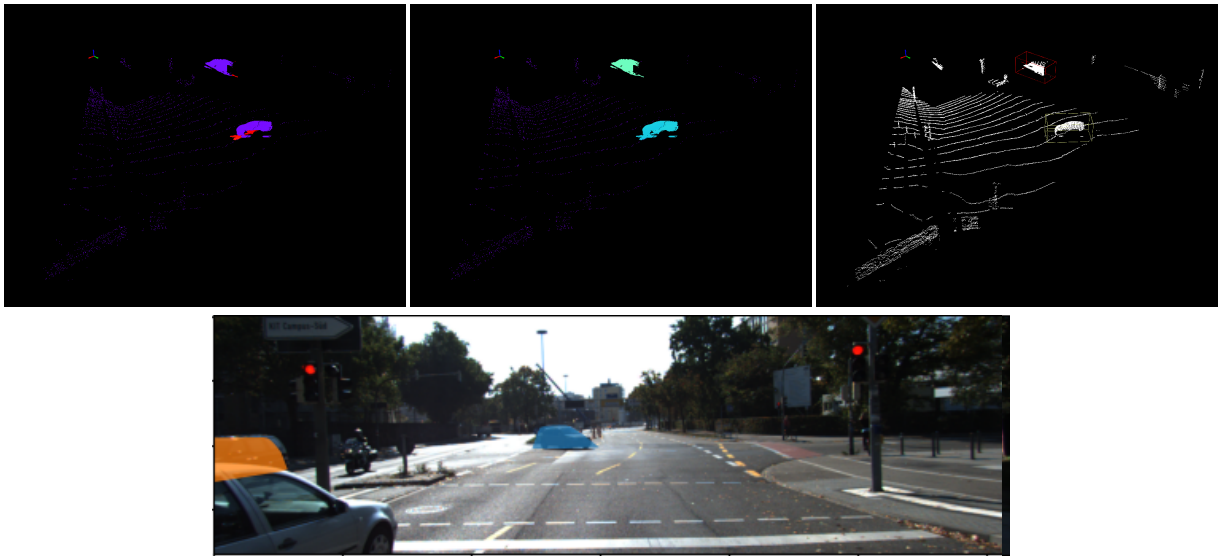


(a) StereoKITTI

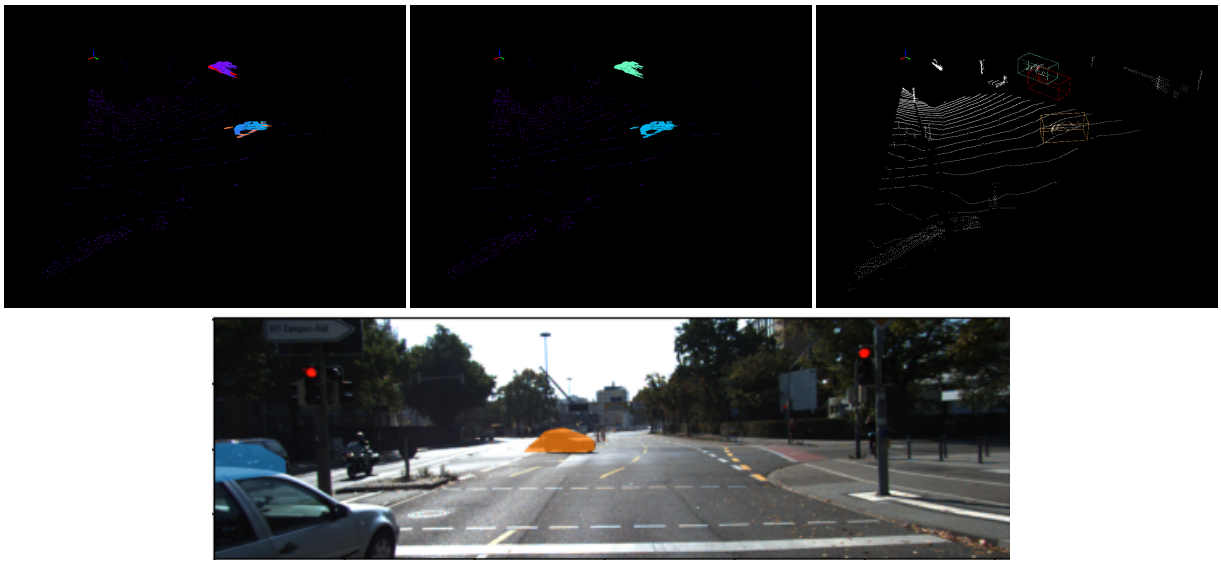


(b) LidarKITTI

Figure 3.11: Scene 5 Visualizations. Same as Figure 3.7. In this scene, the ego vehicle is driving behind another car as two cars approach from the opposite lane. Our method predicts all three moving cars in the stereo setting, but misses the furthest car in the LiDAR setting due to its sparsity.



(a) StereoKITTI



(b) LidarKITTI

Figure 3.12: Scene 6 Visualizations. Same as Figure 3.7. In this scene, the ego vehicle is stopped at a stop light at an intersection while a car and a motorcyclist cross the intersection and approach from the other side of the street. Additionally, another car coming from the same direction makes a left turn at the intersection. Our method is able to identify both moving cars in both settings. Similar to 3.11, the points of the motorcyclist are not present in the stereo setting but are present in the LiDAR setting, and our method successfully identifies at, as indicated by the empty red box.

Chapter 4

Temporal LiDAR Frame Prediction for Autonomous Driving

In this chapter we detail our work on LiDAR prediction. In section 4.1 we discuss the details of our approach, including the network architecture, loss function and training details. In section 4.2 we show and discuss our results and visualizations.

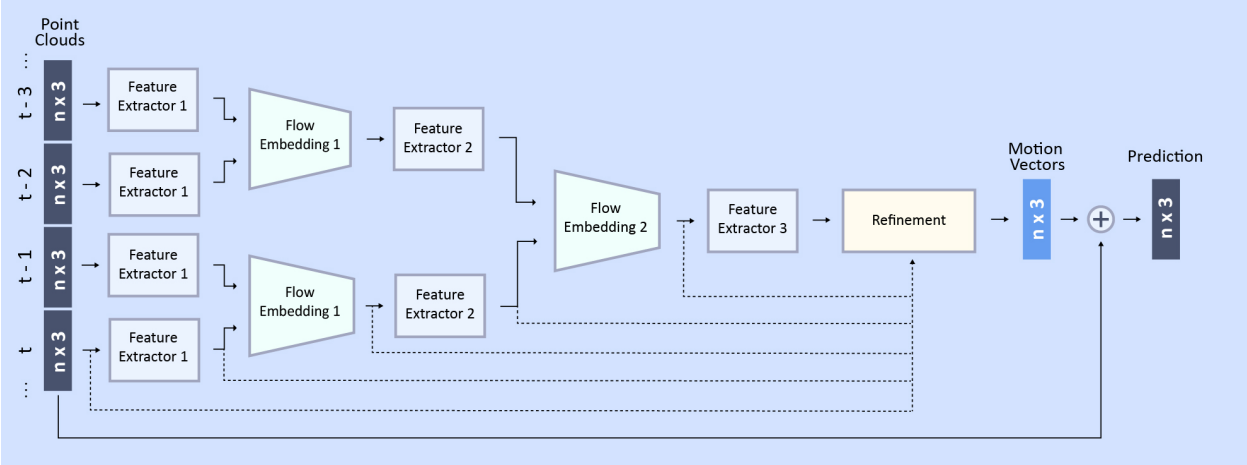


Figure 4.1: **Generic architecture framework.** Our framework takes in the past 4 frames and generates motion vectors to predict the next frame. The specific architecture is determined by which feature extractor is used and whether or not downsampling is used. The refinement module may use any of the previous learned features, as indicated by the dashed lines. For specific architecture details refer to 4.3.

4.1 Method

Our work aims to predict future point cloud frames given previous ones. In this section we introduce our architecture framework, describe several variants of this framework, and share our training details.

4.1.1 Architecture Framework

Our generic architecture framework is illustrated in Figure 4.1. Given the past 4 frames of point clouds $x_{t-3}, ..., x_t$, our network generates x^*_{t+1}, the prediction for x_{t+1}. While 2 frames is sufficient for recovering the velocity of objects in a scene, we condition on 4 frames because at least 3 is required to recover second order dynamics i.e. acceleration, and additional frames provide contextual information. At each stage of the network, we first extract pointwise features from every frame and then learn the dynamics of the scene using the flow embedding layer introduced in [35]. After two such stages, we extract features on the single remaining point cloud and push the features through a refinement module. The refinement layers output a motion vector for each point in x_t. We add these predicted vectors to x_t to generate x^*_{t+1}. We chose to reformulate our task into a motion prediction problem rather than directly regressing the output point cloud because this adds more interpretability to the output of our model, and we found it to be easier to optimize as well. Point clouds in the more distant future can be predicted by applying the model recursively ie. by feeding x^*_{t+1} back into the model as a pseudo groundtruth for x_{t+1}, using this to predict x^*_{t+2}, and so on.

We explore two approaches of modulating the framework to create different architectures. The first approach is the choice of feature extractor. In this paper we experiment with the PointNet++ layer [51] and the EdgeConv layer from [65]. The second approach is whether or not to downsample the features throughout the initial stages of the network and upsample back to the original resolution in the refinement stage. We discuss these in further detail in sections 4.1.2, 4.1.3, 4.1.4.

4.1.2 Feature Extractors

PointNet++ Layer The PointNet++ layer takes an input point cloud $\{p_1, p_2, ..., p_n\}, p_i \in R^3$ and its features $\{f_1, f_2, ..., f_n\}, f_i \in R^c$, and outputs a new set of features $\{f'_1, f'_2, ..., f'_{n'}\}, f'_i \in R^{c'}$. For each point, it groups its neighbors within a given radius and applies the PointNet operation to that local region, producing a new feature vector. More specifically:

$$f'_i = \max_{j|\,||p_j - p_i|| \leq r} h_\theta(f_j, p_j - p_i) \tag{4.1}$$

where r is the radius of the ball query, h_θ is a multilayer perceptron (MLP) with weights θ and input and output dimensions R^{c+3} and $R^{c'}$ respectively, and max is the element-wise maximum function

Module	Downsampling	No Downsampling
Feature Extraction 1	SR=0.25×, mlp=[128, 128]	mlp=[32, 32]
Flow Embedding 1	SR=1×, mlp=[128]	mlp=[32]
Feature Extraction 2	SR=0.25×, mlp=[256, 256]	mlp=[64, 64]
Flow Embedding 2	SR=1×, mlp=[256]	mlp=[64]
Feature Extraction 3	SR=0.2×, mlp=[512]	mlp=[128]
Refinement	Upconv1: SR=5×, mlp1=[512], mlp2=[512] Upconv2: SR=4×, mlp1=[512], mlp2=[512] FeatProp: SR=4×, mlp=[256] mlp: [256, 128, 3]	mlp: [512, 256, 128, 3]

Table 4.1: **Downsampling vs. No Downsampling.** Architectural comparison between downsampling and non downsampling models. SR = sampling rate, Upconv refers to the Set Upconv module introduced in [35], and FeatProp refers to the feature propagation module from [51]. For additional architecture details, refer to 4.3.

EdgeConv Layer The EdgeConv layer takes in only the point features $\{f_1, f_2, ..., f_n\}, f_i \in R^c$, and outputs a new set of features $\{f'_1, f'_2, ..., f'_{n'}\}, f'_i \in R^{c'}$. For each point, it finds its k nearest neighbors (KNN) in the *feature space* and applies a MLP across the point's original feature and the difference between it and its neighbor's features. It then groups all k feature vectors with max pooling. More specifically:

$$f'_i = \max_{j=1...k} h_\theta(f_i^j - f_i, f_i) \tag{4.2}$$

where f_i^j is the feature of the jth nearest neighbor of f_i in the feature space, h_θ is a MLP with input and output dimensions R^{2c} and $R^{c'}$ respectively, and all other symbols are defined as in Eq. 4.1.

These two feature extractors are computationally quite similar; they compute local features and achieve permutation invariance using the symmetric max pooling function. The main difference lies in how they define locality. The PointNet++ layer groups points in the original Euclidean space whereas the EdgeConv layer groups points dynamically in the computed feature space. This allows the EdgeConv layers to have an effectively larger receptive field and compute potentially more descriptive local features, giving it the upper hand in terms of point cloud classification and segmentation performance.

4.1.3 Downsampling

In the original papers, the EdgeConv layer maintains the size of the point cloud while the PointNet++ layer downsamples point clouds by sampling a subset of the points with iterative furthest point sampling (FPS) and computing features for these points alone. Since our model predicts motion vectors for each point, downsampled features need to be up-

	Downsampling	No Downsampling
PointNet++	FlowNet3D [35], PN++ w/ DS	PN++ w/o DS
EdgeConv	EC w/ DS	EC w/o DS

Table 4.2: **Architecture classification.** Primary architectural differences between our proposed architectures and FlowNet3D [35].

sampled back to size of the original point cloud. [35] accomplishes this using the Set Upconv layer. Given a lower resolution point cloud $\{p_1, p_2, ..., p_{n'}\}, p_i \in R^3$ and its features $\{f'_1, f'_2, ..., f'_{n'}\}, f_i \in R^c$, as well as a previously computed higher resolution point cloud $\{p_1, p_2, ..., p_n\}, p_i \in R^3$ with its features $\{f_1, f_2, ..., f_n\}, f_i \in R^c$, the Set Upconv layer applies the PointNet++ operation to each point in the higher resolution point cloud by grouping the points from the lower resolution point cloud. These features are then further processed with an additional MLP. More precisely:

$$f_i^* = h_{\theta_2}(\max_{\substack{j| \, ||p_j - p_i|| \leq r, \\ p_j \in p'}} h_{\theta_1}(f_j, p_j - p_i), f_i) \tag{4.3}$$

where p' indicates the set of points in the lower resolution point cloud. At the final upsampling layer, [35] uses the Feature Propagation layer from [51], which replaces the first MLP with an inverse distance weighted average interpolation. We also adopt this upsampling strategy in our PointNet++ based architectures.

Although EdgeConv has not been used with downsampling, we also investigate this architecture configuration for point cloud prediction and develop novel downsampling and upsampling modules. To do this, we draw parallels between EdgeConv and the PointNet++ layer and design the sampling scheme in the spirit of these parallels. To downsample the points, rather than using FPS in Euclidean space, we compute it in the feature space. To upsample, we utilize the Set Upconv layer, but similarly, we group the points in the previously computed feature space rather than Euclidean space.

Downsampling is beneficial because it helps reduce the computational complexity of the network. However it also reduces the resolution of the features and creates ambiguity when upsampling. Thus networks that use downsampling need to be larger in order to resolve these ambiguities and have sufficient feature content at the architecture bottleneck. For a comparison between downsampling and non downsampling architectures, refer to Table 4.1.

4.1.4 Proposed Architectures

Modulating the feature extractor and sampling strategy of our framework results in four different architectures as seen in Table 4.2: PointNet++ with downsampling (PN++ w/ DS), PointNet++ without downsampling (PN++ w/o DS), EdgeConv with downsampling (EC w/ DS) and EdgeConv without downsampling (EC w/o DS). Specific architecture details are shown in Table 4.3. The radii of the ball queries in the PointNet++ models were chosen

Module	PN++ w/ DS	PN++ w/o DS	EC w/ DS	EC w/o DS
Feature Extraction 1	**PointNet++**: r = 0.5, SR=0.25×, mlp=[128,128]	**PointNet++**: r=0.7, SR=1×, mlp=[32,32]	**EdgeConv**: k=16, SR=0.25×, mlp=[128,128]	**EdgeConv**: k=16, SR=1×, mlp=[32,32]
Flow Embedding 1	**PointNet++**: r = 1.5, SR=1×, mlp=[128]	**PointNet++**: r=1, SR=1×, mlp=[32]	**EdgeConv**: k = 16, SR=1×, mlp=[128]	**EdgeConv**: k = 16, SR=1×, mlp=[32]
Feature Extraction 2	**PointNet++**: r = 1, SR=0.25×, mlp=[256,256]	**PointNet++**: r = 0.7, SR=1×, mlp=[64,64]	**EdgeConv**: k = 16, SR=0.25×, mlp=[256,256]	**EdgeConv**: k = 16, SR=1×, mlp=[64,64]
Flow Embedding 2	**PointNet++**: r = 3, SR=1×, mlp=[256]	**PointNet++**: r = 1, SR=1×, mlp=[64]	**EdgeConv**: k = 16, SR=1×, mlp=[256]	**EdgeConv**: k = 16, SR=1×, mlp=[64]
Feature Extraction 3	**PointNet++**: r = 2, SR=0.2×, mlp=[512]	**PointNet++**: r = 0.7, SR=1×, mlp=[128]	**EdgeConv**: k = 16, SR=0.2×, mlp=[512]	**EdgeConv**: k = 16, SR=1×, mlp=[128]
Refinement	**Upconv1**: k = 16, SR=5×, SS=XYZ, mlp1=[512], mlp2=[512] **Upconv2**: k = 16, SR=4×, SS=XYZ, mlp1=[512], mlp2=[512] **FeatProp**: SR=4×, SS=XYZ, mlp=[256] **mlp**: [256, 128, 3]	**mlp**: input = (feat1, feat2, feat3), widths = [512, 256, 128, 3]	**Upconv1**: k = 16, SR=5×, SS=flow2, mlp1=[512], mlp2=[512] **Upconv2**: k = 16, SR=4×, SS=flow1, mlp1=[512], mlp2=[512] **FeatProp**: SR=4×, SS=XYZ, mlp=[256] **mlp**: [256, 128, 3]	**mlp**: input = (feat1, feat2, feat3), widths = [512, 256, 128, 3]

Table 4.3: **Architecture details.** r = ball query radius, k used in KNN grouping, SR = sampling rate, SS = sampling space, feat and flow refer to the output of the corresponding layer.

to be commensurate to the receptive field of their EdgeConv counterparts parameterized by k. As shown in Table 4.3, we extend our architectures' choice of feature extractor to its flow embedding layers as well.

4.1.5 Loss Functions

For our loss, we need a function that measures the similarity between two point clouds P and Q. Following [19], we use Chamfer Distsance (CD) and Earth Mover's Distance (EMD):

$$L_{CD}(P,Q) = \frac{1}{2}(\sum_{p\in P} \min_{q\in Q} ||q-p||_2^2 + \sum_{q\in Q} \min_{p\in P} ||p-q||_2^2) \tag{4.4}$$

$$L_{EMD}(P,Q) = \min_{\phi\in P\to Q} \sum_{p\in P} ||p-\phi(p)||_2 \tag{4.5}$$

where ϕ indicates a bijection.

As in [19], we utilize a parallelizable approximation of the true EMD [6]. [1] notes that CD does not always remain true to finer, visual similarities between point clouds, and that when optimized, it may overpopulate regions where points are more likely to appear in the ground truth. EMD more accurately captures visual similarity; however, CD still works well at capturing coarser, structural similarities, and we find that is an easier function to optimize. In our loss function, we use a combination of both:

$$L(P,Q) = \alpha L_{CD}(P,Q) + \beta L_{EMD}(P,Q) \tag{4.6}$$

where α and β are parameters chosen with cross validation.

4.1.6 Training Details

There exist two popular approaches for training generative, temporal networks: curriculum approaches [5] and teacher-forcing [29]. In teacher-forcing, the model is trained using only ground truth inputs. However, this prevents the model from learning how to use its own predictions as inputs as it does during test time. Instead, we use a curriculum based approach by training the model with its own predictions as inputs. We first train the model to predict x^*_{t+1}, and once that converges, we train on x^*_{t+2}, feeding in our predicted x^*_{t+1} back into the model. This way, we slowly increase the difficulty as the model becomes capable of learning harder tasks. We repeat this process until the validation loss no longer decreases upon training the next time step.

We train our models on the nuScenes dataset [8], a recently released large scale autonomous driving dataset. It contains over 320,000 point clouds from rotating LiDAR scans captured at 20 Hz with over 34,000 points each. However, many of these points are detecting the roof of the ego vehicle, which is of little interest. In addition, the outer most points are extremely sparse and less relevant to potential downstream driving decisions. So we preprocess the point clouds by selecting an annular region of points between the 12,000th and

Model	CD (m^2)	EMD (m)	Model Size (MB)	Runtime (s)	Memory(MB)
Identity	.2472	34.88	-	-	-
FN3DOOB [35]	1.2084	91.68	14.9	.2946	**669.6**
FN3DA	.1399	33.94	14.9	.2946	**669.6**
PN++ w/ DS	**.1381**	**33.14**	22.1	.2635	783.5
PN++ w/o DS	.1813	37.63	**3.7**	.5079	1007.2
EC w/ DS	.1837	35.49	10.2	**.2591**	907.9
EC w/o DS	.1450	34.23	3.9	.6198	721.3

Table 4.4: **Accuracy and complexity of methods.** The table shows the average CD and EMD across the first 5 future frames, as well as the size, runtime, and memory usage of the models.

34,000th point from the origin. This adequately filters out the ego vehicle and fringes of the point cloud. We keep the point clouds in the original coordinate frame from the LiDAR scanner rather than of transforming them to the static global frame, choosing only to operate with the raw sensor data. However, an interesting avenue of future work could investigate how utilizing this transformation affects prediction accuracy.

Our models use leaky ReLU activations with slope 0.2 followed by batch normalization, except for the layer directly preceding the output. To train them we use the AdamW optimizer [37] with decoupled weight decay and L_2 regularization and employ a cosine annealing learning rate scheduler with restarts each time we advance to the next time step [38]. We chose values of 1 and 0.02 for α and β in the loss function, respectively. For $t+1$, we use a max learning rate of .001 and find that the models converge in 2 epochs, and for all other time steps we use a max learning rate of .0001 and find they typically converge after 1 epoch. We train until $t+3$, after which the loss no longer decreases.

One of the challenges of working with large point clouds is the computational cost. For $t+1$ we used a batch size of 4; however, training future time steps linearly increases the memory usage, dropping the batch size and making our batch normalization layers ineffective. To address this, we trained with regular batch normalization for $t+1$, but for future time steps we instead normalize our features using the learned, running estimates of the mean and variance from the first time step. When training on time steps beyond $t+1$, the weights of the network do not change as much as they do during the initial step. Thus, the learned statistics from $t+1$ are still adequate estimates beyond $t+1$, allowing us to train with smaller batch sizes while preserving accuracy.

4.2 Experiments

In this section we evaluate the performance of our models against several competitive baselines. We provide quantitative analysis on the accuracy and complexity of our model, as well as qualitative visualizations.

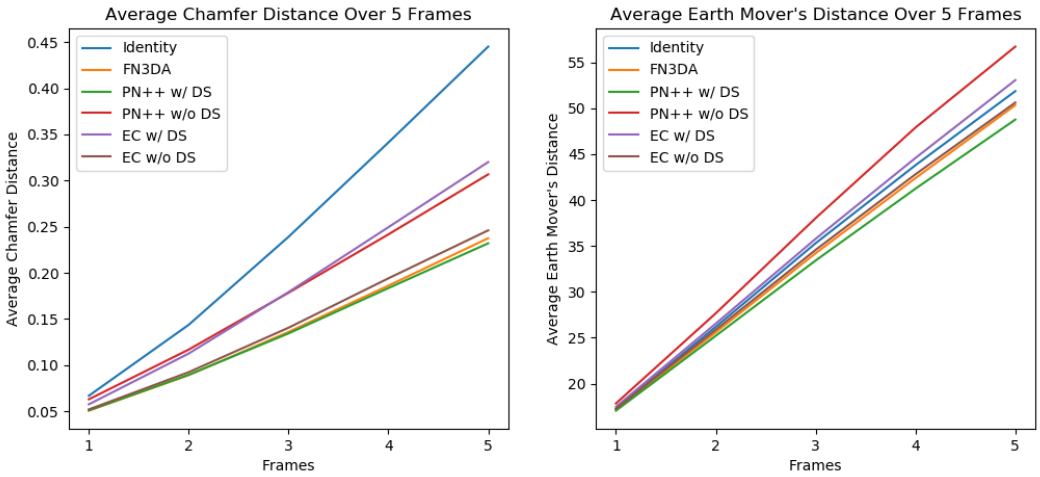


Figure 4.2: **Average Chamfer Distance and Earth Mover's Distance of our method and baselines.** Plotted over 5 frames. FN3DOOB was omitted to make the other methods more distinguishable.

4.2.1 Baselines

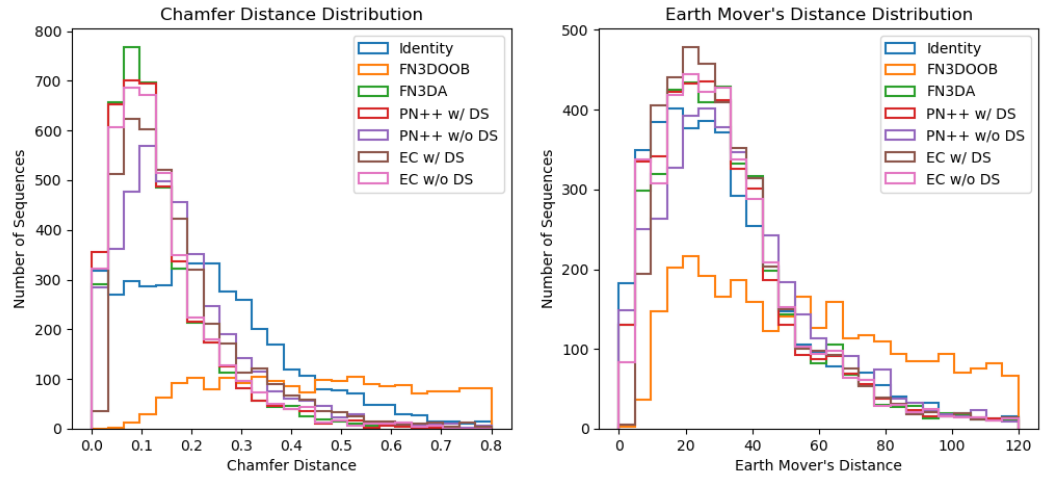
Identity Our first, and most naive baseline, is to use x_t as our prediction for x_{t+1}. We call this the identity baseline.

FlowNet3D Out of the Box (FN3DOOB) This baseline takes FlowNet3D [35] from the original paper trained on FlyingThings3D [42], computes the scene flow from x_t to x_{t-1}, and subtracts that from x_t to generate a prediction for x_{t+1}. Note that predicting the scene flow is not the same as predicting motion vectors that minimize the distance between two point clouds. Each point in the first point cloud plus its flow vector may not necessarily correspond to a point in the second point cloud; rather it results in the corresponding location of the first point in the time frame of the second point cloud. However, given dense enough point clouds, predicting scene flow approximates minimizing the distance between the two point clouds.

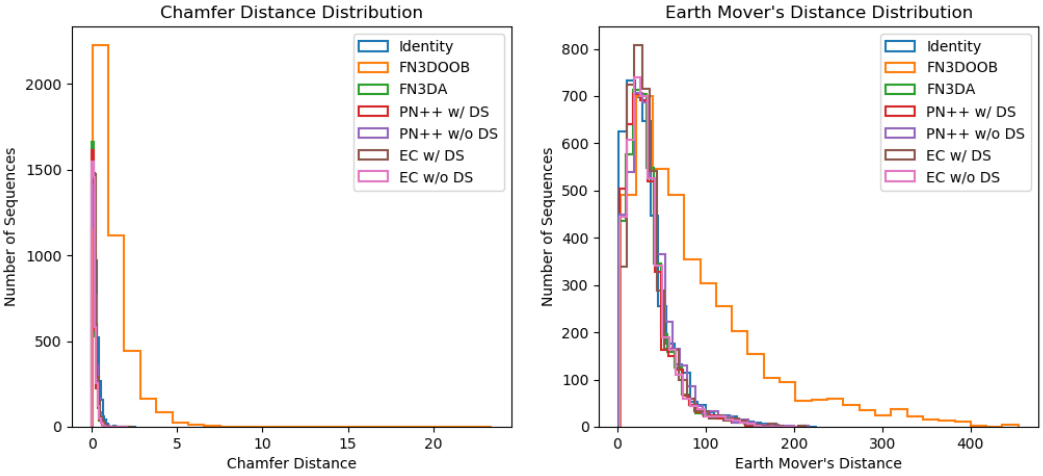
FlowNet3D Adapted (FN3DA) We also take the FlowNet3D [35] architecture and train it directly on our task. FlowNet3D only takes in two point clouds, so we train it on x_{t-1}, x_t to predict x_{t+1}. This network falls under the category of PointNet++ with downsampling and can be seen as the two frame version of PN++ w/ DS. We use the same training procedure that we use for our models.

4.2.2 Quantitative Results

We evaluate our models on the nuScenes test set, consisting of about 70,000 frames pre-processed the same way as our training data. To measure the accuracy of our predictions, we use the CD and EMD between the predicted point clouds and the ground truth. Our



(a) Outliers removed



(b) With outliers

Figure 4.3: **Distribution of Chamfer Distance and Earth Mover's Distance.** Histograms of errors for each method over 4000+ test samples. Each data point is the average of the error over a 5 frame sequence. The first two plots are have outliers removed and the other two show the entire distribution with the x-axis adjusted to range from 0 to the largest data point.

results are shown in Figure 4.2 and Table 4.4. FN3DOOB performs far worse than the other methods and skews the scale of Figure 4.2, so we omit it to make the plot more interpretable. Additionally, we visualize the distribution of errors for each method in Figure 4.3.

Among all the approaches, we find that FN3DA, PN++ w/ DS, and EC w/o DS achieve the lowest CD and EMD, with PN++ w/ DS performing marginally better. So for EdgeConv, not downsampling is important for strong performance, while for PointNet++, downsampling is actually more beneficial. We speculate this is due to the inherent difference in expressivity between the two architectures. The PointNet++ architecture benefits from downsampling because the memory efficiency allows it to utilize wider layers and learn more complex, hierarchical features. Indeed, with about the same memory consumption, our downsampling networks are 4 times wider than their non-downsampling counterparts. However, the EdgeConv architecture is already able to learn complex features on its own with smaller layers and therefore does not benefit as much from a larger network. On the other hand, the feature ambiguity caused by downsampling may be more harmful to EdgeConv, because resolving these ambiguities and upsampling in the feature space is more challenging than in Euclidean space.

Although PN++ w/o DS and EC w/ DS perform better than the identity baseline in terms of CD, they have slightly higher EMD values. This is likely due to EMD's strong correlation with visual similarity. While our deep networks may learn point cloud dynamics, they often exhibit artifacts and struggle to replicate the clean appearance of a raw LiDAR scan. nuScenes high frame rate results in smaller motion between each frame. This allows the identity baseline to have exceptionally low EMD, as it maintains the clean appearance of a raw point cloud while not being penalized harshly for neglecting the dynamics of the scene.

Lastly, we find that FN3DOOB is actually a destructive operation, increasing the CD and EMD more than the identity baseline. We believe this is due to the domain transfer from the synthetic FlyingThings3D dataset to real LiDAR scans, as well as the fundamental difference between the point cloud prediction and scene flow problem.

We also show the size, runtime, and memory usage of our models in Table 4.4. Our models are implemented with PyTorch and tested on an Nvidia Titan RTX. The runtime and max memory allocated values are acquired with a batch size of 1, predicting $t + 1$, on a point cloud with 22,000 points. The downsampling models tend to be significantly faster yet larger than their non downsampling counterparts.

Based on this, we conclude that PN++ w/ DS and EC w/o DS are the most viable models. They both demonstrate high accuracy, but offer different computational advantages. PN++ w/ DS's runtime is about 2x faster, while EC w/o DS's model size is about 6x smaller. So if inference speed is more important, PN++ w/ DS should be used, but if model size is more important, then EC w/o DS should be used.

While FN3DA performs competitively, PN++ w/ DS surpasses it in terms both speed and accuracy, so FN3DA offers no notable advantages. However, it achieves comparable accuracy to PN++ w/ DS while using only two frames, indicating that additional frames may only slightly improve performance. We speculate that conditioning on even more frames

would improve accuracy by negligible amounts.

4.2.3 Visualizations

We visualize our predictions for $t+1$ and $t+5$ on two scenes in Figures 4.4, 4.5, 4.6, 4.7. In the first scene shown in figures 4.4 and 4.5, the ego vehicle is driving past a truck parked next to a line of v-shaped columns. In the magnified portion of the visualization, the truck is in the top right, and the columns are along the left side. Among all the methods, PN++ w/ DS and EC w/o DS most accurately predict the dynamics of the truck and columns. The performance is corroborated in the error visualizations, which show green or yellow regions around the truck and columns for all the methods except for PN++ w/ DS and EC w/o DS. These methods instead show a nearly entirely purple point cloud, indicating close to 0 error.

In the second scene shown in figures 4.6 and 4.7, the ego vehicle is making a left turn at an intersection. At the top left of the the zoomed in view there is a van driving in front of the vehicle, while at the bottom left pedestrians cross the street behind it. This complex movement of the ego coordinate frame combined with the irregular geometries present make this second scene more challenging. PN++ w/ DS best predicts the position of the large barrier to the right of the vehicle, and while all of our methods seem to make reasonable predictions of the van and pedestrians, EC w/ DS seems to do it the most accurately. Interestingly, we noticed that EC w/ DS tends to preserve the orderly appearance of a raw LiDAR scan despite generally having higher EMD, as seen in figures 4.5 and 4.7.

While we only visualize two scenes here, we have done extensive qualitative testing on the entire nuScenes mini dataset (10 scenes) and have verified that on average, PN++ w/ DS and EC w/o DS outperform the other methods, corroborating our quantitative analysis.

4.2.4 Scene Flow Estimation

We also show that our model is able to reasonably estimate scene flow. In Figure 4.8, we visualize motion vectors predicted by our models and baselines for the same scene shown in Figure 4.4, specifically highlighting the columns and the truck. Because the ego vehicle is moving forward, the columns and truck have motion vectors pointing backward. PN++ w/ DS and EC w/o DS perform the best, producing smooth, accurate scene flow, whereas FN3DOOB and PN++ w/o DS exhibit incoherent flow, and FN3DA, EC w/o DS underestimate the magnitude, corroborating Figure 4.4. More flow visualizations in diverse scenarios are available in the supplementary material.

We would like to highlight that our self-supervised scene flow estimation is nontrivial. As mentioned before, minimizing the CD and EMD between two point clouds is not the same as minimizing scene flow. Directly learning self-supervised scene flow between time t and $t+1$ using point cloud similarity metrics would result in degenerate outputs where the predicted vectors merely connect the points. In fact, there are a number of recent papers in the literature that try to regularize this ill-conditioned problem by adding smoothness constraints or cycle consistency [69, 45]. Our work addresses this by instead utilizing prior

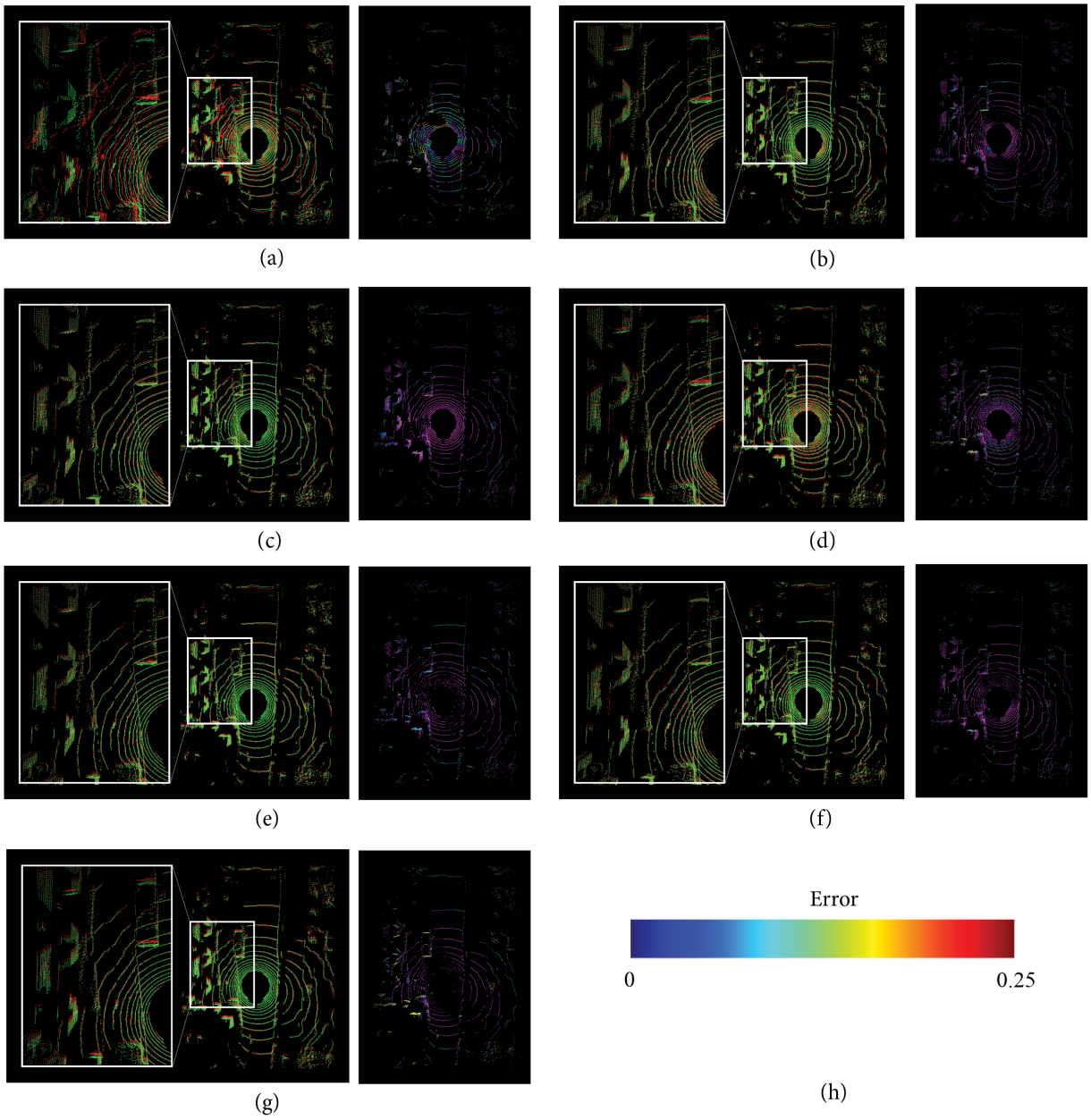


Figure 4.4: **Visualizations.** Error visualizations for (a) FN3DOOB, (b) FN3DA, (c) PN++ w/ DS, (d) PN++ w/o DS, (e) EC w/ DS, (f) EC w/o DS, (g) Identity on $t + 1$. For each method, the middle picture shows the ground truth in green and the prediction in red; the left picture zooms in on a region of interest in the middle picture; the right picture shows the squared distance between each point in the prediction and its nearest neighbor in the ground truth point cloud, with the color bar in (h) indicating the scale of the error. Refer to the identity visualization for scene context.

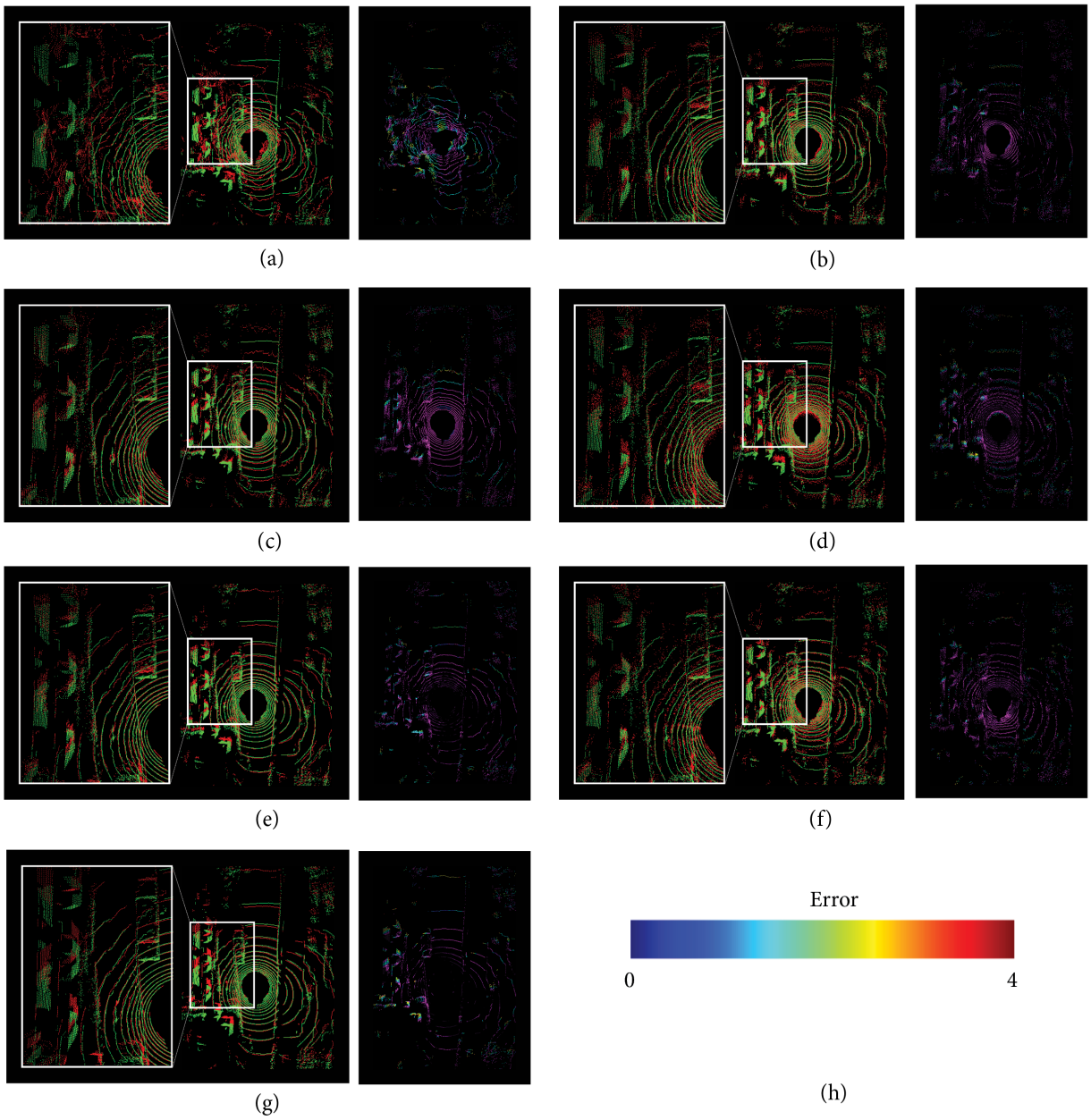


Figure 4.5: Visualization of predictions for $t+5$ on the same scene shown in Figure 4.4. (a) FN3DOOB, (b) FN3DA, (c) PN++ w/ DS, (d) PN++ w/o DS, (e) EC w/ DS, (f) EC w/o DS, (g) Identity, (h) error scale.

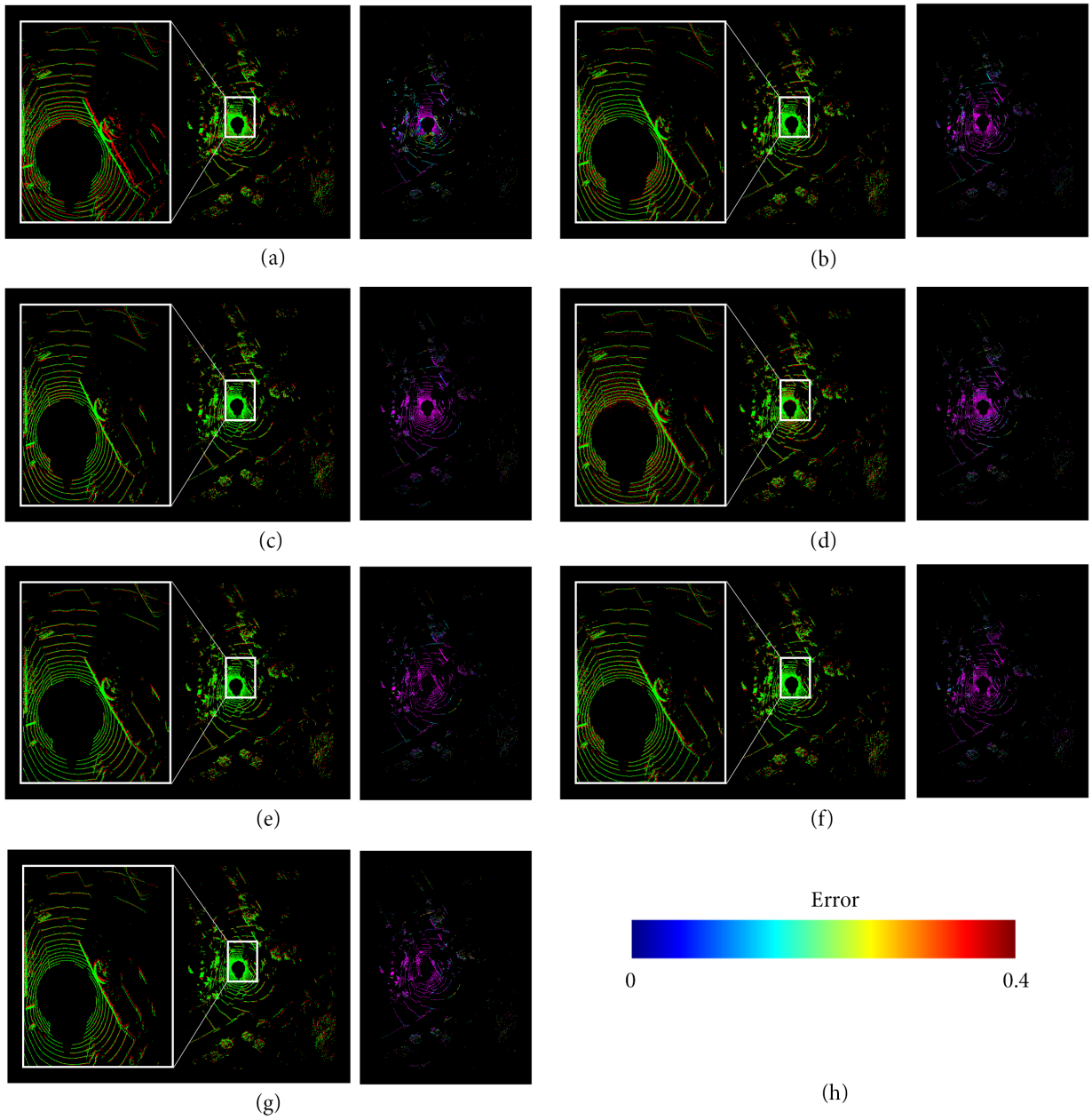


Figure 4.6: Visualization of predictions for $t + 1$ on an additional scene. (a) FN3DOOB, (b) FN3DA, (c) PN++ w/ DS, (d) PN++ w/o DS, (e) EC w/ DS, (f) EC w/o DS, (g) Identity, (h) error scale.

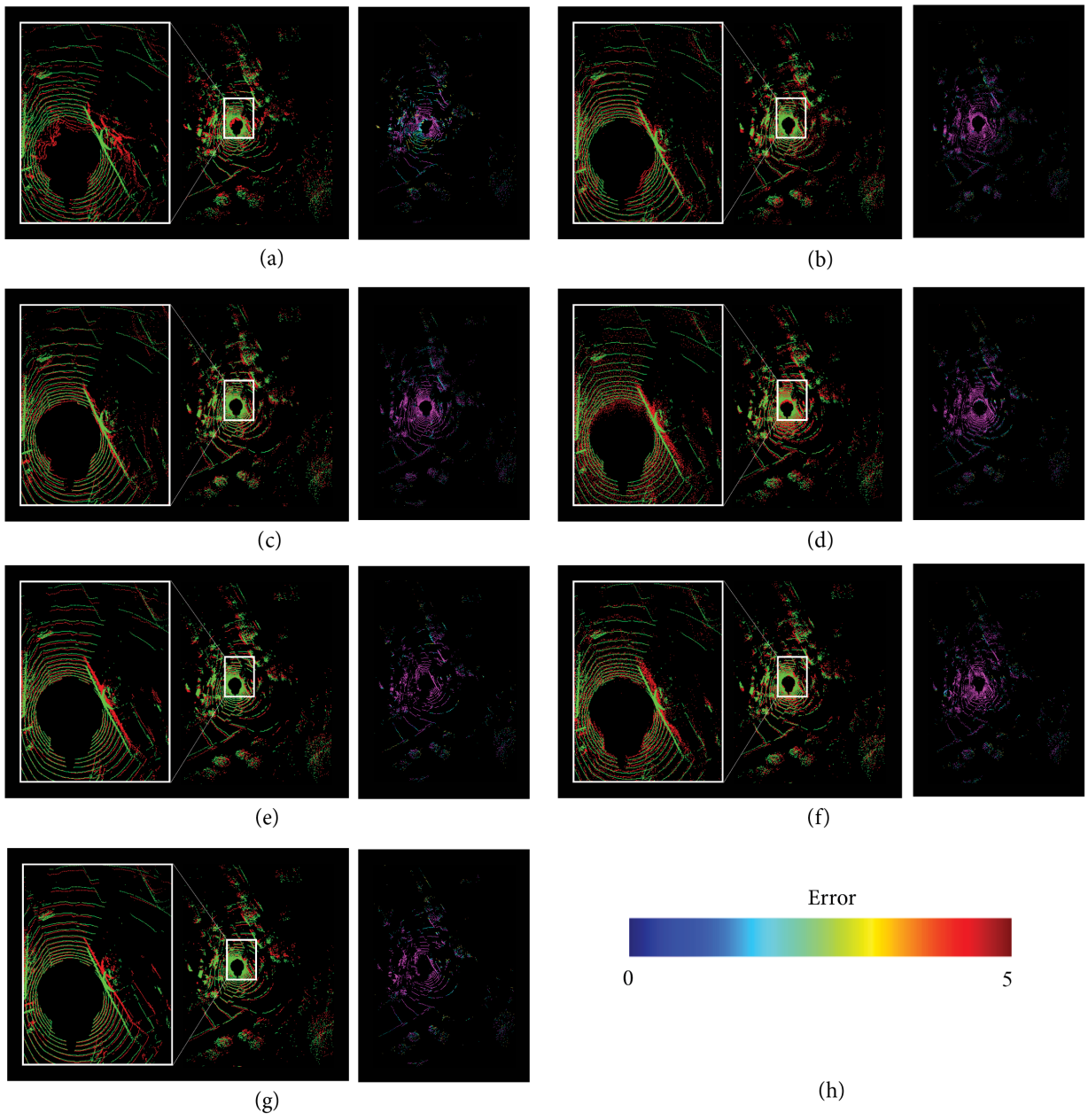


Figure 4.7: Visualization of predictions for $t+5$ on the same scene shown in Figure 4.6. (a) FN3DOOB, (b) FN3DA, (c) PN++ w/ DS, (d) PN++ w/o DS, (e) EC w/ DS, (f) EC w/o DS, (g) Identity, (h) error scale.

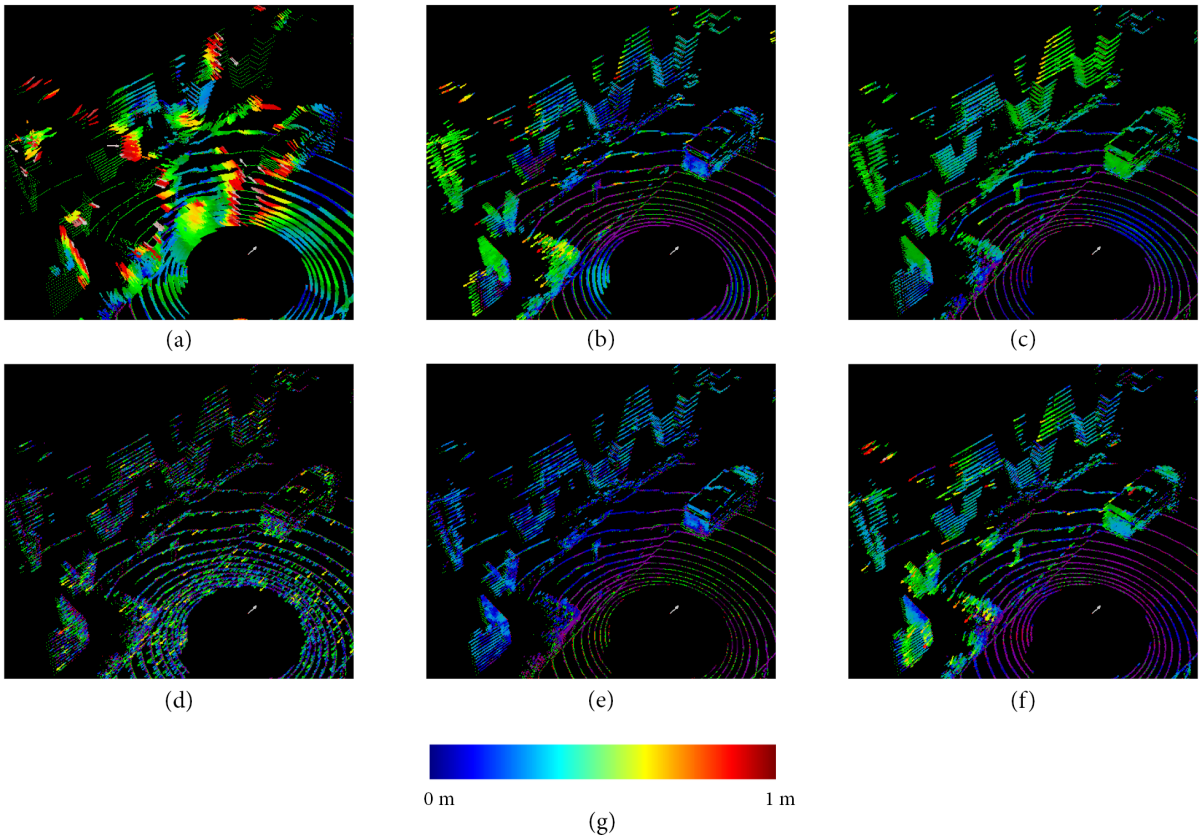


Figure 4.8: **Flow visualization**. Predicted motion vectors for (a) FN3DOOB, (b) PN++ w/ DS, (c) EC w/ DS, (d) FN3DA, (e) PN++ w/o DS, (f) EC w/o DS. The arrows in the visualization indicate our model's predicted motion vectors, and the color corresponds to the magnitude, as indicated by the color bar in (g). Vectors beyond the range of the color bar are omitted.

frames. Because the network is no longer given $t + 1$, it cannot produce the degenerate solution. However, it is still given sufficient information on the scene's dynamics in the prior frames to predict scene flow. Therefore, besides FN3DOOB, which is a trivial extension of FlowNet3D, the remaining approaches we describe are novel in the sense that they regularize the self-supervised scene flow problem in a new way. Here, we only qualitatively evaluate our estimated scene flow as a proof of concept; however, future work could build on this idea and produce more rigorous, quantitative analysis.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this report, we introduce the first scene flow estimation approach that exploits the multi-body rigidity of dynamic scenes without requiring annotated labels. Our approach achieves state of the art performance on the KITTI and nuScenes Scene Flow Benchmark and achieves strong results on the SemanticKITTI dataset for moving object segmentation and ego-motion estimation. Additionally, we explore the task of point cloud prediction by designing a novel class of neural network architectures and training framework. We show that our top models (PN++ w/ DS and EC w/o DS) can generate convincing predictions of future point clouds, and that they are competitive with several strong baselines. Our visualizations help verify our findings and indicate that our models can be used to produce scene flow approximations. Our work has the potential to be applied to numerous downstream tasks such as object tracking and vehicle control. Additionally, neither of our projects require any annotations or labels, and with the growing prevalence of depth sensors and LiDAR, our work will be useful for processing the raw, unlabelled point clouds they generate.

5.2 Future Work

In the future, we hope to improve upon our prediction method by incorporating additional terms like a perceptual loss or smoothness constraint into our self-supervised loss function. For our scene flow estimation approach, we hope to combine our approach with learning for real time inference, and to generalize to arbitrary 3D scenes. Lastly, for both methods there is the potential route improving performance via fusion with camera data.

Bibliography

[1] Panos Achlioptas et al. "Learning Representations and Generative Models For 3D Point Clouds". In: *arXiv preprint arXiv:1707.02392* (2017).

[2] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". In: *arXiv preprint arXiv:1803.01271* (2018). arXiv: 1803.01271 [cs.LG].

[3] Stefan Baur et al. "SLIM: Self-Supervised LiDAR Scene Flow and Motion Segmentation". In: *International Conference on Computer Vision (ICCV)*. 2021.

[4] J. Behley et al. "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences". In: *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*. 2019.

[5] Yoshua Bengio et al. "Curriculum Learning". In: *International Conference on Machine Learning (ICML)* (2009).

[6] D. P. Bertsekas. "A distributed asynchronous relaxation algorithm for the assignment problem". In: *1985 24th IEEE Conference on Decision and Control*. 1985, pp. 1703–1704.

[7] Paul J Besl and Neil D McKay. "Method for registration of 3-D shapes". In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–606.

[8] Holger Caesar et al. "nuScenes: A multimodal dataset for autonomous driving". In: *Computer Vision and Pattern Recognition (CVPR)* (2020).

[9] X. Chen et al. "Moving Object Segmentation in 3D LiDAR Data: A Learning-based Approach Exploiting Sequential Data". In: *IEEE Robotics and Automation Letters (RA-L)* 6 (4 2021), pp. 6529–6536. ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3093567. URL: http://www.ipb.uni-bonn.de/pdfs/chen2021ral-iros.pdf.

[10] Christopher Choy, JunYoung Gwak, and Silvio Savarese. "4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3075–3084.

[11] Alexey Dosovitskiy et al. "Flownet: Learning optical flow with convolutional networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2758–2766.

[12] Hehe Fan and Yi Yang. "PointRNN: Point recurrent neural network for moving point cloud processing". In: *arXiv preprint arXiv:1910.08287* (2019).

[13] Hehe Fan, Yi Yang, and Mohan Kankanhalli. "Point 4D Transformer Networks for Spatio-Temporal Modeling in Point Cloud Videos". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*. 2021.

[14] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[15] Andreas Geiger et al. "Vision meets robotics: The KITTI dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[16] Zan Gojcic et al. *Weakly Supervised Learning of Rigid 3D Scene Flow*. 2021.

[17] Xiuye Gu et al. "Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3254–3263.

[18] Frederic Devernay Diana Mateus Matthieu Guilbert. "Multi-Camera Scene Flow by Tracking 3-D Points and Surfels". In: *Computer Vision and Pattern Recognition (CVPR)* (2006).

[19] Leonidas Guibas Hao Su Haoqiang Fan. "A Point Set Generation Network for 3D Object Reconstruction from a Single Image". In: *Computer Vision and Pattern Recognition (CVPR)* (2017).

[20] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[21] John Houston et al. "One Thousand and One Hours: Self-driving Motion Prediction Dataset". In: *CoRL* (2020).

[22] Frédéric Huguet and Frédéric Devernay. "A variational method for scene flow estimation from stereo sequences". In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE. 2007, pp. 1–7.

[23] Junhwa Hur and Stefan Roth. "Self-Supervised Monocular Scene Flow Estimation". In: *CVPR* (2020). arXiv: 2004.04143. URL: https://arxiv.org/abs/2004.04143.

[24] Junhwa Hur and Stefan Roth. "Self-Supervised Multi-Frame Monocular Scene Flow". In: *CVPR*. 2021.

[25] Eddy Ilg et al. "Flownet 2.0: Evolution of optical flow estimation with deep networks". In: *IEEE conference on computer vision and pattern recognition (CVPR)*. Vol. 2. 2017, p. 6.

[26] Eddy Ilg et al. "Occlusions, Motion and Depth Boundaries with a Generic Network for Disparity, Optical Flow or Scene Flow Estimation". In: *ECCV* (2018). arXiv: 1808.01838. URL: http://arxiv.org/abs/1808.01838.

[27] Mariano Jaimez et al. "A primal-dual framework for real-time dense RGB-D scene flow". In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 98–104.

[28] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[29] Alex Lamb et al. "Professor Forcing: A New Algorithm for Training Recurrent Networks". In: *NIPS* (2016).

[30] Alex H. Lang et al. "PointPillars: Fast Encoders for Object Detection from Point Clouds". In: (2019).

[31] Jake Levinson et al. "An Analysis of SVD for Deep Rotation Estimation". In: *Neural Information Processing Systems (NeurIPS)* (2020).

[32] Ruibo Li et al. "HCRF-Flow: Scene Flow from Point Clouds with Continuous High-order CRFs and Position-aware Flow Embedding". In: *Computer Vision and Pattern Recognition (CVPR)* (2021).

[33] Xueqian Li, Jhony Kaesemodel Pontes, and Simon Lucey. "Neural Scene Flow Prior". In: *Advances in Neural Information Processing Systems* 34 (2021).

[34] Yangyan Li et al. "PointCNN: Convolution On X-Transformed Points". In: *Neural Information Processing Systems (NeurIPS)* (2018).

[35] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. "Flownet3d: Learning scene flow in 3d point clouds". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 529–537.

[36] Xingyu Liu, Mengyuan Yan, and Jeannette Bohg. "Meteornet: Deep learning on dynamic 3d point cloud sequences". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 9246–9255.

[37] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations (ICLR)* (2019).

[38] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017.

[39] William Lotter, Gabriel Kreiman, and David Cox. *Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning*. 2016. eprint: arXiv:1605.08104.

[40] Wei-Chiu Ma et al. "Deep Rigid Instance Scene Flow". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3614–3622.

[41] D. Maturana and S. Scherer. "VoxNet: A 3D Convolutional Neural Network for real-time object recognition". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 922–928.

[42] Nikolaus Mayer et al. "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4040–4048.

[43] Moritz Menze and Andreas Geiger. "Object scene flow for autonomous vehicles". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3061–3070.

[44] 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.

[45] Himangi Mittal, Brian Okorn, and David Held. "Just Go with the Flow: Self-Supervised Scene Flow Estimation". In: *Computer Vision and Pattern Recognition (CVPR)* (2020).

[46] Himangi Mittal, Brian Okorn, and David Held. "PointFlowNet: Learning Representations for Rigid Motion Estimation from Point Clouds". In: *Computer Vision and Pattern Recognition (CVPR)* (2019).

[47] Andreas Geiger Philip Lenz Julius Ziegler and Martin Roser. "Sparse Scene Flow Segmentation for Moving Object Detection in Urban Environments". In: *2011 IEEE Intelligent Vehicles Symposium (IV)* (2011).

[48] Jhony Kaesemodel Pontes, James Hays, and Simon Lucey. "Scene Flow from Point Clouds with or without Learning". In: *International Conference on 3D Vision (3DV)* (2020).

[49] Gilles Puy, Alexandre Boulch, and Renaud Marlet. "FLOT: Scene Flow on Point Clouds Guided by Optimal Transport". In: *European Conference on Computer Vision*. 2020.

[50] Charles R Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *Computer Vision and Pattern Recognition (CVPR)* (2017).

[51] Charles R. Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Neural Information Processing Systems (NeurIPS)* (2017).

[52] Ahmad El Sallab et al. "YOLO4D: A Spatio-temporal Approach for Real-time Multi-object Detection and Classification from LiDAR Point Clouds". In: *Neural Information Processing Systems (NeurIPS) Workshop MLITS* (2018).

[53] René Schuster et al. "Combining Stereo Disparity and Optical Flow for Basic Scene Flow". In: *CoRR* abs/1801.04720 (2018). arXiv: `1801.04720`. URL: `http://arxiv.org/abs/1801.04720`.

[54] Lin Shao et al. "Motion-based Object Segmentation based on Dense RGB-D Scene Flow". In: *IEEE Robotics and Automation Letters* (2018).

[55] Deqing Sun et al. "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8934–8943.

[56] Pei Sun et al. *Scalability in Perception for Autonomous Driving: Waymo Open Dataset*. 2019. arXiv: 1912.04838 [cs.CV].

[57] Hugues Thomas et al. "KPConv: Flexible and Deformable Convolution for Point Clouds". In: *Proceedings of the IEEE International Conference on Computer Vision* (2019).

[58] Salti Tombari and Di Stefano. "Unique Signatures of Histograms for Local Surface Description". In: *European Conference on Computer Vision (ECCV)* (2010).

[59] Arash K. Ushani et al. "A learning approach for real-time temporal scene flow estimation from LIDAR data". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 5666–5673. DOI: 10.1109/ICRA.2017.7989666.

[60] Ashish Vaswani et al. "Attention is All You Need". In: 2017. URL: https://arxiv.org/pdf/1706.03762.pdf.

[61] Sundar Vedula et al. "Three-dimensional scene flow". In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. Vol. 2. IEEE. 1999, pp. 722–729.

[62] Christoph Vogel, Konrad Schindler, and Stefan Roth. "3D scene flow estimation with a piecewise rigid scene model". In: *International Journal of Computer Vision* 115.1 (2015), pp. 1–28.

[63] Christoph Vogel, Konrad Schindler, and Stefan Roth. "3D scene flow estimation with a rigid motion prior". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 1291–1298.

[64] Christoph Vogel, Konrad Schindler, and Stefan Roth. "Piecewise rigid scene flow". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 1377–1384.

[65] Yue Wang et al. "Dynamic Graph CNN for Learning on Point Clouds". In: *ACM Transactions on Graphics (TOG)* (2019).

[66] Zirui Wang et al. "FlowNet3D++: Geometric losses for deep scene flow estimation". In: *The IEEE Winter Conference on Applications of Computer Vision*. 2020, pp. 91–98.

[67] Andreas Wedel et al. "Efficient dense scene flow from sparse or dense stereo data". In: *European conference on computer vision*. Springer. 2008, pp. 739–751.

[68] Xinshuo Weng et al. *Unsupervised Sequence Forecasting of 100,000 Points for Unsupervised Trajectory Forecasting*. 2020. eprint: arXiv:2003.08376.

[69] Wenxuan Wu et al. "PointPWC-Net: A Coarse-to-Fine Network for Supervised and Self-Supervised Scene Flow Estimation on 3D Point Clouds". In: *European Conference on Computer Vision (ECCV)* (2020).

[70] Dan Raviv Yair Kittenplon Yonina C. Eldar. "FlowStep3D: Model Unrolling for Self-Supervised Scene Flow Estimation". In: *Computer Vision and Pattern Recognition (CVPR)* (2021).

[71] Yan Yan, Yuxing Mao, and Bo Li. "SECOND: Sparsely Embedded Convolutional Detection". In: *Sensors* 18.10 (2018). URL: https://www.mdpi.com/1424-8220/18/10/3337.

[72] Gengshan Yang and Deva Ramanan. "Learning to Segment Rigid Motions from Two Frames". In: *CVPR*. 2021.

[73] Gengshan Yang and Deva Ramanan. "Upgrading Optical Flow to 3D Scene Flow through Optical Expansion". In: *CVPR*. 2020.

[74] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. "Center-based 3D Object Detection and Tracking". In: *CVPR* (2021).

[75] Jason Y. Zhang et al. "Predicting 3D Human Dynamics from Video". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2019.

[76] Tinghui Zhou et al. "Unsupervised learning of depth and ego-motion from video". In: *arXiv preprint arXiv:1704.07813* (2017).

[77] Yin Zhou and Oncel Tuzel. "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection". In: *arXiv preprint arXiv:1711.06396* (2017). arXiv: 1711.06396 [cs.CV].