2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

PillarFlowNet: A Real-time Deep Multitask Network for LiDAR-based 3D Object Detection and Scene Flow Estimation

Fabian Duffhauss[1] and Stefan A. Baur[2]

Abstract— **Mobile robotic platforms require a precise understanding about other agents in their surroundings as well as their respective motion in order to operate safely. Scene flow in combination with object detection can be used to achieve this understanding. Together, they provide valuable cues for behavior prediction of other agents and thus ultimately are a good basis for the ego-vehicle's behavior planning algorithms. Traditionally, scene flow estimation and object detection are handled by separate deep networks requiring immense computational resources. In this work, we propose PillarFlowNet, a novel method for simultaneous LiDAR scene flow estimation and object detection with low latency and high precision based on a single network. In our experiments on the KITTI dataset, PillarFlowNet achieves a 16.3 percentage points higher average precision score as well as a 21.4 % reduction in average endpoint error for scene flow compared to the state-of-the-art in multitask LiDAR object detection and scene flow estimation. Furthermore, our method is significantly faster than previous methods, making it the first to be applicable for real-time systems.**

I. Introduction

This paper targets the problem of multitask learning for 3D object detection and scene flow estimation using LiDAR data in a single deep network as depicted in Figure 1. 3D scene flow associates a displacement vector to each point in a point cloud, propagating it forward to its corresponding location in the consecutive point cloud. Estimating 3D scene flow using LiDAR data is no trivial task: The inherent sparsity of measured points in 3D space renders the problem of scene flow estimation ill-posed, as there are practically no pointwise one-to-one correspondences present in two consecutive point clouds. Instead, scene flow must be inferred from the underlying motion of objects in a scene, irrespective of whether a displacement between two point clouds is caused by the sensor's ego motion or whether it is caused by a moving agent.

Recently, the research community's interest in LiDAR object detection and LiDAR scene flow estimation has grown but little work has been conducted towards unifying those tasks, i.e. solving them with a single network. To the best of our knowledge, there exists only one previous method for multitask learning for these tasks using LiDAR data, i.e. the work by Behl et al. [1]. Their network relies on computationally expensive 3D convolutions in order to find correspondences over the space and time domain, rendering their network architecture too slow for real-time applications.

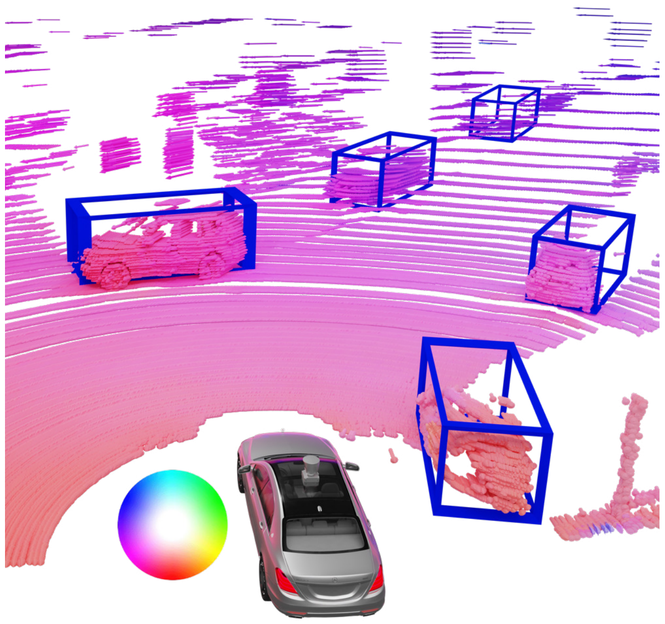

Fig. 1. PillarFlowNet detects objects and predicts scene flow at the same time using LiDAR data only. This scene shows the ego vehicle performing a sharp right turn. For color coding, the predicted flow vectors are projected onto the ground plane and colored according to the standard HSV wheel on the left.

We propose a novel network utilizing a different representation, which is not only significantly faster but also much more accurate in both tasks, outperforming their network in predicting LiDAR scene flow and objects at the same time. It relies on a pillar feature representation in combination with efficient 2D convolutions. This gain in speed makes it the first LiDAR object detection and scene flow multitask network suitable for systems with real-time constraints.

Pointwise scene flow in combination with object detection is more generic than object tracking approaches, as motion of objects that were not explicitly detected can still be reasoned about. Traditional approaches using object detection in combination with tracking have advantages when it comes to stability of detections and vehicle tracks. However, tracking cannot capture motion in cases where no vehicle has been detected. Additionally, tracking frameworks need time to initialize and update their motion model in order to predict a meaningful state of a tracked object. In contrast to that, our network computes meaningful motion estimates even for objects it has not been specifically trained for as shown in Figure 2.

[1]Fabian Duffhauss is with the Bosch Center for Artificial Intelligence, Renningen, Germany. Fabian.Duffhauss@de.Bosch.com

[2]Stefan A. Baur is with the Mercedes-Benz AG, Stuttgart, Germany. stefan_andreas.baur@daimler.com

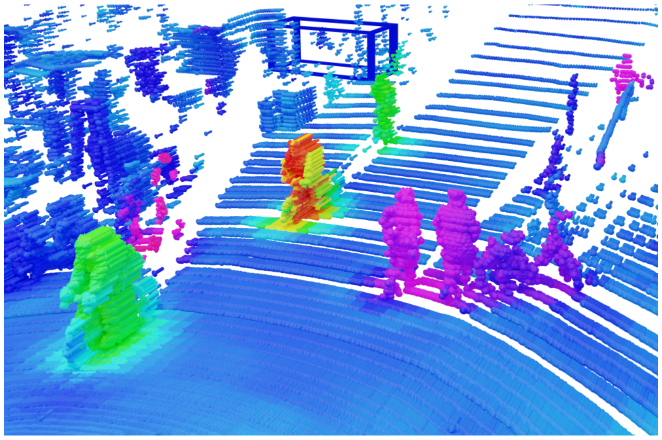

Fig. 2. PillarFlowNet manages to capture motion at a very detailed level. Color-coding the predicted flow vectors by length makes pedestrians and cyclists easily distinguishable from vehicles and static objects.

Our main contributions are:

- We propose PillarFlowNet, a novel end-to-end trainable network for simultaneous LiDAR object detection and scene flow estimation.
- Our method significantly improves multitask LiDAR scene flow estimation and object detection accuracy compared to the state-of-the-art.
- PillarFlowNet is the first multitask LiDAR scene flow and object detection network to achieve real-time performance.

II. Related Work

Our method lies at the intersection of multitask learning, LiDAR object detection, and scene flow estimation. The following paragraphs present relevant literature for all three categories.

A. LiDAR Scene Flow

Over the years, multiple suitable data representations have been established for LiDAR scene flow estimation.

Point set based methods: FlowNet3D by Liu et al. [2] operates on pre-filtered sets of points, MeteorNet by Liu et al. [3] uses MeteorModules computing pointwise features for points and their neighbors. Features and spatiotemporal differences of neighboring points are passed into a feature encoder inspired by PointNet [4], aggregated recursively, and shared using max pooling and feature concatenation.

Graph-based methods: Dewan et al. [5] were among the first to research estimating LiDAR scene flow using energy minimization on a factor graph representation of consecutive point clouds. Very recently, Gu et al. proposed HPLFlowNet [6], capable of estimating LiDAR scene flow on up to 86k points.

Point set based and graph-based methods have in common that they tend to work well on small point sets. The largest point cloud any of these approaches report to have successfully processed is [6] with 86k points, while Liu et al. [2] stop reporting runtimes after 8k points ($325\,\mathrm{ms}$), which is significantly smaller than a real point cloud recorded from a common LiDAR sensor: A Velodyne HDL64 LiDAR records 128k points per revolution at $10\,\mathrm{Hz}$. All of these methods circumvent this problem by removing ground points in a preprocessing step, which has several disadvantages: Besides costing precious runtime, information is lost which can be fatal, especially in cases where the ground segmentation gives false results.

Grid-based methods: Ushani et al. [7], [8] were the first to apply machine learning to LiDAR scene flow estimation, inserting point clouds with removed static background into 3D occupancy grids. Scene flow is estimated using binary classifiers in order to find matches of feature columns between consecutive occupancy grids. More recent work tackles LiDAR scene flow estimation using Deep Learning. Vaquero et al. [9] and Baur et al. [10] use range images as data representation, projecting point clouds into 2D grids and applying 2D convolutions to estimate flow. Wang et al. [11] use a ResNet-50 architecture [12] applying 3D convolutions on an occupancy grid predicting a voxel flow map. They introduced Deep Parametric Continuous Convolutional (DPCC) layers which are then used as refinement on the 3D occupancy flow map. DPCC layers employ the Euclidean space directly as support domain with multilayer perceptrons (MLPs) as kernel functions. Compared to other publications such as [1], [3], [2], [6], [9], [10], [13] they report exceptionally good scene flow precision, which is why we chose to implement their approach as a baseline in section VI.

B. LiDAR Object Detection

For an excellent survey over LiDAR object detection methods the interested reader is referred to [14]. Most relevant for this work are **grid-based methods**. Zhou et al. introduced VoxelNet [15] which first subdivides the point cloud into voxels and then applies Voxel Feature Encoding (VFE) (based on PointNet [4]) to each voxel, performing hierarchical feature encoding, the result of which is then aggregated to a 3D tensor. After some middle convolutional layers the resulting tensors are ultimately fed into a Region Proposal Network (RPN) [16] which outputs a map of objectness probability scores and a regression map for bounding box parameters. Based on VoxelNet [15], Yan et al. introduced more efficient sparse convolutions in their network SECOND [17].

Most relevant for this paper in terms of object detection is the work conducted by Lang et al. [18], who proposed PointPillars, a novel deep neural network based on SECOND [17]. PointPillars splits the input point cloud into vertical 3D columns (pillars) and utilizes PointNet [4] to learn features from each pillar with its points. The features are encoded in a pseudo image enabling the use of an efficient 2D object detection backbone network without requiring 3D convolutions. This allows PointPillars to predict 3D bounding boxes at very high speed.

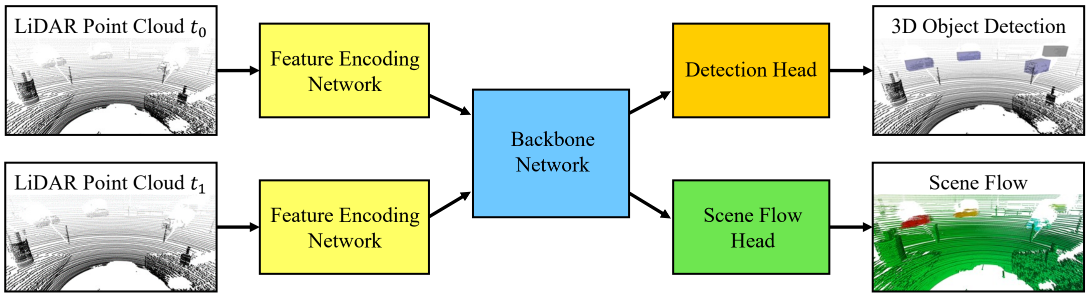

Fig. 3. Structure of our proposed multitask architecture PillarFlowNet. The network works directly on two consecutive raw point clouds and predicts 3D bounding boxes as well as scene flow for all points in the scene.

C. Multitask Learning

Multitask learning attempts to learn multiple tasks simultaneously with the goal of obtaining more general models. While there is a multitude of works that deal with multitask learning in general [19], [20], [21], [22], [23] or multitask learning specifically for LiDAR applications [24], [4], [25], [26] there exists only one paper that deals with simultaneous scene flow estimation and object detection using LiDAR point clouds: PointFlowNet by Behl et al. [1] subdivides two point clouds into separate voxel grids and uses Siamese VFE layers [15] as feature encoder to compute independent feature maps for each one of the consecutive point clouds. These two feature maps are then stacked and fed into a context encoder, which applies vertical downsampling using three strided 3D convolution layers, enforcing a common data representation for all tasks. After this, the network is split into different branches, predicting 3D scene flow for each voxel, ego motion, and object classification scores along with residuals for the box proposals. They evaluate scene flow accuracy of their network on a subset of the KITTI object detection dataset [27], comparing it to multiple baseline methods.

The advantages of our method over the method by Behl et al. [1] are mainly twofold: First, our network is significantly more accurate in both object detection and scene flow estimation. Second, our network is more than twice as fast as theirs, making it the first LiDAR scene flow and object detection multitask network capable of running in real-time. This is mainly due to using a different, more efficient data representation, enabling us to rely on 2D convolutions instead of computationally expensive 3D convolutions.

III. PillarFlowNet Architecture

PillarFlowNet takes two consecutively recorded LiDAR point clouds (each accumulated over 360°) and predicts oriented 3D bounding boxes for cars and vans as well as a 3D scene flow vector for each point in the first point cloud. Figure 3 shows the overall structure of the network. The architecture is subdivided into three parts: (A.) two feature encoding networks for creating representations of the two input point clouds, (B.) a convolutional backbone network that learns a shared representation, and (C.) two output heads for 3D bounding box classification and regression as well as scene flow estimation.

A. Feature Encoding Network

In order to encode both input point clouds to be processed efficiently, we use two identical learning-based feature encoding networks whose structure is illustrated in Figure 4. It is based on the pillar feature network introduced in PointPillars by Lang et al. [18]. The feature encoder subdivides the input point cloud into equally spaced pillars. Empty pillars are discarded, so that a variable number of pillars P is further processed. If a pillar comprises more than $T = 100$ points, a random subset is kept. All remaining points in each pillar are augmented using Cartesian coordinates relative to the pillar's centroid and relative to the pillar's center, creating a tensor with $D = 9$ dimensions as described in [18]. Using a linear layer, a batch normalization layer, and a rectified linear activation function (ReLU), a representation of the augmented point cloud tensor is learned. A maximum operation is applied to the point features in each pillar in order to extract only the most dominant features. The elements of the resulting tensor are scattered into a 3D tensor at their original pillar locations.

B. Backbone Network

The two pseudo images of the feature encoders are further processed by a convolutional backbone network (Figure 5). The backbone network concatenates the pseudo images and applies strided 2D convolutions to create a compact shared representation. Features from differently sized tensors are upsampled using transposed 2D convolutional layers, combining high resolution local features with coarser, more globally aggregated features in a subsequent concatenation. This combines semantic with positional information. While the full pillar resolution is maintained for scene flow estimation, for object detection only half of the resolution proved to be sufficient. For more details regarding number of layers and tensor sizes refer to Figure 5.

C. Output Heads

The concatenated feature tensors are further processed by 2D convolutions for object classification, bounding box regression, and 3D scene flow estimation as shown on the right side of Figure 5. For scene flow estimation, a 3D velocity vector is regressed for each pillar directly. For 3D object detection, a region proposal network with fixed size anchor boxes of two orientations as in [15] is applied. For

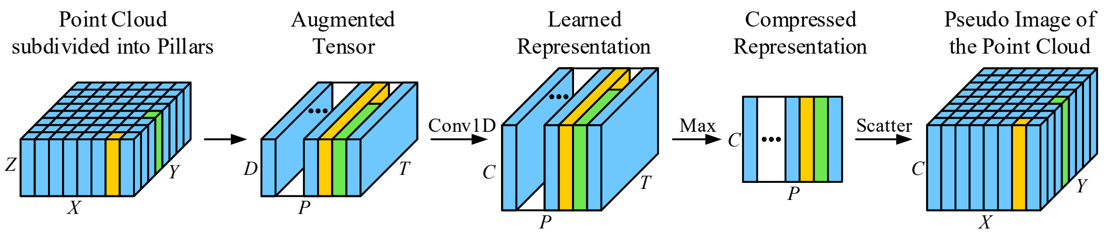
Point Cloud subdivided into Pillars
Augmented Tensor
Learned Representation
Compressed Representation
Pseudo Image of the Point Cloud
Conv1D
Max
Scatter
Z
X
Y
D
P
T
C
P
T
C
P
C
X
Y

Fig. 4. Feature encoding network. Given a raw point cloud it learns a representation that can be further processed efficiently without 3D convolutions.

each anchor box, the regression head predicts the probability that an object is located within it. The regression head predicts the deviations for each anchor box to the actual object using the same encoding as [15], [17]:

$$b = (\Delta x^{\mathrm{p}}, \Delta y^{\mathrm{p}}, \Delta z^{\mathrm{p}}, \Delta l^{\mathrm{p}}, \Delta w^{\mathrm{p}}, \Delta h^{\mathrm{p}}, \Delta\theta^{\mathrm{p}}) \in \mathbb{R}^7. \qquad (1)$$

IV. Implementation Details

In this section, we present details about our implementation, such as hyperparameters and the loss function.

A. Network Details

For object detection, we use anchor boxes with $3.9\,\mathrm{m}$ in length, $1.6\,\mathrm{m}$ in width, and $1.56\,\mathrm{m}$ in height centered at $z = -1\,\mathrm{m}$. We use pillars with a square base of $0.2\,\mathrm{m}$ in width and a maximum number of points within a pillar of $T = 100$. The linear layer of our feature encoding network has $C = 64$ units. We train our network on a single GPU with a mini-batch size of one. We use an Adam optimizer [28] with initial learning rate 1×10^{-3} and an exponential decay factor of 0.85 applied every ten epochs.

B. Loss

The proposed network is end-to-end trainable using a multitask loss function based on [23] that considers the homoscedastic (task-dependent) uncertainties:

$$\mathcal{L} = \frac{1}{2\sigma_{\mathrm{OD}}^2}\mathcal{L}_{\mathrm{OD}} + \frac{1}{2\sigma_{\mathrm{SF}}^2}\mathcal{L}_{\mathrm{SF}} + \log \sigma_{\mathrm{OD}}^2\sigma_{\mathrm{SF}}^2. \qquad (2)$$

σ_{OD} and σ_{SF} are learnable parameters for object detection (OD) and scene flow (SF) respectively which are initialized with one.

Object Detection Loss: The object detection loss $\mathcal{L}_{\mathrm{OD}}$ is comprised of a classification loss $\mathcal{L}_{\mathrm{cls}}$ and a bounding box regression loss $\mathcal{L}_{\mathrm{reg}}$:

$$\mathcal{L}_{\mathrm{OD}} = \lambda_{\mathrm{cls}}\mathcal{L}_{\mathrm{cls}} + \lambda_{\mathrm{reg}}\mathcal{L}_{\mathrm{reg}}. \qquad (3)$$

λ_{cls} and λ_{reg} are manually chosen weighting factors, which we set to $\lambda_{\mathrm{cls}} = 1$ and $\lambda_{\mathrm{reg}} = 2$.

For classification, the focal loss [29] is used, whereby all anchor boxes are declared as positive or negative like in Faster R-CNN [16] with positive and negative IoU thresholds of 0.6 and 0.45 respectively:

$$\mathcal{L}_{\mathrm{cls}} = -\frac{\lambda_{\mathrm{cls}}^{\mathrm{pos}}}{N^{\mathrm{pos}}}\sum_i (1 - p_i^{\mathrm{pos}})^\gamma \log(p_i^{\mathrm{pos}}) - \frac{1 - \lambda_{\mathrm{cls}}^{\mathrm{pos}}}{N^{\mathrm{neg}}}\sum_i (p_i^{\mathrm{neg}})^\gamma \log(1 - p_i^{\mathrm{neg}}). \qquad (4)$$

$\lambda_{\mathrm{cls}}^{\mathrm{pos}}$ is the balancing factor and γ the focusing parameter. We set them to $\lambda_{\mathrm{cls}}^{\mathrm{pos}} = 0.25$ and $\gamma = 2$. p_i^{pos} and p_i^{neg} are the classification probabilities of the i-th positive and negative anchor box respectively. N^{pos} and N^{neg} are the numbers of positive and negative anchor boxes respectively.

For bounding box regression, the smooth $\mathcal{L}_1$ loss of Girshick et al. [30] and the angle loss of Yan et al. [17] are combined:

$$\mathcal{L}_{\mathrm{reg}} = \frac{1}{N^{\mathrm{pos}}}\Bigg(\mathcal{L}_{1,\,\mathrm{smooth}}(\sin(\Delta\theta^{\mathrm{g}} - \Delta\theta^{\mathrm{p}})) + \sum_{i\in\{x,y,z,l,w,h\}} \mathcal{L}_{1,\,\mathrm{smooth}}(\Delta i^{\mathrm{g}} - \Delta i^{\mathrm{p}})\Bigg) \qquad (5)$$

Scene Flow Loss: As most points are static in the world, there usually is an imbalance between static and dynamic points in real-world datasets. This imbalance impairs the network to generalize well for both types of points. Therefore, we use a weighted $\mathcal{L}_1$ loss for scene flow estimation:

$$\mathcal{L}_{\mathrm{SF}} = \frac{1}{K}\sum_{k=1}^{K} \delta_k \left\| \boldsymbol{f}_k^{\mathrm{g}} - \boldsymbol{f}_k^{\mathrm{p}} \right\|_1, \qquad (6)$$

where $\boldsymbol{f}_k^{\mathrm{g}}$ and $\boldsymbol{f}_k^{\mathrm{p}}$ are the k-th ground truth and the corresponding predicted flow vector respectively. δ_k is a weighting factor for dynamic points.

V. Experimental Setup

We conducted extensive experiments which are explained and presented in the following.

A. Dataset

The experiments are performed on the KITTI object tracking dataset [27], which was recorded using a Velodyne HDL64 LiDAR sensor at a frame rate of $10\,\mathrm{Hz}$. We used the ego-vehicle's odometry to annotate static points with ground truth LiDAR scene flow labels by applying its inverse transformation to every point.

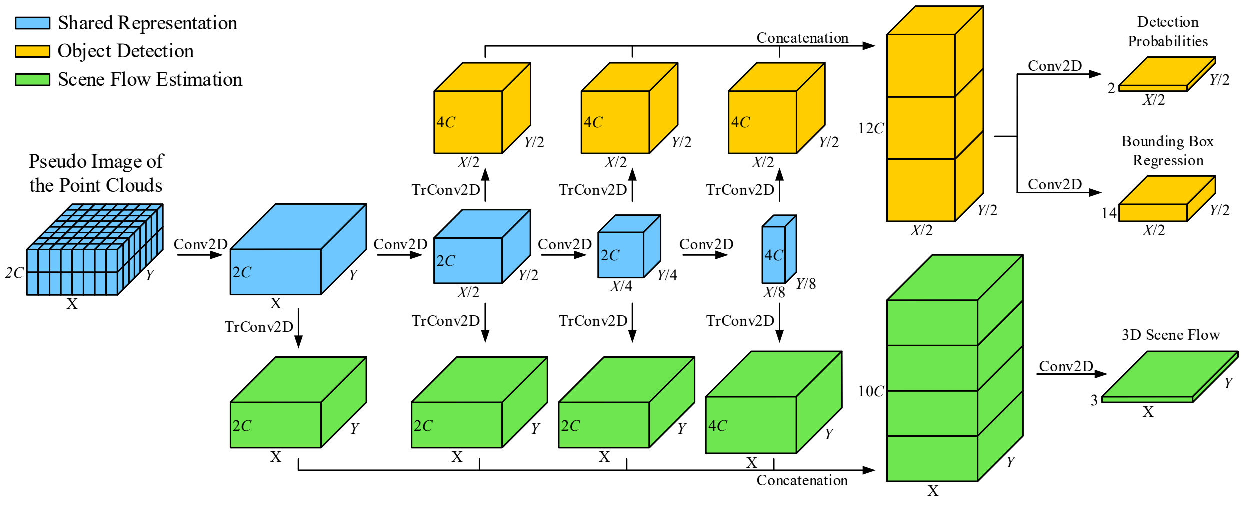

Fig. 5. Convolutional backbone network with output heads. Given the learned 3D representations of both input point clouds, the backbone network initially creates shared feature tensors applying only 2D convolutions. Using transposed convolutions, features from different aggregation stages are taken into account to finally predict 3D bounding boxes with detection probabilities and 3D scene flow vectors.

For dynamic objects, point-wise LiDAR scene flow ground truth is computed using the annotated bounding boxes and their corresponding transformation between consecutive frames. To cope with the high imbalance between static and dynamic points in the KITTI dataset, we set the scene flow loss weighting factor $\delta_k = 1$ for static points and $\delta_k = 10$ for dynamic ones.

For training and testing, we split the KITTI object tracking dataset into two sets according to Table I. The sequences are selected in such a way that both sets have on average a similar number of static and dynamic points.

TABLE I
TRAIN/TEST SPLIT OF THE KITTI OBJECT TRACKING DATASET.

	Training	Test
Selected Sequences	0 – 2, 4, 5, 7, 8, 10, 11, 13 – 18, 20	3, 6, 9, 12, 19
Number of frames	5 633	2 349
Total ratio	70.6 %	29.4 %

B. Data Preprocessing and Augmentation

Before cropping the point clouds from the KITTI object tracking dataset to $[0, 70.4]\,\text{m} \times [-40, 40]\,\text{m} \times [-3, 1]\,\text{m}$ for x, y, and z respectively, multiple augmentations are applied. Firstly, we mirror the given training data sample with a $50\,\%$ chance across the xz-plane. Secondly, we rotate the data around the z-axis by an angle drawn from the uniform distribution $[-45°, 45°]$. Thirdly, we insert five additional moving objects sampled from the training set into each training frame as the dataset only contains annotations for dynamic objects in the camera field of view.

C. Evaluation Metrics

We evaluate our network and the benchmarks on the KITTI object tracking test set presented in Table I. For object detection, we use the average precision (AP) score at $70\,\%$ intersection over union. For scene flow estimation, we use the average endpoint error (AEE), a modified version of the average cosine distance (ACD), the percentage of inliers (endpoint error $EE < 10\,\text{cm}$), and the percentage of outliers ($EE > 30\,\text{cm}$).

Since static objects dominate the scene in most autonomous driving scenarios, the accuracy of scene flow vectors belonging to moving objects has only a small impact on metrics that consider all points equally. This diminishing effect on metrics is in stark contrast to the actual relevance of correctly estimating dynamic objects motions on the safety of operation for an autonomous vehicle. Therefore, we additionally report each metric for dynamic and static points separately in the scene. We define dynamic points as all points in the first point cloud which are inside ground truth bounding boxes.

Regarding the average cosine distance, we propose an adaption which does not penalize the prediction of small scene flow vectors where the direction of the ground truth can be noisy. This is achieved by averaging the cosine distance of K_ϵ ground truth flow vectors, where $\left\|\boldsymbol{f}_k^{\text{g}}\right\|_2 > \epsilon$:

$$ACD = \frac{1}{K_\epsilon} \sum_{k=1}^{K_\epsilon} 1 - S_{\text{c}}(\boldsymbol{f}_k^{\text{g}}, \boldsymbol{f}_k^{\text{p}}) \tag{7}$$

S_{c} is the cosine similarity

$$S_{\text{c}}(\boldsymbol{f}_k^{\text{g}}, \boldsymbol{f}_k^{\text{p}}) = \frac{\boldsymbol{f}_k^{\text{g}} \boldsymbol{f}_k^{\text{p}}}{\left\|\boldsymbol{f}_k^{\text{g}}\right\|_2 \left\|\boldsymbol{f}_k^{\text{p}}\right\|_2} \tag{8}$$

We set $\epsilon = 0.05$ as threshold.

D. Baseline Methods

We compare the performance of our proposed method to multiple baseline methods:

For **simultaneous scene flow & object detection** we use PointFlowNet by Behl et al. [1] as baseline.

TABLE II

RESULTS FOR SCENE FLOW ESTIMATION

			average endpoint error			average cosine distance			inliers			outliers		
	Mode	Aug	all	static	dynamic	all	static	dynamic	all	static	dynamic	all	static	dynamic
PointFlowNet [1]	MT	✓	11.7 cm	11.2 cm	39.2 cm	0.267	0.079	0.571	58.3 %	59.2 %	34.2 %	5.4 %	4.8 %	19.2 %
Ours	MT	✓	9.2 cm	8.7 cm	29.9 cm	0.216	0.054	0.451	79.4 %	80.4 %	52.5 %	4.9 %	4.4 %	18.4 %
PointFlowNet [1]	ST	×	10.4 cm	10.1 cm	21.4 cm	0.145	0.090	0.245	73.0 %	73.5 %	60.2 %	5.6 %	5.3 %	14.0 %
DPCCN [11]	ST	×	9.2 cm	8.6 cm	48.0 cm	0.275	0.049	0.641	70.4 %	71.4 %	42.6 %	2.6 %	2.0 %	18.2 %
Ours	ST	×	**6.9 cm**	**6.7 cm**	**17.9 cm**	**0.091**	**0.046**	**0.186**	**81.2 %**	**81.7 %**	**68.7 %**	**1.5 %**	**1.1 %**	**12.1 %**

For **scene flow**, we chose the Deep Parametric Continuous Convolutional Network (DPCCN) as proposed by Wang et al. [11] as baseline. In their study conducted on their non-public dataset, they have reported an exceptionally small average endpoint error of just 7.8 cm. However, for adapting their network to our experimental setup, we exchanged the ResNet-50 with a ResNet-34 which improved the generalization on the KITTI object tracking dataset significantly. Due to its excessive memory requirements, it was not possible to use FlowNet3D [2] for the full KITTI point clouds. Additionally, we trained PointFlowNet [1] focused on scene flow, i.e. with the object detection loss set to zero.

For **object detection**, we use PointPillars [18] as baseline, as it represents the state of the art in single shot LiDAR 3D object detection. In order to get the best possible object detection performance, we did not limit the maximum number of pillars. Additionally, we trained PointFlowNet [1] focused on object detection, i.e. with the scene flow loss set to zero.

Since the baseline methods as reported in their corresponding papers were not adapted to the KITTI object tracking dataset, we performed an extensive parameter search in order to find the best suited hyperparameters and report only the best results achieved by each method. For all object detection approaches, the data augmentation methods as described in subsection V-B were used. For pure scene flow estimation, augmentation was not necessary to achieve improved results.

VI. RESULTS

The results of the experiments described in section V are presented and discussed in the following. The results for scene flow estimation are shown in Table II and the object detection results in Table III . Results achieved in a multitask setting (i.e. simultaneous object detection and scene flow prediction) are labeled as MT, results achieved in a single task setup, i.e. object detection or scene flow estimation are labeled as ST in the tables.

A. Multitask Performance (MT)

While the object detection performance of PillarFlowNet does not reach the object detection performance of the single task network PointPillars [18], it outperforms the multitask baseline PointFlowNet [1] for object detection significantly. In terms of scene flow, PillarFlowNet's average endpoint errors are significantly lower than PointFlowNet's. Inlier rates for both static and dynamic points are on average 20 percent points higher than PointFlowNet's. Also regarding outlier rates, PillarFlowNet is slightly better than PointFlowNet. Remarkably, PillarFlowNet trained in a multitask setup matches the state-of-the-art DPCCN [11] for scene flow estimation in terms of average endpoint error for static points while being over 37.7 % more accurate on dynamic points.

Qualitative results, i.e. simultaneous predictions of objects and scene flow of PillarFlowNet are shown in Figure 6.

TABLE III

RESULTS FOR OBJECT DETECTION

Method	Mode	AP@0.7
PointFlowNet [1]	MT	38.4 %
Ours	MT	54.7 %
PointFlowNet [1]	ST	45.3 %
PointPillars [18]	ST	**66.3 %**
Ours	ST	65.4 %

It is known that certain tasks in certain scenarios can have synergetic relationships [21]. In order to investigate synergy effects when performing multitask learning, we additionally trained PointFlowNet [1] and our PillarFlowNet for both tasks individually by setting either the scene flow loss or the object detection loss to zero.

B. Singletask Performance (ST)

Our experiments suggest that for both PillarFlowNet and PointFlowNet [1] the synergetic effects from training two seemingly supportive tasks are smaller than the effects from training on a single task individually. Both networks perform better in either object detection or scene flow estimation when being trained to perform a single task exclusively. This demonstrates that for this combination of architectures and tasks, the beneficial effect of focusing on a single task outweighs the synergetic effects that may exist for scene flow and object detection.

Scene Flow: As can be seen in table Table II, in a single task setting, PillarFlowNet surpasses state-of-the-art scene flow estimation baselines. PillarFlowNet is also much more robust than previous approaches, achieving significantly higher inlier rates and significantly lower outlier rates than the baseline methods. As it is to be expected, all methods perform much better on static points than on points belonging to dynamic objects. This underlines the need of evaluating the performance for static and dynamic objects separately.

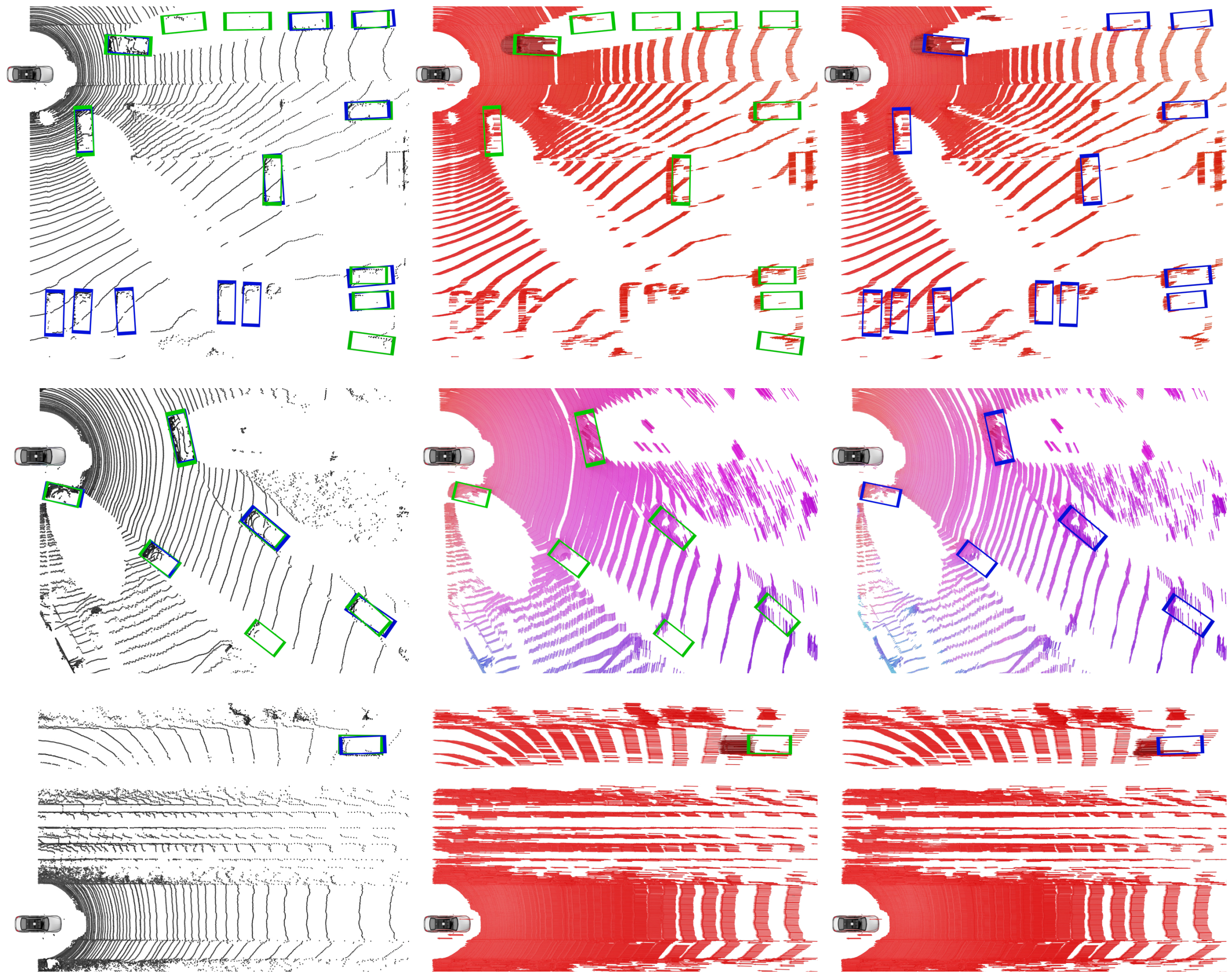

Fig. 6. The first column of images shows PillarFlowNet's object detection prediction (blue) overlayed with ground truth objects (green). Note that PillarFlowNet detects objects outside of the space that objects were labeled in. The KITTI object tracking dataset only contains annotations for objects in the cameras field of view. In the middle column, scene flow and object ground truth is shown. In the right image column, PillarFlowNet's predicted scene flow and objects are shown. Flow vectors are HSV color-coded according to their direction.

Object Detection: LiDAR object detection is a more thoroughly researched field compared to LiDAR scene flow estimation. The dedicated object detection network outperforms our architecture that is designed to perform well on multiple tasks. Noticeably, the performance gap of our multitask network PillarFlowNet towards the state-of-the-art network PointPillars [18] is much smaller than the gap of PointFlowNet [1] towards PointPillars.

C. Runtime

The runtimes were evaluated on a desktop computer with an AMD Ryzen 9 3900X CPU with $3.8\,\mathrm{GHz}$ and an NVIDIA GeForce RTX 2080 Ti GPU using a TensorFlow [31] implementation. The inference of our multitask network takes $87.6\,\mathrm{ms}$. Object detection without scene flow estimation takes $71.7\,\mathrm{ms}$ and scene flow estimation with deactivated object detection takes $86.6\,\mathrm{ms}$. In comparison, PointFlowNet [1] requires with $197.8\,\mathrm{ms}$ 2.3 times more for the same inference.

VII. Conclusion

This paper introduced PillarFlowNet, a novel end-to-end trainable network for simultaneous LiDAR object detection and scene flow estimation capable of real-time performance. In extensive experiments on the KITTI dataset, we compared PillarFlowNet to multiple strong baselines and demonstrated that for simultaneous LiDAR object detection and scene flow estimation it significantly outperforms the state-of-the-art, increasing the average precision score by 16.3 percentage points and reducing the average endpoint error by $21.4\,\%$. Furthermore, we showed that in a single task scene flow training setup PillarFlowNet outperforms the current state-of-the-art by a large margin in terms of endpoint error, while achieving both significantly higher inlier rates as well as significantly lower outlier rates.

Acknowledgment

The authors would like to thank Mr. Behl for kindly providing a reference implementation of PointFlowNet [1].

References

[1] A. Behl, D. Paschalidou, S. Donne, and A. Geiger, "PointFlowNet: Learning representations for rigid motion estimation from point clouds," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2019, pp. 7954–7963.

[2] X. Liu, C. R. Qi, and L. J. Guibas, "FlowNet3D: Learning scene flow in 3d point clouds," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2019, pp. 529–537.

[3] X. Liu, M. Yan, and J. Bohg, "MeteorNet: Deep learning on dynamic 3d point cloud sequences," in *Proc. of the IEEE Int. Conf. on Computer Vision (ICCV)*, Oct. 2019, pp. 9246–9255.

[4] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3d classification and segmentation," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 652–660.

[5] A. Dewan, T. Caselitz, G. D. Tipaldi, and W. Burgard, "Rigid scene flow for 3d LiDAR scans," in *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2016, pp. 1765–1770.

[6] X. Gu, Y. Wang, C. Wu, Y. J. Lee, and P. Wang, "HPLFlowNet: Hierarchical permutohedral lattice FlowNet for scene flow estimation on large-scale point clouds," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2019, pp. 3249–3258.

[7] A. K. Ushani and R. M. Eustice, "Feature learning for scene flow estimation from LIDAR," in *2nd Ann. Conf. on Robot Learning (CoRL)*, Oct. 2018, pp. 283–292.

[8] A. K. Ushani, R. W. Wolcott, J. M. Walls, and R. M. Eustice, "A learning approach for real-time temporal scene flow estimation from LIDAR data," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, June 2017, pp. 5666–5673.

[9] V. Vaquero, A. Sanfeliu, and F. Moreno-Noguer, "Deep lidar CNN to understand the dynamics of moving vehicles," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2018, pp. 1–6.

[10] S. A. Baur, F. Moosmann, S. Wirges, and C. B. Rist, "Real-time 3d LiDAR flow for autonomous vehicles," in *Proc. of the IEEE Intelligent Vehicles Symp. (IV)*, June 2019, pp. 1288–1295.

[11] S. Wang, S. Suo, W.-C. Ma, A. Pokrovsky, and R. Urtasun, "Deep parametric continuous convolutional neural networks," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2018, pp. 2589–2597.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[13] W. Wu, Z. Wang, Z. Li, W. Liu, and L. Fuxin, "Pointpwc-net: A coarse-to-fine network for supervised and self-supervised scene flow estimation on 3d point clouds," *arXiv:1911.12408 [cs.CV]*, Nov. 2019.

[14] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *arXiv:1912.12033 [cs.CV]*, Dec. 2019.

[15] Y. Zhou and O. Tuzel, "VoxelNet: End-to-end learning for point cloud based 3d object detection," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2018, pp. 4490–4499.

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 39, no. 6, pp. 1137–1149, June 2017.

[17] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, Oct. 2018.

[18] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast encoders for object detection from point clouds," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2019, pp. 12 689–12 697.

[19] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun, "MultiNet: Real-time joint semantic reasoning for autonomous driving," in *Proc. of the IEEE Intelligent Vehicles Symp. (IV)*, June 2018, pp. 1013–1020.

[20] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," in *Advances in Neural Information Processing Systems (NIPS)*, Dec. 2018, pp. 527–538.

[21] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2018, pp. 3712–3722.

[22] S. Wang, S. Fidler, and R. Urtasun, "Holistic 3d scene understanding from a single geo-tagged image," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 3964–3972.

[23] R. Cipolla, Y. Gal, and A. Kendall, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2018, pp. 7482–7491.

[24] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, "SBNet: Sparse blocks network for fast inference," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2018, pp. 8711–8720.

[25] W. Luo, B. Yang, and R. Urtasun, "Fast and Furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2018, pp. 3569–3577.

[26] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun, "Multi-task multi-sensor fusion for 3d object detection," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2019, pp. 7345–7353.

[27] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2012, pp. 3354–3361.

[28] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. of the 3rd Int. Conf. on Learning Representations (ICLR)*, May 2015.

[29] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. of the IEEE Int. Conf. on Computer Vision (ICCV)*, Oct. 2017, pp. 2999–3007.

[30] R. Girshick, "Fast R-CNN," in *Proc. of the IEEE Int. Conf. on Computer Vision (ICCV)*, Dec. 2015, pp. 1440–1448.

[31] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. of the 12th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, Nov. 2016, pp. 265–283.