2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

2021 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9077-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICRA48506.2021.9561644

VelocityNet: Motion-Driven Feature Aggregation for 3D Object Detection in Point Cloud Sequences

David Emmerichs[1,2,3], Peter Pinggera[1], Björn Ommer[2]

***Abstract*— The most successful methods for LiDAR-based 3D object detection use sequences of point clouds in order to exploit the increased data density through temporal aggregation. However, common aggregation methods are rarely able to capture fast-moving objects appropriately. These objects are displaced by large distances between frames and naive approaches are not able to successfully leverage the full amount of information spread across time. Yet, especially in autonomous driving scenarios, fast-moving objects are most crucial to detect as they actively take part in highly dynamic traffic situations. This work presents a novel network architecture called VelocityNet which is explicitly designed to temporally align features according to object motion. Our approach extends traditional 3D convolutions by a motion-driven deformation of the convolution kernels across the temporal dimension. The required motion information can be obtained from various sources, ranging from external computation or complementary sensors to an integrated network branch which is trained jointly with the object detection task. The explicit feature alignment allows the training process to focus on the object detection problem and results in a significant increase in detection performance compared to the popular PointPillars baseline, not only for dynamic but also for static objects. We evaluate our approach on the nuScenes dataset and analyze the main reasons for the observed performance gains.**

I. Introduction

In the setting of autonomous robotic agents, it is of particular importance to detect and recognize other moving agents in addition to the static environment. When considering the application domain of autonomous driving (AD), these moving agents are both highly safety-critical and hard to predict. Perception systems may monitor their surroundings with a high frequency and establish an accurate representation of their static environment by aggregating information over a small amount of time. Dynamic elements are harder to detect and recognize correctly, as their appearances change and the aggregation over non-negligible periods of time requires accurate estimates of object motion.

Most current LiDAR object detectors are based on the sequential processing of individual frames (scans) [12], [24], [5], [30]. Only a few state-of-the-art approaches leverage multiple point cloud frames via simple mechanisms such as annotating points with their measurement timestamps [3] or introducing higher dimensional convolutions [15] to incorporate the temporal dimension. These approaches work well for slowly moving objects or static environments, however, we claim that they lack the ability to aggregate fast objects as those are not spatially localized across time.

[1]Mercedes-Benz AG
[2]IWR, Heidelberg University
[3]david_josef.emmerichs@daimler.com

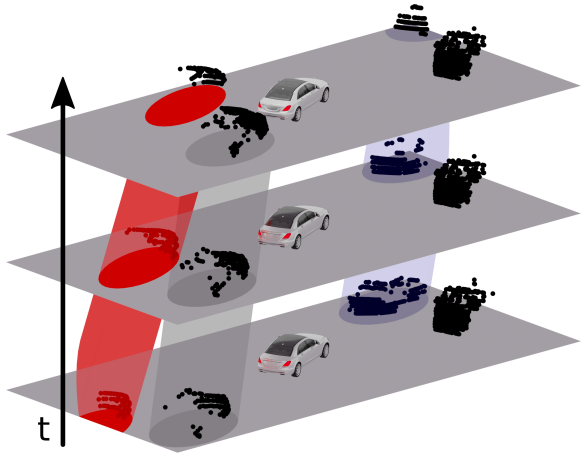

Fig. 1. Our method uses a sequence of ego-motion corrected point clouds (one plane per frame) and a velocity map, visualized by colored cylinders. The absolute velocity is represented by the transparency and the angle, e.g. the left car is moving with a high velocity whereas the right car is static. Our method aligns the 3D convolution operators along these cylinders to support the network in aggregating relevant features.

To facilitate an appropriate method of temporal aggregation exact motion information is needed, as illustrated in Fig. 1. We argue that this information is inherently available in a sequence of LiDAR point clouds. Previous works [25], [17], [26] have shown that motion information can be obtained robustly or even self-supervised from the point cloud data prior to tackling high-level tasks. Therefore, we introduce VelocityNet, a neural network architecture that utilizes time-deformed convolutions which can leverage localized motion information in order to align features along the temporal dimension before applying the convolution kernel. Our approach can be complemented with any object detection post-processing scheme. We investigate the usage of different external sources of motion information as well as learning to produce this information in a multi-task setup together with the object detection task.

In summary, the contributions of this paper are fourfold:

- We propose a novel time-deformed convolution (td-conv) kernel for motion-corrected feature aggregation across time in scale-invariant input representations.
- We introduce a corresponding backbone architecture capable of leveraging motion information from variable sources for optimized aggregation at the feature level.
- We show the effectiveness of our network by comparing it against PointPillars (PP) on the nuScenes dataset [3].
- Furthermore, we perform ablation studies to analyze the main reasons for the observed performance gains.

II. Related Work

A. Object detection in LiDAR point clouds

Early works built upon the success of image-domain object detection by employing well-known methods on hand-crafted transformations of point clouds into image-like data structures [28], [20]. With the emergence of PointNet [18] and its success in classifying sets of points it became possible to directly train networks on point clouds in an end-to-end fashion. VoxelNet [31] and PointPillars [12] push the idea to the extreme by running a small PointNet with shared weights on voxels in a 3D- or Bird's-Eye-View (BEV) grid. More recent architectures, like MVF-Net [30], MVLidarNet [5], and pillar-based object detection [24], leverage multiple views and communicate information between those in order to get the best out of two worlds, e.g. the dense sensor view and the scale- and translation-invariant BEV.

B. Information aggregation over time

The first approaches to utilize temporal information in videos on feature level include flow-guided feature aggregation [33] which uses a pretrained optical flow net to warp and aggregate features accordingly. The introduction of deformable convolutions [7] lead to improved object detection performance not only in images but also in videos [2] by learning deformations according to the object movements.

C. Temporal aggregation in LiDAR object detection

Despite the amount of methods aggregating multiple frames of point clouds for an increased data density, very little research actually investigates how to effectively aggregate information over time in the AD domain. With motion mostly taking place in the ground plane, being sparse and discrete, the structure of motion in the data is very different from dense optical flow in videos.

The Fast and Furious (FaF) approach [15] is the first to explore different levels of temporal aggregation by applying 3D convolutions across two spatial and one temporal dimension. With a focus on tracking and motion forecasting, it demonstrates a benefit to aggregate features across time within the middle instead of the beginning of the network. The MinkowskiNet [6] replicates traditional convolutional architectures but replaces 2D convolutions with higher-dimensional counterparts and specially shaped kernels across the temporal dimension.

With the published nuScenes dataset [3] and the more cost-efficient low-resolution LiDAR sensor (about 30k points per frame compared to over 100k points as in KITTI [10], ARGO [4] or Waymo dataset [22]) it is essential to process a sequence of point clouds. The authors of [3] simply use the PointPillars architecture and annotate the points from 10 consecutive frames with their respective relative timestamp producing one large point cloud with 300k points. Even top-performing methods on this dataset use this simple aggregation strategy [32], [23].

A few works [11], [9], [16], [29] explored the usage of recurrent units like LSTMs and GRUs to process multiple point clouds with most of them correcting for ego-motion.

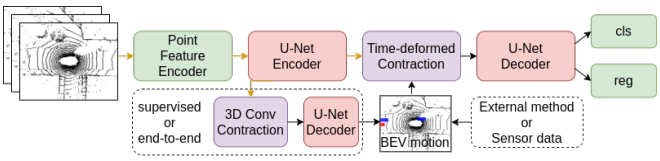

Fig. 2. **Overall network architecture:** VelocityNet applies the point feature encoder and U-Net encoder to multiple frames and uses our time-deformed contraction module to produce a single prediction for the latest frame. Orange arrows denote frame sequences passed between components. Our VelocityNet can leverage motion information that can either come from external sources or be predicted and integrated into the network model via a separate branch. In comparison, the TimeConvNet is using standard 3D convolutions as a contraction mechanism and the PointPillars baseline only processes a single frame and does not require a contraction module at all.

As opposed to our approach, none of the methods above incorporate object motion and instead rely on large receptive fields and encoding all possible movements and objects in the channel dimensions.

MeteorNet [14] operates on point sets directly through a PointNet++-like [19] architecture using time as an additional point coordinate and employing a nearest-neighbor grouping mechanism using scene flow to group along object trajectories. The focus, however, was placed on scene classification, scene flow computation, and semantic segmentation.

III. Proposed Approach

Fig. 2 gives an overview of our network architecture, which will be described in detail in the following sections. Using $\mathcal{T}_T = \{-(T-1), \dots, 0\}$ to denote causally available time steps, our proposed VelocityNet (VN) takes as input:

- a velocity map in Bird's-Eye-View (BEV) representation $\mathbf{V}(t, \mathbf{p}) : \mathcal{T}_T \times \{\mathbf{p} < (H\ W)^T | \mathbf{p} \in \mathbb{N}_0^2\} \rightarrow \mathbb{R}^2$
- a causally available sequence of T point clouds $\mathcal{P} = (\mathcal{P}_{-(T-1)}, \dots, \mathcal{P}_t, \dots, \mathcal{P}_0)$

First, we present our approach to integrating the velocity information into a convolutional neural network (CNN) architecture. Next, we discuss the possibilities of generating the velocity information needed by the network. Finally, we provide a detailed description of our network setup.

A. Time-deformed convolutions

The main component in our VN is the time-deformed convolution (td-conv), inspired by the deformable convolutions (d-convs) [7]. It integrates the velocity information into the network. We first recap the definition of the d-conv:

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n), \qquad (1)$$

where $\mathbf{x}, \mathbf{y}$ represent the input and output feature maps at integer pixel positions $\mathbf{p}$. The set $\mathcal{R} = \left\{\binom{-1}{-1}, \binom{-1}{0}, \dots, \binom{1}{0}, \binom{1}{1}\right\}$ defines the kernel size and shape (here 3×3) and $\mathbf{w}(\mathbf{p}_n)$ for $\mathbf{p}_n \in \mathcal{R}$ the weight matrix at a specific kernel position. With $\Delta\mathbf{p}_n = \vec{0}$ this reduces to a standard convolution. In the setting of d-convs the delta on the pixel positions is learnable and usually depends on the input feature map $\mathbf{x}$.

Similar to d-convs we use bilinear interpolation G for fractional locations $\mathbf{p}$ in $\mathbf{x}(\mathbf{p}) = \sum_{\mathbf{q}} G(\mathbf{q}, \mathbf{p}) \cdot \mathbf{x}(\mathbf{q})$. We denote with $\mathbf{x}_t(\mathbf{p}_0)$ the feature vector at pixel position $\mathbf{p}_0$ and frame id t when dealing with a sequence of T frames $\mathcal{T}_T$. The learned offsets $\Delta\mathbf{p}_n$ are replaced in our method by the velocity map information.

$$\mathbf{y}_t(\mathbf{p}_0) = \sum_{\substack{\Delta t \in \mathcal{T}_{\Delta T} \\ \mathbf{p}_n \in \mathcal{R}}} \mathbf{w}(\Delta t, \mathbf{p}_n) \cdot \mathbf{x}_{t+\Delta t}(\mathbf{p}_0 + \mathbf{p}_n + \Delta t \mathbf{V}(t, \mathbf{p}_0)) \tag{2}$$

Not only spanning over a kernel size across pixel locations but also across a kernel size ΔT along the time dimension, the td-conv is a modified 3D convolution. The deformation in this 3D case is then given by $\Delta\mathbf{p}_{\Delta t,n} = \Delta t \mathbf{V}(\mathbf{p}_0)$.

The main difference between the d-conv and our td-conv is the restriction of the deformation to a simple translational component. Additionally, it lifts this deformation up to 3 dimensions such that it can be applied to a set of consecutive image patches out of a sequence over time.

In the setting of AD and more general in real-world environments, most elements appear static in the world frame. Only very few objects are dynamic, show mostly translational motion components, and rarely scale in size or deform in more complex ways. These observations lead to the introduction of the above-defined td-conv which is able to handle sparse motion patterns. For every feature map $\mathbf{x}_t(\mathbf{p}_0)$ to be causal, we only use the present and past feature frames and align those always to the current frame t.

We implement the td-conv as well as the whole network in TensorFlow [1] using standard framework components like gather and scatter operations.

B. Velocity information at inference time

In contrast to the d-conv the td-conv does not use locally learned positional offsets but instead leverages the consistent velocity maps $V(t, \mathbf{p})$. There are multiple possibilities to generate these velocity maps during training and inference.

External state-of-the-art method. An obvious approach is to use an external method that is specialized in producing this velocity information based on available sensor information like the point clouds themselves. We use the published MotionNet [25] as a representative for this approach.

Multi-task learning of motion and detection. Another possibility is to use a standard point cloud encoder and use these encoded feature maps not only inside the time-deformed convolutions for the object detection but implement a second network head that predicts the velocity maps. These maps are not only handed to the time-deformed convolutions but receive also explicit supervision through ground truth labels during training.

End-to-end trainable velocity maps. Like the offsets in the d-convs, the velocity maps do not necessarily need to be external or need a supervision signal. As the bilinear kernel is differentiable w.r.t. the offsets, we can backpropagate the error through the td-convs onto the velocity map output from a separate network or again a network head on top of a common feature encoding. To be more resource-efficient and to possibly introduce a self-regularizing effect on the velocity output we only investigate the latter option.

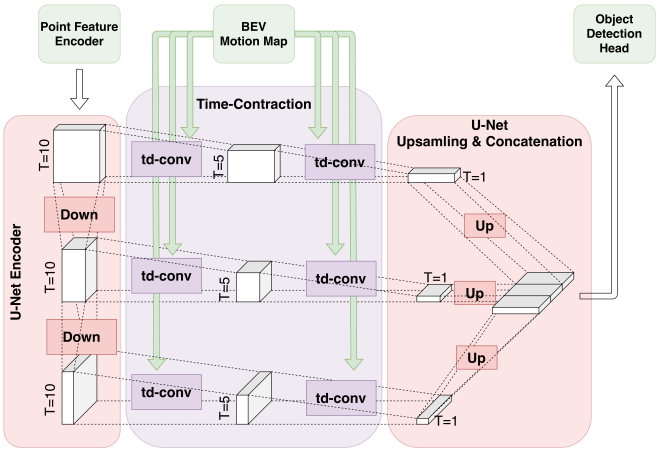

Fig. 3. **Backbone architecture:** The point feature encoder-processed point clouds are stacked along the temporal (vertical edge) dimension. First, the Down blocks produce 3 spatially (horizontal edge) downsampled tensors with increasing channel numbers. The Contract blocks (middle) consisting of td-convs remove the temporal dimension and leverage motion information. Finally, the Up blocks and concatenation produce the decoder output.

Complementary sensors ("ground truth" velocity). There might be scenarios in which more accurate velocity information is available (e.g. radar or frequency-modulated continuous-wave LiDAR data). Therefore, and for ablation study purposes, we also run experiments in which we use the ground truth velocities as input to our method.

C. Overall network architecture

In order for comparability and simplicity, we decide to use the PointPillars (PP) [3] network architecture as a baseline. Our introduced changes for our intermediate TimeConvNet (TCN) baseline and our VelocityNet (VN) architecture are limited to the backbone only.

Encoder. The first component of all our architectures is the point feature encoder (PFE) [12], [31] which takes raw point clouds as input and produces BEV feature maps. With a time-sequence of point clouds, there are two possibilities of how to use the PFE. The PP baseline [3] aggregates all point clouds into a single instance and produces a single BEV feature tensor from all points together. In order for PP to process motion every point is annotated with its relative measurement timestamp. For our intermediate TCN baseline and VN, we apply a weight-sharing PFE across all T input point clouds and let it produce T BEV feature maps which can be stacked along the first dimension to produce a 3D feature tensor.

Backbone. Our overall VN backbone architecture is shown in Fig. 3. We start by introducing the classical PP backbone and present our made changes afterward.

The baseline uses three Down(S, L, C)-sampling blocks and correspondingly three Up(S, C)-sampling blocks. The downsampling blocks are chained and each one consists of L convolutional layers with output channel size C, kernel size 3, and with the first having a stride of S. The three

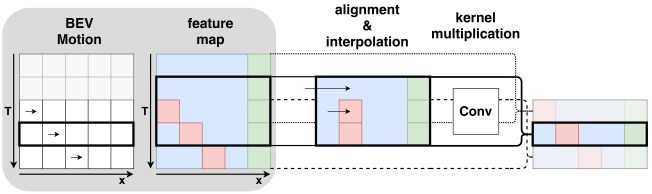

Fig. 4. **Time-deformed convolution:** For simplicity, only a single spatial dimension x is shown. The black frames mark the currently processed position in time and run in parallel to the processing of all timestamps (dashed and dotted). First, features are aligned and interpolated to the kernel's position in space. This interpolation is multiplied with a traditional convolutional kernel. The coloring corresponds to abstract notions of a dynamic object (red, captured by the velocity map), a static object (green), and some background (blue).

intermediate results of each downsampling block are fed into the corresponding upsampling blocks which consist of a transposed convolution of stride and kernel size S and output channel size C. Finally, the feature maps are concatenated along the channel dimension. Each of the aforementioned convolutions is followed by a BatchNorm and a ReLU.

However, using a stack of feature maps as input to the backbone requires additional blocks which can incorporate 3D convolutions. Inspired by FaF [15] we construct our intermediate TCN baseline. We decide to use the same structure for the downsampling but apply it across the T input frames with shared weights. The three feature map stacks of different spatial resolution are then contracted along the time-dimension using 3D convolutions. These Contract(T, L, C) blocks consist of L 3D convolutions each again followed by a BatchNorm and a ReLU. The 3D convolutions have an output channel size of C and a 3×3 spatial kernel using padding similar to Down$(\cdot)$. However, along the temporal dimension, no padding is used and the kernel sizes for the L convolutions are selected such that after the L layers the temporal dimension is contracted to a size of 1.

Having these 3D convolutions inside the backbone, they can now easily be switched out for our novel td-convs leading to our VN. In Fig. 4 we show how this contraction over time and the alignment process can be visualized and how it potentially benefits object detection.

Head. Following the concatenated features of the backbone, a simple linear layer (1x1 convolution) without any normalization or activation encodes the object predictions similar to [12]. For each of the 10 detection classes, we have 1 channel as detection logit, 3 channels for each the 3D location and size, 2 channels for the velocity prediction, and a regression value and classification logit for the object's orientation.

Loss. In the output and loss, we treat each detection class separately. The loss is computed as the sum over all class-specific losses $\mathcal{L} = \frac{1}{|C|} \sum_{c \in C} \mathcal{L}_c$ which in turn sum over all classification and regression losses and are normalized by the number of positive anchors.

$$\mathcal{L}_c = \frac{1}{\max(N_{c,pos}, 1)} \sum_{p \in \left\{ \substack{cls, loc, size, \\ ori, velo, attr} \right\}} \lambda_p \mathcal{L}_{c,p} \qquad (3)$$

The classification loss is the original binary focal loss [13] with its default values. The localization and size loss are the same as in [12], [27], a smooth L1 loss on normalized errors. We also adopt the loss for orientation consisting of the regression part SmoothL1$(\sin(\Delta))$ and a classification loss w.r.t. the anchor box orientation. This classification is omitted for the flip-invariant barrier class. The velocity loss is again a smooth L1 loss on the error

$$\mathcal{L}_{c,velo} = \sum_{i \in x,y} \text{SmoothL1}(v_i^{(p)} - v_i^{(gt)}) \quad . \qquad (4)$$

IV. Experiments

We first introduce the nuScenes dataset and metrics as well as the external method for motion computation. Afterward, we give details on the experimental setup regarding the three network architectures we compare against each other, namely the PP baseline, the TCN and the VN. Finally, we show, compare, and discuss the results of our different experiments.

A. Dataset

To evaluate our novel td-conv and the effectiveness of feature map aligning across time we use the nuScenes [3] dataset. This dataset contains 1 000 scenes each consisting of 20 seconds of data recorded by a vehicle-mounted sensor set containing radar, LiDAR, camera, IMU, and GPS sensors. We only use the LiDAR data recorded at 20Hz and the ego position computed from the IMU and GPS sensor to align point clouds against each other. In addition, the dataset provides dense object annotation with a frequency of 2Hz which allows trajectory interpolation and velocity estimation for each object. As evaluation metrics, we use the NuScenes detection score (NDS) and one of its main components, the mean average precision (mAP) across classes and matching distance thresholds, as introduced in [3].

B. External Motion

As an exemplary external method for producing BEV velocity maps, we use MotionNet [25], which is specifically designed to robustly estimate motion in point clouds. The approach focuses on predicting future movements whereas we require alignment over the past frames. As movements are typically steady, especially for larger objects, and the time spans for the past (0.5 s) and future frames (1 s) are small we take the first motion prediction from the network output and assume all motions to be uniform and linear.

Additionally, MotionNet is designed for a smaller BEV extent of only $[-32\,\text{m}, 32\,\text{m}]$ along x- and y-axis. The NDS, however, requires predictions at distances up to $50\,\text{m}$. Therefore, we simply increase the extent of the BEV to $[-50\,\text{m}, 50\,\text{m}]$ while retaining the voxel resolution of $0.25\,\text{m}$. Another aspect of MotionNet is that it is only trained on frames with enough sweeps in the sequence, therefore requiring to drop certain samples from training and evaluation which lie at the temporal edges of scenes. We show the change in performance introduced by the smaller training and validation sets in Table II.

C. Data augmentation

It is very common in LiDAR object detection tasks to augment the point clouds during training in order to decrease overfitting and therefore improve the robustness of the network. We follow the augmentation protocol from [12] and insert up to 4 objects for each of the classes bus, trailer, and truck. We perform collision checks with existing points and augmented object boxes as well as make sure that the insertions happen in the same place relative to the ego vehicle such that the resulting placements are as realistic as possible. As we focus on object motion, a potentially important aspect of the insertion of additional ground truth objects is the process of distributing these objects across the different frames. We experimented with both static insertion and insertion of dynamic object tracklets. However, we found the trainings of the PP baseline as well as the VN to be unaffected by this. We opted for the dynamic ground truth insertion for all our experiments as it results in more realistic augmentations.

D. PointPillars

Our network follows closely the setup of [3]. The PFE produces a $H{=}W{=}400$-grid of $C{=}64$-dimensional features. We use $\text{Down}(S{=}2, L{=}4, C{=}64)$, $\text{Down}(2, 6, 128)$, $\text{Down}(2, 6, 256)$ and $\text{Up}(S{=}1/2, C{=}128)$, $\text{Up}(1, 128)$, $\text{Up}(2, 128)$, respectively. All networks are trained for 120k iterations with a batch size of 3 using the OneCycle [21] training policy and a maximum learning rate of $1.5 \cdot 10^{-3}$.

E. TimeConvNet

The TimeConvNet (TCN) features an additional contraction module between the encoder and decoder. It uses for all three feature map resolutions $\text{Contract}(T{=}10, L{=}3, C)$ with the respective channel size of the incoming feature map.

F. VelocityNet

The VelocityNet (VN) architecture corresponds directly to TCN with all the 3D convolutions replaced by td-convs. When following the approach of learning the velocity maps in a multi-task setup or an end-to-end trainable fashion we employed a second network branch on top of the encoded feature maps as shown in Fig. 2. The additional stack of Contract blocks using 3D convolutions has the same number of layers as the detection backbone but only half the channel dimension in each layer. Together with the Up blocks and a separate network head, it regresses the velocities.

All the approaches mentioned in section III-B for producing velocity maps for training and inference output just a single map for the most current frame $\mathbf{V}(t = 0, \mathbf{p})$. However, the td-convs require velocity maps for every time step $\mathbf{V}(t, \mathbf{p})$. As argued for the MotionNet output we can make the approximation that all objects move in a uniformly and linear manner. This lets us compute the velocity maps for the previous frames using the relation $\mathbf{V}(t, \mathbf{p} + t\mathbf{V}(0, \mathbf{p})) = \mathbf{V}(0, \mathbf{p})$. This assignment rule might assign multiple different velocity vectors to the same position in the past. In this case, we opt for the larger velocity as those displacements can be handled less effectively by 3D convolutions and are the reason why we introduced td-convs in the first place.

TABLE I

COMPARISON OF MOTION TYPES

net	motion type	NDS	mAP
PP	-	43.0	31.2
TCN	-	45.3	34.1
VN	GT	47.0	34.9
VN	External [25]	**47.3**	**35.9**
VN	Multi-Task	45.8	34.3
VN	End-to-end	43.9	30.2

G. Experimental results

We now compare our VN against the PP baseline and investigate the influence of different aspects of training.

Motion information. We first present the different performance results of our VN regarding the different motion sources. We show the results in Table I. The results of the external velocities from the MotionNet and the ground truth motion are on par. We believe this is due to the external velocities falling short only on extremely hard-to-detect objects. In these cases, the object detection performance does not really improve even with ground truth motion. A possible reason why the external velocities actually perform a bit better might be due to a more focused highlighting of objects that are detectable where the ground truth motion instead also highlights superfluous objects that are simply too hard to detect for the network and are more similar to a noisy effect. When training the VN in an end-to-end fashion through backpropagation inside the bilinear interpolation, or in a multi-task setup with additional supervision of the motion branch, we observe performance similar or worse compared to the TCN. The end-to-end learned motions by this separate branch do not resemble anything meaningful for humans. We suspect that the high number of motion modes and the sparsity of motion in BEV compared to the scenarios [2], [33] make it hard to learn motion cues through the bilinear operator. As also the supervised multi-task learning does not increase the performance compared to TCN, it seems that the object detection task and the motion prediction task do not benefit much from each other, but compete for computational network expressiveness. This counter-intuitive effect was also observed in [8]. Using the external velocities from the fixed MotionNet model alleviates this problem as the VN can completely focus on the detection task. For the following VN experiments we opt for the external velocities.

Network sizes. When introducing a computationally more demanding network architecture one important question is if the performance gain is just related to the increased model capacity and computational demand or if the architecture itself produces a performance boost. To this end, we evaluate a larger PP model with increased spatial BEV resolution and feature channel sizes and present the results in Table II. We doubled the channel size and spatial resolution in every network step resulting in an initial PFE resolution of $H = W = 800$ and $C = 128$ channels. The number of floating-point operations (FLOPs) is counting addition

TABLE II

PERFORMANCE OF DIFFERENT ARCHITECTURE SIZES. PP* SHOWS BASELINE RESULT FOR ORIGINAL NUSCENES TRAIN-VAL SPLIT.

net	HW	C	NDS	mAP	FLOPs$[10^9]$	#params
PP*	400	64	43.2	30.2	44	4.2M
PP	400	64	43.0	31.2	44	4.2M
PP	800	128	45.1	30.7	708	16.8M
TCN	400	64	45.3	34.1	867	13.5M
VN	400	64	**47.3**	**35.9**	868	13.5M

TABLE III

MAP SCORE [%] DEPENDENCY ON DISTANCE AND SPEED

net	static	dynamic	<30 m	30 m-50 m
PP	29.7	41.2	37.3	17.1
TCN	32.4	43.7	40.8	18.2
VN	**34.0**	**45.0**	**42.8**	**18.6**

and multiplication for a single inference of the model and gives a rough estimate over the computational demands. Note, that when running the TCN and VN in an online environment, only a fraction of the tensors in the encoder and contraction modules need computing as most results from the previous time step can be used. As the results show, simply increasing the PP architecture does not necessarily improve the detection mAP score but actually decreases it. The gained capacity is instead used for improving the regression such that the NDS increases. Compared to the 3D convolutional counterpart TCN we have a similar computational demand and NDS but with a 3% increase in mAP. The use of motion information and the td-conv further increases the detection and the NDS and shows the effectiveness of our approach.

Detections over distance and speed. We analyze the mAP score for filtered sets of ground truths and predictions based on velocity (static $< 10\,\mathrm{km/h}$) and distance thresholds. In order to avoid losing prediction-to-ground-truth matches through the filtering, we assign predicted boxes to the same bin as their nearest ground truth box. The results in Table III confirm a significant improvement for moving objects.

Detections from larger timespans. To evaluate how important feature alignment is for larger displacements we compare the performance of the different architectures under a changing sampling frequency of input point clouds. The nuScenes evaluation scheme specifies a maximum of 10 frames at 20 Hz of past data for inference. We evaluate and compare this against trainings on 10 frames at 5 Hz. The computational complexity stays the same as we use the same amount of points as input. We show the NDS and mAP scores in Table IV. We notice that there is a performance drop in all three models, underlying our claim that fast-moving objects are a problem as those are the ones that significantly change appearance in the sequence when changing the sampling frequency. As expected our VN is least affected by the reduced sampling frequency and especially w.r.t. the bare detection task (mAP) it is on par whereas the other models suffer a 2% accuracy drop.

Qualitative results. Based on the above results we can see that the VN outperforms the baseline by a large margin thanks to the increased detection performance of fast objects.

TABLE IV

IMPACT OF DIFFERENT POINT CLOUD SAMPLING FREQUENCIES

input data frequency	network	NDS	mAP
20Hz	PP	43.0	31.2
	TCN	45.3	34.1
	VN	**47.3**	**35.9**
5Hz	PP	41.4 (1.6↓)	28.6 (2.6↓)
	TCN	44.5 (0.8↓)	32.3 (1.8↓)
	VN	**46.6** (**0.7**↓)	**35.9** (**0.0**)

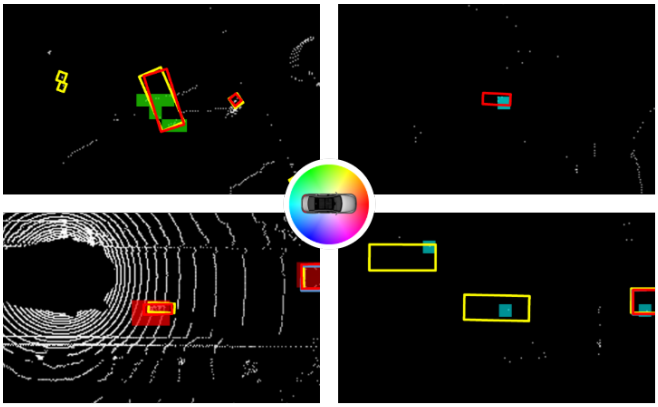

Fig. 5. Comparison of the PointPillars baseline (blue) against our VelocityNet (red) with ground truth annotations marked yellow. All shown velocities are above $40\,\mathrm{km/h}$ and the direction is given by a color overlay. The left two scenes show cases where fast-moving objects are detected by the VN which were missed by the baseline. The upper right is the only case in the validation set where a false positive prediction was made on top of velocities. The lower right depicts two ground truth boxes that were annotated with the more robust MotionNet but which the VN still failed to detect. Note that there is only a single LiDAR point on these objects.

Fig. 5 shows a few cases in which the PP baseline failed to detect a few objects which the VN managed to find mainly due to the velocity information given for that object. Still, there exist failure cases in which either the velocity annotation was wrong or the aggregation had not enough evidence to predict an object.

V. CONCLUSION

In this work, we presented a novel end-to-end trainable 3D object detection network that is capable of aligning feature representations according to the motion of observed objects. This enables the network to consistently outperform representative baseline approaches, in particular on fast objects for which aggregation over time is crucial because of the limited amount of point measurements. Our method can potentially benefit from future advances in low-level motion and scene flow computation. Directly training or finetuning a dedicated motion estimation branch through the bilinear operator as was done in STSN [2] proved to be difficult in our experiments and did not yield the expected performance gains. We believe this is due to the extreme sparsity and higher number of motion modes observed in a single frame. We reserve further improvements of this attractive idea for future work. With a growing exploration and understanding of attention mechanisms, our method can potentially benefit from those by adaptively aggregating features based on the respective amount of information in each feature patch.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] G. Bertasius, L. Torresani, and J. Shi. Object detection in video with spatiotemporal sampling networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 331–346, Sept. 2018.

[3] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuScenes: A Multimodal Dataset for Autonomous Driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, June 2020.

[4] M. F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays. Argoverse: 3D tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2019-June, pages 8740–8749, June 2019.

[5] K. Chen, R. Oldja, N. Smolyanskiy, S. Birchfield, A. Popov, D. Wehr, I. Eden, and J. Pehserl. MVLidarNet: Real-Time Multi-Class Scene Understanding for Autonomous Driving Using Multiple Views. arXiv preprint arXiv:2006.05518, 2020.

[6] C. Choy, J. Gwak, and S. Savarese. 4D spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2019-June, pages 3070–3079, June 2019.

[7] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable Convolutional Networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2017-Octob, pages 764–773, Oct. 2017.

[8] F. Duffhauss and S. A. Baur. PillarFlowNet: A Real-time Deep Multitask Network for LiDAR-based 3D Object Detection and Scene Flow Estimation. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020.

[9] A. El Sallab and I. Sobh. YOLO4D : A Spatio-temporal Approach for Real-time Multi-object Detection and Classification from LiDAR Point Clouds. *NIPS '18 Workshop*, page 8, 2018.

[10] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[11] R. Huang, W. Zhang, A. Kundu, C. Pantofaru, D. A. Ross, T. Funkhouser, and A. Fathi. An LSTM Approach to Temporal 3D Object Detection in LiDAR Point Clouds. arXiv preprint arXiv:2007.12392, 2020.

[12] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2019-June, pages 12689–12697, June 2019.

[13] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. Focal Loss for Dense Object Detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[14] X. Liu, M. Yan, and J. Bohg. Meteornet: Deep learning on dynamic 3D point cloud sequences. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, volume 2019-Octob, pages 9245–9254, Oct. 2019.

[15] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3569–3577, June 2018.

[16] S. Mccrae and A. Zakhor. 3d Object Detection For Autonomous Driving Using Temporal Lidar Data. In *2020 IEEE International Conference on Image Processing (ICIP)*, 2020.

[17] H. Mittal, B. Okorn, and D. Held. Just Go With the Flow: Self-Supervised Scene Flow Estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11174–11182, June 2020.

[18] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017-Janua:77–85, 2017.

[19] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 5100–5109, 2017.

[20] M. Simon, S. Milz, K. Amende, and H.-M. Gross. Complex-YOLO: Real-time 3D Object Detection on Point Clouds. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 1–14, Sept. 2018.

[21] L. N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. arXiv preprint arXiv:1803.09820, Mar. 2018.

[22] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2443–2451, waymods, June 2020.

[23] S. Vora, A. H. Lang, B. Helou, and O. Beijbom. PointPainting: Sequential Fusion for 3D Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[24] Y. Wang, A. Fathi, A. Kundu, D. Ross, C. Pantofaru, T. Funkhouser, and J. Solomon. Pillar-based Object Detection for Autonomous Driving. arXiv preprint arXiv:2007.10323, 2020.

[25] P. Wu, S. Chen, and D. N. Metaxas. MotionNet: Joint Perception and Motion Prediction for Autonomous Driving Based on Bird's Eye View Maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11382–11392, June 2020.

[26] 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 preprint arXiv:1911.12408, 2019.

[27] Y. Yan, Y. Mao, and B. Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10), 2018.

[28] B. Yang, W. Luo, and R. Urtasun. PIXOR: Real-Time 3D Object Detection From Point Clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7652–7660, June 2018.

[29] J. Yin, J. Shen, C. Guan, D. Zhou, and R. Yang. LiDAR-Based Online 3D Video Object Detection With Graph-Based Message Passing and Spatiotemporal Transformer Attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11495–11504, June 2020.

[30] Y. Zhou, P. Sun, Y. Zhang, D. Anguelov, J. Gao, T. Ouyang, J. Guo, J. Ngiam, and V. Vasudevan. End-to-End Multi-View Fusion for 3D Object Detection in LiDAR Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[31] Y. Zhou and O. Tuzel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[32] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu. Class-balanced Grouping and Sampling for Point Cloud 3D Object Detection. arXiv preprint arXiv:1908.09492, 2019.

[33] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei. Flow-Guided Feature Aggregation for Video Object Detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2017-Octob, pages 408–417, Oct. 2017.