MonoSIM: Simulating Learning Behaviors of Heterogeneous Point Cloud Object Detectors for Monocular 3-D Object Detection

Han Sun, Zhaoxin Fan, Zhenbo Song, Member, IEEE, Zhicheng Wang, Kejian Wu, and Jianfeng Lu, Member, IEEE

Abstract—Monocular 3-D object detection is a fundamental but very important task to many applications including autonomous driving, robotic grasping, and augmented reality. Existing leading methods tend to estimate the depth of the input image as auxiliary information for monocular 3-D detection. However, this routine suffers from the inherent gap between depth estimation and object detection, which would accumulate the prediction error. In this article, a novel training paradigm named MonoSIM is proposed. The insight behind introducing MonoSIM is that we propose to simulate the feature learning behaviors of two heterogeneous object detectors. During the training period, monocular detectors simulate the learning behaviors of the target point cloud-based detector. Hence, during the inference period, the steps of feature extraction and prediction process would be similar to the point cloud-based detector as possible. To achieve it, we propose one scene-level simulation module, one RoI-level simulation module, and one response-level simulation module, which are progressively used for the detector’s full feature learning and prediction pipeline. In this paradigm, we adopt the famous M3D-RPN, CaDDN, and GUPNet detectors, and conduct extensive experiments on KITTI and Waymo Open datasets. Results show that our method has achieved a minimum 1% performance improvement for different baselines on various datasets. When compared with current state-of-the-art methods, our method only imposes the almost negligible cost for training and no extra computational cost for inference. Our codes are publicly available at https://github.com/sunh18/MonoSIM.

Index Terms—Deep learning, heterogeneous detectors, monocular 3-D object detection, novel training paradigm, simulated learning behaviors.

I. INTRODUCTION

Accurate 3-D scene perception is an important component in many artificial intelligence scenarios, such as autonomous driving, robotics, and augmented reality [1], [2]. Previous algorithms and solutions based on LiDAR [3], [4], [5], [6] or stereo vision [7], [8] have achieved satisfactory detection performance. However, the high cost and equipment requirements limit the wide application of these methods. Therefore, the low-cost and easy-to-deploy monocular 3-D detection methods [9], [10] become alternative solutions and show great potential.

Existing monocular 3-D detection methods can be roughly divided into direct methods [10], [11], [12], [13] and depth-assisted methods [15], [16], [17], [18], [19], [20], [21], [22], [23]. The former directly predict 3-D bounding boxes from monocular images, while the latter estimate dense depth maps from the given images for point cloud-based 3-D object detectors or use auxiliary data to supplement depth cues for monocular detectors. Direct methods have the advantage of not requiring additional depth information when compared to depth-assisted methods. As a result, they reduce data acquisition costs and simplify system design. However, due to the lack of depth cues, the performance of direct methods in estimating the size and locating distant and occluded targets may decrease. Oppositely, depth-assisted methods can better capture 3-D shape information of targets and handle long-range and occlusion issues by utilizing additional depth information. Among the two kinds of methods, depth-assisted methods which utilize depth estimation as their main core, usually appear better performance since the recovered dense depth maps provide more cues for 3-D geometric and semantic perception. Nevertheless, though depth-assisted methods have achieved acceptable performance, we still face several certain issues.

1) The depth estimation and 3-D object detection are achieved by two different deep networks. The methods such as [15], [16], and [17] are usually trained separately or directly utilize off-the-shelf depth estimation networks. Therefore, there is a gap that the supervision signals and gradients for the 3-D object detection cannot flow back to the depth estimation network to guide its training.

2) Since both two tasks are easy to fall into the local optimal solution, the error of the two tasks would accumulate. Due to the accumulated error, the correct
prior information would not be used well. Such error will also have a negative impact on model optimization.

3) Introducing depth estimation inevitably increases computational costs compared to directly predicting objects from monocular images. For example, existing common methods that utilize depth estimation assistance (AM3D [17], D4LCN [15], and DDMP-3D [16]) typically have inference times of around 200 ms or even up to 400 ms for a single standard KITTI image. In contrast, advanced monocular methods that rely on various priors or constraints instead of estimating depth (MonoDLE [43], MonoRUn [44], and MonoFlex [46]) generally have inference times within 100 ms for a single image.

4) Recent work SGM3D [22] and MonoDistill [21] have circumvented the depth estimation and achieved significant improvements by utilizing existing depth maps as input to guide the monocular detector at the feature space. However, due to the network architecture constraints, they are not convenient to be applied as a flexible paradigm to other methods.

To tackle the above issues, we propose a novel monocular 3-D object detection training paradigm named MonoSIM. The essential insight behind MonoSIM is that we believe the basic principle of depth-assisted methods is to simulate the point cloud-based 3-D object detectors. Specifically, we find that most existing depth-assisted methods prefer to estimate depth maps and project them into the point cloud, or use auxiliary data to guide the training. These processes, in essence, are trying to simulate the input or features of the 3-D object detector. While depth-assisted methods have enhanced the performance of monocular detectors, the potential drawbacks include challenging training optimization, increased computational burden, and limitations in flexible application, which motivate us to think: Can we simulate the 3-D object detector in a more straightforward and flexible way?

The answer is positive. As shown in Fig. 1, in MonoSIM, we propose to simulate the feature learning behaviors of the 3-D object detector instead of simulating its input. To simulate the full feature learning of point cloud-based pipelines, MonoSIM consists of two branches and three modules. The two branches are called the point cloud branch and the monocular image branch, corresponding to the heterogeneous point cloud detector and monocular detector. The three modules are follows.

1) Scene-level simulation module, which aligns the shallow scene-level features of the point cloud detector and monocular detector, aims at increasing the monocular detector’s environmental understanding ability.

2) RoI-level simulation module, which simulates the feature characteristics of point cloud RoIs for monocular RoIs, targets to increase the monocular detector’s ability of finding and locating the potential objects.

3) Response-level simulation module, which uses prediction of the point cloud-based detector as soft labels to guide the loss computing step, aims at increasing the monocular detector’s ability to regress geometric properties of the bounding boxes. The above three modules would progressively help our model inherit the superior detection performance from the point cloud-based detector.

Since our method is model agnostic, it can be applied to many different monocular 3-D object detectors flexibly. In our work, we apply MonoSIM to the famous M3D-RPN [10], CaDDN [20], and GUPNet [47] detectors. Extensive experiments are conducted on KITTI [40], the currently most widely used benchmark, and Waymo Open dataset [41], the currently most large scale dataset. Experimental results show that our method consistently improves the performance of different monocular detectors for a large margin without changing their network architectures on both datasets.

Our contributions can be summarized as follows.

1) We propose MonoSIM, a novel paradigm that enables monocular 3-D object detection methods to simulate the feature learning process of point cloud-based detectors in scene-level, RoI-level, and response-level, respectively.

2) We apply our MonoSIM to several different existing monocular 3-D object detectors. Our method consistently and significantly improves their performance without changing their original network architectures.

3) We conduct extensive experiments on the most widely used KITTI benchmark and the most large-scale Waymo Open dataset to verify the effectiveness of our method.

II. RELATED WORK

A. Point Cloud-Based 3-D Object Detection

Point cloud-based 3-D object detection methods can roughly divided into point-based methods and grid-based methods [6]. Point-based methods are represented by PointNet [28] and PointNet++ [29], which directly extract features on the raw point cloud via deep networks. On this basis, PointRCNN [4] applies PointNET++ as the backbone to explore the accurate location of 3-D proposals. Point-based methods have the problems of high time cost and large amount of calculation, thus some solutions such as AVOD [37] and F-PointNet [36]...
are developed to reduce these effects, which use LiDAR point cloud and RGB images simultaneously. The former converts the point cloud into Bird’s Eye View (BEV) maps, then the feature maps of RGB images and BEV maps are obtained by the FPN network, and finally generates proposals after fusing the two types of features. The latter adopts 2-D region proposals to guide 3-D instance segmentation, reducing point cloud search. However, the quality of 2-D detection will have an uncertain impact on 3-D detection. Then, in grid-based methods, the point cloud is projected to regular grids [5], [25] or divided into voxels [26], [27], and then these divided points are sent to the full connection layer to construct a unified feature representation. Finally, the features are extracted by 2-D or 3-D CNN for prediction. These grid-based methods are generally efficient for accurate 3-D proposal generation, but the receptive fields are constrained by the kernel size of 2-D/3-D convolutions. Due to the sparse characteristics of the point cloud, the above methods often need to combine sparse convolution [30] or densification strategy [31] to enhance feature representation. In addition to mining the spatial clues of the point cloud itself, a class of methods represented by RI-Fusion [32] which can enrich point cloud features with images to achieve performance improvement of point cloud detectors.

**B. Monocular 3-D Object Detection**

Monocular 3-D object detection is a challenging task since recovering precise 3-D information from a single RGB image is an ill-posed problem. A straightforward line of these works proposes to directly predict 3-D objects from the image. For example, Mono3D [33] generates 3-D anchors via object contour and location assumption. Deep3DBox [13] utilizes the constraint relationship between the predicted 2-D boxes and the projected 3-D boxes to calculate new 3-D parameters of targets. DeepMANTA [34] and 3D-RCNN [35] propose to match 2-D object proposals and predefined 3-D CAD models to gradually refine the 3-D parameters. M3D-RPN [10] leverages a depth-aware network to generate 2-D and 3-D proposals simultaneously. Geometric projection is a useful technique that connects 2-D planes and 3-D depth. However, projecting 2-D planes to 3-D depth always leads to error amplification. To address this issue, GUPNet [47] introduces geometry uncertainty projection, which estimates depth uncertainty to help the model learn the depth information of the target. By inferring confidence from depth uncertainty, GUPNet reduces the problem of error amplification during projection. Beyond these methods, there are also many advanced methods to be introduced in recent years [10], [11], [12], and we kindly refer readers to the report [38] for more information about them.

Beyond direct prediction, many works focus on exploring the use of auxiliary data with rich depth cues to supplement information for monocular detectors. For example, PL-MONO [23] uses a depth estimator to generate a depth map by predicting the depth on each image pixel, and then projects it to the pseudo point cloud, which is then sent to the existing 3-D detector as LiDAR signals to predict the target boundary. Next, many other researchers [15], [17], [19], [20] explore different depth estimation strategies, different fusion methods, and different network architectures to try to improve the performance of depth-assisted methods. Representatively, CaDDN [20] performs both probabilistic depth estimation and training 3-D detection in an end-to-end fashion, which to some extent eliminates the disadvantages of depth estimation mentioned in Section I. However, the quality of additional spatial clues still depends on the depth distribution network branch, and its accuracy is weaker than that of point cloud containing explicit spatial information. Nevertheless, CaDDN [20] still achieves the state-of-the-art performance. Thanks to the additional estimated depth information, these methods always perform better than those direct prediction methods.

Some depth-assisted methods attempt to extract robust 3-D features from point clouds, depth maps, or stereo images to enhance feature extraction and target detection. MonoDistill [21] integrates knowledge distillation [24] and introduces a fully symmetric network architecture with two input branches for RGB images and dense depth maps. The interaction between the two modalities occurs in both the feature space and the result space. During training, the rich features from the depth branch guide the optimization of the RGB branch. The depth branch is deactivated during inference, and the model simplifies into a straightforward direct detector. This approach effectively aligns the gap between the feature representations of heterogeneous detectors, helping the monocular detector achieve better performance. However, there is a drawback in this approach: it requires training a teacher network using depth maps first instead of directly applying other pretrained superior point cloud-based detectors without training. Additionally, due to the limitations of homogeneous networks, the potential for better performance by the teacher network is debatable. SGM3D [22] also adopts a two-branch structure similar to MonoDistill, consisting of stereo and monocular branches. However, unlike MonoDistill, SGM3D explicitly transforms the 2-D image into 3-D space and aligns the features in the BEV representation. Thus, SGM3D has an advantage in mitigating the gap in feature representations between heterogeneous detectors. However, it still faces the limitation of not being able to flexibly utilize other preexisting point cloud detectors. Although LIGA-Stereo [48] is not a monocular method, it also uses the point cloud-guided learning strategy for geometric features and proves its feasibility. UVTR [14] integrates a transformer network to unify voxel representation, allowing image and point cloud features to interact spatially in two special designed voxel spaces.

In contrast, MonoSIM does not require the two branches of the network to be the same or similar. In other words, our method is model-agnostic, where the input and networks are both heterogeneous. The advantage is that we can easily apply MonoSIM to many different existing detectors without any changes or retraining the point cloud branch. Instead, we can train the monocular network directly after downloading a pretrained model file and building corresponding simulation modules between the point cloud and the image branches.
Fig. 2. Overall architecture of our proposed MonoSIM, which consists of the monocular image branch, the point cloud branch, and three simulation modules. First, we conduct a structural analysis by roughly dividing the two heterogeneous branches into the backbone, RoI feature extraction, and detection head layers, separately. Next, we flexibly extract scene-level and RoI-level features at the aforementioned three locations based on the specific network configuration. Then, specialized simulation modules at the scene-level and RoI-level are employed to align the features and minimize the expression differences between heterogeneous features. By simulating the feature learning behaviors of the point cloud detector with the monocular image detector, depth cues can be supplemented without additional depth estimation. Finally, at the output response layer, selected soft labels are used to supervise the monocular image branch to estimate the geometric parameters of the object’s pose. This completes the entire process of simulating and optimizing object detection.

III. METHOD

Fig. 2 illustrates the architecture of MonoSIM. In our work, we assume that both the point cloud-based detector and the monocular detector can be split into three subcomponents: the backbone scene-level feature extraction component, the RoI extraction component, and the prediction head with loss function. The above subcomponents constitute the full pipeline of the detectors’ feature learning. Therefore, to simulate the point cloud-based detector, we design the scene-level, RoI-level, and response-level simulation modules. Next, we will introduce them in detail.

A. Scene-Level Simulation Module

The shallow scene-level features are the basis for detectors to perceive the environment, which are usually generated in the front part of networks, so we consider drawing scene-level branches from the backbone part for simulation. The scene-level simulation module is shown in Fig. 3.

1) Scene-Level Feature Alignment: Since the monocular and point cloud-based detectors are completely different structures, the feature expression of different modalities will appear obvious differences. If we simply use the monocular image network to simulate the spatial features of points directly, the monocular network would deviate from the correct optimization direction. Therefore, it is necessary to design special alignment methods to narrow the modal gap.

Monocular scene-level features are defined as $F_{ms} \in \mathbb{R}^{C_{ms} \times H_{ms} \times W_{ms}}$, where $C_{ms}$ is the number of feature channels, $H_{ms}$ and $W_{ms}$ are the height and width of the monocular scene features. Since monocular scene features are already image features, we only need to match the channels of monocular and point cloud scene features. Specifically, denoting the channels provided by the point cloud-based detector is $C_{ps}$, we use a $1 \times 1$ Convolution + BatchNorm + ReLU layer to adjust $C_{ms}$ to $C_{ps}$. Therefore, the aligned monocular scene features $F'_{ms} \in \mathbb{R}^{C_{ps} \times H_{ms} \times W_{ms}}$ can be generated.

We assume that the features of the scene point cloud are $P_{ps} \in \mathbb{R}^{C_{ps} \times N_{ps}}$, where $N_{ps}$ is the number of scene feature points. Due to the modality differences, we need to convert point cloud features into image features. Specifically, we use the render method to convert points into the image plane.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
The features of scene feature points can be expressed as
\[ P_{ps} = \{p_{1}^{ps}, \ldots, p_{N_{pr}}^{ps}\} \in \mathbb{R}^{C_{pr} \times N_{pr}}, \quad Q_{ps} = \{q_{1}^{ps}, \ldots, q_{N_{pr}}^{ps}\} \in \mathbb{R}^{3 \times N_{pr}} \]
representing the 3-D space coordinates of scene feature points, internal parameter matrix of the camera coordinate system is \( K \) and external parameter matrix is \( RT \). We specify the height and width of the output features as \( H_{ms} \) and \( W_{ms} \). Using the PyTorch3D library \cite{3D} to render the scene features \( F_{ps} = \{f_{1}^{ps}, \ldots, f_{N_{pr}}^{ps}\} \in \mathbb{R}^{C_{pr} \times H_{ms} \times W_{ms}} \), and render operation can be defined as a function
\[ F_{ps} = \text{Render}(P_{ps}, Q_{ps}, K, RT, H_{ms}, W_{ms}). \] (1)

However, in some worthless or undefined zones of the rendered features, the point cloud-based detector cannot provide any effective spatial cues, where values are 0 in all channels. These zones should be deleted in the subsequent simulation, so it is necessary to generate a scene mask. Specifically, we denote the scene mask is \( M_{s}(u, v) \in \mathbb{R}^{H_{ms} \times W_{ms}} \), each channel of \( F_{ps} \) is \( f_{i}^{ps}(u, v) \), where \( i \in [1, C_{ps}], u \in [1, H_{ms}], v \in [1, W_{ms}] \), \( M_{s}(u, v) \) can be defined as
\[ M_{s}(u, v) = \begin{cases} 0 & \sum_{i=1}^{C_{ps}} f_{i}^{ps}(u, v) = 0 \\ 1 & \sum_{i=1}^{C_{ps}} f_{i}^{ps}(u, v) \neq 0. \end{cases} \] (2)

2) Scene-Level Simulation Loss: After obtaining the scene mask, we hope that in the limited zones, the monocular detector can simulate the feature distribution of the point cloud-based detector as much as possible. Specifically, we use the \( L_{1} \) norm to design scene-level simulation loss function \( L_{\text{scene}} \)
\[ L_{\text{scene}} = \frac{1}{n_{s}} \| M_{s}(F_{ms} - F_{ps}) \|_{1} \] (3)
where \( n_{s} \) is the number of valid scene features in \( M_{s} \).

B. RoI-Level Simulation Module

Monocular detector simulates the generation of scene-level features and aims to improve the understanding ability of the environment. However, in the entire detection space, the target zones generally occupy less space, where the shallow scene cues are not dense enough. To refine the detection, the monocular detector needs centralized RoI-level cues for simulation at these places. Different from the scene-level branches which are generally generated from the front part of detectors, RoI-level features are mainly generated from the middle or rear part of the network. Therefore, the position and form of the RoI-level branches are flexible. The RoI-level simulation module is shown in Fig. 4.

1) RoI-Level Feature Alignment: The above scene-level feature alignment has introduced a method to align point cloud and image features. If monocular RoI-level features are image type, we can also align the features by this method. Next, we propose another method to align BEV features in RoI-level feature alignment.

Monocular RoI-level features can be described as \( F_{mr} \in \mathbb{R}^{C_{mr} \times H_{mr} \times W_{mr}} \), where \( C_{mr} \) is the number of feature channels, \( H_{mr} \) and \( W_{mr} \) are the height and width of the monocular RoI features. Similar to the scene feature alignment, we adopt \( 1 \times 1 \) Convolution + BatchNorm + ReLU layer to adjust \( C_{mr} \) to \( C_{pr} \) and generate the aligned monocular RoI features \( F_{mr}' \in \mathbb{R}^{C_{pr} \times H_{mr} \times W_{mr}} \), where \( C_{pr} \) is number of feature channels of RoI feature points.

We denote the features of RoI points are \( P_{pr} = \{p_{1}^{pr}, \ldots, p_{N_{pr}}^{pr}\} \in \mathbb{R}^{C_{pr} \times N_{pr}} \), where \( N_{pr} \) is the number of RoI feature points. \( Q_{pr} = \{q_{1}^{pr}, \ldots, q_{N_{pr}}^{pr}\} \in \mathbb{R}^{3 \times N_{pr}} \) represents the 3-D space coordinates of RoI feature points. RoI feature points are divided into voxels with features \( V_{pr} \in \mathbb{R}^{C_{pr} \times X \times Y \times Z} \), where \( X, Y, \) and \( Z \) are the size of voxels. The BEV features \( B_{pr} \in \mathbb{R}^{C_{pr} \times X \times Y} \) are obtained by averaging the voxel features at the same \( (x, y) \), where \( x \in [1, X], y \in [1, Y] \). However, the size of \( B_{pr} \) may be inconsistent with the aligned monocular RoI features \( F_{mr}' \), thus our work adds average pooling to \( B_{pr} \) and generates point cloud RoI features \( F_{pr}' \in \mathbb{R}^{C_{pr} \times H_{mr} \times W_{mr}} \).

After generating \( F_{pr}' \), RoI mask \( M_{r} \in \mathbb{R}^{H_{mr} \times W_{mr}} \) also needs to be generated as same as \( M_{s} \). 2) RoI-Level Simulation Loss: We use the \( L_{1} \) norm to enforce the monocular detector to simulate the RoI-level feature provided by the point cloud-based detector, and RoI-level simulation loss \( L_{\text{RoI}} \) can be formulated as
\[ L_{\text{RoI}} = \frac{1}{n_{r}} \| M_{r}(F_{mr}' - F_{pr}') \|_{1} \] (4)
where \( n_{r} \) is the number of valid RoI features in \( M_{r} \).

C. Response-Level Simulation Module

To enhance the geometric parameter estimation of the object pose, we adopt the soft labels predicted by point cloud-based detectors to supervise the monocular network training.

Since the predicted soft labels have been completely aligned with the ground-truth labels in content and format, we directly replace the ground-truth labels with these soft labels. Response-level simulation loss \( L_{\text{response}} \) can be expressed as
\[ L_{\text{response}} = L_{\text{baseline}} \] (5)
where \( L_{\text{baseline}} \) is defined by the monocular network.

D. Total Simulation Loss

The total loss of the MonoSIM is the combination of the above three parts
\[ L = L_{\text{response}} + \lambda_{\text{scene}}L_{\text{scene}} + \lambda_{\text{RoI}}L_{\text{RoI}} \] (6)
where \( \lambda_{\text{scene}} \) and \( \lambda_{\text{RoI}} \) are fixed loss weighting factors.
E. Application

Note that MonoSIM is model agnostic, we apply it to different existing monocular 3-D object detectors to verify its effectiveness. In our work, we decided to let the monocular detectors simulate the behaviors of PV-RCNN [6] due to PV-RCNN’s superior performance in 3-D object detection and its wide usage. Then, we choose M3D-RPN [10], GUPNet [47], and CaDDN [20] as our baseline monocular detectors. M3D-RPN is a classic anchor-based method while GUPNet is anchor-free, CaDDN is one of the current state-of-the-art BEV-based methods.

The different structures of the four detectors determine the different approaches of dividing scene-level, RoI-level, and response-level components. To avoid any changes on the original networks, our method MonoSIM needs to draw scene-level, RoI-level, and response-level branches from given detectors, considering their characteristics. Please refer to our Github link for more details about the architecture of applying MonoSIM on PV-RCNN, CaDDN, GUPNet, and M3D-RPN.

1) Point Cloud Branch: PV-RCNN integrates 3-D voxel CNN and PointNet-based set abstraction, leveraging the efficient learning and high-quality proposals from the former, and the flexible receptive fields from the latter. We think that the “Voxel Set Abstraction Module” of PV-RCNN encodes the multiscale features from 3-D CNN which contains complete scene information, therefore, we extract scene-level point features $P_{ps}$ from the output of “Voxel Set Abstraction Module.” On the KITTI dataset, $P_{ps} \in \mathbb{R}^{544 \times 4096}$. On the Waymo Open dataset, $P_{ps} \in \mathbb{R}^{256 \times 21600}$.

PV-RCNN is a two-stage detection framework and adopts an RPN network to generate 3-D proposals for RoI grids, this means the RoI grids are listed as potential target areas with centralized features. Therefore, RoI-level point features $P_{pr}$ are extracted from the output of the two-layer MLP after the “RoI-grid Pooling Module.” On both KITTI and Waymo Open datasets, $P_{pr} \in \mathbb{R}^{256 \times 21600}$.

2) Monocular Image Branch:

1) CaDDN adopts ResNet101 [49] as the image backbone to acquire shallow environmental features, and then fuses the channel reduced features with estimated depth information. Therefore, we extract features after “Image Channel Reduce” as the image scene features $F_{ms} \in \mathbb{R}^{64 \times 94 \times 311}$. For RoI-level simulation, we hope that monocular features can approach the distribution of point RoI features as possible before entering the detection head. CaDDN’s detection head takes BEV features as the input, so we branch out the RoI-level features $F_{mr}$ from here. $F_{mr} \in \mathbb{R}^{384 \times 188 \times 140}$.

2) GUPNet adopts CenterNet [50] as its baseline, while CenterNet’s backbone and neck are DLA34 and DLAUp, respectively. DLA34 iteratively integrates the feature information of each layer of the network, aggregating semantic and spatial information, making it suitable to use the network output as the image scene features $F_{ms} \in \mathbb{R}^{32 \times 12 \times 40}$. GUPNet uses DLAUp (neck) to merge output features from the backbone network iteratively and then obtains potential 2-D RoI regions through the 2-D detection head to align the RoI features. Thus, it can be considered that the neck output features contain relatively clear semantic information before feeding into 3-D detection heads. Therefore, we use the neck output features as the image RoI features $F_{mr} \in \mathbb{R}^{64 \times 96 \times 320}$.

3) M3D-RPN is a one-stage detector leveraging the geometric relationship of 2-D and 3-D perspectives, without any extra components, thus the selection of scene-level and RoI-level branches is quite different from the CaDDN and GUPNet. M3D-RPN designs two parallel paths referred to global and local which are connected with the end of the image backbone. The features, which are sent into each path, are convoluted by each proposal feature extraction layer to generate new features ($F_{global}$ and $F_{local}$). $F_{global}$ and $F_{local}$ are then connected to kernels which can be divided into classification and regression branches. Considering point RoI features $P_{pr}$ which contains classification information from PV-RCNN, we extract features from the classification branch as image RoI features, and features from the regression branch as image scene features.

On the global path, we define image scene and RoI features as $F_{ms\_glo} \in \mathbb{R}^{C_{ms\_glo} \times H_{ms\_glo} \times W_{ms\_glo}}$ and $F_{mr\_glo} \in \mathbb{R}^{C_{mr\_glo} \times H_{mr\_glo} \times W_{mr\_glo}}$, separately. On the local path, we define image scene and RoI features as $F_{ms\_loc} \in \mathbb{R}^{C_{ms\_loc} \times H_{ms\_loc} \times W_{ms\_loc}}$ and $F_{mr\_loc} \in \mathbb{R}^{C_{mr\_loc} \times H_{mr\_loc} \times W_{mr\_loc}}$ separately. $C$, $H$, and $W$ represent the channel number, height, and width of the corresponding features, respectively.

The reason for this definition is to follow the design of M3D-RPN which splits into global and local paths. We send $F_{ms\_glo}$ and $F_{ms\_loc}$ into the scene-level simulation module to simulate with $P_{ps}$ at the same time. Therefore, we can get scene-level simulation loss of $F_{ms\_glo}$ and $F_{ms\_loc}$ which are denoted as $L_{scene\_glo}$ and $L_{scene\_loc}$. In the same way, we can obtain RoI-level simulation loss of $F_{mr\_glo}$ and $F_{mr\_loc}$ which are denoted as $L_{RoI\_glo}$ and $L_{RoI\_loc}$.

To leverage the global and local simulation, we use two learned weighting factors to fuse them

$$L_{scene} = \alpha \cdot L_{scene\_glo} + (1 - \alpha) \cdot L_{scene\_loc} \quad (7)$$

$$L_{RoI} = \beta \cdot L_{RoI\_glo} + (1 - \beta) \cdot L_{RoI\_loc} \quad (8)$$

where $\alpha$ and $\beta$ are learnable parameters after sigmoid. On the KITTI dataset, $F_{ms\_glo} \in \mathbb{R}^{512 \times 32 \times 110}$, $F_{mr\_glo} \in \mathbb{R}^{512 \times 32 \times 110}$, $F_{ms\_loc} \in \mathbb{R}^{512 \times 32 \times 110}$, and $F_{mr\_loc} \in \mathbb{R}^{512 \times 32 \times 110}$. On the Waymo Open dataset, $F_{ms\_glo} \in \mathbb{R}^{512 \times 32 \times 48}$, $F_{mr\_glo} \in \mathbb{R}^{512 \times 32 \times 48}$, $F_{ms\_loc} \in \mathbb{R}^{512 \times 32 \times 48}$, and $F_{mr\_loc} \in \mathbb{R}^{512 \times 32 \times 48}$.
IV. EXPERIMENTS

A. Dataset

To verify the effectiveness of our method, we conduct experiments on the KITTI dataset [40] and Waymo Open dataset [41].

KITTI dataset is one of the most widely used 3-D detection datasets. It contains 7481 training samples and 7518 test samples [40]. The training samples are divided into train set (3712 samples) and val set (3769 samples) [20]. We train the model and conduct ablation studies on the train set and val set, and submit the results of the test set to their lead board to compare with the other existing state-of-the-art methods. Following previous methods, we focus on the “Car” category in the KITTI dataset.

Waymo Open dataset is the recently released large-scale autonomous driving 3-D detection dataset, which consists of 798 training sequences, 202 validation sequences, and 150 test sequences [41]. Due to the large amount of data and high frame rate, we sample 20% of training sequences and val sequences to form the train set (30926 samples) and the val set (7839 samples). We detect vehicles, pedestrians, and cyclists in Waymo annotations from images captured by the front camera.

B. Implementation Details

We implement all our code using PyTorch. We use M3D-RPN [10] to conduct experiments on both Waymo Open dataset [41] and KITTI dataset [40], use CaDDN [20] and GUPNet [47] to conduct experiments on KITTI dataset. During training, we remove the original flipping operation in data augment. For training CaDDN on the KITTI dataset, we adopt the Adam optimizer with a batch size of 2. The learning rate is 0.0002. The network is trained for 10 epochs. For training M3D-RPN, we adopt the SGD optimizer with a batch size of 2. The learning rate is 0.004 with a poly decay rate using power 0.9 and eight decay of 0.9. The max iteration of the model is 200,000. For training GUPNet, we adopt the Adam optimizer with a batch size of 2. The learning rate is set as 0.00125 and the max epoch is 140. The decay rate is 0.1 and decay epochs are 90 and 120. All experiments are conducted on a single Tesla V100 (32G) GPU.

C. Results on the KITTI Dataset

We show the performance of our method in Table I and compare it with some state-of-the-art methods. The baseline methods are CaDDN [20], M3D-RPN [10], and GUPNet [47] separately. The evaluation metric is the standard average precision (AP_{3D@0.7} (40 recall positions). Unless specified, the metrics AP_{3D@0.7} and BEV on the KITTI dataset all take IoU = 0.7 in this article. In Table I, it can be obviously found that our MonoSIM improves the three baselines for large margins without changing their original network architectures. Superficially, the AP_{3D@0.7} of MonoSIM based on CaDDN is increased by 1.14%, 0.33%, and 0.85% on easy, moderate, and hard difficulty levels, respectively. On M3D-RPN, MonoSIM achieves a greater improvement. MonoSIM based on GUPNet achieves the best performance among these three baselines by 21.69%, 14.74%, and 13.08% on easy, moderate, and hard difficulty levels, respectively. We contribute the improvement to our training pipeline: simulation on the feature learning behaviors of existing point cloud-based detectors. We can also find that our method outperforms most existing depth-assisted methods. In contrast, to simulate the input of point cloud-based detectors, we simulate their feature learning behaviors, so our method can learn stronger features, hence our performance is better.

D. Results on the Waymo Open Dataset

Table II shows the performance of MonoSIM on Waymo Open dataset. We adopt the official metrics: the mean average precision (mAP) and the mean average precision weighted by heading (mAPH) to evaluate the methods. The evaluation levels are officially defined to two levels (LEVEL_1, LEVEL_2) according to detection difficulty. IoU thresholds are set to 0.7 and 0.5, respectively, to compute the metrics. It can be found that our method improves the performance of the baseline on nearly all difficulty levels and evaluation thresholds. For instance, the overall mAP is improved by 1.25% and 1.16% at level 1 and level 2, respectively, when the IoU threshold is 0.7, which is a significant improvement. When the IoU threshold is 0.5, the improvement is more obvious, for level 1, it is increased from 3.79% to 8.16%, and for level 2, it is increased from 3.61% to 7.58%. Compared
TABLE I

| Method            | Auxiliary Data | Is estimated? | AP_{3D,RoI}@IoU=0.7 | BEV@IoU=0.7 | Runtime (ms) |
|-------------------|----------------|---------------|----------------------|--------------|--------------|
|                   |                |               | easy      | moderate | hard | easy      | moderate | hard |          |
|                   |                |               | 16.50     | 10.74   | 9.52 | 25.03     | 17.32    | 14.91 | 400 |
| AMD [17]          | Depth          | ✓             |           |         |      |           |          |      |      |
| D4LCN [15]       | Depth          | ✓             | 16.65     | 11.72   | 9.51 | 22.51     | 16.02    | 12.55 | 200 |
| Kinem3D [19]     | Multi-frames   | ✓             | 19.07     | 12.72   | 9.17 | 26.69     | 17.52    | 13.10 | 120 |
| CaDDN [20]       | Depth          | ✓             | 19.17     | 13.41   | 11.46 | 27.94     | 18.91    | 17.19 | 630 |
| DDMP-3D [16]     | Depth          | ✓             | 19.71     | 12.78   | 9.80 | 28.08     | 17.89    | 13.44 | 180 |
| SGM3D [22]       | Stereo         | ✓             | 22.46     | 14.65   | 12.97 | 31.49     | 21.37    | 18.43 | 30  |
| MonoDistill [21] | Depth          | ✓             | 22.97     | 16.03   | 13.60 | 31.87     | 22.59    | 19.72 | 40  |
| M3D-RPN [10]     | -              |               | 14.76     | 9.71    | 7.42 | 21.02     | 13.67    | 10.23 | 160 |
| MonoDLE [45]     | -              |               | 17.23     | 12.26   | 10.29 | 24.79     | 18.89    | 16.00 | 40  |
| MonoRun [44]     | -              |               | 19.65     | 12.30   | 10.58 | 27.94     | 17.34    | 15.24 | 70  |
| MonoRCNN [45]    | -              |               | 18.36     | 12.65   | 10.03 | 25.48     | 18.11    | 14.10 | 70  |
| MonoFlex [46]    | -              |               | 19.94     | 13.89   | 12.07 | 28.23     | 19.75    | 16.89 | 30  |
| GUPNet [47]      | -              |               | 20.11     | 14.20   | 11.77 | -         | -        |      | -   |

MonoSIM (based on CaDDN)

| Method            |          | Is estimated? | AP_{3D,RoI}@IoU=0.7 | BEV@IoU=0.7 | Runtime (ms) |
|-------------------|----------|---------------|----------------------|--------------|--------------|
|                   |          |               | easy      | moderate | hard | easy      | moderate | hard |          |
| MonoSIM (based on M3D-RPN) | Depth | ✓ | +1.14 | +0.33 | +0.85 | +0.33 | +0.98 | +0.77 | 140 |
| Improvement on M3D-RPN | Depth | ✓ | +2.17 | +0.98 | +1.67 | +2.70 | +1.06 | +2.13 | 160 |
| MonoSIM (based on GUPNet) | Depth | ✓ | +1.58 | +0.54 | +1.31 | - | - | - | 30 |

TABLE II

| Difficulty | Method            | 3D mAP overall | 3D mAP 0-30m | 3D mAP 30-50m | 3D mAP 50m-∞ | AP_{bev,mono} overall | AP_{bev,mono} 0-30m | AP_{bev,mono} 30-50m | AP_{bev,mono} 50m-∞ |
|------------|-------------------|----------------|--------------|---------------|---------------|----------------------|----------------------|----------------------|----------------------|
| LEVEL_1    | MonoDistill [21]  | 0.42           | 1.23         | 0.14          | 0.03          | 0.25                 | 0.74                 | 0.08                 | 0.02                 |
|            | M3D-RPN [10]      | 0.35           | 1.12         | 0.18          | 0.02          | 0.34                 | 1.10                 | 0.18                 | 0.02                 |
|            | MonoSIM           | 1.60           | 6.08         | 4.00          | 0.01          | 1.59                 | 6.02                 | 0.39                 | 0.01                 |
| LEVEL_2    | MonoDistill [21]  | 0.40           | 1.23         | 0.13          | 0.03          | 0.23                 | 0.73                 | 0.08                 | 0.02                 |
|            | M3D-RPN [10]      | 0.33           | 1.12         | 0.18          | 0.02          | 0.33                 | 1.10                 | 0.17                 | 0.02                 |
| LEVEL_5    | MonoSIM           | 1.49           | 6.05         | 3.38          | 0.004         | 1.48                 | 6.00                 | 0.38                 | 0.004                |

MonoSIM (based on GUPNet)

| Difficulty | Method            | 3D mAP overall | 3D mAP 0-30m | 3D mAP 30-50m | 3D mAP 50m-∞ | AP_{bev,mono} overall | AP_{bev,mono} 0-30m | AP_{bev,mono} 30-50m | AP_{bev,mono} 50m-∞ |
|------------|-------------------|----------------|--------------|---------------|---------------|----------------------|----------------------|----------------------|----------------------|
| LEVEL_1    | MonoDistill [21]  | 6.32           | 12.56        | 5.24          | 1.49          | 3.82                 | 7.35                 | 3.68                 | 1.10                 |
|            | M3D-RPN [10]      | 3.79           | 11.14        | 2.16          | 0.26          | 3.63                 | 7.09                 | 2.07                 | 0.21                 |
|            | MonoSIM           | 8.16           | 25.86        | 3.62          | 0.08          | 8.04                 | 25.48                | 3.55                 | 0.08                 |
| LEVEL_2    | MonoDistill [21]  | 5.87           | 12.50        | 5.06          | 1.28          | 3.55                 | 7.32                 | 3.55                 | 0.94                 |
|            | M3D-RPN [10]      | 3.61           | 11.12        | 2.12          | 0.24          | 3.46                 | 10.67                | 2.04                 | 0.20                 |
| LEVEL_5    | MonoSIM           | 7.58           | 25.75        | 3.49          | 0.07          | 7.47                 | 25.37                | 3.43                 | 0.07                 |

Fig. 5. To visualize the performance (AP_{3D,RoI}@IoU=0.7) changes of MonoSIM with different combinations of λ_{scene} and λ_{RoI}, we present heatmaps where (a)–(c) correspond to easy, moderate, and hard categories, respectively.

with the current state-of-the-art MonoDistill [21], MonoSIM improves the overall mAP by 1.18% and 1.09% at level 1 and level 2, respectively, when the IoU threshold is 0.7. The above results greatly demonstrate the effectiveness of MonoSIM.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
In general, these 63 combinations of hyperparameters (λ) different AP values. As shown in Fig. 5, MonoSIM based on CaDDN exhibits the impact of various combinations of hyperparameters. We have attempted to follow some of its hyperparameter settings. For example, the model’s performance reaches the highest average on the KITTI val set with the smallest standard deviation, that is, the performance is the best and most stable. However, the difference is really minimal, Table IV presents the performance comparison of MonoSIM on the KITTI val and test sets under the optimal (λscene = 5, λRot = 3) and suboptimal (λscene = 1, λRot = 1) combinations. In terms of AP3D|R40, “λscene = 1, λRot = 1” slightly outperforms, while in BEV evaluation, “λscene = 5, λRot = 3” has a slight advantage. Therefore, we retained both parameter configurations and updated the detection results with “λscene = 5, λRot = 3” on the KITTI benchmark.

It is worth noting that when λscene or λRot increases to a certain value, or directly follows the larger weights settings of MonoDistill, our model performance seems to slightly decrease. To analyze the underlying reasons, we think it is because MonoDistill uses an affinity map based on region similarity encoding and the affinity map is equivalent to extracting the interregional correlation information of the scene-level features preliminarily. Therefore, although larger weights are used to reinforce the similarity between the teacher and student networks directly in MonoDistill, this strategy may not be suitable for our MonoSIM.

E. Hyperparameter Analysis

Considering the significant improvement of MonoDistill [21] on detector performance, and its commonalities with our method in simulating learning behaviors, we have attempted to follow some of its hyperparameter settings. For example, λscene = 10, λRot = 1. Under such settings, the AP3D|R40 of MonoSIM based on CaDDN on the KITTI val set are 24.44%, 16.66%, and 14.91% for easy, moderate, and hard order, respectively.

We have conducted more additional experiments to analyze the impact of various combinations of hyperparameters. As shown in Fig. 5, MonoSIM based on CaDDN exhibits different AP3D|R40 performances on the val set of the KITTI. In general, these 63 combinations of hyperparameters (λscene ∈ {0.2, 0.5, 1, 2, 3, 4, 5, 6, 7}, λRot ∈ {0.2, 0.5, 1, 2, 3, 4, 5}) have some impact on the performance fluctuations, but the magnitude is not significant. To select the best combination λscene and λRot, we separately calculated MonoSIM’s average value and standard deviation when fixing either λscene or λRot based on the results shown in Fig. 5. We aimed to balance detection accuracy and robustness by maximizing the average value while minimizing the standard deviation. The results are presented in Table III.

When λscene = 5 and λRot = 3, it can be considered that the model’s performance reaches the highest average on the KITTI val set with the smallest standard deviation, that is, the performance is the best and most stable. However, the

| Table III |
| --- |
| WHEN λSCENE OR λROTI IS FIXED, CALCULATE THE AVERAGE AND STANDARD DEVIATION OF AP3D|R40|IoU = 0.7. MonoSIM IS BASED ON CaDDN. * INDICATES OPTIMAL, AND † INDICATES SUBOPTIMAL. |
| Value | Average | Standard Deviation |
| --- | --- | --- |
| 0.2 | 24.85 | 0.31 |
| 0.5 | 24.59 | 0.16 |
| 1 | 24.81 | 0.16 |
| 2 | 24.79 | 0.12 |
| 3 | 24.90† | 0.11 |
| 4 | 24.76 | 0.10 |
| 5 | 25.07* | 0.10 |
| 6 | 24.71 | 0.09 |
| 7 | 24.48 | 0.08 |

| Table IV |
| --- |
| MonoSIM’S PERFORMANCE ON THE KITTI VAL AND TEST SETS WHEN THE COMBINATION OF HYPERPARAMETERS IS OPTIMAL AND SUBOPTIMAL SEPARATELY. |
| Set | AP3D|R40|IoU=0.7 | BEV|IoU=0.7 |
| --- | --- | --- | --- |
| optimal | 25.07 | 16.96 | 15.11 | 33.50 |
| test | 20.24 | 13.72 | 12.29 | 28.68 |
| sub-optimal | 25.13 | 16.98 | 15.09 | 33.86 |
| test | 20.31 | 15.74 | 12.31 | 28.27 |

Fig. 6. Confidence distribution of PV-RCNN’s prediction results [6] on the KITTI train and val sets.
more wrong or inaccurate classification and location information, which interferes with the training of the monocular detector, so it performs worse than using ground-truth labels. The performance on the Car category is improved when the confidence threshold is set to 0.9, but it still does not reach the level of ground-truth labels due to few reserved annotations. When the threshold value is set to 0.7, the quality and quantity of supervision signals reach a good balance, the monocular detector achieves the best performance, thus we set 0.7 as the threshold to filter soft labels for response-level simulation, which are then used to supervise the training of the monocular detectors.

Finally, we set 0.7, 0, and 0 as the thresholds for the Car, Pedestrian, and Cyclist categories respectively. In Table VI, we compare the number of filtered samples and the ground-truth samples on the KITTI train and val sets.

### G. Ablation Studies

In this section, we conduct ablation studies on the KITTI val set to verify the effectiveness of our three simulation modules. CaDDN [20], M3D-RPN [10], and GUPNet [47] are used as our baselines. Tables VII–IX show the results. In the tables, SFS refers to the scene-level simulation module, RFS refers to the RoI-level simulation module, and RLS refers to the response-level simulation module.

#### 1) Effects of Response-Level Simulation:

In our work, the response-level simulation module uses the prediction of the point cloud-based detector as soft labels to guide the loss computing step, aiming at increasing the monocular detector’s ability to regress geometric properties of the bounding boxes. Experiments 2, 7, and 11 show that by adding this module, the AP$_{3D|R40}$ is increased by 0.63%, 0.37%, and 1.1% using CaDDN as the baseline. The AP$_{3D|R40}$ is increased by 3.17%, 1.42%, and 1.55% using M3D-RPN as baseline. The AP$_{3D|R40}$ of MonoSIM based on GUPNet increased by 1% at the easy level, with slight fluctuations of −0.2% and −0.03% at the moderate and hard levels, respectively. M3D-RPN’s improvement on the easy level is very significant. That is because M3D-RPN is an early work with a relatively simple network architecture, which makes it hard for itself to learn strong features. So, after using our response-level simulation module, the behaviors of the point cloud based detector would greatly help it to make up for its disadvantages.

#### 2) Effects of Scene-Level Simulation:

Experiments 3, 8, and 13 show the results of adding scene-level simulation on the basis of the response-level simulation module. The scene-level simulation module aims at aligning the monocular detectors’ shallow scene-level features with point cloud-based detectors, so that we can further increase the monocular detector’s performance. We find that this module also improves the performance of all baselines at different difficult levels. Although the improvement is not significant when CaDDN and M3D-RPN are used as baselines, there is an obvious improvement when GUPNet is used as the baseline. This is because this module works at the early stage of the feature learning process and helps the model extract the basic environmental features better. Different baselines have varying capabilities in understanding these scene-level features. These features are not directly corresponded to the objects, but they are also helpful.

#### 3) Effects of RoI-Level Simulation:

Experiments 4, 9, and 14 show the results of adding the RoI-level simulation module.
on the basis of the response-level simulation module. The RoI-level simulation module simulates the feature characteristics of point cloud RoIs for monocular RoIs, aiming at increasing the monocular detector’s ability of finding and locating the potential objects. It can be seen from the tables that after adding this module, performance has been further improved compared with only using the RLS module, which demonstrates the effectiveness of RFS.

4) Full Performance on KITTI Val Set: Experiment 5, 10, and 15 are our final version which adopts the full advantages of RLS, SFS, and RFS. Compared with baselines on the KITTI val set, our full MonoSIM improves the performance of the CaDDN detector by 1.56%, 0.67%, and 1.21%, improves the performance of the M3D-RPN detector by 4.38%, 2.39%, and 2.49%, and improves the performance of the GUPNet detector by 1.45%, 0.61%, and 1.39% on \( \text{AP}_{3D|R40} \).

H. Computational Burden Analysis

We compared the computational burden of the state-of-the-art approach MonoDistill [21] and our proposed MonoSIM. Tables X and XI show the related baselines and components during training and inference within the model file, along with their corresponding flops and params. MonoDistill’s baseline is M3D-LRE [43], and MonoSIM’s baseline is GUPNet [47].

By reproducing MonoDistill’s experiment, it is evident that MonoDistill has almost twice the flops and parameters during the training period compared to the inference period. However, the increase in parameters for our MonoSIM is almost negligible, and there is decrease in flops. This is because GUPNet itself has different computational steps during training and inference, which is unrelated to the design of MonoSIM. During inference, both MonoSIM and MonoDistill can avoid additional computational burden.

We investigated the significant difference in computational burden between MonoDistill and MonoSIM methods during training. MonoDistill uses a depth network that is identical to the monocular baseline structure, along with a feature fusion network and some adapt layers, including convolutional queues composed of multiple kernels. On the other hand, MonoSIM presaves the rendered features of the point cloud network and loads them directly during training, which effectively reduces memory usage. The scene-level simulation and RoI-level simulation in MonoSIM use simple and fewer \( 1 \times 1 \) convolutions to achieve feature alignment. The response-level simulation uses soft labels to supervise the training and does not increase any computing burden due to no extra structure, which is similar to MonoDistill.

Overall, during training, MonoSIM’s flops and parameters are significantly fewer than MonoDistill’s, while during inference, MonoSIM can ensure no additional computational burden, which is beneficial for model application.

I. Pedestrian/Cyclist Detection

The detection performance of Pedestrian and Cyclist on the KITTI test set is shown in Table XII. As the GUPNet [47] paper did not publicly release their detection metrics for pedestrians and cyclists, the data in the table are obtained from our reproduction of their experiments.

Compared to the baseline CaDDN, MonoSIM achieved a comprehensive improvement in the Pedestrian category, but a decrease in the Cyclist category. While MonoSIM greatly enhanced the detection capability of Cyclist compared to GUPNet, it showed a slight decrease in the Pedestrian category. We think the reason should be due to insufficient training samples. As the number of samples shown in Table VI, the filtered samples belonging to Cyclist are quite insufficient. Although the number of Pedestrian samples significantly increased after filtering, it is still less than half of the number of Car samples. This imbalanced sample distribution makes it difficult for the model to fully learn the characteristics of different targets. Additionally, it also indicates that there are differences in the ability of different baselines to simulate learning behaviors. However, MonoSIM is still helpful to improve the object detection capability of different baselines to some extent.
Fig. 7. Qualitative results of MonoSIM based on CaDDN [20]. Samples are all from the KITTI val set on the Car category. Red, blue, and green bounding boxes represent ground-truth, baseline and MonoSIM, separately. Depth refers to the distance of the object in front of the camera. In (a)–(f), the baseline cannot perceive the objects while MonoSIM detects them correctly. In (g)–(i), MonoSIM can refine the detected 3-D bounding boxes of the baseline at different distances.

J. Qualitative Analysis

We choose CaDDN [20] and GUPNet [47] as baselines. As shown in Fig. 7(a) and (b), CaDDN is unable to detect the objects that seriously occluded, while MonoSIM performs well. When the target is too far and the baseline fails [see Fig. 7(c)–(f)], MonoSIM still has a greater perceptive ability. Thanks to the spatial cues provided by distant point cloud, MonoSIM can improve the generation of features by learning the cues which are usually lost or incomplete in monocular images. Fig. 7(g)–(i) shows that MonoSIM can also improve the detection performance at different distances, refining the 3-D bounding boxes significantly.

The comparison of the detection performance of GUPNet and MonoSIM on all categories in the KITTI test set is shown in Fig. 8. From the figure, it is evident that MonoSIM has an edge over GUPNet when it comes to handling dense occlusions and distant vehicles. Moreover, it performs better in detecting pedestrians and shows good performance in detecting cyclists in some scenarios.

V. Conclusion

In this work, we propose MonoSIM, a novel monocular 3-D object detection training paradigm, which aims at simulating the feature learning behaviors of superior point cloud-based
detectors. In MonoSIM, one scene-level simulation module, one RoI-level simulation module, and one response-level simulation module are proposed to progressively simulate the full training pipeline of the point cloud-based detector. We have combined MonoSIM with existing PV-RCNN [6], M3D-RPN [10], CaDDN [20], and GUPNet [47] detectors. Experiments on the KITTI dataset [40] and Waymo Open dataset [41] have demonstrated the effectiveness of our method.

However, due to the flexibility and heterogeneity of the point cloud and image baseline models, MonoSIM has not achieved state-of-the-art performance in some metrics. Balancing the flexibility of the paradigm and eliminating data modality differences proves to be a challenging task. It is worth further exploration to investigate whether there exists an interpretable theory in this regard.

REFERENCES

[1] S. Shirmohammadi and A. Ferrero, “Camera as the instrument: The rising trend of vision based measurement,” IEEE Instrum. Meas. Mag., vol. 17, no. 3, pp. 41–47, Jun. 2014.
[2] R. G. Lins, S. N. Givigi, and P. R. G. Kurka, “Vision-based measurement for localization of objects in 3-D for robotic applications,” IEEE Trans. Instrum. Meas., vol. 64, no. 11, pp. 2950–2958, Nov. 2015.
[3] X. Li, Y. Zhou, and B. Hua, “Study of a multi-beam LiDar perception assessment model for real-time autonomous driving,” IEEE Trans. Instrum. Meas., vol. 70, pp. 1–15, 2021.
[4] S. Shi, X. Wang, and H. Li, “PointRCNN: 3D object proposal generation and detection from point cloud,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 770–779.
[5] 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. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 12687–12705.
[6] S. Shi et al., “PV-RCNN: Point-voxel feature set abstraction for 3D object detection,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 10529–10538.
[7] P. Li, X. Chen, and S. Shen, “Stereo R-CNN based 3D object detection for autonomous driving,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 7644–7652.
[8] Y. Chen, S. Shen, and J. Hua, “DSGN: Deep stereo geometry network for 3D object detection,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 12536–12545.
[9] Y. Cai et al., “YOLOv4–5D: An effective and efficient object detector for autonomous driving,” IEEE Trans. Instrum. Meas., vol. 70, pp. 1–13, 2021.
[10] G. Brazil and X. Liu, “M3D-RPN: Monocular 3D region proposal network for object detection,” in Proc. ICCV, Oct. 2019, pp. 9286–9295.
[11] L. Liu, J. Lu, C. Xu, Q. Tian, and J. Zhou, “Deep fitting degree scoring network for monocular 3D object detection,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 1057–1066.
[12] H. Hu, T. Zhao, Q. Wang, F. Gao, L. He, and Z. Gao, “Monocular 3-D vehicle detection using a cascade network for autonomous driving,” IEEE Trans. Instrum. Meas., vol. 70, pp. 1–13, 2021.
[13] V. Desjardins, D. Anguelov, J. Flynn, and J. Kosecka, “3D bounding box estimation with deep learning and geometry,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 7074–7082.
[14] Y. Li, Y. Chen, X. Qi, Z. Li, J. Sun, and J. Jia, “Unifying voxel-based representation with transformer for 3D object detection,” in Proc. NIPS, 2022, pp. 18442–18455.
[15] M. Ding et al., “Learning depth-guided convolutions for monocular 3D object detection,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW), Jun. 2020, pp. 4306–4315.
[16] L. Wang et al., “Depth-conditioned dynamic message propagation for monocular 3D object detection,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2021, pp. 454–463.
[17] X. Ma, Z. Wang, H. Li, P. Zhang, W. Ouyang, and X. Fan, “Accurate monocular 3D object detection via color-embedded 3D reconstruction for autonomous driving,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 6851–6860.
[18] Y. Kim, S. Kim, S. Sim, J. W. Choi, and D. Kum, “Boosting monocular 3D object detection with object-centric auxiliary depth supervision,” IEEE Trans. Intell. Transp. Syst., vol. 24, no. 2, pp. 1801–1813, Feb. 2022.
[19] G. Brazil, G. Pons-Moll, X. Liu, and B. Schiele, “Kinematic 3D object detection in monocular video,” in Proc. ECCV, 2020, pp. 135–152.
[20] C. Reading, A. Harakeh, J. Chae, and S. L. Waslander, “Categorical depth distribution network for monocular 3D object detection,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2021, pp. 8555–8564.
[21] Z. Chong et al., “MonoDistill: Learning spatial features for monocular 3D object detection,” 2022, arXiv:2201.10830.
[22] Z. Zhou et al., “SGM3D: Stereo guided monocular 3D object detection,” IEEE Robot. Autom. Lett., vol. 7, no. 4, pp. 10478–10485, Oct. 2022.
[23] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger, “Pseudo-LiDar from visual depth estimation: Bridging the gap in 3D object detection for autonomous driving,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 8445–8453.
[24] G. Hinton et al., “Distilling the knowledge in a neural network,” 2015, arXiv:1503.02531.
[25] B. Yang, M. Liang, and R. Urtasun, “HDNet: Exploiting hd maps for 3D object detection,” in Proc. Conf. Robot Learn., 2018, pp. 146–155.
[26] Y. Zhou and O. Tuzel, “VoxelNet: End-to-end learning for point cloud based 3D object detection,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 4490–4499.
[27] J. Deng, W. Zhou, Y. Zhang, and H. Li, “From multi-view to hollow-3D: Hallucinated hollow-3D R-CNN for 3D object detection,” IEEE Trans. Circuits Syst. Video Technol., vol. 31, no. 12, pp. 4722–4734, Dec. 2021.
[28] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep learning on point sets for 3D classification and segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 652–660.
[29] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++: Deep hierarchical feature learning on point sets in a metric space,” in Proc. NIPS, vol. 30, 2017, pp. 5105–5114.
[30] B. Graham, M. Engelke, and L. van der Maaten, “3D semantic segmentation with submanifold sparse convolutional networks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 9224–9232.
[31] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, “STD: Sparse-to-dense 3D object detector for point cloud,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 1951–1960.
[32] X. Zhang et al., “RI-fusion: 3D object detection using enhanced point features with range-image fusion for autonomous driving,” IEEE Trans. Intell. Transp. Syst., vol. 23, no. 6, pp. 2147–2156, Jul. 2017, pp. 2040–2049.
[33] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teuliere, and T. Chateau, “Deep MANTA: A coarse-to-fine many-task network for joint 2D and 3D vehicle analysis from monocular image,” in Proc. IEEE Conf. Comput. Vis. Patt. Recognit. (CVPR), Jul. 2017, pp. 2040–2049.
[34] A. Kundu, Y. Li, and J. M. Rehg, “3-DRCNN: Instance-level 3D object reconstruction via render-and-compare,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 2446–2454.
Zhenbo Song (Member, IEEE) received the Ph.D. degree in control science and engineering from Nanjing University of Science and Technology, Nanjing, China, in 2022.

From 2018 to 2019, he was a Visiting Scholar with Australian National University, Canberra, Australia. He is currently a Post-Doctoral Researcher with Nanjing University of Science and Technology. His current research interests include autonomous driving, image processing, 3-D reconstruction, and deep learning.

Zhicheng Wang received the B.S. degree from Northwestern Polytechnical University, Xi’an, China, in 2012, and the M.S. degree in computer science from Tsinghua University, Beijing, China, in 2017.

His research interests include computer vision, especially in detection and human pose estimation.

Mr. Wang was the winner of the COCO Keypoint Detection Challenge in 2017, 2018, and 2019.

Kejian Wu received the Ph.D. degree in electrical and computer engineering from the University of Minnesota, Twin Cities, MN, USA, in 2020.

He currently holds the position of the Chief Scientist and the Head of Augmented Reality Algorithms at Nreal, Beijing, China, focusing on building the algorithmic foundations of future consumer AR/MR glasses. Prior to that, he worked at Google as a Visual-Inertial System Researcher for Daydream from October 2017 to December 2018. Before joining Google, he was a Researcher at the University of Minnesota. His research focused on visual-inertial localization and mapping from September 2013 to December 2016.

Jianfeng Lu (Member, IEEE) received the M.S. and Ph.D. degrees in pattern recognition and intelligent systems from the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China, in 1994 and 2000, respectively.

He is currently a Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology. His research interests include image analysis, machine learning, data mining, and intelligent robots.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.