LIDAR in 3D Object Detection for Autonomous Vehicles

Hong Long Nguyen (Kit)
Harvard University – ES 159
honglongnguyen@college.harvard.edu

Tom McCarthy
Harvard University – ES 159
thomasmccarthy@college.harvard.edu

***Abstract*—Because of its unique ability to deliver a highly accurate 3D representation of the physical environment, LIDAR has quickly become one of the most important sensors on an autonomous vehicle (AV). In this paper, we first explore why and how is LIDAR deployed across the AV stack. Then, we take a deep dive into LIDAR's leading role in the pursuit of solving 3D object detection — one of the biggest open challenges standing in the way of full autonomy. On this dive, we review the progress and the state-of-the-art in object detection frameworks through case studies on four distinct methods of tackling the problem (all using LIDAR data, all published between 2017 and 2019). Spanning the range of approaches to the problem, these methods provide a good sense of the current state of 3D object detection and where it is likely to go next.**

I. Introduction

The autonomy sector of the automotive industry is famously split on the subject of light detection and ranging a.k.a. LIDAR. The overwhelming majority in industry and academia believes that using LIDAR is necessary full self-driving. Tesla and its vocal proponents argue otherwise, with Tesla's CEO Elon Musk notably declaring that LIDAR on autonomous vehicles (AVs) is a "fool's errand" [1].

In principle, LIDAR works exactly like its older cousin RADAR (radio detection and ranging), only instead of emitting and waiting for reflections of radio waves, LIDAR operates with 905 or 1550 nm infrared lasers [2]. Operating with narrow-beamed lasers offers LIDAR one huge asset and one pesky drawback. The benefit we get from the laser LIDAR — its centimeter accuracy in 3D space — is the prime motivator for using this sensor in AVs. The disadvantage of the laser LIDAR — its vertical resolution being linearly proportional to the number of lasers in the sensor — happens to be the cause of one of its leading deterrents: the incredibly high cost (in the $10,000s) which rises with sensor complexity, i.e. the number of lasers[†].

Having seen the number of trailing zeros in that cost figure, one could perhaps see why Tesla is so against using these sensors. Nevertheless, LIDAR's unmatched accuracy is the reason why the industry's status quo is not only to include LIDAR for perception, but also design their localization system based around LIDAR [II]. LIDAR must be included in near-future AVs for one ultimate reason: the stakes are too high for any margin of error. In the reasonable scenario of a small child near the road wearing black at night, the null hypothesis will always be that a sensing system based on vision and radar alone will not be able to reliably detect the human. Unless Tesla or other LIDAR skeptics can convincingly reject this very strong null hypothesis (which is currently nowhere in sight given the state of autonomy), it is safe to say that LIDAR is here to stay.

[†]Most frameworks introduced in this paper uses KITTI data from the 64-laser Velodyne HDL-64E LIDAR costing $75,000. [3]

II. LIDAR's Use Cases in Sensing

The goal of fully autonomous driving presents a number of difficult technical problems. These include mapping, localization, perception, prediction, planning, and controls [4]. The accuracy of LIDAR data proves vital to the sensory tasks of an autonomous driving system, which include mapping, localization, and perception. Each sweep of a LIDAR sensor creates a *3D sparse point cloud*: a precise representation of the sensor's physical surroundings in 3D. As we continue to touch on below, while data from stereo camera arrays can be manipulated to produce similar 3D scene representations, the reliable accuracy of LIDAR cannot be matched.

A. Mapping

High definition (HD) 3D maps fill a critical role in most autonomous vehicle projects. These maps typically consist of 3D data overlain with a semantic feature map (including information like traffic lights, lane divisions, signs, etc.), providing a virtual representation of a vehicle's surroundings. Vehicles use such maps to effectively determine their surroundings in the physical world and navigate through them, using the semantic map information to obey traffic rules [5]. The maps require centimeter-level accuracy because any substantial error here would get amplified further down the vehicle control pipeline.

That's where LIDAR comes in: by combining the LIDAR data from multiple passes of an area, a well-defined 3D point cloud map is created. In addition to differentiating stationary and mobile objects, creating a map from multiple angles and approaches through each area provides a solution to the problem of occlusion. Along with LIDAR data, HD maps also take in RGB camera input to log key additional characteristics (namely color and patterns) semantic objects and landmarks (for later use in localization) [6]. It is possible to generate an HD map using stereo cameras alone but are less accurate than HD maps from LIDAR as well.

B. Localization

In order to effectively utilize a 3D HD map, vehicles must pinpoint their physical position with respect to the map. To do so, virtually the entire sensor suite of GPS, IMU,

odometer, and cameras is deployed to find the AV's location on a traditional 2D map. The remainder of the work must be done by processing LIDAR data with the goal of finding the LIDAR sensor's exact 6 degree-of-freedom pose (translation and rotation) with respect to the HD map frame. This could be done either by geometrically matching the points of static objects from the LIDAR sweep to their equivalents on the HD map using the iterative closest point (ICP) algorithm, or by similar ICP matching of the laser reflectivity off of static objects [6]. Of course, one could always use a combination of both approaches to achieve the best result.

C. Perception

The most technically demanding component to the control pipeline of an autonomous vehicle, perception is the task of using sensors to observe the state of the surrounding scene and analyzing the resulting data both meaningfully and in real-time. This is the visual system of the vehicle that detects and classifies all 3D objects in the environment around it. As the decisions of this module quite literally make the difference between life and death every second, the key problem to solve here is achieving virtually perfect accuracy and a high degree of efficiency. Achieving this has so far been unreachable for the perception task of 3D object detection.

Object detection involves analyzing sensor data to determine the discrete bounding-boxes around all nearby objects [6]. The objects are further classified (i.e. car, pedestrian, cyclist, trash bin, etc.) so that the prediction and planning modules can respond accordingly. Object detection and classification results are important in the localization module as well as the perception module, as the vehicle's relative position to permanent obstacles helps in determining the vehicle's position on the HD map.

As is the case in the models we have discussed previously, there are two main approaches to implementing object detection: using RGB-D data, and using 3D data. Though the field of RGB-D object detection has the advantage of building upon a relatively large base of previous research, it has very serious physical limitations such as perspective distortion and dependence on good lighting [6]. Without these limitations, the less mature field of 3D object detection using real-time LIDAR data thus has more promise going forward. Due to constantly improving machine learning algorithms on the topic, significant advancements have been made towards solving the challenges of perception. As mentioned earlier, this paper will be taking a deeper look at recent academic progress in 3D, LIDAR-based object detection methods.

III. Comprehending LIDAR Data

A. Data Representation

Before looking into how LIDAR is used in 3D object detection, we must first understand the data coming out of this sensor which is given to us in a 3D point cloud (think dotted surfaces). This point cloud data is then represented in three main ways: (1) 3D range-view/3D sparse point cloud, (2) bird's eye view, and (3) voxels. The 3D range-view/sparse point cloud data represents the raw LIDAR output. Sparse point clouds record only the physical points in space where the LIDAR senses obstacles. As the name implies, these point clouds are typically not very dense, as each data collecting sweep of a LIDAR sensor only detects the nearest obstacle in each direction. The most accurate perception results are achieved using 3D point cloud data, at the cost of memory and computational resources.

The bird's eye view representation offers an intuitive compression of 3D point cloud data in 2D by disregarding z-axis data, but makes a serious trade-off between resolution and cost. This representation is able to capitalize on a wealth of existing 2D object detection/classification techniques, which are naturally significantly faster than their 3D counterparts. Bird's eye view representation also combines easily with HD map information. This being said, it would not be feasible to pursue this technique as the primary basis for a perception module, as it is unable to handle occlusion in the z direction and fairs poorly when given scarce data points or small objects.

The voxel representation makes use of a partitioning of the 3D space into discrete voxels (uniform 3D boxes). A series of voxels is then used to represent the LIDAR data, with the occupancy of each voxel stored along with it. This leads to decreased computation and memory cost, with the additional benefit that data analysis can be done with simple 3D convolutional networks. Depending on voxel size, this exhibits similar drawbacks as the bird's eye view representation, and does not consider the specific properties of the raw LIDAR data [6].

B. Data Processing and PointNet

Though standard image processing convolutional pipelines can not be effectively applied to raw 3D point cloud data, revolutionary techniques developed within the past 5 years make handling such data possible without discretization. The most significant of these techniques is PointNet [7], published by Qi et al. in December 2016, which provides a straightforward method for feature learning on unordered point sets. The network architecture has three straightforward stages. First, point-wise features are extracted and transformed. Second, a MaxPooling layer to aggregate those point-wise features. This aggregate feature set is then used to augment point-wise feature sets, allowing the network to predict based on point-wise relationships in addition to global information. Used in most contemporary 3D object detection end-to-end methods, PointNet allows deep neural networks to be applied directly to 3D point cloud data, greatly improving the potential accuracy and descriptiveness of feature learning on raw LIDAR data.

IV. 3D Object Detection Using LIDAR

Unlike LIDAR HD map generation and localization, which are still deemed unnecessary by a vocal minority (namely Tesla) [1], as outlined many times earlier, the merits of the perception module is virtually undisputed. In this section, we will focus on how LIDAR point clouds are used to solve the primary task of perception: 3D object detection, i.e. given sensory inputs from the environment (e.g. a busy intersection),

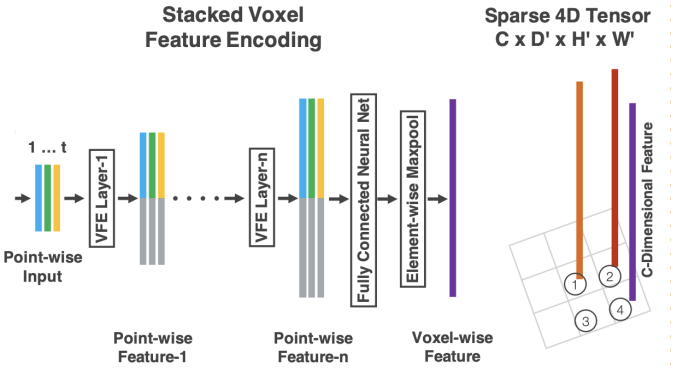

Fig. 1. VoxelNet feature learning network.

output bounding boxes around pertinent semantic objects (cars, pedestrians, traffic lights, etc.).

It is key to realize that LIDAR is by no means the only sensor deployed to solve this problem. Along with LIDAR 3D point clouds, inputs from RGB cameras, as well as potential radar and ultrasonic sensors should all be incorporated into the final detection algorithm. A natural question then arises as to how to fuse the multimodal data effectively. The two main approaches used in practice are commonly known as late fusion and early fusion. For completeness, note that a more complex approach of middle fusion also exists [8]. The late fusion strategy owes its name to the fact that it combines the final outputs of the processing pipelines of individual sensors. In the context of 3D object detection, a late fusion approach would average among bounding boxes generated by each of the LIDAR, camera, and radar modules. This is in contrast with early fusion which combines the multimodal inputs into an integrated higher dimensional format and generates a single set of bounding boxes from this fused input. Late fusion is currently more used in practice for its modularity and relative simplicity in implementation despite its high computational and memory costs. Early fusion, however, is believed to have more potential for it could capture multimodal features as well as process them with a smaller compute and memory footprint [8]. That being said, creating a method with superior efficacy of fusing and later processing unsynchronized data from across multiple measurement spaces remains an open problem; we have yet to see any early fusion detection scheme substantially beat the late fusion standards [bible]. Having established that, let's now delve into some state-of-the art-examples of point cloud object detection techniques used for late fusion (VoxelNet [IV-A], PointPillars [IV-B], and PointVoxel–RCNN [IV-C]), as well as a leading exemplar of the multimodal early fusion approach (F–PointNet [IV-D]).

A. *VoxelNet [9]*

Published in November 2017, VoxelNet is a generic 3D detection network, which creates 3D bounding boxes around objects given 3D sparse point cloud data. A key benefit of VoxelNet (in addition to the following methods we introduce) is its end-to-end nature; it removes the need for manual feature engineering of the 3D point cloud data, enhancing robustness, reliability, and often efficiency. VoxelNet divides raw point cloud data into 3D voxels, groups the point cloud data according to their respective voxels, learns features within each voxel using raw data points, and combines the point-wise features to find voxel-wise features across the sample. 3D convolution layers are further used to augment the voxel-wise feature set. This encoding of the data is fed into a region proposal network, which outputs the bounding boxes of each identified object. This approach combines the benefits of both raw point cloud data and voxel data; point-wise features are extracted as a voxel representation of the data is constructed, and the less computationally intensive voxel representation is used for further computation using a larger portion of the sample.

The most notable innovation introduced by VoxelNet is the chain of voxel feature encoding (VFE) layers, which applies a PointNet on each voxel to learn point-wise features and aggregate them into voxel-wise features [9]. Each data point within a nonempty voxel is augmented with its offset from the centroid of that voxel, creating the input feature set to the voxel's VFE layer. This is fed into a fully connected network (FCN) (comprised of a linear layer, a normalization layer, and a rectified linear unit layer) which outputs the point-wise feature representations mentioned above. The point-wise features are fed into a MaxPooling layer, outputting the aggregate features across the voxel– this is used to augment the point-wise features, giving us a point-wise concatenated feature set. Stacking VFE layers encodes the descriptive spatial information of the points within a voxel. Voxel-wise features are extracted by transforming the stacked-VFE output using another FCN and MaxPooling.

The set of voxel-wise features extracted across the entire dataset are stored as a sparse 4D tensor, where only data from non-empty voxels is stored [9]. As typically over 9/10 voxels are empty, this cuts down on the resource intensiveness of back-propagation, capitalizing on the efficiency of the feature learning algorithm. As mentioned above, the voxel-wise features are then fed into a series of 3D convolutional middle layers to aggregate them with a wider and wider field, giving them a more representational encoding of spatial information. These layers remain a significant computational bottleneck to the model's overall efficiency [10].

Region proposal networks (RPN) are a staple of object detection architectures, used to predict object bounds and objectness scores (how likely a proposed object is actually an object) simultaneously. VoxelNet presents a modification on the RPN [11] introduced by Ren et al. as the final step to network architecture, which uses the feature map outputted by the convolutional middle layers as its input.

Upon its introduction, the authors of the VoxelNet paper stated that based on experiments on the KITTI [12] car detection benchmark, VoxelNet "outperforms the state-of-the-art LIDAR based 3D detection methods by a large margin" [9]. They also state that the unique method of feature learning

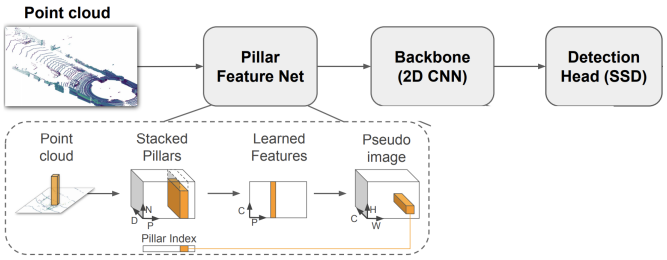

Fig. 2. PointPillars general architecture.

in VoxelNet allows superior 3D detection of pedestrians and cyclists. While VoxelNet no longer performs at the same level of success against more modern object detection methods and is too slow for real-world deployment, its defining VFE layers have been adopted in a number of them.

B. PointPillars [10]

Proposed by Lang et al. in December 2018, PointPillars aims to accomplish the same object-detection goals as VoxelNet with a higher-efficiency approach. Rather than using 3D convolutional layers to enhance representativeness of voxel-wise learned features, PointPillars introduces a method using only 2D convolutional layers. Its novel encoding scheme learns features on the vertical columns (pillars) of the raw point cloud representation, encoding the 3D point cloud into 2D bird's eye view space. This allows for the use of exclusively 2D convolutional layers, which mitigates the respective bottleneck of VoxelNet and removes the necessity of vertical partitioning of data points. Like VoxelNet, the PointPillars network consists of three stages: a feature encoder, convolutional middle layers, and a detection head that outputs 3D object bounding boxes.

In the first stage, the 3D sparse point cloud data is divided evenly into a grid on the x-y plane. This creates a set of pillars, one per grid unit, with each pillar spanning the entire z-range of the point cloud. Like VoxelNet does within voxels, each datapoint within a pillar is augmented with its offset from the centroid of the pillar's datapoints. This data is stored in a dense 2D tensor of fixed size per pillar; if too many data points are within a given pillar, they are randomly sampled to fit the tensor. Features are encoded using a simplified version of PointNet, to which MaxPooling is applied. This results in a 1D aggregate feature set per each pillar, leaving us with a 3D (features per grid unit on the x-y plane) tensor encoding of the original point cloud data. At this point, we essentially are looking at features of each point in a bird's eye view representation of the data.

Secondly, the pillar-wise learned features are fed into 2D convolutional layers, where the efficiency benefits of this approach become apparent (based on the convolution dimensionality reduction compared to the corresponding step in VoxelNet). Concatenation of this output produces a final pillar-wise feature set based on successively wider samples of neighboring pillars.

Lang et al. chose the Single Shot Detector (SSD) [13] object detection scheme to find bounding boxes using this feature set as input. This implements another method to achieve the goal of RPNs; producing bounding boxes for objects and scoring their objectness. SSD (published in 2015) claims to be faster than proposal based object detection schemes, but for our purposes, performance and innovation in this stage is of secondary importance to that of the first two stages.

When released, PointPillars outperformed previous state of the art models in both speed and accuracy by a large margin– the new feature learning scheme improved speed over other methods by a factor of 2-4 on the KITTI benchmark. Looking at the updated KITTI benchmark rankings, PointPillars has since been surpassed by many other methods in terms of accuracy, but its speed remains unmatched. This impressive speed-accuracy trade-off exhibits promising potential in the perception module of autonomous driving, as fast object detection is paramount.

C. PointVoxel–RCNN (PV–RCNN) [14]

Published by Shi et al. on the last day of 2019, the PointVoxel–RCNN[†] (PV–RCNN) model is the currently reigning LIDAR 3D object detection framework [12]. Hinted at by its name, PointVoxel–RCNN integrates the efficiency of voxel feature encoding (VFE) and region proposal (see [IV-A]) and the accuracy PointNet-based set abstraction and learning (PointNet++ [15]) to output reliable bounding boxes. To achieve its highly-optimized performance, PV–RCNN's structure gets quite complex, but generally features four main stages: 1) *voxel CNN feature encoding and proposal generation*, 2) *voxel-to-keypoint scene encoding*, 3) *keypoint-to-grid RoI feature abstraction*, and finally 4) *3D proposal refinement and confidence prediction*.

The *voxel CNN feature encoding and proposal generation* stage is powered entirely by techniques borrowed from VoxelNet [IV-A] and PointPillars [IV-B]. First, the LIDAR point cloud is transformed into voxel-wise feature vectors by a 3D voxel convolutional neural network (CNN). Then, these 3D feature volumes are reduced to a bird's eye view representation and fed into an anchor-based region proposal scheme similar to the final stage of PointPillars. The resulting bounding regions of interests (RoI's) from this stage alone "achieves higher recall performance than [solely] PointNet-based approaches," [14] but it lacks in resolution due to voxelization downsampling. The next two stages of PV–RCNN are there to remedy that.

The *voxel-to-keypoint scene encoding* stage begins with reducing the sparse voxelized scene into a collection of roughly uniformly distributed point cloud *keypoints*. These 2,048 to 4,096 *keypoints* are sampled using the Furthest-Point-Sampling algorithm. These *keypoints* are then processed in the *voxel set abstraction* module which uses PointNet to encode each *keypoint* with its and its neighbor's multiscale voxel-wise features from stage 1). The features-carrying *keypoints* are then

[†]RCNN = Region-based Convolutional Neural Network

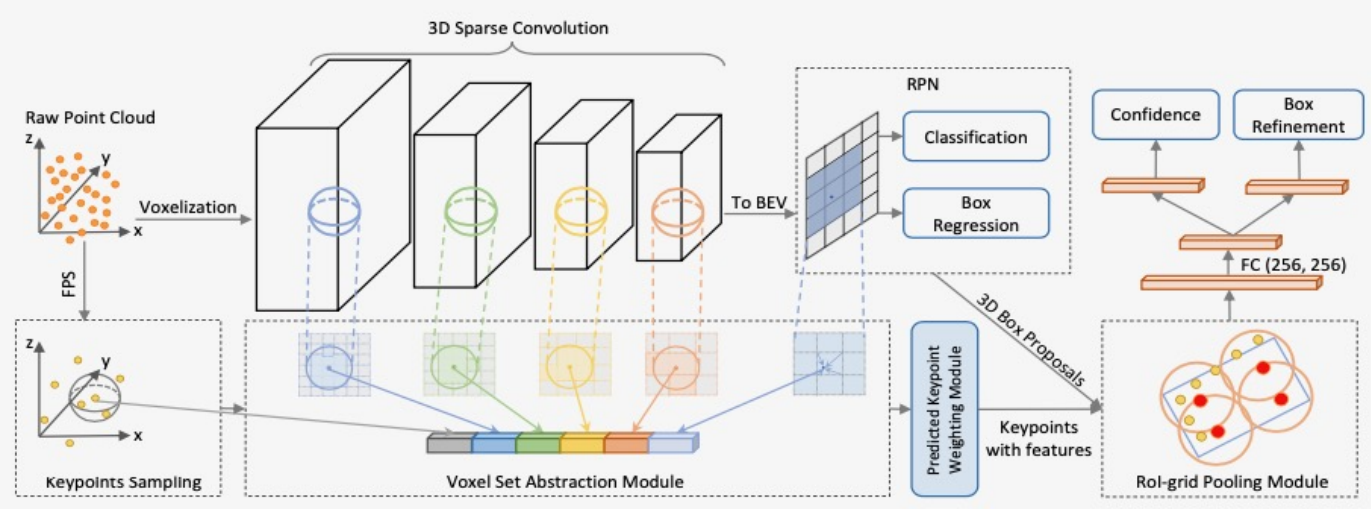

Fig. 3. PV–RCNN architecture.

weighted based on importance (object vs. background) in a sigmoid-activated three-layer perceptron network dubbed the *predicted keypoint module*. These weighted, features-carrying *keypoints* and the RoI from stage 1) are then fed to the next stage.

The *keypoint-to-grid RoI feature abstraction* stage takes all the *keypoints* in and neighboring the proposed RoI's and elaborately aggregates their features. Eventually, the stage employs a two-layer perceptron to transform the pooled features into 256D linear vectors that fully represents points' features of each proposed bounding box.

Given the proposed bounding box from stage 1) and its aggregated feature set from stage 4), *3D proposal refinement and confidence prediction* is just another two-layer perceptron — this time, however, with two branches: one for box refinement and the other for confidence approximation. The entire four-staged PV–RCNN network is trained end-to-end using the equally weighted sum of the local region proposal, keypoint segmentation, and proposal refinement losses.

Unlike all the other frameworks presented whose claims of preeminence have gone out of date, PV–RCNN's claim to fame is that as of the writing of this paper in May 2020, it is reigning the KITTI 3D object detection benchmark in car detection and in the top 15 for the pedestrian and cyclist categories. By only manipulating sparse voxels and then sampled *keypoints*, PV–RCNN is also on the computationally lighter end of 3D detection approaches, achieving its KITTI benchmark results without the need of a GPU [12].

D. Frustum–PointNet (F–PointNet) [16]

Combining the tried and true 2D object detection using R-CNNs and 3D point cloud segmentation using PointNets, in late 2018, Qi et al. introduced one of the leading early fusion 3D object detection schemes to date dubbed the Frustum–PointNet (F–PointNet). The F–PointNet pipeline features three stages in serial: the *frustum proposal* stage, the *3D instance segmentation* stage, and the *amodal 3D box estimation* stage.

The *frustum proposal* stage encodes input images from RGB cameras into using a Feature Pyramid Network [17] to then feed the data into a Fast-RCNN [18] 2D object detection model. Combined with rudimentary depth information (from stereo vision or radar), the detected amodal 2D box is then projected into the 3D space (where our LIDAR point cloud exists) to parametrize a frustum of interest. The output this early fusion *frustum proposal* stage is then a *frustum point cloud* - all LIDAR points that exist within the frustum region - with a one-hot encoded vector for the object class (car, pedestrian, etc.).

Next comes *3D instance segmentation* where the *frustum point cloud* and its category is fed into a 3D instance segmentation PointNet which segregates the frustum points belonging to the object and the ones in the the foreground/background. These segmented object points finally make it to the *amodal 3D box estimation* stage where our output bounding boxes get generated.

Before that last stage, a word on the *amodal* quality. An *amodal* bounding box is one that infers and includes occluded parts of the object — a capability the authors of F–PointNet are very keen on. To generate the *amodal* 3D bounding boxes, the segmented object points from the previous stage are first fed into a lightweight regression PointNet (T-Net) [**?**] to predict the *amodal* object center which is then combined with the segmented object points to enter the final amodal 3D box estimation PointNet model that spits out our resulting bound box results. The three neural nets involved (*3D instance segmentation* PointNet, T-Net, and *amodal 3D box estimation* PointNet) are trained in parallel using a custom multitask loss.

Back when it was introduced in November 2017, the authors claimed that F–PointNet "achieved significantly better results compared with the state-of-the-art" [16]. Now, roughly three years later, the KITTI 3D Object Detection Evaluation leaderboard [12], suggests that though F–PointNet's performance is still very solid, it would only rank middle-of-the-pack

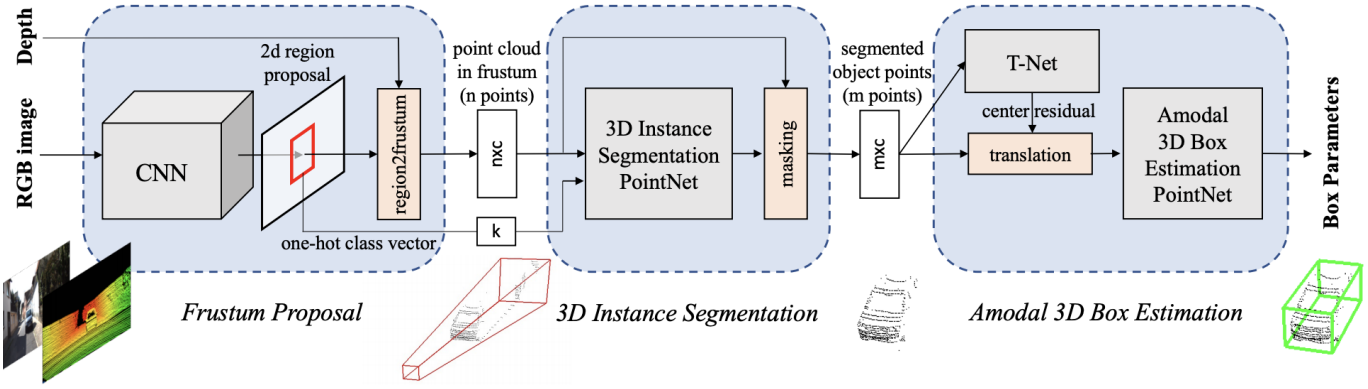

Fig. 4. F–PointNet architecture.

when stacked up against the cutting-edge approaches. Lastly, it is worth pointing out that running F–PointNet requires GPU acceleration, putting it among the computationally more demanding approaches out there [12].

V. Conclusion

The field of 3D object detection using LIDAR data is progressing quickly; the state-of-the-art method today could be built upon and surpassed a month from now. Just in the past three years, dozens of techniques have been introduced to solve these problems. The encouraging performance of these algorithms in simulation, however, does not necessarily represent their performance in the real world. Real autonomous driving is not a single test set of images requiring analysis, and mistakes don't just cause lost benchmark points; for an object detection scheme to be pushed to production, it needs to be reliable and robust enough to respond in real-time to any case thrown at it. Though these algorithms are not currently in a position where they are ready to be deployed on the streets, they provide incredibly promising jumping off points for future work. Down the line, for example, we can expect to see an increase in early-fusion model development, building off of the early success of methods like F-PointNet. We can also expect to see an increase in hybrid methods like PV-RNN, making use of the groundwork lain in the infancy of the field. Regardless, we can rely on the fact that LIDAR will remain central to autonomous driving pursuits, and that the only way we will be able to achieve a future where autonomous vehicles replace human drivers is if our analysis of LIDAR data is perfected.

Acknowledgment

This work was supported by our professor Rob Wood and teaching fellows Jay Li and Irina Tolkova. We also thank our fellow classmates in our Harvard ES 159 class.

References

[1] Tesla, Tesla Autonomy Day is on Mon, April 22nd, Streamed live on YouTube on Apr 22, 2019.

[2] Velodyne Lidar, "Guide to LiDAR Wavelengths," 2018, velodynelidar.com/blog/guide-to-lidar-wavelengths.

[3] Paden Tomasello*, Sammy Sidhu, Anting Shen, Matthew W. Moskewicz, Nobie Redmon, Gayatri Joshi, Romi Phadte, Paras Jain and Forrest Iandola, "DSCnet: Replicating Lidar Point Clouds with Deep Sensor Cloning," 2018, arXiv:1811.07070v2.

[4] C. Urmson, J. Anhalt, D. Bagnell, C. R. Baker, R. Bittner, M. N. Clark, J. M. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. M. Peterson, B. Pilnick, R. Rajkumar, P. E. Rybski, B. Salesky, Y-W. Seo, S. Singh, J. M. Snider, A. Stentz, W. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson, "Autonomous driving in urban environments: Boss and the urban challenge," in The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, George Air Force Base, Victorville, California, USA, 2009, pp. 1–59.

[5] M. Aldibaja, N. Suganuma and K. Yoneda. "LIDAR-data accumulation strategy to generate high definition maps for autonomous vehicles," 2017 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), Daegu, 2017, pp. 422-428, doi: 10.1109/MFI.2017.8170357.

[6] Siheng Chen, Baoan Liu, Chen Feng, Carlos Vallespi-Gonzalez, and Carl Wellington, "3D Point Cloud Processing and Learning for Autonomous Driving," IEEE Signal Processing Magazine, Special Issue on Autonomous Driving, 2020, arXiv:2003.00601.

[7] Charles R. Qi, Hao Su, Kaichun Mo and Leonidas J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR 2017, arXiv:1612.00593.

[8] Di Feng, Christian Haase-Schutz, Lars Rosenbaum, Heinz Hertlein, Claudius Glaser, Fabian Timm, Werner Wiesbeck4 and Klaus Dietmayer, "Deep Multi-modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges," 2020, arXiv:1902.07830v4.

[9] Yin Zhou and Oncel Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," CPVR 2017, arXiv:1711.06396.

[10] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang and Oscar Beijbom, "PointPillars: Fast Encoders for Object Detection from Point Clouds," CVPR 2019, arXiv:1812.05784.

[11] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," 2016, arXiv:1506.01497v3.

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," Conference

on Computer Vision and Pattern Recognition (CVPR), 2012 (Rankings Updated 2020).

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg, "SSD: Single Shot MultiBox Detector," ECCV 2016, 10.1007/978-3-319-46448-0_2, arXiv:1512.02325.

[14] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li, "PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection," 2019, arXiv:1912.13192v1.

[15] C. Ruizhongtai 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 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, 2017, pp. 5099–5108.

[16] C. Ruizhongtai Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from RGB-D data," in Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn., 2018, pp. 918–927.

[17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie, "Feature Pyramid Networks for Object Detection," 2016, arXiv:1612.03144.

[18] Ross Girshick, "Fast R-CNN," ICCV 2015, arXiv:1504.08083.