3D Vehicle Detection on an FPGA from LiDAR Point Clouds

Javier García López
FICOSA ADAS S.L.U
08232, Barcelona, Spain
jgarcia@iri.upc.edu

Antonio Agudo
Institut de Robòtica i Informàtica Industrial CSIC-UPC 08028, Barcelona, Spain
aagudo@iri.upc.edu

Francesc Moreno-Noguer
Institut de Robòtica i Informàtica Industrial CSIC-UPC 08028, Barcelona, Spain
fmoreno@iri.upc.edu

ABSTRACT

In this paper is presented a deep neural network architecture designed to run on a field-programmable gate array (FPGA) for detection vehicle on LIDAR point clouds. This works present a network based on VoxelNet adapted to run on an FPGA and to locate vehicles on point clouds from a 32 and a 64 channel optical sensor. For training the presented network the Kitti and Nuscenes dataset have been used. This work aims to motivate the usage of dedicated FPGA targets for training and validating neural network due to their accelerated computational capability compared to the well known GPUs. This platform also has some constraints that need to be assessed and taken care during development (limited memory e.g.). This research presents an implementation to overcome such limitations and obtain as good results as if a GPU would be used.

This paper makes use of a state-of-the-art dataset such us Nuscenes which is formed by several sensors and provides seven time more annotations than the KITTI dataset of the 6 cameras, 5 radars and 1 Lidar it is formed by, all with full 360 degree field of view. The presented work proves real-time performance and good detection accuracy when moving part of the CNN presented in the proposed architecture to a commercial FPGA.

CCS Concepts

• **Computing methodologies → 3D imaging**

Keywords

Deep learning; Vehicle detection; FPGA; Convolutional Neural Network

1. INTRODUCTION

The emergence of field-programmable gate arrays (FPGAs) in image processing and deep learning is increasing nowadays due to the benefits of such hardware for conducting faster mathematical computations and processing operations, but mostly because of the appearance of new frameworks that allowed developers to port their work to an FPGA in a more straightforward way. Several studies like [4] proof the advantages of considering FPGAs as an option for image processing and deep learning applications.

The goal of this research is to present a comparative of the performance of a 3D vehicle detection based on LIDAR [5] point clouds and a known network architecture such as VoxelNet when running such network on a GPU and on an FPGA. Furthermore, a state-of-the-art dataset like Nuscenes [3] has been used in the training and validation of the proposed method. The needed network adaptions to run on an FPGA platform and with the mentioned dataset are described within this paper.

In order to overcome the challenge of extracting 3D information from 2D data, several researchers have presented works and methodologies that have proven remarkable results in human pose estimation or face expression recognition, e.g. ([6] or [7]).

In the field of vehicle pose estimation, other image-based approaches suggested a system with two cameras separated a known distance for feature matching, 2D detection followed by a 2D-3D matching phase for calculating the 3D position of the vehicles, such as in [8], or even a system in which the ground plane equation is known in advance together with a 2D vehicle detection network ([9] e.g.) to predict as last step the 3D bounding box around the detected vehicles in the image.

To avoid these mentioned constrains, the usage of point clouds from optical sensor such us LIDARs is a good option because these provide already the required 3D information. However the accuracy of these sensors and the difficulty to manage 3D point clouds compared to 2D images have led to these sensors not being widely used in 3D pose estimation problems.

LIDAR sensors in autonomous driving work normally by reflecting light beams from a light source in an internal mirror that outputs the beam outside the sensor toward the object to localize. These sensors rotate around themselves so that they provide depth information of a 360^{o} surrounding area. The rotation frequency together with the number of light beams emitted each cycle are key to obtain accurate environment information to be used for training of a neural network. In addition, LIDAR is not subjected to environmental illumination. Normally these sensor were not used in commercial applications for autonomous driving due to their high price and difficulty for data synchronization. However, lately precise 32 or 64 channel LIDAR sensors have come out in the market with reasonable price and size that make it more reasonable to be integrated in a commercial vehicle. As stated in works such as [10], although graphics processing units (GPUs) are more suitable for parallel processing they do need a high power consumption, which could make them a bottle-neck for the integration of deep learning algorithms into vehicles, as they have limited power supply. In this scenario, FPGA are a low-power consumption option more suitable for embedded applications as they can be programmed as a customized integrated circuit that is able to perform massive parallel processing and data communications on-chip. We believe this is enough motivation for pursuing a breakthrough in the field of deep learning applications on FPGA, since the usage of this platform in commercial vehicles is widely extended already (for image data-stream conversion e.g.) and the appearance of frameworks for porting networks to run on FPGA platforms is boosting up their usage in autonomous vehicles against GPUs.

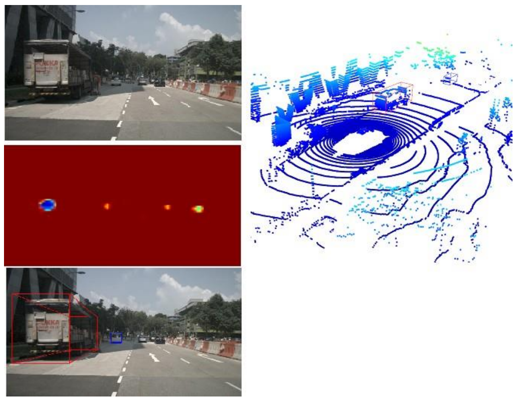

Figure 1: Example of the performance of the presented pipeline in this work for vehicle detection. Planar images are only used for visualization but not for testing.

2. RELATED WORK

2.1 Image based 3D Object Detection

Over the last years, many image-based approaches for 3D object detection have been presented showing several ways of predicting 3D information from 2D images such us [11], [12], [13], [6] or [14].

Studies like [15] have showed very good results when calculating the 3D human pose from joint localization. This is achieved by passing the input image through several hourglass phases (e.g) to generate heatmaps to capture features at various scales. The motivation of doing so is the need of spatial information for calculating the pose. An understanding of the whole body is crutial to prediction the body pose.

The architecture of hourglass architecture is formed by a Convolutional and max pooling layers used to process features down to a very low resolution. After reaching the lowest resolution, the network begins the upsampling to the original resolution and combination of features across scales. Hourglass networks are symmetric, so for every layer present on the way down there is a corresponding layer going up. After reaching the output resolution of the network, two consecutive rounds of 1x1 convolutions are applied to produce the final network predictions. The output of the network will be the mentioned heatmaps where the human joint will be for each one predicted with pixel accuracy [15].

These approaches tend to use the texture information provided by the input images to predict the 3D bounding boxes from 2D images. However, the accuracy of image-based 3D detection approaches are bounded by the accuracy of the depth estimation. One of the reasons that motivate the usage of LIDAR point clouds when solving the issue of the 3D bounding box calculation is this one, since these optical sensors already provide depth measurement and no error is included in the detection pipeline when predicting the depth.

2.2 LIDAR Sensors

There are several LIDAR point clouds disposition or pre-processing approaches for machine learning and deep learning applications used over the years. Studies like [16], [17] or [18] the LIDAR points were projected onto the image for later feature extraction.

Other approaches like [19] created dense depth map from the LIDAR point cloud to afterwards use this as input for machine learning techniques and predict 3D shapes.

On the other hand, more recent works such as [20] proposes a point cloud processing for transforming them to view and top-view image and combine these with the input image. This research uses the sparse point clouds directly from the LIDAR sensor following a similar approach as the one proposed in [1] without treating this input data and avoiding the usage of planar images as training data. The VoxelNet approach then compensates the high disparity and variance of the input data by following these steps:

- - Voxel creation: 3D gridding is calculated through the input scene to divide it in different voxels of a variable size depending on the object to be located. The points belonging to each voxel will then have been grouped after this first step.
- - Random sampling: To avoid the different number of point that could be contained in the different voxels, a random sampling of points inside each voxels with a number of points bigger than a predefined threshold is conducted.
- - Stacked Voxel Feature Encoding: One key of the work of [1] is precisely this encoding step, in which the points inside a voxel are converted into concatenated feature with surface information and geometrical information.
- - Sparse Tensor Representation: Once the voxel features together with the voxel spatial information is obtained, a tensor with this information is created. This representation reduces the memory usage and computation cost during backpropagation.
- - Convolutional middle layers: The convolutional middle layers add more context to the shape description by passing the tensors through convolution, batch normalization and ReLu to add more context to the shape description inside the tensor.
- Region Proposal Network: A probability score map and a regression map are finally calculated by passing the feature maps from the previous CNN to three FC Layers for downsampling-upsampling for obtaining the high resolution feature map.

As shown in Figure 3, a set of 3 convolutional layers, batch normalization (BN) and ReLu follow the data pre-processing of the 3D point clouds. For simplicity and analog to VoxelNet implementation [1], after the last FCN layer before the CNN layers, a Sparse Tensor Representation by processing only the non-empty voxels has been followed in this work. As explained in [21], the obtained tensor representation after passing through several VFE layers leads to tensor containing descriptive information about the shape. The next convolutional layers provide context information to the shape detection already obtained from each Voxel or grid.

One of the breakthroughs of this research is the implementation of the set of CNNs, BN and ReLu layers running on the FPGA. For that purpose, the leg-up 4.0 [22] framework has been used together with ModelSim HLS suite to convert the implemented layers into readable code by the FPGA.

Making use of the mentioned leg-up [22] framework (version 4.0) and the Modelsim HLS design Suite for Intel Arria 10 FPGA the porting from the tensorflow source code of the convolutional, BN and ReLu layers to translated code readable by the FPGA was performed. Nevertheless, for making the best use of the HW resources of the platform and due to memory limitations, the hyper-parameters of the network running on the FPGA were optimized with the q-factor explained in Section 4.2.

3. CHOSEN DATASET

As mentioned before, the datasets chosen for this work are the known Kitti and Nuscenes [3], which was released in its last version in March 2019 and contains more than 7000 samples of images and point clouds fully annotated. The reason of choosing this last state-of-the-art dataset is mainly because the high quality of its labelling and big availability of synchronized sensors. This dataset offers full autonomous vehicle sensor suite composed by 6 cameras, 5 radars and 1 LIDAR. 23 classes and 8 attributes are labelled in each of the 1000 scenes of 20s long each.

However, Nuscenes is based on a 32-channel Lidar when Kitti uses a 64-channel one. This makes that point clouds in the case of the Kitti dataset are more dense and therefore the Voxelnet configuration varies in one case and the other. Another motivation for choosing this dataset for this work is the synchronization assurance between data from different sensors provided by Nuscenes. The data synchronization between sensors it crucial for any image processing methodology that takes samples from different sensors. Being able to match LIDAR point clouds with camera frames taken both at the exact same time, so that a direct matching between LIDAR measurement and object on the image is possible, is highly relevant. In the case of this work, since LIDAR data points were taken for the detection and correspondent images for the visualization this synchronization between data was also an important point. The chosen dataset assures the synchronization of the data of recording time by triggering exposure of a camera when the top LiDAR sweeps across the center of the camera's FOV, as explained in [3].

The fact that both mentioned datasets have different accuracy lead to the need of adapting the VoxelNet architecture to be compatible with 32 channel Lidar sensor as described in 4.1.

4. PROPOSED METHOD

In this section, the implementation of the presented pipeline in this work is described. As previously commented, the network architecture used in this paper is based on the promising VoxelNet due to its good results in object localization with point clouds. However, the CNNs that take place in this architecture were modified and adapted to the application presented since they are running on a FPGA Hardware. This platform has some promising improvements in machine learning and deep learning like being able to speed up heavy computations, however it has some implications that need to be considered during implementation phase.

One of the biggest differences when programming an application that runs on a GPU or on a FPGA is the fact that available memory to handle the multiple meta-parameters and weights during CNNs training is more constrained in a FPGA than in a GPU or CPU. Therefore, one of the breakthroughs of this work is the implementation of a simulated quantization step needed to run the training and validation on an FPGA based on a similar approach as the one presented in [10]. As explained there, when using CPU or GPU floating-point operation are used, which create gradients during training.

This approach for converting floating point data to fixed point shall be designed carefully since this conversion could lead to a considerable accuracy loss, due to the rounding of big variables with several decimals to less bit consuming integer variables. This step of the presented pipeline is defined in 4.2.

4.1 Network Configuration

In this approach, the Nuscenes dataset has been used due to the big amount of training samples available and because it is a state-of-the-art dataset. The election of mentioned Nuscenes dataset and after several tests, following voxel grid sizes has been elected, depending on the object to be localized (For training the network with Kitti dataset, the configuration of the network was the one proposed in [1]):

- -Vehicle detection: The point cloud range considered is [-4,2]x[-40,40]x[0,80] meters along Z, Y and X axis respectively. Therefore, the voxel size will be $\upsilon_D = 0.2$, $\upsilon_H = 0.2$ and $\upsilon_W = 0.2$ meters

which leads to $D' = 30$, $H' = 400$ and $W' = 400$. For this selection, we took into consideration the point cloud density and distribution of the selected dataset and we followed the steps proposed by [1]. As maximum number of points inside a voxel T, we chose 50. A total of 3 middle convolutional layers was selected.

- - Pedestrian detection: The point cloud range considered in this case is [-4,2]x[-20,20]x[0,50] meters along Z, Y and X axis respectively. The voxel size will be also $vD = 0.2$, $vH = 0.2$ and $vW = 0.2$ meters and therefore $D' = 30$, $H' = 200$ and $W' = 1000$. Since the detection of these classes will require a bigger number of LiDAR points in each voxel to have a better perception of the shape, the maximum number of LIDAR points on each voxel in this case was set to 50.

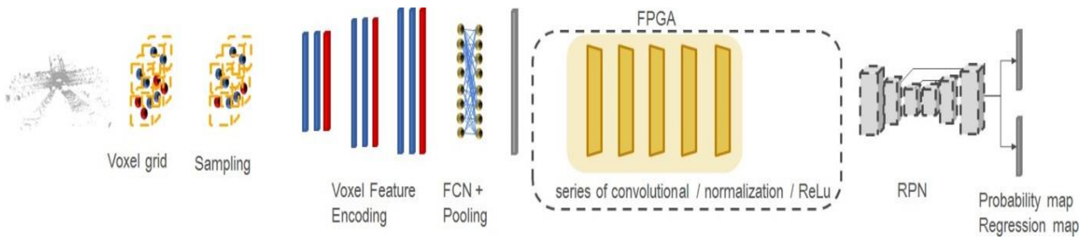

Figure 2: Implemented pipeline for 3D object detection inference and training.

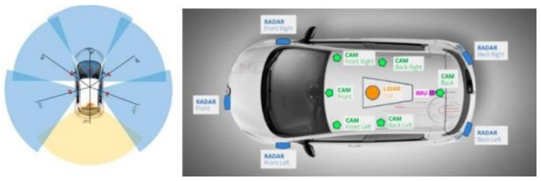

Figure 3: Sensor disposition of the used Nuscenes dataset [3]

4.2 Data Quantization: Preparing the Data for FPGA

As commented in 1, FPGAs have some constraints that shall be addressed during the implementation of the network. Taking these design constraints into consideration, one of the most important processing steps that was included in this work was the quantization of the data to port it from floating point to fixed-point so that it can be processed by the FPGA. Fixed-point variables, weights and operations are normally used in some platforms because they have no native libraries for floating-point usage and because these normally have less memory resources such as FPGA.

Floating point operations require big amount of memory since every value requires normally between 32 and 64 bits each. That is sometimes not an option on platform with no GPU and that is the main reason why these value are normally re-scaled to be stored in smaller data types and also for that reason is this scaling and the chosen precision very critical. Errors due to a bad porting from floating point to fixed point can be very critical and came make a re-projection step to project a point out of an image or an algorithm not converge (e.g). GPU platforms generally use floating-point operations that can generate continuous gradients in the training.

Table 1: Table representing the F1 score (F1) and average precision (AP) for different configuration of q-factors for the data quantization step, Section 4.2

Name	F1 (%)	AP(%)
no quantization	94.05	88.29
quantization with 12 bits	88.25	79.24
quantization with 16 bits	90.59	84.24
quantization with 18 bits	90.81	86.01
quantization with 24 bits	94.07	92.03
quantization with 32 bits	94.66	88.5

To solve the commented problem on FPGA platform this works proposes an implementation for a porting to fixed point from weights and gradients with the following approach.

First the training and validation phase has to be set on the GPU platform. After this first step, a short software to go through all available variables and weights and analyze their data type and possible values during the execution was developed. Like this, we can predict the variables that will overflow if the fixed-point conversion is done and they have to be re-scaled. An adaptive calculation of some factors determining how many bits will be dedicated to integer part and how many for the fractional part has been developed for this purpose. For the candidate variables to suffer from overflow at some point of the training that we the outcome of the mentioned first analysis these factor will change at the moment that a bit overflow is predicted.

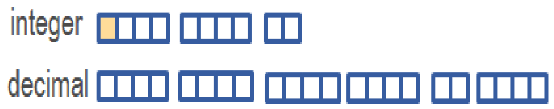

Figure 4: Example of implementation of a parame -ter with 22 bits for the decimal part, 9 bits for the integer a 1 bit for the sign (marked in orange).

If a weight or variable is defined as a factor 10.21 e.g., meaning 10 bits for integer and 21 bits for fractional part, this bit arrangement can vary if during training one possible overflow is detected. This detection is done via some little memory reserved for internal diagnosis (some number of bits being utilized on execution time for integer and fractional part to be used by the localized sensible variables to suffer overflow). This for sure affected the available resources during training but enabled us the possibility of dynamically change these factors.

For the utilized hardware the maximum number of weights and variables to be observed and analyzed during training for this purpose was 4000.

To evaluate the proposed method for factorizing and optimizing the weights and variables, we performed the training of the network with the KITTI [2] dataset for several bit amounts, as shown in Table 3:

Table 2: Table representing the F1 score (F1), average precision (AP) and complete runtime execution of the complete pipeline when doing inference on the test set of 2000 samples.

Name	F1 (%)	AP(%)	Runtime(ms)
Chipnet [10]	94.05	88.29	17.59
Fused CRF[26]	88.25	79.24	2000
Mixed CRF[27]	90.59	84.24	6000
Hybrid CRF [28]	90.81	86.01	1500
LoDNN [29]	94.07	92.03	18
Ours (Kitti)	94.66	88.5	18
Ours (Nuscenes)	87.25	79.54	18

Table 3: Table comparing the performance in 3D detection of the proposed method for 3 levels of occlusion, hard (until 60 % of the object is visible), moderate (80% visible) and easy (fully visible).These results proof that with the proposed method, similar results are obtained when using the Kitti dataset running on a FPGA as in the VoxelNet execution on GPU.

	Car			Pedestrian		
Method	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
VeloFCN [5]	15.20	13.66	15.98	N/A	N/A	N/A
MV (BV+FV) [23]	71.19	56.60	55.30	N/A	N/A	N/A
VoxelNet [1]	81.97	65.46	62.85	57.86	53.42	48.87
Ours (Kitti)	81.82	65.11	61.96	56.89	53.01	47.75
Ours (Nuscenes)	69.24	43.36	41.76	54.44	51.03	44.48

4.3 Convolutional block: preparing data for FPGA

In the method presented in this work a convolutional block similar than the one proposed by [10] has been implemented. One of the main issues of the deep neural network implementation is the so-called vanishing gradient, which means that the gradient of the error used in the back-propagation during training to update the weights gets smaller and smaller on each layer. That leads to the fact that, the deeper the network is, the smaller the gradient would get on each step and therefore the longer the weight update will take. By recirculating the input in the output, similar as proposed in the ResNet [24], the mentioned behavior could be avoided. In equation 1 the formula for weight update is presented, in which η is the learning rate.

$$Wi{+}{=}\ Wi + \eta * \frac{dError}{dWi} \qquad (1)$$

To avoid such behavior a convolutional block based on the good results shown by the [10] proposal has been implemented. As stated there in [10] this proposed convolutional block is based on three paths. One is a direct copy of the input, other with a 3x3 convolutional layer to encode local features and last one is a dilated 3x3 convolutional layer [25] to compute features in further positions, but takes less parameters. Adding these three paths a block equivalent to a 5x5 convolutional is obtained but with fewer parameters, which is helpful to avoid the mentioned vanishing gradient effect.

5. EVALUATION AND EXPERIMENTAL RESULTS

In this section, the experimental results on different datasets are presented. As it can be seen in Table 2 and 3 the presented methodology achieves comparable results with other state-of-the-art approaches in terms of accuracy for vehicle detection with a lower runtime execution (3). Results with Nuscenes dataset seem to be a bit less accurate than other methodologies, but still acceptable when considering that LiDAR sensor has 32-channels and therefore, point clouds have less point density for the classes to be detected.

The visualization of the steps of the presented approach with Nuscenes and KITTI dataset respectively can be seen in Figure 3. In these visualizations the intermediate heatmaps obtained during training are presented together with a projection of the calculated 3D bounding boxes around the detected vehicles. These heatmaps calculated in the output of the FCN layer before the 3 CNN-BN-ReLu phases described in Section 2.2 show the 2D position of the 3D candidates to be a vehicle projected on a 2D space.

The training of this pipeline was done using a training set of 6000 LIDAR sweeps from the Nuscenes dataset and around 3700 samples of KITTI dataset. The validation set is formed by 2000 LIDAR sweeps in Nuscenes and around 3500 in KITTI dataset. The hardware used for the training was 2 NVIDIA GTX 1080 and the FPGA model used for the inference of the trained network is a Arria 10 Intel FPGA with the modelsim-altera software for FPGA development also from Intel.

6. CONCLUSIONS

Here it is presented a method for detecting vehicles from 3D point clouds based on state-of-the-art network architecture such us VoxelNet [1] but adjusted and re-trained on the new dataset Nuscenes, running on a FPGA.

This work presents an implementation based on outstanding works like [10] to adapt the development to run on an FPGA. The results shown in table 3 and 2 demonstrate good comparable results with other methodologies of the presented method running on the KITTI dataset with the particularity that this work is designed and implemented to run on an FPGA compared to the rest of methods presented which run on a GPU.

This paper and the results presented in it motivate the usage of this hardware for deep learning purposes due to its low price (compared to some GPUs) and dedicated hardware architecture for intense computational load.

7. ACKNOWLEDGMENT

This work was supported by the Catalan Government inside the program ”Doctorats Industrials” and by the company FICOSA ADAS S.L.U. J. García López is supported by the industrial doctorate of the AGAUR.

Thanks specially to Christian Fernandez for his help and hard work.

8. REFERENCES

[1] Y. Zhou and O. Tuzel, “Voxelnet: End-to-end learning for point cloud based 3d object detection,” *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4490–4499, 2018.

[2] Andreas Geiger, Philip Lenz, Christoph Stiller and Raquel Urtasun, “Vision meets robotics: The kitti dataset,” *The International Journal of Robotics Research (IJRR)*, 2013.

[3] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, Oscar Beijbom, “nuscenes: A multimodal dataset for autonomous driving,” 2019.

[4] R. Anvari, "Fpga implementation of the lane detection and tracking algorithm," 2010.

[5] Bo Li, T. Zhang, T. Xia, "Vehicle detection from 3d lidar using fully convolutional network," *Robotics: Science and Systems*, 2016.

[6] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d object detection for autonomous driving," *Computer Vision and Pattern Recognition (CVPR),* 2016.

[7] L. Huang, Y. Yang, Y. Deng, and Y. Yu, "Densebox: Unifying landmark localization with end to end object detection," *arXiv:1509.04874,* 2015.

[8] Florian Chabot , Mohamed Chaouch , Jaonary Rabarisoa , Celine Teuliere, Thierry Chateau, "Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[9] L. Novák, "Vehicle Detection and Pose Estimation for Autonomous Driving," 2017.

[10] Yecheng Lyu, Lin Bai and Xinming Huang, "Chipnet: Real-time lidar processing for drivable region segmentation on an fpga," *arXiv preprint arXiv:1808.03506,* 2019.

[11] X. Chen, K. Kundu, Y. Zhu, A. Berneshawi, H. Ma, S. Fidler and R. Urtasun, "3d object proposals for accurate object class detection," *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[12] A. Bulat and G. Tzimiropoulos, "Human pose estimation via convolutional part heatmap regression," *European Conference on Computer Vision (ECCV)*, pp. 4490–4499, 2016.

[13] S. M. Aiden Nibali, Zhen He and L. Prendergast, "3d human pose estimation with 2d marginal heatmaps," *IEEE's winter conference on applications of computer vision (WACV),* 2019.

[14] M. Z. Zia, M. Stark and K. Schindler, "Towards scene understanding with detailed 3d object representations," *Int. J. of Comput. Vision*, 2015.

[15] A.Newell, K. ang and J. Deng, "Stacked hourglass networks for human pose estimation," *European Computer Vision Conference (ECCV),* 2016.

[16] Xiaolong Liu and Zhidong Deng, "A graph-based nonparametric drivable road region segmentation approach for driverless car based on lidar data," *Proceedings of the 2015 Chinese Intelligent Automation Conference,* 2015.

[17] Nicolas Soquet, Didier Aubert and Nicolas Hautiere, "Road segmentation supervised by an extended v-disparity algorithm for autonomous navigation," *Intelligent Vehicles Symposium*, 2007.

[18] Patrick Y Shinzato, Diego Gomes, and Denis F Wolf, "Road estimation with sparse 3d points from stereo data." *Intelligent Transportation Systems (ITSC)*, 2014.

[19] Alejandro González, Gabriel Villalonga, Jiaolong Xu, David Vázquez, Jaume Amores, and Antonio M López, "Multiview random forest of local experts combining rgb and lidar data for pedestrian detection," *Intelligent Vehicles Symposium*, 2015.

[20] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia, "Multi-view 3d object detection network for autonomous driving," *Conference on Computer Vision and Pattern Recognition (CVPR),* 2017.

[21] Y. Xiang, W. Choi, Y. Lin, and S. Savarese, "Data-driven 3d voxel patterns for object category recognition," *Computer Vision and Pattern Recognition (CVPR),* 2015.

[22] A. Canis, J. Choi, B. Fort, B. Syrowik, R.L. Lian, Y.T. Chen, H. Hsiao, J. Goeders, S. Brown, J.H. Anderson, "Legup high-level synthesis," *chapter in FPGAs for Software Engineers, Springer, 2016,* 2016.

[23] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," *Computer Vision and Pattern Recognition (CVPR),* 2017.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," *IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[25] Fisher Yu and Vladlen Koltun, "Multi-scale context aggregation by dilated convolutions," *International Conference on Learning Representations (ICLR)*, 2016.

[26] Liang Xiao, Bin Dai, Daxue Liu, Tingbo Hu, and Tao Wu, "Crf based road detection with multi-sensor fusion," *Intelligent Vehicles Symposium (IV),* 2015.

[27] Xiaofeng Han, Huan Wang, Jianfeng Lu, and Chunxia Zhao, 2017.

[28] Liang Xiao, Ruili Wang, Bin Dai, Yuqiang Fang, Daxue Liu, and Tao Wu, "Hybrid conditional random field based camera-lidar fusion for road detection," *Information Sciences., (XX):1–11, in press*, 2015.

[29] Luca Caltagirone, Samuel Scheidegger, Lennart Svensson, and Mattias Wahde, "Fast lidar-based road detection using fully convolutional neural networks," *Intelligent Vehicles Symposium (IV)*, 2017.