Toward real-time road detection for autonomous vehicles

Lachachi M. Yazid,[a,*] Ouslim Mohamed,[a] Niar Smail,[b] and Taleb-Ahmed Abdelmalik[b]

[a]Universite des Sciences et de la Technologie d'Oran—Mohamed Boudiaf, LMSE Laboratory, Oran Algeria

[b]Universite Polytechnique Hauts-de-France, LAMIH Laboratory, Valenciennes, France

Abstract. Road detection is a vital task for autonomous vehicles, as it has a direct link to passengers' safety. Given its importance, researchers aimed to improve its accuracy and robustness. We look at the task from a holistic point of view, where we aim to balance computation and accuracy. A multimodal road detection pipeline is proposed, which fuses the camera image with the preprocessed LIDAR input. First, the LIDAR input is preprocessed using three-dimensional models inspired from computer graphics to generate image-like representations. Then, the preprocessed LIDAR input is combined with the camera image using a fusion module named inputs cross-fusion module, to reduce the computation amount required by other fusion strategies. To prevent the accuracy loss caused by the computation gain, we introduce the surface normal information to add distinctiveness. Furthermore, we propose a cost/benefit metric to evaluate the trade-off between computation cost and accuracy of road detection approaches. Several tests were conducted using the KITTI road detection benchmark based on deep convolutional neural networks, the obtained results were considered very satisfactory. In particular, the robustness of the proposed approach resulted in accuracies higher than 95% on different road types, comparable to those of the state-of-the-art techniques. In addition to marginally reducing the inference time of the used DCNN on images with a resolution of 1248×352 pixels to 130 ms using an NVIDIA GTX-1080TI. © *2020 SPIE and IS&T* [DOI: 10.1117/1.JEI.29.4.043022]

Keywords: road detection; LIDAR; camera; fusion; surface normal; autonomous vehicle.

Paper 200242 received Apr. 3, 2020; accepted for publication Jul. 28, 2020; published online Aug. 12, 2020.

1 Introduction

Road detection is one of the most important tasks an autonomous vehicle has to perform. This task is charged with determining the drivable area the vehicle uses to navigate. Thus, having a direct association with the safety of its passengers and pedestrians on the road. An unreliable road detection hinders the achievement of higher automation levels as many tasks rely on it,[1–4] making it of great interest to the research community due to the various environmental, sensor, and time constraint challenges.

Multiple sensors have been used to detect the road for autonomous vehicles, such as monocular camera, stereo-camera, and LIDAR. Nowadays, the use of stereo camera has been abandoned in favor of monocular camera, mainly for the ease of use and the small gain in semantic segmentation. The ubiquity of the monocular camera made it ideal for autonomous vehicle research. However, visual noise in images such as shadows or texture similarity proved to be difficult for classical image processing or machine learning.[5,6] With the arrival of deep convolutional neural networks (DCNN), handling the visual noise became easier. Yet, the lack of spatial information in images prevents the DCNN from constantly improving. Recently, the fusion of LIDAR and camera has started to gain popularity, as studies have shown that the two complement each other and are able to surpass image-only limitations.[7–10] However, the different nature of both sensors gives rise to challenges, notably the way the inputs are fused. The LIDAR produces an array of

*Address all correspondence to Lachachi M.Yazid, E-mail: yazid.lachachi@univ-usto.dz

1017-9909/2020/$28.00© 2020 SPIE and IS&T

sparse three-dimensional (3-D) points acquired by bouncing beams of light on real-world objects. These points are in the form of *XYZ* coordinates relative to the LIDAR position, whereas, images are color information in the form of a two-dimensional (2-D) matrix. Due to points sparsity, researchers usually project the LIDAR input into images, then interpolate to make them denser. The interpolation process creates a smoothness effect that needs deeper and heavier DCNN to remedy.

In this work, we introduce the surface normal to eliminate the smoothness effect. The latter adds edge information that distinguishes the different objects in the scenes, easing the DCNN learning. Furthermore, we propose a light fusing strategy that enables the merge of the camera image with the LIDAR interpolated input and the normal surface. Our fusing strategy is designed according to the added surface normal information and does not require doubling the DCNN size like other strategies.[11,12] In addition, to keep coherence between the interpolated LIDAR input and surface normal, our approach inspires from computer graphics and uses 3-D models to compute both.

Model-based road detection architectures are increasing in complexity. As they are getting robust, their computation cost becomes more expensive. These intricate architectures depend upon input preprocessing, custom convolution,[13,14] and complex wiring.[11,12] Although these modifications improve accuracy, they also increase the computation load. This inspired Teichmann et al. [15] to merge low-level tasks, gaining in compute time as a trade-off for accuracy. They propose to perform road, obstacle,[16,17] and pedestrian detection[18,19] with the same architecture, resulting in 93% accuracy in road detection compared to 97% in state of the art. But, given the importance of the detection for safety forbids the accuracy loss. To balance the trade-off between computation cost and accuracy, we propose a cost/benefit metric that can be used as a guide while designing a DCNN. This metric monitors the cost of improvement to the gained accuracy and helps avoiding any unnecessary modification or high accuracy drops.

In this work, we propose a road detection pipeline. The pipeline includes LIDAR input preprocessing, LIDAR-camera fusion module, and segmentation DCNN (shown in Fig. 1). Further, we summarize our contributions as follows:

- We are the first to use surface normal in the context of road detection, to add more distinctiveness.
- We propose a fusion module to merge surface normal with interpolated data and camera image.
- We propose a simplistic way to generate interpolated representation and surface normal, using a unified approach that benefits greatly from GPUs.
- We introduce a cost-benefit metric to evaluate different model's performance.

In Sec. 2, we present the state of the art in road detection. Fusion strategy and LIDAR input preprocessing are then provided in Secs. 3 and 4, respectively. Our cost/benefit metric is then discussed in Sec. 5, followed by our experiments in Sec. 6. Finally, we finish our paper with a conclusion summarizing all our work.

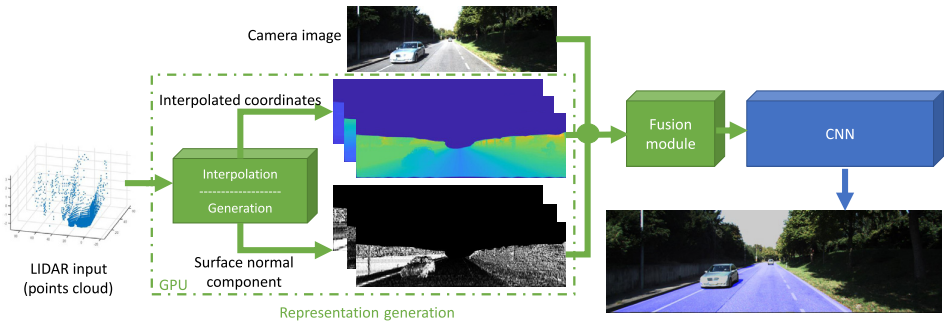

Fig. 1 Our road detection pipeline with the representation generation and fusion module.

2 Related Work

Robust road detection is the foundation for safe navigation. Given its importance, research has been actively ongoing in this area. The aim is to improve segmentation accuracy while maintaining a low computational cost. Road detection approaches are divisible into two categories. The first category is monocular camera-based, where we only use images. In the second, the fusion of LIDAR and camera data is considered, methods in the second group fall into two subcategories: sparse and dense LIDAR inputs.

Machine learning techniques had a big hand in improving road detection algorithms. Researchers took different approaches to solve this challenge, where classifiers have been employed to carry the task. Most notable approaches used SVM,[20]graph-cut,[21] appearance learning,[22] and random decision forest.[23] Despite the progress made with machine learning techniques, they still underperform against model-based approaches. In Ref. 24, Chen and Chen proposed a joint road and road edges detection framework, thus, limiting the detection area to road boundaries. They used a DCNN to retrieve features at different scales then performed a pixelwise classification with a Bayesian model. Munoz-Bulnes et al.[25] attempted to improve the DCNN ability to generalize by never introducing the same image twice. Their approach relied on ResNet-50, which helped them tackle the task as a classification problem. In Ref. 26, Mohan proposed an end-to-end framework, relying on convolutional and deconvolutional layers. He suggested to divide the image into patches and process it in multiple passes. Following a different perspective, Teichmann et al.[15] investigated the possibility of merging the many tasks of the vehicle into a single model. Their model was able to get encouraging result given the number of trained tasks.

Recently, multimodal systems combining a camera with LIDAR started to emerge. In Ref. 11, Chen et al. managed to surpass all approaches listed in the KITTI[27] road detection challenge. The authors proposed to perform altitude difference on LIDAR data, then process it jointly with the camera image. Caltagirone et al.[12] interpolated the LIDAR data guided by the accompanying image as proposed by Ref. 7. The interpolated data were fed with the image to a two-branch architecture, which we will discuss in the following section. Gu et al.[8] proposed a conditional random field (CRF) framework as a new approach in road detection. They compute the final result from geometrically interpolated LIDAR points and image-based road detection. Contrary to the last two approaches that use dense LIDAR input, Refs. 9 and 10 employed sparse LIDAR input in conjunction with model-based detection under the CRF framework to fulfill the task.

All multimodal approaches use some form of input fusion. Although these fusion approaches permit the improvement of accuracy, they impose limitations and overheads.

This is mostly due to challenges with fusion strategies and the input fed to them. Early-fusing the inputs is often unconsidered as it results in less precise segmentation.[12] This is caused by smoothness in the interpolated LIDAR input. Late fusing is the most dominant approach, where each input is processed on a different CNN network and then fuses their rich feature maps to obtain the final result. However, this approach doubles the amount of required computation compared to early fusion for a small improvement. Lastly, cross fusion is the new emerging approach,[11,12] as it is built on late fusion and provides more robustness by connecting the layers of both networks.

In this work, we differ from other approaches by proposing a less computation heavy network based on an early fusing, which can obtain results close to those of the late fusion. Furthermore, it introduces surface normal to cut smoothness caused by interpolation and limits the accuracy loss. Finally, preprocessing is inspired by computer graphics to enable full in-GPU processing.

3 Fusion Module

A fusion strategy is what helps merge two pieces of information of different natures and improves the final accuracy. In the context of this work, the fusion is meant to merge the camera image with the LIDAR interpolated input. This fusion can be as simple as concatenation or as complex as doubling the size of a DCNN. In the context of fusion strategies, doubling the size of

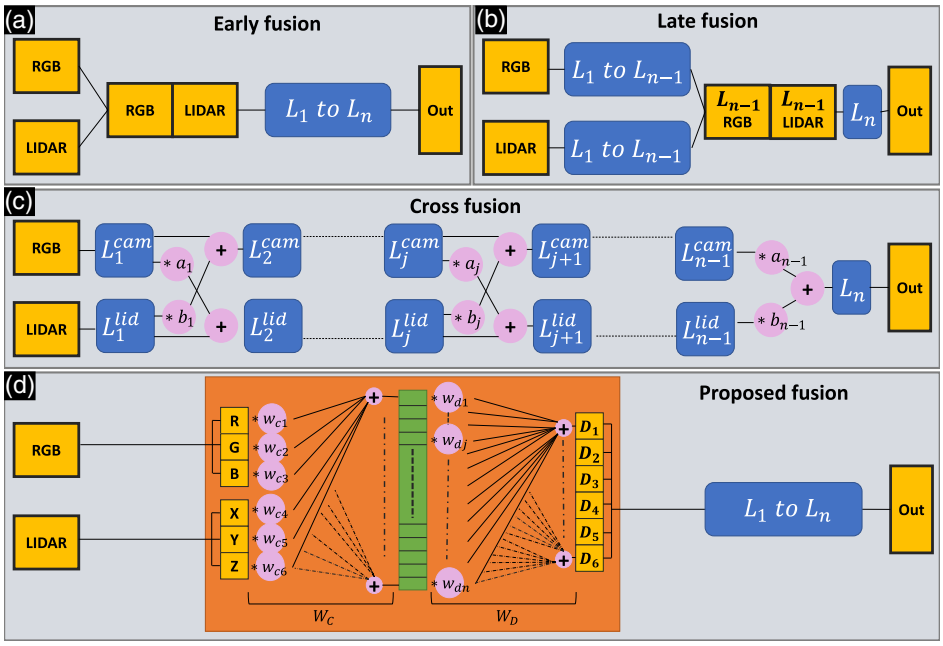

Fig. 2 The different fusion approaches: (a) early fusion, (b) late fusion, (c) cross fusion, and (d) our proposed fusion named inputs cross-fusion. Here, we show how the fusion strategies can be applied on an arbitrary DCNN with *n* layers (*L*). The surface normal input was not shown on purpose, to keep coherence with other approaches.

DCNN implies that for each input type, we duplicate a base DCNN architecture, then we cross-fuse them to obtain a final DCNN. To our best of knowledge, only two works[7,12] dealt with this problem for general-purpose image segmentation DCNN. Here, we investigate the fusion strategies in the literature and further propose our own fusion strategy with inputs cross-fusion module.

3.1 *Existing Approaches*

In this study, we limit the approaches to those that process interpolated LIDAR data and images. We denote the approaches as follows: early fusion [Fig. 2(a)], late fusion [Fig. 2(b)], and cross fusion [Fig. 2(c)]. Premebida et al.[7] were the first to experiment with late fusion, where they proposed to process sensor acquisitions separately. Each sensor had a dedicated CNN to extract discriminative features then merged the outputs with an SVM classifier to get the final segmentation. Caltagirone et al.[12] investigated the other two approaches. The first is the early fusion approach. It concatenates the two sensors' inputs and forwards them to the CNN. This approach processes the inputs together, thus reducing computation cost. The last approach is cross fusion, each sensor is processed in a separate branch with a weighted update from the adjacent branch.

3.2 *Proposed Approach*

In this work, we aim to close the gap between late and early fusion approaches while considering computation costs. By investigating the issues related to interpolation, we showed that additional representations are needed to remedy. Qiu et al.[28] worked on depth prediction with similar sensor configuration. The authors used the surface normal as an intermediate representation to capture more variations in the input. They argue that the change in the normal helps the DCNN generalization ability and adds to it robustness. We propose to use this information to cut smoothness produced by the interpolation. However, we inform the reader that both interpolation and normal estimation must be jointly performed. Separate processing will misalign them and confuse the

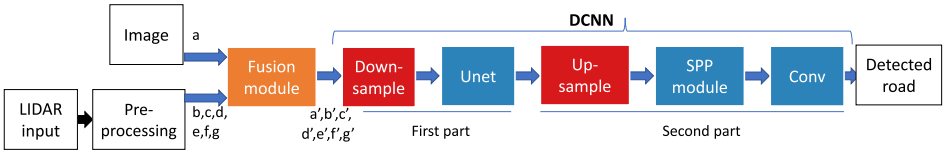

Fig. 3 Detailed view of our proposed road detection pipeline.

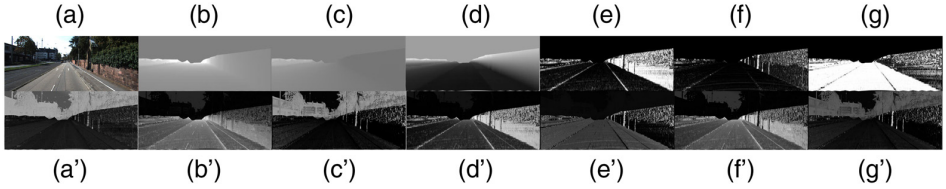

Fig. 4 The input and output of fusion module. First row contains image from camera (a) and generated representations from LIDAR (b to g). Second row shows the fusion module output, where each one (a' to g') is computed by cross fusing all the inputs (a to g).

DCNN learning process. The following section discusses our processing method and how we resolve these issues.

With the added surface normal, our approach uses three inputs: image, XYZ coordinates, and surface normal shown in Fig. 1. Applying late fusion or cross fusion strategy will triple the size of the DCNN compared to early fusion. To keep the model size manageable, we propose to perform a cross fusion on the inputs. The basic idea is to generate distinctive representations by combining all the inputs for each representation [Figs. 4(a)–4(g)]. The inputs cross fusion is performed in two steps. First, we combine the inputs using a weighted sum and generate a high number of features to maximize the chances of learning useful features. In essence, we search to combine color, coordinates, and normal in a meaningful way that eases the DCNN task. Second, we reduce the number of features by applying the weighted sum again. This helps lower computation in the following DCNN layer and remove unuseful features. Figure 2(d) shows our fusion module performing the input cross fusion only on RGB image and XYZ coordinates to keep coherence with previously explained strategies. Further, we formulate the input cross fusion in Eq. (1), where D_i is one of the generated representations, f is the weighted sum function, W_C is the weights to compute the features, and W_D is the weights to reduce the features count:

$$D_i = f\{W_D, f(W_C, [\mathrm{RGB}, XYZ, \mathrm{normal}])\}. \tag{1}$$

Figure 3 shows our road detection approach with DCNN architecture details. We labeled the inputs (a–g) and outputs (a'–g') of our proposed fusion module and provide a visual in Fig. 4. The first row contains the RGB image (a), interpolated LIDAR XYZ coordinates ($b-d$), and the computed normal component ($e-g$). The second row shows the module output. As it can be noticed, the module can produce discriminative representations. (b') gave importance to the LIDAR-covered area. Whereas, (c') emphasized the area that does not have LIDAR coverage. Further, (d') reproduced automatically an output similar to ADT preprocessing[11] designed by a human.

4 LIDAR Input Preprocessing

As shown in Fig. 1, our system takes three types of inputs. The first input comes from the camera, and the other two come from the processed LIDAR input. Figure 5 describes our LIDAR input preprocessing steps to compute surface normal and interpolate LIDAR input, which are named representations. In this work, we propose geometrical interpolation, contrary to guided interpolation.[7,8] Only the LIDAR input is used during interpolation without interrogating the

Fig. 5 Our LIDAR input preprocessing steps.

accompanying image. We differ from approaches in literature by applying computer graphics techniques rather than image processing. We employ 3-D modeling to offload the computation to the GPU and ease the manipulation of the LIDAR input without affecting performance.

4.1 *3-D Model Construction*

A 3-D model is a complex construct composed of vertices and edges. Vertices are 3-D points with x, y, and z coordinates, and edges are connections between vertices. By grouping the edges into triangles, a GPU can be used to process these triangles in parallel and generate image-like representations shown in Figs. 4(b)–4(g). In this work, we compute both interpolation and surface normal using triangles. This assures that an interpolated value uses three closest points, reducing the smoothness. Further, the normal surface can produce sharp edges cutting the smoothness effect. Figure 4 shows the normal component in **e**, **f**, and **g**.

By considering the LIDAR input as vertices, we can create a 3-D model by searching the edges that connect them. Here, we present our steps to recover the connectivity information. First, the LIDAR input is projected into an image I_P. This helps process the points in reduced dimensionality. The point projection is as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{T}.P_R.\mathbf{R}.\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \tag{2}$$

where x, y, z are the coordinates of the point in the real-world, and u, v its coordinates on the image. T is the LIDAR–camera transformation matrix, P_R is the projection matrix, and R is the camera rectification matrix. We then acquire the connectivity by applying a 2-D triangulation of Delaunay. The divide-and-conquer variant is adopted in this work for its proved efficiency.[29] Then, we bind the acquired connectivity with the LIDAR input to get the 3-D model. Figure 6 shows the LIDAR input and the constructed 3-D model. Afterward, we rectify the error caused by occlusion and generate the new inputs. We delve more into details in the following sections.

4.2 *Rectification of Noise in Constructed 3-D Model*

In a LIDAR-camera multimodal configuration, both sensors are distant. Thus, giving the sensors two views of the real world, where the LIDAR can see parts occluded to the camera. The occlusion produces areas in the projection image I_P where values differ greatly. Consequently, deforming the constructed 3-D model as noise. To rectify this challenge, we take advantage of the acquired connectivity. By interrogating the 3-D model, we change each point position depending on its neighbors. If the variance in the neighbors' values is <10 cm, then no changes are made. Otherwise, we average 25% of the closest points to the LIDAR to compute the new position. Here, we set the variance threshold to 10 cm to limit the smoothing effect on road edges, as the averaging will blur the sharpness of surface normal representation around the sidewalk curb. Since the standard height of a curb is 15 cm, the variance for a point in the middle of the curb is 7.5 cm, which we round to 10 cm to account for construction defects. Although our approach is able to construct a 3-D mesh and rectify it, it was primarily developed for road detection. We bring the reader attention to the fact that our approach can be further improved to extend it to other tasks such as obstacles detection and navigation directly from the 3-D mesh.

In Fig. 7, we show results obtained before and after rectification. In Fig. 7(b), the orange box shows the effect of superposed points of the pole and the wall behind it. Figure 7(c) shows how