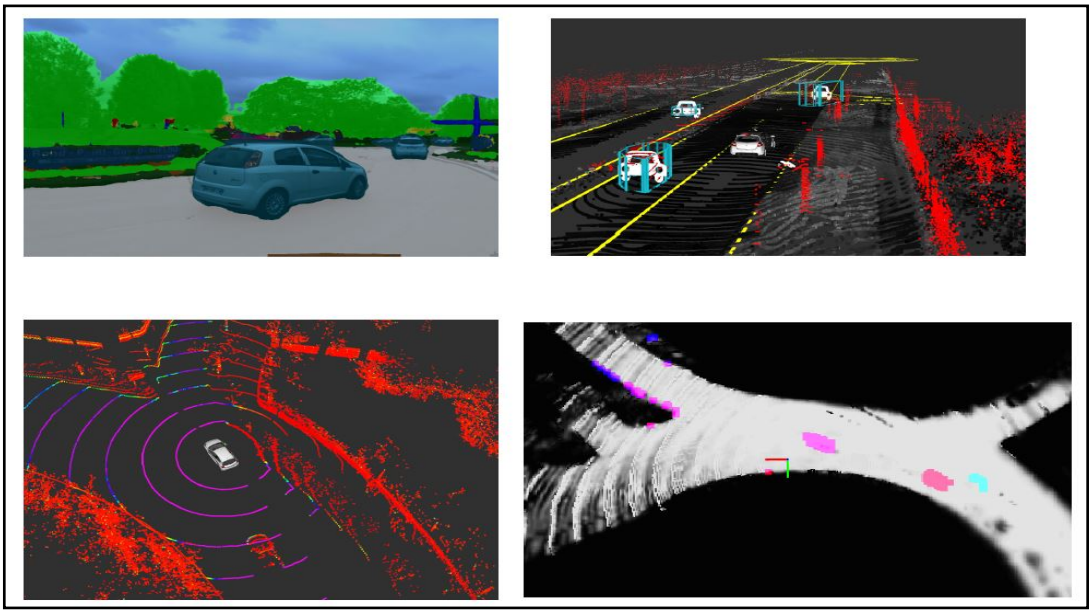
Application of machine learning techniques for evidential 3D perception, in the context of autonomous driving

Édouard Capellier

Par **Édouard CAPELLIER**

Application of machine learning techniques for evidential 3D perception, in the context of autonomous driving

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 14 janvier 2020
Spécialité : Robotique et Sciences et Technologies de l'Information et des Systèmes : Unité de recherche Heudyasic (UMR-7253)
D2534

UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor

by ***Edouard CAPELLIER***

Application of machine learning techniques for evidential 3D perception, in the context of autonomous driving

Application de l'apprentissage machine pour la perception crédibiliste 3D, dans le contexte de la conduite autonome

Robotique et Sciences et Technologies de l'Information et des Systèmes

Defense date: January 14th, 2020
Thesis committee:

Reviewers	Frédéric Pichon	Université d'Artois
	François Goulette	Mines ParisTech
Examiners	Thierry Denoeux	Heudiasyc - UTC
	Samia Ainouz	LitisLab - INSA de Rouen
Supervisors	Véronique Cherfaoui	Heudiasyc - UTC
	Franck Davoine	Heudiasyc - UTC
Invited Member	You Li	Renault SAS

Abstract

The perception task is paramount for self-driving vehicles. Being able to extract accurate and significant information from sensor inputs is mandatory, so as to ensure a safe operation. The recent progresses of machine-learning techniques revolutionize the way perception modules, for autonomous driving, are being developed and evaluated, while allowing to vastly overpass previous state-of-the-art results in practically all the perception-related tasks.

Therefore, efficient and accurate ways to model the knowledge that is used by a self-driving vehicle is mandatory. Indeed, self-awareness, and appropriate modeling of the doubts, are desirable properties for such system. In this work, we assumed that the evidence theory was an efficient way to finely model the information extracted from deep neural networks. Based on those intuitions, we developed three perception modules that rely on machine learning, and the evidence theory. Those modules were tested on real-life data.

First, we proposed an asynchronous evidential occupancy grid mapping algorithm, that fused semantic segmentation results obtained from RGB images, and LIDAR scans. Its asynchronous nature makes it particularly efficient to handle sensor failures. The semantic information is used to define decay rates at the cell level, and handle potentially moving object.

Then, we proposed an evidential classifier of LIDAR objects. This system is trained to distinguish between vehicles and vulnerable road users, that are detected via a clustering algorithm. The classifier can be reinterpreted as performing a fusion of simple evidential mass functions. Moreover, a simple statistical filtering scheme can be used to filter outputs of the classifier that are incoherent with regards to the training set, so as to allow the classifier to work in open world, and reject other types of objects.

Finally, we investigated the possibility to perform road detection in LIDAR scans, from deep neural networks. We proposed two architectures that are inspired by recent state-of-the-art LIDAR processing systems. A training dataset was acquired and labeled in a semi-automatic fashion from road maps. A set of fused neural networks reaches satisfactory results, which allowed us to use them in an evidential road mapping and object detection algorithm, that manages to run at 10 Hz.

Acknowledgements

First of all, I would like to thank my supervisors Veronique Cherfaoui, Franck Davoine, and You Li, for their availability and advice during this 3-year PhD. I also want to express my gratitude to the members of the jury who have agreed to ˙participate in my PhD defense. I would like to thank my colleagues in the Heudiasyc laboratory with whom I have had the chance and the pleasure to collaborate, especially Thierry Monglon and Stéphane Bonnet. I appreciated their help and remarks as well as the discussions we have had. My sincere thanks go to all the people that I have met at Heudiasyc and Renault, and especially to the friends I have made during this 3-year-long journey. I also want to thank Nicolas Caddart who, in the context of the internship that he did at Renault, participated in the last work that will be presented in this manuscript, by helping me to manually label LIDAR scans, and by developing and testing software modules that greatly helped me in my work. Last but not least, I want to express my profound gratitude to my parents, my brother and my partner, who supported me through my moments of doubts, both physically and mentally.

Contents

Chapter 1

Introduction

Contents

1.1 General context

Self-driving urban vehicles have been fantasized for more than thirty years, by engineers and researchers all around the world. Launched in the late 1980's, the Eureka PROMETHEUS Project (Programme for a European traffic of highest efficiency and unprecedented safety) was for instance the first large-scale R&D project in the field of autonomous driving, and involved more than forty academic and industrial partners over Europe. Inventions that date back from this research effort were the basis of many modern advanced driving assistance systems (ADAS), such as autonomous intelligent cruise control (AICC) [1], that are nowadays available in consumer cars. In 2007, the DARPA Urban Challenge demonstrated the feasibility of autonomous driving in quasi-urban environments [2]–[4]. Since then, many experiments, both on controlled areas and open roads, have been led in various locations, although safety drivers or remote operators were still monitoring the vehicles [5]–[7].

One of the most paramount issues to solve, in order to make self-driving urban vehicles socially acceptable and generalized, is possibly the perception task. Perception, in general, is defined as *the ability to see, hear, or become aware of something through the senses (Oxford dictionary, 2019).* Of course, self-driving vehicles do not have senses per se, but instead can rely on various sensors, such as cameras, laser scanners, or radars, to get a raw representation of their environment. The perception task for a robotic system, similarly to what it is for human beings, can then be defined as the process of extracting information from raw sensor inputs. In the case of self-driving vehicles, the information to be extracted is probably extremely specific: one could easily consider that a self-driving urban vehicle has to know where is the road for the autonomous driving task, but do not especially need to know to color of the eyes of all the pedestrians that are present in the scene.

The ability to extract key information from sensor inputs, and to prove that key information will always be extracted, will be what will make self-driving vehicles viable. Indeed, a recent report from NHTSA [8] estimated that, in the US, between 2005 and 2007, the critical reason that led to a car crash could be assigned to the driver in approximately 94% of the cases. Recognition errors, including *driver's inattention, distraction and inadequate surveillance*, and *non-performance errors*, which mostly means sleep, were considered as the actual critical reason in nearly half of those crashes. Theoretically, self-driving vehicles could then outperform human drivers in terms of crash numbers, as computers do not need to sleep, and cannot be distracted as easily as humans. This however supposes that those autonomous systems can perceive their environment, and extract meaningful information, as efficiently as humans, in all cases. Otherwise, self-driving vehicles might only be considered as very expensive robots, that have failure modes that are just different from humans, and that are thus not that useful.

Ensuring that a perception system always reaches a sufficient performance level is a very hard, and composite problem. Indeed, this could require to solve several cognitive, nay philosophical, problems, such as:

- Quantifying the distance between the performances of a perception system, and the performances of human drivers
- Listing exhaustively all the tasks that have to be performed, by a self-driving vehicle, to achieve at least human-like perception
- Ensuring that each one of these atomic tassk is achieved with sufficient performance, in dedicated situations
- Ensuring that each one of these atomic tasks is achieved with sufficient performance, in all situations

We do not know how to solve those issues, as perception systems are often evaluated on a per-pixel basis, that is very different from the way humans understand the world. We do not even know if those issues can be solved, nor if all those issues have to be solved. We however consider as a sound assumption that perception systems, when used in self-driving vehicles and possibly compared with human drivers, should represent their trust in what they perceive as precisely as possible, and as objectively as possible. In this context, machine learning approaches seem of particular interest for perception tasks. Indeed, such algorithms are mathematically optimized to work on datasets, which are assumed to depict, exhaustively, all the possibile use cases that should be covered.

The recent technological achievements, in autonomous driving, were concurrent with the generalization of the use of machine-learning algorithms, in autonomous vehicles. This was justified by the success of Stanley, a prototype

vehicle designed in Stanford University, which won the 2005 DARPA Grand Challenge, and relied on machine learning, especially to detect the drivable area [9]. Since then, several driving oriented labeled datasets were publicly released, such as the KITTI dataset [10] or the CityScapes dataset [11]. Consequently, various and numerous contributions in the field of perception for autonomous driving, that rely on recent machine learning techniques, were proposed over the recent years. Deep convolutional neural networks are especially popular for vision-related tasks, such as semantic segmentation and instance segmentation in 2D images. Machine learning algorithms have the benefit of being as objective as possible, as they basically learn to extract statistical features from a training dataset, that is supposed to be representative of the task they have to solve. For tasks that have objectively correct answers, as it is the case for several perception tasks, using machine learning algorithms could be safer than relying on explitcit, and therefore subjective, models. Yet, this does not mean that machine learning algorithms in general, and deep neural networks in particular, do not have intrinsic limitations and failure modes that have to be accounted for [12]. In this context, we assumed that the theory of belief functions could be of interest, in order to represent more properly the behavior of machine learning algorithms, and deep neural networks in particular.

The theory of belief functions, or evidence theory, extends the commonly used Bayesian framework, by reasoning on the power set of the frame of discernment. A more detailed representation of the knowledge is then obtained. For instance, to model the result obtained after having tossed a dice, the Bayesian framework would typically be used to represent the fact of having either a one, a two, a three, a four, a five or a six. In the same conditions, for the same possible outcomes, the evidence theory will allow for other results to be modeled, such as having an ambiguous result. This could correspond, for instance, to the fact that the dice was tossed, but rolled under a dresser, which makes the result impossible to read. Again, in the context of autonomous driving, being able to obtain such additional and refined knowledge from the perception system is, probably, a desirable property, to model the current performance-level of a perception system.

1.2 Framework of the work and objectives

Our work was focused on the proposition of perception modules, intended to be used by a self-driving urban vehicle, which relied on machine learning to accomplish perception-oriented tasks, and the evidence theory to represent the knowledge they extract. We thus aimed at obtaining objective knowledge that is accurately represented. In particular, we focused on a 3D understanding of the scene, as autonomous vehicles typically need to evolve in a 3D environment. The algorithms were designed to be integrated into the software stack of prototypes of autonomous electric shuttles, called Renault ZoeCab.

We thus primarily focused on the use of RGB images and a LIDAR scanner. Localization sensors, such as GNSS receptors with RTK corrections or odometry sensors, were also used, but only to realign the data over time. HD maps are also available to ZoeCab systems in dedicated areas. By HD maps, we designate geo-referenced maps that have a centimeter-level accuracy. We chose not to rely on them, at least during the actual perception task, as we considered that this pre-existing information is not perceived, per se. We however benefited from them, for offline processings.

We focused on only a subset of all the possible perception tasks, that seemed mandatory to us for the deployment of self-driving urban vehicles. Those tasks are road detection, object detection, and object classification. As ZoeCab systems are intended to be deployed as urban shuttles, the algorithms are tailored for urban and peri-urban areas. We, for instance, do not assume that our road detection algorithm can be used directly on highways. Our algorithms were tested on real-life data acquired from ZoeCab systems and, when possible, trained on data obtained from those prototypes. They were thus implemented as ROS nodes, that can be interfaced with other ROS modules that exist for ZoeCab systems.

This work was part of the research activities of SIVAlab (**LAB**oratoire des **S**ystèmes **I**ntègres pour le **V**éhicule **A**utonome), a joint laboratory between Renault SAS and the Heudiasyc Laboratory, UMR UTC/CNRS 7253. It was funded by a CIFRE fellowship, that was granted from Renault SAS.

1.3 Organization of the thesis

The next chapters of this manuscript are organized as follows.

Chapter 2 briefly presents the current state-of-the art machine learning algorithms to solve typical 2D and 3D perception tasks, in autonomous driving. Most of those approaches rely on the use of very specific architectures, intended to be used on images: convolutional neural networks. As we could not exhaustively cover all the existing perception tasks that have been addressed, we only focus on those that correspond to the modules we chose to develop for ZoeCab systems: road detection, semantic segmentation, and 3D LIDAR object detection and classification.

Chapter 3 covers the general evidence theory, and presents both the framework and usual mathematical tools that are used to reason in this framework. We present the Dempster-Shafer combination rule, its unnormalized variant, several indicators about the quality of evidential mass functions, and how evidential mass functions can be expressed in terms of probability. We also present some applications of the evidence theory in perception for autonomous driving, that

do not necessarily rely on machine learning. Most of those applications are grid mapping systems.

Chapter 4 presents a first, naive, use of the evidence theory, in a sensor fusion context. A deep convolutional neural network, that was previously trained to output semantic segmentation results from RGB images, is used to obtain classification results, that are semantically reinterpreted in the evidence theory. An asynchronous fusion of the segmentation results is realized with observations from a 360° LIDAR scanner. Though flexible, this approach does not completely benefit from the evidence theory, as the outputs are only reinterpreted semantically, while the actual behavior of the neural network could have been modeled using the theory of belief functions.

Chapter 5 thus presents a neural network used as a LIDAR object classifier, that can be reinterpreted as a fusion of simple evidential mass functions. In this work, we mainly focused on the classification of objects as either vehicles, or vulnerable road users. The reinterpretation of the classifier as performing a fusion of simple mass functions allowed us to rely on a simple statistical filtering scheme, to detect and rejects mass functions that are not usable for the final classification of the LIDAR objects, and allow for an operation in open-world.

Chapter 6 builds on this previous work, and proposes two convolutional architectures for road detection in LIDAR scans. One of them is heavily inspired by PointNet [13], and processes LIDAR rings, while the other one is inspired by both PointNet and SqueezeSeg [14], and processes LIDAR scans that are organized as dense range images. Evidential mass functions can be obtained from the systems in both cases, and later used in a fusion framework. A training dataset for those systems was obtained in a semi-automatic fashion, from geo-referenced maps.

Chapter 7 extends section 6, and presents an evidential road surface mapping algorithm, that relies on a fusion of neural networks. LIDAR scans are segmented, and their results are accumulated over time to build a dense representation of the road structure. A conflict analysis, during evidential fusion, is also used to detect road obstacles during the fusion process.

Chapter 8 finally concludes our manuscript, summarizes our contributions, and offers some perspectives about possible future works.

This manuscript is based on the following publications, that we proposed during out PhD research work from 2017 to 2020:

- Capellier, E., Davoine, F., Frémont, V., Ibañez-Guzmán, J., & Li, Y. (2018, November). Evidential grid mapping, from asynchronous LIDAR scans and

RGB images, for autonomous driving. In 2018 21st International Conference on Intelligent Transportation Systems (ITSC) (pp. 2595-2602). IEEE.

- Capellier, E., Davoine, F., Cherfaoui, V., & Li, Y. (2019, June). Evidential deep learning for arbitrary LIDAR object classification in the context of autonomous driving. In 2019 IEEE Intelligent Vehicles Symposium (IV) (pp. 1304-1311). IEEE.

- Capellier, E., Davoine, F., Cherfaoui, V.,& Li, Y. (2019, November). Transformation-adversarial network for road detection in LIDAR rings, and model-free evidential road grid mapping. Proceedings of the 11th workshop on Perception, Navigation and Planning for Intelligent Vehicles. Available online at https://project.inria.fr/ppniv19/program/

- Capellier, E., Davoine, F., Cherfaoui, V., & Li, Y. Fusion of neural networks for road detection in LIDAR scans, and evidential road surface mapping – under submission to the "Journal of Field robotics"

Chapter 2

Machine learning for perception in autonomous driving

Contents

2.1 Introduction

This section focuses on a presentation of how machine-learning is used to solve perception problems, in the context of autonomous driving. As previously said, we only focus on a subset of tasks, that correspond to those that we considered during our thesis. First of all, we will present the datasets that are currently available for the training and evaluation of perception modules. Then, we will briefly present how deep neural networks, which are the most common machine-learning architectures nowadays, are used to process images and LIDAR data. Finally, we will present the current state-of-the-art approaches for road detection, semantic segmentation, and 3D LIDAR object detection and classification that are available on those datasets.

2.2 Which datasets for which tasks ?

As machine-learning algorithms are data-driven approaches, their development depends on the availability of pre-existing datasets. One of the first tasks, in an autonomous driving context, that was solved using machine-learning, is road detection. This was justified by the availability of the KITTI road dataset [15], which consists in RGB images and pixel-level road labels, and the ease of labeling of the road in RGB images. Indeed, relatively accurate road labels can be obtained by polygonal annotation of RGB images. The KITTI road dataset also includes LIDAR scans, that were synchronized and calibrated with regards to the RGB camera that generated the labeled images, so that road detection algorithms developed from the KITTI dataset can also rely on 3D information. However, although road detection is crucial for autonomous driving, many other classes of objects have to be detected. This thus lead to the release of the CityScapes dataset, which consists in 5000 RGB images that were recorded from a driving vehicle. Those images were finely labeled according to nineteen semantic classes that were decided to be useful in the context of autonomous driving (pedestrian, car, truck, road, sidewalk, among others), and 20 000 additional images with coarse labels were also released. Instance-level labels are also available.

Yet, both KITTI and CityScapes are image-oriented dataset and, even if LIDAR scans or disparity maps are made available alongside the labeled data, the evaluation procedures are always biased towards an interpretation in the image domain. For instance, a LIDAR-only road detection approach evaluated on KITTI is expected to produce dense road detections, even if LIDAR scans are sparse by nature. The labeled data also only corresponds to the front view of the vehicles in both cases, which limits the possibility of using those dataset for complete scene understanding. This justified the release, in 2019, of many similar datasets, that are oriented towards 3D object detection, tracking, and scene understanding: the TUBS dataset [16], NuScenes [17], Argoverse [18], Lyft level 5 [19], and the Waymo Open Dataset [20]. All those datasets include RGB images and 3D 360°LIDAR data, in which objects of interests have been labeled via bounding boxes. NuScenes, Argoverse and Lyft level 5 also include road maps corresponding to the area where the raw data was acquired, so as to depict the reality of the road network in the dataset. Some evolutions are expected for these five datasets, such as new kinds of labels, or additional data.

Semantic segmentation of LIDAR scans is another dynamic field, where datasets are emerging. One of those recent datasets is the Paris-Lille-3D dataset [21], which consists in approximately 2 kilometers of labeled LIDAR scans. Yet, the LIDAR used in dataset was mounted with an angle of 30 degrees between the axis of rotation and the horizontal. Also the individual scans are registered into a dense point-clouds, and the evaluation procedure used for this dataset focuses on coarse classes. For instance, roads and sidewalks are subsumed under a ground class. As a result, this dataset is mainly oriented towards the task of semantic

mapping, rather than perception. Similarly to the KITTI dataset, the TUBS dataset for instance recently included fine labels for seven classes of objects in RGB images, which can easily be projected into the LIDAR scans. Finally, again in 2019, the SemanticKITTI dataset was released [22]. SemanticKITTI consists in more than 43000 LIDAR point-clouds, splitted into 22 sequences of consecutive scans that originally belonged to the KITTI dataset, that were labeled at the point-level according to 28 semantic classes.

A dynamic community is thus participating in the release of new datasets that enable the development of perception modules for autonomous driving that heavily rely on autonomous driving. 2019 was especially a very dynamic year in terms of datasets for autonomous driving. The tasks of semantic segmentation of LIDAR point-clouds, semantic SLAM, and map-aided perception will probably gain popularity, as large-scale datasets are now available, to develop and evaluate those approaches. Yet, the fact that some of these new datasets (especially NuScenes, Lyft level 5, and Argoverse) have very similar features, makes it difficult to anticipate whether one of these datasets will become a standard, or if researchers will have to evaluate all of their approaches with regards to those datasets. One might be afraid that the amount of time needed to properly evaluate approaches, on all of these new datasets, will make the global research work focus on very specific sensor setups and situations. Roundabouts are for instance very rare in those datasets. Field experiments, if they ever happen, will also have to be properly justified, especially for the semantic-SLAM, object detection and object classification tasks.

2.3 Deep neural networks architectures for images and LIDAR scans

2.3.1 Artificial neural networks (ANNs)

The most widely used machine learning algorithms, for perception tasks, belong to a very specific class of algorithms: deep neural networks, that extend artificial neural networks (ANN). ANNs are inspired by the formal model of neuron that was proposed by McCulloch and Pitts, which corresponded to the way biological neurons were expected to work in the 1940's. Such a neuron is modeled as a mathematical function, with x an input numerical vector of n values and y the output of function, as follows:

$$y = \phi(w_0 + \sum_{i=1}^{n} w_i x_i) \tag{2.1}$$

ϕ is typically a non-linear function, such as the Heaviside function or the ReLU function. The w_0 parameter is called *bias*, and the bias and w_i parameters are

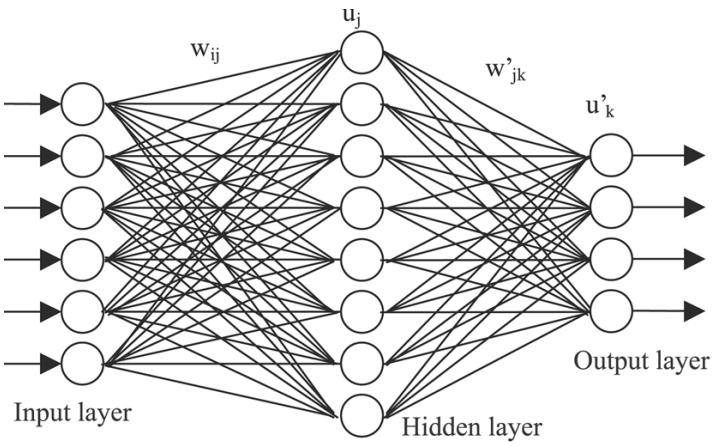


Figure 2.1: Example of artificial neural network (ANN). The input layer corresponds to the input vector, and a numerical vector is outputted by the function.

selected according to the application modeled by this function. For instance, if ϕ is the Sigmoid function, the neuron performs binary logistic regression, and can be used to model the probability that the input belongs, or not, to a class. To model complex functions, those neurons, or units, can be organized in networks. An ANN then consists in a set of units that follow the mathematical model in Equation 2.1, and are organized in stacked layers, as depicted in Figure 2.1. Between the input and output layers, at least one hidden layer of units is present.

Deep neural networks are actually ANNs that have more than one hidden layer. The parameters of each unit are usually learnt, thanks to an objective function, or loss function, that penalizes errors of the system (for instance, misclassifying pixels), and an optimization algorithm, such as the stochastic gradient descent algorithm. In particular, deep neural networks which have all their units connected together are called multi-layer perceptrons (MLP). Other types of deep neural networks have been proposed, to process specific types of data. In particular, convolutional neural networks (CNN) are refined deep neural networks that are used to process images.

2.3.2 Convolutional neural networks (CNNs): ANNs for image processing

The need for specific architectures for images was justified by the fact that fully connected architectures, which would take image pixels as inputs, would be extremely inefficient. Indeed, an MLP that would directly be fed with image pixels would only be able to process images with a fixed number of pixels. Local features would also be hard to extract for such an MLP, as each unit would only reason on the global image. A high sensitivity to shift, distortion and scaling would have to be expected in this case. CNNs, instead of reasoning on full im-

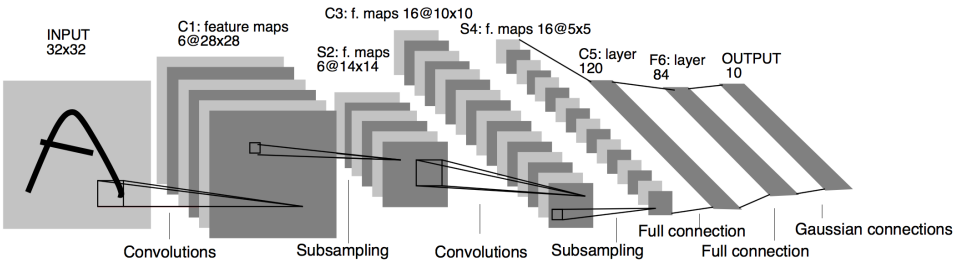


Figure 2.2: LeNet5, a convolutional neural network for handwritten digit recognition. Illustration originally coming from [23].

ages, reason on neighborhoods. Each unit only processes neighboring pixels, in a sliding window fashion. As a result, the w_i parameters in Equation 2.1 are then seen as the parameters of a convolution matrix. The units of such networks are thus relying on a bank of convolutional filters whose parameters are learnt, which justified that such deep ANNs for image processing were called convolutional neural networks. Consecutive convolutional hidden layers are then applied to the image, to extract features from the images. To further enforce invariance to rotation, distortion and scaling, CNNs architecture can also subsample outputs from hidden layers, generally via Max-Pooling or Average Pooling. Finally, fully connected hidden layers can, for instance, be used to reason on the features extracted by the consecutive convolutional and sub-sampling operations. LeNet5, which is depicted in Figure 2.2, follows this typical architecture, and applies it for handwritten digit recognition.

LeNet5 was originally used for a task of image classification. However, CNNs can also be used for semantic segmentation, which is equivalent to per-pixel classification. In this case, the output size of the network is expected to be equal to the size of the input image. Upsampling is thus needed. Most approaches for semantic segmentation of images rely on a hourglass-like architecture: early convolutional and sub-sampling operations are used to extract features from the input image (or *encode* the image); then upsampling and convolutional operations are used, in a decoder network, to predict per-pixel segmentation results, from the encoded features. An example of typical encoder-decoder network for semantic segmentation is SegNet, which is depicted in Figure 2.3. As no fully connected layers are used anymore, such networks can be said to be fully convolutional neural networks (FCN).

Regarding the processing of LIDAR scans, no standard type of architecture, that is as widely used as CNNs and FCNs, currently exists. Several approaches exist however. The machine learning algorithms that are able to process raw LIDAR scans can be split into two main categories. On the one hand, some approaches consider LIDAR scans as unorganized point-clouds, and usually rely on PointNet-like architectures [13]. On the other hand, other approaches are heavily inspired by SqueezeSeg [14], and consider that LIDAR scans are orga-

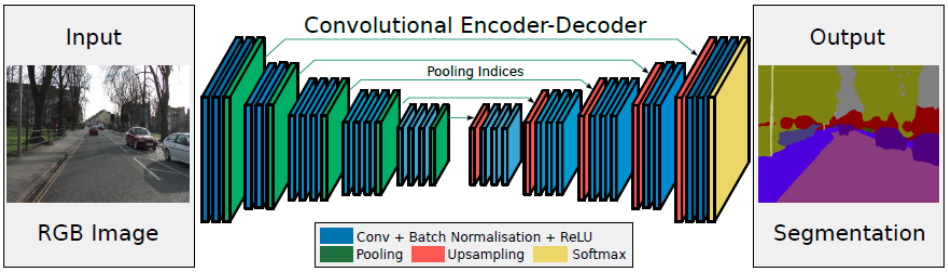


Figure 2.3: SegNet, an encoder-decoder convolutional neural network, for semantic segmentation. Illustration originally coming from [24].

nized point-clouds, which can be processed as range images obtained by spherical projections.

2.3.3 Deep neural networks for LIDAR point-cloud processing

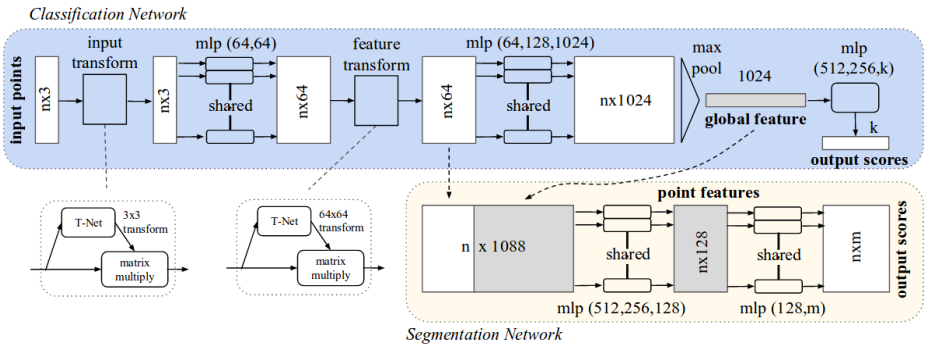


Figure 2.4: PointNet, a general neural network architecture to process unorganized point-clouds. Illustration originally coming from [13].

PointNet applies a common multi-layer perceptron to the features of each point of a point-cloud. A global max-pooling operation is then used to extract a cloud-level feature. Such an architecture was mathematically demonstrated to be able to approximate any set function. Further layers can then be trained to perform for object classification, semantic segmentation, or part segmentation. However, PointNet expects normalized, and relatively constrained inputs, which involves several pre-processing steps. Engelmann et. al. for instance investigated, without reaching fully satisfactory results, several PointNet-like architectures for semantic segmentation of large-scale virtual 3D point-clouds, which relied on the parallel processing of subsets of virtual LIDAR scans [25]. The general architecture of PointNet is depicted in Figure 2.4. Recent evolutions of PointNet, such as PointNet++ [26], aim at extracting information from local neighborhoods of points, to extract more refined features from the input point-cloud.

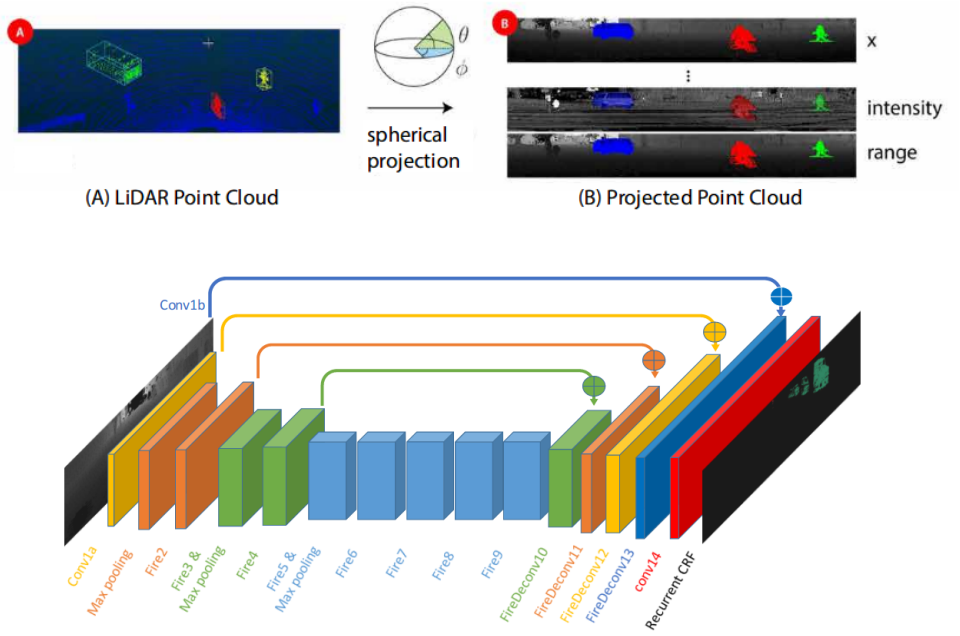


Figure 2.5: SqueezeSeg, a convolutional architecture to process LIDAR scans. Illustration originally from [14]

SqueezeSeg [14], on the other hand, was introduced as a refined version of SqueezeNet [27], a highly efficient convolutional neural network with a limited number of parameters, and a low memory footprint. SqueezeNet heavily relies on Fire layers, that require less computations and use less parameters than traditional convolution layers. SqueezeSeg re-adapts SqueezeNet for semantic segmentation of LIDAR scans, by considering that LIDAR scans are organized point-clouds, which can be represented by range images. Spherical projection can indeed be used to generate dense range images from LIDAR scans. The height of those range images corresponds to the number of lasers that are used by the sensor, and their width is equal to the field-of-view of the sensor divided by its horizontal resolution. The number of channels correspond to the number of features available for each point. This means that the network architecture depends on the specifications of the LIDAR sensor. A conditional random field (CRF), reinterpreted as a recurrent neural network, was also trained alongside SqueezeSeg, to further improve the segmentation results. Recently, SqueezeSegV2 introduced a context aggregation module, to cope with the fact that LIDAR scans usually include missing points, due to the limited range of the sensor, and sensor noise. The input range image used in SqueezeSegV2 also includes an additional channel that indicates whether a valid sensor reading was available, at each angular position. Finally, SqueezeSegV2 extensively uses batch-normalization, contrary to SqueezeSeg. Other refinements regarding the training of the system, and fine-tuning from simulated data, were also proposed.

Both SqueezeSeg-like and PointNet-like architectures then suffer from drawbacks, when used to process LIDAR data. On the one hand, SqueezeSeg-like architectures are designed to process inputs that heavily depend on the specifications of the LIDAR scans that generated them, and it is thus doubtful that they generalize well for any type of LIDAR sensor. PointNet-like networks are mainly designed for constrained outputs, which may require to pre-proces their input LIDAR scan, and struggle to extract local information. However, several new architectures are being proposed, to deal with those drawbacks. KPConv for instance relies on kernel functions that are similar to convolution kernels, but are designed to process neighborhoods of points [28]. This approach currently reaches state-of-the-art results for semantic segmentation on the Paris-Lille-3D dataset, by consecutively processing subsets of the registered point-cloud and using a voting scheme to select the final class of each point. Such a framework seems incompatible with real-time perception, but the original idea behind KPConv might be reused in specific systems that are tailored to process LIDAR scans. On the other hand, similar architectural refinements might also be used in SqueezeSeg-like networks, that process organized LIDAR point-clouds. After having briefly introduced the most common machine-learning architectures to process RGB images and LIDAR scans, we will, in the next section, present how such architectures can be used for road detection, semantic segmentation and object classification.

2.4 Road detection via machine learning, from RGB images and LIDAR scans

The approaches that we will present now were all developed and evaluated from the KITTI road dataset, as this is currently the only dataset in which the road detection task is evaluated individually. State-of-the-art approaches on the KITTI dataset can then benefit from either RGB images, LIDAR scans, or both. First, we will focus on image-only approaches.

With a MaxF-score of 94.88% on the test set of the KITTI road dataset, MultiNet [29] is the third best performing image-only road detection approach on the KITTI road dataset. A CNN encoder, that was previously pretrained on the ImageNet dataset, is used to extract features from the input RGB image. This encoder generates a feature map of 39x12 pixels. Then, three decoders are used to perform jointly three tasks from the encoded features: road detection, street type classification and vehicle detection. As these three types of labels are not available together for the images of the KITTI road dataset, MultiNet is alternatively trained on one of the three tasks, and thus relies on data that does not belong to the KITTI road dataset for its optimization. The road detection task is performed in a fully convolutional fashion, with clasification scores evaluated for each pixels ; the street type classification task is done via fully-connected layers that output a probability score, which indicates whether the RGB scene

depicts a minor road or a highway ; the detection task is done by predicting, for each pixel of the encoded features, a confidence score indicating the presence of a vehicle at this position, and the coordinates of a bounding box in the area around each pixel of the encoded feature map. Those predictions are then upsampled, to fit the original size of the input image. By aiming at performing several tasks at once, MultiNet is expected to encode general features from RGB images, which is expected to facilitate the training. RBNet [30], the second best performing approach for road detection on the KITTI road dataset also relies on multi-tasking. RBNet only relies on the data from the KITTI-road dataset, and works in a fully convolutional fashion. A pretrained encoder is again used to generate the input to two decoders. One performs pixel-level road detection, and the other one performs pixel-level road-boundary detection. The results are refined with regards to the estimated road boundaries, as pixels that are classified as belonging to the road, but outside the detected road boundaries, are considered as false positives. RBNet reaches a Max-F score of 94.97% on the KITTI road test set. Finally, the best image-only approach for road detection on the KITTI road dataset was proposed by Han et. al. in [31]. Again, a multi-task fully convolutional network is used for multi-tasking. A decoder branch predicts per-pixel road classification scores, and another one classifies the road depicted in the scene as either *straight*, *branch*, or *curve*. However, inspired by GANs [32], Han et. al. also propose to train a discriminative network, which must detect whether the road detection and classification system processes labeled data from the KITTI road dataset, or unlabeled data. This way, the system can benefit from unlabeled data during the training. Yet, it thus relies on additional training data, and additional road type labels that Han et. al. had to add themselves. Yet, this system reaches a Max-F score of 95.53% on the KITTI test set.

Other approaches on the KITTI-road dataset only rely on LIDAR scan to perform road detection, but actually rely on image processing techniques. Fernandes et. al. [33] proposed to project and upsample LIDAR points into a 2D image plane, and to detect the road in this image plane via an histogram similarity measure. They reach a Max-F score of 82.72%. Lyu et. al. [34] proposed to train a neural network on range images obtained from the spherical projection of LIDAR scans, similarly to what is done for SqueezeSeg, and to fit a polygon on the predicted road points to obtain a dense prediction. In this work, some LIDAR points are lost on the projection process, as each pixel of the range image could represent several LIDAR points. A Max-F score of 94.05% is reached by this method. Finally, another proposition from Caltagirone et. al. was to project LIDAR points into a 2D sparse feature grid corresponding to a bird's eye view, and to train a convolutional neural network to predict a dense road region from this representation [35]. This method reached a Max-F score of 94.07%.

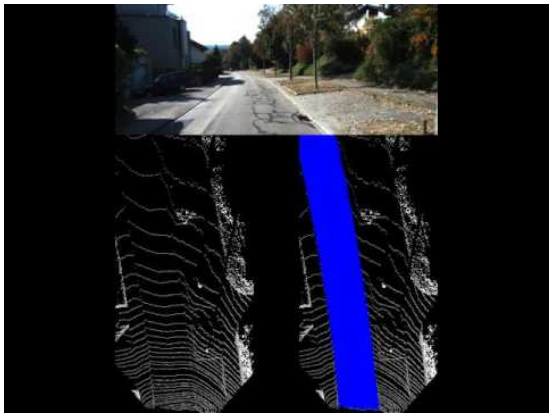


Figure 2.6: Example of road detection result obtained by [35]. The top RGB image depicts the situation, the bottom-left image depicts the LIDAR grid used as input to the neural network, and the bottom˙right presents the corresponding road detection.

Although those approaches are currently the best performing LIDAR-only road-detection approaches on the KITTI dataset, they share the same limitation: they aim at predicting a dense road area from a sparse LIDAR scan, and thus rely on upsampling, since they are evaluated with regards to dense, image-level road labels. All those approaches then predict the presence of road on locations where no actual LIDAR measurements were actually available, which can be considered as an incorrect behavior for a LIDAR-only road detection algorithm. This is for instance the case in Figure 2.6, which presents a result of Caltagirone et. al. Indeed, gaps or small obstacles could be present but remain unobserved due to the sparsity of LIDAR scans. Moreover, due to the limitations of the KITTI dataset, in which the road is only labeled in the front camera view, those systems are not designed to detect the road at 360°, and might require to be significantly modified to detect the road on full LIDAR scans, without any guarantee of success.

The best performing approaches for road detection on the KITTI-road dataset actually benefit from both LIDAR scans and RGB images. Gu et. al. [36] propose to perform a purely geometrical road detection in LIDAR scans, to upsample the results and to project them in the image plane. Road detection in the RGB image can then be performed via an FCN, and the two results can then be fused via a conditional random field. This approach reaches a Max-F score of 95.68%. Another possible approach is to use an FCN to process both LIDAR scans and RGB images. In [37], a two-stream network processes an upsampled LIDAR scan that was projected into the image plane, and the corresponding RGB image. Intermediate fusion of the encoded features is performed by performing an element-wise weighted sum between intermediate LIDAR features and intermediate RGB features. These weights are learnt alongside the other parameters of the network. This approach reaches a Max-F score of 96.03%. PLARD [38], the current best performing approach for road detection on the KITTI road dataset, is conceptually close to this previous work. Indeed, instead of directly learning the weights for the intermediate weighted sum of features, PLARD relies on convolution layers applied to the concatenated RGB and LIDAR features to predict both the weight for the weighted sum, and an additional bias added to the resulting summed feature. LIDAR scans are projected into the image plane,

but each pixel correspond to an altitude gradient, instead of full coordinates. This is justified by the fact that the road typically belong to the ground, and is relatively flat. PLARD reaches a Max-F score of 97.03% on the KITTI road dataset.

Road detection from deep neural networks, on the KITTI dataset, relies on specific architectures that are tailored for this specific task. In order to improve the results, it is common to rely on multi-task approaches, and additional data coming from several sensors, or unlabeled dataset. The task of semantic segmentation, which is more general than road detection, is thus likely to rely on less specific architectures.

2.5 Semantic segmentation of road scenes

According to the leaderboards of the CityScapes dataset, the task of semantic segmentation of RGB images seems to have reached a certain level of maturity. Indeed, the three best performing approaches on CityScapes, which all rely on the use of fully convolutional architectures, have very similar results. Li et. al. proposed GALD, a network relying on global aggregation and local distribution, to model pixel-to-pixel relations [39]. Global aggregation relies on long-range operators, such as Average Pooling, to enforce a consistent segmentation of large objects. Yet, this process tends to over-smooth small objects, which is addressed via local distribution. The local distribution module predicts, in an unsupervised fashion, object masks that are used by the network to focus on features corresponding to objects of interest. GALD reaches a mean Intersection-over-Union score (mIoU) of 83.3 on CityScapes. The second best performing approach on CityScapes was proposed by Zhu et. al. [40]. As video sequences are available in CityScapes, while only specific frames have been labeled, a video reconstruction network is used to propagate manual labels to other frames of the raw original video recording. Additional coarse training data is then constructed. To account for the fact that those reconstructed labels might be imperfect, the penalization of the network, when it predicts wrong and conflicting classes among neighboring pixels, is reduced. This work reaches a 83.5 mIoU on the CityScapes dataset. The currently best performing approach on CityScapes was proposed by Yuan et. al [41]. Their fully convolutional architectures aims at extracting global features from intermediate classification results: a first segmentation of the image is generated, and all the pixels that were classified as belonging to the same class, in the image, are used to build a class-wise feature. This additional class-level information is later used to refine the intermediate classification results by further convolutions. This work reaches a 83.5 mIoU score on the CityScapes dataset.

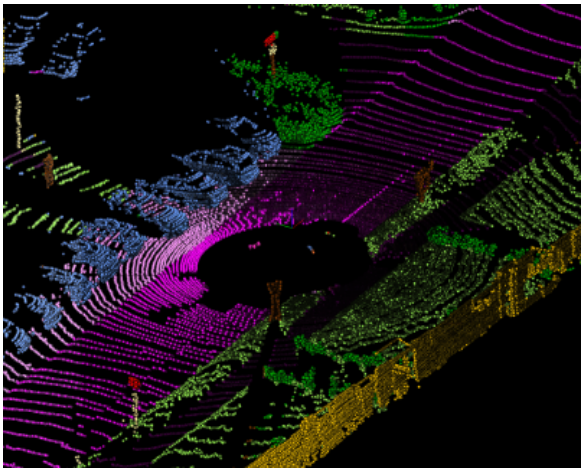


Figure 2.7: Example of semantic segmentation result obtained from RangeNet++ [42].

Semantic segmentation of LIDAR scans for autonomous driving is, comparatively to RGB images, not often addressed, due to the lack of dedicated datasets. The currently best performing approaches are actually SqueezeSeg [14], and its variants. Originally, SqueezeSeg was designed to perform semantic segmentation of road users, from the KITTI dataset. Indeed, 3D bounding box labels corresponding to cars, cyclists and pedestrians are available in the KITTI-object dataset. However, those boxes only correspond to objects that are visible in RGB images. The 3D position was manually inferred from the LIDAR scans, but some objects that are visible in the RGB images are difficult to distinguish in the LIDAR scan, and only actually correspond to a few LIDAR points. Yet, SqueezeSeg was trained on LIDAR scans with point-level labels, that were derived from those bounding boxes. Points that were enclosed within a bounding box were labeled according to the class of the bounding box, with thus a risk of mislabeling LIDAR points due to calibration and synchronization errors. SqueezeSegV2 [43] introduced many refinements, the biggest one being that the network was also trained on synthetic LIDAR data rendered from a video-game. The release of the SemanticKITTI dataset will clearly boost the research efforts towards semantic segmentation of LIDAR scans for autonomous driving. RangeNet++ [42], the baseline method proposed by the authors of the SemanticKITTI dataset, relies on the same idea as SqueezeSeg, and processes LIDAR scans that were organized in range images via spherical projection. Yet, RangeNet++ uses a network that is significantly deeper that SqueezeSeg and SqueezeSegV2, and a post-processing step relying on an optimized KNN algorithm, to correct the predictions of the network. The comparisons made by the authors of RangeNet++ indicate that it vastly outperforms both SqueezeSeg and SqueezeSegV2, but maintains a framerate that is four times lower than SqueezeSegV2 without its CRF layer. As SqueezeSegV2 was initially designed to detect only four classes, while SemanticKITTI includes labels for 19 classes, this result is not counterintuitive. To our knowledge, no recent machine-learning approach, that performs semantic segmentation from both RGB images and LIDAR scans for autonomous driving, was proposed. Again, the release of the SemanticKITTI dataset will probably trigger various research works in this direction.

2.6 3D LIDAR Object detection and classification

Semantic segmentation is probably a paramount task in the context of autonomous driving. However, to track and anticipate the behavior of individual road users, an object-level detection and classification is mandatory. A possible pipeline for such an object-level reasoning would be to, first, detect the objects, and then classify them. Object detection in dense LIDAR scans usually relies on two-step geometrical approaches, which first remove ground points, and then cluster the remaining points. Classifiers can then be used to train to classify each type of object [44]–[46]. In [44], a Naive Bayes classifier is for instance used to classify object tracks as pedestrian, bicyclist, motor bike, passenger car, van or truck. Yet, this assumes that the classified track belongs to one of those classes. The original baseline method proposed to demonstrate the interest of the TUBS dataset [16] was actually an object classification algorithm. LIDAR points that are enclosed into the ground truth boxes are projected into dense images via spherical projection, as done for SqueezeSeg, and a convolutional architecture is used to predict the class of the objects. In both cases, those systems are dependent on the correct operation of the other perception algorithms (clustering for [16], clustering and tracking for [44]).

A more popular trend is to jointly perform the object detection and classification tasks, by training a system to extract objects of interests. Simon et. al. [47] thus proposed to perform bounding box regression and multi-class classification jointly by training an image-domain object detector on feature grids generated from a LIDAR scan. The object detector relies on a convolutional architecture that outputs bonding box dimension and class scores, similarly to what was done in MultiNet. Albeit this approach runs at a high framerate, its general performances are significantly worse than pure LIDAR-domain classifiers, such as SECOND. SECOND [48] splits a LIDAR scan into equally-sized voxels, and applies a PointNet network to the points enclosed in each voxel. Additional convolutions followed by a region proposal network are then used to predict bounding box dimensions and an objectness score for each voxel. However, this approach is computationally challenging. Indeed, the bounding box parameters and regression scores are only calculated for a single class, and different models

Figure 2.8: Example of object detection result result obtained by Simon et. al. [47].

are needed for each class. A similar concept was used in PointPillars [49]. Pillars of LIDAR points by discretizing the x-y plane into an evenly spaced grid. Points that can be projected into an common grid cell are considered to form a pillar. A simplified PointNet is then used to extract a feature vector describing each pillar. Finally, a simple convolutional architecture can be used to perform bounding box regression and classification on the resulting feature map. PointPillars reaches a 0.305 mean Average Precision score (mAP) on the NuScenes dataset. The current best performing approach on NuScenes, for LIDAR object detection and classification, was proposed by Zhu et. al [50]. They rely on sparse 3D convolutions to extract features from the LIDAR scans, and further use convolutional layers for the bounding box regression and classification tasks. However, they observed that the NuScenes dataset is particularly imbalanced, as cars in NuScenes represent more than 43% of the objects, while bicycles represent less than 3% of the labeled objects. To facilitate the training, object classes are thus merged into 6 groups that are less imbalanced and depend on the object dimensions, and the network aims at predicting first the group of each object, and then the actual class. This work reaches a mAP score of 0.528 on the NuScenes dataset.

2.7 Conclusion

In conclusion, we can highlight the fact that machine-learning in general, and deep learning in particular, are now particularly common to address perception tasks for autonomous driving, with state-of-the-art results. Semantic segmentation is especially efficiently addressed by many approaches that have similar results, in terms of pixel-level accuracy. 2019 was especially a significant year for the perception community, as many new datasets have been released, which will facilitate the work of many researchers. However, the availability of such datasets, when addressing the tasks of perception for autonomous driving, can also have pernicious effects. Indeed, the research works are oriented towards specific sensor setups, problems and use cases. For instance, PLARD assumes that the ground is mostly flat, and benefits from the fact that the KITTI dataset focuses on broad and flat roads. It is doubtful whether this system could be used in complex urban environment. Moreover, exhaustively testing new approaches on all the possible dataset will probably be extremely time consuming for future researchers. This would at least have the interest of confirming that approaches that reach satisfactory results, on all the possible datasets, can be considered as efficient in the general case. Using these systems in actual self-driving vehicles will however require efficient software architectures and fusion frameworks to be defined. For that reason, we propose to rely on the theory of belief functions, or evidence theory.

Chapter 3

The evidence theory, and its applications in autonomous driving

Contents

3.1 Introduction

The evidence theory has been originally proposed by Dempster [51], and further refined by Shafer [52] and Smets [53]. By generalizing the probability theory, it allows for finer representations of the available information. In the following section, we will first introduce the general concept of this theory, and the main mathematical tools used in this framework. We will then focus describing how this theory can be used, mainly in grid mapping frameworks, to solve perception tasks for autonomous driving.

3.2 The evidential framework

Let $\Theta = \{\theta_1, ..., \theta_n\}$ be a finite set of all the possible answers to a question. An evidential mass function m is a mapping $m : 2^\Theta \to [0, 1]$ such that $m(\emptyset) = 0$ and

$$\sum_{A \subseteq \Theta} m(A) = 1 \tag{3.1}$$

In the binary case, $n = 2$, and $2^\Theta = \{\emptyset, \theta_1, \theta_2, \Theta\}$. Then, $m(\theta_1)$ represent the amount of evidence towards the fact that the answer is θ_1, and $m(\Theta)$ is the evidence towards the fact that nothing can be said about the answer (i.e. it is unknown). It must be noted that, in this context, $m(\Theta)$ does not support neither θ_1 nor θ_2. If $m(\theta_i) > 0$, θ_i is said to be a focal element, or focal set, of m.

As it is imposed that $m(\emptyset) = 0$, $\emptyset$ cannot be a focal element. However, if this constraint is relaxed, and $m(\emptyset) > 0$, m is said to be *subnormal*. Then, $m(\emptyset)$ can be interpreted as evidence towards the fact that Θ is not exhaustive. If Θ is however considered to be exhaustive, m can be transformed into a *normal* mass function, via Dempster's normalization:

$$K = m(\emptyset) \tag{3.2}$$

$$m(\emptyset) = 0 \tag{3.3}$$

$$m(A) = \frac{m(A)}{1-K}, \forall A \subset 2^{\Theta} \setminus \{\emptyset\} \tag{3.4}$$

In this case, K is the *degree* of conflict.

Two evidential mass functions m_1 and m_2, that are produced by independent sources of information, and share the same frame of discernment that is considered to be exhaustive, can be fused into a new joint mass $m_{1,2} = m_1 \oplus m_2$ via Dempster's rule of combination, as follows:

$$K = \sum_{B \cap C = \emptyset} m_1(B) m_2(C) \tag{3.5}$$

$$m_{1,2}(A) = \frac{1}{1-K} \sum_{B \cap C = A, A \neq \emptyset} m_1(B) m_2(C) \tag{3.6}$$

Otherwise, if the frame of discernment is not exhaustive, the unnormalized Dempster's rule of combination [54] can be used instead:

$$m_{1,2}(A) = \sum_{B \cap C = A} m_1(B) m_2(C) \tag{3.7}$$

An evidential mass function m is *simple* if:

$$\exists A \subset \Theta, m(A) = s, m(\Theta) = 1 - s \tag{3.8}$$

Let $w := -ln(1-s)$ be the *weight of evidence* associated with m. In this case, m can be represented as $\{A\}^w$. Then $\{A\}^{w_1} \oplus \{A\}^{w_2} = \{A\}^{w1+w2}$.

A commonality function Q, associated to a mass function m, can be defined as follows:

$$Q(A) = \sum_{B \supseteq A} m(B), \forall A \subseteq \Theta \tag{3.9}$$

The evidential mass function m can be recovered from the commonality values, as follows:

$$m(A) = \sum_{B \supseteq A} (-1)^{|B|-|A|} Q(B) \tag{3.10}$$

The commonality function of l mass functions to be fused via Dempster's rule of combination can be combined as follows:

$$(Q_1 \oplus ... \oplus Q_l)(A) = \prod_{j=1}^{l} Q_j(A), \forall A \subseteq \Theta \tag{3.11}$$

In particular, the fused mass function can then be obtained from the fused commonalities, and Equation 3.10.

Belief and plausibility functions are defined, respectively, as

$$bel(A) = \sum_{\emptyset \neq B \subseteq A} m(B) \tag{3.12}$$

$$pl(A) = \sum_{B \cap A \neq \emptyset} m(B) \tag{3.13}$$

An evidential mass function can be transformed into a probability distribution. Originally, this was done via the *pignistic transformation* [53]:

$$BetP(\theta_i) = \sum_{A \subseteq \Theta, \theta_i \in A} \frac{m(A)}{|A|} \tag{3.14}$$

This expression was justified by the fact that Dutch Books had to be prevented, when reasoning on probability distributions obtained from evidential mass functions. A Ducth Book happens, in gambling, when incoherent odds lead to a systematic win, or loss. It was initially considered in [55] that, to ensure the absence of Dutch Books, the transformation used to convert evidential mass functions into probability distributions had to be linear. Under this assumption, the only possible transformation is the pignistic transformation.

However, several criticisms have been made regarding this assumption, and this expression. First, a counter-example in which a Dutch Book happens when reasoning on the pignistic transformation was proposed in [56]. Then, it was observed in [57] that the pignistic transformation can be inconsistent with Dempster's rule of combination: use cases can be constructed where fusing several evidential mass functions would increase the support a specific element of the frame of discernment, while the pignistic probability of this element would be decreased after fusion.

To replace the pignistic transformation, the *plausibility transformation* was proposed. Let Pl_P be the probability distribution obtained by plausibility transformation, or plausibility probability function. This distribution is defined as follows:

$$Pl\_P(\theta_i) = \frac{pl(\theta_i)}{\sum_{\theta_j \in \Theta} pl(\theta_j)} \tag{3.15}$$

Case	$m(\theta_1)$	$m(\theta_2)$	$m(\Theta)$	$Pl\_P(\theta_1)$	$Pl\_P(\theta_2)$
1	0.8	0.2	0	0.8	0.2
2	0.78	0.12	0.1	0.8	0.2
3	0.76	0.04	0.2	0.8	0.2

Table 3.1: Example of different evidential mass functions having equal plausibility probability functions.

In particular, this expression is extremely interesting because it has a direct link with Bayesian fusion. Indeed, let $\otimes$ depicts the independent opinion poll fusion of probability distributions. Then, if $m = m_1 \oplus ... \oplus m_l$ is a joint evidential mass function obtained by Dempster's combination rule, and P_m is the corresponding plausibility probability function, then $P_m = P_1 \otimes ... \otimes P_l$.

The plausibility transformation shows that the evidence theory is more expressive than simple probability distributions, as several evidential mass functions can lead to similar plausibility probability functions. We explicit this phenomenon on a binary frame of discernment in Table 3.1, with three different evidential mass functions that have the equal plausibility probability functions. In those example, we see that for the case 1, no ambiguity about the class is present, as $m(\Theta)$ is equal to 0. Yet, in the second case, we see that the support for Θ is not neglectable, indicating a certain level of doubt on both θ_1 and θ_2. Finally, the case 3 depicts a case where θ_2 has an extremely low support, while all the available knowledge does not support θ_1. Now that the basic concepts and ideas behind the evidence theory have been introduced, we can briefly present how it is used top solve perceptions tasks for autonomous driving.

3.3 Use of the evidence theory in perception tasks for autonomous driving

The evidence theory can potentially be used for any perception, with proper modeling. It was for instance used by Wang et. al [58] for stereo-matching of images from the KITTI dataset. The evidence that supports each possible disparity value for a pixel, that is considered to belong to a finite set of values, is modeled by normalizing the cost values associated to each possible location. In particular, the evidence on the frame of discernment is used to model the fact that a given pixel does not have a disparity, which can actually be caused, for instance, by occlusions. Another example of use of the evidence theory was proposed by Magnier et. al. [59], for RADAR object association and tracking. At each time step, the distance between each detected RADAR object, and pre-existing object tracks, is evaluated. From this distance, evidential mass functions can be defined, such that the frame of discernment represents the set of existing tracks to which each object could be associated. The mass on the full frame of discernment is

considered as an uncertainty indicator for each possible association, and the mass on $\emptyset$ as an indicator for the need of new tracks.

However, the evidence theory is mainly used, in robotics and autonomous driving, for occupancy grid mapping, especially from LIDAR scans. Evidential and Bayesian occupancy grids currently coexist. Bayesian grids model the occupancy probability of each cell, while evidential occupancy grids use a frame of discernment corresponding to the fact of being occupied, for a given cell. Evidential grids tend to better represent unobserved areas (which correspond to evidence on the complete frame of discernment), and conflicting information. Indeed, Rummelhard et. al. [60] proposed a Bayesian grid mapping framework, in which the displacement of each potentially moving cell is estimated via a particle filtering approach. Yet, to initialize the filter and create particles, a difference between unobserved and observed cells has to be made. The solution that was chosen was to track the state changes of each cells, and update the occupancy probabilities according to ad hoc transition parameters, as soon as a first observation is made. This use of ad hoc transitions for each cell makes the system particularly sensitive to false positives, while the evidence theory can represent unobserved cells simply via the mass on the frame of discernment. In this case, the explicit transition parameters, and tracking of the state of each cell over time, are not needed anymore.

The first evidential occupancy grid mapping system that relied on LIDAR sensors was proposed in [61]. Evidential mass functions were constructed at the cell level from several stationary-beam LIDAR sensors, via ray tracing and ad-hoc false alarm and missed detection rates. The use of the evidential framework was shown to better represent uncertainty and the fact of not knowing, when compared to a traditional Bayesian fusion scheme. The Caroline vehicle already used evidential occupancy grid maps during the 2007 DARPA Urban Challenge [62], but the reasoning was done globally from both LIDAR scans and point-clouds generated by stereo-vision, without considering the specificity of LIDAR sensors.

It was proposed in [63] to extend the original work from Yang et. al. to a four-layer LIDAR scanner. Based on a proposal from [64], a discount factor was applied to an evidential polar grid, before fusing it with new sensor observations. Evidential mass functions were again generated from ray tracing, and ad-hoc false alarm and missed detection rates. Yu et. al. generalized this model to a more complex 360°Velodyne HDL-64 LIDAR scanner [65]. A first ground-detection step, based on a simple thresholding, is used to classify LIDAR points as either belonging to an obstacle or to the drivable area. Then, evidential mass functions were created at the cell level, from the classification results and from, again, ad-hoc false-alarm and missed-detection rates.

More recent works aim at tackling some intrinsic limitations of LIDAR-based evidential occupancy grid mapping. Nuss et. al. proposed to couple an evidential

model with particle-filtering, in order to estimate the speed of each grid cell, and detect moving objects [66] from a four-layer LIDAR scan. This comes at the cost of various hyper-parameters to manually tune, and a computational complexity, as virtually any occupied cell could be associated with a set of particles. Another recent work aims at predicting dense evidential occupancy grid maps from individual LIDAR scans [67]. Consecutive LIDAR scans are registered, and dense grids are obtained from the resulting point-cloud thanks to a ground removal step, and manually defined false-alarm and missed detection rates. A convolutional neural network is then trained to recreate the dense evidential grids from a feature grid generated from only one of the original scans.

Given the observations that the evidential theory is particularly suitable for occupancy grid mapping, an obvious way to use machine learning approaches in an evidential framework would be to fuse outputs from neural networks into a grid mapping system. In the following section, we thus propose a simple, asynchronous occupancy mapping algorithm, that fuses semantic segmentation results obtained from RGB images, and LIDAR grids obtained from a geometrical model.

Chapter 4

Asynchronous evidential grid mapping from RGB images and LIDAR scans

Contents

4.1 Introduction

We propose an evidential fusion algorithm between LIDAR scans and RGB images. LIDAR points are classified as either belonging to the ground, or not, and RGB images are processed by a state-of-the-art convolutional neural network to obtain semantic labels. The results are fused into an evidential grid to assess the drivability of an area met by an autonomous vehicle, while accounting for incoherences over time and between sensors. The dynamic behavior of potentially moving objects can be estimated from the high-level semantic labels. LIDAR scans and images are not assumed to be acquired at the same time, making the proposed grid mapping algorithm asynchronous. This approach is justified by the need for coping with, at the same time, sensor uncertainties, incoherences of results over time and between sensors, and the need for handling sensor failure. In classical LIDAR/camera fusion, in which LIDAR scans and images are considered to be acquired at the same time (synchronously), the failure of a single

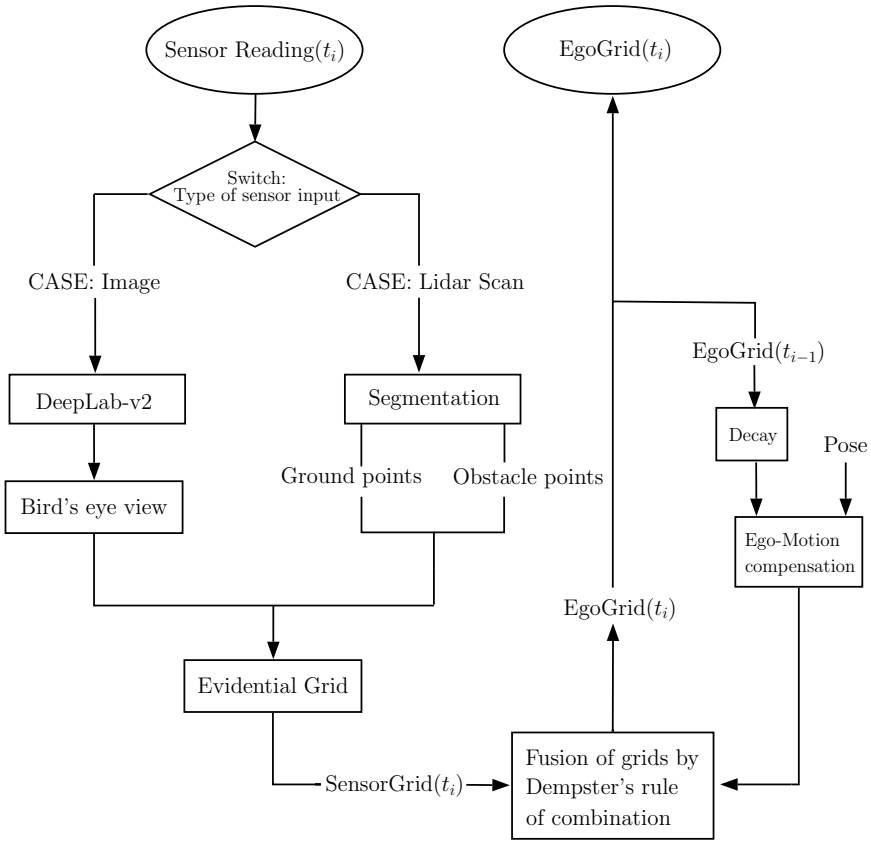


Figure 4.1: Asynchronous fusion based algorithm. Images and LIDAR scans are individually processed once acquired. An evidential grid is then built from the sensor input, and fused with the current global grid, or EgoGrid, based on the current vehicle pose. Thus, no LIDAR/camera synchronization is needed.

sensor leads to the failure of the whole fusion algorithm. On the contrary, the proposed asynchronous approach can be used to fuse contradictory information over time, while allowing the vehicle to operate even in the event of the failure of a single sensor. Experiments on a challenging use case highlight the interest of the method. The general grid mapping system is depicted in Figure 4.1. In order to generate the final evidential grid, individual grids for each sensor reading have first to be computed, and then fused. The final goal is to know, at every moment, whether a location in the perceived environment, i.e. a cell of a grid, can be passed through by an autonomous vehicle. A corresponding frame of discernment Ω was thus defined as $\{D, \neg D\}$, where the two singletons D and $\neg D$ are propositions respectively indicating that a cell is either drivable or non-drivable. It is then possible to derive $2^{|\Omega|}$ subsets from Ω, the set of which form the power set $2^{\Omega} = \{\emptyset, D, \neg D, \Omega\}$. Each singleton of the power set is a proposition. The empty set $\emptyset$ indicates that the cell is not in a state that corresponds to the model, and Ω indicates that the state of the cell is unknown. This explicit quantification of ignorance is one of the specificities of evidential grids.

4.2 From raw LIDAR scans and images to evidential grids

Before being able to use the fusion scheme described previously, evidential grids have to be generated. The following section describes how such grids can be produced from LIDAR scans and images, and how they are fused.

4.2.1 Generating evidential grids from LIDAR scans

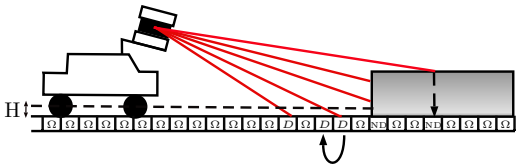


Figure 4.2: Evidential mapping from a LIDAR scan. The state having the largest mass is reported for each cell. The curved arrow indicates the occurrence of backward propagation.

To convert LIDAR scans into evidential grids, an approach inspired by Yu et al.'s polar grid mapping [65] is proposed, to create evidential scan grids in a Cartesian coordinate system. The resulting grids can then be fused over time, based on the odometry of the vehicle. Fig. 4.2 illustrates how evidential grids are built from LIDAR scans. When converting a scan to an evidential grid, drivable areas are those where only points corresponding to the ground are detected, and non-drivable ones are those where an obstacle is detected. The ground is assumed to be flat. Provided that the altitude and rotation of the LIDAR relatively to the ground is known, each point can be projected on this ground plane. This plane is then divided into regular cells to build an evidential grid. To quantify the amount of proof associated with each state in every cell, false alarm and missed-detection rates are defined. A *false alarm* happens if it is wrongly considered, due to sensor noise for instance, that an obstacle is present. On the opposite, a *missed detection* happens when an obstacle that is actually present is not detected by the LIDAR, often because of its size or reflectivity. Let α_{FA} be the false alarm rate in a given cell, n_o the number of points that hit this cell and are classified as obstacles, n_g the number of points that hit the cell and are classified as ground points, and $\alpha_{MD}(n_g)$ the corresponding missed detection rate. For each cell, the corresponding basic belief assignment (BBA), denoted as m_l, is computed as follows:

$$m_l(\emptyset) = 0 \tag{4.1}$$

If no point has hit the cell:

$$m_l(D) = 0, m_l(\neg D) = 0, m_l(\Omega) = 1 \tag{4.2}$$

If all the points that hit the cell are classified as ground:

$$m_l(D) = 1 - \alpha_{MD}(n_g), m_l(\neg D) = 0, m_l(\Omega) = \alpha_{MD}(n_g) \tag{4.3}$$

If at least one point that hit the cell is classified as obstacle:

$$m_l(D) = 0, m_l(\neg D) = 1 - \alpha_{FA}^{n_o}, m_l(\Omega) = \alpha_{FA}^{n_o} \tag{4.4}$$

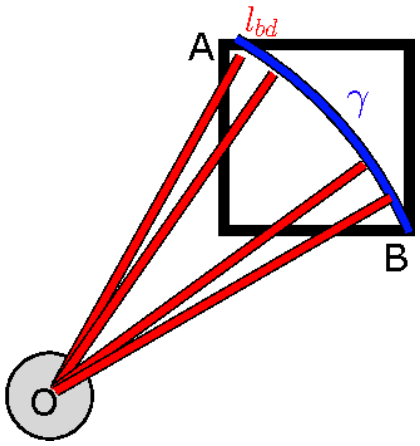


Figure 4.3: Estimating the maximum number of ground points in a grid cell.

On the one hand, the false alarm rate is considered to be the same for every cell. On the other hand, an unique missed detection rate is computed for every particular cell, as the number of laser impacts that can intercept a regular cell depends on its position. Figure 4.3 summarizes the geometrical model that is used to estimate the maximum number of ground points that hit a cell, based on which we estimate missed detection rates. To compute the missed detection rate of a given cell, the maximum number of laser impacts that can occur in this area is computed, and compared with the actual number of points classified as ground points that hit the cell. Given the beam divergence and the horizontal angular resolution of a LIDAR, this maximum number of hits can be deduced from the maximum angle between two points belonging to the cell. For the sake of simplicity and computational workload, the maximum angle is considered to be, for every cell, the maximum angle formed by opposite corners of the cell, named A and B, and the origin of the grid, i.e. the origin of the LIDAR projected on the ground plane, named O. Let o be the size of the diagonal of a cell, b the distance between O and A, and a the distance between O and B. This maximum angle γ can be computed from the law of cosines, as follow:

$$\gamma = \cos^{-1}\left(\frac{o^2 - a^2 - b^2}{-2ab}\right) \tag{4.5}$$

Let l_{bd} be the beam divergence of a LIDAR. The LIDAR is assumed to be in a position such that any cell belonging to the ground can only be hit by a single

rotating laser. Thus, for every cell, the missed detection rate can be estimated as follow:

$$\alpha_{MD}(n_g) = 1 - \frac{n_g \cdot l_{bd}}{\gamma} \tag{4.6}$$

Finally, backward extrapolation, given a maximum threshold H, is performed: masses of cells classified as drivable are propagated to cells where nothing is detected, but obstacles taller than H cannot be present. Figure 4.4 presents an example of evidential grid obtained from a LIDAR scan.

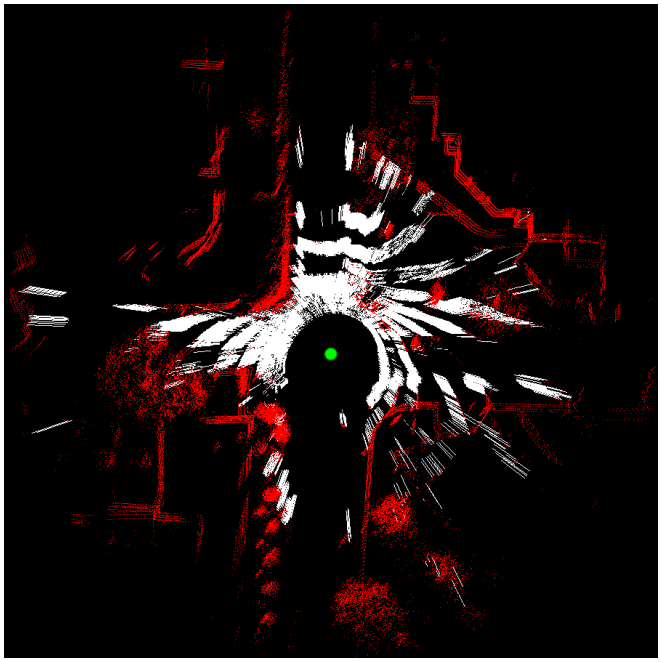


Figure 4.4: Example of evidential grid obtained from a LIDAR scan. The green point indicates the position of the LIDAR sensor. White cells have a mass value for D higher than 0.5 ; red cells have a mass value for $\neg D$ higher than 0.5; black cells have a mass value for Ω higher than, or equal, to 0.5.

4.2.2 Generating evidential grids from segmented images

Fusing evidential grids generated from a LIDAR and pixel-wise segmentation results requires the later to be converted into an evidential grid. Contrary to the work in [68], stereo-vision is not used to estimate the depth of observed objects. Instead, the segmentation results generated from a mono-camera can be projected into a bird's eye view, corresponding to the ground plane in the LIDAR's coordinate system, to generate an evidential grid. The popular pinhole camera model is used to represent the behavior of an undistorted camera. Let $K \in \mathbb{R}^{3\times3}$ be the camera intrinsic matrix. The extrinsic matrix corresponding to the transformation between the camera's coordinate system and the LIDAR's coordinate system is supposed to be known. This matrix is composed of $T \in \mathbb{R}^{3\times1}$

and $R \in \mathbb{R}^{3\times3}$, respectively the translation vector and rotation matrix relating the two coordinate systems. Let $x = (X, Y, Z, 1)^T$ be a LIDAR point. It can be matched to a pixel $y = s \cdot (u, v, 1)^T$ as follow:

$$y = K \begin{bmatrix} R & T \end{bmatrix} x \tag{4.7}$$

Then, the RANSAC algorithm, fed with matched LIDAR points and pixels belonging to the ground plane, can be used to compute the perspective projection matrix $H_{pg} \in R^{3\times3}$, between the ground plane in the LIDAR's coordinate system, and the camera plane. In practice, LIDAR points and pixels belonging to the ground were manually matched together by hand. It is then possible to match each pixel with a grid cell. To keep the computational workload low, only the center of each cell of the resulting grid is matched to a segmented pixel, instead of projecting every pixel in the evidential grid's coordinate system. Let x_{cell} be the known coordinates of the center of a cell, and y_{pixel}, the coordinates of the corresponding image pixel ; y_{pixel} can be computed as follow:

$$y_{pixel} = H_{pg} \cdot x_{cell} \tag{4.8}$$

Finally, the output of a fully convolutional neural network is used to build a new mass function, for every grid cell. The activation values are normalized, for every pixel mapped to a grid cell, via the softmax function, and then used to build a new mass function m_c. Let $\Omega_{cnn} = \{A_0, A_1, ..., A_n\}$ be the set of classes the convolutional neural network has been trained on. Let $z_p = (z_0^p, z_1^p, ..., z_n^p)$ be the corresponding activation values generated during inference by the neural network, for a pixel p. Let $\sigma(z_i^p)$ be the normalized activation for the class A_i and the pixel p, obtained from the softmax function. Then:

$$\sigma(z_i^p) = \frac{\exp z_i^p}{\sum_{k=0}^{n} \exp z_k^p} \tag{4.9}$$

$\sigma(z_i^p) \in [0, 1]$ and $\sum_{i=0}^{n} \sigma(z_i^p) = 1$. Let $\cup_{B_i \in B}$ be the union of all the sets, depicted as B_i, that belong to the B set. It is assumed that there exists a partition of $\Omega_{cnn} = \{A_D, A_{\neg D}, A_\Omega\}$ such that:

$$\cup_{A \in A_D} = D \tag{4.10}$$

$$\cup_{A \in A_{ND}} = \neg D \tag{4.11}$$

$$\cup_{A \in A_\Omega} = \Omega \tag{4.12}$$

A mass m_p can then be computed for each pixel p mapped to a grid cell, as follow:

$$m_p(\emptyset) = 0 \tag{4.13}$$

$$m_p(D) = \sum_{A_i \in A_D} \sigma(z_i^p) \tag{4.14}$$

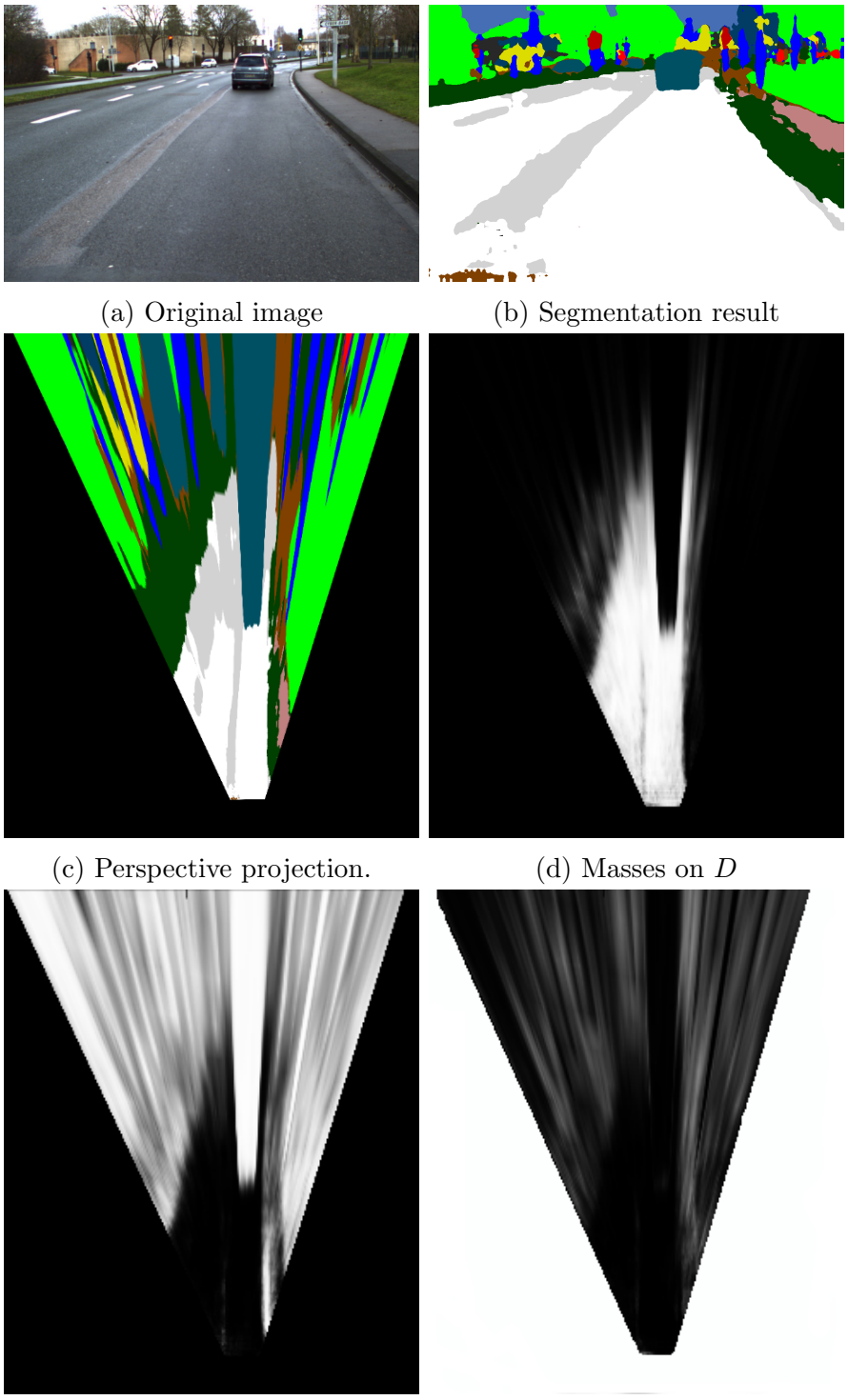
$$m_p(\neg D) = \sum_{A_i \in A_{ND}} \sigma(z_i^p) \tag{4.15}$$

$$m_p(\Omega) = \sum_{A_i \in A_\Omega} \sigma(z_i^p) \tag{4.16}$$

A new evidential grid is then obtained from the segmented image thanks to Equation (4.8), which is used to map each cell of the grid with the corresponding segmentation results, and mass values. Fig. 4.5 illustrates how the mass value for each proposition, in each cell, is derived from the corresponding segmentation result. As showed in Fig. 4.5(c), when objects that do not belong to the ground are projected, they are stretched, which is normal since their presence occludes the ground in the distance. Since the information is meaningful, the grids are kept as they are.

4.2.3 Asynchronous fusion of LIDAR data and image segmentation results as an evidential grid

If the speed vector of the vehicle is available at any instant, and if all the sensor readings are accurately timestamped, grids corresponding to consecutive sensor readings can be processed independently from the type of sensor that issued the raw data. The grid obtained after fusion is called *EgoGrid* (cf. Fig 4.1), and has a BBA denoted as m_{eg} for each cell. At every new sensor reading, issued at a date t_i, a single evidential grid *SensorGrid(t_i)* is generated based on the type of sensor input (cf. Fig 4.1). After the ego-motion of the vehicle is compensated in *EgoGrid(t_{i-1})*, $m_{eg}(\Omega)$ is set to 1 for all the new cells that cover previously absent areas. Then, *SensorGrid(t_i)* can be fused with *EgoGrid(t_{i-1})* into a new *EgoGrid(t_i)* evidential grid via Dempster's conjunctive rule. Thanks to this framework, new sensors can easily be added to the fusion process, and a faulty sensor can be ignored without preventing the system from working in a degraded mode. Furthermore, contradictions between successive frames at a given locations are handled during fusion based on the mass associated to each state, in each frame. However, objects that have potentially moved between successive sensor readings have to be accounted for. No strong dynamic model, for any type of object, is presupposed. Instead, a decay factor is used to force every cell to eventually tend to the unknown state, similarly to [63]. The types of the object present in the scene are made available by using a convolutional neural network, to process the original undistorted RGB image. An unique decay rate can be computed for each cell, based on the likelihood of the presence of a moving object. Let β be the decay rate associated with a cell of the current *EgoGrid*. Four main types of objects, that have similar dynamic behaviors, were identified: four-wheeled vehicles, two-wheeled vehicles, pedestrians and fixed objects. Each behavior is associated with a typical decay rate: β_{4W}, β_{2W}, β_P and β_F. If no semantic information has been previously provided about a cell of the *EgoGrid*, a default value is used as the decay rate. Yet, if semantic information has been available for a given cell of the *EgoGrid* at previous dates, four values, indicating



(a) Original image

(b) Segmentation result

(c) Perspective projection.

(d) Masses on D

(e) Masses on $\neg D$

(f) Masses on Ω

Figure 4.5: Building process of an image-based evidential grid. The picture of a driving scene is segmented by DeepLab-V2 [69], a convolutional neural network, before being projected on the ground plane. The masses for D (drivable), $\neg D$ (non-drivable) and Ω (unknown) are then derived from the activations for each segmented class. In (d), (e), (f), the lighter a pixel is, the larger the mass is.

the likelihood that the object detected at this position correspond to one of those dynamic behaviors, are computed. Those indicators are denoted as L_{4W}, L_{2W}, L_P and L_F. Typically, when DeepLab-v2 is used, L_{4W} is equal to the sum of the activations corresponding to four-wheeled vehicles in the previous frames, for the given cell ; L_{2W} is equal to the sum of the activations corresponding to two-wheeled vehicles in the previous frames ; L_P is equal to the sum of the activations corresponding to pedestrians in the previous frames, and L_F is equal to the sum of the activations corresponding to fixed objects, such as the road, or buildings, in the previous frames. Then, the final decay rate for the cell is given by a weighted arithmetic mean:

$$\beta = \frac{L_{4W}\beta_{4W} + L_{2W}\beta_{2W} + L_P\beta_P + L_F\beta_F}{L_{4W} + L_{2W} + L_P + L_F} \tag{4.17}$$

Before fusing *EgoGrid*(t_{i-1}) and *SensorGrid*(t_i), each cell of *EgoGrid*(t_{i-1}) is updated using β, as follows:

$$m_{eg}(A) = \beta \cdot m_{eg}(A), A \subset \Omega \tag{4.18}$$

$$m_{eg}(\Omega) = 1 - \beta + \beta \cdot m_{eg}(\Omega) \tag{4.19}$$

Then, a Dempster-Shafer fusion can be used to fuse the mass values among the grids, at the cell level, as follows:

$$EgoGrid(t_i) = SensorGrid(t_i) \oplus EgoGrid(t_{i-1}) \tag{4.20}$$

4.3 Experimental results

The evidential fusion scheme was implemented thanks to the software library proposed by Fankhauser et al. [70]. It was tested on real-life driving data collected around HeuDiaSyc Lab in Compiègne, France. The evidential grids are built from a VLP-16 LIDAR and a single HD camera, and the pose and speed of the vehicle were obtained from an IMU. Popular LIDAR/camera datasets, such as KITTI, were not considered for those tests, since one of the specificity of the proposed method is that it is intended to work in an asynchronous fashion. Evidential grids of $(90\times90)m^2$ are built from the collected data, with cells of size $(0.1\times0.1)m^2$. H was empirically set to 0.2m, α_{FA} to 0.05, the default value of β to 0.995, β_{4W} to 0.80, β_{2W} to 0.75, β_P to 0.95 and β_F also to 0.995. LIDAR scans were acquired at 10 Hz, and the camera was freely running at 30 Hz. The extrinsic calibration matrix between the LIDAR's coordinate system and the camera's coordinate system was estimated from the semi-automatic tool offered within the Autoware software stack [71]. The recent and fast algorithm described in [72] was used to classify each LIDAR point as either ground point or obstacle. DeepLab-v2 was finetuned on the publicly available Mapillary Vistas dataset [73], consisting in 25000 real-life driving scenes labeled into 66 object categories, to make it usable in our experiments. To speed up and ease the

fine-tuning of DeepLab-v2, the total number of classes was reduced, by factorizing some of them. A class for unlabeled objects in Mapillary Vistas was also reserved, and included in the loss calculation as an *unknown* class. Doing so, pixels are not forced to be classified into a meaningful class. Thus, in this set up, A_Ω ={*unknown*, *sky*}, as the pixels depicting the sky are not supposed to be part of the ground plane. The *unknown* class in A_Ω corresponds to the pixels that are labeled as *unknown* in Mapillary Vistas. A_D ={*road*, *road marking*, *crosswalk*}, and the remaining classes form $A_{\neg D}$. The classical stochastic gradient descent with momentum was used for fine-tuning DeepLab-v2, with the same parameters as in [69]. The loss function was modified to handle class imbalance within the dataset, by weighting the error for each pixel depending on the target class thanks to median class balancing [74]. The fine-tuning of DeepLab was performed during sixteen epochs, until the validation loss started increasing. Three cases, each highlighting specific advantages and drawbacks of the proposed approach, are presented. They were generated from the same driving sequence, but at different instants. The data collection vehicle was driven in a peri-urban environment and overtaken by another vehicle. During the overtaking, the camera was permanently switched off, to simulate a sensor failure.

4.3.1 Handling sporadic semantic segmentation errors

First, the robustness of the fusion scheme against incoherences between successive sensor readings, and especially sporadic false alarms, is highlighted in Figure 4.6. In the semantic segmentation result, white indicates that the class with the highest activation is "road" ; grey that it is "road marking" ; blue that it is "building" ; purple that it is "sidewalk" ; green that it is "border". In the grids, red cells are those where the largest mass is for $\neg D$; white that the largest mass is for D ; black that it is for Ω ; the green point indicates the origin of the LIDAR, considered to be the vehicle's position. In Fig. 4.6b, many segmentation errors seem to come from the fact that the road is particularly damaged, and was repaired many times. As a result, objects are wrongly considered to be present, especially a building in the bottom-left corner. In the *SensorGrid*, a blue rectangle indicates the cells corresponding to this wrongly detected building. The mass of the cells in this area is larger for Ω, which indicates that even if the activation for the "building" class is the largest, the sum of the classes corresponding to A_Ω is larger. The segmentation result is thus very uncertain in this area. This is not the case for the pixels wrongly classified as "side-walk", but belonging in fact to the road, since small obstacles are detected in *SensorGrid* in front of the vehicle. The $EgoGrid(t_{i-1})$ to be fused with the *SensorGrid* was generated from 6 previous LIDAR scans and 7 previous images. The resulting $EgoGrid(t_i)$ is marginally impacted, as no obstacle is considered to be present in front of the origin, even if a small area is considered to be unknown. This means that the mass for the areas falsely considered to be non-drivable in *SensorGrid* was not very high, compared to the corresponding mass in $EgoGrid(t_{i-1})$ for the D proposition. This shows the interest of fusing all the information over time,

and to consider all the activations of the neural network.

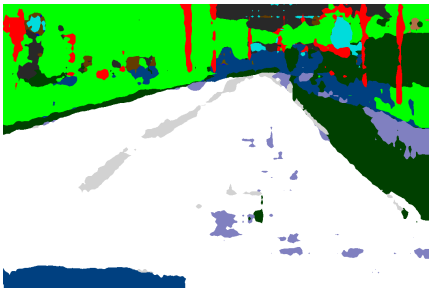
4.3.2 Handling systematically contradictory information

If sporadic errors can efficiently be handled via temporal fusion, among successive frames and sensor inputs, the behavior against systematic errors is not always as satisfactory. The ground segmentation algorithm used in this experimental set-up is mainly designed to detect ground planes. As such, roads and side-walks are often both considered to be drivable in a *SensorGrid* generated from a LIDAR scan. Nevertheless, side-walk borders are efficiently detected, thanks to the gap between roads and side-walks, as shown in Fig. 4.7. Side-walks though remain uncertain, as the corresponding cells are not consistently classified as non-drivable.

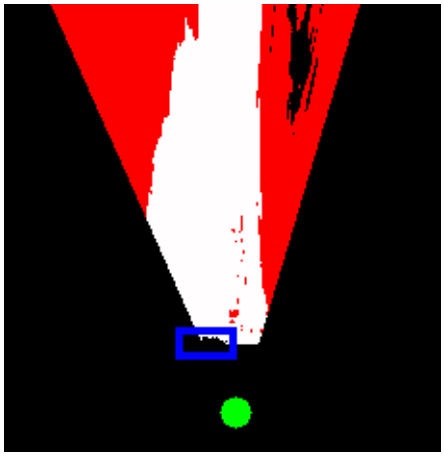
4.3.3 Handling sensor failures

The last case regards the moment when a camera failure was simulated. *SensorGrids* can then only be generated from LIDAR scans. As a result, the *EgoGrid* was updated less often, and the decay was less applied. As shown in Fig. 4.8d, this results in the conservation of outdated information, coming from previous detections. However, the car is still detected, and after a few more scans, the results become more consistent, as seen in Fig. 4.8e. Finally, as the side-walk borders are still detected, an autonomous navigation in such a fail-soft mode would have still been possible.

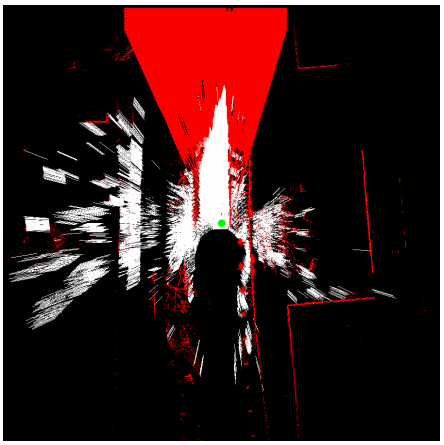
(a) Original image



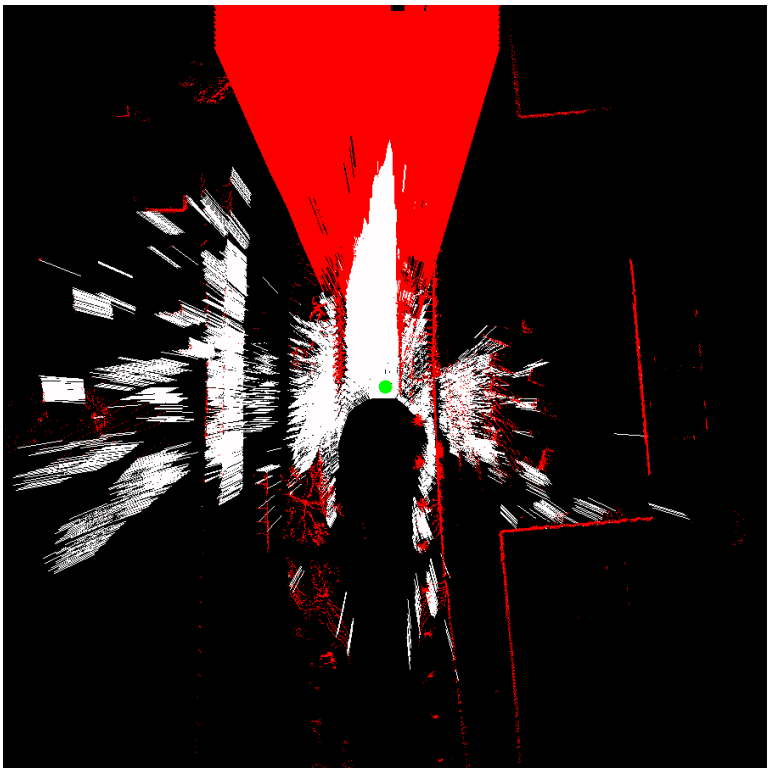
(b) Semantic segmentation



(c) $SensorGrid(t_i)$

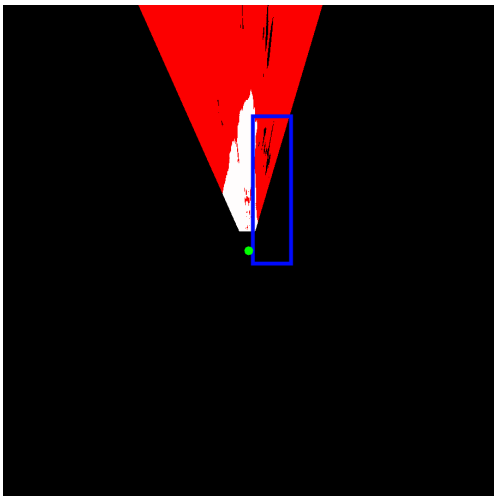


(d) $EgoGrid(t_{i-1})$

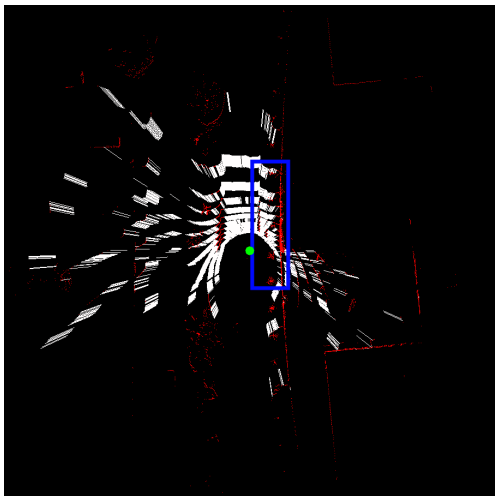


(e) Resulting $EgoGrid(t_i)$ after fusion

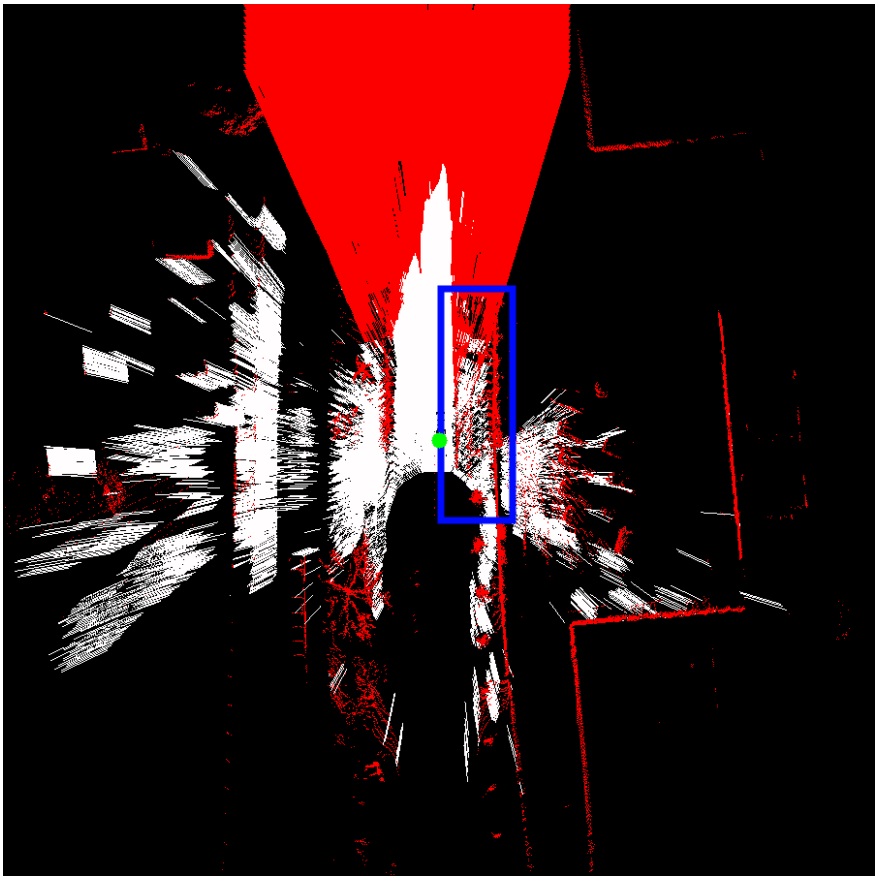
Figure 4.6: Robustness of the fusion against sporadic errors. As displayed in (b), an object was wrongly detected by DeepLab. However, the conversion and fusion of this information in the evidential framework efficiently filtered the semantic segmentation result.



(a) *SensorGrid* generated from an image segmentation result



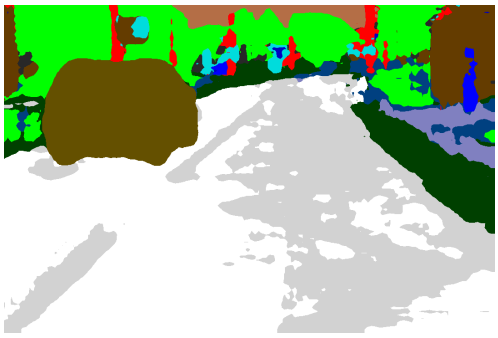
(b) *SensorGrid* generated from a LIDAR scan



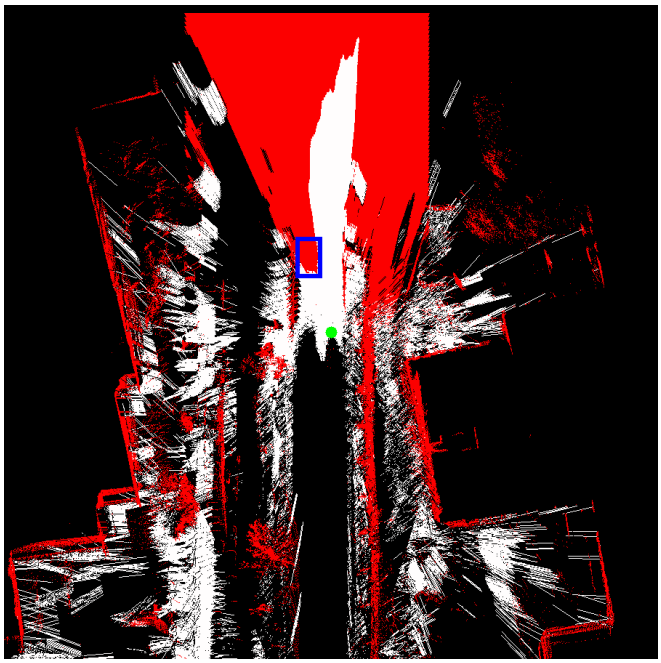
(c) *EgoGrid* generated at this position

Figure 4.7: Result of systematic inconsistencies between *SensorGrids*. The dark blue rectangle indicate the approximative position of a side-walk.

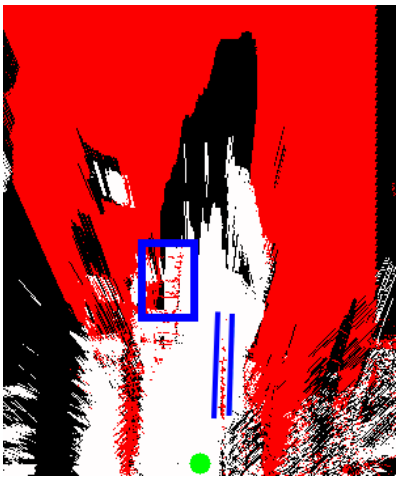
(a) Last image before camera failure



(b) Last segmentation result before camera failure



(c) Last *EgoGrid* before camera failure

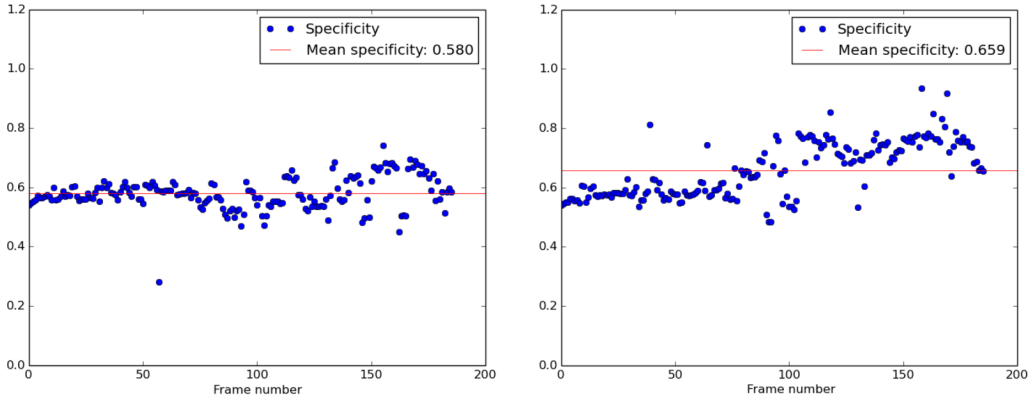


(d) *EgoGrid* after 1s

(e) *EgoGrid* after 2s

Figure 4.8: Handling camera failure. Blue rectangles indicate the actual position of the overtaking car, and blue lines highlight the border of the sidewalk.

4.3.4 Evaluation of the importance of handling moving objects



(a) Mean specificity for a fixed decay rate

(b) Mean specificity for a not-fixed decay rate

Figure 4.9: Comparison of the average specificity for a fixed decay rate, and the proposed class-dependent decay rate

Following the procedure proposed in [68], we evaluate our grids by calculating entropy and specificity values. The effectiveness of the proposed grid mapping scheme can be analyzed from such indicators. Let E_m be the entropy of the mass function m, and S_m its specificity. Let the plausibility of a set A be $pl(A) = \sum_{B \cap A \neq \emptyset} m(B)$.

$$E_m = -\sum_{A \subseteq \Omega} m(A).ln(pl(A)) \tag{4.21}$$

$$S_m = \sum_{A \subseteq \Omega, A \neq \emptyset} \frac{m(A)}{card(A)} \tag{4.22}$$

A high degree of specificity and low-degree of entropy indicate that the mass function is informative and non-ambiguous. The mean entropy and specificity of the mass assignment in the cells of an evidential grid are thus representative indicators of the quality of the whole representation.

Those values were calculated for each frame of the sequence, in two cases: first, with a fixed decay rate of 0.98 for each cell as in [65], and then based on the values of β_{4W}, β_{2W}, β_P and β_F. The camera failure was simulated from the frame 163. In both cases, the entropy was extremely low, and below 0.015. Yet, as shown in Fig.4.9a and 4.9b, the specificity is higher for this sequence when the class of the object in each cell is considered, making the resulting grids more informative. Indeed, the average specificity for the sequence is 0.580 when a fixed decay rate is used, and 0.659 when the decay rate is computed based on the activations of the neural network.

4.4 Conclusion

We presented an asynchronous grid mapping algorithm that relies on LIDAR scans and RGB images. The occupation of each cell is estimated from either a geometrical model associated to LIDAR scans, or a semantic segmentation result generated from an convolutional neural network and the raw image pixels. A limitation of this approach, however, is that it relies on the evidential theory in a purely semantic way. Actually modeling the behavior of the network might lead to more trustworthy results, and open the path to new applications. This is what we tried to do for LIDAR object classification, as presented in the next section.

Chapter 5

Evidential LIDAR object classification

Contents

5.1 Introduction

Detecting and recognizing road users is paramount for autonomous vehicles that are intended to drive on public road. 3D sensors, and especially LIDAR scanners, seem particularly suitable for those tasks. Indeed, an accurate 3D position of the detected objects, with regards to the origin of the sensor, can be obtained from LIDAR scanners. In parallel, unmanned ground vehicles that follow the standard 4D/RCS model [75] rely on processing pipelines that include a segmentation step -to detect objects- and a classification step -to infer the type of each detected object. Using similar design choices in the context of autonomous driving, when working with LIDAR raw data, thus appears natural.

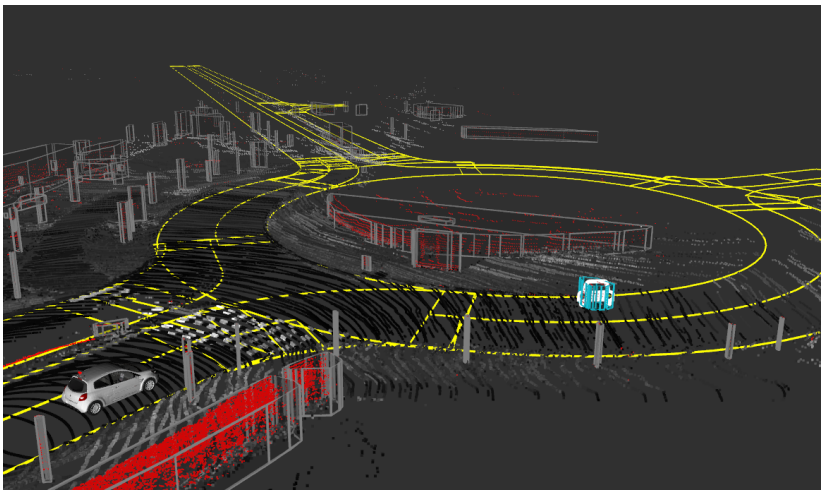


Figure 5.1: Example of output from a LIDAR perception pipeline using the proposed classifier. The yellow lines correspond to a pre-existing map of the scene; the red LIDAR points belong to obstacles ; the grey 3D polygons represent objects classified as unknown objects; the blue 3D polygon represents an object classified as vehicle.

However, a classifier used within such a processing pipeline should be able to cope with any possible object generated during the segmentation step, and always output pertinent results. A naive way to cope with this requirement would be to collect large amounts of data which would be accurately labeled afterwards, and to train a classifier on the resulting dataset. Unfortunately, this method is not guaranteed to cover all the randomness that an autonomous vehicle is likely to meet on public roads. This would then lead to errors in situation understanding. For instance, in the situation in Fig. 5.1, if a pedestrian detector were to consider that the poles on the sides of the roundabout are pedestrians because it wasn't trained to reject them, this would falsely complexify the situation understood by the vehicle.

A more realistic way to grapple with this randomness might be to use classifiers that are able to classify objects as unknown, while having only been trained on known objects. The evidential theory, or Dempster-Shafer theory, in which the unknown is explicitly represented, seems of particular use. Nevertheless, this approach also has two main limits. First evidential labels, in which the fact of not knowing is quantified, are hard to obtain. Then, evidential classifiers usually rely on a closed-world assumption [76]: objects classified as unknown are usually ambiguous ones with regards to the training dataset.

To address those two limits, a multi-task multi-layer perceptron (MLP) is trained on labeled LIDAR objects, and its outputs are reinterpreted as a fusion of evidential mass functions. This is accomplished via an extension of T. Denoeux's recent work on generalized logistic regression (GLR) classifiers [77], which enables statistically incoherent features to be filtered. This work only

aims at classifying vehicles (cars, trucks) and vulnerable road users (pedestrians, two-wheeled vehicles), while classifying other objects as unknown without having represented them explicitly in the training dataset. Fig. 5.1 highlights the interest of such a system: the object that was not classified as an unknown one was the vehicle in the roundabout, although the classifier wasn't trained on the other objects.

5.2 Evidential end-to-end formulation of binary logistic regression classifiers

5.2.1 Binary generalized logistic classifiers

Let a binary classification problem with $X = (x_1, ..., x_d)$, a d-dimensional input vector, and $Y \in \Theta$ a class variable. Let $p_1(x)$ be the probability that $Y = \theta_1$ according to the fact that $X = x$. Then $1 - p_1(x) = p_2(x)$ is the corresponding probability that $Y = \theta_2$. Let w be the output of a binary logistic regression classifier, trained to solve the aforementioned classification problem. A generalized binary logistic regression classifier corresponds to the case where there exists a C-dimensional vector x_c and such that $x = (\phi_1(x_c), ..., \phi_d(x_c))$. Then, $p_1(x)$ is such that:

$$p_1(x) = S(w) = S(\sum_{i=1}^{d} \beta_i \phi_i(x_c) + \beta_0) \tag{5.1}$$

with S being the sigmoid function, and the β values being usually learnt alongside those of the potentially non-linear ϕ_i mappings. In Equation 5.1, w exactly corresponds to the output of a multi-layer perceptron trained as a binary GLR classifier.

5.2.2 Binary GLR classifiers as a fusion of simple mass functions

The sigmoid function is strictly increasing. Then, in Equation 5.1, the larger w is, the larger $p_1(x)$ is and the smaller $p_2(x)$ is. Moreover, w can be rewritten as follows:

$$w = \sum_{j=1}^{d} w_j = \sum_{j=1}^{d} (\beta_j \phi_j(x_c) + \alpha_j) \tag{5.2}$$

with

$$\sum_{j=1}^{d} \alpha_j = \beta_0 \tag{5.3}$$

Each w_j can then be seen as piece of evidence towards θ_1 or θ_2, depending on its sign. Let us assume that the w_j values are weights of evidence of simple mass functions, denoted by m_j. Let $w_j^+ = max(0, w_j)$ be the positive part of w_j,

and let $w_j^- = max(0, -w_j)$ be its negative part. Whatever the sign of w_j, the corresponding m_j can be written as

$$m_j = \{\theta_1\}^{w_j^+} \oplus \{\theta_2\}^{w_j^-} \tag{5.4}$$

If the m_i mass functions are independent, the Dempster-Shafer operator can be used to fuse them together. The resulting mass function obtained from the output of the binary logistic regression classifier, noted m_{LR} is as follows:

$$m_{LR} = \oplus_{j=1}^{d}(\{\theta_1\}^{w_j^+} \oplus \{\theta_2\}^{w_j^-}) = \{\theta_1\}^{w^+} \oplus \{\theta_2\}^{w^-} \tag{5.5}$$

with $w^+ = \sum_{j=1}^{d} w_j^+$ and $w^- = \sum_{j=1}^{d} w_j^-$. From Equation 5.5, m_{LR} can be expressed as follows:

$$m_{LR}(\theta_1) = \frac{[1 - \exp(-w^+)]\,(\exp(-w^-))}{1 - K} \tag{5.6a}$$

$$m_{LR}(\theta_2) = \frac{[1 - \exp(-w^-)]\,(\exp(-w^+))}{1 - K} \tag{5.6b}$$

$$m_{LR}(\Theta) = \frac{\exp(-w^+ - w^-)}{1 - K} \tag{5.6c}$$

with (5.6d)

$$K = \left[1 - \exp(-w^+)\right]\left[1 - exp(-w^-)\right] \tag{5.6e}$$

By applying the plausibility transformation in Equations 3.15 to this mass function, the following probability can be obtained:

$$p_{m_{LR}}(\theta_1) = S(w) \tag{5.7}$$

which exactly corresponds to the output of the GLR classifier, depicted in Equation 5.1. This means that any binary GLR classifier can be seen as a fusion of independent and simple mass functions, that can be derived from its parameters. In the case of a multi-layer perceptron, its output can be converted into a mass function via Equations 5.6, only using the output from its penultimate layer and the parameters of its final layer. However, the α_i values in Equation 5.3 have to be estimated.

5.3 End-to-end evidential interpretation of a binary GLR classifier and online statistical filtering

T. Denoeux proposed to explicitly compute the α_i values after the training, so that the resulting simple mass functions are the most uncertain ones. This means that the weights of evidence of those mass functions should be as small as possible, which leads to the following minimization problem [77]:

$$minf(\alpha) = \sum_{i=1}^{n}\sum_{j=1}^{d}(\beta_j\phi_j(x_i) + \alpha_j)^2 \tag{5.8}$$

with $\{(x_i, y_i)\}_{i=1}^{n}$ being the training dataset, and $\alpha = (\alpha_i, ...\alpha_d)$.

However, we observe that this minimization problem can be instead solved during the training, under the assumption that the last layer of the MLP performs Instance Normalization [78]. Let $\upsilon(x_c) = (\upsilon_1(x_c), ..., \upsilon_d(x_c))$ be the mapping modeled by all the consecutive layers of the MLP but the last one ; let $\overline{\upsilon_j}$ be the mean value of the υ_j function on the training set, and $\sigma(\upsilon_j)^2$ its corresponding variance. The output of the MLP depicted in Equation 5.1 then becomes:

$$p_1(x) = S(w) = S(\sum_{j=1}^{d}(\beta_j \frac{\upsilon_j(x_c) - \overline{\upsilon_j}}{\sqrt{\sigma(\upsilon_j)^2 + \epsilon}}) + \sum_{j=1}^{d} \alpha_j) \tag{5.9}$$

In particular, it can be noticed that α parameters are already present, given the expression of Instance Normalization. After develepment, the expression in 5.8 becomes:

$$minf(\alpha) = \sum_{j=1}^{d} {\beta_j}^2 (\sum_{i=1}^{n} \phi_j(x_i)^2) + n \sum_{j=1}^{d} {\alpha_j}^2 + 2 \sum_{j=1}^{d} \beta_j \alpha_j \sum_{i=1}^{n} \phi_j(x_i) \tag{5.10}$$

The third term in this expression is equal to 0 when using Instance Normalization. Indeed, Instance Normalization produces centered features, meaning that $\sum_{i=1}^{n} \phi_j(x_i) = 0$. We rewrite the first term in the expression, again considering that Instance Normalization is used in the final layer. We assume that ϵ can be neglected with regards to the $\sigma(\upsilon_j)^2$ values. This assumption is reasonable in the general case, as the value of ϵ is chosen to be small, and is only used for numerical stability. The minimization problem in Equation 5.8 becomes after development:

$$minf(\alpha) = \sum_{j=1}^{d} \beta_j^2 \frac{\sum_{i=1}^{n} (\upsilon_j(x_c) - \overline{\upsilon_j})^2}{\sigma(\upsilon_j)^2} + n \sum_{j=1}^{d} {\alpha_j}^2 \tag{5.11}$$

It has to be noted that:

$$\sigma(\upsilon_j)^2 = \frac{1}{n} \sum_{i=1}^{n} (\upsilon_j(x_c) - \overline{\upsilon_j})^2 \tag{5.12}$$

From 5.11 and 5.12, we get:

$$minf(\alpha) = n \sum_{j=1}^{d} \beta_j^2 + n \sum_{j=1}^{d} \alpha_j^2 \tag{5.13}$$

The minimization problem can then be trivially solved in an online fashion, by simply applying weight decay to the parameters of the final Instanec Normalization layer. Applying Batch Normalization to the final layer of the MLP also has another interest: it can be the basis for an online statistical filtering scheme.

Let $z(\upsilon_j(x_i)) = \frac{\upsilon_j(x_i) - \overline{\upsilon_j}}{\sigma(\upsilon_j)}$ be the Z-score of $\upsilon_j(x_i)$. Under the assumption that the υ_j function can be modeled as a random variable following a normal distribution, a simple thresholding can be used to define confidence levels: the larger the

Z-score is, the more unlikely to happen $\upsilon_j(x_i)$ is. Moreover, the Central Limit Theorem states that a sum of independent random variables can be modeled as a normal distribution [79]. If the MLP mainly implements linear functions, the $\upsilon_j(x_i)$ values can be approximately considered as sums of random variables. Statistically abnormal $\upsilon_j(x_i)$ values can then be rejected by a simple thresholding on their Z-score.

Again under the assumption that ϵ can be neglected with regards to the $\sigma(\upsilon_j)^2$ values, the w_j values in Equation 5.2 can be seen as:

$$w_j \approx \beta_j * Zscore(\upsilon_j(x_c)) + \alpha_j \tag{5.14}$$

When trying to classify inputs without any guarantee that only pertinent objects will be passed to the classifier, the Z-Score can be used to detect objects that are extremely different from the training set, and for which the mass function obtained from w_j is unlikely to be reliable. Abnormal objects with regards to the application domain of the classifier can then be easily accounted for, by introducing an additional hyperparameter. Let $ZMax$ be a threshold value. During inference, each w_j for which the $Zscore$ overpasses the $ZMax$ value can then be excluded from the final fusion process. The final mass values are then only computed by fusing simple mass functions whose w_j weights are reliable, with regards to the training set. Shutting down a w_j based on the $ZScore$ is equivalent to the following procedure:

$$w_j = \begin{cases} 0,\ if\ |\frac{\upsilon_j(x_i)-\overline{\upsilon_j}}{\sqrt{\sigma(\upsilon_j)^2+\epsilon}}| > Zmax \\ \\ \beta_j * \frac{\upsilon_j(x_i)-\overline{\upsilon_j}}{\sqrt{\sigma(\upsilon_j)^2+\epsilon}} + \alpha_j,\ otherwise \end{cases} \tag{5.15}$$

According to Equation 5.4, the m_j mass function corresponding to the case where $|\frac{\upsilon_j(x_i)-\overline{\upsilon_j}}{\sqrt{\sigma(\upsilon_j)^2+\epsilon}}| > Zmax$ becomes:

$$m_j(\theta_1) = 0,\ m_j(\theta_2) = 0,\ m_j(\Theta) = 1 \tag{5.16}$$

This *vacuous* mass function represents a completely uninformative piece of evidence, over the frame of discernment. By this online statistical filtering scheme, the final mass function in Equation 5.6 is then affected, and the value for $m_{LR}(\Theta)$ is increased. From this formalism, an evidential multi-task multi-layer perceptron was designed, and trained to classify LIDAR objects as either vehicles, vulnerable road users, or unknown objects.

Observations

A statistical outlier, for which the $Zscore$ overpasses $ZMax$, should not theoretically be associated with a vacuous mass function. Indeed, it would make sense to instead associate this event with an unnormalized mass function maximizing the evidence on the empty set, as this would correspond to classification in open-world. Therefore, m_j would be in this case:

$$m_j(\emptyset) = 1,\ m_j(\theta_1) = 0,\ m_j(\theta_2) = 0,\ m_j(\Theta) = 0 \tag{5.17}$$

Yet, this mass function is not usable in practice. It is indeed the absorbing element of the unnormalized rule of combination (cf. Equation 3.7). This means that if only one of the w_j values are detected as outliers, based on their associated $Zscore$, the final mass function after fusion would also be equal to the one in Equation 5.17, making the classifier overly sensitive to rare and atypical input features. On the contrary, using a *vacuous* mass function allows for a finer representation, as it only affects some of the w_j weights, and not the whole resulting mass. Due to this expected behavior, we instead chose to simply follow the procedure in Equation 5.15. Section 5.6 will deepen this discussion on the use of unnormalized mass functions.

5.4 Evidential classification of LIDAR objects

5.4.1 Training dataset

Although publicly available datasets of labeled LIDAR objects exist, such as the KITTI dataset [10], they do not include any explicitly unknown objects. Moreover, we assumed that the classification of LIDAR objects must be performed after a detection step. Then, a coupling with a pre-existing detection system, and a dataset with labeled unknown objects, were needed to test the evidential framework that was previously defined, when applying it to LIDAR object classification. Raw point clouds were thus acquired from a ZoeCab platform, via a Velodyne VLP-32C sensor. ZoeCabs are robotized electrical vehicles, based on Renault Zoes, that are augmented with perception and localization sensors, and intended to be deployed as autonomous shuttles in urban and peri-urban areas.

Figure 5.2: The ZoeCab data acquisition platform used to collect LIDAR objects ; the LIDAR sensor stands on top of the vehicle

The final dataset is the result of three independent recordings: two that happened on different dates in Guyancourt, France, and one in Toulouse, France. In total, this represents approximately ninety minutes of raw data. The data acquisition platform is depicted in Fig. 5.2. A ground detection algorithm [72] was used to detect the ground, and the points that did not belong to the ground were clustered into objects, via a real-time implementation of the DBSCAN algorithm [80]. Detected objects that comprised less than ten points and were further than 45 meters from the vehicle were rejected. Object tracks were then created by associating and tracking the remaining objects over time via a simple Extended Kalman Filter. For each track, a single tracklet was then manually

labeled as either "unknown", "car", "truck", "bike" or "pedestrian", and the label was propagated to all the other objects of the track. Tab. 5.1 depicts the number of samples for each class in the dataset.

label	**number of samples**
Car	91297
Truck	9713
Pedestrian	3461
Bike	946
Unknown	10492

Table 5.1: Number of LIDAR objects per class in the dataset

Although the main goal of this work was only to classify vehicles and vulnerable road users while rejecting unknown objects, extra labels were needed during training. Indeed, trucks, which are way larger than cars, could for instance easily be considered as outliers in a dataset of vehicles. This could be problematic with the statistical filtering scheme presented in the previous section.

A bounding box was fitted to each object by using the Variance Minimization algorithm in [81], and each object was converted into a vector of nine features:

- Distance between the centroid of the box and the sensor;
- Length, width and height of the fitted bounding box;
- Mean distance between the object points and the centroid of the fitted bounding box, and the corresponding standard deviation;
- The three eigenvalues, computed from a principal component analysis on the Euclidean coordinates of the object points;

5.4.2 Model

A multi-task multi-layer perceptron, depicted in Fig. 5.3, was trained on the dataset of LIDAR objects. The neural network includes linear layers, PReLU activation layers and batch normalization layers. The PReLU activation was used since it always applies a linear function to its input, though its behavior depends on the sign of the input. The multi-task behavior corresponds to the model defined in section IV, which applies to binary GLR classifiers. The MLP has four outputs, corresponding to four binary GLR classifiers, as four different classes are present, at least during the training. For each object x, the multi-task MLP can then predict four probabilities:

- $P_x(P)$: probability of the object being a pedestrian
- $P_x(B)$: probability of the object being a bike

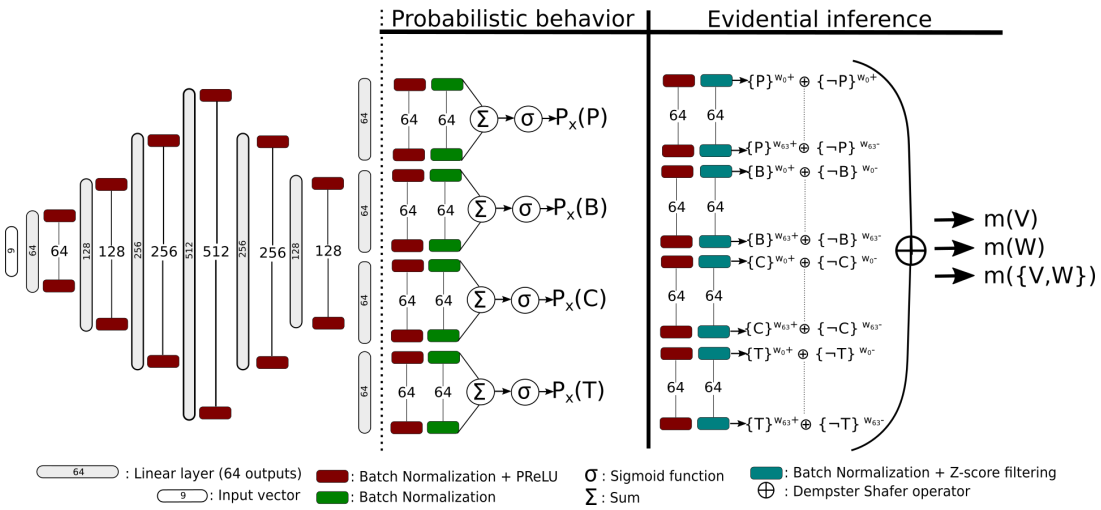


Figure 5.3: The proposed multi-task architecture

- $P_x(C)$: probability of the object being a car
- $P_x(T)$: probability of the object being a truck

Let $\neg$ represent logical negation. From Equation 5.6, those probabilities can be converted into evidential mass functions:

$$m(P),\ m(\neg P),\ m(\{P, \neg P\}) \tag{5.18a}$$
$$m(B),\ m(\neg B),\ m(\{B, \neg B\}) \tag{5.18b}$$
$$m(C),\ m(\neg C),\ m(\{C, \neg C\}) \tag{5.18c}$$
$$m(T),\ m(\neg T),\ m(\{T, \neg T\}) \tag{5.18d}$$

Let $\Omega = \{V, W\}$ a frame of discernment, V representing the fact that an object is a vehicle, W representing the fact that an object is a vulnerable road user. This new frame of discernment is justified by the fact that the original goal of the work is only to classify vehicles and vulnerable road users. It is assumed that the fact of not being a car or a truck (resp. a pedestrian or a bike) is not considered as an evidence towards the fact of being a vulnerable road user (resp. a vehicle). The mass functions in Equation 5.18 can then be projected into this new frame of discernment as follows:

$$m_p(V) = 0,\ m_p(W) = m(P),\ m_p(\{V, W\}) = 1 - m(P) \tag{5.19}$$
$$m_b(V) = 0,\ m_b(W) = m(B),\ m_b(\{V, W\}) = 1 - m(B) \tag{5.20}$$
$$m_c(V) = m(C),\ m_c(W) = 0,\ m_c(\{V, W\}) = 1 - m(C) \tag{5.21}$$
$$m_t(V) = m(T),\ m_t(W) = 0,\ m_t(\{V, W\}) = 1 - m(T) \tag{5.22}$$

Those four mass functions can then be fused via the Dempster-Shafer operator, to get the final mass value m generated from the MLP:

$$m = m_p \oplus m_b \oplus m_c \oplus m_t \tag{5.23}$$

Algorithm 1 Interval Dominance decision rule on $\{V, W\}$

if $1 - Bel(V) \leq 1 - Pl(W)$ **then**
    The object is classified as a vehicle
else if $1 - Bel(W) \leq 1 - Pl(V)$ **then**
    The object is classified as a vulnerable road user
else
    The object is classified as unknown
end if

5.4.3 Model training

The multi-task MLP was implemented in PyTorch. The evidential formulation is not used during inference. The training is only done on the object of the "pedestrian", "bike", "car" and "truck" classes, which compose a dataset of pertinent objects noted D_p. The "unknown" objects are only used to create a one class D_u dataset, which will only be used to evaluate the evidential output of the MLP. The parameters of the Batch Normalization layers are estimated during the training. Thus, no Dropout was used, as the statistics in the Batch Normalization layers have to be as accurate as possible to justify the behavior proposed in Equation 5.15. Moreover, the training iterations were done on a single batch composed of all the pertinent objects. A training set D_{pt} and a validation set D_{pv} were created from D_p by a 70/30 split. As seen in Tab. 5.1, D_p is very unbalanced. D_{pt} was then refined by randomly sampling objects of the "car" class, and by using the SMOTE algorithm [82] on the "pedestrian" and "bike" classes, to realign the number of samples for each class on the number of "trucks" in D_{pt}. The resulting refined training dataset is noted D'_{pt}.The ADAM optimizer was used with its default parameters, and a learning rate of 0.001. Moreover, following the results in Equation 5.13, a weight decay of 1e-5 was used on the linear parameters of the final Batch Normalization layers. The training was done during 400 epochs, and the selected model was the one that minimized the training loss, over the 400 epochs. Let y_{pi}, y_{bi}, y_{ci}, y_{ti} be binary indicators respectively indicating whether x_i belongs to the class "pedestrian", "bike", "car" or "truck". The loss function is a sum of cross-entropies:

$$
\begin{aligned}
-\Bigg[& \sum_{x_i \in D'_{pt}} (y_{pi} log P_{x_i}(P) + (1 - y_{pi}) log(1 - P_{x_i}(P))) \\
& + \sum_{x_i \in D'_{pt}} (y_{bi} log P_{x_i}(B) + (1 - y_{bi}) log(1 - P_{x_i}(B))) \\
& + \sum_{x_i \in D'_{pt}} (y_{ci} log P_{x_i}(C) + (1 - y_{ci}) log(1 - P_{x_i}(C)) \\
& + \sum_{x_i \in D'_{pt}} (y_{ti} log P_{x_i}(T) + (1 - y_{ti}) log(1 - P_{x_i}(T)))) \Bigg]
\end{aligned}
\tag{5.24}
$$

Pedestrian or not		**Bike or not**	
Accuracy	F1-score	Accuracy	F1-score
0.993	0.939	0.996	0.833
Car or not		**Truck or not**	
Accuracy	F1-score	Accuracy	F1-score
0.983	0.990	0.989	0.943

Table 5.2: Probabilistic classification results on D_{pv}

5.4.4 Evaluation

Probabilistic evaluation

First of all, the proposed multi-task MLP can be evaluated after training on the validation set D_{pv}, only using its initial probabilistic outputs. No Z-score filtering is used in this case, as this would not be meaningful with regards to the *Sigmoid* function S. An object is considered as classified into a class when the corresponding probabilistic output is higher than 0.5 (for e.g., if $P_x(C) > 0.5$, then x is classified as "car", otherwise it is classified as "not car"). In Tab. 5.2, the results for each classification task are given as accuracy scores and F1-scores. The results are satisfactory, as all these indicators are above 0.9 except the F1-score for the bike classification. This can be explained by the significantly lower number of "bikes" compared to the other classes, which justified the use of the SMOTE algorithm. The results for the car and pedestrian classes are still satisfactory, although under-sampling and oversampling were used on these classes.

Method	**IoU**	**Accuracy**	**F1-score on V**	**F1-score on W**	**F1-score on Ω**
Ours, probabilistic output with no Z-score filtering	0.312	0.729	0.890	0.408	0.377
Ours, evidential output with no Z-score filtering	0.320	0.733	0.890	0.412	0.388
Ours, evidential output with $ZMax = 2.58$	0.558	0.825	0.938	0.458	0.675
Ours, evidential output with $ZMax = 1.96$	0.682	0.872	**0.945**	0.570	0.786
Ours, evidential output with $ZMax = 1.65$	**0.725**	**0.897**	0.929	**0.661**	**0.825**
One-class SVMs [83]	0.507	0.661	0.672	0.556	0.660

Table 5.3: Classification results on D_{rc} ; V stands for "vehicle", W stands for "vulnerable road user", Ω stands for "unknown" object

Evidential evaluation

The evaluation of the evidential outputs generated from the MLP is done with regards to the $\Omega = \{V, W\}$ frame of discernment, with the mass functions generated from Equations 5.6, 5.18, 17 and 5.23. The *interval dominance* (ID) preference relation in [84], and depicted in Algorithm 1, is used to classify objects based on the mass values on V and W. The $ZMax$ value in Equation 5.15 is still to be defined. When working with Gaussian random variables, the three common thresholds to work with Z-scores are 2.58, 1.96, and 1.65, respectively

Method	Accuracy	F1-score (V)	F1-score (W)
Ours, $ZMax = 1.65$	0.914	0.959	0.890

Table 5.4: Results of the evidential classification on D_{pv}

corresponding 99%, 95% and 90% confidence levels [79]. The MLP is thus tested with those three possible $Zmax$ values.

The decisions based on the evidential mass functions generated from the MLP are compared with decisions based on its probabilistic outputs, and classification results obtained from a set of one-class SVMs [83]. One-class SVMs are commonly used when trying to detect unknown objects. Moreover, such SVMs can be trained and tested directly on the dataset of LIDAR objects that was created in the context of this work. For a fair comparison with the proposed multi-task MLP, four one-class SVMs are trained on the D_{pt} dataset. Each one of those four SVMs is trained on one class of D_{pt}: "car", "truck", "pedestrian" or "bike". The following classification rule is used to classify objects as vehicles or vulnerable road users from either the probabilistic outputs of the MLP, or the set of four one-class SVMs:

- If an object is classified as a pedestrian or a bike, or as both, and neither as a car nor as a truck, then it is classified as a vulnerable road user (W);
- Else, if an object is classified as a car or as a truck, or as both, and neither as a pedestrian nor as a bike, then it is classified as a vehicle (V);
- Otherwise, the object is classified as unknown (Ω);

To simulate a test on real-life conditions, the set of SVMs and the MLP with the corresponding classification rules are tested on $D_{rc} = D_u \cup D_{pv}$, the union of the validation dataset and the dataset of unknown objects. The results are presented in Tab. 5.3.

Based on the Intersection Over Union (IoU) scores, the best performing approach is the evidential classification with $ZMax$ equal to 1.65. This version is also the best on practically all the indicators, except the F1-score on V. The interest of Z-score filtering with an evidential formulation of a neural network is visible. Indeed, the worst performing approach is the purely probabilistic one, and the IoU scores increase with the $ZMax$ values. The Z-Score filtering scheme proved to be efficient, as the F1-score for Ω is equal to 0.825 when ZMax is equal to 1.65, although the system was never trained on those unknown objects. Vulnerable road users are still challenging to correctly classify though, as the best F1-score for W is only 0.661. This can again be explained by the fact that the original dataset was highly unbalanced. As seen on Fig. 5.4, using evidential classification with $ZMax = 1.65$ leads to the desirable feature that, on D_{rc}, all the wrongly classified vehicles and vulnerable road users were classified as unknown objects. Moreover, it is also to be noted that 81% of the vulnerable road users are correctly classified, and that the low F1-score for this class is explained by the 19% that are classified as unknown objects and the 8% of unknown objects

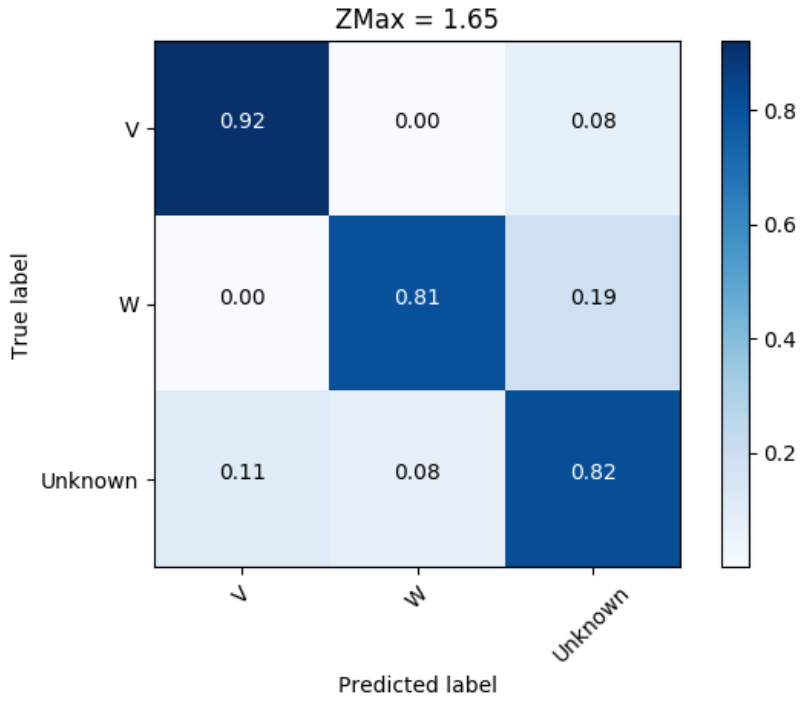


Figure 5.4: Normalized confusion matrix for evidential classification with $ZMax$=1.65 on D_{rc}

that are classified as vulnerable road users. This can also be seen in Tab. 5.4, which indicates the accuracy and F1-scores only computed on D_{pv}. The IoU score and F1-score for Ω are not reported as these values are not meaningful anymore. In this case, the F1-score for W is equal to 0.890, which is more satisfactory. What's more, given that the dataset was created in a semi-automatic fashion, it can be assumed that a certain amount of vulnerable road users were wrongly labeled, making it challenging to classify them. In the next section, we thus propose an evidential road mapping approach that relies on road detection in LIDAR scans, via neural networks, and the conversion of the classification results into evidential mass functions.

5.5 Examples

In the following section, we present some real-life examples where LIDAR objects were classified by our system, with a $ZMax$ value of 1,65.

Crossroad

Figure 5.5 presents a scans recorded in a crossroad. The yellow lines represent a pre-existing map of the area. Red points are points that belong to a LIDAR object, other points are colored according to the returned intensity. To better understand the scene, ground points were accumulated over different frames, but

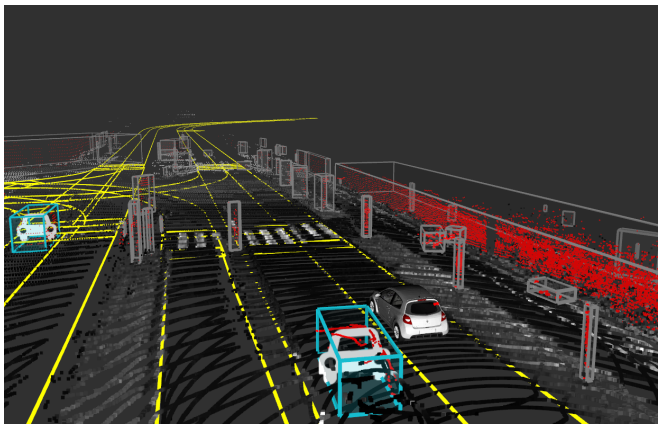


Figure 5.5: Crossroad with two vehicles

the obstacle points only correspond to a single scan. Grey objects are classified as unknown, blue ones as vehicles. On this scene, the classifier can detect, and reject, all the non pertinent objects, without having been trained on them, and the two vehicles are properly detected.

Example of pedestrian

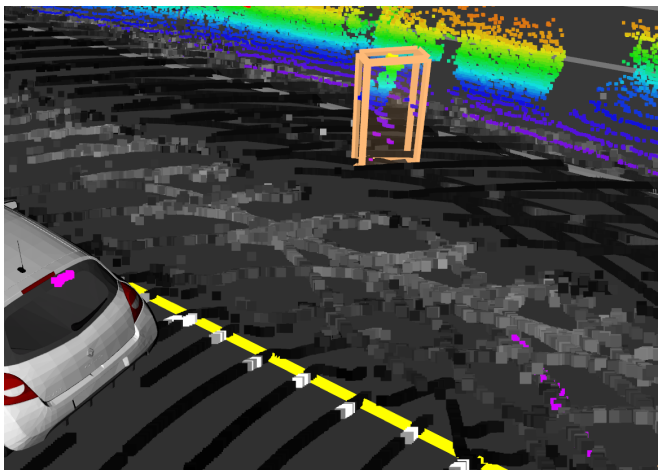


Figure 5.6: Pedestrian on a sidewalk

The orange object in Figure 5.6 is a pedestrian on a sidewalk, and is classified as a vulnerable road user. Points are colored according to their height to help the reader understand the geometry of the scene.

Handling clustering errors

In Figure 5.7, the clustering algorithm wrongly merged the vehicle in front of the autonomous platform with a portion of a wall on the side of the road. Such an incoherent object is also considered as unknown by the classifier. However, it was not explicitly trained to reject objects generated after clustering errors.

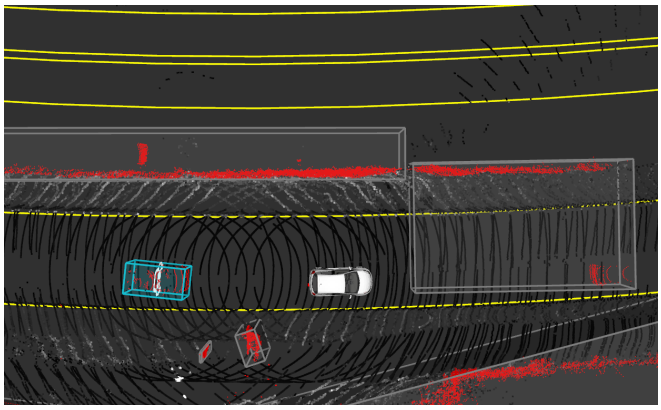


Figure 5.7: Misclustered object classified as unknown

5.6 Discussion on the use of unnormalized mass functions

5.6.1 Proper filtering of the simple mass functions

In the observations of Section 5.3, we observed that using unnormalized mass functions would be more significant, in the context of open-world classification. Yet, simply setting the mass value for the empty set to one was assumed to lead to an over sensitive filtering scheme, since such a mass function is the absorbing element of the unnormalized combination rule. We made some tests on our validation dataset, to confirm this assumption. We still used several $ZMax$ threshold to filter the simple mass functions that were expected to be irrelevant, with regards to the training set, and replaced them by $m(\emptyset) = 1$. We kept the I-D rule for the decision making but, since there's no difference between Ω and $\emptyset$ in our labeled unknown objects, objects that were not classified as vehicles or vulnerable road users were classified as unknown objects in the broad sense. We tested the same $ZMax$ thresholds as in Section 5.4.4, and report the corresponding confusion matrices in Figure 5.8.

We observe that even for a $ZMax$ value of 2.58, the results are significantly worse than those in Figure 5.4, as more objects are falsely considered as unknown ones. For $ZMax = 1.96$ values, nearly all the objects are classified as unknown, and for $ZMax = 1.65$, all the objects are considered as unknown. This confirms the expected over sensitivity of the classifier, when the filtered mass functions are replaced by $m(\emptyset) = 1$. A way to solve this issue would be to finely quantify the mass values on $\emptyset$, instead of setting its value to one when statistical outliers are detected. This is however a complex problem, as $\emptyset$ represents what is not modeled by the network, based on the frame of discernment. Yet, being able to estimate those masses on $\emptyset$ after the training, and during the deployment in open-world conditions, would probably lead to significantly better performances, and a finer representation of the knowledge for each object.

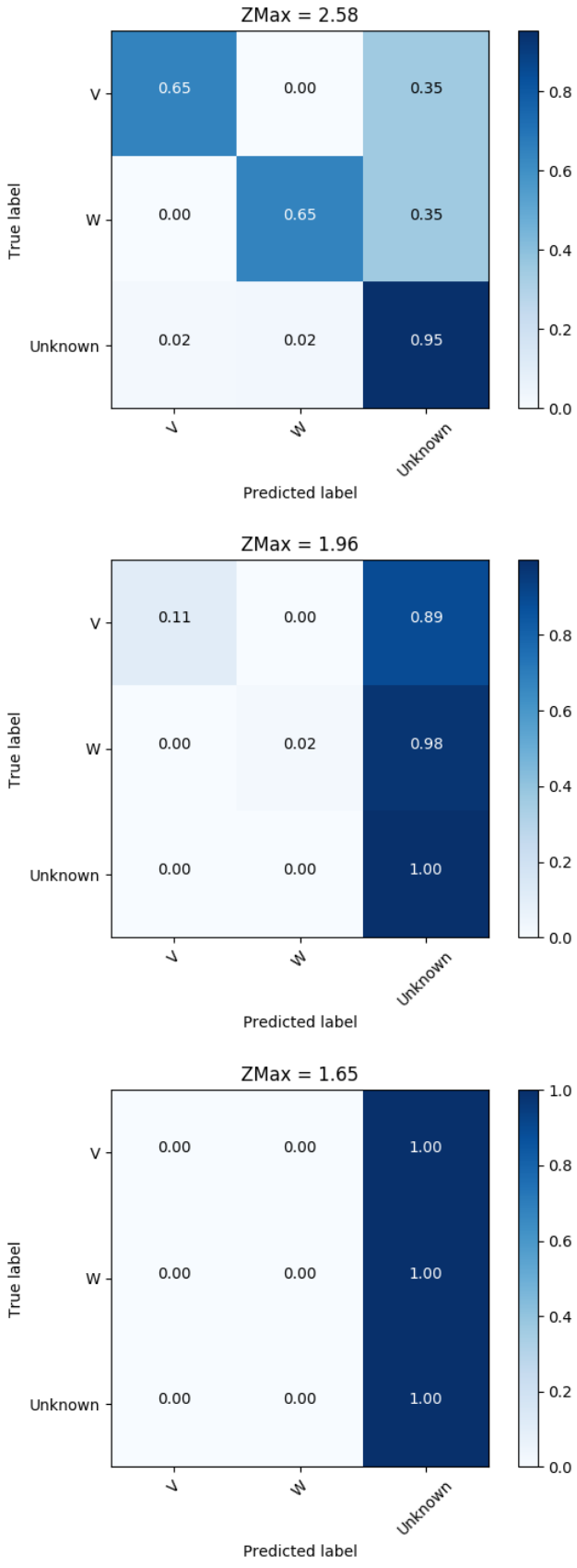


Figure 5.8: Results after having replaced the filtered mass functions by $m(\emptyset) = 1$

5.6.2 Representation of unknown objects

As previously said, the unknown objects in our labeled dataset are unknown in the broad sense. They were just considered as neither vehicles nor vulnerable road users. Nevertheless, by using unnormalized mass functions, a finer representation of the information available for these objects is possible. The mass on Ω could correspond, for instance, to objects that are "between" vehicles and vulnerable road users, such as motorbikes or electric scooters; the mass on the $\emptyset$ would then represent objects that are completely out of the frame of discernment. Evaluating such a behavior is hard, since very specific, and very subjective, labels would then be needed. This is one of the reason why we focused on unknown objects in the broad sense.

In Equation 5.23, we used the normalized version of Dempster's operator. We could have also used the unnormalized version, so as to have non-zero mass values on the empty set. As we were focusing on unknown objects in the broad sense, doing so would not change the final results we obtained, provided that we sticked to the I-D rule. Indeed, the belief and plausibility values in the normalized case can be obtained from their counterpart in the unnormalized case, by dividing them by $1 - m(\emptyset)$. The ordering between the belief values and the plausibility values are thus the same in the normalized and unnormalized cases. Therefore, the results would be the same, since we do not specifically classify objects as $\emptyset$ or Ω.

Yet, using the unnormalized combination rule, and having non-zero values on $\emptyset$, might have practical interests, for perception. One example that can be thought of, for instance, is to select region of interests in which additional information is needed. Objects that have high mass values for $m(\Omega)$ are ambiguous by definition, and thus, trying to extract more information about them, to confirm their class, makes sense. This would not theoretically be the case for objects with high mass values on $\emptyset$, as they are expected to be out of the frame of discernment that the system is designed for. Agile LIDAR sensors, that allow to change their scanning pattern on the fly, are currently emerging. It would make sense to use such LIDAR to increase the number of points that are returned, for objects associated with high mass values on Ω. It would be then possible to have a better understanding about them. A less sensor-

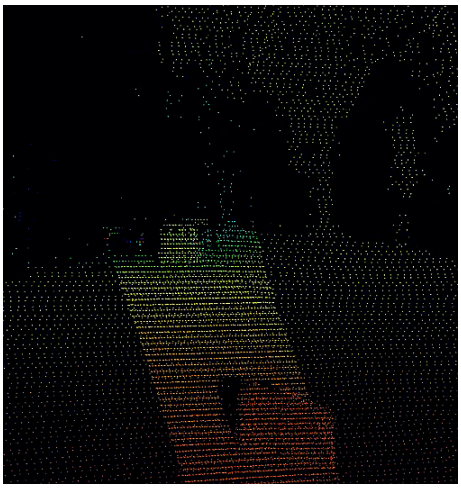


Figure 5.9: Example of scan from an agile LIDAR: the AEye iDAR. The scan was adjusted to be denser between selected angular positions, and coarser in other areas. This behavior can be adjusted on-the-fly.

dependent approach might also be to deploy specific algorithms to refine the knowledge about objects for which the mass value on Ω is high.

5.7 Conclusion

We presented a classifier of LIDAR objects, that takes geometrical features as inputs, and returns evidential mass values indicating whether the object is a vehicle, a vulnerable road user, or unknown. This is done by reinterpreting the classifier as a set of simple mass functions that are fused together. A simple statistical filtering scheme is used to account for objects that seem incoherent, with regard to the training set. Further work will have to benefit from the use of unnormalized mass functions, to improve the classification results and to allow for the implementation of new functionalities. These unnormalized mass functions might be of interest, for instance, in the context of active perception, where the perception system focuses on regions of interests that are selected on-the-fly. As the reinterpretation of GLR classifiers, as a fusion of simple mass functions, lead to interesting results for object classification, we considered that this technique might also be of interest in the context of evidential grid mapping. In particular, we investigated, in the following chapters, the possibility to create evidential grids that represent the road surface, from LIDAR scans only. We were aiming at improving the grid mapping system described in Chapter 4, especially regarding the processing of LIDAR scans. Indeed, many geometrical assumptions were made, which limited the applicability of this grid mapping system in a general context.

Chapter 6

Road detection in LIDAR scans

Contents

6.1 Introduction

When operating in urban or peri-urban areas, autonomous vehicles need an accurate representation of the road, as no other area is supposed to be drivable. In ideal conditions, geometrical models could detect the road in LIDAR scans, at the cost of a manual tuning of hyper-parameters, and a lack of flexibility. We instead propose to rely on refined versions of the SqueezeSeg and PointNet

architectures, to detect the road in 360°LIDAR scans. The road detection results can later be fused into evidential road grid, after having reinterpreted the output of the neural networks as a collection of simple evidential mass functions. Since road labels were needed for training, a soft labeling procedure relying on a localization error model, and lane-level HD maps, was used to generate coarse training and validation sets. An additional test set was manually labeled to properly evaluate the road detection results. The system was trained and evaluated on real-life data, and a suboptimal implementation of the road mapping and object detection algorithm maintains a 10Hz framerate. Our contributions in this chapter are then twofold:

- TadNet, a refined version of PointNet for road detection in LIDAR rings
- RoadSeg, a refined version of SqueezeSeg for road detection in LIDAR scans

6.2 TadNet: Transformation-adversarial network for point-level road detection in LIDAR rings

6.2.1 Ring-level PointNet

Typically, dense LIDAR sensors rely on stacked lasers that individually sweep the scene. A LIDAR ring represents a set of points that is obtained after the sweep of the environment by a single laser of a LIDAR. To detect the road in LIDAR scans, without having to transform the raw points into another representation, a classifier inspired by PointNet can be used. To leverage the limitations of PointNet that were exposed in Sec. 2.3.3, the processing is done at the ring level. Indeed, the maximum number of points that a LIDAR ring can include can be computed from the angular resolution of the LIDAR. Then, contrary to what was done in [13] and [25], no sampling of the point-cloud is needed. Moreover, LIDAR rings are often dense, especially at short range, since each laser sweeps the whole scene. This was expected to facilitate the reasoning of a PointNet-like network. However, LIDAR rings vary significantly among each others: a ring acquired by a top laser and a bottom laser include points that were acquired at very different distances. A training scheme, inspired by the recent successes of generative-adversarial networks (GAN) in the image domain [32], was proposed to cope with this issue.

6.2.2 Transformation-adversarial network for LIDAR rings

GANs rely on the conjunction of two alternatively trained systems. The first one, called the generator, is optimized to generate artificial samples that are as realistic as possible. The second one, called the discriminator, is trained to discriminate real and artificial samples. The two systems are competing against each other: the generator aims at fooling the discriminator, and the discriminator aims at detecting samples generated by the generator. Similarly, we propose a

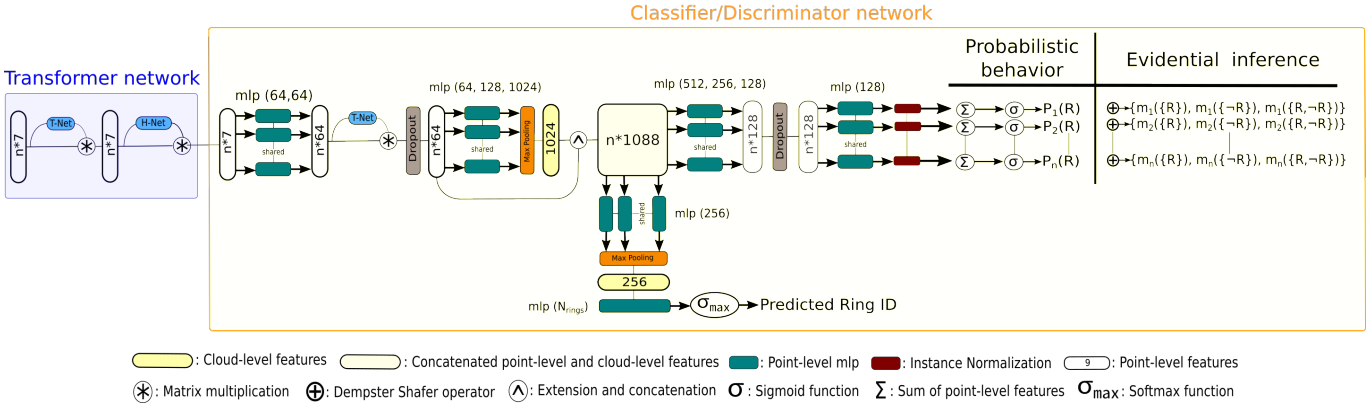


Figure 6.1: Transformation-adversarial network for road-detection in LIDAR rings

Transformation-adversarial network, or TAdNet, depicted in Figure 6.1. It is composed of a Transformation network (blue sub-network in Figure 6.1), and a Classification/Discrimination network (yellow sub-network in Figure 6.1). In the original PointNet, T-Nets predict affine transformation matrices applied to the whole input cloud, and to intermediate features extracted by point-level MLPs. Those T-Nets are optimized during the training, alongside the other parameters of the network.

The Transformation network that we propose, which also applies a transformation predicted by a T-Net to the input, is optimized separately from the rest of the system. To cope with the variability among LIDAR rings, the Transformation network also includes an H-Net. This H-Net (standing for *homothety network*) processes the transformed point-cloud obtained from the transformation predicted by the T-Net, and predicts an explicit rescaling factor, that is applied to the coordinates of all the points. The input points are represented by their Cartesian coordinates (x,y,z), spherical coordinates (ρ,ϕ,θ), and their intensity. To account for the risk of redundancy among the point features, the ϕ and θ angles are the uncorrected azimuth and zenith at which the point was acquired, while the Cartesian coordinates are obtained after correction. Let h be the scale predicted by the H-Net. Then, the coordinates of the input points are rescaled as follows: $x_* = hx$, $y_* = hy$, $z_* = hz$, $\rho_* = h\rho$. All the other features are left unchanged.

The Transformation network is then expected to learn to remap all the LIDAR rings into a constrained range, that is suitable for the road classification task. We assumed that it should be difficult to predict the ring ID of properly remapped and constrained LIDAR rings. The Transformation network is thus trained alongside a Classification/Discrimination network, and aims at generating similar LIDAR rings. This Classification/Discrimination network is in fact a multi-task PointNet, without any initial T-Net. It has to both perform road detection among the LIDAR points, and predict the ID of the LIDAR ring that it processes. This ring ID is predicted from the output of a small PointNet-

like subnetwork that is fed with the vector of concatenated point-level features and cloud-level features, that can be obtained after the max-pooling operation that every PointNet-like network uses. Following the results in Equation 5.13, Instance-Normalization and L2 regularization are used on the outputs, so that the α parameters lead to cautious evidential mass functions, with significant mass values that are assigned to the frame of discernment. Indeed, TadNet was intended to be used to perform evidential road grid mapping.

6.2.3 Training procedure

A PointNet-like system is typically trained with a multi-task loss. In the context of this study, the problem is point-level road detection in LIDAR rings. The loss chosen for this task, noted L_{ce} was the classical cross-entropy loss. The second component of the loss used for the training was a geometrical regularization loss, common with other PointNet-like networks. Let A be the transformation matrix predicted by the T-Net inside the Classification/Transformation network. This 64 by 64 matrix is more difficult to optimize than the simple transformation matrix predicted by the first T-Net, but should be as orthogonal as possible. Then, the loss on A to minimize, noted L_{geo}, is:

$$L_{geo}(A) = ||I - AA^T||^2 \tag{6.1}$$

Finally, the ring ID prediction error is again evaluated from a cross-entropy loss, calculated from the actual ring ID. We note this loss L_{ring}. Let L_{Tr}, the loss used to optimize the Transformation network, and L_{CD}, the loss used to optimize the Classification/Discrimination network. For each ring, let P_{road}, Y_{road}, P_{Ring} and Y_{Ring} be, respectively, the point-wise predicted probability that each point belongs to the road, the corresponding road labels, the predicted ring ID and the corresponding ring label. Then:

$$\begin{aligned} L_{CD} &= L_{ce}(Y_{road}, P_{road}) \\ &\quad + L_{ce}(Y_{Ring}, P_{Ring}) \\ &\quad + L_{geo}(A) \\ L_{Tr} &= L_{ce}(Y_{road}, P_{road}) \\ &\quad - L_{ce}(Y_{Ring}, P_{Ring}) \\ &\quad + L_{geo}(A) \end{aligned}$$

The whole system is trained thanks to the algorithm 2. Preliminary experiments, whose result are presented in Section 6.5, seemed to show that such a system would outperform regular PointNet networks. However, the resulting system was twice as long to train, when compared to PointNet, due to the adversarial training algorithm that we used. Moreover, its inference time was incompatible with real-time constraints, even with high end GPUs. We thus explored the possibility to instead propose a refined version of SqueezeSeg for road detection, that we call RoadSeg.

Algorithm 2 Training of the proposed system

```
Transformation network: T ;
Classification/Discrimination network: CD ;
N training rings are available ;
for e epochs do
    for N/n iterations do
        Sample n batches (b_0, .., b_t) from the training set
        for i in range(n) do
            b_i^* = T(b_i)
            RoadClassif, RingID = CD(b_i^*)
            Update CD from L_CD
        end for
        for i in range(n) do
            b_i^* = T(b_i)
            RoadClassif, RingID = CD(b_i^*)
            Update T from L_Tr
        end for
    end for
end for
```

6.3 RoadSeg: a deep learning architecture for road detection in LIDAR scans

To properly detect the road in LIDAR scans, we propose an architecture that is heavily inspired by SqueezeSeg [14] and SqueezeSegV2 [43], as those networks are particularly efficient to process LIDAR scans. However, we introduce several refinements to the original architectures and training schemes, so as to better fit the task of road detection, and allow for evidential fusion of segmented scans. We name the resulting architecture "RoadSeg".

6.3.1 Dense range image generation from LIDAR scans

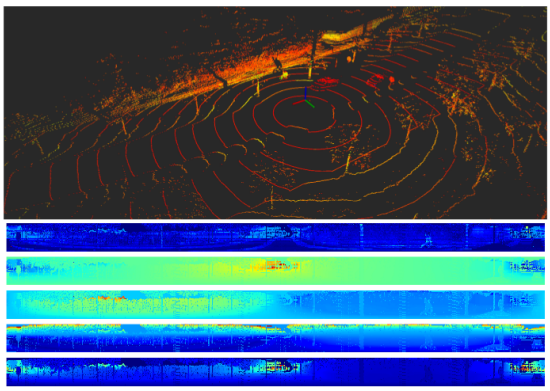


Figure 6.2: Conversion of a LIDAR scan into a multi-channel range image by spherical projection. The channels in the lower part correspond to the returned intensity, the Cartesian coordinates, and the measured range.

Similarly to SqueezeSeg, RoadSeg processes LIDAR scans that are stored into dense range images. Such a representation corresponds to raw sensor measurements, as LIDAR scanners actually return distances at fixed angular positions. The dimensions of those range images then depend on the specifications of the LIDAR that was used to generate them. Our dataset was generated with VLP32C sensors that were operating at 10Hz, and scanning at 360°. This sensor relies on 32 stacked lasers, each scanning at a different zenith angle, and has a horizontal angular resolution of 0.2°when operating at 10Hz. As such, only 32*360/0.2 angular positions can be observed by the sensor. Each LIDAR scan of the training, validation and test set can then be represented by a 32*1800*C grid, with C being the number of features available for each point. This grid can be considered as an image with an height of 32 pixels, a width of 1800 pixels, and C channels. Let (x, y, z) be the Cartesian coordinates of a LIDAR point. Let α, β be the indexes of the pixel that correspond to this point, and RingId the index of the laser that measured the point. A RingId of 0 indicates that the point was measured by the topmost laser, and a RingId of 31 indicates that the point was measured by the lowest laser. Then α and β are such that:

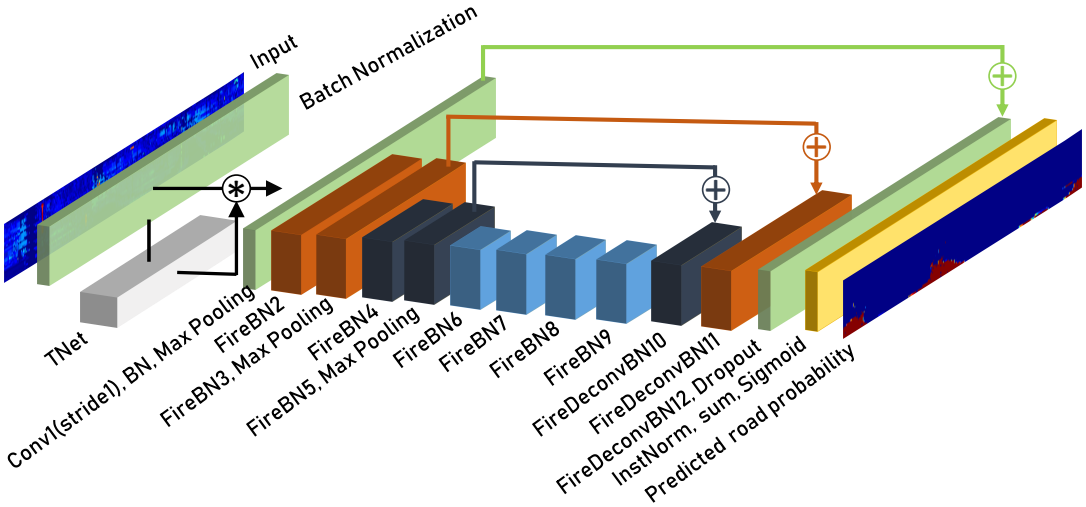
$$\alpha = arcsin(\frac{z}{\sqrt{x^2 + y^2 + z^2}}) * 5$$

$$\beta = RingId$$

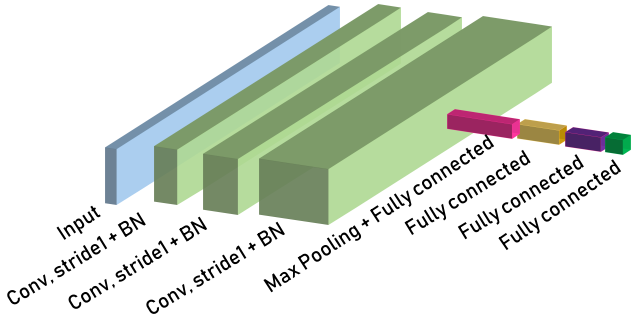
The channels of the pixel at the (α,β) position can then be filled with the features of the corresponding point. Figure 6.2 presents the resulting ranges images obtained after spherical projection of LIDAR points. In practice, the VLP32C scanner returns ethernet packets containing measured ranges that are already ordered by laser and azimuth position. The range images can then be obtained directly from the sensor, without having to pre-process a Cartesian LIDAR scan. This projection was used in the generation of the labeled data,

as the scans of the test set were easier to label in Cartesian coordinates, and the LIDAR points had to be localized in Cartesian coordinates before being projected into our HD maps. But when run on the vehicle, RoadSeg processes range images that are directly obtained from the ethernet packets.

In total, we chose to represent a LIDAR point by its Cartesian coordinates, its spherical coordinates, and its returned intensity. Additionally, a validity channel was added, as done for SqueezeSegV2. A point is valid, and the corresponding validity is equal to 1, if the range that was measured at its angular position is strictly positive. Otherwise, the point is not valid, which can correspond to missed-detections, or lost ethernet LIDAR packets.



(a) General road classification architecture



(b) T-Net of RoadSeg

Figure 6.3: RoadSeg, a refined SqueezeSeg-like architecture for road detection and evidential fusion

6.3.2 From SqueezeSeg to RoadSeg

Figure 6.3 depicts the RoadSeg architecture, which is very close to the original SqueezeSeg architecture, as it also relies on the use of Fire and Fire-deconvolutional

layers, and skip connections. RoadSeg also only downsamples horizontally, as SqueezeSeg and SqueezeSegV2. RoadSeg uses one downsampling step less than SqueezeSegV2. Indeed, the first convolutional layer of RoadSeg has a stride of 1 in both direction, while SqueezeSeg has a horizontal stride of 2. The kernel size is however still equal to 3. This is justified by the fact that, in the case of road detection, downsampling might make road hard to distinguish at long-range. SqueezeSeg benefited from this downsampling because it was originally designed for semantic segmentation of road users, which are often relatively close to the sensor. RoadSeg then also has one deconvolutional layer less than SqueezeSeg. Additionally, as done in SqueezeSegV2, RoadSeg uses Batch Normalization after each convolution, both inside and outside the Fire layers.

An initial Batch Normalization is also applied on the input of the network, as a normalizing operation. This mechanism was proposed to replace the manual normalization step, that is used in both SqueezeSeg and SqueezeSegV2. As the normalization parameters initially used in SqueezeSeg and SqueezeSegV2 are computed from the train set, manual normalization assumes that the sensor setup is the same in the train, validation and test sets. As our labeled LIDAR scans were not acquired by the same vehicles, this assumption does not hold anymore. Applying Batch Normalization to the inputs of RoadSeg is thus a way to train the network on inputs that are not exactly normalized. A T-Net, inspired by PointNet [13], can also be used to realign the LIDAR scans, by predicting a 3×3 affine transformation matrix that is applied on the Cartesian coordinates of the LIDAR points. The TNet is composed of three 1×1 convolutional layers, with respective output sizes of 32, 64 and 512 ; a channel-wise max-pooling; and three linear layers having output sizes of 256, 128 and 9. Finally, according to the results of Equations. 5.9 and 5.13, the final convolutional layer is replaced, in RoadSeg, by an Instance-Normalization layer and a sum over the output channels.

In order to limit the total number of weights in RoadSeg, the number of output channels produced by the Fire layers were reduced, with regards to their counterparts in SqueezeSeg and SqueezeSegV2. Table 6.1 compares the Fire layers in RoadSeg and in SqueezeSeg/SqueezeSegV2. The Fire-deconvolutional layers were left unchanged. The Conditional Random Field (CRF) layer that was, originally, refining the outputs of SqueezeSeg and SqueezeSegV2, was removed for several reasons. First, as it was originally implemented as a recurrent neural network, generating evidential mass functions from this layer was more complex than from a regular convolutional layer, or an Instance-Normalization layer. Secondly, the removal of this layer was a way to reduce the computational cost of RoadSeg. Finally, experiments showed than the CRF layer was degrading the performances of SqueezeSegV2, for the road detection task, and was thus assumed not to be suitable for RoadSeg.

	Output channels	
Layer	SqueezeSegV1/V2	RoadSeg
Fire2	128	96
Fire3	128	128
Fire4	256	192
Fire5	256	256
Fire6	384	256
Fire7	384	256
Fire8	384	256
Fire9	512	256

Table 6.1: Comparison of the fire layers in RoadSeg and SqueezeSeg

6.4 Automatic labeling procedure of LIDAR scans from lane-level HD Maps

Reliable semantic segmentation labels are usually generated manually, by expert annotators. This however proves to be expensive and time consuming. Automatic labeling can instead be used to generate labeled LIDAR scans, on which a road detection algorithm could be trained. Ideally, an error model should be associated to an automatic labeling procedure, as the resulting data is likely to include errors. We thus propose to softly label the road in LIDAR scans from pre-existing lane-level maps, and according to a localization error model.

6.4.1 Soft-labeling procedure

The automatic soft-labeling procedure used in the context of this work is presented in Figure 6.4. It is assumed that the LIDAR scans can be acquired in areas where reliable geo-referenced maps, with correct positions and road dimensions, are available. Moreover, the LIDAR scans are supposed to be acquired from a moving data acquisition platform, which includes a 360° LIDAR scanner that can perceive the ground and obstacles, and a GNSS localization system which returns an estimation of its pose in the map frame, and a corresponding covariance matrix. Those sensors are considered to be rigidly attached to the acquisition platform, and calibrated together. The coordinates in the map frame are expressed in terms of northing and easting offsets with regards to an origin. This origin is close to the data acquisition platform. Under the classical assumption that the localization error follows a zero-mean Gaussian model [85], an uncertainty ellipse can be obtained from the covariance matrix associated with the current pose. A probability of belonging to a mapped road can then be estimated, for each LIDAR point.

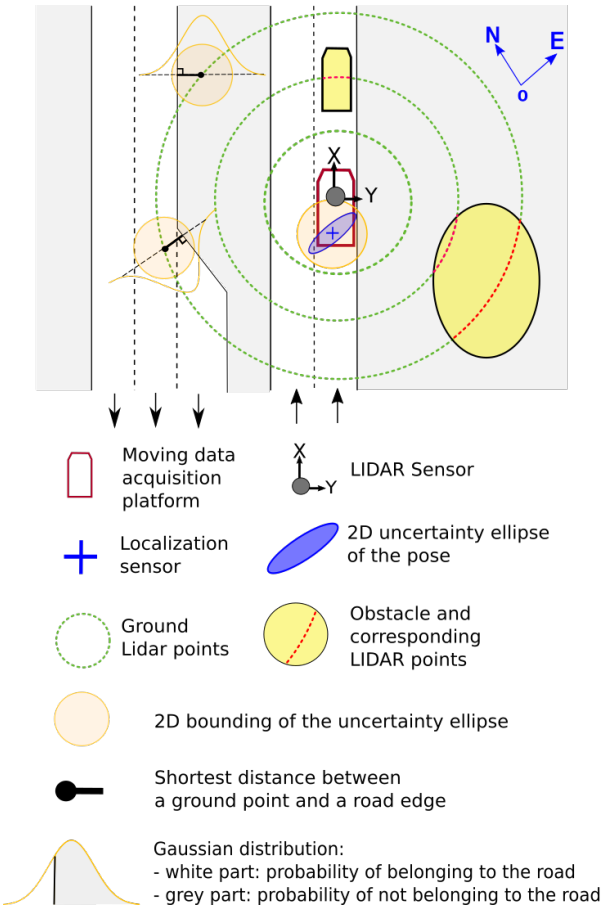


Figure 6.4: Automatic soft-labeling of LIDAR scans from road maps

Let x_i^l represent the coordinates a LIDAR point, and x_i^m the corresponding point after projection in the map frame. Let y_i, the labeled probability that x_i^l belongs to the road. First, a ground detection step is used to segregate obvious obstacles from points that potentially belong to the road. It is assumed that the algorithm does not falsely classify ground points as obstacles. If x_i^l does not belong to the detected ground, then $y_i = 0$, since it belongs to an obstacle. Otherwise, x_i^l is projected into the map frame. Given that the projection into the map frame relies on a rigid transformation, the localization uncertainty of the resulting x_i^m is equal to the uncertainty of the pose measured by the GNSS localization system. The closer from a road edge x_i^m is, the lower the probability that it belongs to the road is. Let d_i be the minimum distance between x_i^m and a mapped road edge. The covariance matrix given by the GNSS localization system could, ideally, be used to estimate the standard-deviation of the Gaussian distribution that models the error on d_i, by considering the heading of the road in the map frame [86]. However, the value of this heading can be ambiguous in curbs or roundabouts. To account for those use cases and facilitate the computations, a bounding of this standard-deviation, noted σ_b, is thus estimated instead. Let σ_n and σ_e be, respectively, the standard-deviation in the northing and easting

directions; then σ_b is estimated as follows:

$$\sigma_b = Max(\sigma_n, \sigma_e) + \gamma \tag{6.3}$$

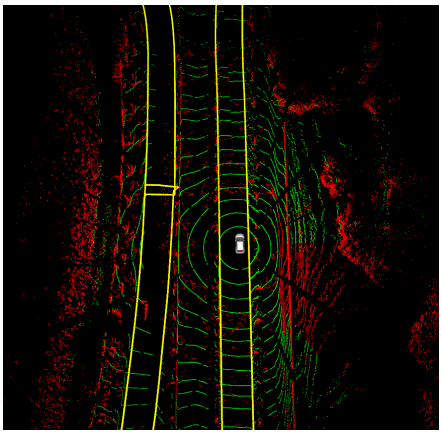
The γ term is an ad-hoc parameter, that is used to account for the uncertainty in the measures of the LIDAR sensor, and possible errors in the estimated extrinsic calibration parameters. Then, if x_i^l belongs to the ground, y_i can be estimated as follows: If x_i^m falls into a mapped road:

$$y_i = \int_{-\infty}^{d_i} \frac{1}{\sigma_b\sqrt{2\pi}} \exp^{-\frac{1}{2}(\frac{x}{\sigma_b})^2} dx \tag{6.4}$$

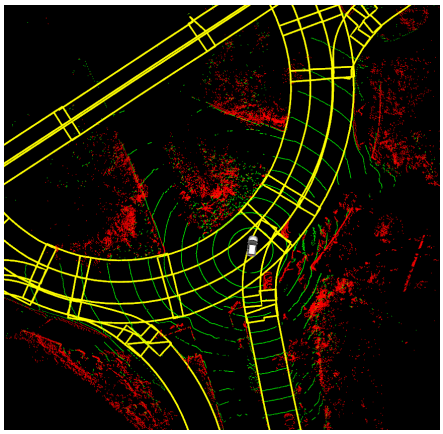
Otherwise:

$$y_i = 1 - \int_{-\infty}^{d_i} \frac{1}{\sigma_b\sqrt{2\pi}} \exp^{-\frac{1}{2}(\frac{x}{\sigma_b})^2} dx \tag{6.5}$$

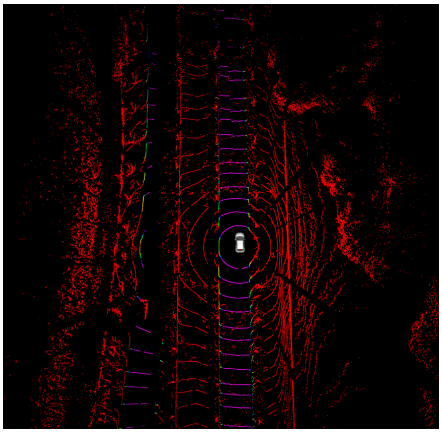
Figure 6.5 presents two use cases and the resulting softly labeled LIDAR scans.



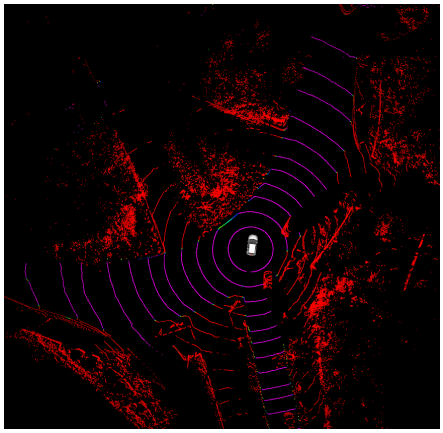
(a) Use case 1: two opposite lanes



(b) Use case 2: Roundabout



(c) Use case 1: Automatic labels



(d) Use case 2: Automatic labels

Figure 6.5: Use cases and examples of automatically labeled LIDAR scans. The data acquisition platform is depicted in white, and the map in yellow. Green points are pre-classified as ground points ; the purpler a point is, the higher its probability of belonging to the road is; the redder a point is, the lower its probability of belonging to the road is.

6.4.2 Data collection and resulting dataset

Appropriate data was needed to apply this automatic labeling procedure. Accurate maps and localization were needed. Open source maps, such as OpenStreetMap, were thus not considered, as the road dimensions, and especially their width, are rarely available and accurate. The NuScenes dataset [87] includes LIDAR scans and an accurate map ; however, the localization of the vehicle, though extremely accurate, is not provided with uncertainty indicators, which prevented us from using this dataset. Data was thus collected in several areas where Renault S.A.S has access to lane-level HD-map. Those maps follow the framework described in [88].

Data acquisition platform

ZoeCab platforms were used to collect data. In this work, a VLP32C Velodyne LIDAR running at 10 Hz was used with a Trimble BX940 inertial positioning system, coupled with an RTK Satinfo, so that centimeter-accurate localization can be achieved when RTK corrections are available. The other sensors were not used for the data collection. The PPS signal provided by the GPS receiver was used to synchronize the computers and sensors together.

(a) A ZoeCab platform (b) Velodyne VLP32C and GNSS antenna

Figure 6.6: Data collection platform for the generation of the automatically labeled dataset

Coarse training dataset for TadNet

2334 scans were then recorded, and coarsely labeled from our lane-level maps, in Guyancourt. These scans were actually originally used to create a first training/validation dataset for TadNet. The validation subset was obtained via a 70/30 split of this original data. The validation data only correspond to the earliest and latest fifteen percents of this initial dataset, to ensure that the validation is different enough from the training set.

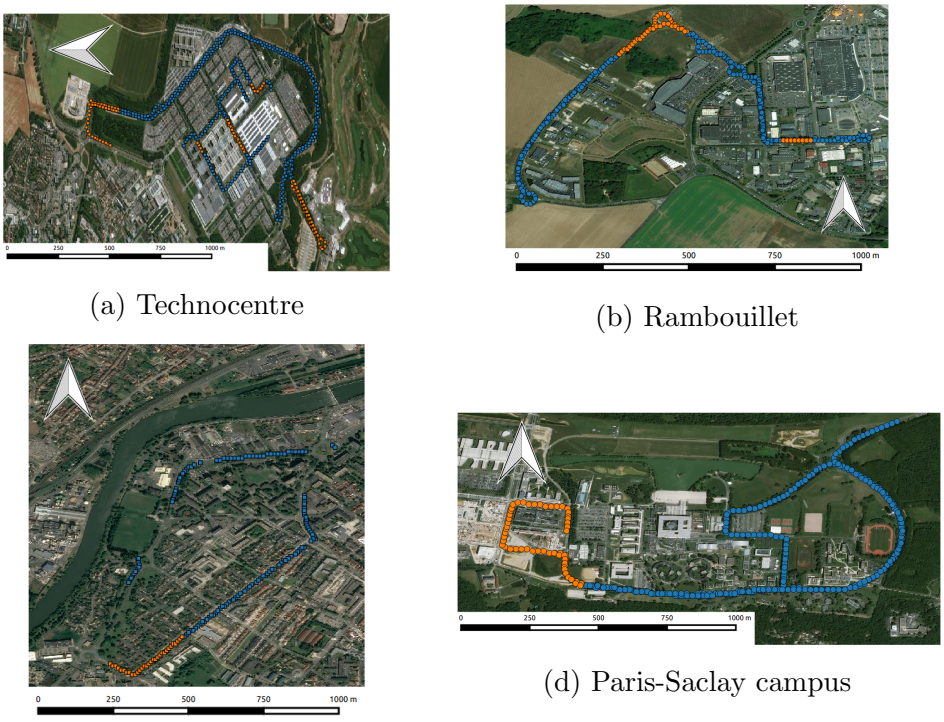
Coarse training dataset for RoadSeg

In order to get diverse training samples, LIDAR and localization data was acquired in four different cities and locations in France, with two different ZoeCab systems:

- Compiègne
- Rambouillet
- Renault Techocentre
- Paris-Saclay campus

One of the vehicles was used to collect the data in Compiègne and Rambouillet, and the other one was used in Renault Technocentre and the Paris-Saclay campus. All those locations are urban or peri-urban areas. The sensor setup was

similar, but not identical, between those vehicles, with a a VLP32C LIDAR and GNSS antenna on top of the vehicle and the Trimble localization system in the trunk. LIDAR scans were only automatically labeled every ten meters, to limit the redundancy in the training set. As an accurate localization was needed to label the LIDAR point-clouds from the map, the LIDAR scans were only labeled when the variance in the easting and northing direction, associated with the pose estimated by the localization system, were lower than 0.5m. The γ parameter in Equation 6.3 was empirically set to 10 cm. The initial ground detection was realized thanks to a reimplementation of the algorithm described in [72]. Figure 6.7 presents the areas in which data was collected. The total number of collected and labeled samples for each location is reported in Table 6.2.



(a) Technocentre

(b) Rambouillet

(c) Compiègne

(d) Paris-Saclay campus

Figure 6.7: Automatically labeled dataset, in four locations. Each point indicates a position where a LIDAR scan was automatically labeled, and the white arrows are oriented towards the north direction. The blue points correspond to the training set, the orange points correspond to the validation set.

	Technocentre	Rambouillet	Compiègne	Paris-Saclay Campus	Total
Number of samples	647	337	160	356	1500

Table 6.2: Number of automatically labeled LIDAR scans

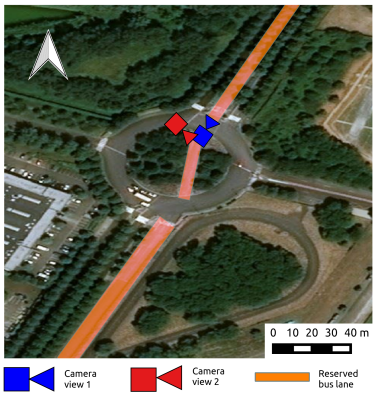
Test dataset for RoadSeg

Though a training and validation dataset were obtained, they would not enable road detection algorithms to be properly evaluated, as the obtained labels are not binary, and may still include errors. An additional test set, that would only be used to evaluate the performances of the road detection, was thus also created. As no open-source dataset uses a VLP32C LIDAR scanner, this test dataset was specifically labeled by hand for our work. The platform used to collect the data was the ZoeCab system that was used to collect the Renault Technocentre and Paris-Saclay datasets, and was driven in Guyancourt, France. Yet, the LIDAR sensor that was used to collect the test data was different from the one that was, previously, used to create the automatically labeled dataset. Indeed, we observed that some differences with regards to the returned intensity exist among different VLP32C LIDAR sensors, even if the sensors are properly calibrated. Additionally, the sensor was put five centimeters lower than the position that was used, previously, to collect the training and validation sets. 347 scans were manually labeled to create a test set to evaluate RoadSeg. Those 347 scans actually correspond to a subset of the dataset of the original dataset collected for TadNet. The manual labeling procedure thus consisted in, manually ,refining the coarse labels that were already obtained, from the lane-level maps of the ZoeCab systems. Figure 6.8 presents the locations where the test dataset was collected.

Figure 6.8: Generation of the test dataset in Guyancourt. Each green point indicates a location where a manually labeled LIDAR scan was recorded.

A specificity of the Guyancourt area is that there exist reserved lanes for buses that are physically separated from the rest of the road. Those lanes might have unique and particular set-ups. Figure 6.9 presents for instance a bus lane that goes through a roundabout, while the other vehicles have to drive in circles. In Figure 6.9b, it can be observed that the bus lane was separated from the other lanes before the roundabout. In Figure 6.9c, it can be seen that the part of the bus lane that goes through the roundabout does not have the same texture as the other parts of the road. As those bus lanes are very particular, and might have a different texture from the rest of the road, they were labeled as belonging to a *Do not care* class. In our manual labels, we considered that classifying those bus lanes as roads was not relevant, but not an error per se, as they are still

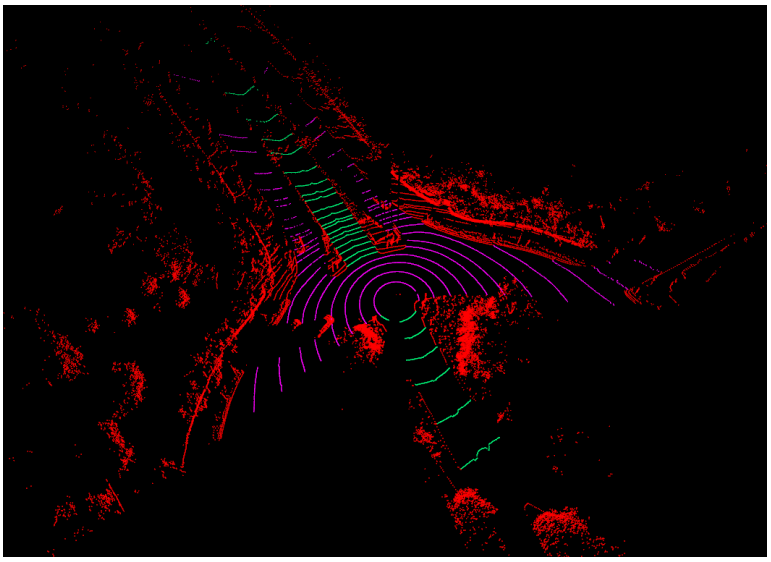
technically roads. They were then not considered in the evaluations done on the Test set.



(a) Example of situation in the test set, with a reserved bus lane that goes through a roundabout

(b) Camera view 1 ; the orange mask represents the reserved the bus lane

(c) Camera view 2 ; the orange mask represents the reserved the bus lane



(d) Corresponding manually labeled LIDAR scan. Purple points are labeled as *road*, red points as *obstacle*, and green points as *do not care*, since they belong to a reserved bus lane.

Figure 6.9: Example of reserved bus lane in Guyancourt that goes through a roundabout

6.5 Evaluation of the performances of TadNet

TadNet was thus first evaluated on our original Guyancourt coarse dataset, which consisted in 2334 automatically labeled scans recorded in Guyancourt. TadNet was not kept in our following experiments, because the inference time that it needed to process a full LIDAR scan was approximately of 140ms on a TitanX GPU, which made it incompatible with real-time operation. Moreover,

	All labels		0-1 labels	
Model	F1-score	Accuracy	F1-score	Accuracy
PointNet [13] - *ring*	0.868	0.973	0.907	0.983
PointNet [13] - *scan*	0.899	0.980	0.933	0.988
TAdNet - *ours*	**0.933**	**0.987**	**0.959**	**0.993**

Table 6.3: Classification results for PointNet on LIDAR scans and rings, and for the proposed TAdNet, on the validation set

its training can take up to two days on a TitanX GPU, making the finetuning of the system, and the exploration of new architecture refinements, extremely hard in practice.

We report the classification results in Table 6.3. Three systems were evaluated: the proposed Transformation-Adversarial Network (TAdNet), a ring-level PointNet, and a scan-level PointNet, to quantify the interest of the refinements introduced with TAdNet. The point-level MLPs were following the original architecture proposed in [13], with a ReLU activation function and systematic use of Batch Normalization. The three systems were implemented in PyTorch. The two PointNets consisted in exactly the same layers as TAdNet, except for the H-Net and the ring-ID prediction subnetwork that were removed. Instance Normalization was still used, as the resulting systems were all intended to be used for model-free evidential road-grid mapping. The Adam optimizer [89] was used for the three networks, with a learning rate of 0.0001. Empirical observations showed that, instead of only applying L2-regularization to the final layer of the networks, applying it to all the parameters led to better numerical stability. Then, a weight-decay of 0.0001 was applied to all the parameters of the three networks, except for the parameters of the Transformation-network in TAdNet, on which a weight-decay of 0.00001 was applied. All the T-Nets and the H-Net were initialized with identity transformations. TAdNet and the ring-level PointNet were trained on mini-batches including 64 rings, and the scan-level PointNet was trained on mini-batches of 2 scans, as each scan was composed of 32 rings.

We report F1-scores and accuracies on the full validation dataset, and on only the 96.5% of 0-1 labels. Indeed, due to our soft labeling procedure, some labels were not binary. In the case of non-binary labels, a point was considered to be labeled as a road-point if its labeled probability was higher than 0,5. And a point was considered to be classified as road if the predicted probability was higher than 0,5. Table 6.3 reports the respective results of those approaches in the validation set. All approaches have satisfactory results, even if TAdNet outperforms all the approaches in all the indicators. The interest of the rescaling performed by TAdNet is obvious, as the ring-level PointNet is by far the worst performing approach, while TAdNet outperforms the scan-level PointNet, even

though it only processes rings.

As RoadSeg was a more efficient network, all the further developments were made from this architecture. The performances of the RoadSeg are mainly evaluated on our manually labeled test set. The KITTI dataset is also used to give some preliminary, though limited, results, as it is one of the most commonly used dataset for road detection.

6.6 Evaluation of the performances of RoadSeg

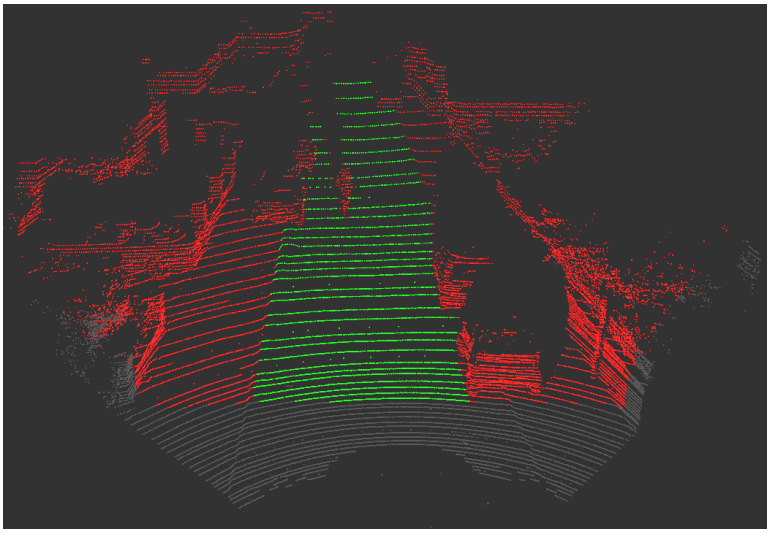
6.6.1 Evaluation on the KITTI Road dataset

Contrary to other approaches that are evaluated on the KITTI road dataset, we evaluate ourselves at the scan level, and do not aim at predicting the presence of road in areas that are not observed by the LIDAR sensor. No publication to the KITTI road benchmark was then possible. The labels of the KITTI road dataset are only given in an image plane corresponding to a camera whose intrinsic calibration parameters are available, alongside extrinsic calibration parameters with regards to a Velodyne HDL64 LIDAR, which was synchronized with the camera. The image road labels can then be projected into the LIDAR scans, to create ground-truth for road detection in LIDAR scans. Each point was then represented by its Cartesian coordinates, its reflectance, the range measured by the LIDAR sensor, and its validity. The scans are then represented as $6 \times 64 \times 512$ images.

The TNet of RoadSeg only predicts transformations for the Cartesian coordinates, and left the ranges and intensities unchanged. The LIDAR scans can only be labeled partially, because the camera view only covers a section of the corresponding LIDAR scan. Moreover, some LIDAR points can easily be mislabeled. Indeed, the labels in the KITTI road dataset are not accurate, especially around road objects, and synchronization and calibration errors between the sensors exist in the KITTI dataset. Figure 6.10 displays an example of labels obtained from the KITTI dataset. Short range points are unlabeled, most of 360°scan is excluded from the labeling procedures, and the image labels around the cyclist and the parked vehicles are noticeably coarse. Public labels are only available for the 289 scans that are present in training set of the KITTI road dataset.

Given the small number of labeled scans that is available in the KITTI dataset, a 5-fold cross-validation procedure was used to estimate the performances of RoadSeg. RoadSeg was compared with both SqueezeSegV2, and SqueezeSegV2 without its CRF layer. Each model was trained for 200 epochs on 4 of the folds. The training procedure was the same for the three models, and followed the one originally used by SqueezeSegV2, except that the batch size was set to 16 for all the models. The training was repeated until each model was trained on every

(a) Labels from the KITTI road dataset in the image plane



(b) Labeled LIDAR scan by projection of the image labels. Points that do not intersect the field-of-view of the camera are not displayed. Green points are labeled as *road*, red points as *not road*, and grey points are *unlabeled*.

Figure 6.10: Generation of LIDAR labels from the KITTI road dataset.

possible combination of folds. The 5 folds were the same for each model. After the 200 epochs, the selected parameters for each model correspond to those that maximize the F1-score over the 4 training folds. Only the points that are both valid and labeled are considered in the loss function and by those metrics. We considered that a point for which a probability of belonging to the road was strictly higher that 0.5 was predicted as being a road point. Finally, average F1, Recall, Precision and IoU scores were computed from the results of each model on the corresponding test folds, and reported in Table 6.4. We also report maximum and minimum scores over the 5 test folds, alongside the average execution time of each model on an NVidia TitanX GPU.

	Precision			Recall			F1-score			IoU			Inference time
Model	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Mean
SqueezeSegV2 without CRF	0.927	0.957	0.888	0.911	0.967	0.835	0.918	0.943	0.879	0.849	0.891	0.784	**16.3 ms**
SqueezeSegV2 with CRF	0.928	0.942	**0.899**	0.907	0.964	0.830	0.917	0.946	0.873	0.847	0.897	0.775	17.8 ms
RoadSeg	**0.941**	**0.974**	0.825	**0.976**	**0.994**	**0.965**	**0.957**	**0.978**	**0.891**	**0.920**	**0.958**	**0.803**	30.8 ms

Table 6.4: Comparison of the fire layers in RoadSeg and SqueezeSeg

The amount of data that is available on the KITTI dataset is probably inadequate for proper training, and thus limits the the relevance of those evaluations. The limitations of the projection procedure also prevent us from considering that those results are completely reliable. However, they tend to show that RoadSeg vastly outperforms SqueezeSegV2 for road detection. This however comes with an increased mean inference time. Yet, RoadSeg still processes LIDAR scans at a high rate, with a mean inference time of 30.8 ms. We also observe that the CRF layer of SqueezeSegV2 seems not to improve the performances in road detection. This may mean that using a CRF alongside SqueezeSegV2 may require other hyperparameters, or kernels, for road detection. This could also mean that the CRF layer struggles with large objects, such as the road, as it aims at locally smoothing the segmentation results. More reliable results are given in the next section, as the models are evaluated on our manually labeled dataset, after having been trained on the automatically labeled dataset that was generated from our HD Maps.

6.6.2 Evaluation on the manually labeled Guyancourt dataset

Classification performances

To give more significant results, we report metrics on our manually labeled test set. All the approaches were only trained thanks to the 1500 automatically labeled LIDAR scans that were recorded in Compiègne, Renault Technocentre, Rambouillet and the Paris-Saclay Campus, and thus have never been trained on the area where the manually labeled data was recorded. The training/validation split was exactly the one presented in Figure 6.7, and all the approaches were trained with the same procedure, which is close to the one used for the KITTI dataset except for some details. The batch size was reduced to 10 for all the approaches. This was needed because, while the scans from the KITTI dataset only cover the front view, our training set is composed of 360°scans. This also justified to modify the behavior regarding the padding of the feature maps processed by the networks. Indeed, the left and right sides of the inputs actually correspond to neighboring areas. The left (respectively right) padding is then obtained by mirroring the right (respectively left) side. We tested several variants. First of all, SqueezeSegV2, SqueezeSegV2 without CRF, RoadSeg, and RoadSeg without TNet have been trained on the whole automatically labeled dataset, and each point was represented by the eight available features (Cartesian coordinates, spherical coordinates, intensity and validity). We also trained four additional and lighter networks.

- RoadSeg-Intensity: the points are only represented by their intensity, ele-

vation angle, and validity.

- RoadSeg-Spherical: the points are only represented by their spherical coordinates, and their validity
- RoadSeg-Cartesian: the points are only represented by their Cartesian corrdinates, and their validity
- RoadSeg-Cartesian without TNet: like RoadSeg-Cartesian, but without the TNet

RoadSeg-Intensity and RoadSeg-Spherical do not include TNets. Indeed, TNets normally predict an affine transformation matrix for Cartesian coordinates. Yet, in the case of RoadSeg-Intensity and RoadSeg-Spherical, only spherical features are used. A spherical transformation could be predicted by a modified TNet. Yet, since the elevation and azimuth angles are unique for each position in the range image, such a transformation would be equivalent to an image deformation, which could complexify the further convolutions used in RoadSeg. RoadSeg-Intensity processes the elevation angle, alongside the intensity, to cope with the individual behavior of each LIDAR laser. Indeed, the intensity might be inconsistent among each LIDAR laser. Each model was trained ten times on the training set, to account for the possible variability in the results due to the random initialization of the network. At each training session, the parameters that were kept were those that maximize the F1-score on the validation set. We again report, in Table 6.5, average F1, Recall, Precision and IoU scores reached by all the models on the test set. We also report maximum and minimum scores alongside the average execution time of each model on an NVidia TitanX GPU. As the test set is manually labeled, and considered as reliable, it is a proper way to evaluate what was learnt by the networks from the automatically labeled dataset.

	Precision			Recall			F1-score			IoU			Inference time
Model	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Mean
SqueezeSegV2 without CRF	**0.880**	**0.910**	**0.860**	0.798	0.827	0.778	0.836	0.859	0.795	0.719	0.731	0.660	12,3 ms
SqueezeSegV2 with CRF	0.861	0.899	0.832	0.792	0.806	0.780	0.825	0.835	0.805	0.702	0.717	0.674	14,1 ms
RoadSeg with TNet	0.856	0.880	0.820	0.859	0.910	0.798	0.856	0.880	0.836	0.749	0.785	0.646	33,4 ms
RoadSeg without TNet	0.827	0.855	0.794	0.890	**0.9316**	**0.8668**	0.857	0.875	0.843	0.750	0.778	0.711	11,7 ms
RoadSeg-Intensity	0.769	0.801	0.747	0.808	0.887	0.765	0.787	0.811	0.753	0.649	0.682	0.603	**11.5** ms
RoadSeg-Spherical	0.857	0.897	0.825	0.878	0.904	0.841	0.866	0.883	0.850	0.764	0.741	**0.790**	11.6 ms
RoadSeg-Cartesian with TNet	0.853	0.891	0.816	**0.893**	0.916	0.860	0.872	**0.887**	0.859	0.773	**0.798**	0.754	33,1 ms
RoadSeg-Cartesian without TNet	0.867	0.890	0.832	0.885	0.918	0.857	**0.876**	0.886	**0.866**	**0.779**	0.796	0.763	11,6 ms

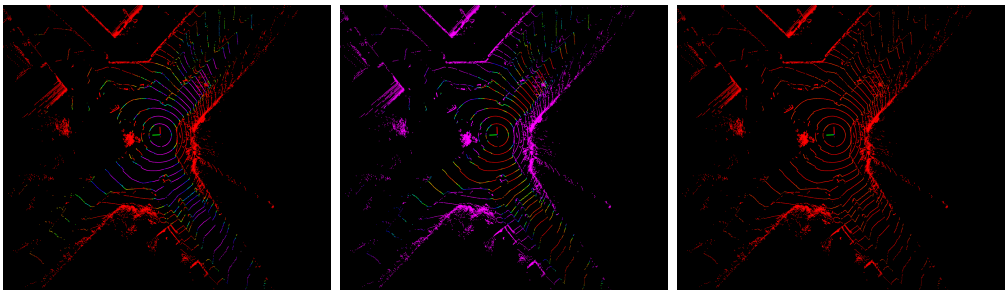
Table 6.5: Comparison of variants of RoadSeg and SqueezeSegV2

Interestingly, SqueezeSegV2 without CRF seems to have the best precision on the test set. Yet, this is compensated by its poor performances in terms of recall. We can assume that this is mainly due to the additional sub-sampling that it uses with regards to RoadSeg: SqueezeSegV2 cannot properly process points at long-range, and considers a significant amount of remote road points as obstacles. We again observe that the use of the CRF degrades the performances for road detection. RoadSeg, when used without the TNet and trained on the full

set of features, outperforms SqueezeSegV2, while being slightly faster. The use of the TNet does not seem relevant when training on the full set of features, as it doubles the inference time while slightly degrading the performances, except for the precision scores. Another interesting result is that RoadSeg-Spherical and both versions of RoadSeg-Cartesian outperform the networks that are trained on the full set of features. The best F1-score and IoU are even reached by a specific instance of RoadSeg-Cartesian that relies on a TNet. However, RoadSeg-Cartesian without TNet is faster, and easier to train since it has best F1-score and IoU in average. This network thus seems to be the best trade-off for road-detection. However, the performances reached so far are still relatively low. This can be explained by the automatic labeling procedure that we used to generate the train set, as it is very sensitive to unmapped roads, localization errors and improper obstacle filtering. This results also highlight the difficulties for RoadSeg and SqueezeSeg to process numerous features for a given point, as the best performing approaches only process a limited number of features. A possible improvement might then consist in fusing the results from several networks that process different sets of features.

Evidential fusion of neural networks

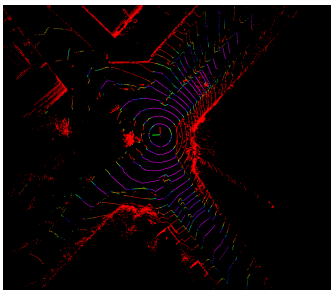
RoadSeg has been designed to allow for the generation of evidential mass functions from its outputs. A straightforward way to fuse several RoadSeg-like networks could then be to rely on an evidential fusion. We thus propose to use the model described in Equation 5.6 to generate evidential mass functions for each LIDAR point, from a set of neural networks. The resulting mass functions can then be fused, for each point, thanks to Dempster's rule of combination that is described in Equation 3.4. Figure 6.11 displays examples of evidential mass functions that can be obtained from all the variants of RoadSeg that are expected to be independent. Those mass functions are directly obtained after the training, from the biases of the final Instance-Normalization layer, and without any optimization of the α vector. The plausibility transformation in Equation 3.15 can then be used to compute the fused probability that each points belongs to the road, and the results can then be evaluated again on our test set. Given to the properties of plausibility transformation [57], fusing the evidential mass functions obtained from the different networks via Dempster's rule, and then applying plausibility transformation on the resulting mass function, is equivalent to a Bayesian independent opinion poll fusion among the probabilities obtained from each network. This means that a classification which purely relies on a Bayesian interpretation of the probabilities, obtained from the networks, only depends on the sum of the values of the α vector in Equation 5.2, which is supposed to be equal for all the possible α vectors. As we have chosen to consider that LIDAR points are classified as belonging to the road when the probabilities that are predicted by the network are higher than 0.5, we are in this case. We thus report in Table 6.6 recomputed F1-score, Precision, Recall and IoU obtained after the fusion of different sets of networks. To prevent data incest,



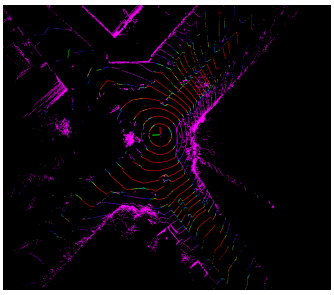
(a) $m(\{R\})$ from RoadSeg-Cartesian without TNet

(b) $m(\{\neg R\})$ from RoadSeg-Cartesian without TNet

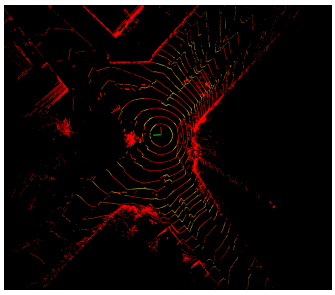
(c) $m(\{R, \neg R\})$ from RoadSeg-Cartesian without TNet



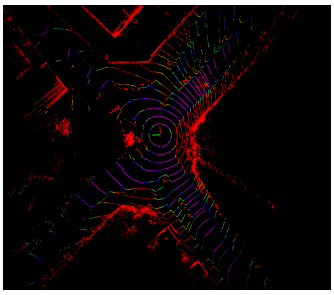
(d) $m(\{R\})$ from RoadSeg-Cartesian



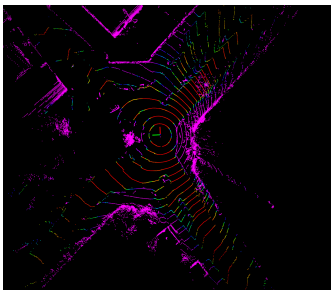
(e) $m(\{\neg R\})$ from RoadSeg-Cartesian



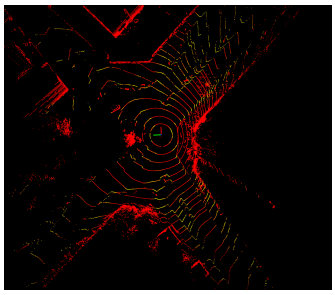
(f) $m(\{R, \neg R\})$ from RoadSeg-Cartesian



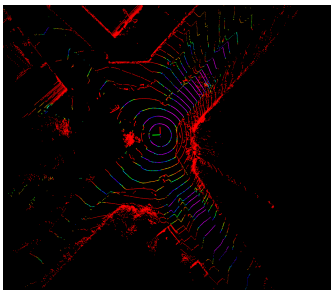
(g) $m(\{R\})$ from RoadSeg-Spherical



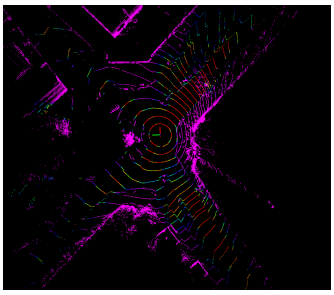
(h) $m(\{\neg R\})$ from RoadSeg-Cartesian



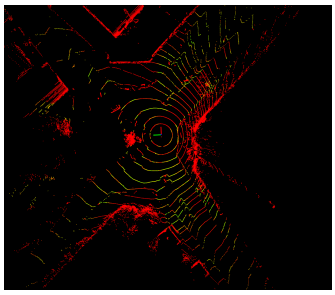
(i) $m(\{R, \neg R\})$ from RoadSeg-Spherical



(j) $m(\{R\})$ from RoadSeg-Intensity



(k) $m(\{\neg R\})$ from RoadSeg-Intensity



(l) $m(\{R, \neg R\})$ from RoadSeg-Intensity

(m) Color map

Figure 6.11: Example of evidential mass functions, obtained from the weights optimized after the training, from all the variants of RoadSeg that can be fused together

only RoadSeg-Intensity, RoadSeg-Spherical, RoadSeg-Cartesian and RoadSeg-Cartesian without TNet are considered. We also do not fuse RoadSeg-Cartesian and RoadSeg-Cartesian without TNet together, again to prevent data incest. Since the networks are then not trained jointly, a set of parameters has to be chosen for each one, among the ten that are available after the evaluations that were previously done. For a fair comparison, the parameters that were selected for each network correspond to the best ones in terms of F1-score on the validation set. Indeed, they are not necessarily the best performing ones on the test set. We thus only report one Precision, Recall, F1-score and IoU for each combination.

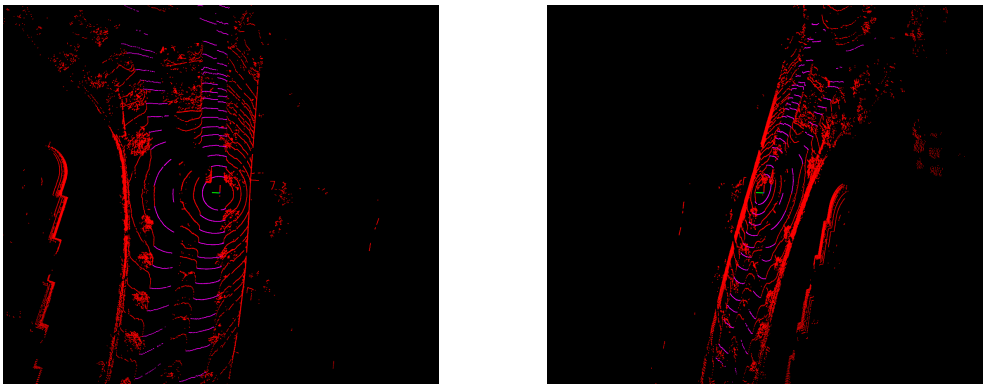
Intensity	Spherical	Cartesian	Cartesian (w/o TNet)	Precision	Recall	F1-score	IoU
✓	✓			0.8986	0.8588	0.8782	0.7829
	✓	✓		0.8932	**0.8991**	0.8962	0.8119
	✓		✓	0.9114	0.8740	0.8923	0.8055
✓		✓		0.8869	0.8830	0.8849	0.7936
✓			✓	0.8920	0.8501	0.8705	0.7707
✓	✓		✓	**0.9178**	0.8694	0.8929	0.8066
✓	✓	✓		0.9098	0.8904	**0.9000**	**0.8182**

Table 6.6: Evaluation of the results of several fusion schemes among RoadSeg networks

Fusion leads to significantly better Precision and F1-scores and IoU scores, with Recall scores that are in par with the best ones that are obtained from single networks. Fusion then prevents false-positives from happening, without significantly increasing the false-negative rate. Especially, the fusion of RoadSeg-Intensity, RoadSeg-Spherical and RoadSeg-Cartesian leads to the best results. The obtained road detection is satisfactory, as the F1-score is equal to 0.9, and the IoU is higher than 0.8. This is significantly better than the performances of each individual network, which seems to confirm that, originally, RoadSeg did not have enough parameters to properly detect the road from all the available features. This result moreover is particularly satisfactory because the training and validation labels were obtained automatically.

The use of a TNet improves the results, but that comes at the cost of an increased inference time, as observed in Table 6.5. We however consider that the use of the TNet is relevant, when using evidential fusion. Indeed, the risk of data incest among RoadSeg-Spherical and RoadSeg-Cartesian exists, because the Cartesian coordinates can be obtained from the Spherical ones, and vice-versa. As Dempster's rule of combination is designed to fuse independent mass functions, this risk of data incest should better be limited when operating in real-life conditions. As the TNet transforms the input to the fire modules by an affine transformation, this dependence between RoadSeg-Cartesian and RoadSeg-Spherical is less likely to happen. To support this claim, we observe that just fusing RoadSeg-Spherical and RoadSeg-Cartesian leads to the second best results in terms of F1-score and IoU, while the fusion of RoadSeg-Spherical

and RoadSeg-Cartesian without any TNet is significantly less accurate. We also show in Figure 6.12 a LIDAR scan belonging to the test set, and the point-cloud obtained after transformation by the TNet used in the evidential fusion. The transformation predicted by the TNet is normally applied to a point-cloud that was normalized by batch-normalization, but for the sake of clarity, we apply it on the original point-cloud. Indeed, the normalized point-cloud is, by definition, more compact, and harder to visualize. We can see that, in this case, the TNet mainly applies rotations around the x and z axes. The global alignment of the scan is thus modified. The initial Batch-Normalization layer of RoadSeg-Spherical could also be seen as a global affine transformation applied to the scan. Yet, it is extremely unlikely that both a RoadSeg-Cartesian using a TNet and a RoadSeg-Spherical actually apply the same affine transformation to their input. The TNet can then be considered as a way to enforce independence among RoadSeg-Cartesian and RoadSeg-Spherical.



(a) Original test scan (b) Test scan transformed by the TNet

Figure 6.12: Effect of the TNet on a LIDAR scan. Purple points were manually labeled as road points, red ones as not belonging to the road. The two pictures correspond to the same field of view and scale.

6.6.3 Comparison of the evidential mass functions obtained from the fused RoadSeg networks

As a reminder, RoadSeg networks are trained as SqueezeSegV2 was originally on the KITTI object dataset, and thus use a weight decay of 0.0001, which was also needed by the results displayed in Equation 5.13. We propose to compare the different evidential mass functions that can be obtained from the fusion of RoadSeg-Cartesian, RoadSeg-Spherical and RoadSeg-Intensity, as this is our best performing model, so as to verify their behavior depending on how the α vector is obtained. Especially, we focus on the points of the test set that are misclassified by this fusion of networks. A desirable behavior would be to have uncertain mass values generated for those falsely classified points. We chose to rely on the decomposable entropy for the evidential theory, defined in [90], to quantify

Set(s) used for post-processing					
None	train	validation	additional_train	additional_test	Mean Entropy on misclassified points
✓					0.3537
	✓				0.3663
	✓	✓			0.3664
			✓		0.3676
				✓	0.3668
	✓	✓	✓		0.3669
	✓	✓		✓	0.3664
	✓	✓	✓	✓	0.3668

Table 6.7: Comparison of the mean evidential entropies that were generated, for misclassified points, by RoadSeg-Intensity, RoadSeg-Spherical and RoadSeg-Cartesian

the uncertainty level of the mass functions obtained from the fused RoadSeg networks. This entropy is similar to Shannon's entropy, especially in the sense that an uncertain evidential mass function will lead to a high entropy. In our case, this entropy measure H on a mass function m is computed as:

$$\begin{aligned} H(m) = &- (m(\{R\}) + m(\{R, \neg R\}))log_2(m(\{R\}) + m(\{R, \neg R\})) \\ &- (m(\{\neg R\}) + m(\{R, \neg R\}))log_2(m(\{\neg R\}) + m(\{R, \neg R\})) \\ &+ m(\{R, \neg R\})log_2(m(\{R, \neg R\})) \end{aligned} \tag{6.6}$$

We report in Table 6.7 the mean entropy on the misclassified points, from evidential mass functions obtained by solving the minimization problem described in Equation 5.8 on several sets. We compare those values with their equivalent obtained without post-processing of the weights. The considered sets for the post processing were the training set, the training and validation sets, a collection of 2221 unlabeled random LIDAR scans that were acquired at the same locations as the training and validation sets, and a collection of 695 unlabeled scans acquired in Guyancourt alongside the test set. The latest sets correspond to a tenth of the whole sequences that were recorded to make the training, validation and test set. To ensure variety among the scans, the difference between the timestamps of those unlabeled scans is at least of one second. Those sets are denoted respectively as *additional_train* and *additional_test*.

The lowest mean entropy on the misclassified points corresponds to the vanilla results, when no post-processing is used on the weights of the networks. This is thus the most over-confident case. However, all the values of mean entropy are extremely close among each other. Interestingly the maximum entropy is achieved when only *additional_* train is used for post-processing. The use of data similar to the test set does not seem particularly useful and, strangely, the use of bigger sets did not necessarily lead to more cautious evidential mass functions. As a conclusion, evidential mass functions can be generated only using the training and validation sets, and potentially additional similar and unlabeled data. The

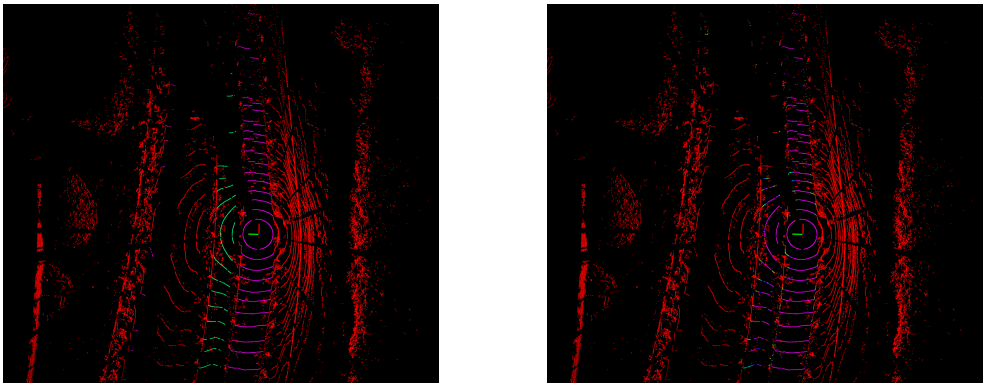
use of Instance Normalization and weight decay is confirmed as a way to obtain near-optimal evidential mass functions during the training, as no differences are perceptible in our case in terms of classification results, while the differences in terms of entropy are barely visible. It could even be considered that no post-processing should be used, since the dataset on which it should be computed is not clear, but this should be confirmed by further experimental and theoretical analyses.

6.6.4 Examples of results

We now present some segmentation results from the test set, that were generated from the fusion of RoadSeg-Cartesian, RoadSeg-Spherical and RoadSeg-Intensity.

Best F1 score

We first show the scan for which the fused networks reach the best results, in terms of per-scan F1-score. On this situation, the fused networks achieved an F1-score of 0.9569, and an IoU of 0.9173. As displayed in Figure 6.13, this is a very simple situation with a straight road, and no other vehicles, although a small part of the scan is missing. The fact that the system can handle such simple situations is reassuring. We also point out the fact that the reserved bus lane, labeled in green, was fully classified as road. Following the policy we previously exposed, those points corresponding to the bus lane were not considered in the evaluation.



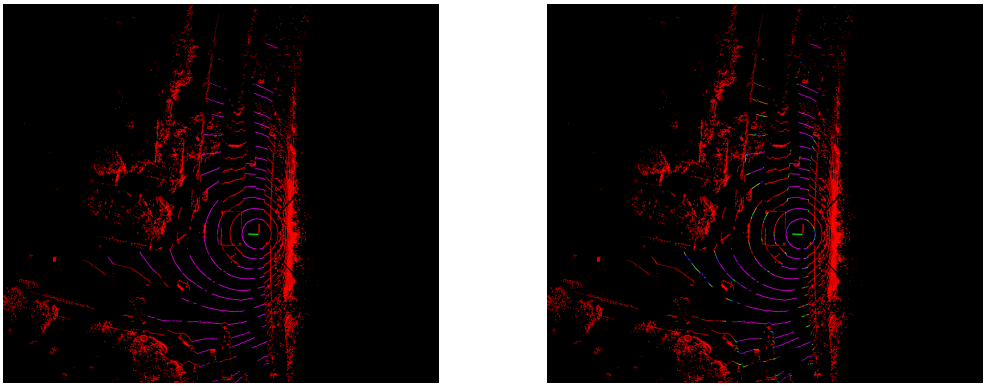
(a) Labels (b) Predicted road probabilities

Figure 6.13: Scan for which the network achieves the best F1-score. On the left side, purple points were manually labeled as road; red ones as obstacles; green ones as do not care, as they correspond to a reserved bus lane. On the right, the purpler a point is, the higher the probability of being a road point is high.

Worst F1 score

We show here the scan for which the fused networks achieve the worst results in terms of F1-score and IoU. The F1-score for this scan is equal to 0.8971, and the

IoU is equal to 0.8133. The errors are mainly localized on the left side, because the central median partially occluded this area. The results on the ego-lane are still satisfactory.

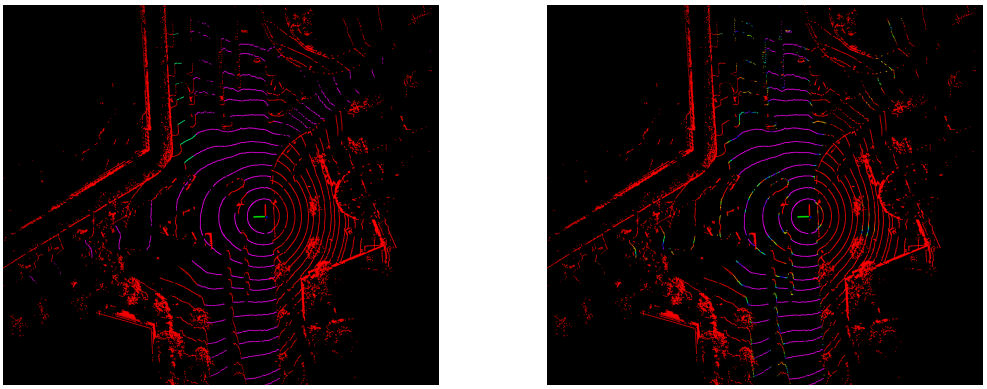


(a) Labels (b) Predicted road probabilities

Figure 6.14: Scan for which the fused network achieve the worst F1-score.

Crossroad

We present a result at a crossroad. Most of the road surface is classified as road. Narrow roads are undetected. This is because those roads are hard to distinguish when projected into the range images processed by the RoadSeg networks. The network achieves an F1-score of 0.9271 and an IoU of 0.8641 on this scan.

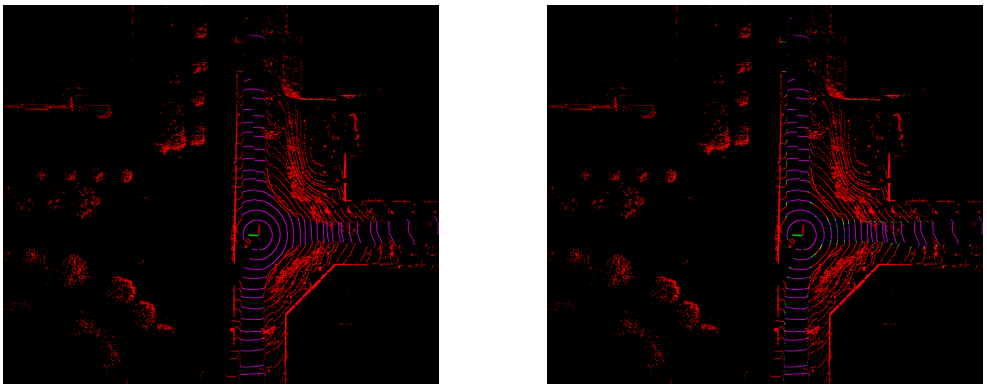


(a) Labels (b) Predicted road probabilities

Figure 6.15: Example of result at a crossroad.

Junction

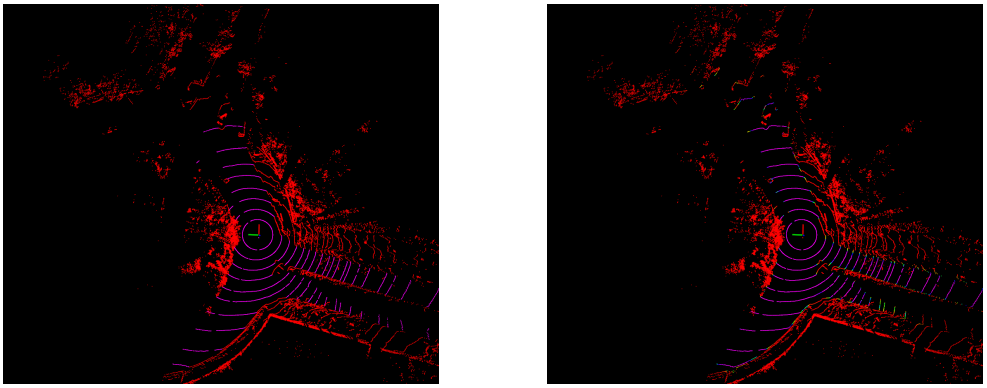
To counterbalance the results on the previous use case, we present results obtained at a junction. Again most of the road surface is properly detected. However, the networks consider that the entrance to the road on the right is narrower than what it actually is. This is because of our label generation procedure, which relied on a map. Indeed, many of the junctions in those maps were mapped similarly, which under-estimated entrance width that were considered to be equal to the roads' length. The fused networks still achieve an F1-score of 0.9323 and IoU of 0.8732 on this scan.



(a) Labels (b) Predicted road probabilities

Figure 6.16: Example of result at a junction.

Roundabout



(a) Labels (b) Predicted road probabilities

Figure 6.17: Example of result at a roundabout.

We conclude with a roundabout. The results are very satisfying, as most of the actual road surface is properly detected. The remote vehicle is also properly considered as an obstacle. The central median in front of the vehicle is however partially considered as belonging to the road. The fused networks achieve an F1-score of 0.9010 and an IoU of 0.8199 on this scan.

6.7 Conclusion

The fusion of RoadSeg-Cartesian, RoadSeg-Spherical and RoadSeg-Intensity leads to very promising results, especially in straight roads and roundabouts. Junctions and crossroads can also be processed with a certain efficiency, although the network is limited by what was present in the training set. Indeed, very few crossroads were automatically labeled, and the approximative representation of the junctions in the maps that we used influence the final results of the network. Hopefully, the fact that these results have been achieved with a relatively small training set, that was automatically labeled, indicates that there is probably room for improvement. Additional and finer training data would certainly correct the behavior of the fused networks. As TadNet and RoadSeg were originally designed to be used for evidential road mapping, we propose in the next section to focus, again, on evidential grid mapping. We propose a LIDAR only evidential road surface mapping and object detection algorithm, that could potentially be extended with the other works of this thesis, and that relies on our road detection results.

Chapter 7

Application of RoadSeg: evidential road surface mapping

Contents

7.1 Introduction

From the satisfactory results that can be obtained by fusing several RoadSeg networks, and evidential mass functions that can be generated from their outputs, an usable representation of the road for a navigation algorithm can be created, by fusing consecutive road detection results. We chose to rely on an evidential grid mapping framework. Inspired by the work in [63], in which it is observed that the conflict induced by the fusion of evidential grids can correspond to moving objects, we propose an evidential road mapping algorithm, that generate both a grid depicting the actual road surface, and a list of moving road obstacles. We consider that the area below a moving road obstacle should be considered as road, and that the road grid should only depict the reality of the road, independently of the presence of obstacles. Figure 7.1 depicts the whole algorithm, and Figure 7.2 presents a possible output of the system. In the next sections, we present its different steps in details.

7.2 Projection on the xy-plane of the segmentation results

As an evidential fusion of RoadSeg-Intensity, RoadSeg-Spherical and RoadSeg-Cartesian leads to the best classification performances, the algorithm processes the segmentation results obtained after evidential fusion of the three networks.

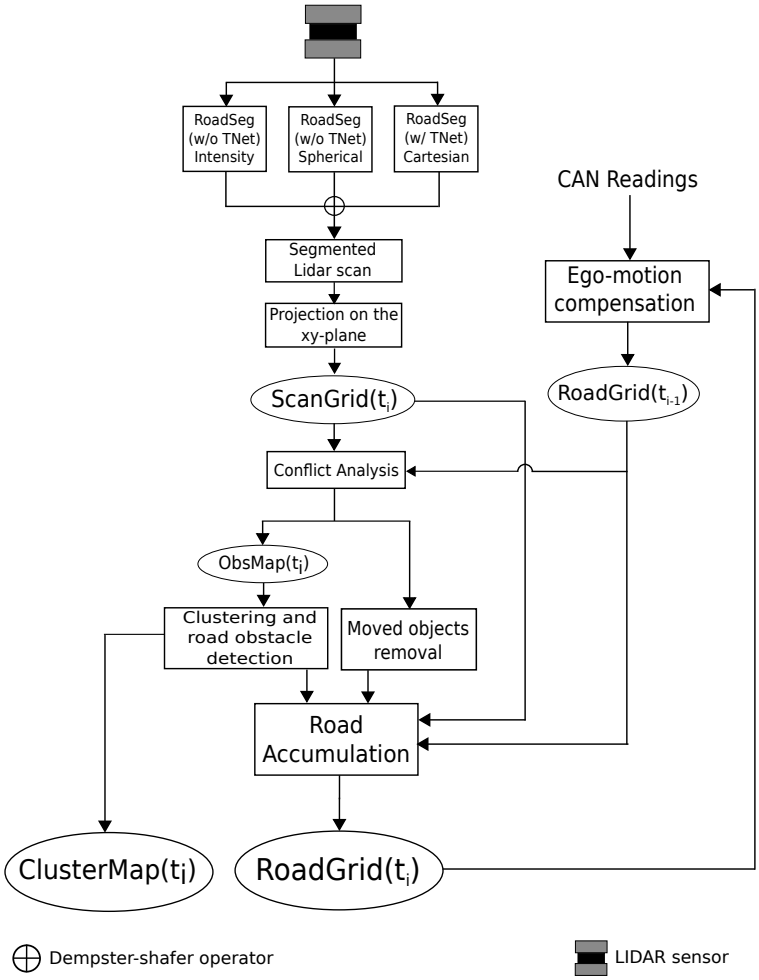


Figure 7.1: General evidential road mapping and road object detection algorithm

We use the original weights obtained after the training, without any post-processing, for the generation of evidential mass functions. We arbitrarily chose to make the road grid correspond with the xy-plane, in the reference coordinate system used by the LIDAR. This plane is split into equally sized grid cells, which cover a pre-defined area around the sensor. The state of each cell of index i can be represented by three evidential mass values $m_i(\{R\})$ (road), $m_i(\{\neg R\})$ (not road) and $m_i(\{R, \neg R\})$ (unknown). Similarly to what is done at the LIDAR point level, those evidential mass values respectively quantify the evidence towards the fact that the i^{th} cell belongs to the road, does not belong to the road, or is in an unknown state. A straightforward way to compute $m_i(\{R\}), m_i(\{\neg R\})$ and $m_i(\{R, \neg R\})$ is to project, into the xy-plane, all the LIDAR points, and the evidential mass values that are obtained after the fusion of the RoadSeg networks. Then, $m_i(\{R\}), m_i(\{\neg R\})$ and $m_i(\{R, \neg R\})$ can be obtained by fusing the mass values of the points projected into the grid-cell i, thanks to the Dempster's rule of combination. To reduce the computational complexity of this projection and fusion step, each grid cell should be processed in parallel. For the sake of clarity,

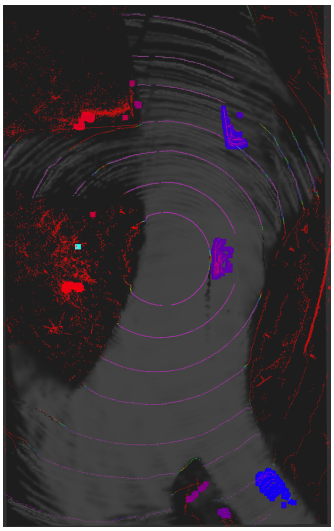


Figure 7.2: Results obtained from the road mapping and object detection pipeline. The LIDAR scan classified by the three RoadSeg network is visible. Below, a greyscale RoadGrid represents the belief for $m(\{R\})$ in each cell. The clustered objects (mainly vehicles on the road) are colored according to their cluster id.

we drop the cell-index i. The number of points projected into each grid-cell is unknown, and varies over time and for each cell of the grid. To solve this issue, we rely on the rewriting of the Dempster-Shafer operator in terms of *commonality functions* [52].

For $\Omega = \{R, \neg R\}$ our binary frame of discernment, a commonality value $Q(A)$ can be computed from a mass function m for each element $A \in 2^{\Omega}$, as follows:

$$Q(A) = \sum_{B \supseteq A} m(B) \tag{7.1}$$

The evidential mass function m can be recovered from the commonality values, as follows:

$$m(A) = \sum_{B \supseteq A} (-1)^{|B|-|A|} Q(B) \tag{7.2}$$

Commonality functions can be used to fuse n evidential mass functions into a fused mass function m_{res} as follows:

1. Compute Q_1,...,Q_n from the n mass functions, using Equation 7.1
2. For each $A \in 2^{\Omega}$, $Q_{res}(A) = \exp(\sum_{j=1}^{n} ln(Q_j(A)))$
3. Compute m^*_{res} from Q_{res}, the unnormalized version of m_{res} using Equation 7.2
4. Normalize m_{res} as follows: $\forall A \in 2^{\Omega} \setminus \{\emptyset\}$, $m_{res}(A) = K m^*_{res}(A)$ with $K = 1/(1 - m(\emptyset))$; $m(\emptyset) = 0$

This procedure is equivalent to consecutively applying the Dempster's rule of combination among the n evidential mass functions. However, this formulation enables the projection and fusion operations to be reinterpreted as an operation on a 2D histogram.

The log-commonalities associated to each point can trivially be computed in parallel, after the fusion of the results generated by the three RoadSeg networks. If n corresponds to the number of points that are projected into a grid cell, and whose evidential mass functions have to be fused, $\sum_{j=1}^{n} ln(Q_j(A))$ can be computed by histogramming the x and y coordinates of each point, and by weighting the samples by the corresponding log-commonalities. The evidential mass values associated to each cell can then be recovered for each cell. We call the resulting evidential grid, which was only generated from a single scan, a *ScanGrid*.

7.3 Conflict analysis

In order to generate a dense representation of the road, ScanGrids have to be fused over time. Let a *RoadGrid* be an evidential grid that has been obtained by accumulating several previous ScanGrids. A RoadGrid is supposed to only represent the road surface, without considering objects that might stand on the road. The latest ScanGrid is noted ScanGrid(t_i), and the latest RoadGrid available is noted RoadGrid(t_{i-1}). Let m_{t_i} be the evidential mass function that correspond to a given cell of ScanGrid(t_i), and $m_{t_{i-1}}$ the evidential mass function of the cell of RoadGrid(t_{i-1}) that is at the same position. A naive way to accumulate ScanGrids would be to use, again, the Dempster-Shafer operator to fuse all the m_{t_i}s with the corresponding $m_{t_{i-1}}$. Yet, as depicted in the Figure 7.3, this could lead to a catastrophic accumulation of objects over time, that would affect the estimated road surface, without corresponding to actual objects. The current LIDAR scan is depicted in green and red. Green points are classified as road points, and red points as obstacles (not road). Under the scan, an evidential grid corresponding to the simple accumulation of ScanGrids is depicted. White cells are classified as road cells ($m(\{R\}) > 0.5$), black cells as obstacle cells ($m(\{\neg R\}) > 0.5$), and the grey ones are in an unknown state ($m(\{R, \neg R\}) > 0.5$). We can observe that the vehicles that are driving on the road create artifacts, and cells that intersect their trajectories are falsely considered as not belonging to the road surface. Consequently, it must be ensured that the objects that stand on the road, and are potentially moving, are not falsely fused with the RoadGrid. We introduce two frame of discernments: $\Omega_{obs} = \{O, \neg O\}$ and $\Omega_{displaced} = \{D, \neg D\}$. The first one models the presence of road obstacles that do not belong to the road surface, as the O proposition. The second one models the fact that a previously present road obstacles is no longer present, as the D proposition. This case typically corresponds to a vehicle that was static on the road, while the RoadGrid was being generated, and started moving.

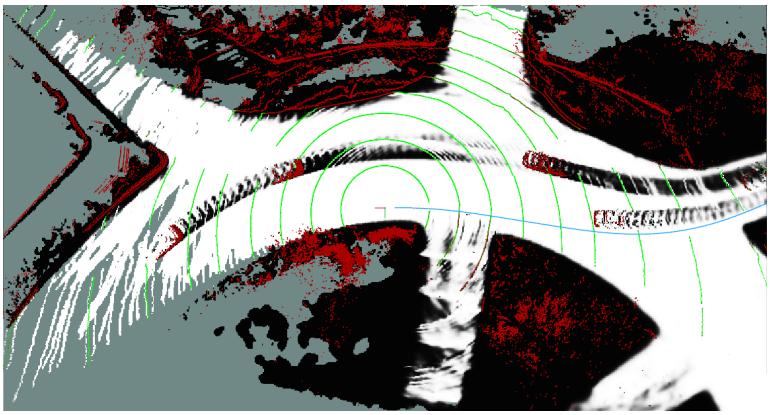


Figure 7.3: Example of RoadGrid obtained by accumulating ScanGrids without considering the objects that stand on the road. White cells have an $m(\{R\})$ value higher than 0.5, black cells have an $m(\{\neg R\})$ value higher than 0.5, grey cells have an $m(\{R, \neg R\})$ value higher than 0.5.

Ω_{obs} can be used to detect which cells of the ScanGrid(t_i) should not be fused with RoadGrid(t_{i-1}). The evidential mass functions in this frame of discernment can be computed, for each cell, from the conflict between the $m_{t_{i-1}}$ and m_{t_i} mass functions. Indeed, a high value for both $m_{t_i}(\{\neg R\})$ and $m_{t_{i-1}}(\{R\})$ can indicate that a moving road obstacle is currently observed in the corresponding ScanGrid(t_i) cell, and thus should not be fused with RoadGrid(t_{i-1}). However, it could also indicate that the neural network has trouble detecting a given road, meaning that the corresponding cells should instead be fused. Let m_{obs} be the evidential mass function associated to a cell of ScanGrid(t_i) under the Ω_{obs} frame of discernment. We propose to compute m_{obs} as follows:

$$m_{obs}(\{O\}) = \alpha(\overline{Z}) * m_{t_{i-1}}(\{R\}) * m_{t_i}(\{\neg R\}) \tag{7.3a}$$

$$m_{obs}(\{\neg O\}) = 0 \tag{7.3b}$$

$$m_{obs}(\{O, \neg O\}) = 1 - m_{obs}(\{O\}) \tag{7.3c}$$

This formulation supposes that only $m_{t_{i-1}}(\{R\})$ and $m_{t_i}(\{\neg R\})$ can indicate the presence of a road obstacle. The α function computes a discounting factor, which depends on the mean z coordinates of the points that have been projected, while creating the ScanGrid(t_i), into the considered grid cell. This mean elevation is noted $\overline{Z}$. As the LIDAR used by the ZoeCab systems is put on the roof of the vehicles, $\overline{Z}$ is typically negative when only ground points have been projected into a grid cell. We define α as follows:

$$\alpha(z) = min(exp(\nu(z + \xi)), 1) \tag{7.4}$$

This function only generates discounting factors in the]0,1] range. The ξ parameter indicates the absolute value of the height from which the conflict does not have to be discounted. The ν factor monitors the growth of $\alpha(z)$.

Similarly, we can define $m_{displaced}$ as the evidential mass function associated to a cell of ScanGrid(t_i) under the $\Omega_{displaced}$ frame of discernment. We propose to compute $m_{displaced}$ as follows:

$$m_{displaced}(\{D\}) = (1 - \alpha(\overline{Z})) * m_{t_i}(\{R\}) * m_{t_{i-1}}(\{\neg R\}) \tag{7.5a}$$

$$m_{displaced}(\{\neg D\}) = 0 \tag{7.5b}$$

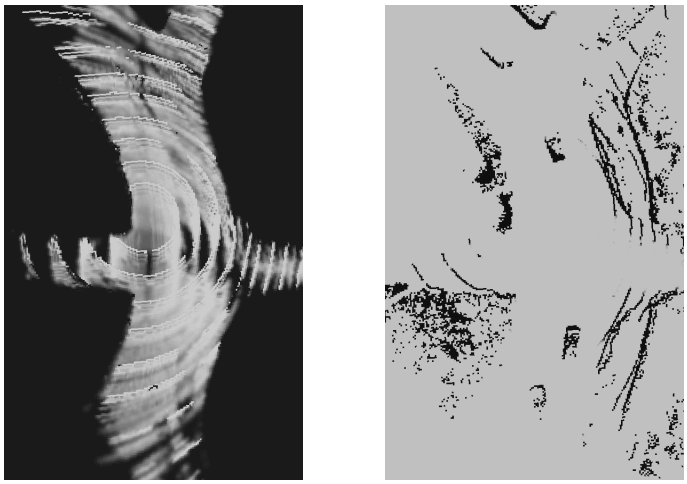
$$m_{displaced}(\{D, \neg D\}) = 1 - m_{displaced}(\{D\}) \tag{7.5c}$$

Additionally, grids cells that should not be considered anymore as occupied in RoadGrid(t_{i-1}) can easily be detected from $m_{displaced}$. The grids in RoadGrid(t_{i-1}) for which $m_{displaced}(D)$ is higher than 0.5 are reinitialized to a fully unknown state: $m_{t_{i-1}}(\{R\}) = 0$, $m_{t_{i-1}}(\{\neg R\}) = 0$, $m_{t_{i-1}}(\{R, \neg R\}) = 1$.

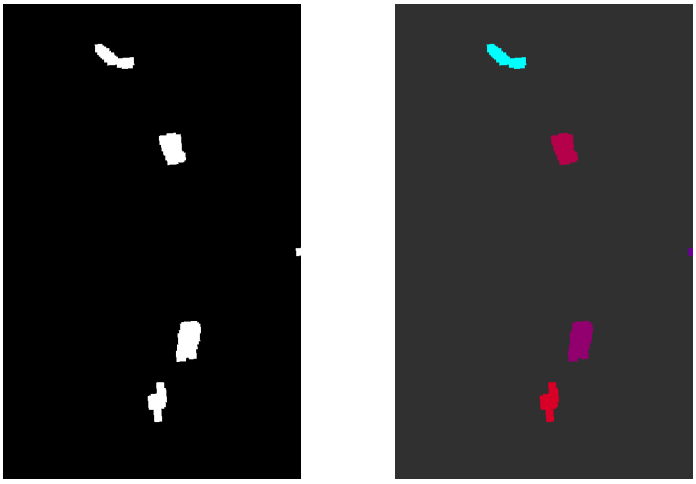
7.4 Clustering and road object detection

Similarly, the m_{obs} mass can be used to detect grid cells that belong, in a ScanGrid, to a moving road obstacle, and should not be fused with RoadGrid(t_{i-1}). We consider that static obstacles do not have to be detected, as they would cover an area that is not drivable. A binary ObsMap(t_i) map is created from ScanGrid(t_i) and the m_{obs} values. This binary map represents, for each cell, the presence of a road obstacle. A binary cell is set to 1 if the corresponding cell in ScanGrid(t_i) has an $m_{obs}(O)$ value higher than 0.5. Otherwise, it is set to 0.

ObsMap(t_i) can be used to generate a list of detected road obstacles. First of all, a 5×5 maximum filter is applied to ObsMap(t_i), to inflate the detected objects. This pessimistic behavior is justified by the need of taking into account the fact that the LIDAR points at the edges of those objects, when having been projected into the grid cells, might have been projected into cells where road points were also present. The α function might then be affected, and return an under-confident discounting factor. This maximum filtering is also used to connect grid cells that belong to the same physical obstacle, which might not be the case because of the sparsity of the LIDAR scans. Finally, ObsGrid(t_i) is converted into a grid of cluster ids, noted ClusterMap(t_i), by connected component labeling, with an 8-connectivity. In each cell of ClusterMap(t_i) is indicated the id of the cluster to which the cell belongs, or 0 if the cell does not correspond to a clustered object. This ClusterMap(t_i) can be seen as a list of localized road obstacles. Afterwards, the cells of ScanGrid(t_i) for which a cluster id has been returned are also reinitialized to a fully unknown state: $m_{t_i}(\{R\}) = 0$, $m_{t_i}(\{\neg R\}) = 0$, $m_{t_i}(\{R, \neg R\}) = 1$. Each grid used in this step is presented in Figure 7.4.



(a) $m(\{R\})$ in RoadGrid(t_{i-1}) (b) $m(\{\neg R\})$ in ScanGrid(t_i)



(c) Max filtered ObsMap(t_i) (d) ClusterMap(t_i)

Figure 7.4: Grids used for clustering and road object detection.

7.5 Road accumulation and ego-motion compensation

As potential road objects and displaced objects have been removed, ScanGrid(t_i) and RoadGrid(t_{i-1}) can trivially be fused, by simply using the Dempster-Shafer operator on m_{t_i} and $m_{t_{i-1}}$ for each grid cell. The resulting RoadGrid(t_i) is then available for a navigation system, and can be fused with new incoming LIDAR scans. However, when a new ScanGrid will be generated, the displacement of the vehicle over time will have to be considered, before fusing it with a RoadGrid. An odometry model is thus needed to reproject the RoadGrid. A odometry model can be used to track the movement of the acquisition platform, and reproject the RoadGrid when a new ScanGrid will be available. Cells of the RoadGrid that are not projected into the area covered by the new ScanGrid are dropped. New cells that cover previously unobserved areas are initialized to a fully unknown

state, with a mass value of 1 for $\{R, \neg R\}$.

7.6 Implementation and example of output

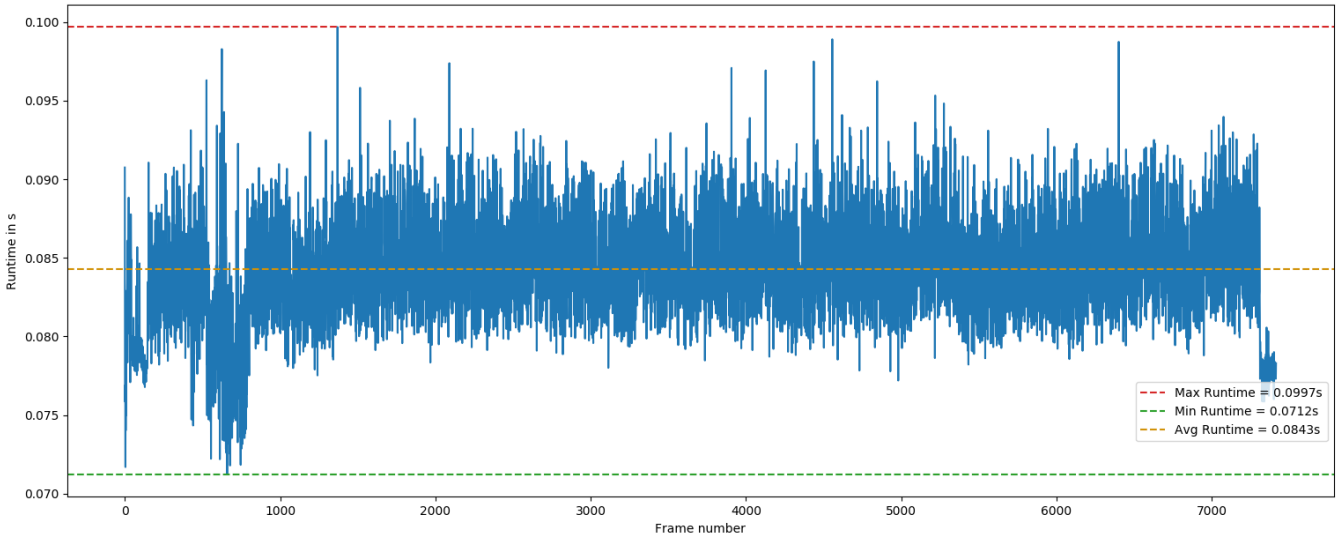
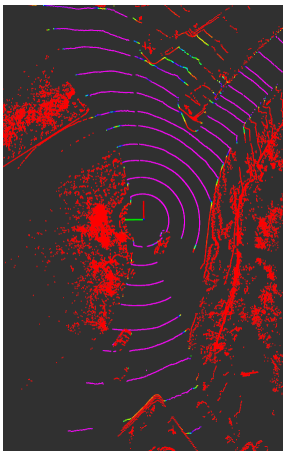


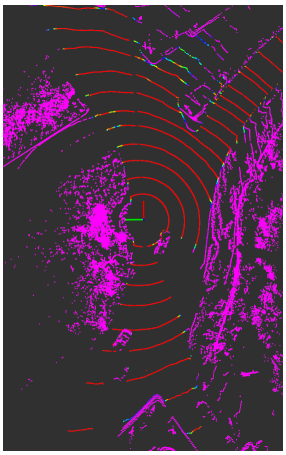
Figure 7.5: Runtime of the road mapping and object detection algorithm relying on the fused RoadSeg networks

The algorithm was implemented as a Python ROS node. The inference and evidential fusion of the neural networks is done via the PyTorch framework, and the operations on the grid are performed thanks to the Numpy, OpenCV and Scipy libraries. The TitanX GPU that was used for the training is reused for the inference of the neural networks, but all the grid operations are done on an Intel i7-6700K octacore CPU. An $80m \times 50m$ grid is computed around the vehicle, with a cell size of $0.2m$. Only points that have a Z coordinate in the [-2.5,0] range are considered. The odometry is evaluated from an Extended Kalman filter relying on a classical Constant Turn Rate and Velocity (CTRV) model. The CAN networks provides the system with speed an heading direction measurements at 10 Hz, and a yaw rate measurement at 100Hz. The CTRV model normally also fuses position measurements obtained from a GNSS sensor, but we chose to rely on a pure CAN odometry, so as to be agnostic to the localization system that is in use. The ν and ξ values in the α function are respectively set to 4 and 1,5. Similarly to what was done for the training, validation and test sets, the LIDAR scans are obtained from a VLP32C running at 10Hz. We report in Figure 7.5 the temporal behavior of the algorithm over a 12-minute recording session in Guyancourt. The measured run-times cover the unpacking of the LIDAR scans, the inference of the neural networks, their fusion, and all the steps of the grid-level mapping and detection algorithm. Our current implementation manages to

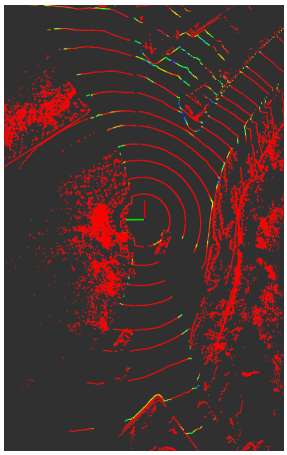
match, on this recording session, the publication rate of the LIDAR, as the run time is always below 100ms. Yet, the processing time is sometimes very close to 100ms, due to significant jitter. We thus cannot guarantee that the LIDAR scans will always be processed at 10Hz with the current implementation. However, the fact that most of the current implementation relies on standard functions, without extensive use of the GPU, indicates that the performances will be improved by using a dedicated, pure GPU implementation of the functions used in this road mapping and object detection algorithm. We report an additional example of the outputs that are available from the algorithm, in Figure 7.6. This example highlights one limitation with using conflict analysis for object detection: objects at road edges tend to generate false positives, especially when the odometry is imprecise.



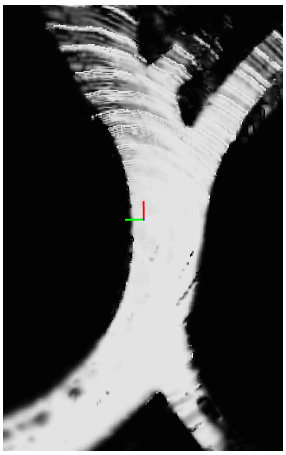
(a) $m(\{R\})$ for each point



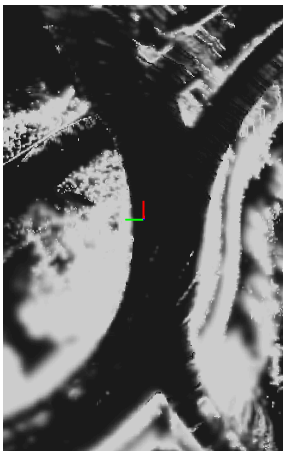
(b) $m(\{\neg R\})$ for each point



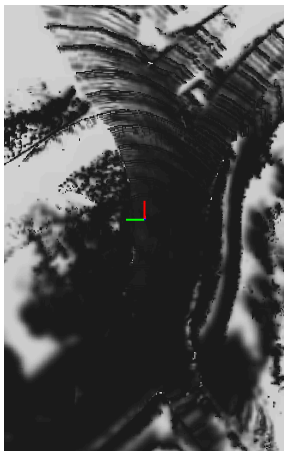
(c) $m(\Omega)$ for each point



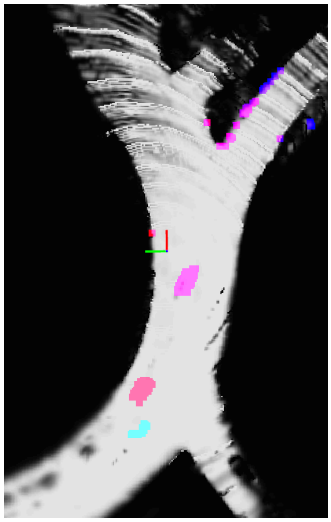
(d) $m(\{R\})$ in RoadGrid



(e) $m(\{\neg R\})$ in RoadGrid



(f) $m(\Omega)$ in RoadGrid



(g) ClusterMap and RoadGrid

Figure 7.6: Outputs from the road mapping and object detection algorithm

7.7 Conclusion

We presented an algorithm that uses this road detection system to map the road surface over time, and cluster road objects. A simple CPU/GPU implementation of this algorithm is able to process LIDAR scans at approximately 10 Hz, which fits the usual publication rate of state-of-the-art LIDAR sensors. Additional training data is likely to lead to even better results, which is easy to obtain from our automatic label generation procedure, provided that accurate maps are available. Multi-class classification would also be valuable, especially for road object detection.

Chapter 8

Conclusion

Contents

8.1 Conclusion

8.1.1 Asynchronous evidential grid mapping from RGB images and LIDAR scans

We proposed a grid-based asynchronous fusion algorithm of LIDAR scans and RGB images. A new Cartesian mapping scheme from LIDAR scans was proposed, and a way to cope with possibly moving objects based on their semantic class was evaluated. The use of an adaptive decay rate, computed from semantic classification results, seems to be an efficient way to generate a meaningful representation, even when moving objects are present. Furthermore, the interest of asynchronous fusion was highlighted. Processing each individual sensor reading, and temporally aligning the generated grids, is a flexible and efficient way to fuse information, while allowing the fusion system to continue working although one of the sensors has failed. Real-time performances have not been reached yet, especially because of the use of DeepLab-v2, and because each operation on grid cells are done iteratively. The modular design of our algorithm will however allow us to introduce more efficient image processing systems in the future. The need for a more robust LIDAR classifier, especially in non-flat areas or in presence of side-walks, was also highlighted by our experiments.

8.1.2 Evidential LIDAR object classification

An evidential classification system for LIDAR objects, represented as feature vectors, was proposed. The algorithm, which was trained as a probabilistic classifier and converted as an evidential classifier afterwards, effectively classifies unknown objects without having been trained on them. Evidential classification is compatible with real-time constraints: on a TitanX Pascal GPU, all the LIDAR objects in D_{rc} (which is composed of 30190 objects) can be classified at once in 0.4s with the current Pytorch implementation. Thus, several refinements are possible, and will be explored in future works. First of all, the input vector representing a LIDAR object could be replaced by an input vector encoded by a PointNet architecture [13], as nothing guarantees that the chosen features are the most appropriate ones to classify unknown LIDAR objects. Yet, this would require to define strategies to cope with the limitations and requirements of PointNet regarding its inputs. Secondly, the proposed Z-score filtering scheme relies on a Gaussian assumption that is probably not completely exact: thus, more refined selection strategies for the $ZMax$ threshold would potentially improve the results. Moreover, defining relevant strategies to generate unnormalized mass functions, based on statistical filtering, would be valuable, and allow for a more detailed and accurate representation of the knowledge available for each object. Finally, the definition of strategies to use this system within an autonomous vehicle, in a fusion framework and with heuristics (for e.g. "a moving object is more likely to be pertinent"), is also a direction that has to be explored.

8.1.3 Road detection in LIDAR scans for evidential grid mapping

We proposed, and evaluated, two road detection systems that rely on deep learning to detect the road in LIDAR scans. We also presented a road mapping and object detection algorithm that fuses road detection results from several networks, and manages to run at 10 Hz. A simple CPU/GPU implementation of this algorithm is able to process LIDAR scans at approximately 10 Hz, which fits the usual publication rate of state-of-the-art LIDAR sensors. The results might be further improved thanks to additional training data. A refinement of the training procedure, to cope with the label noise in the automatic labels, is also a possible research direction to improve the system. Multi-class classification would also be valuable, especially for road object detection, but this would come at the cost of, either, semantically enhanced maps, or intensive manual labeling. This would however be very useful to detect more objects, as conflicts analysis only allows us to detect objects on the road, but not on the sidewalks for instance. Huge LIDAR datasets for semantic segmentation are emerging [22], but the burden of manual labeling is still a reality, especially for LIDAR scans since each sensor produces very different scans, in terms of granularity and resolution. This makes the use of external data way more complex than for image segmentation.

8.2 Perspectives

8.2.1 Towards an unified evidential perception system

We were not able to propose an unified perception system relying on all the perception modules that we developed. This was mainly due to time constraints, and an unavailability of perception platforms and data during the first year of the thesis. We however believe that those three modules could be linked together straightforwardly, by replacing the LIDAR processing pipeline in our asynchronous algorithm by out newest LIDAR road mapping algorithm. Conflict analysis could then be used to, again, detect objects, but this time in grids that would be generated from both RGB images and LIDAR scans. Classification would however be tricky, as conflict analysis and connected component labeling leads to very coarse clusters, that can actually correspond to several objects. Properly reclustering, and classifying, LIDAR objects that are detected by conflict analysis would be a challenging, and extremely interesting, project.

Instead of relying on conflict analysis to detect objects, a neural network, trained on adequate data, could be used to perform the detection. Nevertheless, the system would then have to work without any human supervision. We instead believe that an adequate World Model [75], designed to process our grids, would be able to track the objects that we detect by conflict analysis. This World model could also be used to model the detected objects in the context of the scene, which could be reused in our LIDAR object classifier, to properly estimate mass values on the empty set. The World model could for instance estimate the relevance of objects in a scene, based on their location with regards to a pre-existing HD map, and mass values on the empty set might be built from this information. A very high object where the World model expects a pole or a building, according to a pre-existing map, might be for instance associated with a large mass value on the empty set, and our classifier would then be used to refine the available information, and verify whether it is a vehicle, a vulnerable road user, or something in between. More refined grids could also be obtained via semantic segmentation of LIDAR scans, but additional training data and labels would then be required.

A complete evidential representation of the environment could then be obtained: the drivability of an area can be inferred from our road detection system, a clustering algorithm could be used to detect objects, and our LIDAR objects classifier could be used to localize vulnerable road users and vehicles. Such various observations can be used as inputs to further path planning systems. A recent research direction that might be of interest in this context is mid-to-mid driving: a deep neural network takes as input a representation of the environment obtained from a perception system and pre-existing maps, and outputs a path that the vehicle should follow, as done in *ChaufferNet* [91]. Figure 8.1 presents an example of output of *ChaufferNet*. Yet, those systems usually assume that

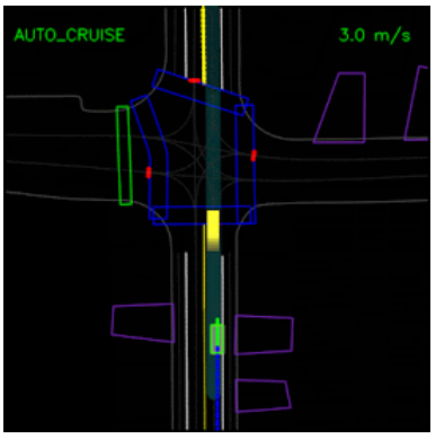


Figure 8.1: Example of output from [91]. The blue points represent the past path, the green points represent the future path chosen by the network. The other features in the image represent the information extracted by the perception system, and a pre-existing map of the environment. In particular, the yellow box represents a vehicle.

the perception system generated a reliable representation of the environment. The possibility of integrating an evidential interpretation of both the input and outputs of such systems, that are very dependent on the accuracy of their input, could potentially make them less sensitive to conflicts and uncertainties in the perception process.

8.2.2 Towards evidential perception integrity

We also believe that the use of the theory of belief functions, in a perception system, would be extremely useful in defining, and measuring, perception integrity indicators. The concept of integrity is traditionally related to the performances of a localization system. It can be defined as the measure of the trust that can be placed in the information supplied, in this context, by a localization system [92]. This trust is evaluated via a protection level, that quantifies the maximum error in localization, in the event of an undetected localization error. A naive extension of this concept, for perception, could then rely on a measure of the risk caused by an unperceived object. However, this definition can not be complete, as perception is a composite task: to be perceived, an object must, at least, be detected, confirmed, classified, tracked, and localized. Focusing on objects is also unsatisfactory, as perception systems must also detect areas that are traversable in the environment (typically for autonomous vehicles: the road). Such continuous notions can not be represented by discrete objects. A more fruitful approach to defining perception integrity might be to consider that, instead, it is *a measure of the certainty of the knowledge represented by the perception system, in a given area.* Instead of returning an integrity measure at the system level, as it is the case for localization, the perception integrity should be queried over a region of interest, which would be defined by other modules.

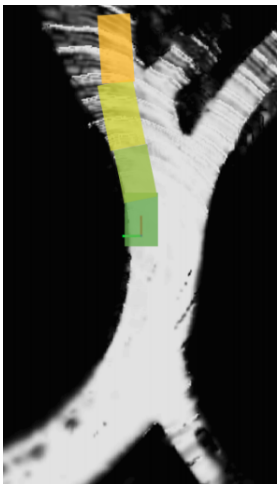


Figure 8.2: Simulated perception integrity over areas of the environment. An evidential road grid, in grey and black, is overlaid with rectangles that are colored according to the mean evidential entropy, over the grid cells that they cover. The rectangles are assumed to be areas where a path planners wants to verify the integrity of the information. In real-life, those rectangles would obviously be way smaller.

The perception integrity of this area would represent all the information returned by all the perception modules (road mapping, object detection, object tracking,...). The evidential framework would then be of particular interest, as it explicitly represents the fact of doubting (mass values on the complete frame of discernment, on subsets that have a cardinality higher than one), and not knowing (mass on the empty set). In Figure 8.2, we simulated such a system, but only focused on the road. We assumed that a path planner wanted to verify the quality of the information over a possible path, and asked a perception integrity value over four areas: the four color rectangles in Figure 8.2. The road surface in Figure 8.2 was mapped thanks to the system in Chapter 7, and thus each grid cell was associated with evidential mass values on $\Omega = \{Road, notRoad\}$. Each rectangle was colored according to a perception integrity value, that we chose to be the mean evidential entropy of the cells in the rectangle. Choosing this entropy as an integrity value assumes not knowing is not an issue for the system, provided that it knows that it does not know. An efficient decision-making system is unlikely to want to drive through unobserved areas. Only ambiguity among the mass values matters in this case. Integrating the available information about the objects presents in those rectangles, according to the mass values returned by other perception modules, and fusing it with this evidential entropy value regarding the presence of road, is likely to lead to usable perception integrity measures.

Bibliography

[1] B. Van Arem, J. Hogema, and S. Smulders, "The impact of autonomous intelligent cruise control on traffic flow," in *Intelligent Transportation: Realizing the Future. Abstracts of the Third World Congress on Intelligent Transport Systems (ITS)*, 1996.

[2] C. Urmson, J. A. Bagnell, C. Baker, M. Hebert, A. Kelly, R. Rajkumar, P. E. Rybski, S. Scherer, R. Simmons, S. Singh, *et al.*, "Tartan racing: A multi-modal approach to the darpa urban challenge," 2007, Visited on 2019-10-9. [Online]. Available: `https://kilthub.cmu.edu/articles/Tartan_Racing_A_Multi-Modal_Approach_to_the_DARPA_Urban_Challenge/6561125/1`.

[3] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, *et al.*, "Junior: The stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[4] A. Bacha, C. Bauman, R. Faruque, M. Fleming, C. Terwelp, C. Reinholtz, D. Hong, A. Wicks, T. Alberi, D. Anderson, *et al.*, "Odin: Team Victortango's entry in the darpa urban challenge," *Journal of field Robotics*, vol. 25, no. 8, pp. 467–492, 2008.

[5] J. Krafcik, "Waymo one: The next step on our self-driving journey," Visited on 2019-10-9. [Online]. Available: `https://medium.com/waymo/waymo-one-the-next-step-on-our-self-driving-journey-6d0c075b0e9b`.

[6] C. Hampel, *Autonomous services with Renault and Transdev*, Visited on 2019-10-9. [Online]. Available: `https://www.electrive.com/2019/05/21/autonomous-services-with-renault-and-transdev/`.

[7] *Another milestone for easymile: The first fully driverless service*, Visited on 2019-10-9. [Online]. Available: `https://easymile.com/another-milestone-for-easymile-the-first-fully-driverless-service-of-our-ez10-driverless-shuttle/`.

[8] S. Singh. (2015). Critical reasons for crashes investigated in the national motor vehicle crash causation survey. Visited on 2019-10-9, [Online]. Available: `http://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115`.

[9] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[10] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[11] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The CityScapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3213–3223.

[12] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013.

[13] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[14] B. Wu, A. Wan, X. Yue, and K. Keutzer, "SqueezeSeg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3D lidar point cloud," in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 1887–1893.

[15] J. Fritsch, T. Kuehnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[16] C. Plachetka, J. Rieken, and M. Maurer, "The tubs road user dataset: A new lidar dataset and its application to cnn-based road user classification for automated vehicles," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 2623–2630.

[17] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "Nuscenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.

[18] M.-F. Chang, J. W. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, "Argoverse: 3d tracking and forecasting with rich maps," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[19] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet, *Lyft level 5 av dataset 2019*, `https://level5.lyft.com/dataset/`, 2019.

[20] *Waymo open dataset: An autonomous driving dataset*, 2019.

[21] X. Roynard, J.-E. Deschaud, and F. Goulette, "Paris-lille-3d: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification," *The International Journal of Robotics Research*, vol. 37, no. 6, pp. 545–557, 2018.

[22] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "Semantickitti: A dataset for semantic scene understanding of lidar sequences," in *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[23] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[25] F. Engelmann, T. Kontogianni, A. Hermans, and B. Leibe, "Exploring spatial context for 3D semantic segmentation of point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 716–724.

[26] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems*, 2017, pp. 5099–5108.

[27] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[28] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," *arXiv preprint arXiv:1904.08889*, 2019.

[29] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun, "Multinet: Real-time joint semantic reasoning for autonomous driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018, pp. 1013–1020.

[30] Z. Chen and Z. Chen, "Rbnet: A deep neural network for unified road and road boundary detection," in *International Conference on Neural Information Processing*, Springer, 2017, pp. 677–687.

[31] X. Han, J. Lu, C. Zhao, S. You, and H. Li, "Semisupervised and weakly supervised road detection based on generative adversarial networks," *IEEE Signal Processing Letters*, vol. 25, no. 4, pp. 551–555, 2018.

[32] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[33] R. Fernandes, C. Premebida, P. Peixoto, D. Wolf, and U. Nunes, "Road detection using high resolution lidar," in *2014 IEEE Vehicle Power and Propulsion Conference (VPPC)*, IEEE, 2014, pp. 1–6.

[34] Y. Lyu, L. Bai, and X. Huang, "Chipnet: Real-time lidar processing for drivable region segmentation on an fpga," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2018.

[35] L. Caltagirone, S. Scheidegger, L. Svensson, and M. Wahde, "Fast lidar-based road detection using fully convolutional neural networks," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2017, pp. 1019–1024.

[36] S. Gu, Y. Zhang, J. Tang, J. Yang, and H. Kong, "Road detection through crf based lidar-camera fusion," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 3832–3838.

[37] L. Caltagirone, M. Bellone, L. Svensson, and M. Wahde, "Lidar–camera fusion for road detection using fully convolutional neural networks," *Robotics and Autonomous Systems*, vol. 111, pp. 125–131, 2019.

[38] Z. Chen, J. Zhang, and D. Tao, "Progressive lidar adaptation for road detection," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 3, pp. 693–702, 2019.

[39] X. Li, L. Zhang, A. You, M. Yang, K. Yang, and Y. Tong, "Global aggregation then local distribution in fully convolutional networks," *accepted to BMVC2019*, 2019.

[40] Y. Zhu, K. Sapra, F. A. Reda, K. J. Shih, S. Newsam, A. Tao, and B. Catanzaro, "Improving semantic segmentation via video propagation and label relaxation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8856–8865.

[41] Y. Yuan, X. Chen, and J. Wang, "Object-contextual representations for semantic segmentation," *arXiv preprint arXiv:1909.11065*, 2019.

[42] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, "Rangenet++: Fast and accurate lidar semantic segmentation," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[43] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer, "SqueezeSegV2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 4376–4382.

[44] M. Himmelsbach and H.-J. Wuensche, "Tracking and classification of arbitrary objects with bottom-up/top-down detection," in *2012 IEEE Intelligent Vehicles Symposium*, IEEE, 2012, pp. 577–582.

[45] I. Bogoslavskyi and C. Stachniss, "Efficient online segmentation for sparse 3d laser scans," *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, pp. 1–12, 2017. [Online]. Available: `https://link.springer.com/article/10.1007%2Fs41064-016-0003-y`.

[46] P. Burger, B. Naujoks, and H.-J. Wuensche, "Fast dual decomposition based mesh-graph clustering for point clouds," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 1129–1135.

[47] M. Simon, S. Milz, K. Amende, and H.-M. Gross, "Complex-yolo: Real-time 3D object detection on point clouds," *arXiv preprint arXiv:1803.06199*, 2018.

[48] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[49] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.

[50] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu, "Class-balanced grouping and sampling for point cloud 3d object detection," *arXiv preprint arXiv:1908.09492*, 2019.

[51] A. Dempster, "Upper and lower probabilities induced by a multivmulti mapping," *The Annals of Mathematical Statistics*, vol. 38, 1976.

[52] G. Shafer, *A mathematical theory of evidence*. Princeton university press, 1976, vol. 42.

[53] P. Smets, Y.-T. Hsia, A. Saffiotti, R. Kennes, H. Xu, and E. Umkehren, "The transferable belief model," in *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, Springer, 1991, pp. 91–96.

[54] F. Pichon and T. Denœux, "The unnormalized dempster's rule of combination: A new justification from the least commitment principle and some extensions," *Journal of Automated Reasoning*, vol. 45, no. 1, pp. 61–87, 2010.

[55] P. Smets, "Decision making in the tbm: The necessity of the pignistic transformation," *International Journal of Approximate Reasoning*, vol. 38, no. 2, pp. 133–147, 2005.

[56] P. Snow, "The vulnerability of the transferable belief model to dutch books," *Artificial Intelligence*, vol. 105, no. 1-2, pp. 345–354, 1998.

[57] B. R. Cobb and P. P. Shenoy, "On the plausibility transformation method for translating belief function models to probability models," *International Journal of Approximate Reasoning*, vol. 41, no. 3, pp. 314–330, 2006.

[58] F. Wang, A. Miron, S. Ainouz, and A. Bensrhair, "Post-aggregation stereo matching method using dempster-shafer theory," in *2014 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2014, pp. 3783–3787.

[59] V. Magnier, D. Gruyer, and J. Godelle, "Implementation of a multi-criteria tracking based on the dempster-shafer theory," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2015, pp. 463–468.

[60] L. Rummelhard, A. Nègre, and C. Laugier, "Conditional monte carlo dense occupancy tracker," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, IEEE, 2015, pp. 2485–2490.

[61] Tun Yang and V. Aitken, "Evidential mapping for mobile robots with range sensors," in *2005 IEEE Instrumentation and Measurement Technology Conference Proceedings*, vol. 3, May 2005, pp. 2180–2185. DOI: 10.1109/IMTC.2005.1604562.

[62] F. W. Rauskolb, K. Berger, C. Lipski, M. Magnor, K. Cornelsen, J. Effertz, T. Form, F. Graefe, S. Ohl, W. Schumacher, *et al.*, "Caroline: An autonomously driving vehicle for urban environments," *Journal of Field Robotics*, vol. 25, no. 9, pp. 674–724, 2008.

[63] J. Moras, V. Cherfaoui, and P. Bonnifait, "Moving objects detection by conflict analysis in evidential grids," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2011, pp. 1122–1127. DOI: 10.1109/IVS.2011.5940561.

[64] P. Smets, "Data fusion in the transferable belief model," in *Proceedings of the Third International Conference on Information Fusion*, vol. 1, Jul. 2000, PS21–PS33 vol.1. DOI: 10.1109/IFIC.2000.862713.

[65] C. Yu, V. Cherfaoui, and P. Bonnifait, "An evidential sensor model for velodyne scan grids," in *13th International Conference on Control Automation Robotics & Vision (ICARCV)*, IEEE, 2014, pp. 583–588.

[66] D. Nuss, S. Reuter, M. Thom, T. Yuan, G. Krehl, M. Maile, A. Gern, and K. Dietmayer, "A random finite set approach for dynamic occupancy grid maps with real-time application," *The International Journal of Robotics Research*, vol. 37, no. 8, pp. 841–866, 2018.

[67] S. Wirges, C. Stiller, and F. Hartenbach, "Evidential occupancy grid map augmentation using deep learning," in *IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018, pp. 668–673.

[68] C. Yu, V. Cherfaoui, and P. Bonnifait, "Evidential occupancy grid mapping with stereo-vision," in *Intelligent Vehicles Symposium (IV), 2015 IEEE*, IEEE, 2015, pp. 712–717.

[69] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *arXiv preprint arXiv:1606.00915*, 2016.

[70] P. Fankhauser and M. Hutter, "A universal grid map library: Implementation and use case for rough terrain navigation," in *Robot Operating System (ROS)*, Springer, 2016, pp. 99–120.

[71] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[72] P. Chu, S. Cho, S. Sim, K. Kwak, and K. Cho, "A fast ground segmentation method for 3D point cloud," *Journal of information processing systems*, vol. 13, no. 3, pp. 491–499, 2017.

[73] G. Neuhold, T. Ollmann, S. Rota Bulò, and P. Kontschieder, "The Mapillary Vistas dataset for semantic understanding of street scenes," in *Proceedings of the International Conference on Computer Vision (ICCV), Venice, Italy*, 2017, pp. 22–29.

[74] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2650–2658.

[75] J. S. Albus, "4D/RCS: A reference model architecture for intelligent unmanned ground vehicles," in *Unmanned Ground Vehicle Technology IV*, International Society for Optics and Photonics, vol. 4715, 2002, pp. 303–311.

[76] P. Smets *et al.*, "What is dempster-shafer's model," *Advances in the Dempster-Shafer theory of evidence*, pp. 5–34, 1994.

[77] T. Denoeux, "Logistic regression, neural networks and dempster–shafer theory: A new perspective," *Knowledge-Based Systems*, vol. 176, pp. 54–67, 2019.

[78] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[79] D. J. Rumsey, *U Can: statistics for dummies*. John Wiley & Sons, 2015.

[80] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, 1996, pp. 226–231.

[81] X. Zhang, W. Xu, C. Dong, and J. M. Dolan, "Efficient l-shape fitting for vehicle detection using laser scanners," in *Intelligent Vehicles Symposium (IV)*, IEEE, 2017, pp. 54–59.

[82] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[83] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[84] M. C. Troffaes, "Decision making under uncertainty using imprecise probabilities.," *International journal of approximate reasoning.*, vol. 45, no. 1, pp. 17–29, 2007.

[85] K. A. B. Ahmad, M. Sahmoudi, and C. Macabiau, "Characterization of GNSS receiver position errors for user integrity monitoring in urban environments," in *ENC-GNSS 2014, European Navigation Conference*, 2014.

[86] Z. Tao and P. Bonnifait, "Road invariant extended kalman filter for an enhanced estimation of gps errors using lane markings," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 3119–3124.

[87] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "Nuscenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.

[88] F. Li, P. Bonnifait, J. Ibanez-Guzman, and C. Zinoune, "Lane-level map-matching with integrity on high-definition maps," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2017, pp. 1176–1181.

[89] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[90] R. Jiroušek and P. P. Shenoy, "A decomposable entropy of belief functions in the dempster-shafer theory," in *International Conference on Belief Functions*, Springer, 2018, pp. 146–154.

[91] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018.

[92] O. Le Marchand, P. Bonnifait, J. Ibañez-Guzmán, and D. Bétaille, "Vehicle localization integrity based on trajectory monitoring," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2009, pp. 3453–3458.