University of Padova

Department of Information Engineering

Master Thesis in Computer Engineering

Lane Detection System for Intelligent Vehicles using Lateral Fisheye Cameras

Supervisor
Prof. Stefano Ghidoni

Master candidate
Alex Valente

Co-supervisors
Dr. Francesca Ghidini
Senior Algorithm Engineer, Ambarella Inc.

Dr. Pietro Cerri
Senior Manager, Ambarella Inc.

9^{th} December 2019
Academic Year 2018/2019

Acknowledgments

This master thesis is the result of intense work lasted several months, but also the conclusion of an adventure that started five years ago with the beginning of my University career. They have been unforgettable years, made of very good moments but also others in which my principal thought was "I have so many tasks to complete, I will never finish in time!". But every time, working assiduously until the last useful moment, I managed to conclude with success, and this moment has finally arrived also for my final and most important work. What is sure is that I have to thank a lot of people for having helped me with the achievement of this goal.

A special thank to my supervisor Prof. Stefano Ghidoni and my co-supervisors Dr. Francesca Ghidini and Dr. Pietro Cerri: during the last months their assistance, advice and incredible willingness have certainly been crucial for the success of this work.

Thanks also to my colleagues in VisLab, included my already cited co-supervisors and all the Ph.D. students (with a particular mention to Andrea, I have promised him a citation and here it is!) I have had the opportunity to know: they have welcomed me about five months ago and I will difficultly forget the good times I have spent there.

Thanks to Prof. Alberto Broggi for the opportunity he has given me at VisLab, it has made me grow both professionally and personally.

Thanks to my colleague, housemate, and friend Anthony: we have shared a lot of very good moments during the last months, we have talked about disparate topics (sometimes making fun of us) and we have supported each other to overcome all the challenges our work has put in front of us. His company has certainly been one of the most precious during this adventure.

Thanks to all my friends and my university colleagues, for all the funny moments I have had with them, for the good times spent together and for their support.

Thanks to my friend Davide: we shared so many moments, ideas and laughs during the last years, we have studied together almost everyday to help and support each other. Knowing that I could count on him in every moment was very precious for me.

Last but not least, a very special thank to my parents Paola and Massimo, to my

brother Alberto and to all my family: they have always believed in me and their continuous support during all the days of my life has been one of the most important thing that has allowed me to arrive where I am today.

Contents

List of Figures

List of Tables

Abstract

The need for safety on roads has made the development of autonomous driving one of the most important topics for Computer Vision research. Among them, lane detection is a crucial stage for a great number of Advanced Driving Assistance Systems (ADAS), being essential for road lane following and robust ego localization. During the last decades, more and more approaches have been presented in the literature. However, only a minimal part of them makes use of lateral fisheye cameras, although their significant advantages.

This thesis will focus on the lane detection problem using images obtained with lateral fisheye cameras, firstly by studying the state-of-the-art and the theory about the spherical model, then by developing two methods to solve this challenging task. While the first is based on traditional Computer Vision, searching Dark-Light and Light-Dark transitions and then pairing them, the second makes use of a Convolutional Neural Network originally thought for image segmentation. The approaches are then compared, obtaining the second to outperform the first one. Future developments are finally presented, focusing on the possibility of using a hybrid approach that could merge the advantages of both the methods discussed.

1

Introduction

In recent years the automotive industry has become one of the most dynamic ones and it is continuously evolving. Nowadays, companies are focusing on the development of many new technologies and most of them have in common one main goal which is the safety of people on roads.

Although Europe has some of the safest roads in the world, in 2018 there were about 25100 road fatalities [1]. It is important to point out that 95% of accidents are due to human errors [2], like the distraction caused by the use of mobile phone (11% of the total [3]) or the tiredness of the driver. To make these numbers as small as possible the European Union is working to force the introduction of some advanced safety systems that could save lives. Anyway, this necessity is not only felt in Europe: American and Asian governments are also working to reach the same purpose.

More and more security systems have been implemented during the last decades to avoid or at least to mitigate consequences of road accidents, but it is clear that autonomous driving will be the ultimate answer to the search of innovative methods to make drivers safer. The principal aim of autonomous driving is to avoid most of the accidents that could be caused by the distraction of the driver or by a wrong perception of the world around him/her. At the same time, there are also economic advantages: we can think about lower insurance expenses for drivers, but the real revolution could be related to the introduction of autonomous public transport (that has already been experimented in some cities), capable of offering services to people in every moment, leading also to the progressive lack of necessity of owning a vehicle. Autonomous driving revolution is strictly related to other phenomena like electrification and car-sharing, leading to the birth of smart cities due to the future need of communication between vehicles and objects like semaphores (in this case we are talking about V2I communication, with the meaning of Vehicle-to-Infrastructure).

A self-driving car is defined as a vehicle equipped with sensors and controlled by artificial intelligence. According to the Society of Automotive Engineers (SAE), there exist six levels of vehicle automation (see Figure 1.1):

0. **No automation**: the driver controls the car without any support from a driver assistance system;

1. **Driver assistance**: the vehicle can assist with some functions (one at a time), but the driver still has the control of accelerator, brakes and has to monitor the surrounding environment. Examples of Level 1 systems are Adaptive Cruise Control and Collision and Pedestrian warning;

2. **Partial automation**: the vehicle can assist the driver with steering and acceleration. The driver must always be ready to take control of the vehicle and he/she is still responsible for most safety-critical functions and monitoring the environment. Examples of Level 2 systems are ADAS (Advanced Driver Assistance Systems), such as Lane Control Assistance. The majority of high-end vehicles systems shipped these days belong to this class (for example, Tesla Autopilot is a SAE Level 2 system);

3. **Conditional automation**: the vehicle has the ability to detect the environment using sensors such as cameras and LIDARs and can make decisions autonomously. It can control the steering, the accelerator and the brakes. However, the driver must be ready to take control if the system is unable to execute the task;

4. **High automation**: the vehicle does not require human interaction in most circumstances. It can detect the environment and take decisions autonomously (such as change lanes, turn,...). However, a human still has the option to manually override the system. If the system notices that the conditions for driving autonomously are not met, the vehicle can inform the driver that it is not able to handle the current complex situation. If the driver ignores a warning alarm, the vehicle moves into safe conditions, for example by pulling over;

5. **Full Automation**: the vehicle does not require human attention and can handle every situation. It does not have steering wheel or pedals for acceleration and braking.

Today there exist and are becoming of normal use only systems that have the capabilities described by levels up to 2 and we are approaching to realize level 3. Levels 4 and 5 are far from becoming an everyday reality, but things are changing rapidly. Indeed, the more and more sophisticated solutions proposed, in particular in the last ten years, have made this goal closer. Nowadays autonomous driving is one of the most important topics in Computer Vision research.

Among the large number of systems needed for autonomous driving, lane detection is crucial because it allows a vehicle to localize itself inside the road by knowing its position with respect to the lane boundaries. The main goal of the thesis is the development of a system of this type to make sure that the vehicle maintains its position

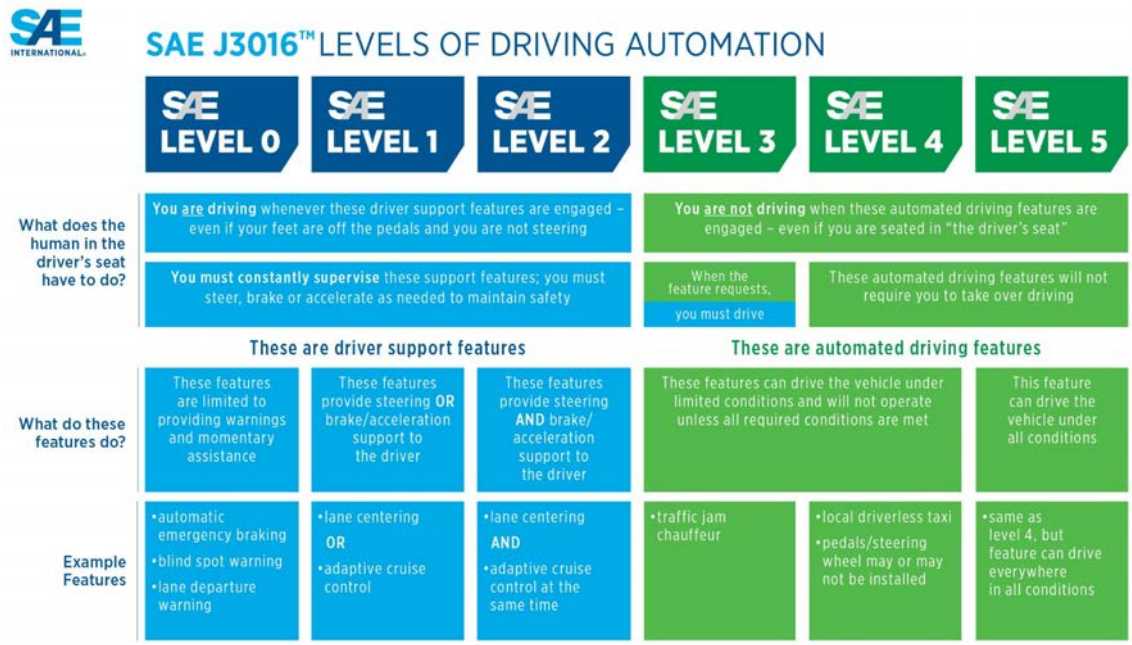


Figure 1.1: SAE Levels of automation.

on the road. Another reason to develop it is the usefulness for another important task such as mapping. For most of the applications, lateral positioning is not performed by using only sensors like GPS as it is simply not available where the sky is not directly visible and because it is not sufficient for fine-grain positioning. To achieve this goal it is needed to make use of meaningful landmarks, and road markings are one of the best candidates for automotive applications. What distinguishes our approach from the others is the use of lateral fisheye cameras in place of more traditional pinhole cameras. As we will see in Chapter 3, they let us have a wider field-of-view and so more information can be captured with every single sensor. Especially, they allow to have all-around-view monitoring by using only four cameras, one for each side of the vehicle. This lets us develop a cheaper final system, allowing also non-high-end vehicles to offer an important safety system built-in. This requires, however, our software to be developed taking into account computational complexity and efficiency since it could be run on embedded systems that usually have non-high-end performances and real-time constraints. A lot of data is acquired with these cameras but it is not used only for detecting lanes since there exist also other purposes, for instance to find parking slots, requiring a robust object detection system.

We can easily understand why lane detection is one of the most tackled topics in Computer Vision research applied to autonomous vehicles. It is essential for every Advanced Driver Assistance System and this is why in the last decades a lot of progress have been made to make systems more and more reliable and capable of good performances

in different scenarios.

Nowadays there already exist commercial ADAS that are able to maintain the vehicle inside its lane autonomously, but at the same time the best results are indeed achieved only in highways, which are highly predictable and structured. Urban environments are more complicated since a lot of more complex situations are possible: the road is not mainly straight and has often variable curvatures, there are intersections and traffic circles, lane markings can be not present at the center of the roadway or they can be faded, deteriorated or simply not well visible because of shadows. But we have not to forget that lane markings can also be of different colors or have patterns, like solid, dashed and double lines. Also the meaning can be different as they can belong to parking slots or pedestrian crossings. The number of combinations is huge when we stop thinking only about highways. And we have not considered the differences between the roads of different countries around the world. It is clear that it is all but an easy task.

Being a building block for more sophisticated systems, a lane detector can be used in different scenarios:

- **Lane Keeping Assist:** the system has to alert the driver every time his vehicle crosses the lane boundaries. To make it possible, the lane boundaries have to be detected with respect to the vehicle trajectory. This system is usually used in highway environments where the task is easier to be performed due to the nature of the road type. In urban environments it would be more challenging since it should work also in case one of the road markings is not visible or is not present at all;

- **Forward Collision Warning:** the aim of the system is to warn the driver and eventually take control of brakes in case we are approaching too quickly a vehicle in front of us. However, it is needed to detect the lane our vehicle is inside to identify which other vehicle is in the same lane. Also this system is usually used only in highways;

- **Adaptive Cruise Control:** this system maintains the selected speed and, if we are approaching another vehicle in front of us, decelerates until stopping if necessary. In more sophisticated applications, the vehicle restarts to accelerate when the traffic returns to move (we can talk about **Traffic Jam Assist** in this case). Requiring some of the features offered by Lane Keeping Assist and Forward Collision Warning, lane detection is essential for this system;

- **Complete autonomous driving:** even if Traffic Jam Assist is the closest ADAS to autonomy (it is an advanced SAE Level 2 system), real full autonomy needs the vehicle to identify all the road situations and consequently all the types of lanes

that exist in the world, not only in highway environments. This is why lanes detection is still a very difficult task.

Before starting the analysis of used approaches, it is important to understand in detail what our purpose is. We want to develop a lane detection system capable of good performances mainly in low speed situations, useful for mapping, localization and perception. The main goal is helping the vehicle to maintain its position inside the lane. Our short-range system has to work both in highway and urban environments.

What we have to point out is what finding a road line means. Since we are searching lateral lines, cameras installed at the sides of the vehicle are more suitable for this type of application. In a lateral image, however, more than one line can be found. What we will consider as the target line is what we call the principal line, defined as the visible and longest one that is at the same time the closest to the vehicle. While at right it always corresponds to the end-of-roadway line, at left, depending on the position of the vehicle, it could correspond to the one that separates the lanes or to the opposite end-of-roadway line when the other one is not visible. In the case of a dashed line, all the dashes will be considered as part of the principal line. Figure 1.2 shows some examples of principal line detection in different situations.

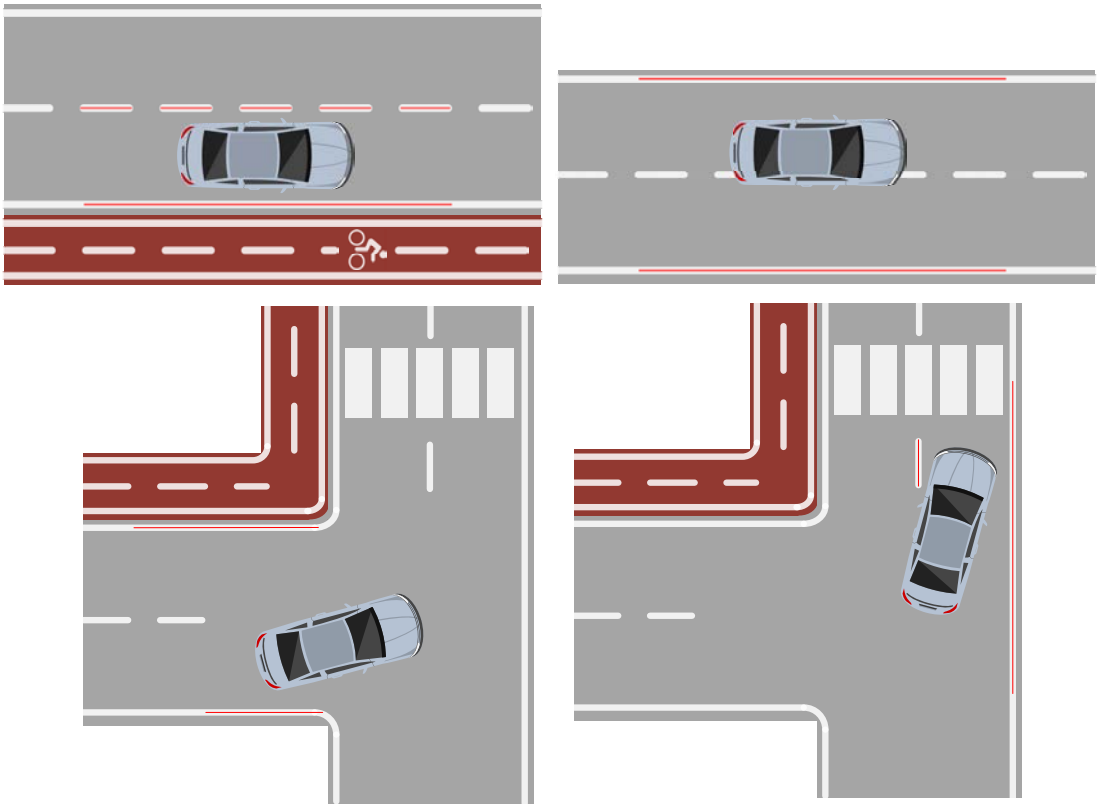


Figure 1.2: Examples of principal lines detection.

The problem of finding road lines has been faced by many researchers, even if fisheye lenses are less used. All the known solutions are based on two main approaches: traditional Computer Vision and Deep Learning.

Classical Computer Vision techniques have been used since the 1990s for lane detection. Even if they are older than Deep Learning, they are still largely used because they allow easier detection of lines when they are clearly visible. However, relevant features that characterize the subject of the recognition have to be found by a person, different from the second approach where they are extracted and chosen automatically by the algorithm. Also the lower computational complexity is an advantage of these techniques.

The second approach is more recent and it is based on the use of Convolutional Neural Networks (CNNs). More and more new applications of this paradigm have been found in the last decade since it usually allows to obtain very good performances with a human effort lower than the one required by the previous methods. Nevertheless, finding a good neural network architecture to complete a task is usually challenging, so the research in this field is very active. CNNs training also requires lots of data, that have to be manually annotated to represent the ground truth. Also computation times are a big problem, but it is usually solved by using specific GPU-based architectures that allow an excellent level of parallelism.

The thesis is structured as follows: Chapter 2 presents state-of-the-art implementations of lane detection systems by using traditional computer vision and CNNs; in Chapter 3 the spherical model is described; Chapter 4 presents the first approach that makes use of traditional computer vision techniques; Chapter 5 is about the approach with CNNs; for each approach a starting consideration about pros and cons is made, then all steps and, in the case of CNNs, data used are described, results are analyzed and at the end possible future work is presented; in Chapter 6 these two approaches are compared; finally, in Chapter 7 some considerations are made to conclude the thesis.

2

Lane detection systems: state-of-the-art

In this chapter we present current state-of-the-art approaches employed to develop lane detection systems. The first paragraph is about the techniques that make use of classical Computer Vision algorithms whereas the second one is focused on the use of Convolutional Neural Networks.

2.1 Traditional Computer Vision

Classical Computer Vision methods have been successfully used during the last decades for tasks ranging from the simplest to the most complex. Lane detection is one of the latter. Traditional steps that characterize most of the traditional Computer Vision-based methods are feature extraction with low-level image processing algorithms, such as Canny for edges detection; lane model fitting with Hough-based approaches; lane tracking, in order to have stable information by integrating over time and predicting the lane position in the future with a Kalman filter. Given the large number of implementations in which they are applied, these methods are considered very reliable. The following paragraphs describe some of the most significant lane detectors based on these techniques.

In [5] road markings are detected in frontal pinhole images by finding Dark-Light-Dark (DLD) transitions. This detection is performed by searching separately Dark-Light (DL) and Light-Dark (LD) transitions and then matching them. Horizontal gradient computation is done taking into account the expected width of road markings and its variation due to the perspective effect.

In [6] the same authors extend the previous work and develop a system that combines stereovision-specific techniques with grayscale image processing. Also in this case DLD transitions detection is performed, but a prediction step is added among stages. The latter, that uses a Kalman filter and takes into account motion parameters of the

ego vehicle, is able to delimit search intervals for detection. This system can also detect freeform lanes.

In 2010 Felisa and Zani [7] proposed a robust lane detection system that uses just one monocular camera and whose requirements for processing were low enough to be compatible with mainstream DSP units of these years. It is based on an transformation in Bird's-Eye View (BEV), DLD highlighting (or DLDLD for double lane markings), clustering of the points found, approximation of the clusters, comparison of resulting segments lists with existing lane markings on previous images (by means of tracking), expansion to join non-connected components, and finally scoring and thresholding. Figure 2.1 shows some output examples of the algorithm.

Figure 2.1: Examples of output images produced by Felisa and Zani's algorithm. Images taken from [7].

The work in [8] is based on an initial search for possible lines made row by row, considering a road marking as present if a DLD transition with a specific width, depending from the perspective, is found. Another extraction algorithm presented in [9] is used for the same task and the results are combined to obtain a more robust lanes detector. The algorithm is tested on both real and synthetic images.

In [10] the goal is to detect more than a single lane without assuming they are parallel, unlike the majority of other approaches presented until that moment. To do so, the authors make use of Conditional Random Fields (CRFs), strong models typically used for solving multiple association tasks. This requires several robust characteristics of the segments to be exploited in order to make correct linkings among them. In Figure 2.2 some results are presented.

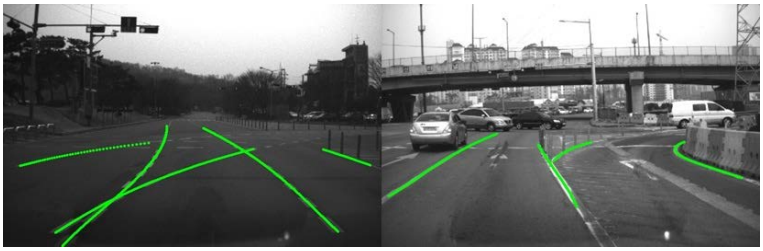


Figure 2.2: Example of application of CRFs for multi-lane detection. Note that these urban cases cannot be solved by considering parallelism. Images taken from [10].

All the works presented until now use only frontal pinhole images as input, but in more recent years the use of fisheye cameras has become prominent as we can see by the increasing number of the related research papers.

For instance, in [15] an hybrid approach for localization that combines the advantages of LiDAR and Around-View-Monitoring is used. Fisheye images are firstly undistorted and perspective is transformed with a Bird's-Eye View (BEV) conversion. What the paper demonstrates is how much sensor-fusion-based methods are capable of providing more reliable and stable results without needing to adopt too computationally-demanding algorithms. However, the main problem remains the high cost of a good LiDAR suitable for such delicate tasks.

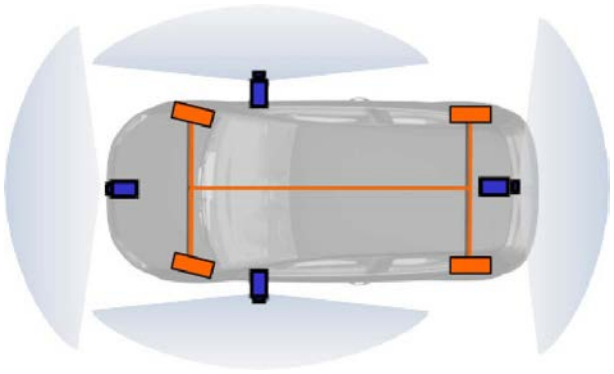


Figure 2.3: Camera setup used in [11–13]. Images taken from [13].

The works presented in [11, 12] are among the first that, instead of using the front view, obtain lane detection only from lateral views. In this case, fisheye cameras are more appropriate because surrounding objects may be very close to the car: it is easy to understand why perspective cameras would not be a good choice. This type of cameras are also more advisable for the cases in which the vehicle moves with low speed. Indeed, in case of higher speed, pinhole cameras are more suited as they allow to see better what is far from them. With fisheye cameras, however, detecting straight lines becomes a more complex problem since in omnidirectional images they correspond to circular arcs when projected onto the equivalence sphere. More details about this can be found in Chapter 3. In [12], using these projections to map lines, the spherical Hough transform [14] and the common Random Sample Consensus (RANSAC) algorithm, good results are achieved, at least in good illumination situations (see Figure 2.4). The camera setup used is those of Figure 2.3.

A similar approach is proposed in [13] to detect parking slots with the same camera setup. This method implements an extension of the Progressive Probabilistic Hough Transform in the space obtained from the reprojection of the fisheye image points in the equivalence sphere. An example of the results can be seen in Figure 2.4.

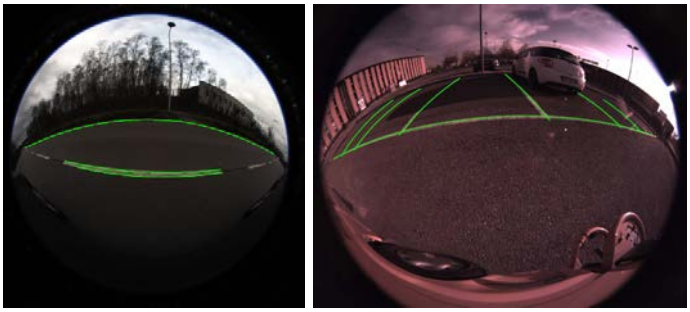


Figure 2.4: Some results obtained in [12] and [13]. Images taken from the papers.

2.2 Deep Learning

In the last decades, Deep Learning has become widely used to perform very different tasks, including detection and recognition. In particular, for image processing Convolutional Neural Networks (CNNs) have become prominent. Most of the recent publications related to lane detection systems make use of them. The development of even more powerful GPUs and platforms for scientific calculations has surely been responsible for making these applications possible.

Pros and cons of deep learning based methods will be presented in detail in Chapter 5.1.

2.2.1 Simple CNN-based methods

One of the first works that makes use of deep neural networks for lane detection is [16]. The method used in this paper can be considered as a bridge between old and new approaches since it combines Convolutional Neural Networks with Random Sample Consensus (RANSAC) algorithm. After having detected the edges, if the road scene is simple lane detection does not involve the use of the CNN and only the RANSAC algorithm is exploited. Otherwise, if the scene is more complex and includes trees, fences or intersections, lanes are more difficult to be found robustly because of noisy edges. So, CNN is used before applying the RANSAC algorithm in order to extract only a Region-Of-Interest where to search road markings.

The first method purely based on deep learning is [17]. Two laterally-mounted cameras are used and the goal is the estimation of the position of lane markings with centimeter-precision in an end-to-end approach using a classification architecture. The name of the used CNN is DeepLanes. Given an image with road markings, the output is the position of the lower part of the line that is closer to the camera (in other words, the road marking that is closer to the bottom of the image). This position is given

as a row number and, in the simpler case, only one is found. If no line is visible, a non-valid row number is provided. This method is useful if the line we are searching is straight while it is not suitable if it has a curvature, since having only one row index does not give any information about the orientation. To solve this issue, the same paper proposes an extension of the method: the image is divided, for example, into five parts and the estimation of the position of the lane marking is done for each slice. This allows to obtain multiple points that can be used to calculate the lane orientation. The evaluation of DeepLanes shows a precision of 98.96%, a recall of 99.9% and 99.04% of the row indexes found have an error lower or equal to five pixels. In Figure 2.5 the results show that the method is able to detect lanes even in case of broken or faded lines and also if they are shaded.

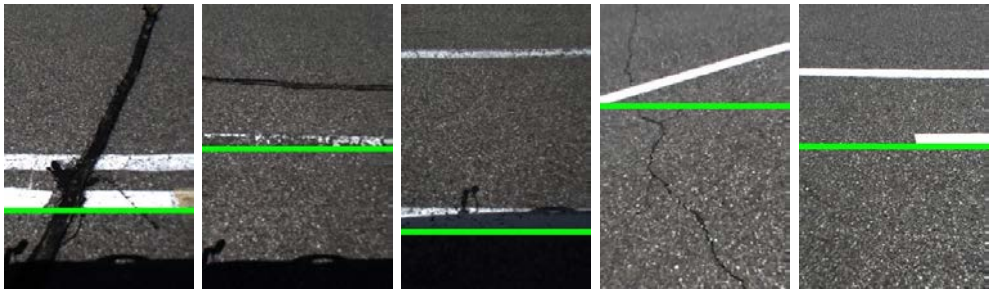


Figure 2.5: Examples of output images produced by DeepLanes. Images taken from [17].

Another one is [18], that presents an end-to-end trainable multi-task network called VPGNet. It predicts not only the lanes position and the marking types but also road sign markings and the vanishing point of the scene at the same time. The dataset used is made of about 20000 images captured under various weather conditions during different times of the day. When tested on Caltech lanes dataset, VPGNet obtains an F1 Score of 0.884 on the first set of data and 0.869 on the second one. An example of the output of this system is shown in Figure 2.6.

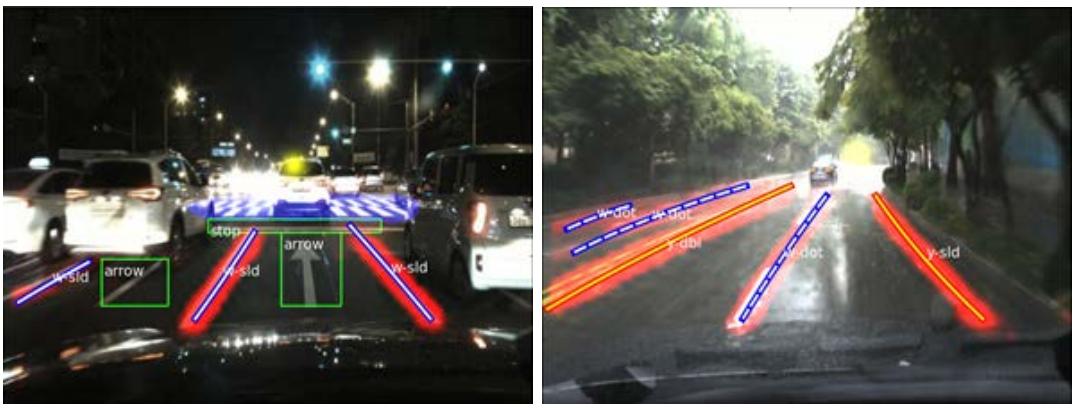


Figure 2.6: Examples of output images produced by VPGNet. Images taken from [18].

2.2.2 Semantic segmentation-based methods

Recent years have been characterized by the introduction of new approaches that make use of semantic segmentation for lane detection, making possible to take advantage of the various network architectures developed for this task. It is indeed a powerful method to parse an image and assign a category to each pixel exploiting the structure of the scene represented in an image. Nowadays almost every non-trivial semantic segmentation algorithm is based on CNNs.

The first end-to-end Fully Convolutional Network (FCN) for semantic segmentation was presented in 2014 by Long et al. [19]. It consisted of a common pre-trained CNN architecture on a large scale dataset (in this case ImageNet [20]) whose fully connected layers have been replaced with convolutional and upsampling layers. This new kind of architecture was called encoder-decoder: the encoder is used to extract and encode the information contained in the initial image into lower resolution feature maps, the decoder instead preserves the context information from the lower resolution maps while recovering fine-grained details through upsampling layers. The task related to the encoder can be done with state-of-the-art CNNs.

Starting from that moment more and more fully convolutional networks have been proposed, such as U-Net [21], SegNet [22], DeepLab [23], RefineNet [24] and PSPNet [25]. However, the main problem of these works was the high number of parameters of the models, which made them unsuitable for real-time applications.

Such problems were solved by adding custom layers like the *bottleneck* of ENet [26], whose aim is to reduce the number of parameters and floating-point required operations. Another example is ERFNet (Efficient Residual Factorized ConvNet) [27], a residual-based network that instead makes use both in the encoder and in the decoder of non-bottleneck-1D layers. Differently from previous networks, these layers do not imply 2D convolutions but instead a double number of 1D convolutions, leading to more accuracy and more efficiency at the same time. Moreover, the decoder is small and it does not contain max-unpooling layers but only simple deconvolution layers. This network achieves state-of-the-art performances on Cityscapes dataset but it is less computational demanding when compared to other models, making it suitable for real-time applications like those related to Intelligent Vehicles. Its architecture is shown in Figure 2.7.

After having introduced some architectures used for semantic segmentation, we present some applications focused on the detection of road markings.

In [28] Pan proposed an approach based on segmentation to find lane boundaries by using a new architecture called Spatial CNN (SCNN). This kind of neural network allows capturing spatial relationships of pixels across rows and columns and is particularly suitable for detecting long continuous shape structures or large objects, such as traffic

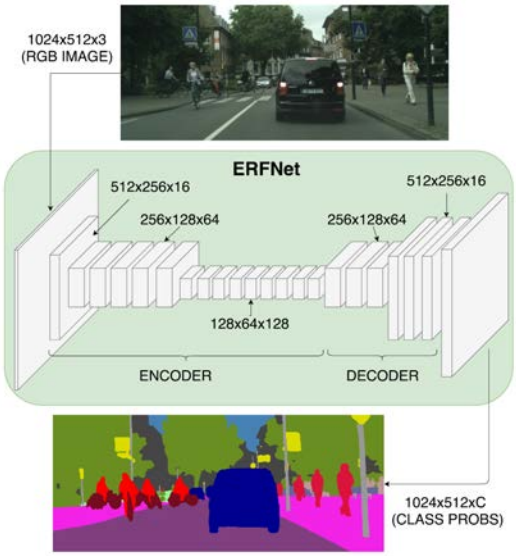


Figure 2.7: ERFNet architecture. Images taken from [27].

lanes. The network proposed achieved state-of-the-art results on the TuSimple [29] lane detection dataset in 2017. Figure 2.8 shows the output of SCNN compared with a classical CNN architecture for segmentation.

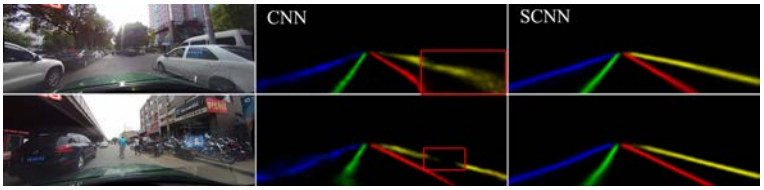


Figure 2.8: Comparison between CNN and SCNN in lane detection. Image taken from [28].

In [30], even if the main purpose is finding parking slots, road markings are required to be detected. Fisheye cameras are installed on each edge of the vehicle and images acquired are firstly undistorted, then perspective is warped and finally they are used for stitching, obtaining a Panoramic Surround View (PSV) image. The network used for segmentation is based on an HFCN (Highly Fused Convolutional Network) [31] and an extra VH-stage is performed before the up-sampling operation to further learn linear features without destroying feature information learnt in lower layers of the network. This stage is important since lane markings appear in PSV images as vertical lines while parking slots as perpendicular lines. Figure 2.9 shows some examples of detection.

In [32], the authors make us aware of a limitation on the use of segmentation for lanes detection: these methods are limited to detecting all the lanes as a unique one or at most they can detect a pre-defined number of lanes considering each of them as a different class (as done in [33]). So they propose to cast detection problem as an

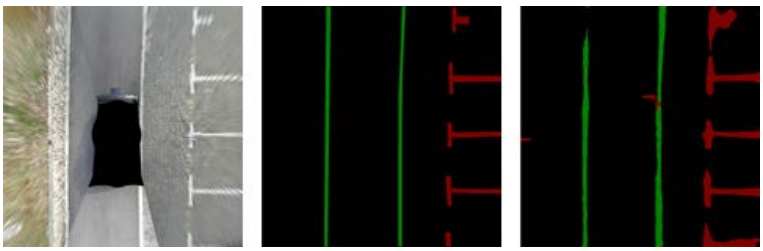


Figure 2.9: From left to right: PSV image, Ground Truth and result of VH-HFCN. Images taken from [30].

instance segmentation problem. This means that the network predicts each lane as a unique instance and so the number of output lanes is not fixed. The network built is called LaneNet. After a shared encoder, initial part is split into two branches: a segmentation one, which generates a binary mask indicating the position of lanes, and a pixel embedding branch. The output is clustered to obtain lane instances. The system also makes use of a second network called H-Net used for subsequent lane fitting. This system reaches the 4th place on the TuSimple challenge [29].

An accurate lane-level localization approach using an Around-View-Monitoring system is used in [34]. Road features like boundaries and markings are detected by using pixel-wise segmentation of raw fisheye images. Differently from others, this paper proposes coarse-scale localization (CSL) and fine-scale localization (FSL) methods for high-accurate localization. The experiments in urban environments demonstrate that the proposed approach achieves centimeter-level localization accuracy with five centimeters in the lateral direction and seventeen centimeters in the longitudinal direction.

Today's state-of-the-art method is ENet-SAD [35]. A novel knowledge distillation approach, called Self Attention Distillation (SAD), allows a model to learn from itself and gains substantial improvement without any additional supervision or labels. What the authors of the paper observe is that attention maps extracted from a model trained to a reasonable level would encode rich contextual information, that can be used as a form of "free" supervision for further representation learning through performing topdown and layer-wise attention distillation within the network itself. SAD has been incorporated in ENet model and the result is a new light model that overperforms the state-of-the-art SCNN, reaching an accuracy of 96.64% on TuSimple dataset [29]. Figure 2.10 shows an example of the application of this method.

Figure 2.10: From left to right: input image, result of ENet-SAD (97.2% of accuracy) and result of SCNN (95.2% of accuracy). Images taken from [35].

2.3 Our work

As seen is previous paragraphs, most of the state-of-the-art methods for lane detection use only frontal pinhole cameras (the works presented in [5–8, 10, 18, 28, 35] are some examples) with a limited field of view and consider the perspective to understand if a light segment on the surface is actually a road marking. More recent works use instead fisheye cameras, that are laterally mounted, like in [15], or positioned on the four edges of the vehicle. Then, an Inverse Perspective Mapping is computed to undistort images (see also [30]), in order to use the algorithms originally thought to work with pictures taken with conventional cameras.

What makes our work innovative is the use of only two laterally-mounted fisheye cameras, taking advantage of their large field of view to capture as much information as possible from the surroundings. They are indeed used for short-range monitoring since the road boundaries at left and at right are supposed to be close to the vehicle when it is inside its lane. Apart from the kind of cameras utilized, another innovative idea is the direct work on the images obtained after the mapping from the captured ones to those that adopt the spherical model. No other computation, such as Inverse Perspective Mapping (IPM), is considered, in order to keep the computational complexity of the system as low as possible allowing it to be eventually ported to embedded platforms. However, since no datasets with this type of images are publicly available, a new one has been created and manually annotated, drawing in each image the segments that identify the position of the middle of the principal line.

The use of two different approaches, a more traditional one and a more recent one, gives us the possibility of understanding in which scenarios the first works better than the other, or vice versa. This makes us conscious of the advantages and the limitations of them, allowing the proposal for a future approach that exploits the strengths of each of them.

3

Spherical model

Wide angle cameras such as fisheyes are particularly suitable for lateral views as they let us observe all the surrounding objects from the rear to the front of a vehicle. In this chapter, spherical model, that is the one that we use, other than one of the most popular representation of fisheye images, will be explained theoretically.

3.1 Fisheye lenses

Fisheye lenses are ultra wide-angle lenses whose main goal is to provide images with a large Field-Of-View. Their name comes from fishes vision from beneath the water, since what they see is an ultrawide hesemispherical view. The term *fisheye* was coined in 1906 by Robert W. Wood, an American physicist and inventor. Applications are the most disparate, for example video-surveillance, augmented reality, in action cameras and in automotive. In the latter they are broadly used because using only four fisheye cameras is enough to have a 360 degrees view of what is close to the vehicle (we call it all-around-view monitoring).

However, having such a wide field-of-view has as direct consequence the fact that, when the acquired image has to be mapped to a plain surface, a non-linear mapping that causes strong visual distortions is needed. The geometry that describes this mapping is complex. Note that the distortion in the central part of the image is low but it becomes particularly visible as we go to the edges.

Unfortunately, classical Computer Vision methods, that are developed to work with images obtained with pinhole cameras, do not generalize well to this type of images. This means that we have two possibilities: make fisheye images to be as similar as possible to pinhole ones or adapt algorithms to the new camera model.

3.2 Spherical model in details

Images captured with cameras equipped with fisheye lenses can be projected onto a virtual unitary sphere. This is the result of the unified model formalized for omnidirec-

tional vision. As the field-of-view is not clearly of 360 degrees but it is usually limited to 180 degrees, not all the area of this sphere will be covered by acquired points. This is not the only existing model: as an alternative, we can indeed consider the cylindrical model. The main advantage of spherical model against others is the quantity of details that are kept with the mapping, greater compared to the cylindrical model, especially vertically. Figure 3.1 shows an example of mapping of a fisheye image to the unitary sphere. Figure 3.2, instead, shows how an image captured with a fisheye lens is then projected adopting the spherical model.

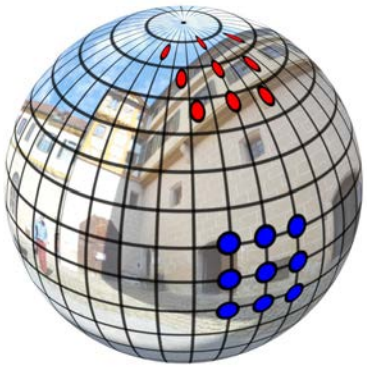


Figure 3.1: Spherical representation of a fisheye image. Image taken from [36].

Figure 3.2: At left, a captured fisheye image. At right, its representation adopting the spherical model.

When working with images captured by a camera, and more specifically in automotive environments, we consider five reference systems:

World frame: it is the absolute reference system, integral with the ground and fixed over time. It is also known as $body_0$ since its position generally corresponds to the

location of the Body frame when the vehicle is in its initial pose;

Body frame: it is integral with the vehicle and positioned in front of it. For this reason it is sometimes called *vehicle* frame. Extrinsic parameters obtained with camera calibration and that the images provide about its position are given with respect to this frame. The reason for having this frame is to have a common coordinates system for all the sensors that can be installed into the vehicle. In this way, sensor fusion operations become easier since all the data the sensors have captured can be referred to the same frame;

Sensor frame: it is a Front-Left-Up (X the depth, Y increasing to the left and Z increasing to the top) reference system integral with the camera;

Camera frame: its origin is the same of the sensor frame, but it is a Right-Bottom-Front (X increasing to the right, Y increasing to the bottom, Z the depth) reference system. This means it is obtained from the sensor frame with a simple swap of the axes. With respect to the image frame, its origin is in the position (u_0, v_0);

Image frame: it is used to indicate the position, in pixels, of a point in the image. Only two components are defined, that are u and v: u indicates the column number, v the row number.

Figure 3.3 shows the relative position of each of the frames just introduced. Note that, only for simplicity, in this case sensor frame is obtained without additional rotations of the camera, differently from what usually happens. Sensor and camera frames have the same origin but they are shown separately only for visualization purpose.

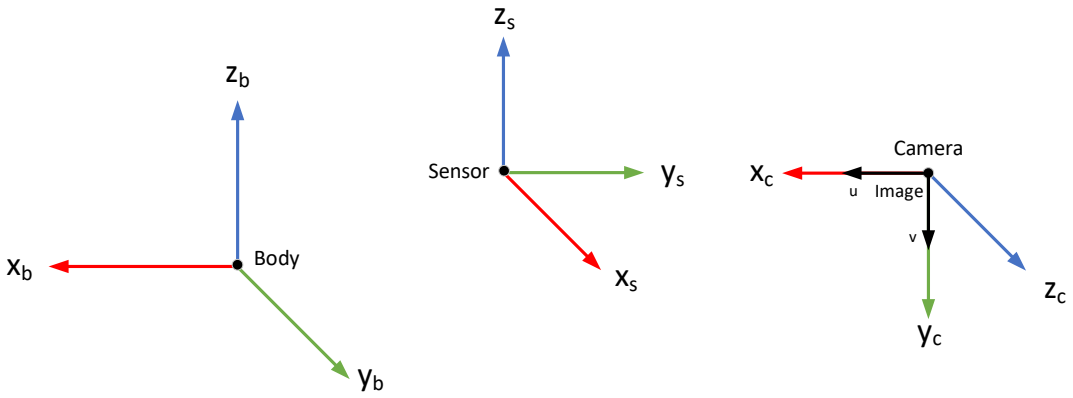


Figure 3.3: Body, sensor, camera and image frames.

We can define a point in camera frame as:

$$\begin{cases} x = t \cdot \sin(\alpha_u) \\ y = t \cdot \cos(\alpha_u) \cdot \sin(\alpha_v) \\ z = t \cdot \cos(\alpha_u) \cdot \cos(\alpha_v) \end{cases} \tag{3.1}$$

t represents the length of the ray that links a point in camera frame to the corresponding one in body frame. α_u and α_v are defined as follows (see also Figure 3.4):

$$\alpha_u = \frac{u - u_0}{k_u} \qquad \alpha_v = \frac{v - v_0}{k_v} \tag{3.2}$$

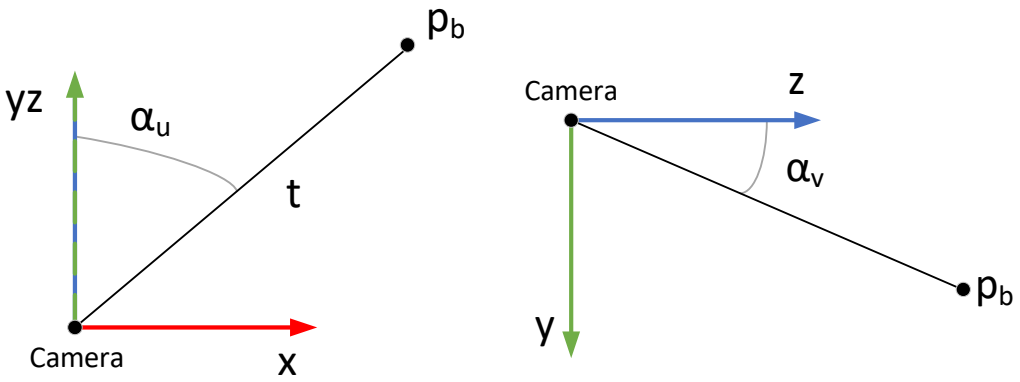


Figure 3.4: α_u and α_v representations with respect to the axis of the camera frame.

u_0, v_0, k_u and k_v are intrinsic parameters of the camera. u_0 and v_0 indicate the position of the origin of the camera frame with respect to the upper-left point of the image provided by the sensor, expressed in pixels. k_u and k_v are the inverse of the pixel dimensions in the u and v directions and they are expressed in pixels/radians:

$$k_u = \frac{W}{FOV_h} \qquad k_v = \frac{H}{FOV_v} \tag{3.3}$$

W and H are the width and the height of the image, whereas FOV_h and FOV_v are respectively the horizontal and vertical Fields-Of-View of the fisheye camera used.

Given the extrinsic parameters of the camera (translation and rotation), we can define the translation vector as:

$${}^bT_c = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix}^T \tag{3.4}$$

Note that, since there is no translation between sensor and camera frames, the matrix provided can be used to go directly from camera to body frame.

The rotation matrices are defined respectively for *roll*, *pitch* and *yaw* angles as:

$$R_\rho = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\rho) & -\sin(\rho) \\ 0 & \sin(\rho) & \cos(\rho) \end{bmatrix} \tag{3.5}$$

$$R_\theta = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{3.6}$$

$$R_\gamma = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

The rotation matrix from camera to body frame is:

$$^bR_c = {^cR_b}^{-1} \tag{3.8}$$

$$^cR_b = {^c\Pi_s} \times R_\rho^{-1} \times R_\theta^{-1} \times R_\gamma^{-1} \tag{3.9}$$

where $^c\Pi_s$ is called axes permutation matrix. This matrix is needed to go from the sensor Front-Left-Up reference system to the camera Left-Bottom-Front coordinates system. The former is typical in the automotive environment. So, to deal with this difference, we define:

$$^c\Pi_s = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \tag{3.10}$$

Once we have the rotation matrix ^cR_b and the translations, we can define the rototranslation matrix from camera to body frame as:

$$^bRT_c = \left[\begin{array}{ccc|c} \ddots & \vdots & \iddots & \vdots \\ \cdots & {^bR_c} & \cdots & {^bT_c} \\ \iddots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.11}$$

We can easily calculate for each point in camera frame the corresponding point in body frame in this way:

$$p_b = {^bRT_c} \times p_c \tag{3.12}$$

Note that p_b and p_c are expressed in homogeneous coordinates.

4

First approach: traditional computer vision

In the previous chapters, state-of-the-art techniques have been presented. Our first goal is to provide an innovative approach to the problem of lane detection without using neural networks. The images used are captured with two lateral fisheye cameras and the spherical model introduced in Chapter 3 is adopted.

4.1 Pros and cons

As briefly mentioned in the introduction, the use of Computer Vision methods is the older and, in a certain way, the simpler approach to detection problems. They are however of proven reliability and require less computational resources than other methods. Nevertheless, they have one important advantage when compared to Deep Learning techniques: we can easily understand why something works or not and, consequently, it is simpler to take the most adequate countermeasures. By using this approach everything is deterministic.

On the other hand, traditional Computer Vision also has drawbacks: the number of hand-crafted parameters needed for a good detection sometimes tends to explode. These ones also need to be tuned and every time a new one is added a new tuning could be necessary, leading to multiple trade-offs.

4.2 Description of the approach

Our approach is made of several steps. We start with a low level processing, considering the single pixel as the reasoning unit. Then we will talk about sets of pixels called clusters. Since they should contain all the points that belong to a single line, they are forced to be part of a line by fitting them with a polynomial function. Finally, our reasoning is made on these lines, whose detection is the main goal of this application. In the next paragraphs all the steps followed by our algorithm are described in detail.

4.2.1 Images preprocessing

All the images used have a size of 1024x992 pixels (1024 rows and 992 columns) and are in YUV (YCbCr to use the correct name when we consider digital coding like our case) format, specifically NV12. It is specified as follows:

- Y (luminance) channel is made of the first $width * height$ bytes, since all the values are represented with 8 bits;
- C_b (blue chrominance) channel is made of $(width * height)/2$ bytes;
- C_r (red chrominance) channel is made of other $(width * height)/2$ bytes.

C_b and C_r bytes are interleaved. The division by two of the number of bytes of C_b and C_r channels is due to the downsampling applied to them. The reason can be found in the Human Visual System: human eyes are more sensible to luminance than chroma, so this fact can be exploited by compressing the quantity of information regarding colors. The images are then converted into RGB color space and finally rotated by 90 degrees counterclockwise since the vehicle used for tests has installed cameras that are rotated by 90 degrees clockwise.

The other preprocessing steps are:

Colors to grayscale: our purpose is to find road markings and more specifically the principal one. The lack of color information makes impossible to understand the meaning of a line. However, we suppose the presence of horizontal signage on the road surface. We decided to convert all the images from RGB color space to grayscale with the only aim of simplify the problem and hence the algorithm. Adding colors will be an extension of the problem. Due to this conversion, images that are used in next steps have a single channel and 8 bits per pixel.

Scaling: in order to deal with a lower number of pixels and so to speed up all the operations that are part of the algorithm, the input image has been scaled by a factor of 2, causing the size to become 496x512 (we are using as before the convention *rows* x *columns*, but remember that the image has been rotated). This leads to a loss of small details, but at the same time this allows us to reduce the noise captured by the sensor.

Progressive Gaussian Blur: the surface of the road is obviously not smooth and the asphalt often has irregularities. They cause the image to have a lot of useless details in the section where the lines are expected to be. However, we do not need a such level of detail, we only need a coarse representation of what the road

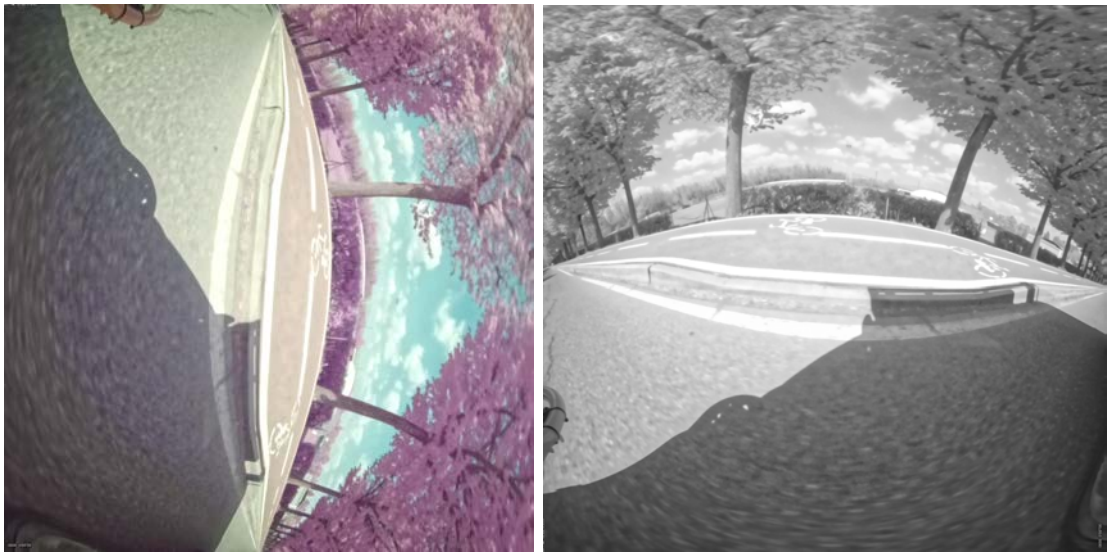


Figure 4.1: Input image at left; After rotation and grayscale conversion at right.

surface contains. We do not pretend to detect lines that have a representation thinner than, for example, 2 pixels. The details captured become noise for our algorithm, so to eliminate the biggest part of it we decided to apply a Gaussian Blur to the image. Since details are sharper as we get closer to the camera, blur is not identically applied to all the image, but we consider a kernel size that decreases as we get far from the camera: we start from a 9x9 kernel, then 7x7, 5x5 and finally 3x3 for the zone where no interesting lines are supposed to be found. In Figure 4.2 an example of application of this progressive Gaussian Blur is shown.

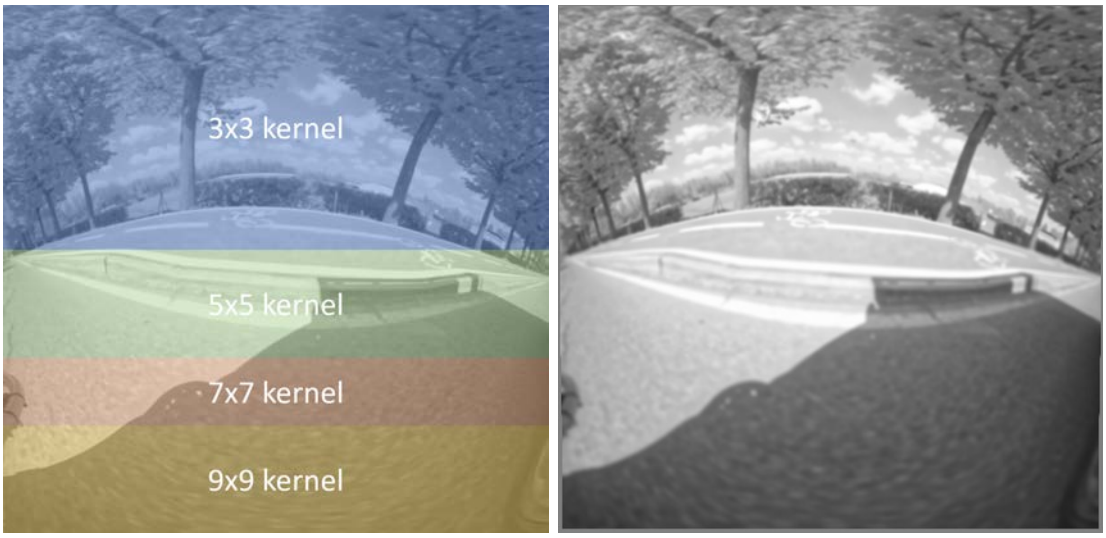


Figure 4.2: Progressive Gaussian Blur.

Crop: the input image contains the road surface, but also a lot of details that are not useful for our final purpose. So we decided to crop the picture vertically by eliminating the first third of the image starting from the top. Our choice is motivated by the fact that that area contains what is over the horizon and surely we will not find a road marking here. Note that this is valid only for our specific camera setup.

We also decided to crop the image horizontally, keeping out of our Region-Of-Interest 1/15 of the image at left and at right. This time the reason is only practical: the distortion caused by the fisheye lens makes the lines to converge at the edges and so, taking as an example the image provided in Figure 4.1, the line that delimitates the roadway and the closer one that belongs to the bicycle path become difficultly to be distinguishable.

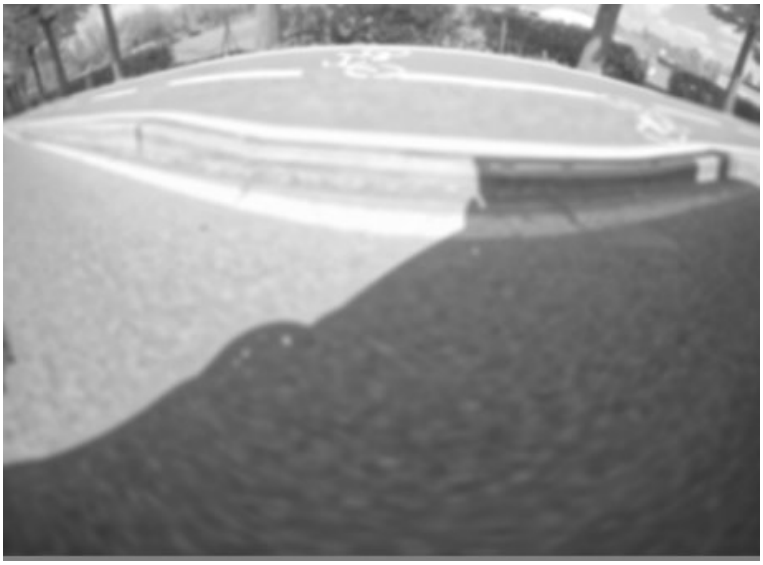


Figure 4.3: Images obtained after the extraction of the ROI.

4.2.2 Low-level lines extraction

At this point, the image to be processed is the one of Figure 4.3. Starting from here, the image progressively loses all the parts that are useless for our final purpose.

In the following subparagraphs all the approaches that have been followed will be presented, indicating what are the advantages and which drawbacks have been decisive to make us choose a different path.

Enhancement of input image: the first idea consisted in applying a gamma curve to the input image with a value of gamma lower than one. The aim is to highlight road markings by eliminating dark areas of the grayscale image and enhancing

the lighter ones. In this way we try to lose information about non-lines zones as much as possible, avoiding at the same time the remained noise caused by the irregularities of the asphalt. We remind that the biggest part of it has already been removed with Progressive Gaussian Blur, as described in Paragraph 4.2.1. However, to keep the computational complexity as low as possible, we decided to replace the gamma curve with a linear ramp that leads to a similar result. The function applied to each pixel of the image is the following:

$$enhancedImage[i,j] = \max\left(255 \cdot \frac{(originalImage[i,j] - x)}{255 - x}, 0\right)$$

In the last expression x corresponds to the enhancement strength, that is a value from 0 to 255.
x + 1 is the minimum value of brightness of a pixel that will be kept. The value of enhancement strength is chosen by an algorithm that analyzes the exposure of the image: the more the image is overexposed, the more the strength is.

The main problem of this idea is that if a road marking is not white or is deteriorated due to its age or it is simply in a shadow, this enhancement is likely to cause lines to disappear.

To solve this issue we decided to combine two images: the first one is the image obtained with the algorithm just presented whereas the second one is the result of contrast enhancement. The resulting image will take from the first one the zones where no information is lost (the value of the enhanced pixel is different from 0), while all the other pixels will be taken from the picture with enhanced contrast. In this way both clear and in-the-shade road markings become more visible, but at the same time there are a lot of small dark zones on very light background: the first ones are caused by the not complete loss of information resulting from the application of the ramp function, the second ones by the saturation obtained as consequence of the contrast increase in already bright zones. These significant brightness differences make lines difficult to be distinguished from artifacts since what we will use to find them are dark-light transitions. Figure 4.4 shows an example of the application of this method.

Also local enhancements methods that take into account the average brightness in windows of 5x5 pixels have been considered, but the result in every case has been more a noise highlight than more visible lines. These reasons have made us decide to avoid using any enhancement method and use the original image for the next steps.

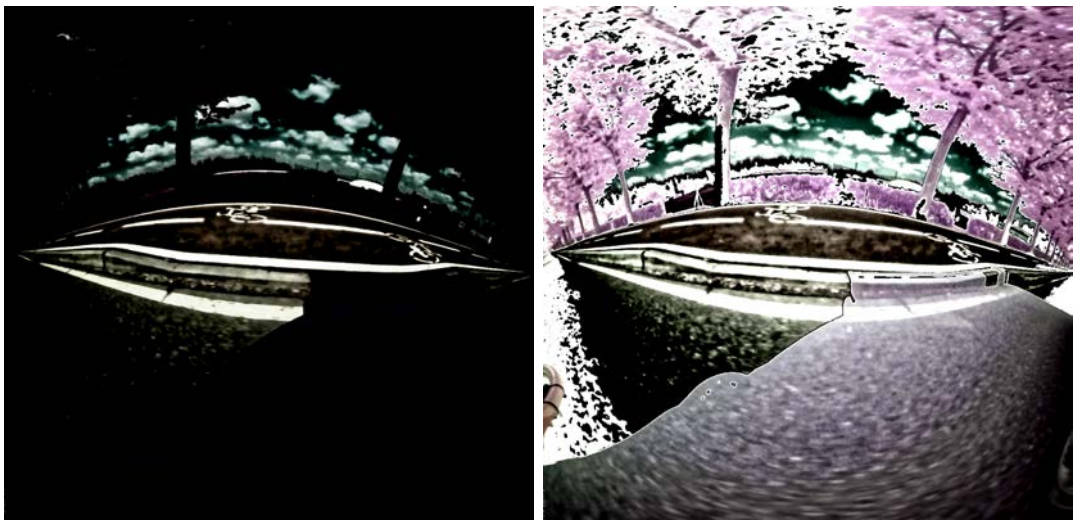


Figure 4.4: Enhancement with ramp function and hybrid method.

Extraction of DLD transitions: the idea comes from the observation of road markings. We can easily note that a line is a bright zone surrounded by a darker area. This pattern is called Dark-Light-Dark (DLD) and it is the main and more visible feature that distinguishes a line from the rest of the road surface.

The implementation of the algorithm that extracts Dark-Light-Dark transitions is based on [7]. It was modified to adapt to our case since the images the original version is supposed to deal with are frontal and obtained with an Inverse Perspective Mapping (IPM). As we are using lateral images, we changed the algorithm to support horizontal lines. From the initial image we obtained a new one defined as follows:

$$\begin{aligned} DLD[i,j] = \max\Bigg(\min\Bigg(&image[i,j] - image\left[i, \frac{kernelSize}{2}\right], \\ &image[i,j] + image\left[i, \frac{kernelSize}{2}\right]\Bigg), 0\Bigg) \end{aligned} \tag{4.1}$$

The size of the kernel is not the same for the whole image, it is progressively decreased as we get far from the camera: the value goes from 2 to 40. This variation is essential because the kernel dimension should be as close as possible to the size in pixels of the line that has to be found. At the beginning the variation used was linear, but then we decided to move on to a non-linear one considering the distortion caused by the fisheye lens. Also kernel orientation should have followed the same distortion. The results with linear variation were bad as expected, instead with the non-linear variation they were good enough but the approach was abandoned since the computational complexity was too high and not acceptable for a

real-time application: the use of kernels with a dimension up to 40 was certainly the responsible, but it was not avoidable.

Extraction of DL and LD transitions: this approach is based on [5,6]. Unlike the previous one, what we search is not a Dark-Light-Dark pattern but more simply significant differences of brightness. In other terms, we are searching Dark-Light and Light-Dark patterns separately. This time the size of the kernel is fixed to 11, whereas the orientation is still variable. Figure 4.6 shows the eight different kernels used.

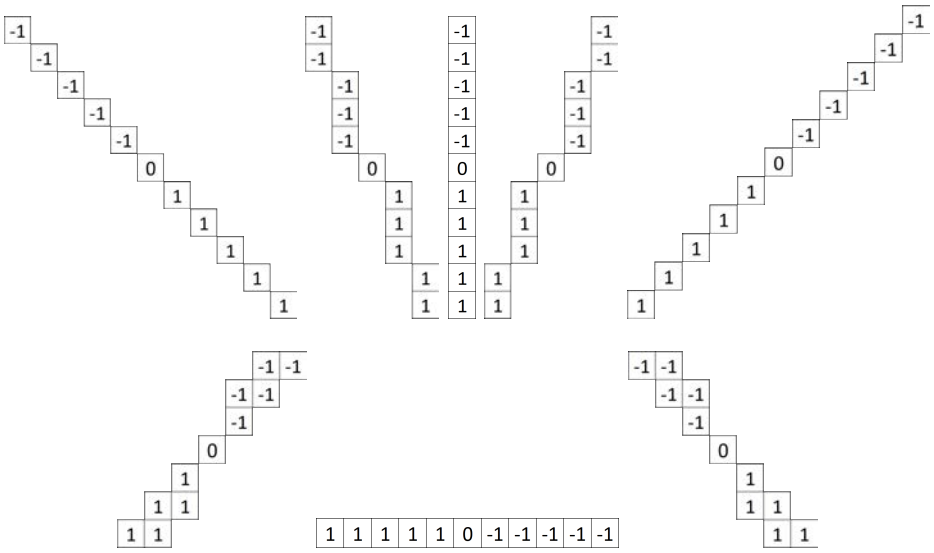


Figure 4.5: Kernels initially used for searching DL transitions.

Each kernel is used for convolution in different zones of the image, following the expected orientation of a line. However, this is a limitation since we are expecting to have only road images like the one in Figure 4.1 where the vehicle is perfectly parallel to the line and it is in a straight road, causing a more difficult detection of lines on curves. For this reason, we decided to replace all these kernels with only three: the vertical one, the horizontal one and its opposite. The vertical kernel is used for convolution on the whole image, the horizontal one shown in Figure 4.5 is used only in the left part of the image while the last one on the right part. The choice of having two different horizontal kernels is motivated by the fact that we want each line to be represented as a rising edge and a corresponding falling edge. By using lateral cameras and having the most of the lines to be horizontal, what we will obtain by using the same kernel from the left side to the right side of the image (by using the opposite one on the whole image the result is similar) is an image where a road marking is represented on the left as a falling edge on top with its corresponding rising edge on bottom and a rising edge on top with

its corresponding falling edge on bottom as we go to the right side. This causes a more difficult search of corresponding edges since they have a different meaning on the left and on the right side. Our approach with two opposite kernels makes the problem easier for the next steps. So what we have done is calculating the horizontal and the vertical gradients. In Figure 4.6 the magnitude of the gradients is shown.

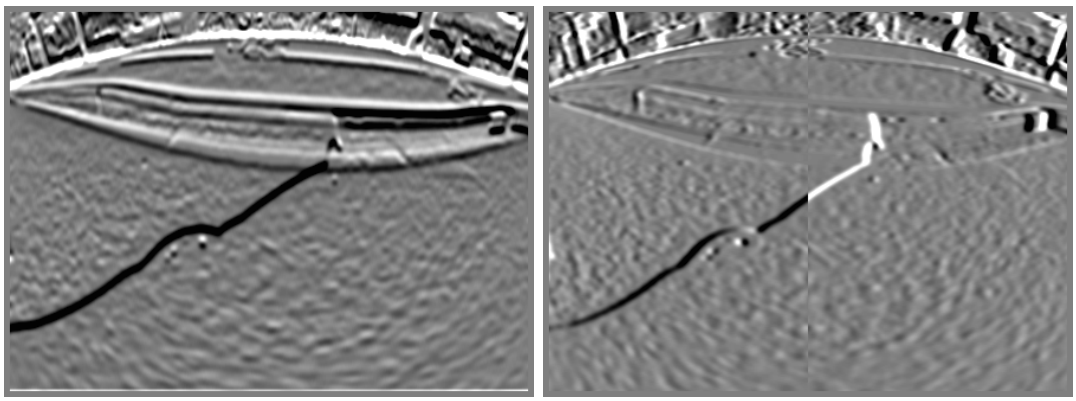


Figure 4.6: Vertical and horizontal gradients magnitude.

The next step consists in obtaining the total gradient magnitude and orientation by merging the ones just calculated. The gradient magnitude is calculated as:

$$m[i,j] = \text{sign}(m_v[i,j]) \cdot \sqrt{(m_v[i,j])^2 + (m_h[i,j])^2} \tag{4.2}$$

where m_v is the vertical magnitude and m_h is the horizontal magnitude. Since with lateral cameras it is more probable to see horizontal lines (that are best highlighted with a vertical kernel), the sign of the vertical magnitude has been taken as the overall one.

The values of orientation are calculated as

$$o[i,j] = \text{atan2}(m_v[i,j], m_h[i,j]) \tag{4.3}$$

and they has been quantized for simplicity and to avoid close points to have different gradient orientation only due to noise. Different colors of Figure 4.7 have this meaning. Not all the classes used for quantization have the same size: for example, we decided all orientations going from 56.25 to 123.75 degrees to be part of the same class since the vertical kernel has resulted to be able to highlight most of the edges having these orientations. Moreover, we omitted all the orientations close to horizontal as they were likely to belong to noise instead of significant lines that had to be detected.

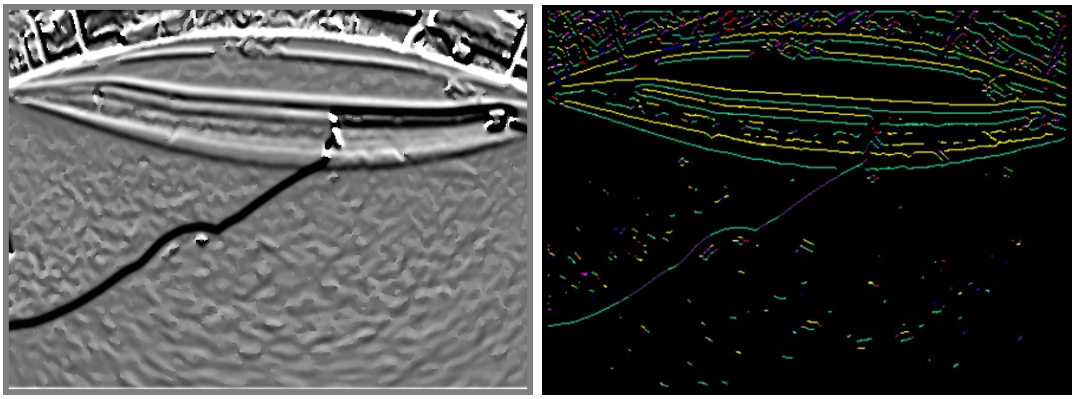


Figure 4.7: Gradient magnitude and orientation quantized map.

At the end of DL-LD transitions extraction, non-maxima suppression is performed. As suggested in [5], it has been done in a little different way from usual, mainly because we are using a wide kernel and it sometimes means a slower variation of the gradient as the distance from the edge increases. Algorithm 1 describes how our method works.

Algorithm 1: Modified non-maxima suppression

if $|input[i,j]| < |input[i-1,j]| \vee |input[i,j]| < |input[i+1,j]|$ **then**
  $output[i,j] = 0$;
else if $|input[i,j]| = |input[i-1,j]| \wedge |input[i,j]| < |input[i-2,j]|$ **then**
  $output[i,j] = 0$;
else if $|input[i,j]| = |input[i+1,j]| \wedge |input[i,j]| < |input[i+2,j]|$ **then**
  $output[i,j] = 0$;
else $output[i,j] = input[i,j]$;

After that, applying a threshold is useful to keep only the gradients that correspond to the greatest brightness variations. This is done particularly because also all the irregularities of the pavement are highlighted by the kernel applied since they respond well to the DL or LD pattern. Figure 4.8 shows the results of non-maxima suppression and thresholding.

4.2.3 DLD pairs search

For each of the resulted maxima obtained after the thresholding step, we want to perform a search of the corresponding minimum, as suggested in [6]. The problem now encountered regards the interval of the search. According to the 40^{th} article of Italian traffic laws, road markings can have a thickness that goes from 10 to 15 centimeters, based on the type of road (urban or highway). To keep a certain degree of tolerance,

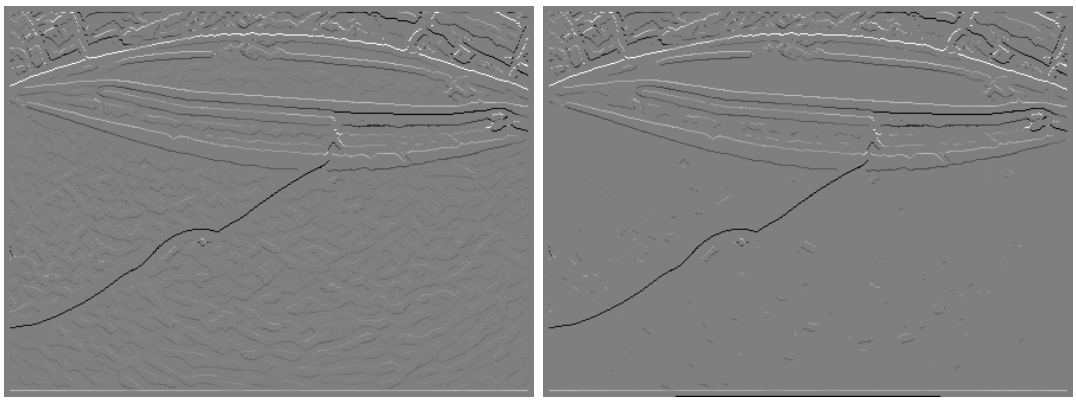


Figure 4.8: Non-maxima suppression and thresholding.

we decide to consider as minimum and maximum distance between a rising edge and its corresponding falling edge respectively 8 and 22.5 centimeters. However, we need to convert this measure in the real world to a number of pixels in our image.

What we need to do is to find for each point in camera frame the corresponding point in body frame, then to select in this coordinates system the points that are respectively 8 cm and 22.5 cm far from it (going vertically). These two points are then reprojected in camera frame and the distance calculations from the origin point are made. These distances are then saved in two look-up tables, one for the minimum and one for the maximum number of pixels to check during the search of pairs.

Using all intrinsic and extrinsic camera parameters, we can find a mapping between coordinates in camera frame and coordinates in body frame. We need, however, to suppose that all the points we are taking in the body frame are on the ground (this means that all the points in body coordinates system must have z = 0). This condition is correct in most of the cases, but it can cause problems if part of a line is above a bump. All the points that in the reality do not respect this supposition are wrongly mapped, especially the ones that do not belong to the road, for example over the horizon points.

Remembering the spherical model presented in Chapter 3, we can define a point in camera frame as:

$$\begin{cases} x = t \cdot \sin(\alpha_u) \\ y = t \cdot \cos(\alpha_u) \cdot \sin(\alpha_v) \\ z = t \cdot \cos(\alpha_u) \cdot \cos(\alpha_v) \end{cases} \tag{4.4}$$

t represents the length of the ray that links a point in camera frame to the corresponding one in body frame. Each point (u, v) of the image is associated to a pair of angles (α_u, α_v) and the relation is indicated in Equation 3.2. However, the camera

frame coordinates of Equation 4.4 are dependent from t, so for each point of the image we have to calculate it.

Starting from the basis, the canonical equation of a plane on a three-dimensional space is:

$$ax + by + cz + d = 0 \tag{4.5}$$

Given a point $O_b = (0, 0, 0, 1)$ and a versor $v_b = (0, 0, 1)$ (note that the point satisfies the z = 0 condition), to obtain the equation of the plane in the body frame we substitute a, b and c coefficients with the components of the vector v_b. We get:

$$z + d = 0 \tag{4.6}$$

Now, we impose the passage for O_b, obtaining:

$$d = 0 \tag{4.7}$$

We can now substitute the value of d just found in Equation 4.6. The obtained plane is:

$$z = 0 \tag{4.8}$$

The corresponding point and versor in camera frame O_c and v_c can be obtained by using the inverse of Equation 3.12:

$$O_c = {}^cRT_b \times O_b \tag{4.9}$$

$$v_c = {}^cR_b \times v_b \tag{4.10}$$

As we have done for body frame, in camera frame the plane where all points are situated is defined by the Equation 4.5. We can now substitute the coefficients of v_c and we impose the passage for O_c, obtaining

$$v_c^0 \cdot O_c^0 + v_c^1 \cdot O_c^1 + v_c^2 \cdot O_c^2 + d = 0 \tag{4.11}$$

and by consequence

$$d = -v_c^0 \cdot O_c^0 - v_c^1 \cdot O_c^1 - v_c^2 \cdot O_c^2 \tag{4.12}$$

v_c^0, v_c^1 and v_c^2 are respectively the first, second and last component of the vector v_c. O_c^0, O_c^1 and O_c^2 are the same for the point O_c.

For each point of the camera frame p_c ,whose components are defined in Equation 4.4, we can impose the passage for it. Note that these components are all still dependent from t. So we get:

$$v_c^0 \cdot p_c^x(t) + v_c^1 \cdot p_c^y(t) + v_c^2 \cdot p_c^z(t) + d = 0 \tag{4.13}$$

And then the value of t for a specific point in camera frame is computed as:

$$\begin{aligned} t &= \frac{-d}{v_c^0 \cdot \sin(\alpha_u) + v_c^1 \cdot \cos(\alpha_u) \cdot \sin(\alpha_v) + v_c^2 \cdot \cos(\alpha_u) \cdot \cos(\alpha_v)} \\ &= \frac{v_c^0 \cdot O_c^0 + v_c^1 \cdot O_c^1 + v_c^2 \cdot O_c^2}{v_c^0 \cdot \sin(\alpha_u) + v_c^1 \cdot \cos(\alpha_u) \cdot \sin(\alpha_v) + v_c^2 \cdot \cos(\alpha_u) \cdot \cos(\alpha_v)} \end{aligned} \tag{4.14}$$

Now we know the values of t, so we can calculate the values of the components of each point in camera frame with the equations given in 4.4. For each of them a projection on the body frame is done by using Equation 3.12. The point at minimum distance in body coordinates is:

$$p_{b_{min}} = (p_b^x, \quad p_b^y + 0.08, \quad p_b^z, \quad 1) \tag{4.15}$$

It is reprojected in camera frame:

$$p_{c_{min}} = {}^cRT_b \cdot p_{b_{min}} \tag{4.16}$$

The value of t for this point is calculated as:

$$t_{c_{min}} = \sqrt{{p_{c_{min}}^x}^2 + {p_{c_{min}}^y}^2 + {p_{c_{min}}^z}^2} \tag{4.17}$$

The values of α_u and α_v that correspond to this point are obtained as:

$$\alpha_{u_{min}} = \arcsin\left(\frac{p_{c_{min}}^x}{t_{c_{min}}}\right) \qquad \alpha_{v_{min}} = \arcsin\left(\frac{p_{c_{min}}^y}{t_{c_{min}} \cdot \cos(\alpha_{u_{min}})}\right) \tag{4.18}$$

The values of u_{min} and v_{min} are calculated in this way:

$$u_{min} = \alpha_{u_{min}} \cdot k_u + u_0 \qquad v_{min} = \alpha_{v_{min}} \cdot k_v + v_0 \tag{4.19}$$

At the end the minimum distances in u and v directions from the original point p_c are computed as

$$minDist_u = \lfloor |u_{min} - u| \rfloor \qquad minDist_v = \lfloor |v_{min} - v| \rfloor \tag{4.20}$$

and they are saved in two look-up tables to have them available during the search

of pairs.

The same procedure is followed to calculate the maximum distances in u and v directions, replacing 4.15 with

$$p_{b_{max}} = (p_b^x, \quad p_b^y + 0.225, \quad p_b^z, \quad 1) \tag{4.21}$$

To sum up, at this point we have two matrices that, for each pixel of the image containing a falling edge of the image gradient, contain respectively the minimum and the maximum distance from it to the corresponding pixel that should contain its rising edge. These distances are not in meters but in pixels and the conversion has been possible thanks to the intrinsic parameters of the camera used. For the maximum distance, an additional tolerance in pixels is considered. Moreover, the values of minimum and maximum distance calculated are not comprehensive of the gradient orientation, they only indicate how far is the corresponding pixel vertically. This is why an additional calculus is needed:

$$minDist[i,j] = \text{round}\left(\left|\frac{minDist[i,j]}{\sin(o[i,j])}\right|\right); \tag{4.22}$$

$$maxDist[i,j] = \text{round}\left(\left|\frac{maxDist[i,j]}{\sin(o[i,j])}\right|\right); \tag{4.23}$$

The pairs search is done on the image obtained from the non-maxima suppression and thresholding. The steps are the following:

1. We start from the bottom of the image and do a row-major scan of the image;
2. For each pixel having a negative value (that corresponds to the belonging to a falling edge) we start exploring the surrounding pixels following the opposite of the gradient orientation calculated for the current pixel;
3. If a pixel with a positive value (that corresponds to the belonging to a rising edge) is encountered at a distance between the minimum and the maximum value, additional checks are performed:
   - The average value of brightness of pixels that separate this pixel from the origin is calculated to be used in the next point;
   - The pair gradients must be similar in magnitude and opposed in sign. A similarity index based on [5] is computed in this way:

     $$\text{S}(M1, M2) = \frac{|M1 + M2|}{\max(-M1, M2)} \tag{4.24}$$

     It is important to remember that M1 is the value of a pixel that belongs to

a falling edge, so it has a negative value. This explains the signs used. If the average value of brightness just calculated is greater than a threshold $T_{brightness}$, the similarity must be smaller than a *similarity high threshold* T_{high}. In the opposite case the similarity must be smaller than a *similarity low threshold* T_{low}. The explanation is the following: if the average brightness of pixels among the pair considered is high, we are more sure we are considering a road marking and therefore we can relax the conditions of gradient similarity. In the other case the condition is more strict because we are not so sure that we have a lane marking. The values used are $T_{brightness} = 120$, $T_{high} = 0.8$ and $T_{low} = 0.3$;

- The difference of brightness among the first point inside the possible line and the first point outside, calculated for the origin pixel but also for the target one.

If the conditions indicated in the second point are respected, the pixel is considered as a candidate for the pairing. At the end, the chosen pixel is the one whose value of magnitude is the closest to the origin one (in absolute value) and that have a lower difference between the two values calculated in the third point. Note that this last condition makes the algorithm capable of finding in a new way DLD transitions combining DL and LD transitions.

The choice of starting from a falling edge and searching the corresponding rising edge instead of doing the opposite is motivated: since we are using lateral images where straight lines are represented as curves and falling edges are below rising edges, the former has a number of pixels that is greater than the latter because of the distortion caused by the fisheye lens. Therefore, every point of the first correspond to a point of the second one after following the gradient orientation. At the end, all the points of the falling edge have a corresponding point of the rising edge and also the opposite is true (in reality, what we obtain is that one point of the rising edge can have more than one corresponding point). If we had done the opposite, we would have obtained that every point of the rising edge would have had a corresponding point on the falling edge, but the opposite would have been false. So, we would have lost parts of the falling edge because for each origin point one and only one point is considered as valid for the pair creation.

Once pairs are found, only the middle point is considered. Figure 4.9 shows the results of this step.

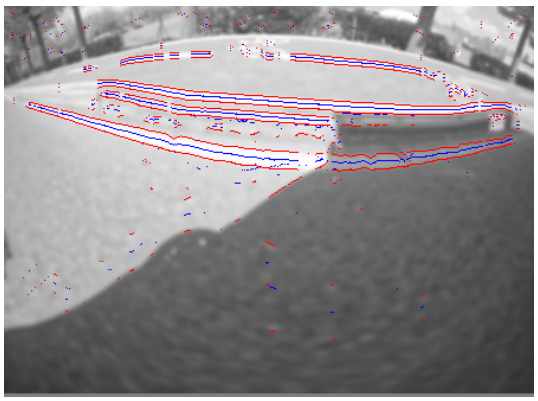


Figure 4.9: Found pairs.

4.2.4 Clustering

The purpose of this step is to group the points obtained with pairs search. In Figure 4.10 the resulting binary image with these points is shown. A typical method like k-means clustering is not suitable for our case because it can create only circular clusters and their number have to be fixed. This is why a custom clustering algorithm is implemented.

Given a reference pixel, the expansion mask used to find candidates to be added to the same cluster has a size of 9 pixels vertically (no line is represented as straight because of the distortion caused by the fisheye lens, so most of them have consecutive pixels whose row numbers are different of more than one) and 10 pixels horizontally (there could be pixels belonging to the same line that are far from each other, for example due to noise). The reference pixel is at the left extreme of the mask and due to this choice the image is examined by columns. A row-major scan could have caused problems like lines divided into two segments: in the case of a straight lane marking, that is represented with the spherical model as a curve that goes up and then goes down, the first point found would be its maximum. Starting from this point and using the expansion mask just indicated leads to the creation of a cluster containing only the right points of the line. The adopted solution is however not the only possible: using a mask with the reference pixel in the middle could have been the same.

Each point found in this way is then considered as reference pixel and the search continues until no new candidates for the current cluster are found.

In order to make next operations more efficient, the list of points of each cluster is populated by employing the insertion sort algorithm. Some of the steps will indeed require points to be sorted by the number of column.

4.2.5 Filtering

Clustering allows to find all the possible lines present in the input image, but sometimes some clusters have nothing in common with this and are only made of noise points that are close together. The following metrics have been used to distinguish valid clusters from useless ones:

- **Number of pixels:** we considered a minimum number of pixels to classify a cluster as significant. This parameter is particularly useful to avoid most part of the small clusters made only of noisy points;
- **Sparsity index:** what we expect is to have points that are as consecutive as possible. In other terms, the column index difference between one point and the following should have a low value, one if the line is continue. Also row index differences should be small: two has been considered the maximum value acceptable to avoid penalties. To calculate the last differences, however, a previous processing is necessary since more than one point per column has become part of the same cluster. In a case like this, indeed, it is necessary to consider only a single point per column to have a unique value of row difference. We decide to take a point whose row index is the mean of their row indexes. The so-called sparsity index is calculated as follows:

$$s = \frac{\sum_{i=1}^{size(C)} \max\left(\left(|C[i]_y - C[i-1]_y| - 1\right) + \left(|C[i]_x - C[i-1]_x| - 2\right), 0\right)}{rows(C) \cdot cols(C)} \tag{4.25}$$

  The calculated value is normalized dividing the sum of row and column differences by the area occupied by each cluster. Without it a line with some points skipped due to noise would have a greater sparsity index than a small group of noisy points. The only clusters kept are those whose sparsity index is lower than a certain threshold.

4.2.6 Polynomial fit

To start adopting the more correct concept of "line" in place of the more general "cluster", a polynomial fit is done. In this step all the points that belong to the same cluster are fitted with a polynomial function. The degree of the polynomial chosen is three, but only if a condition is satisfied: no more than one stationary point (i.e. local maximums, minimums or inflections) have to be present inside the part of the function that we

take to replace the points of the cluster. If the condition is not satisfied, a polynomial function of degree two is used for fit.

Moreover, for each point of the function the distance (as number of rows) between the point obtained with the fit and the original point found until the last step is computed. The sum of these distances is considered as a parameter for accepting or not the proposed fit. However, if the proposed fit is not considered as acceptable, the cluster is invalidated. This is because it is probable that a cluster like this does not contain a true line but only a series of noisy points not sparse enough to be marked as invalid before.

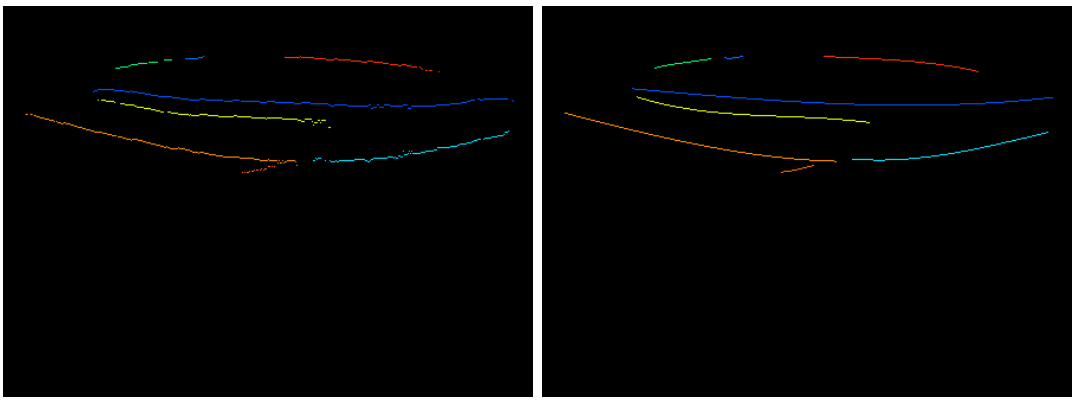


Figure 4.10: Clusters found and result of fit.

4.2.7 Lines linking and validation

Not all the segments found during the last step represent an entire line because some of its points, as we can see in Figure 4.10, can be not close enough to become part of the same cluster.

An easy solution to this problem can be the use in 4.2.4 of an expansion mask with a greater horizontal size. Anyway, this would cause more noise to become part of the clusters. Another problem that this variation could cause is the union of clusters that contain different lines. The value chosen is already the result of a trade-off between the number of clusters made and the noise captured.

The approach adopted, instead, is made of two steps.

Step 1

The first part of the algorithm makes use of two parameters: the position and the orientation of the head and the tail of lines. Taken the tail of a line and the head of another one, these lines are linked if:

- The tail is on the left with respect to the head;

- The difference between the column number of the position of the tail and the one of the head is lower than a threshold (i.e. the lines are close enough horizontally);
- The difference between the row number of the position of the tail and the one of the head is lower than a threshold (i.e. the lines are close enough vertically). This threshold depends on:
  - The difference on the columns number between tail and head: the more tail and head are close, the less is the difference on the rows number allowed, since as they become closer they also have to be more aligned;
  - The orientation of tail and head: the more their slope is, the more is the allowed difference on the number of rows.
- The difference between the orientation of the tail and the head must be lower than a threshold (i.e. the orientations are almost the same or they are opposite but with a similar absolute value);
- The union of them does not involve the intersection of other lines.

After that, resulted lines are checked to keep only longer ones. The threshold is more strict than that used in 4.2.5 because most part of the smaller segments are expected to have been linked just now if they are effectively part of a line. The resulting lines are those of Figure 4.11.

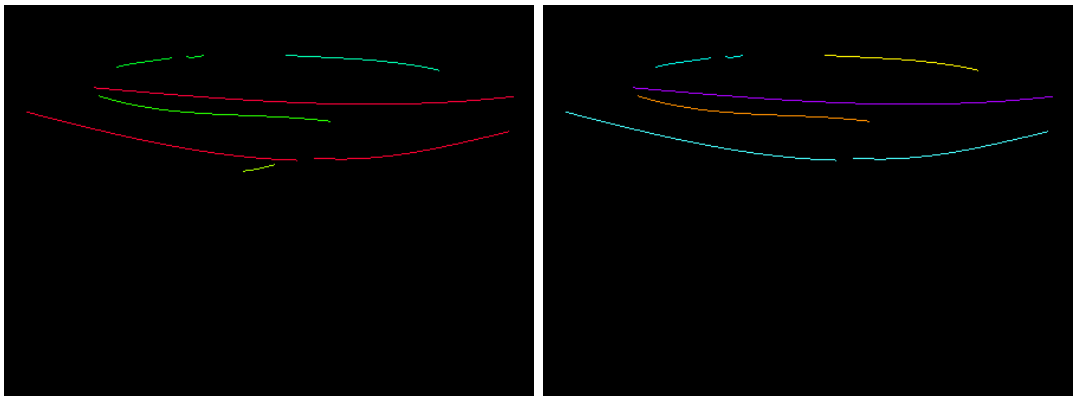


Figure 4.11: Lines obtained after the first step of linking and validation.

Step 2

After that, another more sophisticated linking algorithm is used. The parameters that it considers are:

- The distance between two lines;

- The orientation difference between tail and head of two consecutive lines;

- The error on polynomial fit of these two lines: all the points of the considered lines are used to fit a polynomial function of degree 3. The error is the sum of the distances between each on them and the corresponding point that belongs to the obtained curve.

The linkings that, considering these parameters and some thresholds whose values have been empirically determined, are certainly wrong are excluded a priori. The values of thresholds are more relaxed compared to that used during the first step of lines linking. An overall score has been calculated for each possible linking by using a weighted average of the parameters just listed: the distance has 50% of the total weight, whereas the orientation difference and the error on fit have respectively a weight of 20% and 30%. Figure 4.12 shows an example of matrix where each element represents the value of overall score obtained by the pair (i, j), where i and j are respectively the row index and the column index.

	0	**1**	**2**	**3**	**4**
0		x	x	x	x
1			x	x	x
2				x	x
3					0,21
4					

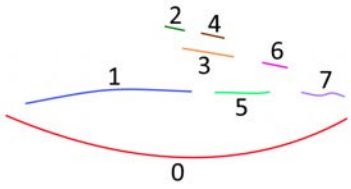
Figure 4.12: Overall score matrix: a lower score corresponds to a higher probability of linking.

The symbol **x** means that the corresponding couple is considered as surely not linkable. Since the lines are sorted by the position of their first point and given the condition that requests the tail of the first line to be before the head of the second line, only the right upper part of the matrix needs to be populated and checked. The algorithm used to make linkings is indicated in Algorithm 2. In Figures 4.13, 4.14, 4.15, 4.16 and 4.17, all the steps of an example of its application is provided. This example makes use of a different image of the dataset to better understand how linkings are done. By using the image of Figure 4.1, indeed, only one linking is possible and the starting matrix is the one in Figure 4.12

After all groupings are done, another filtering is done by deleting all the lines that are shorter than a threshold. This threshold is even greater than that used in step 1. The reason is the same as before.

Algorithm 2: Lines linking - Step 2

```
do
    Initialize and populate score matrix
    linkingDone ← false
    foreach row i of the score matrix do
        if i-th line has not been linked then
            Sort scores in ascending order
            trial ← 0
            while trial < available scores do
                Take the column index j of the element (i, j) with the lower score
                 available (i.e. scores[trial]) // It is the best possible linking for i
                // Check if it is the best linking also for j
                Check the scores in column j and find the minimum
                if the minimum is in the i-th row then
                    (i, j) is the best linking for i and j
                    Replace the j-th line with the linking of i and j
                    Mark the i-th line as already linked
                    linkingDone ← true
                    Exit foreach
                end
                else
                    There exist a better linking for j
                    trial ← trial + 1
                end
            end
        end
    end
while linkingDone = true;
```

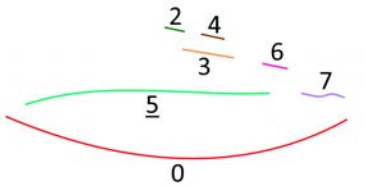


	0	1	2	3	4	5	6	7
0		x	x	x	x	x	x	x
1			x	x	x	0,13	0,2	x
2				0,34	0,01	0,5	x	x
3					x	x	0,06	0,22
4						x	0,1	0,13
5							x	0,12
6								0,32
7								

Figure 4.13: Line 0 has no possible linking as its row in the matrix only contains **x**. Line 1 can be linked with either line 5 or 6, but 5 is chosen because of the lower score. (1, 5) is the best linking for both lines 1 and 5 since the 5-th column does not contain a lower score.

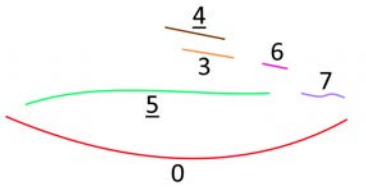
4.2.8 Lines linking with CRF

The second linking step described in Algorithm 2 is not the only method that was considered to perform this task. The use of Conditional Random Fields (CRF) has been



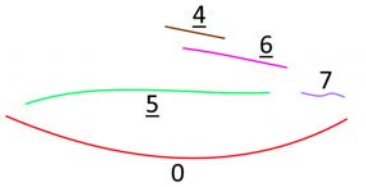
	0	1	2	3	4	5	6	7
0		x	x	x	x	x	x	x
1			x	x	x	x	x	x
2				0,34	0,01	0,5	x	x
3					x	x	0,06	0,22
4						x	0,1	0,13
5							x	0,08
6								0,32
7								

Figure 4.14: The union between lines 1 and 5 has become the new line 5. Line 1 does not exist anymore.
Among the possible linkings for line 2, the one with line 4 is the best.



	0	1	2	3	4	5	6	7
0		x	x	x	x	x	x	x
1			x	x	x	x	x	x
2				x	x	x	x	x
3					x	x	0,06	0,22
4						x	0,13	0,3
5							x	0,08
6								0,32
7								

Figure 4.15: The best choice to link line 3 is line 6. Also for the latter it is the best choice possible.

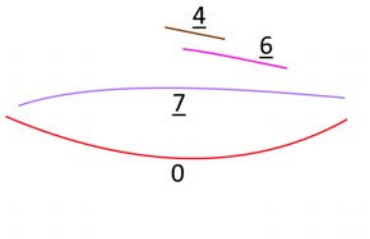


	0	1	2	3	4	5	6	7
0		x	x	x	x	x	x	x
1			x	x	x	x	x	x
2				x	x	x	x	x
3					x	x	x	x
4						x	x	0,3
5							x	0,08
6								0,31
7								

Figure 4.16: Line 5 has as only valid linking possibility line 7.

taken into account.

CRFs are a powerful statistical tool often used for prediction tasks where the context or the state of neighbors is important: sequential prediction problems are an example. They can also be seen as a graphical model with an undirected graph where nodes represent random variables and edges the dependence between them.



	0	1	2	3	4	5	6	7
0		x	x	x	x	x	x	x
1			x	x	x	x	x	x
2				x	x	x	x	x
3					x	x	x	x
4						x	x	x
5							x	x
6								x
7								

Figure 4.17: The original line 0 and the new lines 4, 6 and 7 are the output of the algorithm.

In our case we first need to create the graph and then, being a machine learning method, we need to create a training set with examples of correct linkings and others that have not to be done. We also need to add two constraints: a tail can be connected at most to one head and linked lines must not overlap.

Our implementation is based on [10], but the main difference is the type of images (and, consequently, the shape of the lines) we have to deal with: the authors of the paper apply the method to front view images captured with a pinhole camera. Because of this, the number of features of the road markings that can be used in that case to understand if a linking is possible or not is greater than ours and the values of the features themselves can be thresholded easily. While in a perspective view two segments are part of the same road marking if their orientation are similar, with a fisheye image this is not possible since lines are represented as circular arcs. In our case, indeed, the orientation of a circular arc does not have a meaning.

Despite this, CRFs were implemented anyway in our algorithm and the features that have been considered are the distance between the tail and the head of two consecutive lines and their orientation difference.

However, the greater computational complexity required was not motivated by the results obtained. This is why this method was abandoned in favor of Algorithm 2.

4.2.9 Principal line search

The last step consists in selecting from the lines found so far the main one. As indicated in Chapter 1, we define as principal line the road marking that is the longest and at the same time the closest to the camera. Since the length and the closeness are sometimes not directly related (it is possible that the closest line is not also the longest one or vice versa), a relevance score is calculated for each line to obtain a trade-off between the parameters. The distance is considered as more important: this is why it has a weight of 60% in the calculation. By consequence, the weight of the length is 40%. However, if

the principal line is made of more than one segment (for example if it is a dashed line), the output should contain all these segments. A calculation of the common columns of the first and the second relevant lines has been computed and five cases have been considered:

- **Case 1:** the secondary line starts before the most relevant line and ends before it;
- **Case 2:** the secondary line starts after the most relevant line and ends before it. In this case the secondary line is shorter than the other;
- **Case 3:** the most relevant line starts after the secondary line and ends before it. In this case the most relevant line is shorter than the other, but it has been chosen as main line because it is closer to the camera than the secondary one;
- **Case 4:** the secondary line starts after the most relevant line and ends after it;
- **Case 5:** the secondary line starts and ends before or after the principal one.

When we are in the first or fourth case, the secondary line is kept as valid and it becomes part of the output only if common columns are less than half the length of the longer line between the candidates. In the fifth case the two lines have no common columns (we could have a dashed line), so both are kept. In the second and third cases, instead, the secondary line is ignored.

Figure 4.18 shows the result of all the computation of this algorithm.

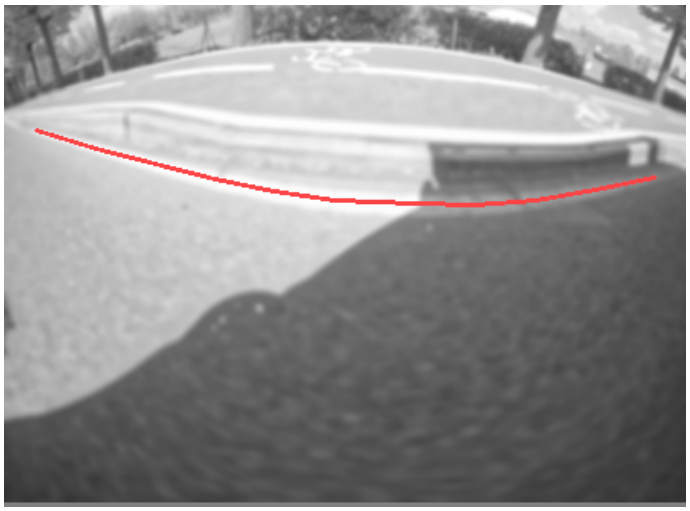


Figure 4.18: The final output of the traditional CV-based algorithm.

4.2.10 Final algorithm flow

To sum up, the actual steps the final algorithm is composed of are:

1. **Preprocessing**, consisting of a conversion from YUV format to RGB and then to grayscale; a counterclockwise rotation; a scaling by a factor of 2; a Progressive Gaussian Blur application to eliminate most of the noise caused by the road surface; a vertical crop, to avoid searching lines where we are sure they are not; and finally an horizontal crop, to keep different lines distinguishable in the edges;
2. **Search of DLD transitions by means of DL and LD transitions**, by using the gradient magnitude and orientation calculated with only a vertical and an horizontal kernel. DLD transitions are not directly sought because of the big kernels requested for gradient calculation;
3. **DLD pairs search**, exploiting the mapping between camera and body frame to know how far each pair of pixels should be;
4. **Clustering** of the points found so far;
5. **Filtering** of clusters, eliminating those that contain only noise;
6. **Polynomial fit** of the points of each cluster;
7. **Lines linking and validation - first step**;
8. **Lines linking and validation - second step**, used in place of CRFs for results and computational complexity reasons;
9. **Search of the principal line**.

4.3 Analysis of results

The algorithm has been tested on a dataset with 1010 images captured in different weather (sunny, rainy and clear sky) and illumination (day, night and interiors) conditions. The same test set has also been used for the second approach of Chapter 5. More details about data can be found in Table 5.5.

The metrics used are *precision*, *recall*, *accuracy* and an overall *F1 score*. They are defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{4.26}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.27}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.28}$$

$$F1_{score} = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4.29}$$

To calculate them it is necessary to count the number of positive and negative detections.

First of all, we calculate for each image how many lines the ground truth contains (in other terms, the number of annotations of the current frame). Since annotations only contain the principal line, an indication of more than one segment means the main line is a dashed line. We then calculate how many lines have been found among those annotated. Firstly, for each line we calculate the number of points that are not far more than 10 pixels from the correct position. A line is considered as found if the predicted one has a the number of points just calculated that is lower than one third the size of the actual line. Even if one third can seem a low number of points, we have to remember that for technical purposes only (as indicated in Paragraph 4.2.1) the horizontal crop of the image has caused the loss of some points of the lines at the edges of the image. Annotations are instead not affected by it. The difference between these numbers is the number of segments not found.

After that, the number of predicted lines and how many among them are true are calculated. A line is considered as true using the same condition just described. The difference between them indicates how many predicted lines are not part of those annotated.

Every image is classified as:

- **True positive:** there are one or more annotated lines and at least one of them has been correctly found. This means that it is not necessary to find all the dashes of a dashed line to indicate it as detected;
- **True negative:** there are no annotations and no lines have been found;
- **False positive:** three cases are possible: there are no annotations but at least one line has been found; the correct line has not been detected (false negative) but something else (a non-principal line or noise) has been recognized as the main line; the correct line has been detected but something else has been indicated as part of the principal line. However, even if more than a false positive is present, only one is counted. This is done for fairness, since a correctly detected dashed line is counted as only one true positive no matter how many dashes are found;

- **False negative:** there are one or more annotated lines but no segments have been found or the detected ones are incorrect.

Note that an image can have multiple classifications due to false positives in combination with true positives and false negatives. In Figures 4.19, 4.20, 4.21 and 4.22 some examples are provided for each classification.

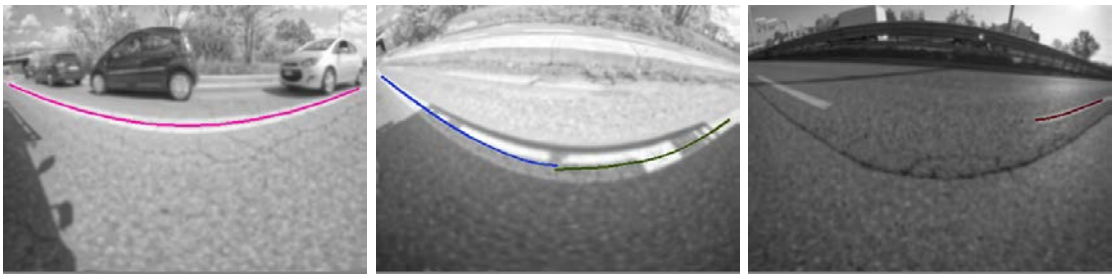


Figure 4.19: True positives.

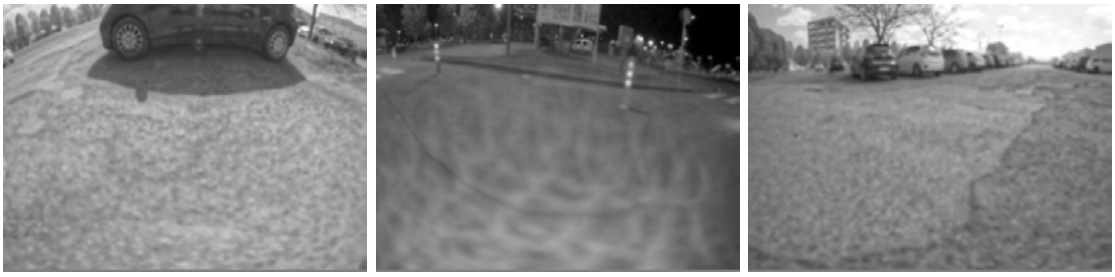
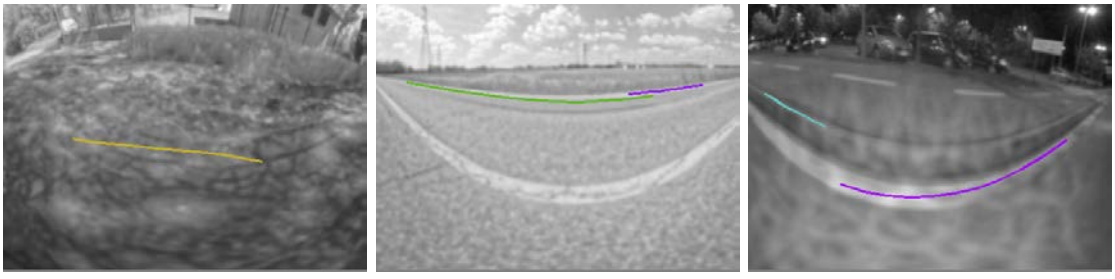


Figure 4.20: True negatives.



(a) Only false positive. (b) With false negative. (c) With true positive.

Figure 4.21: False positives.

The results of detection are divided by the type of images in order to make better considerations about them. The groups considered are four: daylight and sunny, daylight and rainy, night, and interiors pictures (for instance, underground parkings). As we can see from Figure 4.26 and Tables 4.1 and 4.2, the algorithm achieves an accuracy of 43,79% and an F1 Score of 56,57% on the entire test set. However, it works better with

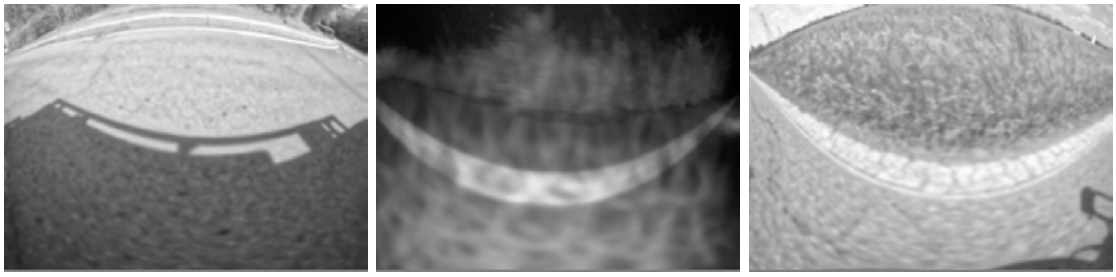


Figure 4.22: False negatives.

images in sunny and daylight conditions, reaching 47,93% of accuracy and an F1 Score of 63,88% in this case. This can be explained by the fact that this is the ideal condition for detection and the tuning of parameters have been done by using mostly images of this kind.

Overall results are worse in the case of rain, since the drops of water on the lens (as shown in Figure 4.23) make the lines not visible and the gradient calculated is highly affected by this.

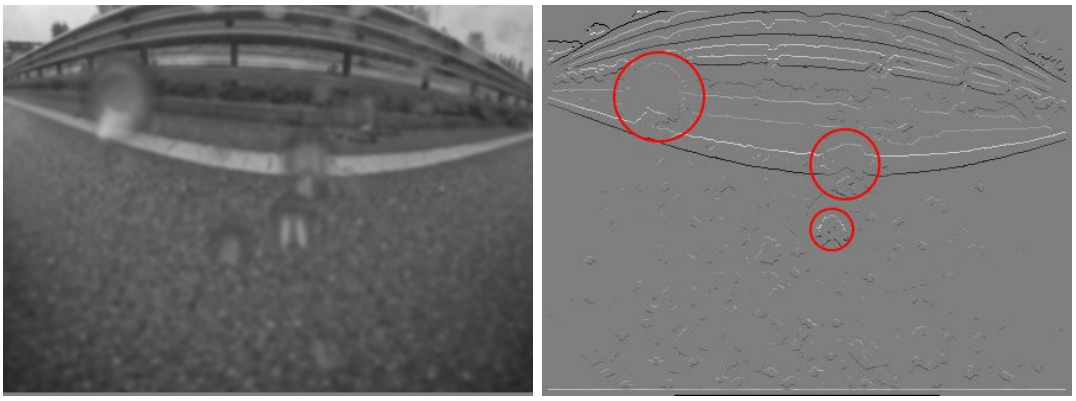


Figure 4.23: Water drops on the lens make the line undetectable. The right image shows the magnitude of the gradient calculated.

Results on night images are the closer to daylight ones: with an F1 Score of 56,66%, the value is only 7,22% lower. The same happens when considering the accuracy, whose value is 42,39%. It is important to point out that no images of this type have been used to tune the parameters. The high number of false positives is due to the presence in most of the images of strong noise caused by the illuminator installed in the vehicle used to capture the images, as we can see in Figure 4.24.

When the algorithm is tested on interiors images, instead, we achieve a F1 Score of only 3,13%. The cause is the yellow color of the annotated line, that is effectively the principal line. The algorithm, as we could expect, indicates as main line a white one since the conversion from RGB to grayscale makes colored lines difficultly detectable. In Figure 4.25 an example is provided. It is important to observe that this part of the

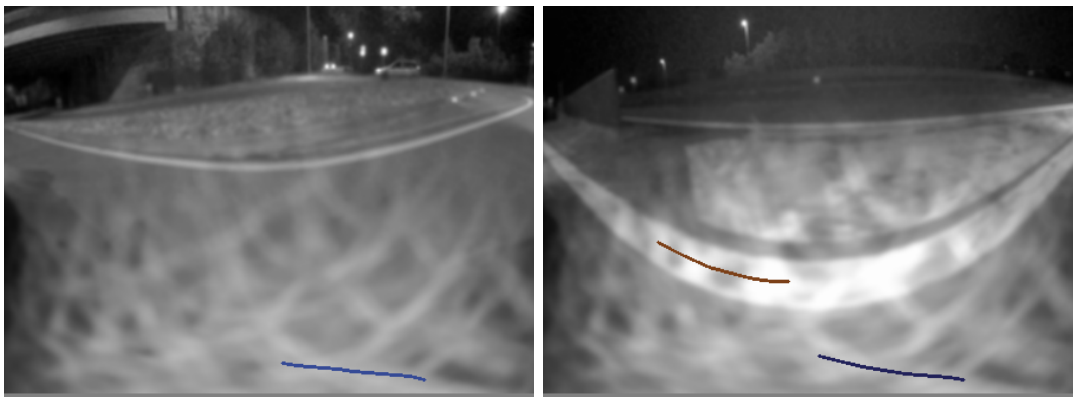


Figure 4.24: Noise caused by the illuminator in night images.

dataset is the most difficult the algorithm has to deal with and could require specific parameters to handle the different illumination condition.

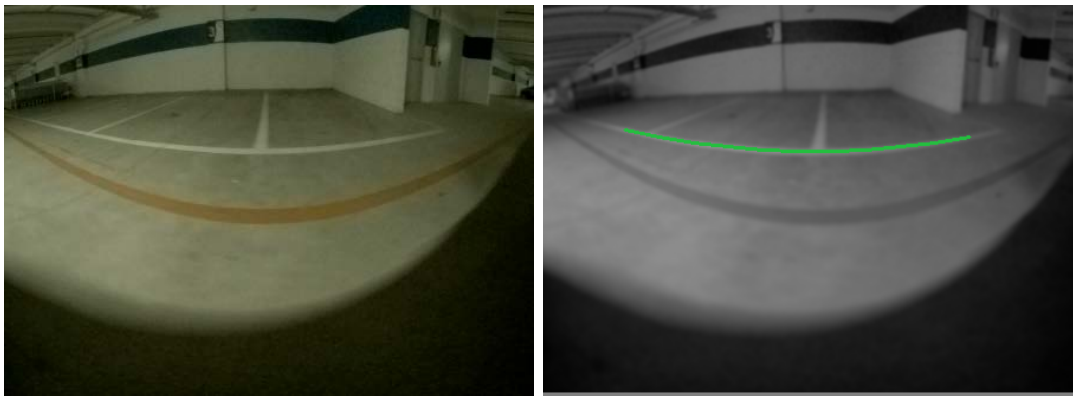


Figure 4.25: Yellow lines in interior images are not detected with traditional CV-based algorithms that use RGB to grayscale conversion.

Illumination	**Weather**	**Images**	**TP**	**TN**	**FP**			**FN**
					FP only	**w. FN**	**w. TP**	
Daylight	Sunny	475	267	11	11	96	9	186
	Rainy	165	35	43	3	15	0	84
Night	Clear	310	151	19	15	77	14	125
Interiors	Interiors	60	1	16	1	19	0	42
		1010	454	89	30	207	23	437

Table 4.1: Results of the algorithm on the test set. TP are true positives, TN true negatives, FP false positives (with the variations "only false positive", "false positive and a false negative", and "false positive and a true positive"), FN false negatives.

In general, however, the dataset created contains a lot of road markings that are very difficult to recognize: situations like faded lines are very common, since a non-negligible part of the images are captured in rural roads. This could explain the low

Illumination	Weather	Images	Precision	Recall	F1 Score	Accuracy
Daylight	Sunny	475	69,71%	58,94%	63,88%	47,93%
	Rainy	165	66,04%	29,41%	40,70%	43,33%
Night	Clear	310	58,75%	54,71%	56,66%	42,39%
Interiors	Interiors	60	4,76%	2,33%	3,13%	21,52%
		1010	63,59%	50,95%	56,57%	43,79%

Table 4.2: Metrics obtained testing the algorithm on the test set.

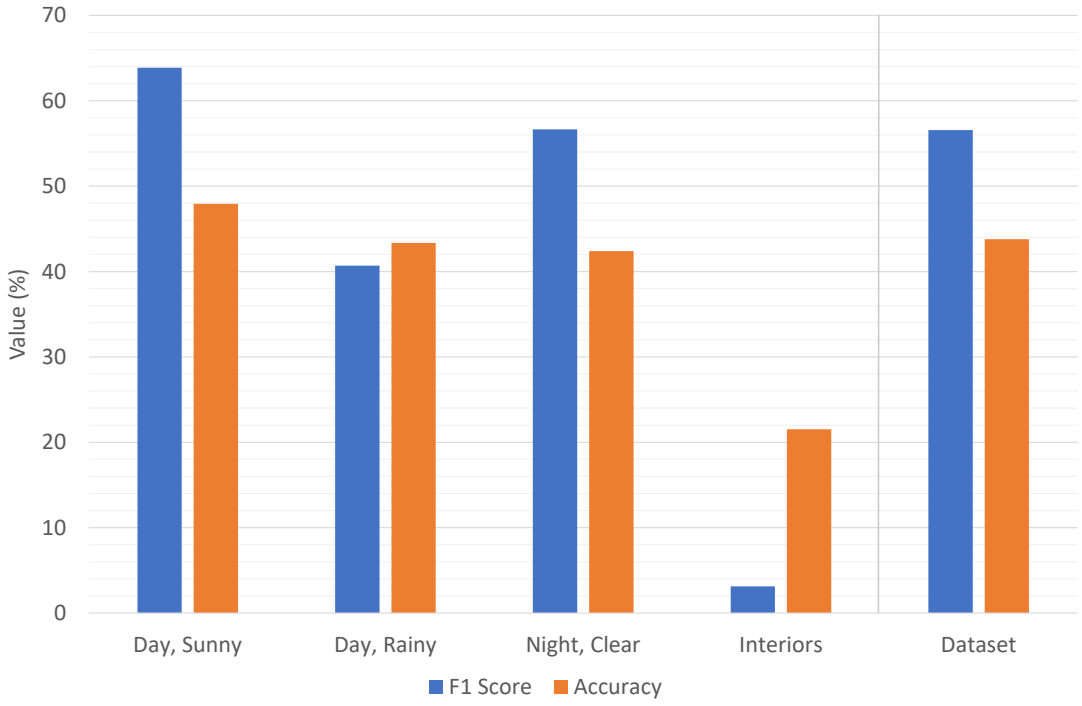


Figure 4.26: Results of the traditional CV-based method, divided by scenario and global.

values of the results achieved.

4.4 Future work

Our work can be further developed to make the detector more robust in difficult situations. For instance, performances can be improved in the case we are not dealing with a straight line but with a curve. Moreover, new more complex conditions can be added to create the linkings between found segments. We provide a list of enhancements that have been thought but that are not yet implemented:

Pixels-to-meters conversion of measures: lines length and distances between them, besides the distance from the camera, are considered with their measure in pixels.

We can employ the transform between camera and body coordinates used in 4.2.3 to convert each distance between points from pixels to meters. In this way the thresholds are not fixed for every zone of the image (remember that two pixels in the middle of the picture correspond to a greater measure in meters than two pixels close to the camera) but they vary following the distortion caused by the fisheye lens;

Inverse Perspective Transform: linkings between segments in 4.2.7 can be done by using different conditions. We can map these segments in Bird's-Eye View representation. In this way lines that are straight in reality become straight (and no longer arcs of circumference) and so we can consider the correspondence of the orientation of the whole segments instead of only the tail and the head of each of them. These more robust features could be the key for retrying the use of CRFs, being the obtained situation more similar to that presented in the paper [10] that suggests their usage. Note that we are not proposing a mapping of the entire image that would be too heavy for a typical automotive built-in platform, but only a transformation of the points that belong to the lines found.

5

Second approach: Convolutional Neural Networks

In this chapter, an alternative approach to the lane detection problem is presented. As more recent works described in 2.2, the detection is now performed by using a Convolutional Neural Network to understand if, and in the positive case how much, this method outperforms the one presented in the last chapter.

5.1 Pros and cons

Convolutional Neural Networks are essentially Neural Networks architectures that are best suitable for performing image-based tasks. What characterize them is the type of layers they are composed of. The most common and specific layers we can find in a CNN are convolutional and pooling layers. The aim of the former is to learn spatial relations between pixels, while the main role of the latter is to reduce the amount of information for next layers. Indeed, since the input is an image that have two dimensions (and it can be made of multiple channels, three in the case of an RGB picture), the amount of input data easily exceeds the quantity of information that other types of neural networks usually have to deal with.

Neural network-based methods for image processing are more and more used because of their significant advantages. The main reason is that they do not rely on application-specific or hand-crafted features, that are instead learnt automatically during training. In this way the model found could potentially handle every possible situation: it is only needed to have its representation in the dataset. At the same time there is no need for a priori segmentation of the input image.

The biggest drawback, however, remains the need for lots of data to train these neural networks. Images belonging to these large datasets should be as different as possible among them and, this is usually the main problem encountered, must be anno-

tated. Annotation, indeed, is one of the most time-consuming tasks related to the use of CNNs. For semantic segmentation, for example, annotating one single image could require up to one hour, depending on the number of classes needed. And the number of images required to have significant results could be greater than some thousands. Moreover, deep learning also requires a lot of computational power for training tasks so that specific architectures for scientific calculations are needed to make the duration of these operations at least acceptable. Requiring high parallelization, using an high-end GPU (or more than one GPU), that is more suitable for this kind of tasks, is better than having a high-end CPU because of the higher number of processing units a GPU is made of. Requiring very powerful and by consequence very expensive hardware, this type of computation is often reserved only to big companies and research institutes. This does not happen for traditional computer vision methods since their computational complexity is usually so low as to be accessible to nearly anyone.

5.2 Problem definition

Our work is based on those proposed in [17]. Before presenting the network architecture used, it is important to point out what we want our system to do, and so what are the inputs and the outputs.

We give as input an image in YUV format with 1024 rows and 992 columns and we convert it in RGB color space. An example of image is the one at left in Figure 4.1. The further processing is the same discussed in Paragraph 4.2.1. However, this time only the scaling is needed and neither rotation nor grayscale conversion are performed. This means that, using the notation (rows x columns x channels), we have an input image of size (512x496x3).

We want to obtain the principal line (as defined in Chapter 1 and in Paragraph 4.2.9) composed by only one pixel per row. More specifically, the output is a vector of 512 (rows number) elements. Each of them contains the column number of the pixel in that row where the principal line to be found is. If in that row no pixel of that line is present, the value assigned is 496 (that is the first positive number that has no meaning as column number since their values are in the interval between 0 and 495). In other terms, we are performing a classification task with a number of classes equal to 497.

5.3 CNN Architecture

The network used is based on ERFNet [27]. As indicated in Paragraph 2.2.2, it is an encoder-decoder-based CNN used to perform semantic segmentation. Because of this,

its output has the same size of the input, unlike our implementation requirements. By consequence, some variations are needed to make the output be a vector instead of a matrix:

- At the end of the encoder the tensor obtained has size (128x64x62). We permute the rows and the columns obtaining a tensor of size (128x62x64) and then we reshape it to have a ((128*62)x64) = (7936x64) tensor. This operation is done to have an easier control of the number of feature maps we want as output, since 497 is not reachable with upsampling because it is an odd number;
- A single-dimension convolutional layer has been added as 17^{th} layer to return to 128 features map in order to avoid channels variations in the decoder;
- The number of output feature maps of the last layer of the decoder has been changed to indicate the number of classes of our classification algorithm.

The CNN obtained after these variations has 3 513 069 parameters, more than the 2 159 374 parameters of the original version of ERFNet. This is caused particularly by the convolutional layer at the end of the encoder that adds 1 015 936 parameters. However, also the higher and constant number of feature maps of the layers of the decoder are responsible for this increase on the number of parameters.

The architecture of the CNN used is explained in Table 5.1.
At the end we obtain a (497x512x1) tensor. Each of the 512 vectors is made of 497 elements whose values are in the range $(-\infty, +\infty)$. They are processed by applying the Softmax function (defined in Equation 5.1) so that their values are in the range (0, 1) and they add up to 1.

$$S(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{496} e^{x_j}} \quad i \in [0, 496] \tag{5.1}$$

In this way, we have a probability distribution for each of the 497 classes. The predicted classes are found as follows:

$$output[k] = \arg\max_i S(x_i) \quad k \in [0, 511], \;\; i \in [0, 496] \tag{5.2}$$

The loss function we use is the sum of two terms:

- **Cross-Entropy Loss:** this loss function measures the performance of a classification model whose output is a probability value from 0 to 1. Given for each row i of the image the output of the Softmax function S_i and a vector L_i that contains 0-valued elements except in the position that corresponds to the true class (where its value is 1), we define this loss as follows:

	Layer	Type	Out f.m.	Out size	Parameters
		Input tensor	3	512×496	
ENCODER	1	Downsampler	16	256×248	396
	2	Downsampler	64	128×124	7088
	3-7	5x Non-bottleneck 1D	64	128×124	248 320
	8	Downsampler	128	62×64	37184
	9	Non-bottleneck 1D (dilated 2)	128	64×62	197 632
	10	Non-bottleneck 1D (dilated 4)	128	64×62	197 632
	11	Non-bottleneck 1D (dilated 8)	128	64×62	197 632
	12	Non-bottleneck 1D (dilated 16)	128	64×62	197 632
	13	Non-bottleneck 1D (dilated 2)	128	64×62	197 632
	14	Non-bottleneck 1D (dilated 4)	128	64×62	197 632
	15	Non-bottleneck 1D (dilated 8)	128	64×62	197 632
	16	Non-bottleneck 1D (dilated 16)	128	64×62	197 632
		Permute and reshape	7936	64×1	
	17	Convolution 1D	128	64×1	1 015 936
DECODER	18	Upsampler	128	128×1	49 536
	19-20	2x Non-bottleneck	128	128×1	198 144
	21	Upsampler	128	256×1	49 536
	22-23	2x Non-bottleneck	128	256×1	198 144
	24	Upsampler	497	512×1	127 729

Table 5.1: Layers of the ERFNet-based Convolutional Neural Network used. *f.m. = feature maps.*

$$\ell(L,S) = \sum_{i=0}^{rows-1} \left(- \sum_{j=0}^{classes} w_j \cdot L_{ij} \cdot \log(S_{ij}) \right) \tag{5.3}$$

w_j contains for each class the value of its weight. In our case, all the classes have the same weight that equals to one;

- **Continuity measure:** this term takes into account the difference between the class indexes of consecutive rows. The aim is to have a sum of differences with a low value that means as few hops as possible, in order to obtain continue lines. The penalty for each hop is considered only if it does not appear also in the annotations.

A different CNN architecture was initially used as an alternative. It was based on DeepLanes [17] and, although the main architecture was preserved, some parameters were changed to handle our data: apart from the size of the input and the output, we modified the values of the parameters of the convolutional layers. Since these changes were not trivial, finding the best values would have required lots of attempts. This is the reason why we then decided to use instead a neural network like ERFNet whose architecture we are sure to work correctly.

The original implementation of DeepLanes has also another disadvantage: its output is the position of the point of the line that is closer to the camera. To obtain all the points of the line we have to divide the image in a number of slices equal to the number of rows of the image and then predict the position of the point for each slice: this is not an efficient way.

5.4 Dataset

The dataset used is made of 3705 color images. All of them are taken in Parma (Italy) in different weather (sunny and rainy) and illumination (day, night, interiors) conditions. For the capture of night images an illuminator has been used. As we can see from Table 5.2, most of the images (2005, 54,12%) represent a daily and sunny condition. An important information given is the number of images with at least one line present (indicated in the tables as *Annotated*). Our system, indeed, has to learn what a principal line is but also it has to recognize where there is not a line. Giving examples with no detection is important to reduce the number of false positives. However, it is also essential to provide a greater number of images with detections, otherwise this could affect the number of false negatives.

Images are split into three groups: training set (Table 5.3), validation set (Table 5.4) and test set (Table 5.5). We worked to make the distributions of the first two sets similar. Test set is instead a separate case since we should not know its distribution, otherwise we could have added to training set more images of a specific type in order to obtain better results when trying the computed model on it: this would have been a cheat. Because of this, the distribution chosen for the test set is completely uncorrelated to the ones of the other two sets: 60% day - 40% night has been considered as a fair distribution. All the images indicated in these tables have been used for quantitative results whereas other 180 images taken in Taipei (Taiwan) have been used for qualitative tests only. Figure 5.1 shows the dataset distribution graphically.

Note that, since no public datasets with fisheye images representing road situations exist, its creation has required the capture and the annotation of all the images.

Data augmentation was applied randomly to the dataset, including transformations like vertical flip (remember that the images are rotated), RGB color channels shift and gamma curve application. To avoid night images to become too dark and daylight ones to get overexposed, the value of gamma was chosen randomly from the interval [0.7, 1.3].

COMPLETE DATASET					
Illumination	**Weather**	**Images**	**Images (%)**	**Annotated**	**Annotated (%)**
Daylight	Sunny	2005	54,12%	1828	91,17%
	Rainy	640	17,27%	497	77,66%
Night	Clear	900	24,29%	638	70,89%
Interiors	Interiors	160	4,32%	130	81,25%
		3705	100%	3093	83,48%

Table 5.2: Dataset images distribution.

TRAINING SET					
Illumination	**Weather**	**Images**	**Images (%)**	**Annotated**	**Annotated (%)**
Daylight	Sunny	1230	64,06%	1106	89,92%
	Rainy	190	9,90%	138	72,63%
Night	Clear	460	23,96%	273	59,35%
Interiors	Interiors	40	2,08%	33	82,50%
		1920	100%	1550	80,73%

Table 5.3: Training set images distribution.

VALIDATION SET					
Illumination	**Weather**	**Images**	**Images (%)**	**Annotated**	**Annotated (%)**
Daylight	Sunny	300	38,71%	268	89,33%
	Rainy	285	36,77%	240	84,21%
Night	Clear	130	16,77%	88	67,69%
Interiors	Interiors	60	7,74%	54	90%
		775	100%	650	83,87%

Table 5.4: Validation set images distribution.

TEST SET					
Illumination	**Weather**	**Images**	**Images (%)**	**Annotated**	**Annotated (%)**
Daylight	Sunny	475	47,03%	454	95,58%
	Rainy	165	16,34%	119	72,12%
Night	Clear	310	30,69%	277	89,35%
Interiors	Interiors	60	5,94%	43	71,67%
		1010	100%	893	88,42%

Table 5.5: Test set images distribution.

5.5 Train and validation setup

All experiments are conducted using the PyTorch 1.1 framework with CUDA 9 and CuDNN backends. Our model is trained using two NVIDIA Titan Xp (Pascal archi-

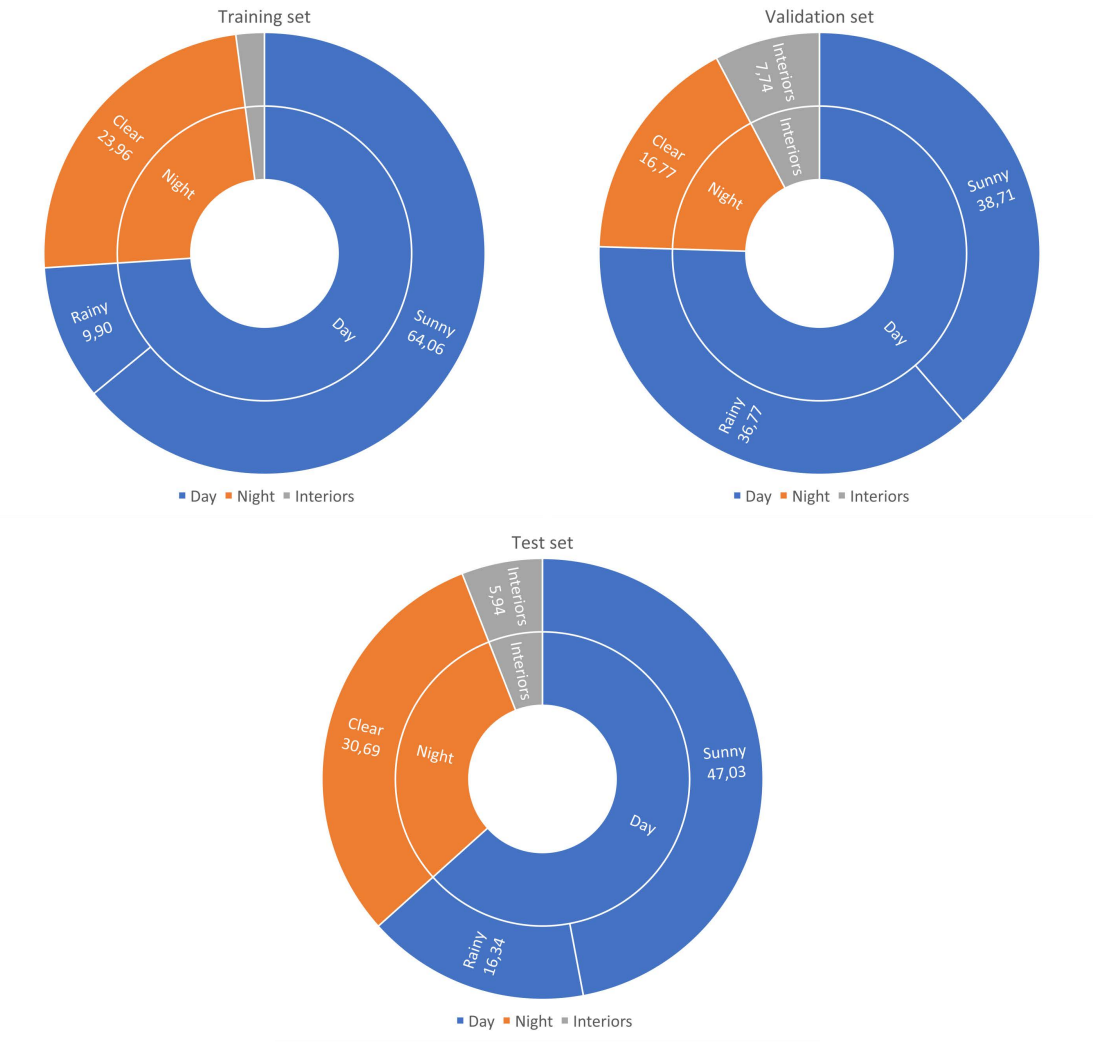


Figure 5.1: Graphs representing the distribution of training, validation and test sets.

tecture) with 12 GB of GDDR5 VRAM, running in parallel. The Adam optimization [30] of stochastic gradient descent is used. Training is performed using as parameters a batch size of 6, a momentum of 0.9, a weight decay of $1e^{-4}$, and with a starting learning rate of $5e^{-4}$. The latter is divided by a factor of 2 every time the training error stops decreasing for a while, in order to accelerate convergence. The number of epochs used for training is 250.

As we can seen from Figure 5.2, the training loss decreases rapidly at the beginning (it is obvious since in that moment the neural net has not learnt enough to generalize well on data) and continues decreasing over the epochs. This decrease becomes less relevant after about 130 epochs. To decide which is the best model, however, we consider the validation loss. Its value, after an initial increase, decreases rapidly and reaches its minimum value after 86 epochs (1,523). After that, its value, even if not monotonically,

tends to increase.

The trend of validation loss with respect to training loss indicates that the model is overfitting. This is caused by the images that, although they are a good starting point since they represent different situations, are sometimes very similar because they are limited to the roads of an only city.

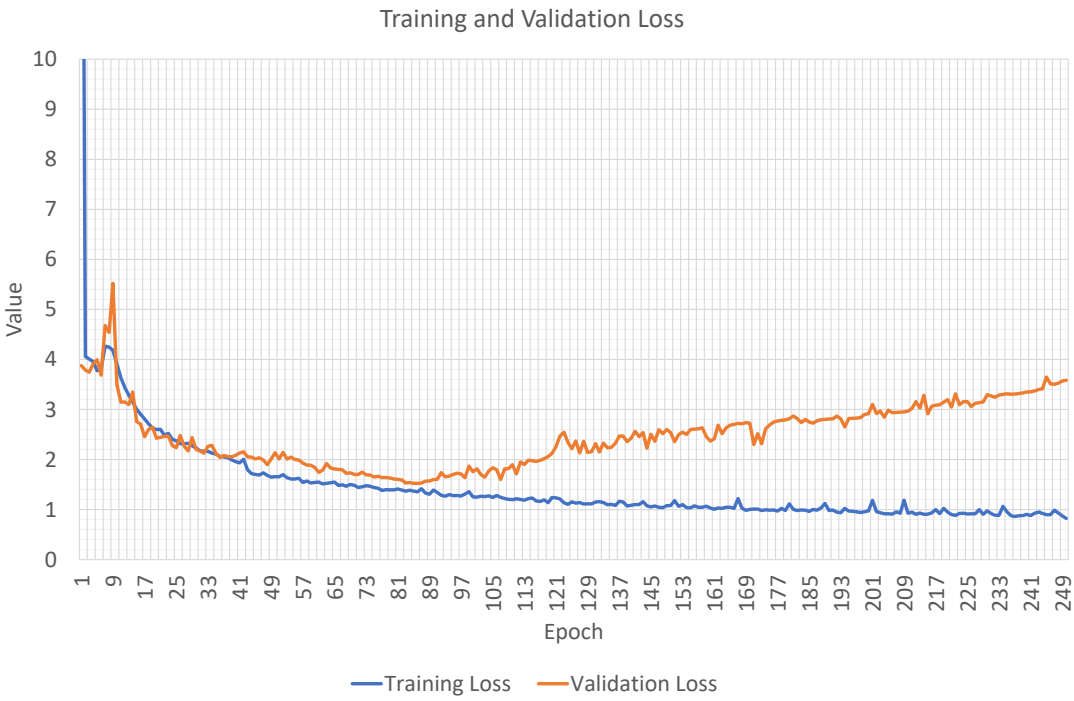


Figure 5.2: Training and validation losses.

5.6 Analysis of results

The same metrics of Chapter 4.3 (precision, recall, accuracy and F1 score) have been used to express the results of the CNN-based approach. Also what we consider as true or false detection is the same. However, since the prediction of line/not-line is done row by row and it is possible to have isolated pixels that would be counted as a small line, we decided to consider as line only series of 10 consecutive predictions different from 496 (no line). All these considerations are done in order to make the results of the two approaches directly comparable.

Tables 5.6 and 5.7 show the results achieved by predictions made on test set. The overall F1 score has a value of 85,45% whereas the obtained accuracy is 76,92%. This approach works better in a night scenario, reaching an F1 score of 92,9%, even if in

daylight illumination conditions the result is close as the score obtained is 86,94% (calculated using both the values belonging to sunny and rainy weather, not indicated in the tables). Interiors images are those that have the lowest correct predictions rate: the cause is certainly the very low number of images of this type (40 out of 1920) that are present in the training set, leading to a poor generalization on this type of data.

Illumination	**Weather**	**Images**	**TP**	**TN**	**FP**			**FN**
					FP only	**w. FN**	**w. TP**	
Daylight	Sunny	475	386	20	12	18	20	67
	Rainy	165	110	33	13	4	6	9
Night	Clear	310	255	33	1	2	15	21
Interiors	Interiors	60	1	15	2	24	0	42
		1010	752	101	28	48	41	139

Table 5.6: Results of the algorithm on the test set. TP are true positives, TN true negatives, FP false positives (with the variations "only false positive", "false positive and a false negative", and "false positive and a true positive"), FN false negatives.

Illumination	**Weather**	**Images**	**Precision**	**Recall**	**F1 Score**	**Accuracy**
Daylight	Sunny	475	88,53%	85,21%	86,84%	77,63%
	Rainy	165	82,71%	92,44%	87,3%	81,71%
Night	Clear	310	93,41%	92,39%	92,9%	88,07%
Interiors	Interiors	60	3,7%	2,33%	2,86%	19,05%
		1010	86,54%	84,4%	85,45%	76,92%

Table 5.7: Metrics obtained testing the CNN on the test set.

Examples of true positives can be seen in Figure 5.4. The images shown belong to different scenarios, that are urban and highway in daylight and night conditions. We can note that in these cases the neural network is able to detect the principal line also with multiple dashes and when the line is not straight.

The approach presents some problems when the lines to be detected are too far or too close to the camera, as shown in Figure 5.5. These cases are classified as false negatives, despite a small part of the line (that is, however, made of less than one third of the points of the correct line, see Paragraph 4.3) is correctly detected. This could be explained by the low number of training images where the principal line is not close to the center of the picture. The CNN, indeed, has probably learnt that the line should be in that area of the image, leading to (as happened for interiors) poor generalization.

The achieved results with images captured in rainy conditions (F1 score of 87,3% and accuracy of 81,71%, that is second only to night results) are the consequence of the robustness to occlusions, like the water drops on the lens, of this method, opposed to the highly affected results obtained with the other approach. Figure 5.6 shows the detections had in this condition.

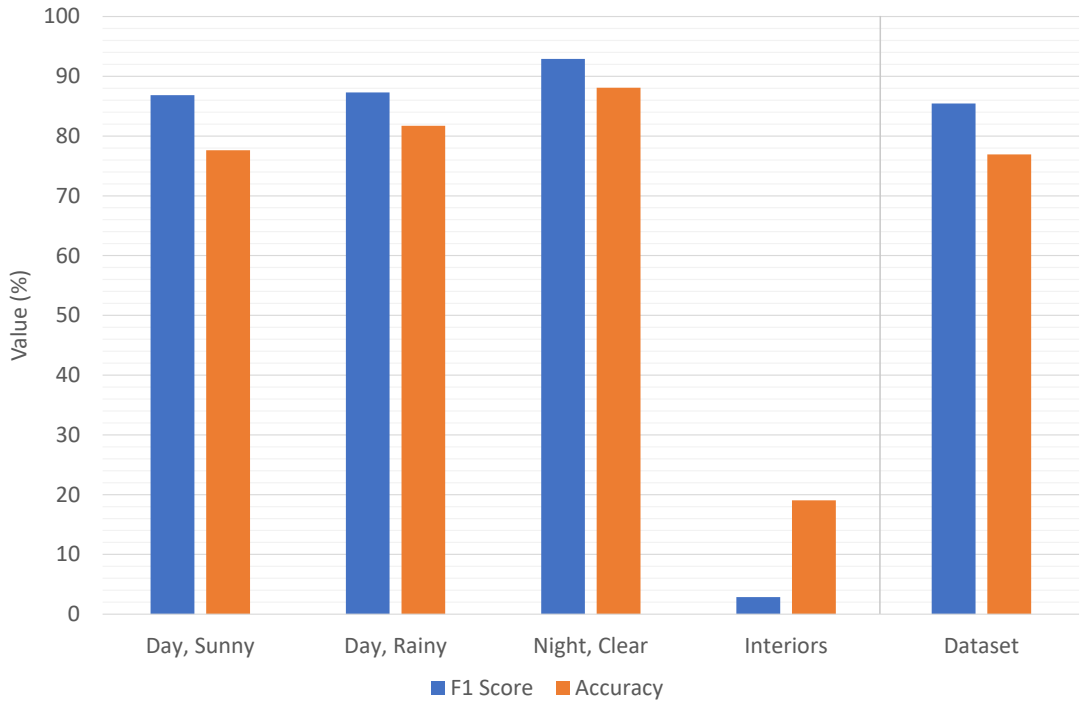


Figure 5.3: Results of the CNN-based method, divided by scenario and global.

Nevertheless, this method is not immune to false positives (see Figure 5.7), even if pedestrian crossings are usually correctly detected as non-lines (Figure 5.8). This type or error was instead common in the previous approach.

Interior images have the same problem of the traditional CV-based approach. Indeed, white lines are predicted as principal lines and yellow ones, that in these cases would be the correct ones, are never detected. While in the previous approach the reason was the conversion from RGB to grayscale of the input image, this time the misdetection is motivated by the fact that the number of yellow lines present in the training set is too low to make the network learn that a line can also be of a color different from white. Note also that the yellow of these lines, due to the low illumination condition, is dark. In case of lighter conditions, as visible in Taiwan images, the detection of yellow lines works better. Figure 5.9 shows the results just mentioned. The only true positive in interiors indicated in Table 5.6 is not a yellow line detected, but a white line that was annotated because the yellow one was not sufficiently visible.

To conclude, what we can do to improve the results just presented is to populate the dataset with more images that represent more situations. The categories must be correctly balanced, to avoid, for example, all the problems we encountered in interiors images. Pictures must contain not only lines in the middle but also more examples of

Figure 5.4: Correct detections obtained with the CNN in different scenarios.

far and very close lines. They could be obtained also by simply making the vehicle used for captures move inside its lane or by overtaking.

5.7 Future work

Division into bins: the current approach solves a classification problem with a lot of classes, causing also the number of parameters to be high. It is possible to divide the problem into multiple steps, avoiding the direct prediction, for each row, of the index of the column that contains the line (if any). The columns are divided in bins of equal size and the classification problem has the aim of indicating in which of these bins the line is. A supplementary bin is used to indicate that the line is

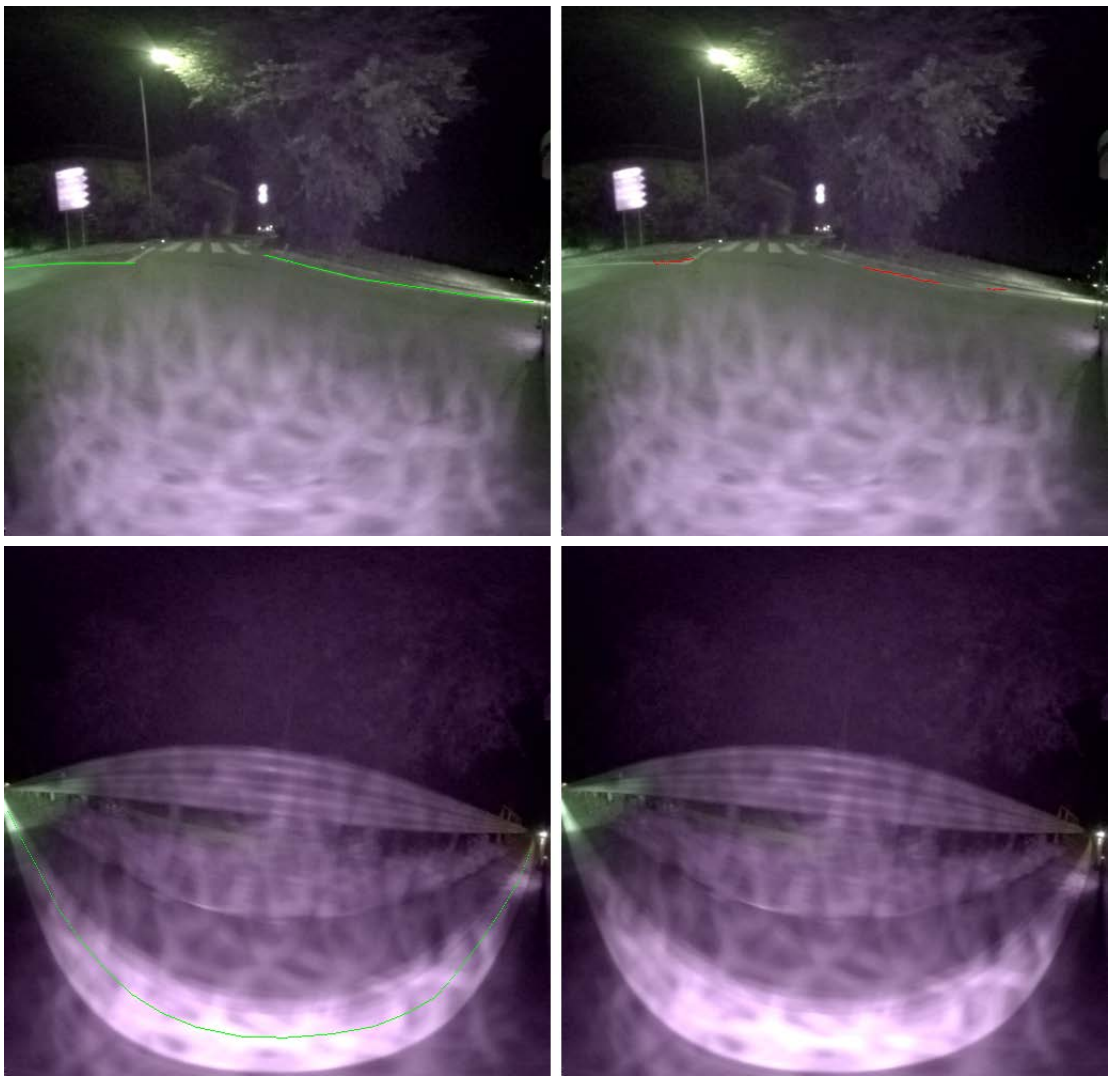


Figure 5.5: False negatives obtained with the CNN. At left there is the annotation, at right the detection.

present in none of the others. As in the current method, the prediction is done for each row. After that, only if the classification result is correct, a regression problem is solved to obtain the actual column index that contains the line as an offset from the center of the bin;

Use of CNN for low-level lines search: better results can be obtained by merging the two approaches presented. What makes the traditional Computer Vision-based approach less robust than the second one is the dependence from the initial pre-processing of the image that starts with the gradients calculation. The result is highly affected by noise and indeed several steps have the main goal of eliminating it as much as possible. However, sometimes this causes the loss of correct lines

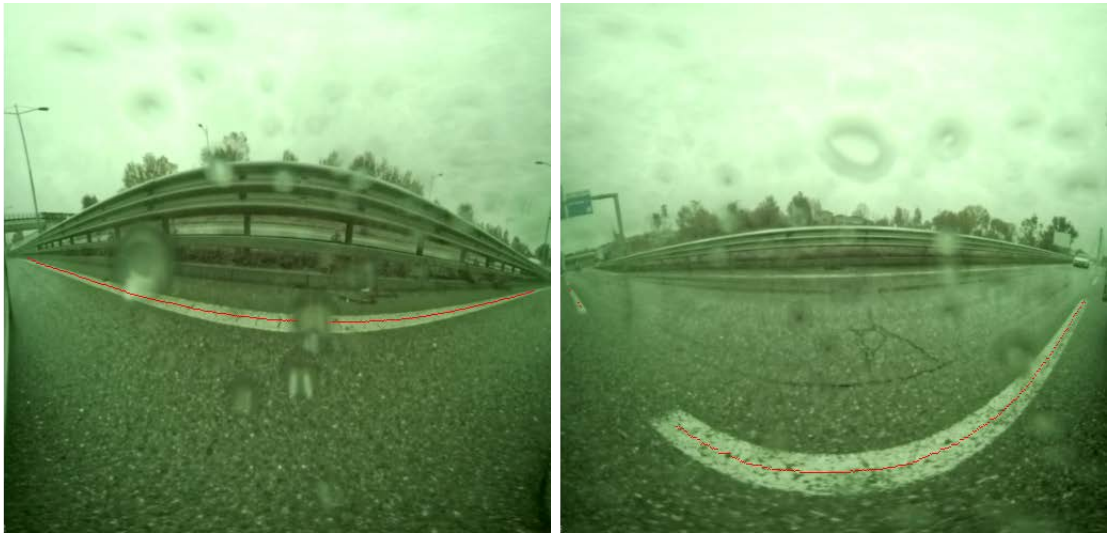


Figure 5.6: Water drops do not affect considerably the detection.

Figure 5.7: False positives obtained with the CNN.

that are incorrectly classified as noise or, more often, not all the noise is actually removed causing the presence of false positives. It is possible to use the output of the CNN for low-level line search, in place of the blur, the crop, and the steps 2, 3 and 4 indicated in Paragraph 4.2.10. The following steps of the first method are indeed useful to obtain a higher-level representation of found road markings, since they are not yet considered as a set of points but part of a line described with a specific polynomial function, leading to the possibility of making more complex reasonings. A representation like this, moreover, is more suitable for the use in other tasks.

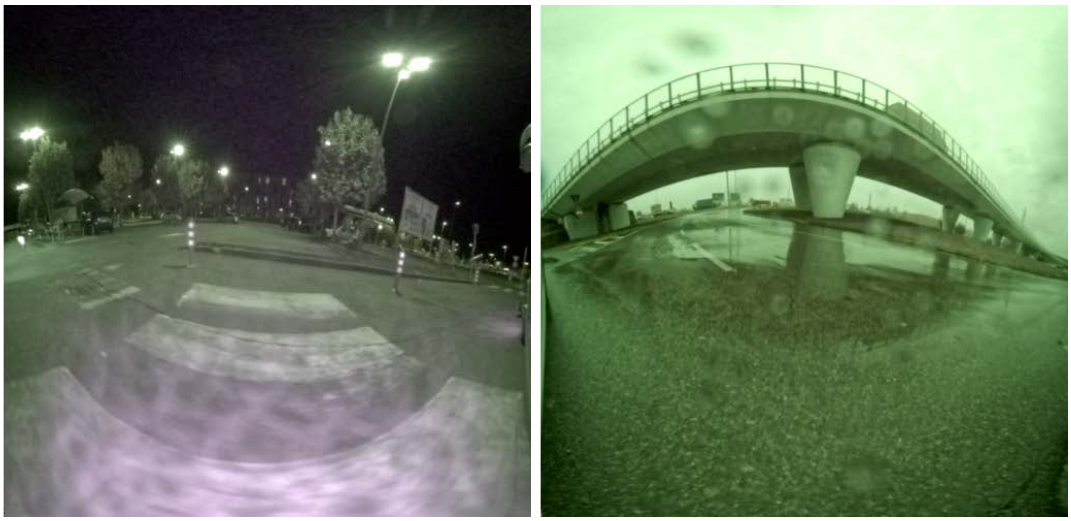


Figure 5.8: True negatives obtained with the CNN.

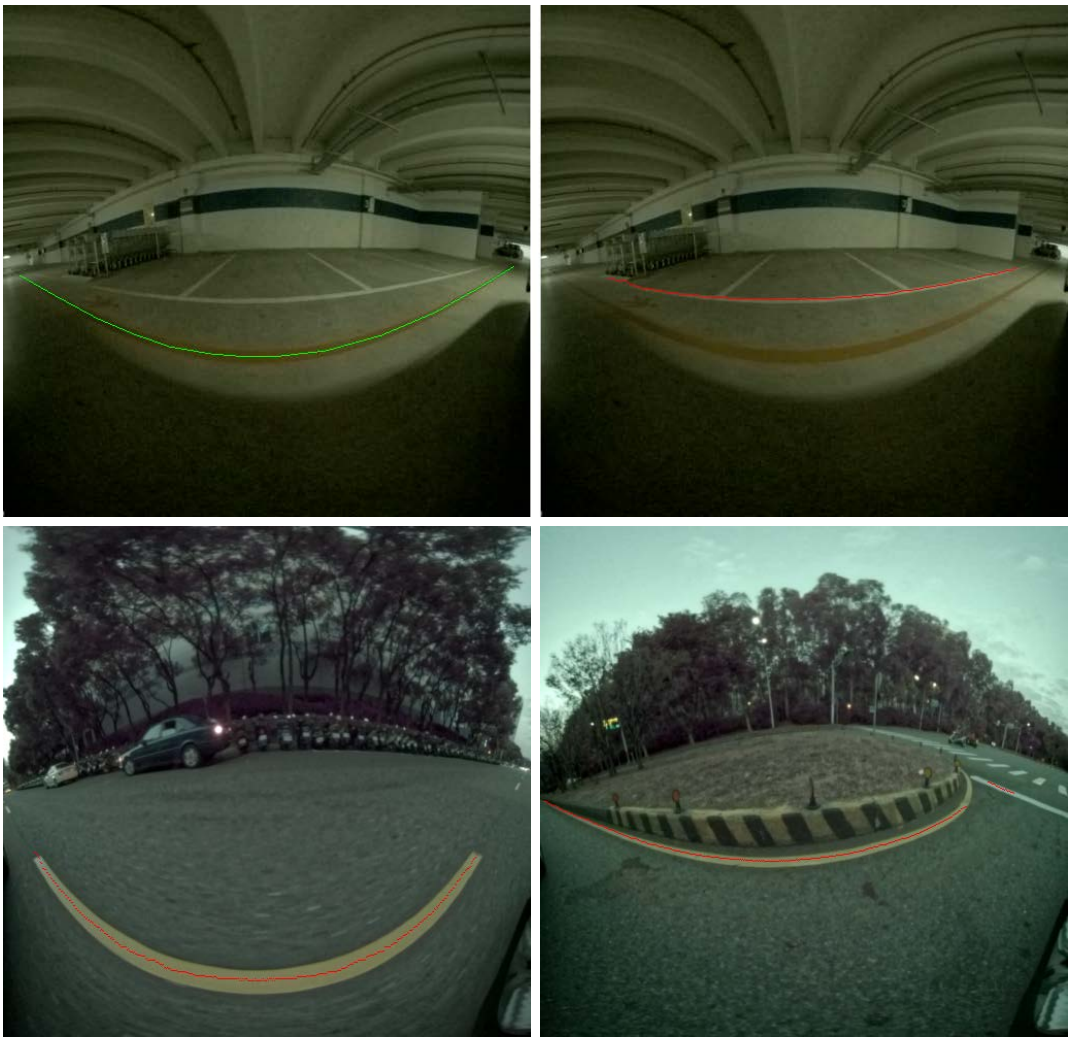


Figure 5.9: Yellow lines misdetection and detection.

6

Comparison of approaches

In this Chapter, the two approaches discussed are analyzed focusing on their major advantages and drawbacks. They are then compared referring to the different methods used to detect lanes described in Chapters 4 and 5.

6.1 Recap of pros and cons

In previous chapters traditional Computer Vision-based and Neural Networks-based methods have been largely discussed, making pros and cons of each of them appear. Summing up all the arguments expressed, classical Computer Vision approaches have been used for decades and this is a guarantee of proven reliability. They are the perfect solution for the recognition of objects whose relevant features are clearly visible. On the other hand, the task of finding them must be manually executed and the number of parameters, that have to be hand-tuned, can easily become very high. This is the main drawback of these methods, but if the reasoning behind the features extracted is good, the system obtained is robust. Moreover, it is also deterministic: this is another important advantage, since we can explain why it works or not and consequently it is easier to take countermeasures. The last advantage, that is sometimes wrongly considered secondary due to the continuous development of more powerful architectures, is the low computational power these systems require. Indeed, frequently these algorithms have to be executed in embedded systems, where using as few resources as possible is crucial.

Deep Learning, and more specifically Convolutional Neural Networks, is applied in lots of fields, and automotive is one of the most challenging. As discussed in [4], it is undeniable that in recent years these methods have outperformed traditional Computer Vision for difficult problems. They generally work better than previous methods thanks especially to the ability of performing well also when they are used to make predictions on unseen data. In this case we say that the obtained model generalize well on data. Human-based extraction of relevant features is no longer needed since the algorithm automatically finds them. On the other hand, the need for lots of annotated and as

various as possible data, for a network architecture that has to be demonstrated to be suitable for a specific task and the relevant time needed for training are important drawbacks. Due to the high computational complexity required, this method is not easily accessible when used for complex tasks. Moreover, it is more difficult to adapt these algorithms to embedded systems. Although these techniques are not deterministic but only probabilistic (like every other machine learning method), there exist several studies in literature whose aim is to understand what and how a neural network learns.

The best choice could be mixing Computer Vision and Deep Learning, obtaining hybrid approaches with better performances, also in terms of time. For example, as proposed in [4], in a video surveillance system it is possible to use Computer Vision algorithms to detect faces in a scene and then they can be passed to a Neural Network for identification. In this way, the former does a pre-processing finding the relevant zones to check and the latter needs to deal only with a limited area of the image, saving significant computer resources. Moreover, topics like 3D vision, panoramic stitching, geometric deep learning and visual SLAM are valid candidates for hybrid algorithms. In these cases, Computer Vision is useful for retrieving some information easily from images, without requiring the Network to learn it. A different approach is, for instance, those proposed in Chapter 5.7.

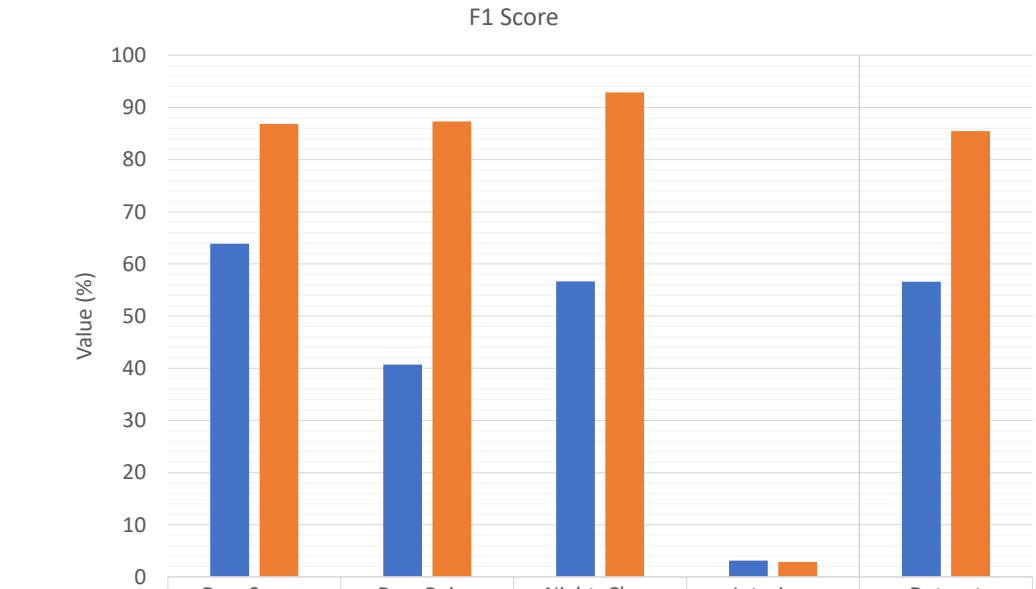
6.2 Results comparison

The results show that the CNN-based method, without any doubt, works better than the other one in most of the cases, outperforming it by about 30%. Graphs in Figures 6.1 and 6.2 show the differences in terms of F1 score and accuracy (that are the main metrics chosen for expressing the results of the tests and are defined in Chapter 4.3).

Considering the whole dataset, the more traditional approach achieves an F1 Score of 56,67% and an accuracy of 43,79%, to be compared respectively with 85,45% and 76,92% of the other one: what we obtain is a difference of 28,88% for the F1 score and of 33,13% for the accuracy. Only the interiors images reward the classical approach, but the difference is negligible. Furthermore, we remember that images in this illumination condition are affected by a problem caused by the presence of yellow principal lines in place of white ones.

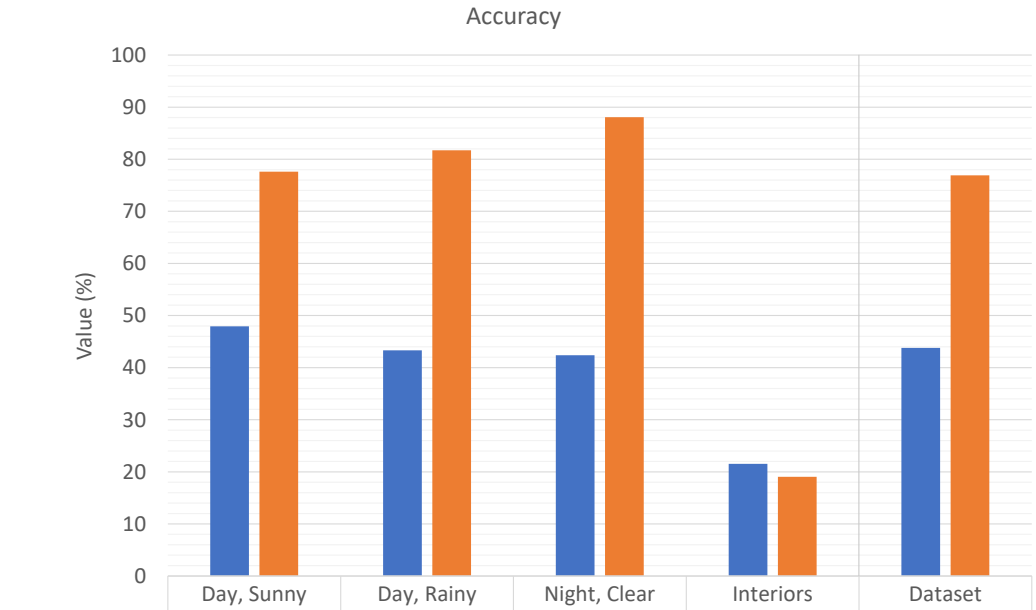
The second approach is able to detect efficaciously faded road markings while the other one fails if the line is difficultly visible, as we can see from the comparison in Figure 6.3.

Also shaded lines are problematic, since the the gradient of an image having a partially shaded road marking shows multiple rising edges (due to the brightness difference between the road surface and the line, but also because of the difference between the



	Day, Sunny	Day, Rainy	Night, Clear	Interiors	Dataset
Traditional CV	63,88	40,70	56,66	3,13	56,57
CNN	86,84	87,30	92,90	2,86	85,45

Figure 6.1: Comparison of F1 Score for the two approaches.



	Day, Sunny	Day, Rainy	Night, Clear	Interiors	Dataset
Traditional CV	47,93	43,33	42,39	21,52	43,79
CNN	77,63	81,71	88,07	19,05	76,92

Figure 6.2: Comparison of accuracy for the two approaches.

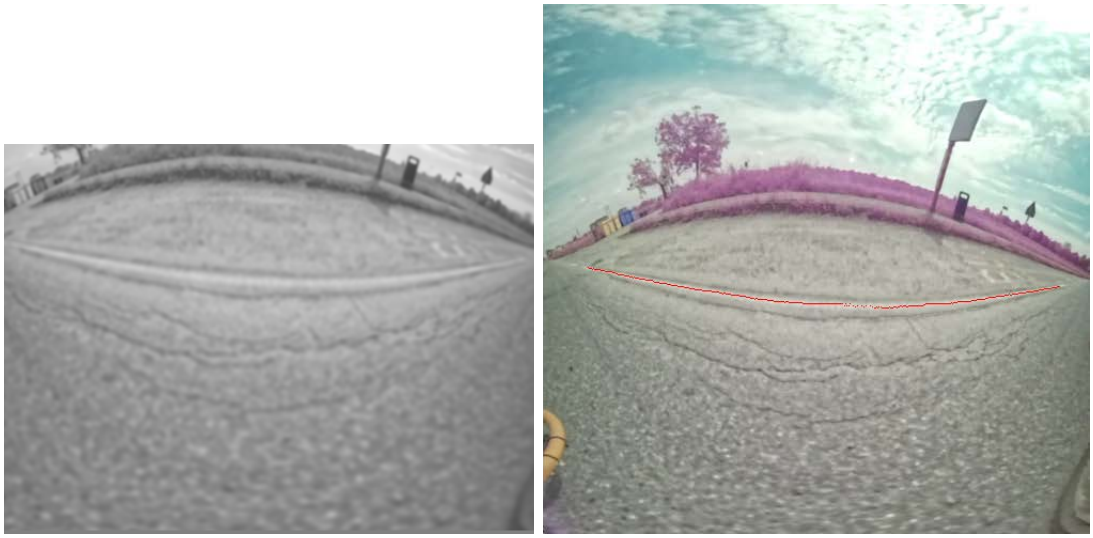


Figure 6.3: Faded line not found with traditional CV and correctly detected with CNN.

shaded and the illuminated area of the same line) and falling edges. This causes, in the best case, the detection as road marking of only its illuminated part, as visible in Figure 6.4.

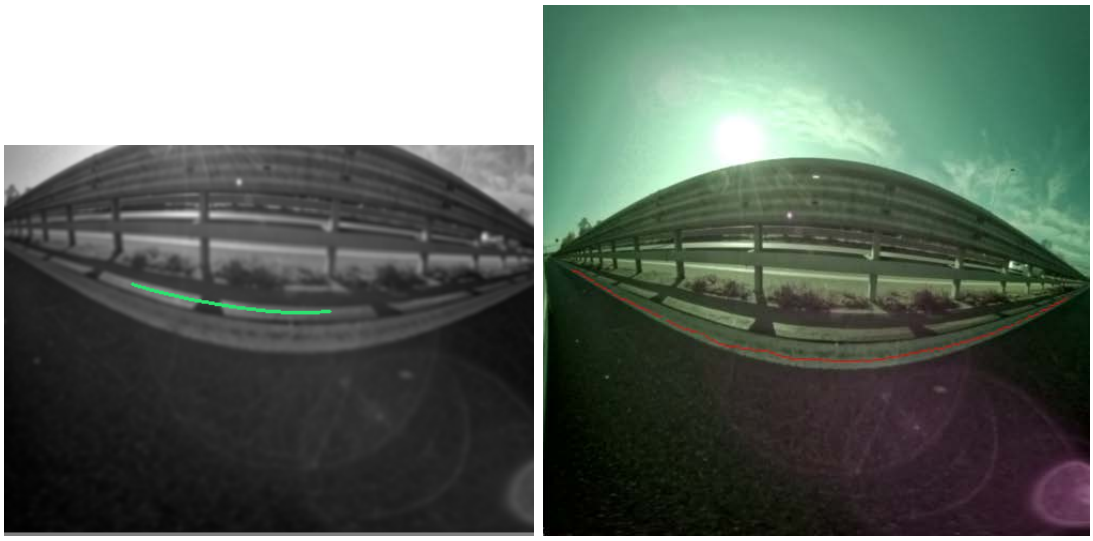


Figure 6.4: Shaded line wrongly detected with traditional CV and found with CNN.

The lack of dependence from the gradient calculation is an advantage also for the detection in night images because of the presence of artifacts caused by the illuminator. They are, indeed, correctly recognized as DLD transitions, even though they do not represent a road marking. This makes us understand that the number of hand-crafted features used to detect a road line and to distinguish it from other objects, although it is not low since all of them are manually selected and tuned, is insufficient to deal with

all the possible situations. At the same time, we can deduce that the algorithm does not actually recognize road markings but only objects that have some of the characteristics of a line. The CNN, instead, actually learns what a road marking should be and the ability to recognize it improves as the number and the variety of the training images given increases. We should not forget, however, that the number of parameters used is orders of magnitude higher (3 513 069 versus 31): so many parameters would be impossible to be managed by a human being.

7

Conclusions

In this work we studied in detail the different techniques presented over the years for the detection of lanes by finding the road boundaries. Most of them make use of standard pinhole cameras, while they seldom take advantage of wide-angles cameras like fisheye or even omnidirectional ones.

However, the use of lateral fisheye cameras is not common, so we decided to study this possibility since it has several advantages like the high quantity of information captured, in particular in the horizontal direction. The innovation is also the absence of a computationally-demanding process like an Inverse Perspective Mapping (IPM), leading us to work directly on spherical images, whose distortions are not negligible. We then proposed two solutions to this problem, the first with the use of traditional Computer Vision techniques alone, the second based on Deep Learning, specifically with the use of a Convolutional Neural Network. After the creation of a custom annotated dataset with 3705 images representing road scenes, we implemented the algorithms and we trained them on it.

The results show that the second and more recent method outperforms the other one in almost all the scenarios taken into account, that are daylight and night illuminations, sunny and rainy weather conditions, and finally interiors. Only in the latter the performances of the first method result better, even if the difference is very low (only 1%) and the reason is the presence of dark yellow lines in place of typical white ones the systems are usually called to detect, as explained in detail in Chapters 5.6 and 6.2. It is important to remember that the images of the dataset also contain lines that are very difficult to detect.

The use of two different approaches to solve the same problem has given us the opportunity of understanding the advantages and the limitations of them. While the development of the traditional computer vision-based method required us to manually find the most relevant characteristics of road markings of different types and with the most disparate conditions, at the same time this allowed us to change the behavior of the algorithm by either adding new parameters or tuning better an already-existing one. This means that the robustness of the algorithm is strictly related to the quality of the features found and to the parameters tuning. This becomes more and more difficult as

the number of conditions the system has to work in increases. The second approach, instead, required us to find a valid neural network architecture for the specific task and to train it to obtain a model that generalizes well on data. The results are improvable essentially by giving more data as input to the network, by applying changes to the loss function or with several modifications to the architecture.

As a consequence, the high complexity of the situations an autonomous driving system must deal with is the reason why nowadays most of the approaches introduced are not yet based on traditional computer vision alone. The synergy between the two approaches could be the key for a new kind of hybrid algorithms that combine the advantages of them.

Bibliography

[1] "2018 road safety statistics: what is behind the figures?", 4th April 2019, https://europa.eu/rapid/press-release_MEMO-19-1990_en.htm

[2] "Road fatality statistics in the EU (infographic)", 16th April 2019, https://www.europarl.europa.eu/news/en/headlines/society/20190410STO36615/road-fatality-statistics-in-the-eu-infographic

[3] "Cell phone use while driving: summary", February 2018, https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/pdf/ersosynthesis2018-cellphone-summary.pdf

[4] N. O' Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, J. Walsh, "Deep Learning vs. Traditional Computer Vision", *Conference Paper, IMaR Technology Gateway, Institute of Technology Tralee, Tralee, Ireland*, April 2019.

[5] R Danescu, S. Nedevschi, "Robust Real-Time Lane Delimiting Features Extraction", *Proceedings of* 2^{nd} *IEEE International Conference on Intelligent Computer Communication and Processing (ICCP 2006)*, Cluj Napoca, Romania, pp. 77-82, September 1st - 2nd, 2006.

[6] R. Danescu, S. Nedevschi, M. M. Meinecke, T.B. To, "Lane Geometry Estimation in Urban Environments Using a Stereovision System", *Proceedings of the 2007 IEEE Intelligent Transportation Systems Conference*, Seattle, USA, pp. 271-276, September 30th - October 3rd, 2007.

[7] M. Felisa, P. Zani, "Robust monocular lane detection in urban environments", 2010.

[8] E. Pollard, D. Gruyer, J.P. Tarel, S.S. Ieng, A. Cord, "Lane Marking Extraction with Combination Strategy and Comparative Evaluation on Synthetic and Camera Images", *IEEE Conference on Intelligent Transportation Systems*, Washington, DC, USA, October 5-7, 2011.

[9] T. Veit, J. Tarel, P. Nicolle, and P. Charbonnier, "Evaluation of road marking feature extraction", *IEEE 11th International Conference on Intelligent Transportation Systems*, pp. 174-181, 2008.

[10] J. Hur, S. Kang, S. Seo, "Multi-Lane Detection in Urban Driving Environments Using Conditional Random Fields", *IEEE Intelligent Vehicles Symposium*, Gold Coast, Australia, June 23-26, 2013.

[11] S. Li, Y. Shimomura, "Lane marking detection by side fisheye camera", *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Acropolis Convention Center, Nice, France, pp. 606-611, September 22-26, 2008.

[12] R. Boutteau, X. Savatier, F. Bonardi, J. Y. Ertaud, "Road-line detection and 3D reconstruction using fisheye cameras", *Proceedings of the IEEE Annual Conference on Intelligent Transportations Systems (ITSC)*, The Hague, The Netherlands, October 6-9, 2013.

[13] G. Bacchiani, M. Patander, A. Cionini, D. Giaquinto, "Parking Slots Detection on the Equivalence Sphere with a Progressive Probabilistic Hough Transform", *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2017.

[14] P.V.C. Hough, "Method and means for recognizing complex patterns", U.S. Patent 3069654, December 1962.

[15] H. Lee, S. Kim, S. Park, Y. Jeong, H. Lee, K. Yi, "AVM / LiDAR Sensor based Lane Marking Detection Method for Automated Driving on Complex Urban Roads", *IEEE Intelligent Vehicles Symposium*, Redondo Beach, CA, USA, June 11-14, 2017.

[16] J. Kim, M. Lee, "Robust Lane Detection Based On Convolutional Neural Network and Random Sample Consensus", textitInternational Conference on Neural Information Processing, November 2014.

[17] A. Gurghian, T. Koduri, S. V. Bailur, K. J. Carey, V. N. Murali, "DeepLanes: End-To-End Lane Position Estimation using Deep Neural Networks", *IEEE Conference on Conputer Vision and Pattern Recognition (CVPR) Workshops*, Ford Research and Innovation Center, Palo Alto, California, 2016.

[18] S. Lee, J. Kim, J. S. Yoon, S. Shin, O. Bailo, N. Kim, T. Lee, H. S. Hong, S. Han, I. S. Kweon, "VPGNet: Vanishing Point Guided Network for Lane and Road Marking Detection and Recognition", *IEEE International Conference on Computer Vision (ICCV)*, pp. 1965-1973, 2017.

[19] J. Long, E. Shelhamer, T. Darrell, "Fully convolutional networks for semantic segmentation", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

[20] J. Deng, W. Dong, R. Socher, L. Li, K. Li, L. Fei-Fei, "Imagenet: A large-scale hierarchical image database", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

[21] O. Ronneberger, P. Fischer, T. Brox, "U-Net: Convolutional networks for biomedical image segmentation", *International Conference on Medical image computing and computer-assisted intervention*, pages 234-241, Springer, 2015.

[22] V. Badrinarayanan, A. Kendall, R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation", *arXiv preprint, arXiv:1511.00561*, 2015.

[23] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs", *arXiv preprint, arXiv:1606.00915*, 2016.

[24] G. Lin, A. Milan, C. Shen, I. D. Reid, "RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, p. 5, 2017.

[25] H. Zhao, J. Shi, X. Qi, X. Wang, J. Jia, "Pyramid scene parsing network", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2881-2890, 2017.

[26] A. Paszke, A. Chaurasia, S. Kim, E. Culurciello, "ENet: a Deep Neural Network architecture for real-time semantic segmentation", *arXiv preprint, arXiv:1606.02147*, 2016.

[27] E. Romera, J. M. Alvarez, L. M. Bergasa, R. Arroyo, "ERFNet: Efficient Residual Factorized ConvNet for real-time semantic segmentation", *IEEE Transactions on Intelligent Transportation Systems*, pp. 263-272, 2018.

[28] X. Pan, J. Shi, P. Luo, X. Wang, X. Tang, "Spatial as Deep: Spatial CNN for Traffic Scene Understanding", *arXiv preprint, arXiv:1712.06080*, 2017.

[29] "TuSimple benchmark", November 2019, https://github.com/TuSimple/tusimple-benchmark

[30] Y. Wu, R. Yang, J. Zhao, L. Guan, W. Jiang, "VH-HFCN based Parking Slot and Lane Markings Segmentation on Panoramic Surround View", *IEEE Intelligent Vehicles Symposium*, Changshu, Suzhou, China, pp. 1767-1772, June 26-30, 2018.

[31] T. Yang, Y. Wu, J. Zhao, L. Guan, "Semamtic segmentation via highly fused convolutional network with multiple soft cost functions", *arXiv preprint, arXiv:1801.01317*, 2018.

[32] D. Neven, B. De Brabandere, S. Georgoulis, M. Proesmans, L. Van Gool, "Towards End-to-End Lane Detection: an Instance Segmentation Approach", *arXiv preprint, arXiv:1802.05591*, 2018.

[33] J. Kim, C. Park, "End-To-End Ego Lane Estimation Based on Sequential Transfer Learning for Self-Driving Cars", *CVPR Workshops*, pp. 1194-1202, 2017.

[34] L. Deng, M. Yang, B. Hu, T. Li, C. Wang, "Semantic Segmentation-Based Lane-Level Localization Using Around View Monitoring System", *IEEE Sensors Journal*, vol. 19, no. 21, pp. 10077-10086, November 2019.

[35] Y. Hou, Z. Ma, C. Liu, C. C. Loy, "Learning Lightweight Lane Detection CNNs by Self Attention Distillation", *arXiv preprint, arXiv:1908.00821*, August 2019.

[36] B. Coors, A. P. Condurache, A. Geiger, "SphereNet: Learning Spherical Representations for Detection and Classification in Omnidirectional Images", In: *V. Ferrari, M. Hebert, C. Sminchisescu, Y. Weiss, Computer Vision - ECCV 2018*, Lecture Notes in Computer Science, vol 11213. Springer, Cham, 2018.