Kuschnig Lukas, BSc

Lane Detection and Ego-Vehicle Localization in Multi-Lane Scenarios

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Bischof Horst
Institute for Computer Graphics and Vision

Dipl.-Ing. Schinagl David, BSc
Institute for Computer Graphics and Vision

Graz, Austria, June. 2020

Abstract

Lane detection and tracking as well as ego-lane estimation and relative ego-vehicle localization in multi lane scenarios is a crucial task for current Advanced Driver Assitstance Systems (ADAS) and Automated Driving (AD) applications. This thesis is devoted to develop and evaluate different concepts for vision-based lane detection and tracking. Based on in-depth literature research three different algorithms have been implemented. The algorithms specifically differ in the method for lane segmentation but follow a common processing pipeline. A special focus was put on the evaluation of traditional and machine learning based methods. One of the algorithms uses conventional techniques for region based lane feature extraction, whereas the remaining algorithms use deep learning architectures for segmentation. Hough transform in combination with RANSAC model fitting is used in the detection stage of all algorithms. To increase the performance, tracking with the help of a Kalman filter was implemented. In addition, a LiDAR sensor is used to improve the lane detection by utilizing the 3D point cloud for road plane fitting, road boundary extraction and outlier removal in the segmentation masks. In a final step, an evaluation of the performance and accuracy based on labeled ground truth data was done. To have an estimate if and how well the incorporation of LiDAR point clouds improves the overall performance and accuracy, the algorithms are separated again into a basic version and an advanced version for evaluation. In comparison with each other, the basic versions do not use LiDAR point clouds for outlier detection or frame-wise road plane fitting and only have a static Region Of Interest (ROI). The results show, that all algorithms are able to perform well under normal as well as challenging conditions. The Advanced Conventional Algorithm and the deep-learning based algorithms show close results, but the Advanced DeepLab Algorithm performs best. Due to its high Precision and Recall as well as its low standard deviation of the detection error, the algorithm achieves an overall high detection rate combined with a high accuracy.

Keywords. vision-based lane detection, lane tracking, ego-lane estimation, deep learning, segmentation

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date	Signature

Acknowledgments

First and foremost I would like to thank those who supported me in the course of this work and made it possible at all. I would like express my gratitude to my supervisor Prof. Bischof who gave me the opportunity to write the thesis at the *Institute of Computer Vision and Graphics* and supported me in all kinds of questions during this thesis.

I would also like to thank David Schinagl who guided me through the whole thesis. I would like to thank him for being open to questions at any time and providing constructive suggestions in so many areas of this thesis. Because of his in depth knowledge in any area of computer vision we had very productive discussions what drove the development for this thesis forward.

Furthermore, I would like to thank my supervisor Orasche Gerbert who gave me the opportunity to write this thesis in collaboration with the AVL List GmbH and supported me in all matters. Last but not least, I would like to thank my colleagues Steffen Metzner, Julian Filwarny, Mansuet Gaisbauer and Pankit Shah who laid the foundation for the thesis by bringing in the idea of the core theme to be dealt with and who were on my side whenever I had questions.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Motivation

Over the years Advanced Driver Assitstance Systems (ADAS) and Automated Driving (AD) applications became more and more mature and complex. In the beginning, the systems had to work on the basis of minimal resources. Nowadays, such systems consist of a large number of sensors and control units, which characterize the surroundings of the vehicle in order to derive actions from it. Such systems can be used to intervene in the driving process, such as, parking assistants, lane change assistants or collision warning systems.

In order to guarantee and increase the safety of *ADAS* and *AD* applications, the validation and testing of such systems is of high importance. The aim of the Dynamic Ground Truth (DGT) team from AVL List GmbH is to develop a reference measurement system, that reliably acquires ground truth data for the evaluation of ADAS applications and scenario generation/extraction using a combination of high-resolution measuring systems. As for most current advanced driver assistance systems lane detection and tracking is one of the fundamental aspects for the DGT system. Furthermore, it is crucial to determine a high-quality ego-vehicle localization within the lane to improve the global positioning and the general quality of the estimated lanes and structural correspondences in virtual scenarios, especially in multi lane scenarios of highways.

Many researches focus on the study of vision-based lane detection methods. In order

to improve the development of future vision-based lane detection and tracking, it is important to gain know-how about several computer-vision based approaches for lane detection, ego-vehicle localization and ego lane estimation. In addition, one would like to gain knowledge about the differences, limitations and drawbacks of conventional and deep learning-based approaches for lane detection and tracking.

1.2 Proposed Methods

In the course of this thesis three different algorithms have been implemented and compared among each other. Furthermore the algorithms have been compared with state-of-the-art algorithms of other research outcomes. As already mentioned, a special focus is kept on the comparison of conventional techniques and deep-learning based techniques for lane segmentation. In the course of this thesis following questions should be answered:

1. Is there a significant difference in accuracy and performance between conventional image processing techniques compared to deep learning based approaches?

2. Do deep learning based approaches overcome instabilities and failures in complex situations such as tunnel drives or dynamic illumination?

3. Does the use of an additional LiDAR improve the overall performance of lane detection?

4. Is it possible to estimate the vehicles lateral position in the ego-lane with an accuracy of 5 cm?

To answer this hypothesis, first, in-depth literature research is done. Based on this research a general processing pipeline is derived and three different solutions for lane detection and tracking are implemented. One solution is based on conventional image processing techniques, one algorithms is based on semantic segmentation and the third algorithm is based on instance segmentation using deep learning. The algorithms follow the same processing pipeline and exclusively differ in the lane segmentation task. To test and evaluate the algorithms, a set of performance metrics is defined which can be deployed to evaluate different kinds of lane detection algorithms. Additionally, some experimental results are done to find out the advantages and disadvantages as well as the operational limits of the algorithms. Based on the results the best algorithm for the DGT system is proposed.

1.3 Structure of the Thesis

The thesis is structured as follows:

1. **Chapter 2: Background**
   In this chapter a general overview of the methods and the fundamentals of image processing techniques and deep learning is given to better understand the core concepts of the algorithms implemented in this thesis.

2. **Chapter 3: Literature Review**
   This chapter is devoted to discuss the outcome of in-depth literature research in the field of lane detection and tracking.

3. **Chapter 4: Concepts**
   In this chapter the three solutions and all its methods implemented in this thesis are discussed in detail. The algorithms follow the same processing pipeline and mainly differ in the image segmentation stage. Based on the pipeline of the respective algorithms the sections are structured accordingly starting with pre-processing and ending up with the detected lanes and position estimation of the vehicle.

4. **Chapter 5: Evaluation**
   This chapter describes the used data sets, the testing criteria, some experimental results and the evaluation results of each individual algorithm. To have an estimate if and how well the LiDAR point clouds improve the result, the algorithms are divided into a basic version and an advanced version for evaluation. The basic version does not utilize LiDAR point clouds for lane detection whereas the advanced version does.

5. **Chapter 6: Conclusion**
   In this chapter a conclusion is made and the ideal solution for individual use cases is proposed. Furthermore, a proposal for future development is done.

2 Background

Contents

This chapter is devoted to give a general overview about the concepts of image processing techniques. It will give an overview about image formation and will explain the fundamentals of neural networks and other core methods required to better understand the algorithms implemented in this thesis.

2.1 Notations

Before we start to discuss the core concepts in detail it is necessary to introduce the mathematical notations used in this thesis. The notations are summarized in Table 2.1. Scalars are depicted with italic fonts(e.g., a or b) whereas vectors and matrices are represented in bold(e.g., $\mathbf{v}$ or $\mathbf{M}$). Note that, vectors are written in lower case and matrices are represented in upper case letters. Mapping functions from one space to another are depicted as calligraphic letters in upper case format (e.g., $\mathcal{M}$). Vector spaces are represented in upper case double lined format(e.g., $\mathbb{Z}$).

Entity	Notation
Scalars	a, c_i
Vectors	$\mathbf{v}$
Matrices	$\mathbf{M}$
Vector norm	$\|\|\mathbf{v}\|\|$
Vector Spaces	$\mathbb{R}^n$
Mapping Function	$\mathcal{P} : \mathbb{R}^3 \rightarrow \mathbb{R}^2$

Table 2.1: List of notations used in this thesis.

2.2 Image Formation

To describe the projection of the real world scene to the camera scene a projection model must be defined. For this thesis the projective camera model is used for 3D to 2D image projections. This camera model is based on the pin hole camera model and is most widely used in computer vision. The pin hole camera model is based on the assumption that the relation between image coordinates and world coordinates is projective, meaning that the projection of straight lines from world space also leads to straight lines in camera space. Due to the dimensional extension from camera perspective in $\mathbb{R}^2$ to world coordinates in $\mathbb{R}^3$, each image point can be described as the intersection point of the image plane and the ray spanned by the 3D projection center and 3D world point, as depicted in Fig. 2.1.

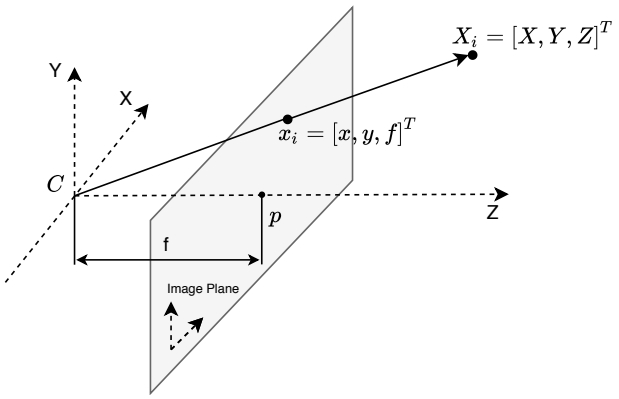


Figure 2.1: Pinhole camera model.

From a mathematical perspective the projection of an world point $\mathbf{X_i}$ to the corresponding point $\mathbf{x_i}$ in the image plane can be computed with the camera matrix $\mathbf{P}$ which is defined as

$$\mathbf{P} = \mathbf{K}\,[\mathbf{R}|-\mathbf{RC}] \quad . \tag{2.1}$$

The matrix $\mathbf{K}$ is the calibration matrix embedding the camera`s intrinsic parameters and enables the transformation of points from a relative world coordinate system to the camera

coordinate system and visa versa. The matrix is defined as

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \quad , \tag{2.2}$$

with fx and fy describing the focal length $\mathbf{f}$ and px and py describing the pixel coordinates of the principle point $\mathbf{p}$. The focal length $\mathbf{f}$ is the distance from the center of projection to the camera plane and the principle point $\mathbf{p}$ is the point of intersection of the optical axis with the image plane. Since the calibration matrix just describes the transformation to a relative coordinate system in $\mathbb{R}^3$, the matrices $\mathbf{R}$ and $\mathbf{C}$ describe the rotation and translation to a world coordinate system. The perspective transformation from a homogeneous world point $\mathbf{X_i}$ to its corresponding homogeneous image point $\mathbf{x_i}$ can be computed as follows:

$$\mathbf{x}_i = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & -\mathbf{RC} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.3}$$

or in a more compact form

$$\mathbf{x}_i = \mathbf{P}\mathbf{X}_\mathbf{w} \quad . \tag{2.4}$$

The get the final pixel coordinates the homogeneous image point has to be normalized to $\mathbf{x} = (x/z, y/z, 1)^{\mathrm{T}}$. The discussed projective transformation model describes the transformation of real world points to image points under ideal conditions. Other effects like distortions are not considered till now. The most common distortion types are introduced by camera lenses and can be classified in radial and tangential distortion. The radial distortion effect leads to straight lines represented as curved lines in image space, what means points are not projected to the point of the ideal pinhole camera model, but to another position. Tangential distortion can occur due to a misaligned lens what leads to the effect that regions with the same distance to the camera seem to differ in distance and appear to be closer or farther away. This effects can be compensated by choosing a proper lens distortion model. For this thesis the widely used Brown-Conrady model introduced by Brown [7] was chosen. With this model radial distortion is solved by

$$\begin{aligned} x_{corrected} &= x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \\ y_{corrected} &= x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \end{aligned}$$

and tangential distortion is solved by

$$\begin{aligned} x_{corrected} &= x + 2p_1 xy + p_2(r^2 + 2x^2) \\ y_{corrected} &= y + p_1(r^2 + 2y^2) + 2p_2 xy \quad . \end{aligned}$$

with the five distortion coefficients $k1, k2, k3, p1, p2$.

The intrinsic and extrinsic calibration parameters as well as the distortion coefficients are determined in a camera calibration procedure. The most common procedure for camera calibration is Zhangs Method [61]. For this thesis the parameters are given and the camera calibration is not part of it, therefore it will not be discussed further.

2.3 Random Sample Consensus

Random Sample Consensus (RANSAC) is the most popular robust estimator in computer vision and was first introduced by Fischler and Bolles [18] in 1980. It is a non-deterministic algorithm to estimate the parameters of a global model from a set of data points containing outliers in an iterative way. The algorithms is non-deterministic and returns different solutions on different computations. Furthermore, it relies on the assumption, that there exist a sufficient amount of inliers in the given data set. Fig. 2.2 shows an example of line model parameter estimation based on RANSAC.

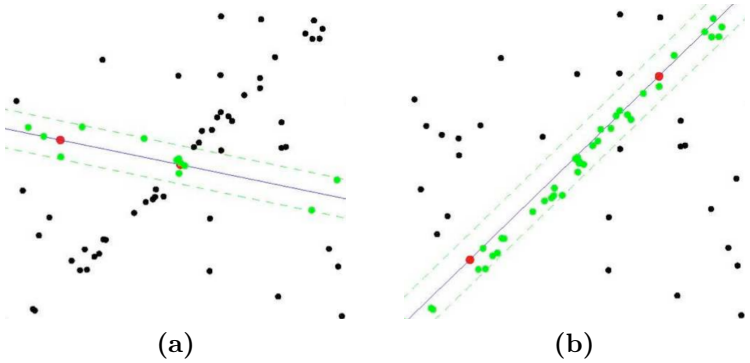


Figure 2.2: Example of line model parameter estimation based on RANSAC algorithm with a) a bad estimate containing sparse amount of inliers and b) a good estimate with maximum number of inliers. Picture taken from [37].

The principle of the algorithm is summarized in Algorithm 1. The goal is to find the parameters θ of a given model $\mathbf{M_0}$ that optimize a cost function $J*$ in a given data set $\mathbf{D}$. At first a set of data points $\mathbf{s}\epsilon\mathbf{D}$ is randomly selected. Afterwards, the corresponding model or respectively its parameters based on this samples are computed. Based on the estimated model parameters, the error to each data point in $\mathbf{D}$ is computed and the inliers within a given thresh hold are selected. After a maximum number of iterations the model with the most amount of inliers is selected.

Algorithm 1 RANSAC Algorithm

$i \leftarrow 0$
while $i < N$ **do**
    $i = i + 1$
    Randomly select a set of data points $\mathbf{s} \epsilon \mathbf{D}$
    Compute model parameters θ_i based on samples s
    Compute cost J_i of the model
    if $J_i < J*$ **then**
        $J* \leftarrow J_i$
        $\theta* \leftarrow \theta_i$
    end if
end while
Compute model parameters $\theta*$ considering all inliers
$\theta \leftarrow \theta*$

2.4 Artificial Neural Networks

Artificial Neural Networks (NNs) are designed to cope with complex problems such as classification, regression or pattern recognition. The principle design of neural networks is inspired by the human brain. In the human brain neurons are connected with each other spanning a complex network, where each neuron is triggered if its inputs meet a specific hypothesis. Fig. 2.3 shows the simplest model of a neuron used in a *NN*. In a neuron, an arbitrary number of weighted inputs x_i with some bias b is summed. If the result meets some hypothesis, such as exceeding a threshold, an output is created according to a specific activation function. The most common activation functions are the sigmoid function, the hyperbolic tangent function and the rectified linear activation function. More details about activation functions can be found in [40]. From a mathematical perspective a neuron is defined as

$$y = f\left(\sum_i x_i w_i + b\right) \quad , \tag{2.5}$$

with y as output, x_i as the inputs, w_i as the weights, b as bias and f as the non-linear activation function.

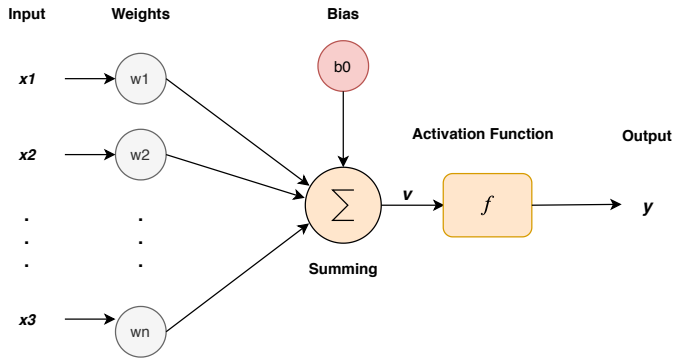


Figure 2.3: Model of a neuron in a *NN* with x as the input array, w as the weights, b as bias and f as the activation function.

In a *NN* an arbitrary amount of neurons is individually connected, that the output of some neurons forms the input of neurons in another layer. Fig. 2.4 represents a potential architecture of a *NN* consisting of several neurons. Depending on the amount of layers, *NNs* are often referred as Deep Neural Network (DNN).

Generally speaking, Neural Networks are trained by feeding the network with examples and comparing the output with some ground truth based on a loss-function. According to the comparison the weights and biases of the individual neurons are adjusted. After several iterations the network starts to differentiate between several classes. This process is called gradient descent and will be discussed in Section 2.5. If a models output shows a high accuracy with trained data, but performs poor with unseen data of same context, the model is over fit. This can be avoided based on several techniques such as regularization, early stopping or data augmentation that will not be discussed in this thesis. For more details see [40].

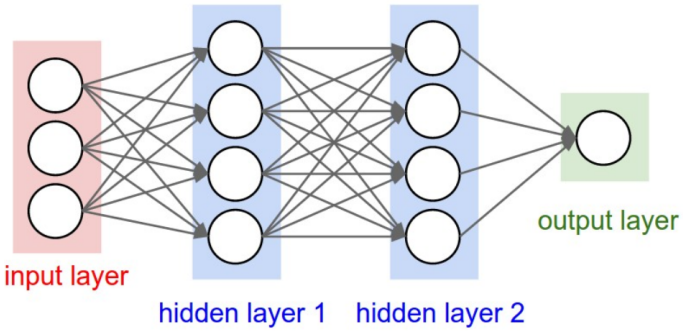


Figure 2.4: Basic architecture of a neural network with two hidden layers. Image taken from [28].

2.5 Gradient Descent and Backpropagation

Gradient descent is a common technique used in supervised learning to find the local minimum of a differentiable function by making steps in the negative direction of the gradient. For a better understanding, one can think about finding the lowest point on a landscape. Starting from an arbitrary point someone makes steps in the direction of the steepest descend until a minimum is reached.

In more detail, the output $\mathbf{y}$ of a *NN* is compared with a reference value based on a loss function L. The goal is to find the weights and biases, denoted as $\mathbf{W}$ which minimize the error of the loss function

$$\min_{\mathbf{W}} \frac{1}{N} \sum_{Ni} L\left(gt_i, y_i | \mathbf{W}\right) \quad . \tag{2.6}$$

To do so, the models parameters $\mathbf{W}$ are updated iteratively with backpropagation, a technique to adjust the weights and biases by propagating the gradients from output to input using the chain rule. The goal is to find the solution which minimizes Eq. (2.6). This can be described as

$$\mathbf{W^{t+1}} = \mathbf{W^{t}} - \alpha \sum_{i} \frac{\partial L\left(gt_i, y_i | \mathbf{W}\right)}{\partial \mathbf{W}} \quad , \tag{2.7}$$

with α as the learning rate and t as the number of iteration.

In *NN*, a more common implementation of gradient descent is *stochastic gradient descent*. With stochastic gradient descent an approximation of the gradient is done by considering just a random subset of the data often called "mini batch", where the size of the mini batch correlates with the goodness of the approximation of the gradient. This approach is computational less expensive and is therefore an optimization of gradient descent, what is a special advantage in applications with big data sets.

2.5.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a form of Neural Networks specially designed to extract features by introducing convolutions and was first introduced by Kunihiko et al. in [31]. *CNNs* are able to process multi-dimensional input data and are capable to consider spatial and temporal dependencies. Therefore, *CNNs* have a particular advantage in image processing or audio processing tasks. A *CNN* typically consists of three layers, each with another functionality: a convolutional layers, pooling layers and one or more fully connected layers [40].

The layers can be arranged, one behind the other in an arbitrary manner and as often as required, as can be seen in Fig. 2.5.

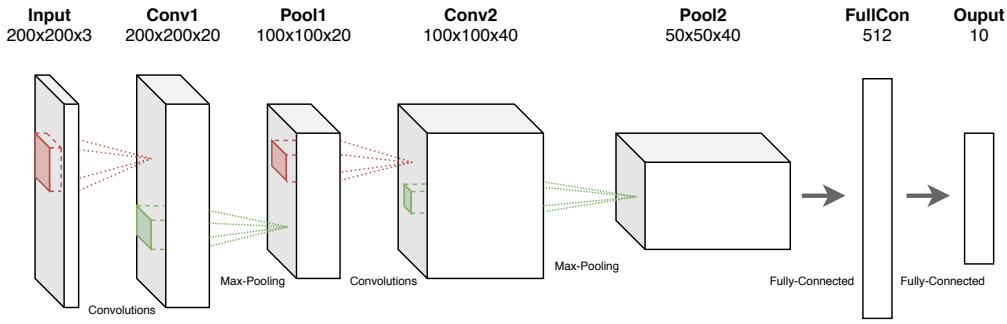


Figure 2.5: Potential architecture of a *CNN*.

Convolutional Layer

In convolutional layers the input is convoluted with a kernel of a given size. Taking multi-dimensional data as input, the kernel usually has the same number of dimensions as the input. In the case of a 3-channel RGB image the kernel has 3 dimensions as well. However, the size of the kernel, more specific the width and height of the kernel, can vary. The main objective in a convolutional layer is to extract features where the number of layers influences the granularity of the feature extraction. Starting with low-level features like edges, gradients or intensities, the architecture adapts to high-level features with the increase of the number of layers.

Pooling Layer

Due to the fact that computing convolutions, weights and parameter optimization of a network can be computational expensive, pooling layers are primary used to reduce the dimension. Furthermore, a pooling layer can be used to make the network more invariant to small displacements of the input by slightly shifting the input. Pooling does not effect the depth of the input, only the area. The reduction of the dimensions is mainly done with max-pooling or average pooling that returns the maximum value or respectively the average of a region covered by a kernel of a given size. Pooling is done for each layer separately and leads to more in-variance in terms of small translations and distortions [33].

However, there are some drawbacks in reducing the dimension, such as the loss of spatial information during each pooling step. There exist several pooling structures to address this problem [12]. Fig. 2.6, shows four different architectures. A *image pyramid* architecture introduces several parallel convolutions pipelines that get combined. The *Encoder-Decoder* architecture enables the transition of spatial information between pooling and unpooling layers. With *atrous convolution* the spatial information is retained by increasing the Field of View (FoV) instead of lowering the resolution. *Spatial pyramid pooling* is similar to the *image pyramid*, but with different pooling stages instead of different inputs.

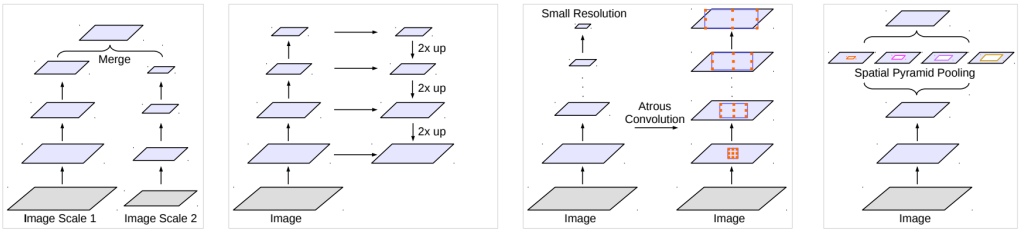


Figure 2.6: Four different architectures including a image pyramid structure, encoder-decoder structure, atrous convolution structure and spatial pyramid pooling(from left to right) to cope with the loss of spatial information. Image taken from [12].

Fully Connected Layer

Fully connected layers are mainly used in classification tasks. To learn the non-linear combination of the features extracted in the earlier layers, fully connected layers are used as a termination of the structure. The multidimensional layer containing features is flattened to a 1-D vector and is fed into a feed-forward *NN* that learns a non-linear function that is normalized with a function such as soft-max.

Fully Convolutional Network (FCN)

Fully Connvolutional Networks (FCNs) are an extension of *CNNs*. Beside many other fields of application they are often used for semantic segmentation tasks. Long et al. [36] show that an *FCNs* trained end-to-end and pixel-to-pixel, exceeds the State-of-the-Art (SotA) in semantic segmentation. By replacing the fully connected layer of a *CNN* with convolutional layers, a class probability map is generated enabling pixel-wise classification. Because pixel-wise classification needs an output with the same size as the input, the coarse probability map is up-sampled back to the given input size. Due to upsampling and deconvolutions the segmentation results at the structures boundaries often are inaccurate. This leads to unclear edges of the objects and makes the objects look blobby or smudgy. There are several methods to improve the accuracy such as a special architecture for pooling or conditional random fields [32]. For detailed information see [40]. *FCN* are often extended to an encoder-decoder structure. Badrinarayanan et al. [2] proposed an *FCN* with encoder-decoder structure consisting of 13 encoding layers followed by 13 decoding layers, as can be seen in Fig. 2.7.

2.6 Hough Transfrom

Hough-Transform was invented by Hough. [24] in 1962 and is a popular technique to estimate simple shapes such as lines or circles in an image. The idea behind the Hough transform is, that points connected by a line can be found by transforming the individual points to a specific space called Hough space. In the Cartesian coordinate system, a line

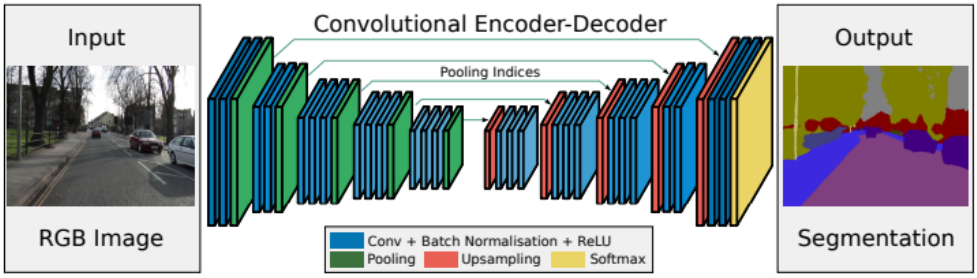


Figure 2.7: Architecture of SegNet consisting of an encoder stage and a decoder sage that upsamples its input based on the transferred pooling indices. Image taken from [2].

can be described as

$$y = ax + b \quad . \tag{2.8}$$

However, this line can also be represented as a single point in Hough space by plotting a against b. Consequently, the Hough transfrom of lines is a mapping from a line to a point in another space. A point in Cartesian coordinates can also be mapped to Hough space which is represented as line. With that, a point is mapped to all possible lines that pass through it.

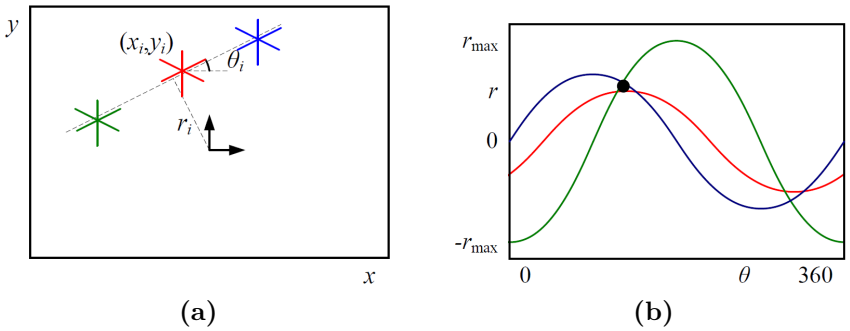
There is one problem with the presentation in Eq. (2.9). When the line is vertical, the gradient is infinitely large and cannot be represented in Hough space. Therefore, lines are represented in polar coordinates instead and defined as

$$r = x \cos \phi + y \sin \phi \quad . \tag{2.9}$$

Whenever a series of points in a Cartesian coordinate system is connected by a line, the equation of that line can be found by first plotting each point in the Cartesian coordinate system to the corresponding line in Hough space, and then finding the point of intersection in Hough space. The point of intersection in Hough space represents the r and ϕ values describing the line that passes consistently through all of the points in the series. Fig. 2.8 shows visualizes the estimation of the parameters of the line passing through three points.

2.7 Kalman Filter

The Kalman Filter is a popular algorithm in signal processing introduced by Kalman [26] in 1960. It provides a framework for state estimation and prediction of dynamic systems which are described as mathematical model over time. The Kalman Filter is working under the assumption that the system is linear and the noise in the filter is based on a Gaussian distribution. The Kalman Filter can be split in two processes called *prediction* step and *correction* as can be seen in Fig. 2.9.



(a) (b)

Figure 2.8: Hough Transform with **a** showing a line passing through three points. **b** sows the parameters of the potential lines which pass through each point. The point of intersection provides the line equation. Image taken from [54].

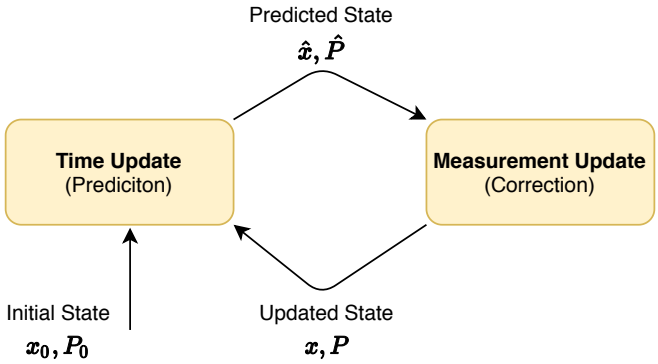


Figure 2.9: Operation of Kalman Filter.

Prediction Step

In a *prediction step* the Filter produces a prediction of the current state $\mathbf{x}_k$, also called *a priori* state, of the model considering the previous state $\mathbf{x}_{k-1}$ with its uncertainty in the previous period. The state $\mathbf{x}$ is a n-dimensional vector containing the necessary parameters to mathematically describe the model. The predicted state at a given time stamp is defined as

$$\hat{\mathbf{x}}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{a}_k + \mathbf{w}_k \tag{2.10}$$

with $\hat{\mathbf{x}}_k$ as thepredicted *a priori* state, $\mathbf{x}_{k-1}$ as the previous state, $\mathbf{A}$ as the state transition matrix associating the *a priori* state to the previous state, $\mathbf{a}_k$ as the control input, $\mathbf{B}$ the control matrix associating the control input to the state and a random variable $\mathbf{w}_k$ representing the noise of the state transition.

Furthermore, the error covariance also called *a priori* covariance estimate is predicted as follows

$$\hat{\mathbf{P}}_k = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q} \tag{2.11}$$

with $\hat{\mathbf{P}}_k$ as predicted *a priori* covariance matrix, $\mathbf{P}_{k-1}$ as the previous error covariance

and $\mathbf{Q}$ as the covariance of the process noise.

Correction Step

In a *correction step* the estimates are updated to an improved state estimation(*a posteriori* state) by incorporating current observations or measurements with its uncertainties.

The observation or measurement is given by

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \tag{2.12}$$

with $\mathbf{z}_k$ as the measurement vector, $\mathbf{H}$ as measurement matrix associating the measurement vector to the state and $\mathbf{v}_k$ as measurement error.

To find the weight of the new information a Kalman gain $\mathbf{K}_k$ is computed as follows

$$\mathbf{K}_k = \hat{\mathbf{P}}_k\mathbf{H}^T \left(\mathbf{H}\hat{\mathbf{P}}_k\mathbf{H}^T + \mathbf{R}\right)^{-1} \quad , \tag{2.13}$$

with $\mathbf{R}$ as observation covariance noise.

Afterwards the state is updated using the Kalman gain $\mathbf{K}_k$ and the measurement vector $\mathbf{z}_k$ with following equation:

$$\mathbf{x}_k = \hat{\mathbf{x}}_k + \mathbf{K}_k \left(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k\right) \tag{2.14}$$

In a final step the error covariance estimate is updated as follows:

$$\mathbf{P}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}\right)\hat{\mathbf{P}}_k \tag{2.15}$$

where $\mathbf{I}$ is the identity matrix.

3 State-of-the-Art in Lane Detection and Tracking

Contents

The aim of this thesis is to implement and evaluate methods for vision-based lane estimation and the ego-vehicle localization relative to the lane markings. Therefore, it is necessary to properly detect all lane markings in a given image. This chapter will introduce methods developed in related work and will give an overview of the key methods of image processing in the field of lane detection and tracking.

3.1 Lane Detection and Tracking in General

Lane detection and tracking is a well researched area in the field of computer vision and plays an important role in the field of Automated Driving (AD) and Advanced Driver Assitstance Systems (ADAS). A detailed awareness of the lanes position and shape is not only important to keep the vehicle on track or to compute the lateral vehicles position, it also enables the estimation of roads geometry. Numerous methods have been developed and proposed in an attempt to reliably detect lanes. Ali G Ulso et al. [8] group lane detection methods into:

- Sensing of magnetic wires embedded into the lanes

- Distance measurement to reference points based on Ground Truth (GT) maps
- Measurement of reflection markers on the lane
- Vision-Based systems

The first three methods usually provide more accurate and reliable results in difficult situations like high speed or bad weather conditions. However, those methods require a road infrastructure in place what is expensive and often not available. Consequently, vision-based systems have the highest potential in the field of *ADAS* and *AD*. Thus, the focus of this thesis and the next chapters focus on vision-based systems.

3.2 State-of-the-Art Pipeline for Lane Detection and Tracking

The traditional vision-based lane detection and tracking is based on the computation of either handcrafted or machine learned features and the estimation of lane markings based on conventional techniques like vanishing point estimation, Hough transform or RANSAC model fitting. As an outcome of Section 3.1, one can generalize the steps for lane detection as follows:

1. Starting with the input stream in form of images or LiDAR point clouds, a pre-processing step is done. The State-of-the-Art (SotA) incorporates the selection of a Region Of Interest (ROI) and Inverse Perspective Mapping (IPM). Both can either be computed in a static or an adaptive way. Furthermore, conventional image processinglike color transformations, blurring and noise reduction is done.

2. The feature segmentation stage incorporates methods to compute features for the lane detection and tracking step. The most common methods in this stage are outlier removal and features extraction. This results in a filtered image, commonly in form of a binary segmentation mask, which segments the frame into background and lane markings. The segmentation task can either be done based on heuristic techniques for image filtering or it can be computed based on machine learning approaches, most commonly by the use of Convolutional Neural Networks (CNNs).

3. After pre-processing the image lanes can be detected based on various techniques such as Hough transform, RANSAC model fitting or vanishing point estimation.

4. By taking advantage of temporal relationships, correct estimated lanes are tracked to improve the overall quality of detection. This is commonly done by the use of a Kalman filter or a particle filter.

5. After the estimation of the necessary information of the lane markings, further processing, like the computation of the vehicles lateral offset, can be done.

Based on this knowledge one can generalize the process of detecting lanes into a pipeline consisting of the four stages *Pre-Processing*, *Segmentation*, *Lane Detection* and *Tracking*, as shown in Fig. 3.1. The steps will be discussed in detail in the upcoming chapters.

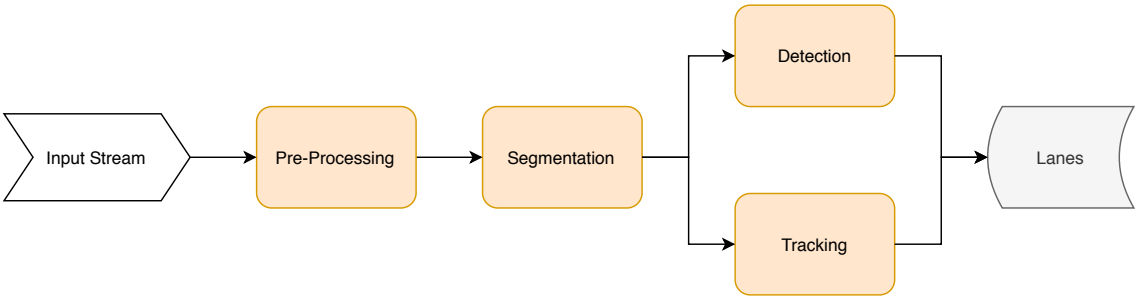


Figure 3.1: Basic pipeline for lane detection and tracking.

3.3 Pre-Processing

Pre-Processing is an important step in lane detection. The goal of this step is to extract only the necessary information of an image for further processing stages. The quality of the pre-processing stage directly correlates with the accuracy of the lane detection. The pre-processing stage again can be split into *image smoothing*, computation of *ROI* and *IPM*.

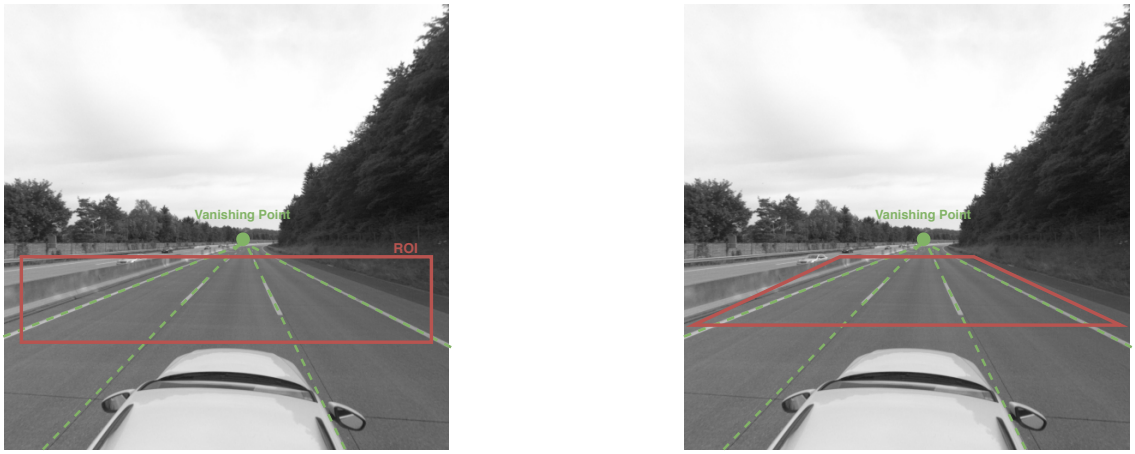
3.3.1 Image Smoothing

Image smoothing is a common approach to reduce noise in images by applying special filters. In [50], [5] and [49] a 2D Gaussian Filter is used to blur the image. Foda and Dawoud in [19] use a Median filter for salt and pepper noise reduction. Morphological operations like dilation and erosion or opening and closing respectively for noise reduction are used in [23] and [60].

3.3.2 Region of Interest (ROI)

Images used for lane detection often do not only cover the road plane, but also the horizon or other irrelevant objects. By defining a suitable region of interest, the computational cost and the error-proneness of lane detection can be reduced drastically. The computations should consider regions which only contain necessary information. There are several ways of finding a suitable *ROI*. The *ROI* either can be defined in a static way or can be computed in an adaptive way. In [42] and [41] vanishing point extraction is used to define the best possible *ROI*. The *ROI* is simply defined as the area below the computed vanishing point.

Due to the perspective effect of image projection, parallel lanes in world coordinates lead to intersections in image plane at the vanishing point. As a result, the form of the lanes is trapezoidal. Pomerleau in [45] extend the solution of [42] and [41] with a perspective model. This leads to a trapezoidal *ROI* which fits the road surface better and contains less irrelevant information.



(a) Normal ROI **(b)** Trapeziodal ROI

Figure 3.2: Definition of the ROI with a) showing normal ROI based on vanishing point and b) showing a trapezoidal ROI considering the perspective effect.

3.3.3 Inverse Perspective Mapping (IPM)

In numerous researches *IPM* is done after the definition of the *ROI*. *IPM* is utilized in [5], [53] and many other studies. With *IPM* the image is transformed from camera perspective to a perspective as if the scenario is seen from top, often called bird-eye perspective. As can be seen in Section 3.3.3, the transformation removes the perspective effect what leads to a change of cross shaped lanes into parallel lanes in the bird-eye view. This enables the consideration of structural correspondences and leads to a better feature extraction and outlier removal.

From a mathematical perspective *IPM* is the transformation of points in 3D Euclidean space $(x, y, z) \in \mathbb{R}^3$ to a 2D space $(u, v) \in \mathbb{R}^2$ described by the Homography $\mathbf{H}$. In [5] the transformation $\mathbf{H}$ is estimated based on simple trigonometry under the assumption that the intrinsic and extrinsic camera parameters are known. A more robust solution is proposed by [43]. They take use of multi modal sensor fusion of the camera with a LiDAR. The results show that their approach is more robust in scenarios with obstacles in the scene. Another approach is shown by Neven et al. [39]. Here Neural Network (NN) is used to predict the parameters of the Homography.

(a) (b)

Figure 3.3: Inverse perspetive mapping with a) as input image and b) as the bird-eye image. The yellow line indicates the area of the image that was used for *IPM*. Image taken from [43].

3.4 Segmentation

This stage is about extracting the necessary features for lane detention in the following stage. Further outlier removal based on spatial correspondences and other special techniques is commonly done in this stage. The result of this stage commonly is a mask that segments lane markings and background. The feature extraction can either be done based on basic image processing techniques like edge detection, color segmentation or adaptive thresholding. Many researches use a combination of those techniques. Furthermore, segmentation can be treated as a semantic segmentation by using deep learning architectures.

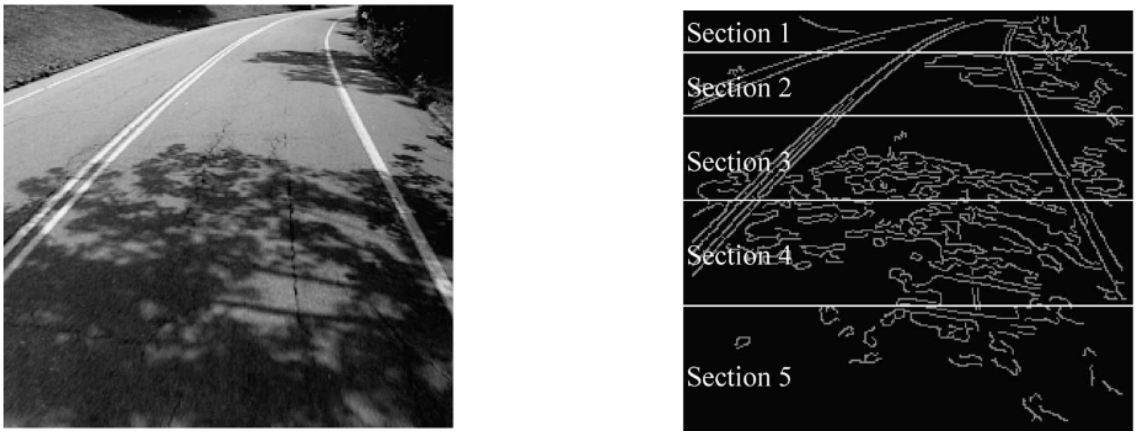
3.4.1 Conventional Segmentation

Conventional methods for image segmentation use the raw image as input and utilize the characteristics of lane markings in images space such as edges, colors or textures for detection.

Edge-Based Segmentation

Because of the sharp transition in contrast and, as a result, the strong gradient from lane markings to road surface, edge-based segmentation is a popular technique for conventional feature extraction of lane markings, especially in the case of solid or dashed lane markings. There exist several edge filtering techniques like Roberts [47], Sobel [52], Prewitt [46] and Canny Edge detection [9], whereby Canny Edge detection leads to a more accurate result because of its non-maxima suppression and hysteresis thresholding. In the early days of vision-based lane detection Schneiderman and Nashman [49] used edge extraction in combination with model fitting to consecutively update the road model. This is done under the assumption that the lane markings are present and well visible. The computations are done in the image plane only without the use of vehicle motion information.

Doshi et al. [17] investigated in several edge detection techniques, such as Canny, Sobel and Roberts. Their evaluation focuses on the accuracy and performance in the context of lane detection. Their results show, that Canny Edge detection and Sobel lead to a better performance than the others. Considering the combination of execution time and accuracy, the Sobel operator is the overall winner.



(a) **(b)**

Figure 3.4: Result of canny edge detection with a) as original image and b) as resulting segmentation mask. Image taken from [59].

However, edge based segmentation works well under fairly good conditions, but still leads to bad results in more complex situations such as images with a in-homogeneous rod structure as can be seen in Section 3.4.1

Color-Based Segmentation

Color-based segmentation helps to segment the lane markings from road surface and other objects with the help of color information. Color thresholding can be done in several color spaces such RGB, HSV, or gray scale. Nieto et al. [41] use gray scale information with Expectation-Maximization to estimate a multi class likelihood model of the road. In [57] the RGB image is converted to HSI color space what enables color thresholding based on saturation and intensity. Tran et al. [56] improved the HSI color model by changing the computation of the intensity value of the RGB image. Their results can be seen in Section 3.4.1.

However, color-based methods potentially lead to bad results in scenarios with complex illumination. Therefore, those methods commonly are combined with other methods such as edge-based segmentation.

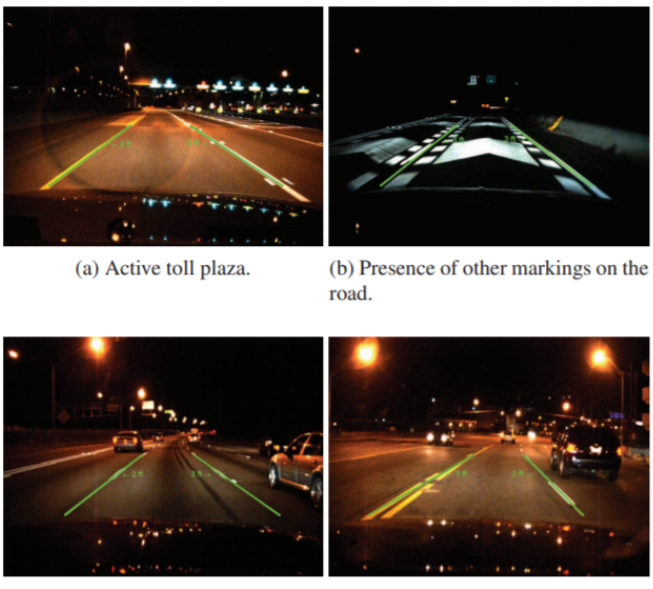
(a) (b) (c)

Figure 3.5: Result of color segmentation proposed in [56] with a) as original image and b) as result based on normal HSI conversion and c) as result based on their approach. Image taken from [56].

Adaptive Thresholding

Similar to color-based segmentation, adaptive threshholding helps to segment the lane markings from road surface and other objects with the help of intensity information.

Borkar et al. [5] used a combination of adaptive thresholding and *IPM* to generate a binary segmentation image which is split into multiple regions - one for each lane. Line centers for sampling are detected based on Hough transform. As can be seen in Fig. 3.6, the results show that their algorithm is reliable, even in difficult conditions like busy streets under low illumination.



(a) Active toll plaza. (b) Presence of other markings on the road.

(c) Busy Highway. (d) Busy city streets.

Figure 3.6: Lane detection results of [5] under variable conditions.

Another approach using adaptive thresholding is proposed by Son et al. [53]. After extracting strong features via thresholding, they use *IPM* to transform the image to a bird eye view. The region below the vanishing point, obtained by a projective model, is selected as a preassigned *ROI*. After the feature extraction, noise is removed and outliers are limited to a minimum in a clustering step. An improved Random Sample Consensus (RANSAC) (Feedback RANSAC) algorithm preventing wrong detections was used to fit a polynomial lane model. To improve the performance, lanes are classified as temporary, extinct or valid. Tracking is only done for lanes classified as valid by the use of a Kalman filter.

3.4.2 Segmentation with Neural Networks

Because conventional methods are unreliable in more complex situations (e.g., dynamic illumination, scenes with many occlusions, etc.) the current state-of-the-art is to use deep learning architectures for image segmentation based on either semantic segmentation or instance segmentation.

Semantic segmentation is the task of assigning a class label to each pixel, whereas multiple objects are treated as one entity. Instance segmentation, on the other hand, is similar to semantic segmentation. Instead of treating multiple objects of the same class as one entity, instance segmentation treats those objects as individual ones.

The current state-of-the-art is to use networks that build up on a *CNN* and train it with pixel-wise annotated images. Various types of *CNNs* can be used for the segmentation task, whereby the architecture of the networks is quite similar.

LaneNet

Neven et al. [39] proposed LaneNet, a modification of the encoder-decoder network ENet for end-to-end instance segmentation, where each lane is handled and labeled as its own instance. Fig. 3.7 gives a general overview of the system. Two branches using the same encoder stage of the network were developed. A segmentation network outputs a binary segmentation map which separates background and lane markings. Furthermore, an instance segmentation branch is used to separate the lanes by using a special clustering loss function proposed by Brabandere et al. [6]. Afterwards, clustering is done to generate a binary with each lane labeled as an own instance. Since LaneNet outputs a binary segmentation mask, curve fitting in combination with *IPM* is done. By using an additional neural network (H-Net) which predicts the parameters of a perspective transform with 6 Degrees of Freedom (DOF), the transformation matrix for *IPM* is computed for each frame.

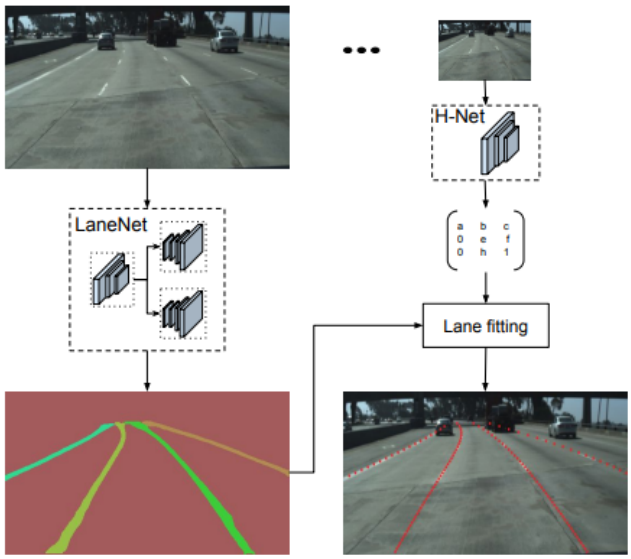


Figure 3.7: System overview of [6]. Lane markings are labelled as an output of LaneNet. The lane pixels are transformed to bird eye view by the transformation matrix outputted by H-Net. In a final step lane fitting is done.

DeepLabv3+

Upsampling and deconvolutions lead to a spatial loss in the decoder stage of a network. As a result, inaccurate object boundaries can be seen. Furthermore, scaling is of high importance. If a network was only trained with small objects, it might performs bad dealing with big objects and via versa. Consequently, the front-end network in encoder-decoder structures is of great importance.

To address this, Deeplab, an open source semantic segmentation network from Google developed by Chen et al. [11], was developed. The architecture is based on an encoder-decoder structure, with several available networks for feature extraction such as MobileNet [48], ResNet [22] or Exception [14].

As already mentioned, due to the loss of spatial information in pooling layers, segmentation networks lack in accuracy at object boundaries. There exist several methods to minimize this problem, as shown in Fig. 2.6. To cope with scale in-variance and the loss of spatial information, Atrous Spatial Pyramid Pooling (ASPP), a combination of atrous convolutions and spatial pyramid pooling, was introduced with DeepLab. With the concept of atrous convolution, the kernels Field of View (FoV) is adjusted to gather large scale features, while using the same number of parameters and retaining the original input resolution. From a mathematical perspective the atrous convolution is defined as

$$y[i] = \sum_k x[i + rk]w[k] \quad . \tag{3.1}$$

with the atrous rate r defining the spacing of the filter kernel, $\mathbf{x}$ as the input and $\mathbf{w}$ as the kernel. Fig. 3.8 shows atrous convolutions with a 3x3 kernel and different rates, whereas a rate $r = 1$ corresponds to a normal convolution.

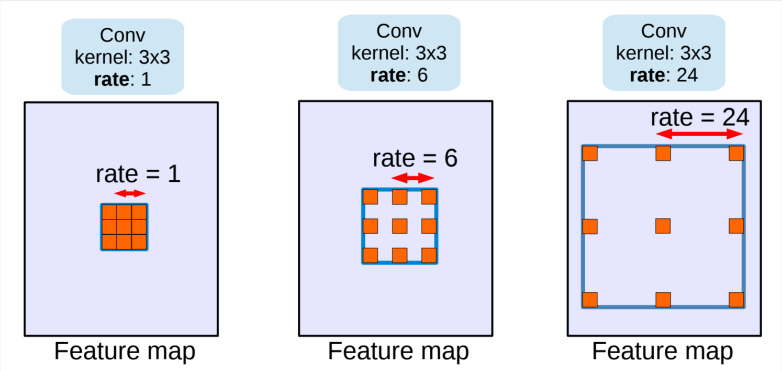


Figure 3.8: Comparison of atrous convolutions with a 3x3 kernel size and different rates. Image taken from [12].

The combination of atrous convolution with different rates enables feature extraction on different scales without the loss of spatial information and is referred as Atrous Spatial Pyramid Pooling (ASPP). Furthermore, DeepLab uses a convolution technique called *atrous depth-wise convolution*. With that, the computational demands are significantly reduced while the performance is not effected. This is done by breaking the convolution down to an atrous depth-wise convolution followed by a point-wise convolution. DeepLabv3 is based on a modified model of ResNet-101 [22] with atrous convolutions, *ASPP* and bilinear upsampling in the encoder module.

Deeplabv3+ extends Deeplabv3 by a encoder-decoder structure and *atrous depth-wise convolution* in the *ASPP* and decoder stage. The architecture of DeepLabv3+ is shown in Fig. 3.9. For detailed information see [13].

Spatial Convolutional Neural Network (SCNN)

Because lane markings cover only a small part of an image and often are occluded or partly not painted, the optimization for semantic lane segmentation is a challenge. Typically *CNNs* are based on a convolutional layer-by-layer stacked structure. Pan et al. [44] developed Spatial Convolutional Neural Network (SCNN), a *CNN* that exploits spatial information in one layer by performing sequential message passing. With SCNN they replaced the typical layer-by-layer convolutions with slice-by-slice convolutions which enables row-wise and column-wise message passing between pixels in one layer. In Fig. 3.10 you can see an illustration of the architecture of *SCNN*. The forward computation of an input $\mathbf{X}$ with the Kernel $\mathbf{K}$ is defined as

$$\mathbf{X'}_{i,j,k} = \mathbf{X}_{i,j,k} + f \sum_{m} \sum_{n} \mathbf{X}_{m,j-1,k+n-1} \times \mathbf{K}_{m,i,n} \tag{3.2}$$

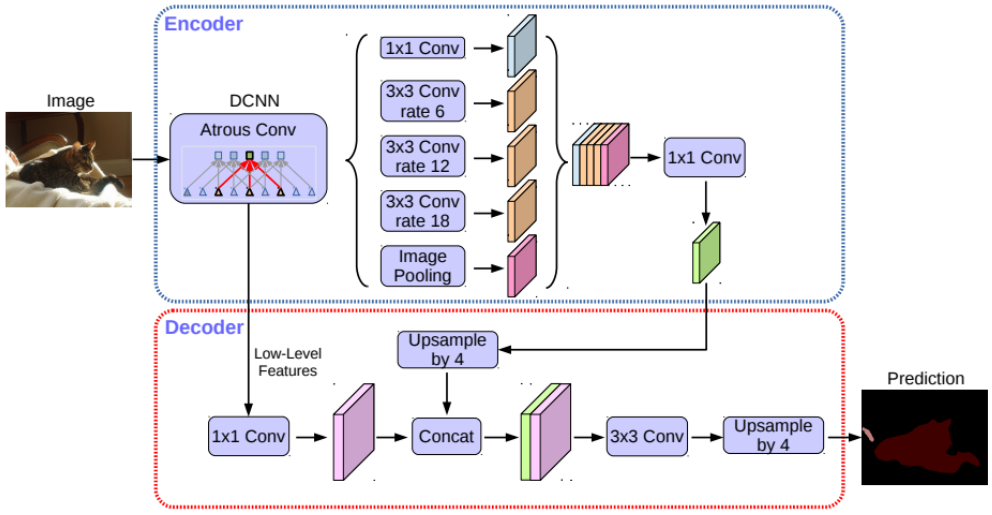


Figure 3.9: Deeplabv3+ encoder-decoder architecture as an extension of DeepLab3. Image taken from [13].

with f as an activation function. $\mathbf{X}_{i,j,k}$ is the value of the input at channel i, row j, column k and $\mathbf{X}_{i,j,k}$ is the weight between the element in channel j of the current slice and the element in channel i with an offset of k columns of the last slice. Note that with *SCNN* the computation is done in four different directions(up, down, left, right) and the kernel weights are transmitted across all slices.

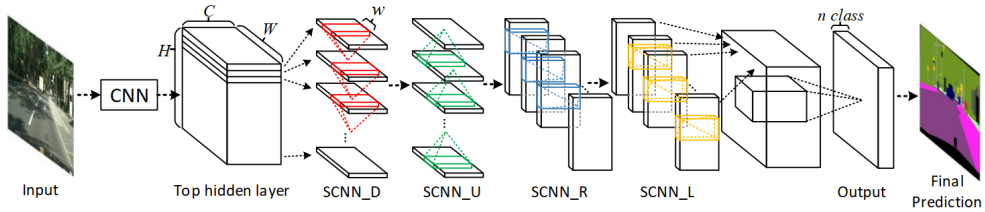


Figure 3.10: Architecture of SCNN with convolutions in the four different directions. Image taken from [44].

Instead of performing a pixel-wise semantic segmentation, *SCNN* outputs a probability mask, indicating the probability of the center of the lane marking at the corresponding pixel. The results of *SCNN* were tested on CityScapes dataset[1] [15] and show, that SCNN better learns spatial relationships. In comparison with ReNet [58], MRFNet[30] and ResNet-101 [22], it better captures long continuous structures leading to less unconnected parts in the segmentation mask, as you can see in Fig. 3.11.

[1]Freely available data set for academic research: `https://www.cityscapes-dataset.com/`

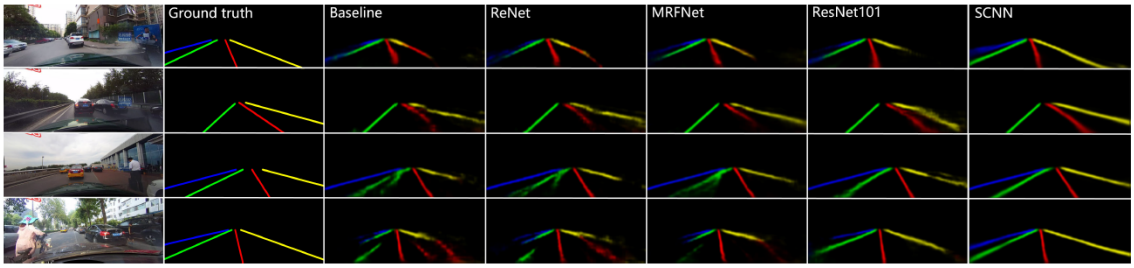
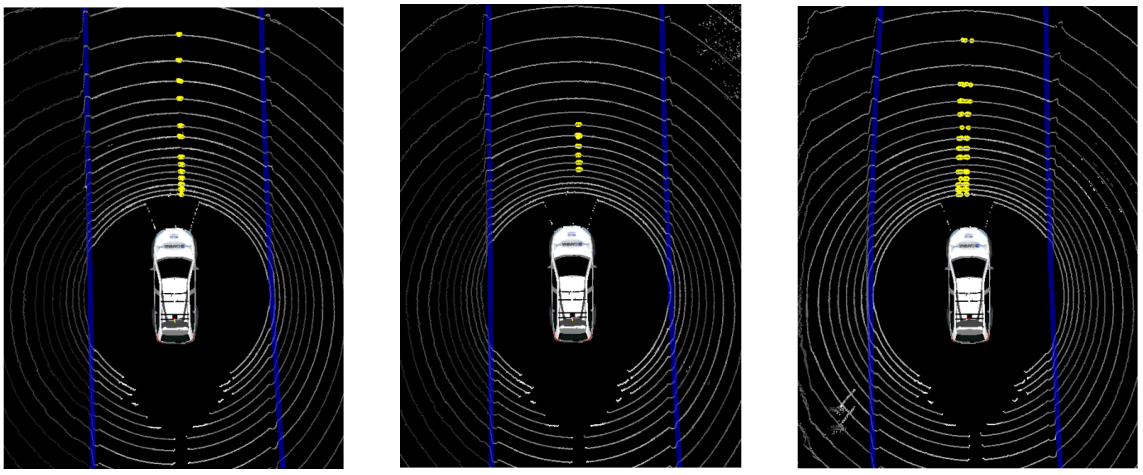


Figure 3.11: Comparison of SCNN with results of other networks. Image taken from [44]

3.4.3 Point Cloud based Segmentation

There is research in which 3D sensors such as LiDARs are used to segment the *ROI* [21, 27, 35]. Lane marker detection is done based thresholding the intensity information of the whole LiDAR scan to segment lane markings and road surface. E.g. in [27] they used Otsu`s thresholding method. Due to the lack of intensity, often GPS data is used to overlay or combine multiple subsequent LiDAR scans. In Fig. 3.12 one can see the output of road marking detection proposed in [27]. They show that their system is capable of detecting any kind of road marking. For evaluation they used the Monte Carlo Localisation algorithm to treat the road marking detection as vehicle localisation problem. The drawback of using LiDAR only is the lack in density and reliability especially in situations with occlusions.



(a) continuous line **(b)** dashed line **(c)** double line

Figure 3.12: Lane marking detection based on LiDAR intensity thresholding. Yellow points indicate lane markings whereas blue lines correspond to curbs. Image taken from [27].

3.5 Lane Detection

In the lane detection stage, the necessary information like position, curvature or radius of the lane are computed based on the lane features extracted in the previous step. The most common lane detection methods found in research are the Hough transform and *RANSAC* model fitting.

3.5.1 Hough Transform

As discussed in Section 2.6, Hough transform is a well known and popular technique to detect primitive shapes that are describable in a mathematical form. Since, lane markings can be approximated as straight lines for short distances, Hough transform can be used to detect lines in a image. Many researches utilize this technique to detect the position of lane markings. Borkar et al. [5] use the Hough transform to find the line base position for further detailed lane sampling, as can be seen in Fig. 3.13. One of the drawbacks of Hough transform is, that the performance is bad for curved lanes. Therefore, it is often combined with other techniques. Wang et al. [59] e.g., subdivide their image into multiple regions and apply Hough transform for each region to obtain an initial estimate for model fitting.

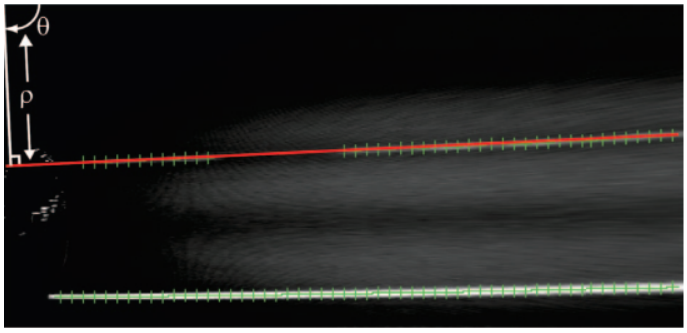


Figure 3.13: Candidate points sampled based on line parameterized with θ and ρ found with Hough transform. Image taken from [5].

3.5.2 RANSAC Model Fitting

Another popular approach for lane detetion is RANSAC lane fitting. The theory of RANSAC is discussed in Section 2.3. With RANSAC model fitting a sample of feature points extracted in the segmentation step is taken to form the hypothesis of a lane or lane marking based on a specific model such as a spline or a quadratic model. The different hypothesis are tested against the other samples and the hypothesis with the highest number of inliers is selected. A huge advantage of RANSAC model fitting is, that the line detection is not restricted to straight lines. It can cope with variable shapes based on the choice of the model.

Borkar et al. [5] extend their low resolution Hough with RANSAC line fitting of a straight line based on the parameters ρ and θ extracted by the Hough. In [53] an improved RANSAC algorithm is used to fit a quadratic line model to the feature points. They use a second scoring function for RANSAC based on the feedback of the previous curvature and lane edges.

3.6 Tracking

Many researches use a tracking stage to cope with outliers and to predict the parameters of a lane marking. Due to the utilization of temporal relationships, tracking shows a significant improvement in accuracy and in case of a lane not being detected in short-term, the prediction of the tracking stage can be used for processing. The most common tracking algorithms used in researches are Kalman Filter and Participle Filter.

3.6.1 Kalman Filter

Due to the computational advantages and its easiness in implementation, Kalman filter is a widely used method for tracking. The theory of Kalman filter is dicussed in Section 2.7. A common approach is to directly track the parameters computed by the Hough transform, as can be seen in [5]. The state transition matrix and observation vector are composed of the parameters ρ and θ and their derivatives. Son et al. [53] use the parameters of the quadratic model and their derivatives for the state transition matrix and the current parameters of the quadratic model for measurement update. Due to its linearity, the output of the Kalman Filter can be vulnerable to complex situations such as outliers or missing feature points. Therefore, Tian et al. [55] use an extended Kalman Filter (EKF), which is specially designed to cope with non-linear dynamic systems. Their results show, that their algorithm adopts to many driving scenarios even in complex environments.

3.6.2 Particle Filter

Particle Filter belong to the sequential Monte Carlo methods and generalize the methods of the Kalman Filter. With this filter a set of samples, referred as particles, is propagated over time to describe the probability density distribution of any state-space model. In comparison with Kalman Filter, Particle filters are less vulnerable to noise and outliers. There is still a trade off between computation costs and accuracy since the number of particles directly influences the number of calculations.

Many researches apply Particle filter in their lane detection approach. Berriel et al. [3] use the Particle Filter in combination with cubic spline modeling. The result can be seen in Fig. 3.14. The yellow points indicate all the particles and the red line represent the output based on the best particles.

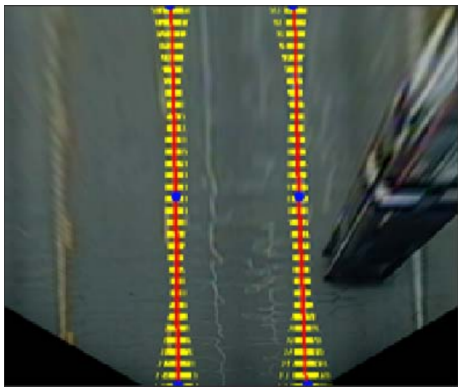


Figure 3.14: Lane detection and tracking result based on particle filter in combination with cubic spline modeling. The red lines represent the splines and the yellow dots represent the particles. Image taken from [3].

3.7 Discussion

In this chapter the current state-of-the-art for lane detection and tracking was discussed. It was shown that there are numerous approaches for vision based lane detection.

Due to in depth research, a general flow chart for lane detection and tracking could be exposed. As illustrated in Fig. 3.1, the general flow includes the four main processing steps: Pre-Processing, Segmentation, Detection and Tracking. In many researches the image is transformed to a bird eye perspective called *IPM*. Due to te removal of the perspective effect lane markings appear parallel. There are conventional techniques using traditional image filtering techniques like edge-based extraction, adaptive thresholding or color based segmentation to segment the lane markings from background. In complex situations the conventional techniques often are unreliable. Therefore, many researches use machine learning-based approaches. There are solutions which take over the segmentation task only, whereas other methods also treat the detection task as a machine learning problem and directly output the necessary lane parameters. For tracking most commonly Kalman filters or particle filters are used, but the number of researches using Kalman filter dominates, due to its easiness in implementation and its better computing performance.

Based on this in-depth research, three different algorithms following a common processing pipeline for lane detection have been implemented. The algorithms primary differ in the segmentation stage. One algorithms is based on a combination of conventional segmentation techniques such as color thresholding and adaptive thresholding and the other two algorithms use deep learning models for semantic segmentation. Furthermore, LiDAR point clouds are used to improve the computation of the *ROI* and to remove potential outliers in the segmentation mask.

Due to the promising results of Borkar et al. [5], Hough transform in combination with RANSAC line fitting is used for lane detection. A quadratic model was chosen to describe the individual lane markings. A Kalman filter taking the parameters of the

quadratic model as input is used for tracking. The concepts will be discussed in detail in the following chapter.

4 Concepts

Contents

In the previous chapter an in depth-research on related work and state-of-the-art concepts in lane detection and tracking was done. Based on that, three different algorithms following the same processing pipeline for lane detection have been implemented. The algorithms primary differ in the segmentation stage. One algorithm is based on a combination of conventional segmentation techniques, such as color thresholding and adaptive thresholding, and the other two algorithms use deep learning models for lane segmentation. Furthermore, LiDAR point clouds are used to improve the computation of the Region Of Interest (ROI) and to remove potential outliers in the segmentation mask.

Due to the promising results of Borkar et al. [5], Hough transform in combination with RANSAC line fitting is used for lane detection. A quadratic model was chosen to describe the individual lane markings and a Kalman filter is used for tracking. The concepts will be discussed in detail in the following sections.

4.1 Overview

In order to compare conventional lane detection algorithms with machine learning-based lane detection algorithms and to evaluate the advantages and disadvantages of those, three different lane detection and tracking algorithms have been implemented and tested. Those algorithms include two deep learning based approaches and one conventional approach. The lane detection, tracking and post-processing is the same for all three algorithms. this gives an ideal base for the comparison of the different segmentation concepts. Furthermore, LiDAR point clouds are used to improve the overall detection rate and accuracy.
Fig. 4.1 illustrates the general processing pipeline (highlighted in blue) including all the methods(yellow) that will be discussed in this chapter. In this thesis the traditional pipeline is extend by a *Post-Processing* stage, in which the data recording and the vehicles ego-position is estimated. Therefore, the pipeline of the algorithms can be split into the five main stages *Pre-Processing, Segmentation, Detection, Tracking* and *Post-Processing.*

Again, the three algorithms follow the same processing pipeline and use the same concepts for lane detection, tracking and post-processing. The main difference of the algorithms is in the segmentation stage and the corresponding pre-processing stage, marked as red. As can be seen in Fig. 4.1 the different stages are:

1. *Pre-Processing*: The pre-processing stage consists of up to four sub tasks: RANSAC Plane Fitting, Inverse Perspective Mapping (IPM), *ROI* extraction and Image Smoothing. The different algorithms either implement all the sub-tasks or only utilize individual sub-tasks.

2. *Segmentation*: The segmentation stage is the key stage where the algorithms differ. The different concepts for segmentation are:

   (a) *Conventional Segmentation* utilizing a combination of conventional image filtering techniques.

   (b) *Semantic Segmentation* using DeepLabv3+

   (c) *Instance Segmentation* using Spatial Convolutional Neural Network (SCNN)

3. *Lane Detection and Tracking*: For lane marking detection and tracking Hough Transform and RANSAC Line Fitting in combination with an Extended Kalman Filter for tracking have been implemented.

4. *Post-Processing*: Based on the estimated position of the lane markings, the ego-lane and the vehicles relative position within the ego-lane can be estimated and recorded in a post-processing stage.

The next sections are devoted to discuss the individual stages in detail. To keep a clear structure, this chapter describes one pipeline. The sections are arranged in a way that they start with Pre-Processing and end up with the Post-Processing stage. Since the algorithms

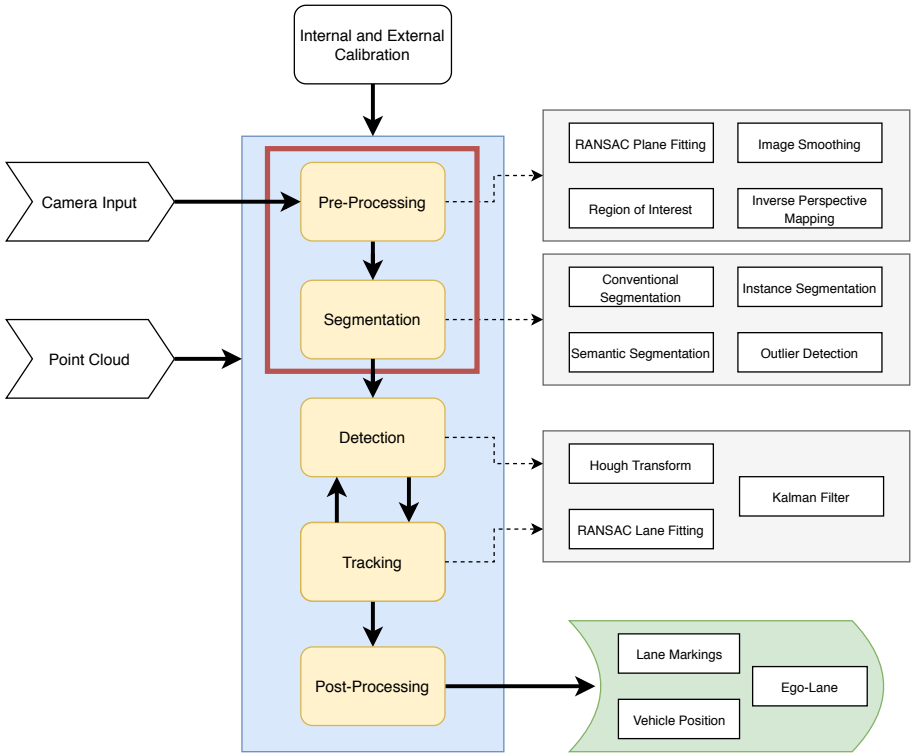


Figure 4.1: Processing pipeline of the proposed algorithms.

mainly differ in the segmentation stage, the section discussing the *segmentation* is split into three subsections in which the different concepts of the individual algorithms are described in detail. During the implementation several variations with different parameters and different combinations were implemented and tested, but in this chapter only the best results are discussed.

4.2 Image Formation

Before we start to discuss the concepts in detail, some important remarks about image formation have to be done. First of all, even tough camera calibration is an important step in computer vision, it will not be discussed in detail. The images used in this thesis either are already undistorted or the parameters for the undistortion of the images are known. Therefore, it is expected that the internal calibration matrix $\mathbf{K}$ is available. Some concepts implemented in this thesis utilize LiDAR data. Therefore, the point clouds for the corresponding frames have to be available and the external calibration containing the rotation matrix $\mathbf{R}$, the camera position $\mathbf{C}$ and the translation $\mathbf{t}$ have to be known.

4.3 Pre-Processing

As stated in Section 3.3 the pre-processing step is to extract only necessary information of the input for further processing. That commonly includes the three sub tasks *ROI* selection, *IPM* and gray scaling. Additionally, RANSAC Plane Fitting (RPF), a pre-requirement for further concepts used in this thesi, is introduced in this thesis. Depending on the concepts for further processing, the proposed methods for lane detection utilize a combination of the tasks.

4.3.1 RANSAC Plane Fitting

Before discussing the individual pre-processing concepts of the algorithms, RANSAC road plane fitting must be addressed. The process of lane detection can be improved by determining the position and orientation of the road with respect to the camera or LiDAR. Among others, it enables road boundary extraction and the computation of the Homography $\mathbf{H}$ for *IPM*. For this purpose, RANSAC plane fitting based on LiDAR point clouds is used. Depending on the use case, RANSAC Road Plane Fitting is done frame-wise or only once during a calibration procedure to estimate the parameters for *IPM*. The theory of RANSAC was already discussed in Section 2.3, therefore only the steps of finding the equation of the plane and the transformation of the LiDAR points is discussed detail.

Camera-LiDAR Projection

In a first step, the 3D LiDAR points have to be transformed to camera coordinate system by using the rotation matrix $\mathbf{R}$ and the translation $\mathbf{t}$. The theory about image formation was discussed in Section 2.2. Fig. 4.2 illustrates the relations between a 3D LiDAR point $\mathbf{X}$ in world coordinates and the point $\mathbf{x_i}$ in image plane or the 3D point $\mathbf{X_C}$ in camera coordinate system. First, the 3D LiDAR points have to be transformed to 3D points in the local camera coordinate system. Afterwards the 3D point $\mathbf{X_C}$ is projected to the image plane using the internal calibration matrix $\mathbf{K}$. The LiDAR points, the external calibration parameters $\mathbf{R}$ and $\mathbf{t}$ to transform those points to a local camera coordinate system and the calibration matrix $\mathbf{K}$ to project those points to the image plane are provided by the Institute of Computer Graphics and Vision (ICG) of the TU Graz.

To make this algorithm work, one can assume that close LiDAR points, directly in front of the car, belong to the road. Therefore, only a small *ROI* in front of the car for RANSAC plane fitting was selected. Furthermore, one can assume that the road covered by this *ROI* is flat. As a consequence, the point cloud can be cropped in lateral and longitudinal direction to reduce its complexity and the computational costs. The residual points are used for RANSAC plane fitting.

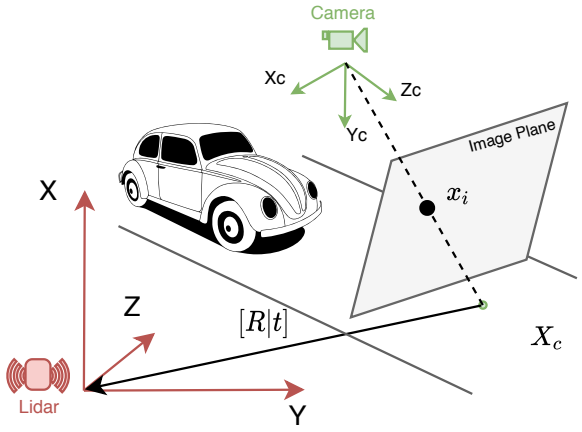


Figure 4.2: Camer-LiDAR Formation.

Plane Fitting

Assuming one has a set of 3D points and wants to fit a plane to them, one can estimate the plane parameters as described in this section. Mathematically a plane can be described with a distance d and the planes normal vector $\mathbf{n} = [a, b, c]^T$ that for a point $\mathbf{p} = [x, y, z]^T$ on this plane $\mathbf{np} + d = 0$ is valid. This can be rewritten as

$$ax + by + cz + d = 0 \quad . \tag{4.1}$$

The equation above using four values is over-determined for points in 3D space. Therefore, one can eliminate one component by assigning $c = 1$. One can rewrite the equation to

$$ax + by + d = -z \quad . \tag{4.2}$$

Incorporating the LiDAR points one can solve the least squares solution of $\mathbf{Ax} = \mathbf{B}$ or in a more detailed form

$$\begin{bmatrix} x0 & y0 & 1 \\ x1 & y1 & 1 \\ & \cdots & \\ xn & yn & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ d \end{bmatrix} = \begin{bmatrix} -z0 \\ -z1 \\ \cdots \\ -zn \end{bmatrix} \quad . \tag{4.3}$$

Since there are only three unknowns and there are usually more than three 3D points in a LiDAR scan, the system is over-determined. The equation can be solved by using the

left pseudo inverse $\left(\mathbf{A^T A}\right)^{-1}\mathbf{A^T}$ and is defined as

$$\begin{bmatrix} a \\ b \\ d \end{bmatrix} = \left(\mathbf{A^T A}\right)^{-1}\mathbf{A^T B} \quad . \tag{4.4}$$

The plane can be described by its normal vector $\mathbf{n} = [a, b, -1]^T$ and the distance d.

4.3.2 Region of Interest (ROI)

As discussed in Section 3.3.2, the selection of a suitable *ROI* reduces the computational costs and improves the reliability of further lane detection steps in terms of outliers. In the course of this thesis a static as well as an adaptive *ROI* is used. For both methods the LiDAR point clouds are used.

Static Region of Interest

With the static method a *ROI* in front of the car in 3D world coordinates is selected. For this purpose, the result of road plane fitting described in Section 4.3.1 is used. This leads to a trapezoidal form in image plane what enhances the common approach of a rectangular *ROI*. illustrates the computation of the *ROI*.

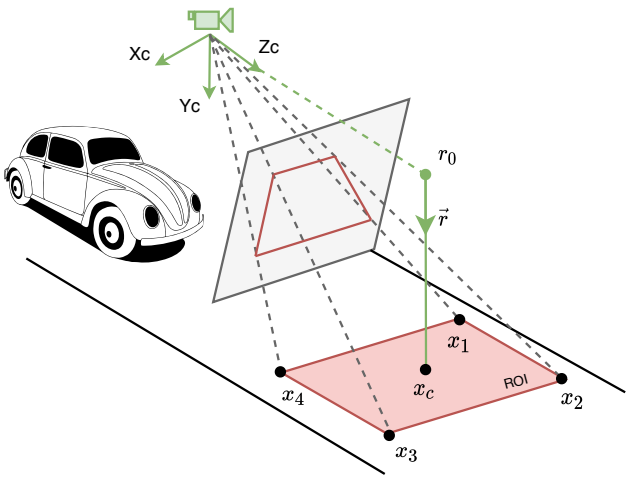


Figure 4.3: Definition of Static Region of Interest.

The static *ROI* is selected by, first, defining the longitudinal offset to the center of the *ROI* on the z-axis of the camera coordinate system. Based on this offset, the coordinates of a 3D point denoted as $\mathbf{r_0}$ can be estimated. Starting from that point, a ray is defined in the direction along the Y-axis in camera coordinate system. The ray can be described

by the vector equation

$$\mathbf{x_c} = \mathbf{r_0} + \mathbf{r}t \quad t \in \mathbb{R} \quad , \tag{4.5}$$

with $\mathbf{r_0}$ as starting point on this ray, $\mathbf{r}$ as direction and t as scalar. The 3D center $\mathbf{X_c} \in \mathbb{R}^3$ of the *ROI* in camera coordinates can be computed based on the ray-plane intersection between the previously defined ray and the road plane estimated in Section 4.3.1. Since the dot product of two perpendicular vectors is 0, one can write

$$(\mathbf{x_c} - \mathbf{x_p}) \cdot \mathbf{n} = 0 \tag{4.6}$$

with $\mathbf{x_c}$ as point of intersection, $\mathbf{x_p}$ as arbitrary point on the plane and $\mathbf{n}$ as plane normal.

The point of intersection can be computed by first substituting the ray equation into Eq. (4.6):

$$\mathbf{n} \cdot (\mathbf{r_0} + \mathbf{r}t - \mathbf{x_p}) = 0 \quad . \tag{4.7}$$

Solving for t gives

$$t = \frac{\mathbf{n} \cdot (\mathbf{x_p} - \mathbf{r_0})}{\mathbf{n} \cdot \mathbf{r}} \quad . \tag{4.8}$$

The point of intersection $\mathbf{x_c}$ is computed by inserting t into Eq. (4.5).

Knowing the plane parameters and the point of intersection, the area spanned by the *ROI* can be defined and the corresponding image coordinates can be computed. This is done by specifying four 3D points around the center point $\mathbf{x_c}$ and projecting those points to 2D image points using the calibration matrix $\mathbf{K}$ as described in Section 2.2.

Adaptive Region of Interest

Even if a static *ROI* is specified, it often includes unnecessary information such as highway guardrails or crash barriers. Especially in scenarios driving on an outer lane, the *ROI* often just partly covers the road. Some approaches are vulnerable in terms of feature extraction and segmentation of such objects. Therefore an adaptive *ROI* on top of the static one is used in this thesis.

The computation of the adaptive *ROI* uses the given LiDAR point clouds to additionally extract the road boundary out of it. Therefore, again the road plane parameters computed in Section 4.3.1 and the estimated static *ROI* are used. At first, the LiDAR points are cropped in order to remove the points not belonging to the static *ROI*. Afterwards, 3D points not belonging to the road plane are removed. This is done by removing points with a L2 norm higher than a given threshold to the plane. Based on that result the convex hull spanned by the remaining points is computed. Finally, the width of the larger static *ROI* is reduced to fit the width of the convex hull previously computed. Fig. 4.4 shows the result after the computation of the adaptive *ROI*. The width of the initial static *ROI* was reduced to fit the road boundary.

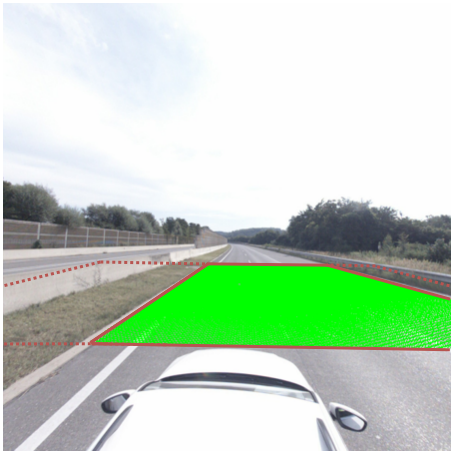


Figure 4.4: Visualization of adaptive *ROI*. The dashed line represents the large static *ROI* and the continuous line represents the adaptive *ROI*. The green points represent the projected LiDAR points.

4.3.3 Inverse Perspective Mapping

As already discussed, *IPM* is a common technique to transform the image to a perspective as if seen from top, called bird eye-perspective. Because the perspective effect was removed, lanes appear parallel and have a constant lane width. *IPM* drastically improves further steps like lane detection and outlier removal.

IPM is a mapping of the original image points $I(u, v)$ to the corresponding point in the bird-eye view $I(x, y)$. The transformation can be described as

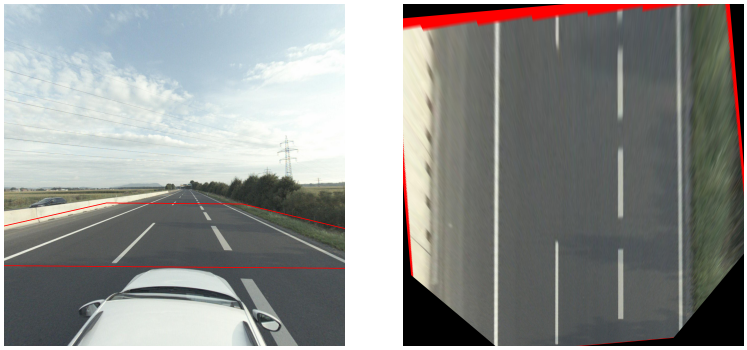
$$\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} = \mathbf{H} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad , \tag{4.9}$$

with $\mathbf{H}$ as homography or transformation matrix that describes the relation of two images visualizing the same surface.

The homography is obtained by solving Eq. (4.9) for four known point correspondences. Therefore, the points of the static roi are selected in Section 4.3.2 and projected to the image plane. The real world relations of those points are known, so one can choose the corresponding points in the bird eye image to compute the homography. The geometric relations of the corresponding points have to be preserved to get a valid result. The result of *IPM* is shown in Fig. 4.5.

Gray Scale Conversion

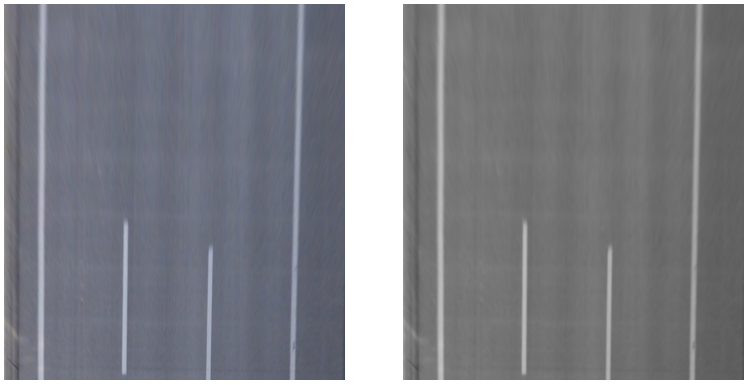
With gray scaling the image is converted from RGB color format to a gray scale format, where each pixel is represented by only one value - its intensity. This means bright pixels



(a) Original Image **(b)** Result of *IPM*

Figure 4.5: Example of *IPM*

receive a high intensity value, whereas dark pixels receive a low value, as can be seen in Fig. 4.6.



(a) Original image **(b)** Gray scaled image

Figure 4.6: Example of gray scaled image.

By considering the fact that lane markings are brighter than the road, one can take advantage out of that in terms of lane segmentation in further steps. With gray scaling the complexity of differentiating white or yellow lane markings from the background is reduced. Instead of considering all three color channels in RGB color format the problem is reduced to only one channel or value. The conversion of a RGB pixel to its gray value is described by

$$Y = 0.299R + 0.587G + 0.114B \quad . \tag{4.10}$$

4.4 Segmentation

This is the key stage in which the algorithms for lane detection and tracking differ. As discussed in Chapter 3 the segmentation step is about extracting the road markings from the scene. This section describes in detail the three different segmentation concepts that are were implemented this thesis.

4.4.1 Conventional Segmentation

As the name indicates, the *Conventional Segmentation* is built up on traditional image filtering techniques. Because lane markings often are worn out, what can lead to unclear edges, the conventional segmentation in this thesis is not based on edge based techniques like Sobel or Canny Edge. Instead, region based filtering techniques, which are based on the assumption that pixels on lane markings are brighter and different in color than the road, are used. Therefore, a combination of Color threshholding, adaptive threshholding based on the method of Otsu and a second derivative Gaussian filter is used to extract the lanes from the image. Fig. 4.7 shows the pipeline for segmentation. Starting with either the IPM image or the grayscaled IPM image computed in Section 4.3, the corresponding filters are applied and the results are combined. After an outlier detection step the final segmentation mask, which is used as input for the lane detection stage, is generated.

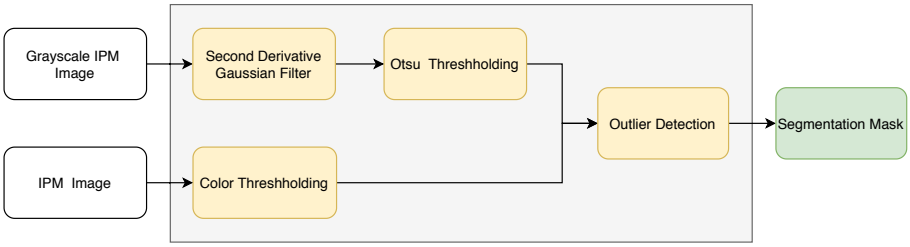


Figure 4.7: Conventional Segmentation Pipeline.

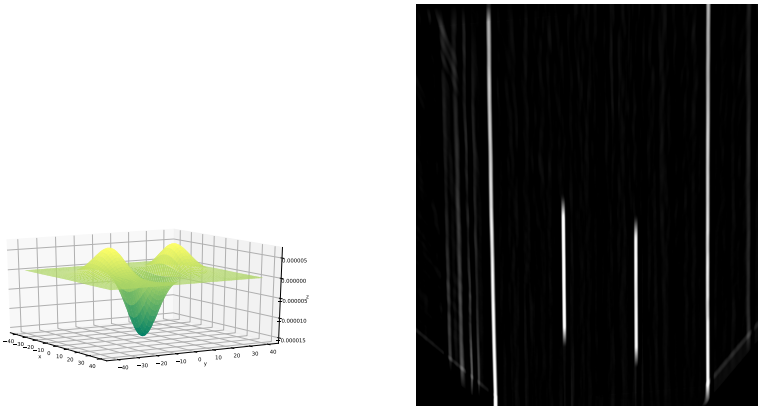
Second Derivative Gaussian Filter

Because of the top view perspective of the IPM image, lanes are parallel and show the same width over distance. Assuming that lane markings have a high intensity surrounded by dark regions a second derivative Gaussian filter in y-axis is applied to the gray scaled IPM image. The second derivative of the Gaussian function is described as

$$\frac{\partial^2 G\left(x,y,\sigma\right)}{\partial^2 y} = \left(-1+\frac{y^2}{\sigma^2}\right)\frac{e^{\frac{x^2+y^2}{2\sigma^2}}}{2\pi\sigma^4} \quad . \tag{4.11}$$

In the filtered image, bright pixels, such as pixels on lane markings, show a high negative intensity value. To reduce the number of outliers and to get the best possible response of

lane markings, the variance σ is set to the average width of highway lane markings. The brighter the pixel, the higher the negative value or response. The values of the filtered image are normalized to a range from 0 to 255 and the lanes can then be extracted by applying a suitable threshold. Fig. 4.8 shows an illustration of the Gaussian kernel and the resulting image after convolution.



(a) Second Derivative Gaussian Kernel **(b)** Filtered image

Figure 4.8: Second derivative Gaussian kernel and the filtered image.

Otsu Threshholding

In case of worn-out lanes, the values after filtering could be similar to the responses of other obstacles such as guardrails. Therefore an adaptive threshold is used to exclude outliers while still being able to detect worn-out lanes. Otsu `s method is used to perform automatic treshholding of gray scaled images. With the algorithm a threshold value is found that minimizes the weighted class variance to split up a gray scaled image into two classes. This is done by iterating through all possible threshold values within a give range and finding the value leading to a minimum of the sum of the two classes. Fig. 4.9 shows the result after thresholding the filtered image with Otsu `s method including the histogram and the estimated threshold value.

Color Thresholding

Guardrails sometimes lead to outliers in segmentation methods based on intensity values only. Therefore, another popular filtering approach based on the color values is introduced in this thesis. Since lane markings primary are white, the color information of the RGB image can be used to improve the performance of segmentation. For that, the RGB values of the image are converted to HSL colour space. As can be seen in Fig. 4.10, a color in HSL space can be described based on the three values *hue, saturation and lightness* and white values can be separated based on the lightness value.

(a) Filtered image with second derivative gaussian kernel

(b) Thresholded image with Otsu `s method

(c) Histogram

Figure 4.9: Otsu threshholding with input image, resulting image and histogram (from left to right). The dashed line in the histogram represents the estimated threshold value.

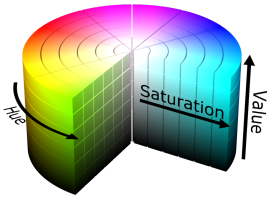


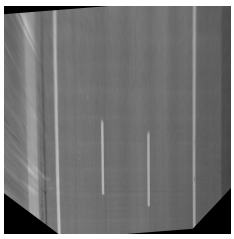
Figure 4.10: HSL colour space. Image taken from [10].

The results of the specific filtering steps and the final segmentation mask after conventional image segmentation can be see in Fig. 4.11.

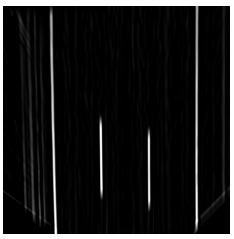
Outlier Detection

After the traditional segmentation steps, the segmentation mask may still contains outliers, e.g. caused by other cars or bad road structures. For this reason a concept for outlier detection was introduced in this thesis. The outlier detection consists of two consecutive steps. First, a structural analysis is done. By taking a segmentation mask as shown in Fig. 4.11 as input, the connected components in the segmentation mask are computed and the width and area of those components is estimated. Assuming that lane markings must have a specific width and must cover a minimum area, small outliers can be filtered by comparing the area and the width of the components with a specific threshold.

In the second step, the LiDAR point cloud is used to evaluate the residual components. Therefore, the road plane estimated in Section 4.3.1 and the points belonging to that plane are used. First, the points on the road plane have to be projected to the image plane and in a second step projected to the IPM image. To classify the individual components as outliers or not, the amount of LiDAR points in the area covered by the individual

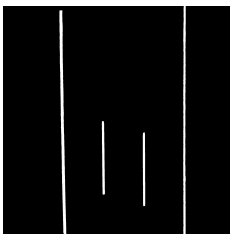


(a) Input image



(b) Filtered image with second derivative gaussian kernel

(c) Thresholded image with Otsu `s method

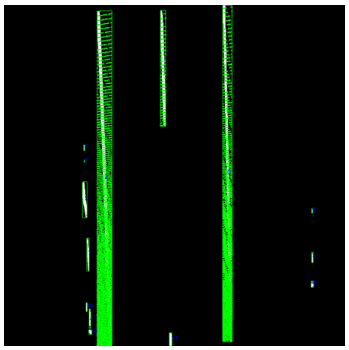


(d) Color thresholding

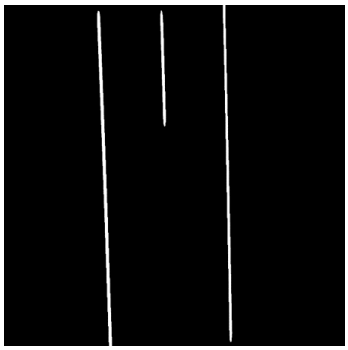
(e) Segmentation Mask

Figure 4.11: Results of conventional image segmentation of lane markings.

components is examined. Components not belonging to the road will be filtered out as can be seen in Fig. 4.12.



(a) Initial Segmentation Maks



(b) Filtered Result

Figure 4.12: Example of outlier detection. The individual components are highlighted in green borders and the corresponding LiDAR points are represented as green points.

4.4.2 Segmentation using DeepLabv3+

Another segmentation method used in this thesis is based on semantic segmentation by the use of Deeplabv3+. As already mentioned Deeplabv3+ is an open source image segmentation network incorporating an encoder-decoder structure and has several freely available network backbones. Atrous Spatial Pyramid Pooling (ASPP) enables feature extraction without the loss of spatial information and by using *atrous depth-wise convolution* the computational demands are significantly reduced. For simplicity we will refer DeepLab in the further course.

Fig. 4.15 b) shows an example of the segmentation mask using the finally trained DeepLab model. Although DeepLab is optimized to produce clear edges at class boundaries, there are still minimal deviations. Therefore, the predicted segmentation mask is combined with the result of the second derivative Gaussian filter and Otsu`s threshholding, which was discussed in Section 4.4.1. DeepLab is trained on the original image, consequently the segmentation mask is in camera perspective. Therefore, the mask is transformed with *IPM* after the prediction step. In a last step the same outlier detection as discussed in Section 4.4 is used to generate a final segmentation mask. The result can be seen in Fig. 4.12 b). The pipeline for image segmentation using Deeplab is shown in Fig. 4.13.

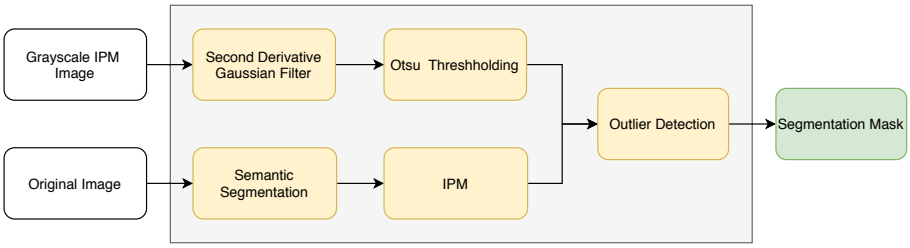


Figure 4.13: Segmentation pipeline using DeepLabv3+.

Data Sets

For training and evaluation two data sets were used. In a first step, the ApolloScape data set [25] for lane segmentation was used. The ApolloScape data set consists of over 110.000 frames of different street scenarios in different cities. For training several subsets containing different scenarios such as tunnel drives or highway drives were selected. The images scenarios and difficulties are uniformly distributed. Therefore, a simple holdout cross validation was used in this thesis. This means the data set was split into a training set, which is used for training, and a test set, which is used for the performance evaluation. The data is split as shown in Table 4.1.

However, the quality of the ground truth images from the ApolloScape data set is insufficient for a highly accurate segmentation of lane markings. Therefore, in a second step the model was fine tuned on a data set provided by the TU Graz. The data set

Data Sets	Number of Images
Training Set	21763
Test Set	4200

Table 4.1: Split of ApolloScape data set

Data Sets	Number of Images
Training Set	1289
Test Set	328

Table 4.2: Split of TU data set

includes annotated images of a part of the ALP.Lab test region[1] and consists of highway scenarios in Central Styria. Compared to the ApolloScapes data set, the data set has a reduced amount of available frames but has highly accurate annotations. For fine tuning again hold out cross validation was used and the data set is split as shown in Table 4.2.

Transfer Learning

Training the Deeplab model can be done in two different ways. One could train the model from scratch. Another approach is to reuse a pre-trained model for a specific task and update the classifier weights to make the model work on another related task. This is called transfer learning. In comparison, the use of a pre-trained model takes significantly less time to achieve good results. Therefore, transfer learning with a model which was already trained on CityScapes [15] and ImageNet [16] was used. The networks backbone is the Xception architecture proposed by Chollet [14] using the atrous rates [6, 12, 18] for an output stride[2] os = 16 and the atrous rates [12, 24, 36] for os = 8.

The pre-trained network is able classify 19 different classes such as cars, passengers or bicycles. However, lane marking detection is only about distinguishing between lane marking and background. Therefore, the classifier was tweaked to only distinguish between two classes.

For this purpose, the ground truth data must be available as color indexed images where each index represents a unique class. The image is available as 1-channel grayscale image containing values between 0 and 255. Some areas in the image are undefined or should not be considered for learning. Therefore, an additional label called *ignore label* is used. Pixels marked as *ignore label* are excluded in the learning process. The CityScapes data set consist of 19 different classes whereas lane segmentation is a two-class segmentation problem. The goal is to separate a lane marking from the rest of the image denoted as background. Therefore, the ground truth data consists of three different index values consisting of background, lane marking and an ignore area. Fig. 4.14 represents an

[1] For more information about ALP.Lab follow `https://www.alp-lab.at/`
[2] Ratio of input image size to size of output feature map

Parameter	Value
Number of Iterations	10000
Atrous Rates	[6, 12, 18]
Batch Size	2
Output Stride	16
Decoder Output Stride	4
Fine Tune Batch Normalization	False
Label Weight Background	5
Label Weight Lane Marking	95

Table 4.3: Training parameters for DeepLab model

example of an ground truth image used training.

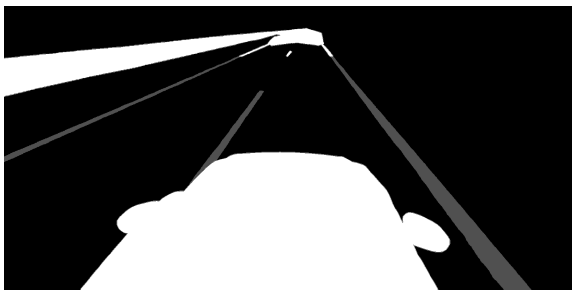
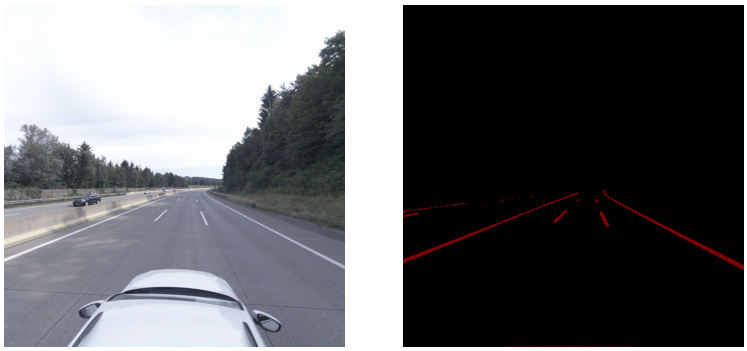


Figure 4.14: Grund truth image for training the Deeplab model with white areas representing ignore areas, black areas representing background and grey areas representing lane markings.

The model was fine tuned with the parameters shown in Table 4.3. For both sets 10.000 iterations were used. Due to limited memory on the hardware, the batch size could only be set to 2. Taking into account that the number of pixels belonging to lane markings is comparably low to the overall amount of background pixels, the weights for calculating the loss during training were set accordingly.



(a) Input image **(b)** Result

Figure 4.15: Semantic Segmentation with DeepLab

4.4.3 Instance Segmentation using SCNN

As already discussed, the task of optimizing semantic lane segmentation is a challenge for typical *SCNNs*. Because the slice-by-slice convolutions of *SCNN* enable row-wise and column-wise message passing between pixels in one layer and therefore lead to a better learning of spatial relationships, a pretrained model of *SCNN* is used for instance segmentation.

Instead of performing a pixel-wise semantic segmentation, *SCNN* outputs a probability mask for each lane, indicating the probability of the center of the lane marking at the corresponding pixel.

Datasets

The model used in this thesis is trained with the CULane data set [44] and the TU Graz data set. The CULane data set consists of 133.235 frames from different drives in Beijing. The annotations were done manually with cubic splines. The data set was split as shown in Table 4.4.

Data Sets	Number of Images
Training Set	88880
Test Set	34680

Table 4.4: Split of CULane data set

Again, the model was fine tuned using holdout cross validation on the data set provided by the *ICG*, which was already introduced in Section 4.4.2.

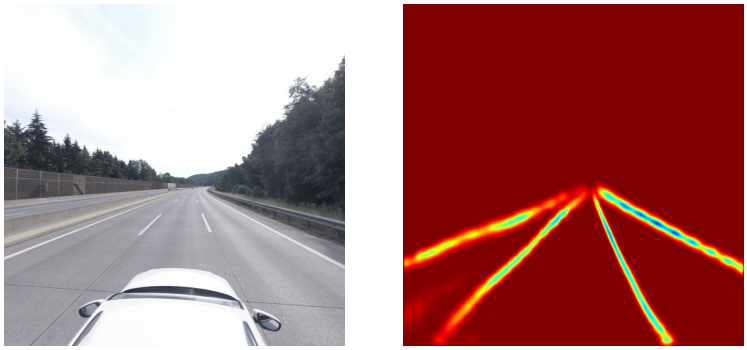
Fine Tuning

The annotation of the ground truth pixels has to be available in a .text file in which the center of each line is represented by key points in x and y coordinates. Therefore, the ground truth data of the TU Graz data set was converted to the correct data format first. The annotated images of the TU Graz data set were created by sampling the center of each lane from the ground truth images from Section 4.4.2. Afterwards, the training was done with the parameters shown in Table 4.5.

Parameter	Value
Number of Iterations	10000
Initial Learning Rate	0.01
Batch Size	4
Learning Decay Rate	0.1

Table 4.5: Training parameters for SCNN model

As already mentioned, instead of performing a pixel-wise semantic segmentation, *SCNN* outputs a 1-channel probability mask indicating the probability of the center of the lane marking at the corresponding pixel for each lane. Fig. 4.16 shows the result of a scene from the data set provided by the TU Graz. It must be mentioned, that the *SCNN* outputs a mask for each lane, but for visualization those images were combined to one mask.



(a) Input image **(b)** Result

Figure 4.16: Semantic Segmentation with SCNN. Note: For visualization the different probability masks were combined to one mask.

4.5 Lane Detection

In the course of this thesis a combination of Hough transform and RANSAC lane fitting for lane detection was implemented. Hough transform is used to detect the base position of the lines in the image whereas RANSAC line fitting is used to fit a mathematical model to the lane marking. Since the curvature of highways is constrained and vehicles drive at high speed a quadratic model is sufficient to mathematically describe a lane marking.

It must be mentioned that the algorithm using *SCNN* for segmentation does estimate the lane base position based on Hough transform. The *SCNN* incorporates instance segmentation and outputs a probability map for each lane separately, which means, that no distinction has to be made between the different lanes. Therefore, only RANSAC model fitting is done for lane detection in the algorithm based on *SCNN*.

4.5.1 Line Model

As discussed in Section 3.5, researchers use different models to describe the properties of the lines. Depending on the use case an appropriate model is used. Sometimes it is sufficient to describe a lane as straight line, whereas other cases require a more descriptive model such as splines or polynomials. This thesis aims on multi lane detection on highway scenarios. The curvature of highway lanes is restricted, due to the high vehicle speed of the road participants. Therefore, a quadratic model was chosen to represent a lane. The

quadratic model is described as

$$y = ax^2 + bx + c \quad , \tag{4.12}$$

with a, b and c as model parameters and x, y as coordinates. The lane parameters can be used to calculate the slope μ, the curvature k and the lateral offset Δy as shown as follows:

$$\Delta y = -c \tag{4.13}$$

$$\mu = -\arctan(b) \tag{4.14}$$

$$k = \frac{2a}{(b^2+1)^{3/2}} \tag{4.15}$$

4.5.2 Hough Transform

The theory of Hough transform is represented in Section 2.6. The segmentation mask as shown in Fig. 4.11 or Fig. 4.15 is taken as input to the Hough transform algorithm. The algorithm tries to detect lines in an binary image and outputs detected lines in the image. The detected lines are expressed as

$$\rho = x\cos\theta + y\sin\theta \tag{4.16}$$

with ρ as the distance on the X-axis and θ as the angle.
Depending on the lane width, the Hough transform outputs multiple lines for each lane marking. The line with the highest score or the line with the largest number of votes within a given lateral distance is selected for further processing. Some further refinements have been implemented by making use of simple lane marking characteristics. Since the input is an IPM image, one can assume that the angle of lane markings is within a given region. As a result, a minimum and a maximum angle θ is defined to filter out unreasonable detections. Furthermore, only lines with a given lateral distance between are considered for further processing.

4.5.3 RANSAC Model Fitting

The residual lines are sampled vertically along its length with a given vertical step size. To find the lane center for each point a one-dimensional Gaussian filter with a variance of the average lane marking width was used. In a refinement step RANSAC algorithm using a least mean square estimation is used to fit a quadratic model to each line estimated by the Hough transform. In this case RANSAC estimates the model parameters a, b and c of Eq. (4.12). The parameters with the highest score are used to refit a new quadratic model considering all sample points. The lane detection stage is summarized in Algorithm 2.

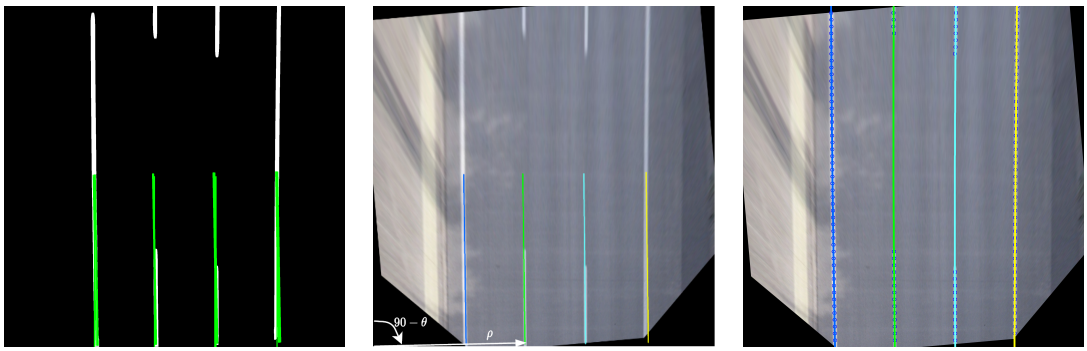
Fig. 4.17 shows an example of lane marking detection. Fig. 4.17 b illustrates the highest scoring lines described with ρ and θ after Hough transform. In c the lines are represented

Algorithm 2 Lane Detection

```
1: procedure LineBaseDetectionHoughTransform(segmentationMask)
2:     Detect lines with Hough transform
3:     Outlier removal                ▷ Considering simple lane marking characteristics
4:     Estimate highest scoring lines
5: end procedure

6: procedure RANSACModelFitting(lines, segmentationMask)
7:     for all lines do
8:         Sample line
9:         for m ← 0 to MaxIterations do
10:            Randomly select a subset of the sample points of the lane
11:            Fit a quadratic model to the subset using least mean square estimation
12:            Compute a scoring function J_i
13:            if J_i ¡ threshold then
14:                Save the parameters
15:                Update the threshold
16:            end if
17:        end for
18:        Re-fit the highest scoring model considering all sample points
19:    end for
20: end procedure
```

with the quadratic model estimated with RANSAC line fitting. The corresponding sample points are represented as blue circles.



(a) Lines detected by Hough transform **(b)** Highest Scoring Lines **(c)** Lane Detection Result

Figure 4.17: Example of lane detection with **a** showing the detected lines and **b** showing the highest scoring lines. **c** shows the result after RANSAC model fitting.

4.6 Line Tracking

After lane detection the individual lanes are classified and tracked. The two most common techniques used for lane tracking are Kalman filter and particle filter. A particle filter would require a high number of particles for each lane, what would lead to higher computational costs. Therefore, a Kalman filter was chosen for lane tracking. Fig. 4.18 shows the overall architecture used to track and classify the detected lanes.

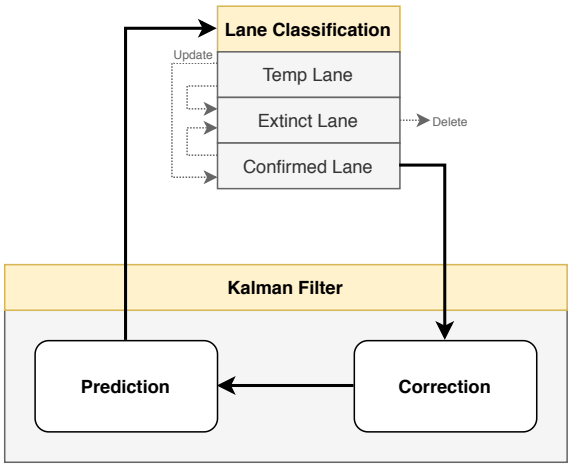


Figure 4.18: Architecture for lane tracking including classification

4.6.1 Line Classification

The lane classification implemented in this thesis is based on the concept proposed by Borkar et al. [5]. After a lane lines is detected, it either has to be associated to a line from the previous frame or has to be classified as a new line. The association is done by determining the lateral distance between the lanes. If the distance between a lane from the previous frame and a lane of the current frame is below a certain threshold, the lane is assigned to the specific lane.

As stated in Table 4.6, three different types of lanes have been defined for classification: *Temp Line, Confirmed Line* and *Extinct Line*. A new lane is classified as *Temp Lane*. If the lane marking was detected more than n times it becomes a *Confirmed Line*. Furthermore, *Confirmed Line* and *Temp Line* become *Extinct Line* if they were not detected m times. In the implementation we set the values of m and n to 3. As can be seen in Fig. 4.18 only *Confirmed Lines* are tracked by the Kalman Filter.

4.6.2 Kalman Filter

As already mentioned, Kalman Filter is the preferred technique for lane tracking. The theory of Kalman Filter was discussed in Section 2.7. For each *Confirmed Line* a separate

Temp Line	Confirmed Line	Extinct Line
A line is detected less than n times	A *Temp Line* is detected more than n times	A *Confirmed Line* was not detected more than m times

Table 4.6: Classification of lines

Kalman Filter is used. The measurements used for the correction step are the line model parameters $[a, b, c]^T$ of the *Confirmed Lines*. The state vector $\mathbf{x(t)}$ consist of the model parameters and their derivative approximated by the difference of the values between the current frame and the previous frame. The state vector $x(t)$ and the measurement vector $\mathbf{z(t)}$ are defined as

$$\mathbf{x(t)} = \begin{bmatrix} a & \dot{a} & b & \dot{b} & c & \dot{c} \end{bmatrix}^T \tag{4.17}$$

and

$$\mathbf{z(t)} = [a \quad b \quad c]^T \quad . \tag{4.18}$$

The matrix $\mathbf{H}$ which is used to associate the two vectors is defined as

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad . \tag{4.19}$$

The state transition matrix $\mathbf{A}$ is a 6x6 matrix and defined as

$$\mathbf{A} = \begin{bmatrix} 1 & dt & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & dt & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & dt \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.20}$$

with dt representing the time interval between two consecutive frames.

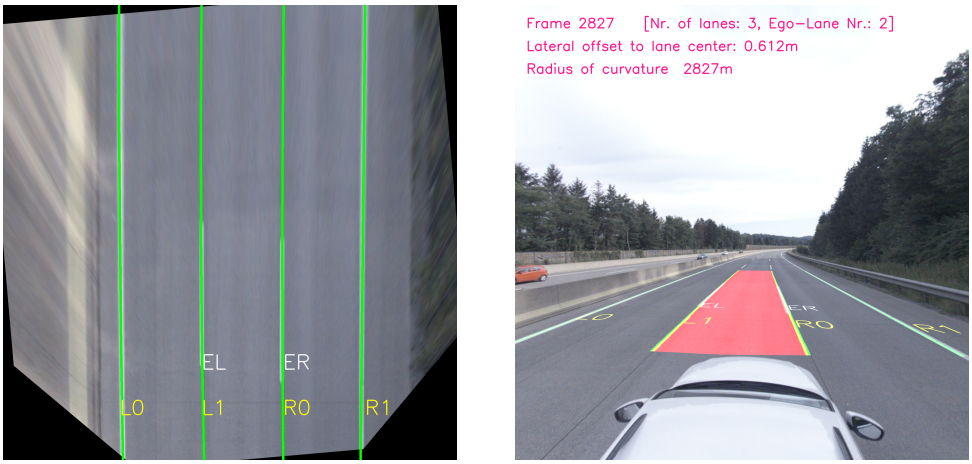
The covariance matrices Q nd R were defined as diagonal matrices with a white Gaussian probability distribution with fixed values.

4.7 Post-Processing

In the last stage the detected lane markings are used to estimate the different lanes in an image and the lateral offset of the vehicle to the center of its ego-lane. Furthermore, the detected lanes including meta information and the vehicles position are recorded in a specific format.

4.7.1 Lane Estimation and Ego-Vehicle Position

As discussed in Section 4.6, lanes are classified into three different types. To estimate the individual lanes and to compute the vehicles lateral position within the ego-lane, only *Confirmed Lines* are used. The detected lines are examined pairwise to obtain the individual lanes and the ego-lane respectively. For the computation the base position of the vehicle in pixels is set to the middle of the image. The two closest lines to the vehicles base position are classified as ego-lane markings. The ego-lane lines are used to estimate the lateral vehicles position to the center of the lane and the curvature of the lane. The curvature is computed with Eq. (4.15). The computations are done in bird-eye perspective, therefore the corresponding points and parameters are transformed to camera perspective in a final step. The final result of lane detection showing the estimation of ego-lane, curvature and lateral vehicle position can be seen in Fig. 4.19.



(a) Result of lane detection and tracking in bird-eye view

(b) Final result of lane detection

Figure 4.19: Result of lane detection with **a** showing the classified lines and **b** showing the final result including estimation of ego-lane, curvature and lateral vehicle position.

4.7.2 Data Recording

The output of the lane detection consists of two files with different data formats. Each file contains different information.

A .csv file is used to store basic information related to each frame. The file contains the lateral vehicles position and curvature of the ego-lane. Furthermore, the information about the road such as number of lanes and the ego-lane as well as the base position and the model parameters in image coordinates for each lane marking are stored in this file.

In addition, the sample points computed in Section 4.5 including meta information such as lane number and lane type are stored in a .shp file. The shape file format is a

specific vector storage format to store the geographic location of geometric features such as points, lines and polygons. Each lane marking is stored as polygon which is created by the corresponding sample points. The data is stored in the WGS84 coordinate system. Fig. 4.20 shows a cutout of a shape file containing lane marking of the A2 highway near Graz.

To store the sample points, they have to be transformed back from image coordinates to global coordinates. The procedure of transforming 3D lidar points to the image plane was already discussed in Section 4.2 and Section 4.3.2. The steps now have to be done in reverse order. First, the sample points are transformed from bird-eye-view back to camera perspective using the homography **H** computed in Section 4.3.3. Since the camera position **C** in the WGS84 coordinate system is known, a ray spanned by the 3D camera position and the 2D image point can be defined. The point of intersection of this ray and the road plane estimated in Section 4.3.1 can be computed as done in Section 4.3.2. An illustration of the ray and the point of intersection can be found din Fig. 4.2. The resulting point of intersection is the sample point in 3D world coordinates.

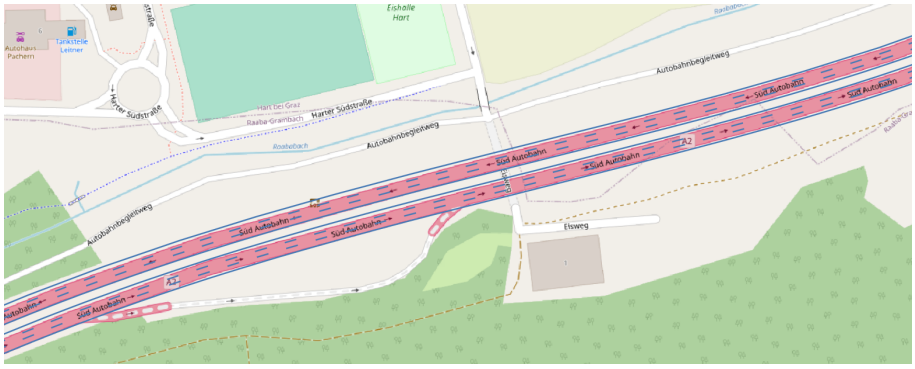


Figure 4.20: Snippet of a shapefile representing a part of the A2 highway near Graz. The blue lines represent the polynomial lane markings.

4.8 Conclusion

In this chapter the concepts of the three different algorithms for lane detection in multi-lane scenarios have been discussed. The choice of the concepts is based on in-depth literature research which was discussed in Chapter 3. The algorithms aim is to extract the lane markings, detect the lanes and compute the vehicle`s lateral position within the ego-lane with high precision.

All three algorithms follow the processing pipeline consisting of the 5 main stages *Pre-Processing, Segmentation, Detection, Tracking* and *Post-Processing* as discussed in Section 4.1. Two of the three algorithms are based on semantic segmentation using deep learning whereas one algorithm uses conventional techniques to segment the lane markings. For the sake of simplicity, the algorithms are named as follows:

1. *Conventional Algorithm*
2. *DeepLab Algorithm*
3. *SCNN Algorithm*

In the *Pre-Processing* stage advanced techniques utilizing LiDAR point clouds, which highly improve the lane detection of the algorithms, have been implemented. For each frame RANSAC plane fitting is used to estimate the road planes parameters with respect to the principle point of the camera. The plane parameters are used to extract the road boundaries and to generate an adaptive *ROI* which only covers the road area. Furthermore, the road plane parameters are used to estimate the homography **H** for *IPM*. In an image smoothing step, noise is removed and the image is converted to gray scale for further processing.

As already mentioned, the three algorithms primarily differ in the concept of image segmentation. In the *Conventional Algorithm* a combination of conventional image filtering techniques is utilized to compute the segmentation mask for further lane detection. The *DeepLab Algorithm* uses a well known deep learning model called DeepLab for image segmentation. The segmentation mask is furthermore combined with the segmentation mask of feature extraction based on a second derivative Gaussian filter and Otsu thresholding. The *SCNN Algorithm* uses a special designed Convolutional Neural Network (CNN) for instance segmentation.

The segmentation masks estimated by the different algorithms are taken as input for the line detection stage. The line detection stage uses a combination of Hough transform and RANSAC model fitting. The Hough transform is used to estimate the base position of the individual lines, whereas RANSAC model fitting is used to estimate the parameters of the lines described by a quadratic model. Compared to the other two algorithms, the *SCNN* outputs a probability mask for each lane. This means that the lines are already differentiated. Therefore, the line detection in *SCNN Algorithm* consists of lane sampling only. The RANSAC model fitting does not need the estimation of the position and distinction between the individual lines.

In the tracking stage the detected lines are classified as *Temp Line, Confirmed Line* and *Extinct Line*. A Kalman Filter is used to track *Confirmed Lines* only. The Kalman filter is designed to take the model parameters of the quadratic lane model as input.

In a final post-processing stage the different lanes, the lateral offset of the vehicle to the center of its ego-lane and the curvature of the ego-lane are computed and recorded. Therefore, the lane sample points are transformed to 3D points in a global coordinate system and written to shape files whereas the remaining parameters are written to a .csv file.

In the upcoming chapter the results of the different algorithms are evaluated in terms of accuracy and reliability.

5 Evaluation

Contents

This chapter is devoted to present the results of the testing and the evaluation of all three algorithms discussed in the previous chapter. For the evaluation different scenarios have been defined to test the algorithms under different conditions. The results will be discussed in the upcoming sections.

5.1 Overview of the Algorithms

This section gives an overview of the different algorithms which will be evaluated. As discussed in Chapter 4, three different algorithms have been developed which follow the same processing pipeline. Two of the algorithms use deep learning architectures whereas one algorithm uses conventional image processing methods for image segmentation. In addition, LiDAR point clouds are used to improve the lane detection with road plane fitting, road boundary extraction and outlier detection.

Additionally, to get an estimate if or how well the incorporation of LiDAR points improve the results, the three algorithms are also evaluated without the use of the LiDAR points. Therefore, for evaluation the three implemented algorithms discussed in Chapter 4 are separated again into a basic version, which does not use LiDAR point clouds, and an advanced version, which utilizes LiDAR point clouds.

For the sake of simplicity the algorithms are named as follows and consist of the following steps:

1. *Basic Conventional Algorithm*
   - Pre-Processing: Gray scale conversion + Static Region Of Interest (ROI) + Inverse Perspective Mapping (IPM)
   - Segmentation: Conventional segmentation
   - Detection and Tracking: Hough transform and RANSAC line fitting + Kalman filter
2. *Advanced Conventional Algorithm*
   - Pre-Processing: Gray scale conversion + RANSAC plane fitting + Adaptive *ROI* + *IPM*
   - Segmentation: Conventional segmentation + Outlier Detection
   - Detection and Tracking: Hough transform and RANSAC line fitting + Kalman filter
3. *Basic DeepLab Algorithm*
   - Pre-Processing: Gray scale conversion + Static *ROI* + *IPM*
   - Segmentation: Semantic segmentation
   - Detection and Tracking: Hough transform and RANSAC line fitting + Kalman filter
4. *Advanced DeepLab Algorithm*
   - Pre-Processing: Gray scale conversion + RANSAC plane fitting + Adaptive *ROI* + *IPM*
   - Segmentation: Semantic segmentation
   - Detection and Tracking: Hough transform and RANSAC line fitting + Kalman filter
5. *Basic SCNN Algorithm*
   - Pre-Processing: Gray scale conversion + Static *ROI* + *IPM*
   - Segmentation: Instance segmentation
   - Detection and Tracking: RANSAC line fitting + Kalman filter
6. *Advanced SCNN Algorithm*
   - Pre-Processing: Gray scale conversion + RANSAC plane fitting + Adaptive *ROI* + *IPM*

- Segmentation: Instance segmentation
- Detection and Tracking: RANSAC line fitting + Kalman filter

Each algorithms follows the processing pipeline as shown in Fig. 4.1. The *Advanced* algorithms utilize all concepts discussed in Chapter 4 with each algorithm using the corresponding segmentation method. The pre-processing of the algorithms includes RANSAC road plane fitting, the computation of an adaptive *ROI* and *IPM*. The segmentation is individual for each algorithm as discussed Section 4.4. The lane detection and tracking as well as the post-processing is the same for all algorithms and is discussed in Section 4.5, Section 4.6 and Section 4.7.

The **Basic** algorithms differ in pre-processing and in the segmentation stage. Since there are no LiDAR point clouds available a static *ROI* and *IPM* with a fixed homography is computed. Gray scale conversion is not effected by the absence of the LiDAR points and is still done. The individual segmentation steps follows the same pipeline as discussed in Section 4.4.1, but the final outlier detection is not done. The lane detection and tracking as well as the post-processing stays the same.

Summarized, the main difference is that in the basic version neither frame-wise road plane fitting or an adaptive *ROI* is computed, nor LiDAR based outlier detection is done. The algorithms exclusively use a static *ROI* and a fixed Homography **H** for *IPM*.

5.2 Datasets

The advance versions of the algorithms expect LiDAR points clouds including the external calibration parameters as input. Therefore, the algorithms have to be evaluated on a data set which provides images as well as LiDAR point clouds and the calibration parameters. Due to the lack of free data sets that include images, ground truth data for lanes and point clouds with the calibration parameters, the algorithms have been evaluated based on the ALP.Lab data set.

To collect this data set, more than 400 km of the ALP.Lab test region have been recorded with a Leica Pegasus 2 - Ultimate Dual Head camera by the Joanneum Research. The hardware setup comprises six 12 MP cameras, a 24 MP panorama camera and two LiDAR sensors. Based on that data, a HD map of the ALP.Lab test region was created in cooperation with the Institute of Computer Graphics and Vision (ICG) from the Graz University of Technology. The map includes road markings which are used for evaluation of the lane detection of the algorithms. The HD map was purchased by the AVL List GmbH, but the images for evaluation have been provided by the *ICG* from the Graz University of Technology.

As discussed in Section 4.4, the models for semantic segmentation have been trained on images showing parts of the ALP.Lab test track. However, for training only of some parts of the ALP.Lab test region have been used. For the evaluation different parts and situations of the whole data set, which were not used for training, have been chosen. In

addition, the data set was split into two different categories with various difficulties for general lane detection and tracking algorithms:

1. *Normal Scenarios*:
   Normal conditions are defined as scenarios with good lighting conditions, an ideal homogeneous road structure with clear lane markings and a limited amount of shadows cased by other objects.

2. *Hard Scenarios*:
   Hard conditions are defined as scenarios with complex illumination, many road participants, tunnel drives or worn out lane markings.

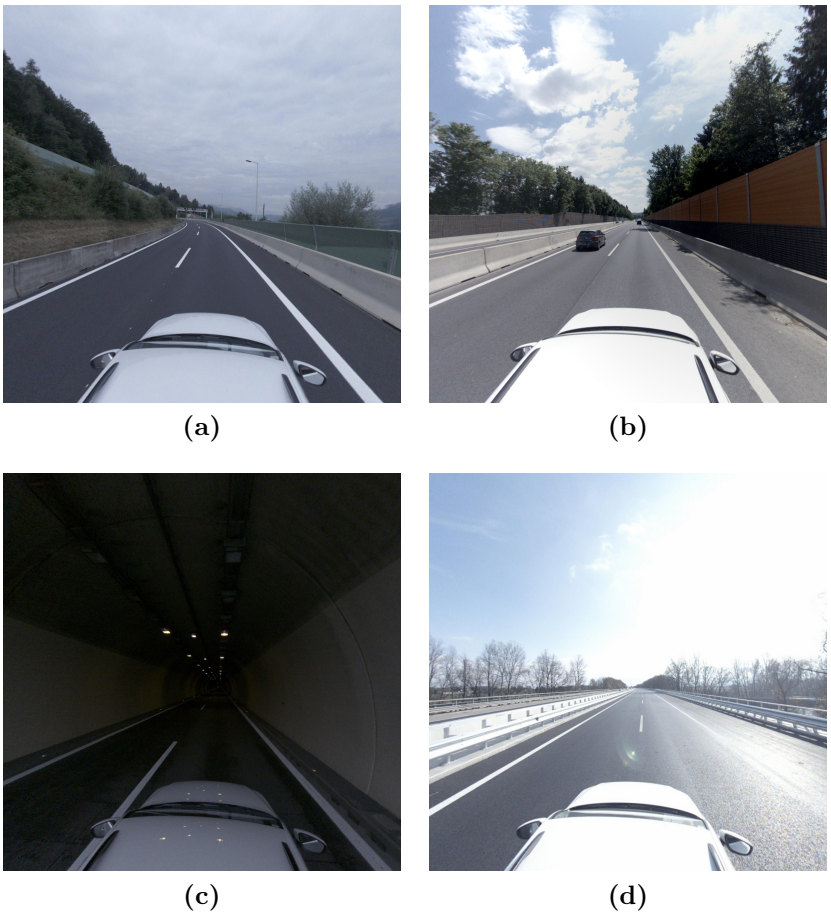
Scenario	Nr. of Images	Type	Image Dimensions
Normal	2000	Highway	2000x2000
Hard	400	Highway, Tunnel	2000x2000

Table 5.1: Datasets used for evaluation

The details of the data sets are shown in Table 5.1. In Fig. 5.1 some images of the defined scenarios are shown.

5.3 Evaluation Criteria

As already mentioned, there is a large number of researches in lane detection and tracking. However, a comparison of the results between the different researches is difficult because of the variation of the techniques and different data sets used. For example, the algorithms implemented in this thesis additionally expect LiDAR point clouds as input. Nevertheless, there criteria have been defined which enable the comparison of different algorithms even if the concepts and data sets differ: *Processing Speed*, *Performance* and *Accuracy*.
The computation of the performance and the accuracy is done in image space, more precisely in the bird-eye view of each frame. The result of the lane detection algorithms are poly lines including the sample points estimated in the lane detection stage. Since the Ground Truth (GT) lanes are available in the .shp file format, the data global 3D data points of the GT lane markings have to be transformed to the image plane and afterwards to the bird eye perspective. This is done in the same way as discussed in Section 4.3. There the 3D LiDAR points are transformed to the image plane and then to the bird eye view by using the homography $\mathbf{H}$. Fig. 5.2 shows an example of a bird-eye image used as base for the evaluation. Afterwards each lane marking is assigned to a GT lane marking with the smallest Hausdorff distance as proposed in [34]. It must be mentioned that multiple detected lane markings can be assigned to the same GT lane marking, but not visa versa.



(a) **(b)**

(c) **(d)**

Figure 5.1: Some images used for evaluation with **a** and **b** showing images under normal conditions and **c** and **d** showing images under hard conditions.

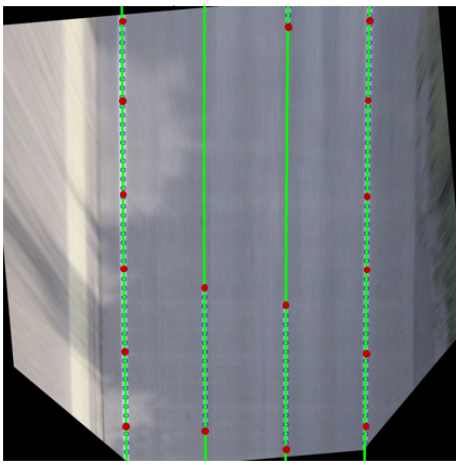


Figure 5.2: Example of *GT* points and detected lanes in bird eye perspective with the green lines representing the predicted poly lines, the blue circles representing the sample points and the red circles representing the *GT* sample points.

5.3.1 Processing Speed

The processing speed is considered as the time of processing one frame and heavily relies on different factors such as the used hardware, the used programming language and the dimensions of the input image. Nevertheless, considering all factors one can get a good estimate of the capabilities and potentials of the algorithms.

5.3.2 Performance

To evaluate the performance of the algorithms the harmonic mean(F1-score) of Precision and Recall is computed by the number of True Positives (TP), False Positives (FP) and False Negatives (FN).

Therefore, lanes have to be compared with its assigned detected lanes for each frame. If the detected lanes distance to the *GT* lane is within a given threshold the lane is considered as *TP*. A lane is marked as *FP* if the distance exceeds the threshold. Furthermore, if there is no lane assigned to a *GT* lane, it is marked as a *FN*. There are no false *GT* lanes, therefore, the value of True Negatives (TN) is unknown.

Precision and Recall or the f1-score respectively is a popular evaluation metric in computer vision and machine learning. Precision is referred as the quantity of the right predictions that the algorithm made and is defined as

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.1}$$

Recall is defined as the quantity of the right predictions the model made concerning the total positive values present and is defined as

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.2}$$

Based on precision and recall the harmonic mean or the F1-Score can be computed which is defined as

$$\text{F1-Score} = 2 \quad \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad . \tag{5.3}$$

In addition, the True Positive Rate (TPR) and the False Detection Rate (FDR) is used for to evaluate the performance. The *TPR* is the recall value and the *FDR* is defined as the rate of false predictions with respect to the overall lanes. The *FDR* should be as low as possible.

5.3.3 Accuracy

In addition to the performance or reliability of the algorithms, one also wants to know how accurate the detection of the estimated lanes is. To get an estimate about the accuracy of the individual algorithms, the standard deviation and the absolute mean of the error is computed. In this regard, the lines are not considered as a whole, but the individual

sample points and the corresponding *GT* points. As can be seen in Fig. 5.2, the sample points of the *GT* lane markings have a low resolution and the samples only indicate the beginning and the end of dashed lane markings. To compute the results, the individual errors between the *GT* sample and the closest sample point of the estimated lane marking is determined. The results are shown in Section 5.4.

5.4 Results

In this section the results of the metrics applied to the data sets specified in Section 5.2 are discussed and compared with the results of other researches.

5.4.1 Processing Speed

As mentioned in Section 5.3 the processing speed depends on several factors such as the hardware, the programming language and the dimensions of the input image. The proposed algorithms are a Proof of Principle to demonstrate the potential in terms of accuracy and reliability. TNevertheless, the evaluation of the processing speed can give an good estimate of the potentials of the algorithms.

The algorithms have been tested on a desktop with an Intel(R) Core(TM) i5-6300U CPU at 2.4 GHz and 8 GB RAM. The used graphics card is a NVIDIA GeForce RTX 2080 with 8GB memory. It must be mentioned, that the GPU was just used for the inference step in the segmentation stage of the deep learning based algorithms. The algorithms have been implemented in Python3 with OpenCV 4.1.0 and Tensorflow 1.15.

Table 5.2 shows the average processing speed of the implemented algorithms and those of other researches[4] [5] [29]. Comparing the implemented algorithms with each other, one can see that among the basic algorithms, the basic SCNN Algorithm is the fastest one with 3.9 FPS and the advanced DeepLab Algorithm is the slowest one with 0.65 FPS. This is due to the fact that, the segmentation Conventional Algorithm was implemented in Python only using the CPU. The segmentation of the deep learning based algorithms was done by using the GPU for the inference step what results in a slightly faster segmentation. Compared to the basic versions, the processing speed of the advanced algorithms is significantly slower. This is due to the combination of handling a huge amount of LiDAR points and the frame-wise RANSAC road plane fitting to compute the adaptive *ROI* and remove outliers, see Chapter 4 and Section 4.4.1.

Comparison with state-of-the-art algorithms

In comparison with other researches, Borkar et al. [5] implemented their algorithms in a scripting language as well. They were able to achieve 1.25 FPS. Bertozzi and Broggi [4] and Kim [29] were able to reach 10 FPS in average and both algorithms have been implemented in C++. As can be seen in Table 5.2, the image dimensions of 512 x 256

Algorithm	Frame Dimensions	Frames per Second (FPS)	Setup	Programming Language
Basic Conventional	2000 x 2000	3.1	Intel(R) Core(TM) i5-6300U with 8GB RAM	Python3
Advanced Conventional	2000 x 2000	1.4	Intel(R) Core(TM) i5-6300U with 8GB RAM	Python3
Basic DeepLab	2000 x 2000	1.9	Intel(R) Core(TM) i5-6300U with 8GB RAM and NVIDIA GeForce RTX 2080	Python3
Advanced DeepLab	2000 x 2000	0.65	Intel(R) Core(TM) i5-6300U with 8GB RAM and NVIDIA GeForce RTX 2080	Python3
Basic SCNN	2000 x 2000	3.9	Intel(R) Core(TM) i5-6300U with 8GB RAM and NVIDIA GeForce RTX 2080	Python3
Advanced SCNN	2000 x 2000	2.1	Intel(R) Core(TM) i5-6300U with 8GB RAM and NVIDIA GeForce RTX 2080	Python3
Bertozzi et.al. [4]	512 x 256	10.0	Full-custom massively parallel hardware.	C++
Bokar et.al. [5]	640 x 480	1.25	Intel(R) Core(TM) i7-4770 CPU at 3.40 GHz and 8.00GB RAM	MATLAB R2015b
Kim et.al [29]	176 x 120	10.0	Intel Pentium 4 at 3 GHz	C++

Table 5.2: Processing speed of the algorithms

and 176 x 120 are significantly lower than the 2000 x 2000 pixels of our data set. Even if Python as scripting language is comparatively slow in relation to a compiler language such as C++, the overall performance of the implemented algorithms is comparably good, by taking the input dimensions into account.

5.4.2 Performance

In this section the performance is evaluated based on the F1-Score of Precision and Recall. In the case of a lane detection algorithm, the precision indicates if the ground truth lanes have been detected considering wrong detentions. The value of the recall corresponds with the amount of right detections of GT lanes. Both values should be as high as possible. Consequently, the higher the F1-Score, the better the overall performance and reliability.

As mentioned in Section 5.2 the data set was split into two sub sets containing images of normal conditions and hard conditions respectively. For the evaluation the F1-Score of Precision and Recall is computed for both sub-sets separately.

Normal Scenarios

The normal scenarios include a total of 3822 *GT* lanes, which are used to estimate the *TP*, *FP* and *FN* of the lane detection algorithms. As can be seen in Table 5.3, all algorithms show a low amount of *FP* and *FN* and a resulting high F1-Score in scenarios with normal conditions. The Basic Conventional Algorithm algorithm achieved the lowest score of 0.9759 whereas the Advanced SCNN Algorithm shows the best result of 0.9888. It can be seen that the advanced algorithms show slightly better results than the basic versions. It is noticeable that the algorithms primarily differ in the detection of *FP* with the Basic Conventional Algorithm reaching an outstanding amount of 116 *FP* detections. The results of the algorithms are comparably good and only show minimal differences among each other. The high Precision and Recall of the algorithms shows, that almost all lanes are detected and the rate of false detected lanes is quite low.

Algorithm	# of TP	# of FP	# of FN	Precision	Recall	F1-Score
Basic Conventional	3753	116	85	0.9700	0.9819	0.9759
Advanced Conventional	3737	10	69	0.9973	0.9778	0.9874
Basic DeepLab	3723	12	99	0.9968	0.9741	0.9853
Advanced DeepLab	3754	5	68	0.9987	0.9822	0.9904
Basic SCNN	3740	6	82	0.9984	0.9785	0.9884
Advanced SCNN	3747	3	75	0.9992	0.9804	0.9897

Table 5.3: Performance on data set with normal scenarios

Hard Scenarios

The hard scenarios include a total of 1851 *GT* lanes. Table 5.4 shows the results of the algorithms performing under challenging conditions. As expected, the Precision and Recall and the F1-Score of all algorithms is generally lower than under normal scenarios . It can clearly be seen that the basic Conventional Algorithm algorithms shows the worst results with an outstanding high amount of not detected lanes and, as a result, a low Recall of 0.75 and a F1-Score of 0.78. It can be seen that the deep-learning based algorithms outperform the conventional algorithms in challenging scenarios. Even if the conventional algorithms show good results under normal conditions, the amount of *FP* and *FN* is outstanding higher, especially of the basic Conventional Algorithm. The deep-learning based algorithms show better and more stable results. This might be due to the fact that the deep-learning models were also trained on some challenging scenarios and therefore, better cope with such scenarios in the segmentation stage. Furthermore, it is noticeable the advanced algorithms show better results than the basic one.

Algorithm	# of TP	# of FP	# of FN	Precision	Recall	F1-Score
Basic Conventional	1381	270	470	0.8139	0.7461	0.7785
Advanced Conventional	1559	260	292	0.8571	0.8422	0.8496
Basic DeepLab	1609	100	242	0.9415	0.8693	0.9039
Advanced DeepLab	1694	108	157	0.9401	0.9152	0.9275
Basic SCNN	1602	28	249	0.9828	0.8655	0.9204
Advanced SCNN	1646	28	205	0.9833	0.8892	0.9339

Table 5.4: Performance on data set with hard scenarios

Comparison with state-of-the-art algorithms

Fig. 5.3 shows the total *TPR* and the *FDR* of the implemented algorithms performing on both data sets and the results from other researches. As discussed in Section 5.3, the *TPR* is the value of the Recall and the *FDR* is the rate of wrongly detected lanes. Since there was no freely available data set including point clouds and images, the results of researches working on different data sets had to be compared. In this section the *TPR* and the *FDR* are compared with the averaged results of Son et al. [53], Aly [1] and Guo et al. [20] working on the Caltech Lane Datasets [1]. To compare the implemented algorithms with other state-of-the-art algorithms, the total *TPR* and the *FDR* of both data sets were computed. Nevertheless, even if the data sets differ, the comparison still gives an estimate about the potential and robustness of the algorithms implemented in this thesis.

As can be seen in Fig. 5.3 the reference algorithms achieved an average *FDR* in the range of 8.88% and 12.01% and a high *TPR* up to 99.0% with [20] having the lowest value of 94.2%.

The results show that the performance of the implemented algorithms is comparably good. Among them, the basic Conventional Algorithm shows the worst results with a False Positive Rate (FPR) of only 0.9 % and a *FDR* of 17.05%. It is noticeable that the deep-learning-based approaches outperform the reference researches in terms of *FDR*, but none of the implemented algorithms was able to achieve a *TPR* of 99.0 %. Nevertheless, except the basic Conventional Algorithm, the results still represent a high detection rate and the *TPR* just slightly differs among the algorithms.

5.4.3 Accuracy

In this section the standard deviation and the absolute mean error of the individual sample points of the detected lanes are evaluated. Table 5.5 shows the individual as well as the total results of the implemented algorithms for normal and hard conditions. It must be mentioned that *IPM* leads to a reduction in the accuracy. However, the difference in accuracy was minimal and therefore, was neglected. The Mean Absolute Error (MAE) and the Standard Deviaton of the Error (SDE) were computed after *IPM* in birds eye perspective. As can be seen, under normal conditions all algorithms are able to achieve

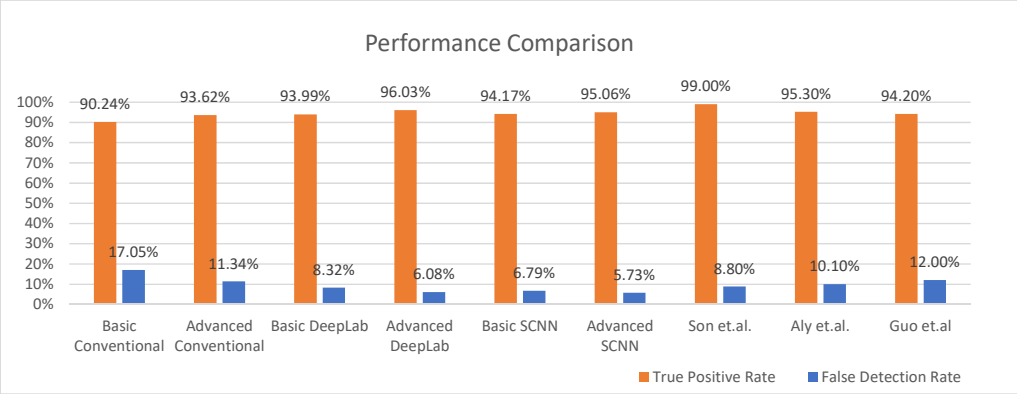


Figure 5.3: Comparison of performance with state-of-the-art algorithms

Algorithm	Normal Conditions		Hard Conditions		Total	
	MAE (cm)	SDE (cm)	MAE (cm)	SDE (cm)	MAE (cm)	SDE (cm)
Basic Conventional	3.29625	2.2115	13.0979	9.2591	8.1971	5.7353
Advanced Conventional	3.0869	2.1037	9.8982	6.9687	6.4925	4.5362
Basic Deeplab	3.4066	2.2433	11.0116	7.1157	7.2091	4.6795
Advanced Deeplab	3.0770	2.1553	9.8217	6.9165	6.4494	4.5359
Basic SCNN	4.1606	2.4614	13.2997	9.4437	8.7301	5.9526
Advanced SCNN	4.0896	2.3344	11.9907	8.5040	8.0402	5.4192

* SDE = Standard Deviation of Error, MAE = Mean Absolute Error

Table 5.5: Accuracy under various conditions

reasonable results with a std. deviation of around 2.2 cm. The Advanced Conventional Algorithm was able to achieve an std. deviation of 2.1 cm what is the best result of all algorithms. As already mentioned, hard conditions, inter alia, include lanes which are partly covered, worn out or only partly recognizable due to challenging illumination or an in-homogeneous road surface. Furthermore, in-homogeneous road structures also lead to higher errors in lane sampling. Nevertheless, considering that, the algorithms are still able to achieve quite reasonable results.

The total results show that the Basic SCNN Algorithm achieves the worst results in terms of accuracy. This is due to the fact, that the lane sampling is just about sampling the max value of the individual probability map. Even if the reliability of the deep-learning based algorithms is quite good, the predicted lanes or the lane centers of the Spatial Convolutional Neural Network (SCNN) model seem to come with a certain deviation. Furthermore, it can clearly be seen, that the Advanced Conventional Algorithm, as well the Advanced DeepLab Algorithm stand out with a std. deviation of 4.54 cm each.

Comparison with state-of-the-art algorithms

The results are compared with the work of McCall and Trivedi [38] and Sivaraman and Trivedi [51] which evaluated their accuracy of the sample points on the ego-lanes. As can be seen in Fig. 5.4, the implemented algorithms show comparably very good results and outperform the reference algorithms in terms of accuracy. Even the algorithm with the worst results shows a significantly lower standard deviation and mean absolute error.

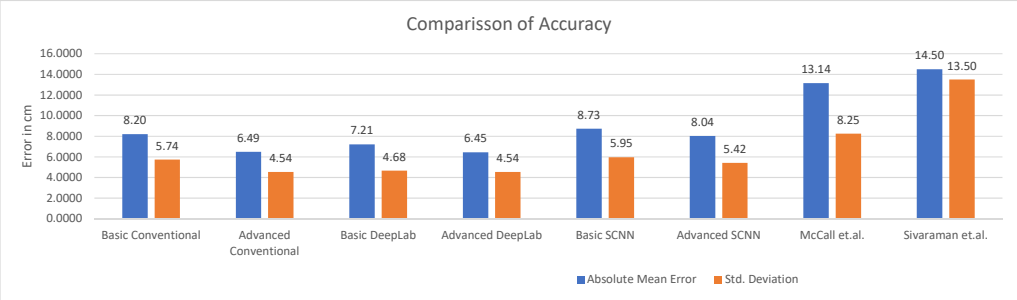


Figure 5.4: Comparison of lane detection accuracy with state-of-the-art algorithms

5.5 Experimental Results

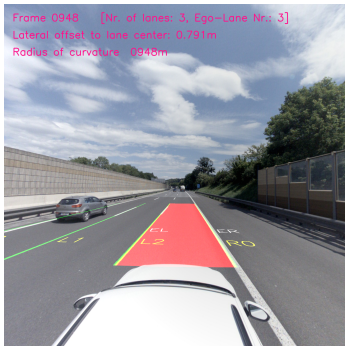
Beside the general evaluation of the algorithms some experiments with the algorithms performing under various conditions have been made. In this section the experimental results and the limitations of the algorithms are presented.

Normal Scenarios

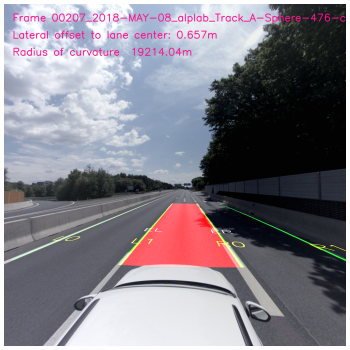
Fig. 5.5 shows some results of the algorithms working under various normal driving conditions. Even the basic algorithms work well and output correct positions of the lane markings. The advanced algorithms show similar results. Therefore, Fig. 5.5 only represents the results of individual algorithms. It is noticeable that, even if there are other objects on the road or shadows covering lanes, the lanes are well detected.

Hard Scenarios

Fig. 5.6 shows some results of the algorithms working under hard conditions. Here again scenarios are shown where all algorithms performed similar. Therefore, the results of individual algorithms are shown. Fig. 5.6a shows a scene with a bad road structure that could lead to miss classified lanes. Fig. 5.6b shows a tunnel driving scenario and Fig. 5.6c shows a scenario with many road participants. Due to the huge area covered by the truck in Fig. 5.6c, none of the algorithms was able to detect the left outer lane.



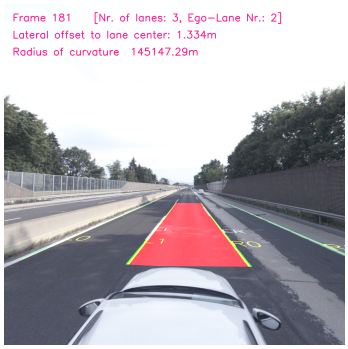
(a) *Basic Conventional Algorithm*



(b) *Basic DeepLab Algorithm*

(c) *Advanced Conventional Algorithm*

Figure 5.5: Results of normal driving scenarios



(a) Advanced Conventional Algorithm

(b) Advanced SCNN Algorithm

(c) Advanced DeepLab Algorithm

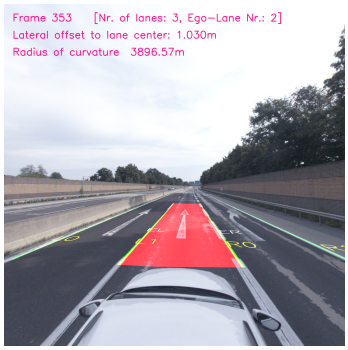
Figure 5.6: Results of hard driving scenarios

Special Cases

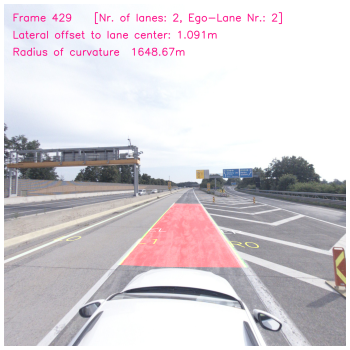
The algorithms were also tested in some special scenarios and the results are shown in Fig. 5.7. All algorithms are able to handle situations where the lane markings are complicated as shown in Fig. 5.7a, b and c. As can be seen in Fig. 5.7d, a sudden change of exposure such as driving in a tunnel is handled well by all of the algorithms.

Influence of Advanced Processing

There are special cases where the algorithms are prone to errors. For example white road participants lead to failures in the detection of the Basic Conventional Algorithm as can be seen in Fig. 5.8c. This is due some false lane evidences in the segmentation mask produced by the car. The advanced algorithm is able to handle this situation as can be seen in Fig. 5.8b and d. In relation to the conventional algorithm, one can see that

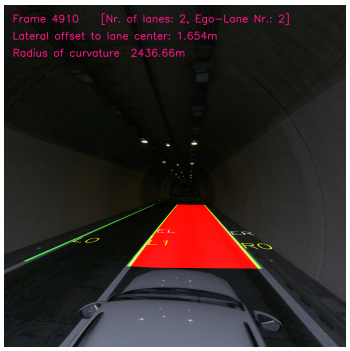


(a) Advanced Conventional Algorithm



(b) Advanced DeepLab Algorithm

(c) Basic DeepLab Algorithm



(d) Advanced DeepLab Algorithm

Figure 5.7: Results of special driving scenarios

the outlier detection of the advanced algorithms significantly improves the quality of the segmentation mask and the stability of the algorithms.

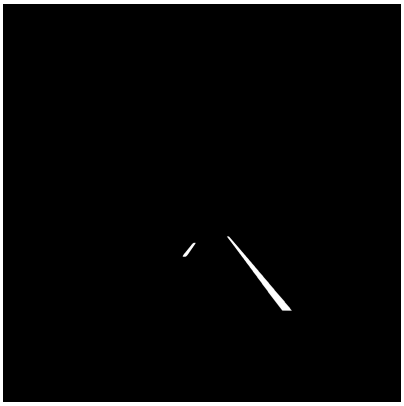
Furthermore, this is a good example to also illustrate the advantage of the tracking stage. Even if there is no third lane in the segmentation mask, it is still tracked, as can be seen in Fig. 5.8b and d. However, due to the huge area covered by the bus for multiple frames, none of the algorithms was able to track the outer left lane.

As can be seen in Fig. 5.9, the adaptive *ROI* significantly improves the lane detection. Without the road plane fitting and the resulting adaptive *ROI*, objects, which do no belong to the road, might be potentially detected as lane markings. This is illustrated in Fig. 5.9a, c.

Limitations

During the process of testing, several limitations of the systems were found. Situations with a critical illumination or reflections potentially lead to failures. In scenarios with multiple lane markings such as can be seen on construction places, none of the algorithm

(a) Basic Conventional Algorithm



(b) Advanced Conventional Algorithm

(c) Basic Conventional Algorithm

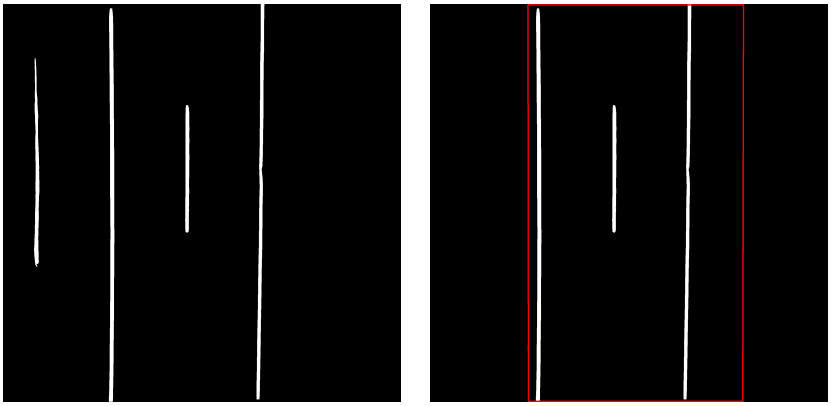
(d) Advanced Conventional Algorithm

Figure 5.8: Influence of Outlier Detection and Tracking

is able to distinguish between the different lanes and achieve reasonable results (Fig. 5.10a, b). If the exposure becomes too low, none of the algorithms is able to detect a lane at all. An example can be seen in Fig. 5.10c. Due to reflections, the algorithms potentially miss some lanes, as can be seen Fig. 5.10d.

5.6 Conclusion

In the course of this thesis three different algorithms have been implemented. To have an estimate if and how well the incorporation of LiDAR point clouds improves the overall performance and accuracy, each algorithm was separated again into an *advanced* version which uses LiDAR point clouds and a *basic* version which does not. As a result, six algorithms have been evaluated.
Data from the ALP.Lab data set was used for the evaluation. Furthermore, the images



(a) Basic Conventional Algorithm

(b) Advanced Conventional Algorithm

(c) Basic Conventional Algorithm

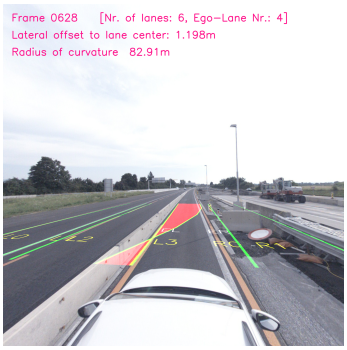
(d) Advanced Conventional Algorithm

Figure 5.9: Influence of Adaptive *ROI*

have been separated into different categories with various difficulties.

In the course of the evaluation, the processing speed, the performance and the accuracy have been analyzed and compared with the results of other researches. As mentioned several times, the main focus was on accuracy and performance. The algorithms are Proof of Concepts and have been implemented in Python. However, the processing speed could give an good estimate about the potentials of the algorithms. Obviously the advanced implementations of the algorithms are quite slower than the basic versions. The fastest algorithm is the Basic SCNN Algorithm followed by the Basic Conventional Algorithm. Comparing only the advanced implementations, again the Advanced SCNN Algorithm is the fastest followed by the Advanced Conventional Algorithm.

In terms of performance and accuracy, the Advanced Conventional Algorithm and the deep-learning based algorithms show close results. However, the Advanced DeepLab Algorithm is the overall winner. Due to its high Precision and Recall as well as its



(a) Advanced DeepLab Algorithm

(b) Advanced SCNN Algorithm

(c) Advanced Conventional Algorithm

(d) Advanced DeepLab Algorithm

Figure 5.10: Limitations of the algorithms

low standard deviation of the error, the algorithm shows an overall high detection rate combined with a high accuracy.

Furthermore, it is noticeable that the incorporation of LiDAR point clouds significantly improves the detection rate of the algorithms. Nevertheless, the algorithms accuracy is just slightly influenced.

By doing some experimental tests, it was shown that the Basic Conventional Algorithm is prone to errors in situations with white road participants. White road participants potentially corrupt the segmentation image what could lead to wrongly detected lanes. The advanced Conventional Algorithm is able to filter that by the use of LiDAR points. Furthermore, the algorithms tend to fail in challenging scenarios which incorporate low exposure, light reflections or constructions places. Nevertheless, in general it was shown that the algorithms mainly perform well and are able to handle several complicated situations such as tunnel drives, situations with many road participants, roads with an inhomogeneous surface or complicated lane markings.

6 Conclusion and Future Work

6.1 Conclusion

In the course of this thesis in-depth research in state-of-the-art methods for lane detection and tracking in multi lane scenarios was done. A special focus was on the analysis of the differences, limitations and draw backs in using conventional techniques as well as deep-learning based techniques for image segmentation.

The development took place gradually. Initially some literature research in terms of current research outcomes in the field of lane detection as well as ego-lane estimation and its current limitations and challenges in highway scenarios was done. Furthermore, a general processing pipeline for lane detection and tracking consisting of five stages was derived. In a first step a conventional solution for lane detection and ego-vehicle positioning was implemented. In a next step, an investigation in image segmentation based on deep learning was done. As a result, three algorithms have been implemented. Each algorithm follow the processing pipeline consisting of the five stages: *Pre-Processing, Segmentation, Detection, Tracking* and *Post-Processing*. The algorithms primary vary in the segmentation stage of the general processing pipeline. That means that the algorithms differ in the generation of the segmentation mask which is taken as input for the further processing stages. In addition, a LiDAR sensor is used to improve the lane detection by utilizing the 3D point cloud for road plane fitting, road boundary extraction and outlier removal in the segmentation masks.

In a final step, an evaluation of the performance and accuracy based on labeled ground truth data was done. The ALP.Lab data set was used to compare the lane detection output with the ground truth. To have an estimate if and how well the incorporation of LiDAR point clouds improves the overall performance and accuracy, the algorithms are separated again into a basic version and an advanced version. In comparison with each other, the basic versions do not use LiDAR point clouds for outlier detection or frame-wise road plane fitting and only have a static Region Of Interest (ROI). Ultimately, it came down to six algorithms for evaluation - two for each each implemented algorithm. The different

approaches and metrics were analyzed to represent the most accurate and reliable solution for lane detection, ego-vehicle localization and ego-lane estimation for highway scenarios. As discussed in Chapter 5, all algorithms are able to perform well under normal as well as challenging conditions. The Advanced Conventional Algorithm and the deep-learning based algorithms show close results, but the Advanced DeepLab Algorithm shows the best. Due to its high Precision and Recall as well as its low standard deviation of the detection error, the algorithm achieves an overall high detection rate combined with a high accuracy. In Chapter 1 some hypotheses in terms of lane detection have been made, which can be answered here. First of all, the use of an additional LiDAR does improve the stability or reliability of the lane detection algorithms. There is a significant difference in performance between heuristically image processing techniques compared to deep learning based approaches. In fact deep learning based approaches overcome instabilities and failures in complex situations such as tunnel drives or dynamic illumination. As can be seen in Section 5.4, the deep learning based approaches show a higher quality in segmenting the lanes under challenging conditions, what leads to an better overall detection rate. Taking a look at the results of the accuracy, the Advanced DeepLab Algorithm shows the best result going head to head with the Advanced Conventional Algorithm. The SCNN Algorithm is stable and reliable in detecting the evidence of a lane, but comes with an inaccuracy. It must be mentioned again, that the accuracy was computed for True Positivess (TPs), therefore, a missed lane was not considered in the computation of the deviations. The results of the Conventional Algorithm are good, but one has to consider the stability or reliability.

To estimate the lateral vehicles position within the ego-lane, the position of two lanes is needed. As can be seen in Section 5.4.3 the standard deviation of the error in lane detection is around 5 cm in total. Therefore, the the algorithms are able to to estimate the vehicles position with a std. deviation of around 5 cm as well.

6.2 Application and Future Work

The results show that the Advanced DeepLab Algorithm comes with the best compromise of performance and accuracy. Consequently, is the best solution for lane detection in multi lane scenarios. Nevertheless, considering the processing speed, the algorithm is the slowest and the algorithm is not real-time capable. Specifying the best algorithm is a matter of the use case and the dedicated requirements. Some systems do not have real-time requirements. Here it is sufficient to detect the lanes in an offline processing step and accuracy and stability are the main criteria. Here it is definitely recommended to use the Advanced DeepLab Algorithm.

In case real-time capability in combination with a high performance and accuracy is required, it is recommended to improve the Advanced Conventional Algorithm. Even if the Advanced DeepLab Algorithm shows the overall best accuracy and performance, the deep-learning model is a bottleneck. It would be complicated to optimize the models archi-

tecture in terms of processing speed. A re-implementation of the Advanced Conventional Algorithm in C++ and the introduction of parallel computing would be a more efficient and less complicated way to improve the processing speed.

Beside the improvement of the processing speed, there are further options to improve the performance and the accuracy of lane detection. The overall estimation of the ego-lane and the vehicles position could be improved by incorporating GPS information and a HD map of the track. Furthermore, one could investigate in the optimization of the deep-learning based approaches. The current architectures for image segmentation could be optimized in terms of processing speed and accuracy or one could investigate in the development of a specially optimized deep-learning architecture for image segmentation.

A

List of Acronyms

AD	Automated Driving iii, 1, 17, 18
ADAS	Advanced Driver Assitstance Systems iii, 1, 17, 18
ASPP	Atrous Spatial Pyramid Pooling 25, 26, 46
CNN	Convolutional Neural Network xiii, 11–13, 18, 24, 26, 57
DNN	Deep Neural Network 10
DOF	Degrees of Freedom 24
FCN	Fully Connvolutional Network 13
FDR	False Detection Rate 64, 68
FN	False Negatives 64, 67
FoV	Field of View 12, 25
FP	False Positives 64, 67
FPR	False Positive Rate 68
GT	Ground Truth xiv, 18, 62–67
ICG	Institute of Computer Graphics and Vision 36, 49, 61
IPM	Inverse Perspective Mapping 18–21, 23, 24, 31, 34, 36, 40, 41, 46, 57, 60, 61, 68, 69
MAE	Mean Absolute Error 69
NN	Neural Network xiii, 9–11, 13, 20
RANSAC	Random Sample Consensus 8, 24, 29
ROI	Region Of Interest iii, xiv, 18–20, 24, 28, 31, 33, 34, 36, 38–40, 57, 60, 61, 65, 72, 74, 77
RPF	RANSAC Plane Fitting 36
SCNN	Spatial Convolutional Neural Network 26, 27, 34, 48–50, 57, 69
SDE	Standard Deviaton of the Error 69
SotA	State-of-the-Art 13, 18

TN	True Negatives 64
TP	True Positives 64, 67, 78
TPR	True Positive Rate 64, 68

Bibliography

[1] M. Aly. Real time detection of lane markers in urban streets. *CoRR*, abs/1411.7113, 2014. URL `http://arxiv.org/abs/1411.7113`. (page 68)

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015. URL `http://arxiv.org/abs/1511.00561`. (page 13, 14)

[3] R. Berriel, E. Aguiar, V. Vieira, and T. Oliveira-Santos. A particle filter-based lane marker tracking approach using a cubic spline model. In *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, 08 2015. doi: 10.1109/SIBGRAPI.2015.15. (page 30, 31)

[4] M. Bertozzi and A. Broggi. Gold: a parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE Transactions on Image Processing*, 7(1):62–81, 1998. (page 65, 66)

[5] A. Borkar, M. Hayes, and M. T. Smith. Robust lane detection and tracking with ransac and kalman filter. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 3261–3264, Nov 2009. (page 19, 20, 23, 29, 30, 31, 33, 53, 65, 66)

[6] B. D. Brabandere, D. Neven, and L. V. Gool. Semantic instance segmentation with a discriminative loss function. *CoRR*, abs/1708.02551, 2017. URL `http://arxiv.org/abs/1708.02551`. (page 24, 25)

[7] D. C. Brown. Close-range camera calibration. *PHOTOGRAMMETRIC ENGINEERING*, 37(8):855–866, 1971. (page 7)

[8] A. G. U. H. P. M. CÌS_{xx} (*a*)kmakci. *Automotive control systems*. Cambridge University Press, 1 edition, 2012. URL `http://gen.lib.rus.ec/book/index.php?md5=cfc650fefc17fb0aabe31140f4257db5`. (page 17)

[9] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. (page 21)

[10] Cao, S. P., X. Wenjiao, and P. Zhong. Lane detection algorithm for intelligent vehicles in complex road conditions and dynamic environments. *Sensors*, 19:3166, 07 2019. doi: 10.3390/s19143166. (page 44)

[11] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016. URL `http://arxiv.org/abs/1606.00915`. (page 25)

[12] L. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017. URL `http://arxiv.org/abs/1706.05587`. (page 12, 13, 26)

[13] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018. URL `http://arxiv.org/abs/1802.02611`. (page 26, 27)

[14] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016. URL `http://arxiv.org/abs/1610.02357`. (page 25, 47)

[15] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. (page 27, 47)

[16] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009. (page 47)

[17] N. Doshi, B. Peddagolla, S. Kathiresan, N. Prabhu, and D. Zadeh. On the lane detection for autonomous driving: A computational experimental study on performance of edge detectors. 06 2018. (page 22)

[18] M. Fischler and R. Bolles. *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*. AD-a460 585. sri international menlo park ca artificial intelligence center, first edition, 1980. URL `https://books.google.at/books?id=We8UuAAACAAJ`. (page 8)

[19] S. G. Foda and A. K. Dawoud. Highway lane boundary determination for autonomous navigation. In *2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (IEEE Cat. No.01CH37233)*, volume 2, pages 698–702 vol.2, Aug 2001. doi: 10.1109/PACRIM.2001.953728. (page 19)

[20] J. Guo, Z. Wei, and D. Miao. Lane detection method based on improved ransac algorithm. In *2015 IEEE Twelfth International Symposium on Autonomous Decentralized Systems*, 2015. (page 68)

[21] A. Hata and D. Wolf. Road marking detection using lidar reflective intensity data and its application to vehicle localization. In *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 584–589, Oct 2014. doi: 10.1109/ITSC.2014.6957753. (page 28)

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`. (page 25, 26, 27)

[23] R. Hota, S. Syed, S. Bandyopadhyay, and P. Radha Krishna. A simple and efficient lane detection using clustering and weighted regression. In *Proceedings of the 15th International Conference on Management of Data*, 01 2009. (page 19)

[24] P. V. C. Hough. Method and means for recognizing complex patterns. 12 1962. doi: Patent3,069,654. (page 13)

[25] X. Huang, X. Cheng, Q. Geng, B. Cao, D. Zhou, P. Wang, Y. Lin, and R. Yang. The apolloscape dataset for autonomous driving. *CoRR*, abs/1803.06184, 2018. URL `http://arxiv.org/abs/1803.06184`. (page 46)

[26] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960. (page 14)

[27] S. Kammel and B. Pitzer. Lidar-based lane marker detection and mapping. In *2008 IEEE Intelligent Vehicles Symposium*, pages 1137–1142, June 2008. doi: 10.1109/IVS.2008.4621318. (page 28)

[28] A. Karpath. Convolutional neural networks for visual recognition, 2019. URL `http://cs231n.github.io/neural-networks-1/`. (page 10)

[29] Z. Kim. Robust lane detection and tracking in challenging scenarios. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):16–26, 2008. (page 65, 66)

[30] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. *CoRR*, abs/1210.5644, 2012. URL `http://arxiv.org/abs/1210.5644`. (page 27)

[31] F. Kunihiko. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 1980. (page 11)

[32] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL `http://dl.acm.org/citation.cfm?id=645530.655813`. (page 13)

[33] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 1558-2256. doi: 10.1109/5.726791. (page 12)

[34] J. Liang, N. Homayounfar, W.-C. Ma, S. Wang, and R. Urtasun. Convolutional recurrent network for road boundary extraction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. (page 62)

[35] P. Lindner, E. Richter, G. Wanielik, K. Takagi, and A. Isogai. Multi-channel lidar processing for lane detection and estimation. In *2009 12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–6, Oct 2009. doi: 10.1109/ITSC.2009.5309704. (page 28)

[36] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014. URL `http://arxiv.org/abs/1411.4038`. (page 13)

[37] J. Matas. Ransac in 2011 (30 years after). 2011. (page 8)

[38] J. C. McCall and M. M. Trivedi. Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. *IEEE Transactions on Intelligent Transportation Systems*, 7(1):20–37, 2006. (page 70)

[39] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans, and L. V. Gool. Towards end-to-end lane detection: an instance segmentation approach. *CoRR*, abs/1802.05591, 2018. URL `http://arxiv.org/abs/1802.05591`. (page 20, 24)

[40] M. A. Nielsen. Neural networks and deep learning, 2018. URL `http://neuralnetworksanddeeplearning.com/`. (page 9, 10, 11, 13)

[41] M. Nieto, J. Arrospide, and L. Salgado. Road environment modeling using robust perspective analysis and recursive bayesian segmentation. *Mach. Vis. Appl.*, 22:927–945, 11 2011. doi: 10.1007/s00138-010-0287-7. (page 19, 20, 22)

[42] H. Oliveira, A. Boukerche, E. Nakamura, and A. Loureiro. Localization in time and space for wireless sensor networks: An efficient and lightweight algorithm. *Perform. Eval.*, 66:209–222, 03 2009. doi: 10.1016/j.peva.2008.10.009. (page 19, 20)

[43] M. Oliveira, V. Santos, and A. Sappa. Multimodal inverse perspective mapping. *Information Fusion*, 09 2014. doi: 10.1016/j.inffus.2014.09.003. (page 20, 21)

[44] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang. Spatial as deep: Spatial CNN for traffic scene understanding. *CoRR*, abs/1712.06080, 2017. URL `http://arxiv.org/abs/1712.06080`. (page 26, 27, 28, 49)

[45] D. Pomerleau. Ralph: rapidly adapting lateral position handler. In *Proceedings of the Intelligent Vehicles '95. Symposium*, pages 506–511, Sep. 1995. (page 20)

[46] J. M. S. Prewitt. *Object enhancement and extraction*, chapter Picture Processing and Psychopictorics, pages 75–149. New York: Academic Press, 1970. (page 21)

[47] L. Roberts. *Machine Perception of Three-Dimensional Solids*. Garland Publishing, New York, 01 1963. (page 21)

[48] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. (page 25)

[49] H. Schneiderman and M. Nashman. Visual processing for autonomous driving. In *[1992] Proceedings IEEE Workshop on Applications of Computer Vision*, pages 164–171, Nov 1992. doi: 10.1109/ACV.1992.240315. (page 19, 21)

[50] S. Sehestedt, S. Kodagoda, A. Alempijevic, and G. Dissanayake. Robust lane detection in urban environments. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 123–128, Oct 2007. doi: 10.1109/IROS.2007.4399388. (page 19)

[51] S. Sivaraman and M. M. Trivedi. Integrated lane and vehicle detection, localization, and tracking: A synergistic approach. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):906–917, 2013. (page 70)

[52] I. Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014. (page 21)

[53] Y. Son, E. Lee, and D. Kum. Robust multi-lane detection and tracking using adaptive threshold and lane classification. *Machine Vision and Applications*, 30, 10 2018. doi: 10.1007/s00138-018-0977-0. (page 20, 24, 30, 68)

[54] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, first edition, 2010. ISBN 1848829345. (page 15)

[55] M. Tian, F. Liu, and Z. Hu. Single camera 3d lane detection and tracking based on ekf for urban intelligent vehicle. In *2006 IEEE International Conference on Vehicular Electronics and Safety*, Dec 2006. doi: 10.1109/ICVES.2006.371626. (page 30)

[56] T.-T. Tran, C.-S. Bae, Y.-N. Kim, H.-M. Cho, and S.-B. Cho. An adaptive method for lane marking detection based on hsi color model. *Communications in Computer and Information Science*, 93:304–311, 08 2010. doi: 10.1007/978-3-642-14831-6_41. (page 22, 23)

[57] Tsung-Ying Sun, Shang-Jeng Tsai, and V. Chan. Hsi color model based lane-marking detection. In *2006 IEEE Intelligent Transportation Systems Conference*, pages 1168–1172, Sep. 2006. doi: 10.1109/ITSC.2006.1707380. (page 22)

[58] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. C. Courville, and Y. Bengio. Renet: A recurrent neural network based alternative to convolutional networks. *CoRR*, abs/1505.00393, 2015. URL `http://arxiv.org/abs/1505.00393`. (page 27)

[59] Y. Wang, E. Teoh, and D. Shen. Lane detection and tracking using b-snake. *Image and Vision Computing*, 22:269–280, 04 2004. doi: 10.1016/j.imavis.2003.10.003. (page 22, 29)

[60] Yinghua He, Hong Wang, and Bo Zhang. Color-based road detection in urban traffic scenes. *IEEE Transactions on Intelligent Transportation Systems*, 5(4):309–318, Dec 2004. ISSN 1558-0016. doi: 10.1109/TITS.2004.838221. (page 19)

[61] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000. (page 8)