CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF MASTER’S THESIS

Title:	Prototype of vision-based driver assistant
Student:	Bc. Tomáš Kohout
Supervisor:	Ing. Josef Gattermayer
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of winter semester 2016/17

Instructions

The goal of this thesis is to explore possibilities of road lane recognition to achieve better traffic safety on a functional prototype of a driver assistant.

Instructions:
1) Conduct a survey of previously created vision-based driver assistants.
2) Explore and implement a road lane recognition algorithm.
3) Design and implement a prototype of driver assistant that will:
a) use road lane recognition algorithm on video input to recognize current road lane,
b) display a warning if a car approaches to a road lane boundary to increase driver safety.
4) Explore possibility of using the driver assistant in the context of mobile applications.

References

Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague September 23, 2015

Czech Technical University in Prague
Faculty of Information Technology
Department of theoretical computer science

Master's thesis

Prototype of vision-based driver assistant

Bc. Tomáš Kohout

Supervisor: Ing. Josef Gattermayer, Ph.D.

5th May 2016

Acknowledgements

I would like to thank both my supervisors Ing. Josef Gattermayer, Ph.D. and Yen-Lin Chen, Ph.D. for the support during the development of the thesis. I would also like to thank my family for the encouragement and help with obtaining resources for the thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on 5th May 2016 ....

Czech Technical University in Prague
Faculty of Information Technology
© 2016 Tomáš Kohout. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Kohout, Tomáš. *Prototype of vision-based driver assistant.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Cílém této práce je vytvořít prototyp silničního asistenta s použitím vizuální detekce. Video z dopředu orientované kamery je zpracováno algoritmem založeným na Houghově transformaci a Kalmanově filtru. Prototyp je schopen rozpoznat pozici silničního pruhu a zobrazit varování v případě, že se řidič blíží ke kraji pruhu. Aplikace pro mobilní telefony byla dále vytvořena na základě tohoto prototypu.

Klíčová slova Silniční asistent, Počítačové vidění, Rozpoznání silničního pruhu.

Abstract

The goal of this thesis is to develop a prototype of the vision-based driver assistant. The prototype use an image sequence from a front-mounted camera as the input that is processed via algorithm based on Hough Transform and Kalman filter. The position of the current lane is determined and warning is displayed in case of the lane departure. Additionally, an application for mobile phones is created based on the prototype.

Keywords Vision-based assistent, computer vision, Road lane recognition.

Contents

Citation of this thesis viii

Introduction **1**

Motivation and objectives 1
Problem statements 1
State of the Art 2

1 Previous implementations **3**

1.1 Mobileye 3
1.2 iOnRoad 4

2 Analysis and Design **7**

2.1 Requirements 7
2.2 Modules and techniques 7
2.3 Image Pre-processing 7
  2.3.1 Color threshold 8
  2.3.2 Fixed intensity threshold 8
  2.3.3 Adaptive Intensity threshold 8
  2.3.4 Marking width threshold 9
  2.3.5 Conclusion 9
2.4 Feature extraction 9
  2.4.1 Prerequisites 10
    2.4.1.1 Inverse Perspective Mapping 10
    2.4.1.2 Canny edge detector 11
  2.4.2 Particle filtering 12
  2.4.3 Recursive Bayesian segmentation 14
  2.4.4 Hough transform 14
  2.4.5 Conclusion 17
2.5 Model fitting 17
  2.5.1 Line filtering 17

2.5.1.1 Direction 17
2.5.1.2 Vanishing point 17
2.5.1.3 Direction in the real world 18
2.5.2 Line merging 19
2.5.3 Marking center detection 20
2.5.4 Current lane fitting 25
2.6 Time integration 25
2.6.1 Particle Filter 25
2.6.2 Kalman Filter 25
2.7 Image to World Correspondence 28
2.7.1 Homogeneous coordinates 28
2.7.2 Affine transformation 29
2.7.3 3D Projection 30
2.7.3.1 Orthographic projection 30
2.7.3.2 Weak perspective projection 31
2.7.3.3 Perspective projection 31
2.7.4 Pinhole camera model 31
2.7.5 Inverse perspective projection 34

3 Implementation and testing 37
3.1 Computer vision framework 37
3.1.1 VXL 38
3.1.2 ccv 38
3.1.3 libCVD 38
3.1.4 OpenCV 38
3.1.5 Conclusion 39
3.2 Implementation 39
3.2.1 The Basic Image Container 39
3.2.2 Image capture 40
3.2.3 Image pre-processing 40
3.2.4 Feature Extraction and Model Fitting 40
3.2.5 Lane Tracking 41
3.2.6 Image to world 41
3.2.7 Inverse Perspective Mapping 41
3.2.8 Renderer 42
3.2.9 Settings 42
3.2.10 Lane departure warning 42
3.2.11 Mobile application 42
3.3 Results 43

4 Conclusion 47
4.1 Possible future work 48
4.1.1 Car and pedestrian detection 48
4.1.2 Traffic sign detection 48

4.1.3 Vanishing point detection 48
4.1.4 Dashed line detection 48
4.1.5 Road boundary detection 49
4.1.6 Curved line fitting 49

Bibliography **51**

A Installation **55**
A.0.1 Desktop 55
A.0.2 iOS Mobile Application 55

B Contents of CD **57**

List of Figures

1.1 Mobileye algorithm output 4
1.2 iOnRoad application 5

2.1 Road Lane algorithm diagram 8
2.2 Marking width formula 9
2.3 Inverse perspective mapping 10
2.4 Horizontal kernel convolution 11
2.5 Vertical kernel convultion 11
2.6 Sobel gradient 11
2.7 Sobel direction 11
2.8 Original road image 12
2.9 Canny Edges 12
2.10 Initiation 13
2.11 Progress 13
2.12 Low certainty 13
2.13 Output of Bayesian segmentation 14
2.14 Original Hough line representation 15
2.15 Hough line representation 15
2.16 ρ and θ line representation 15
2.17 Hough space graph 16
2.18 Output of Hough transform 16
2.19 Direction line filtering 18
2.20 Vanishing point 18
2.21 Real world direction line filtering 19
2.22 Marking with multiple lines 19
2.23 Lane marking 1-D sample 20
2.24 Marking intensities 21
2.25 Marking profile 21
2.26 Result of cross-correlation 22
2.27 Markings center points 22

2.28 Linear least squares 23
2.29 Example of a line fitted using vertical distance 23
2.30 Example of a line fitted using perpendicular distance 24
2.31 Coefficient vector for Least Squares 24
2.32 Markings fitted line 24
2.33 Current state in Kalman filter 26
2.34 Transition matrix 27
2.35 Control matrix 27
2.36 Control vector condition 27
2.37 Measurement 27
2.38 Homogeneous coordinates 29
2.39 Affine transformation 29
2.40 Augmented matrix 29
2.41 Translation 29
2.42 Scale 30
2.43 Rotation 30
2.44 Road in 3D space 30
2.45 Projected road 30
2.46 Perspective projection 31
2.47 Perspective projection matrix 31
2.48 Camera top view 32
2.49 Camera side view 32
2.50 Extrinsic matrix 33
2.51 Intrinsic pinhole camera model 34
2.52 Intrinsic matrix 34
2.53 Plane translation and rotation 35

3.1 Example of successfully detected lane in the country road 45
3.2 Lane departure warning 45
3.3 Detection in highway multi-lane scenario 45
3.4 Detection of curved marking 46
3.5 Example of erroneous detection caused by faded left marking . . . 46

List of Tables

3.1 Error in the highway and country scenario 44
3.2 Error in the urban scenario 44
3.3 The performance in FPS 44

Introduction

Motivation and objectives

Road lane recognition is a well known problem throughout the Computer vision. To know the position of the road lane and therefore the cars position within it can be useful for various types of situations. Whether the information is used for warning, cruise control or fully autonomous driving the potential for a car accident is reduced. The problem is also often associated with the traffic signs recognition, pedestrian recognition and vehicle recognition. Because of the usefulness of the road lane recognition, subject is quite well studied and documented. With the recent rise of self-driving cars, there is a great interest in perfecting the road lane recognition. There is number of papers focusing on the subject, but there are not many implementations available.

The goal of the thesis is therefore to create a functioning prototype of the road assistant that will alert the driver when is approaching the lane boundary in real-time. The assistant can be beneficial when the driver is not giving full attention to the road or has fallen into a microsleep [1].

The thesis was partially developed at National Taipei University of Technology in Taiwan under the supervision of Yen-Lin Chen Ph.D.

Problem statements

The problem of the road lane recognition is very complex. The road geometry, markings quality, weather conditions and light can vary significantly. There is a large number of papers focusing on various sub-problems and it is not possible nor intended to cover all of these in this Master Thesis. Instead the focus is set on one subset of the problem:

- Recognize road markings. Road marking is man-made stripe of paint that is put onto the road surface in order to convey official information. It can be either full or dashed.

- Use *road markings* to recognize current road lane. Road lane is part of roadway used for single vehicle bounded by *road markings* on each side.
- Track current lane in consequent frames, even if the *road markings* are not always available. E.g. with dashed road marking.
- Develop system to recognize whether the user is approaching the boundary using information from the steps above.

State of the Art

In the recent years car companies were pushing for better active safety features or even complete autonomy in their vehicles. With the arrival of cameras and other sensors the research in the area accelerated. Some companies like Google use in their self-driving car LIDAR[1] systems to scan the road. These systems, however, are expensive and not widely accessible. There is still need for the visual sensors to recognize markings and traffic signs on the road. According to another car company Tesla owner Elon Musk the *LIDAR* system doesn't make sense in a car context [2]. In Tesla Autopilot [3] they use unique combination of cameras, radar, ultrasonic sensors and data to automatically steer down the highway, change lanes, and adjust speed in response to traffic. It is clear from the recent development that vision-based systems are taking over functions previously performed by a radar. For example Forward Collision Detection and Adaptive Cruise Control[2] was previously domain of radar systems.

This means that Computer vision[3] is slowly taking dominant role in the autonomous driving. Of course, there will be always a need for other redundant systems in order to be ensure high level of safety. But for a road assistant the computer vision is usually sufficient as it does not have to 100% accurate all the time. Compared to autonomous system, the assistant can also allow higher level of false positives because it does not trigger potential live-threatening action (such as braking). There is already a few of commercial vision-based driver assistants available as a standalone system or a smart-phone application.

In the academic sphere there has been a lot of research invested in different algorithms used for the road lane recognition. These algorithms range from very robust but computational demanding to less complex and fast. These algorithms, however, are rarely implemented as real-time systems in C++ or other language. This thesis use some of the algorithms with some adjustments.

---

[1] Lidar (also written LIDAR, LiDAR or LADAR) is a surveying technology that measures distance by illuminating a target with a laser light.

[2] Adaptive cruise control is an optional cruise control system for road vehicles that automatically adjusts the vehicle speed to maintain a safe distance from vehicles ahead.

[3] Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions.

Chapter 1

Previous implementations

In this part we will discuss previous implementations of visual-based assistants.

1.1 Mobileye

Mobileye [4] is the recognized global pioneer in collision avoidance systems and according to the Mobileye website one of the largest artificial vision development center on the planet. Its standalone system Mobileye 560 consist of a front-mounted camera and a small display. Mobileye can warn drivers when the car approach the lane boundary. It can also issue a forward collision warning, pedestrian collision warning, speed limit warning and safe headway distance warning.

Mobileye can connect to car internal system to receive information when the turn signal is turned on. The camera have to be mounted and calibrated precisely by certified engineers in order for the system to work. This means it is in no way easy plug-in system.

The advantage of Mobileye is use of their own hardware. This means they can use their custom designed chips with baked in computer vision algorithms for better performance [5]. This helps to increase the complexity of the algorithms. The disadvantage of the system is its higher price around $800 plus additional installation costs.

Mobileye's lane detection algorithm is not publicly available. It can be only assumed from a promotional video showing the output of algorithm in Figure 1.1 that it is using some kind of advanced b-spline [4] algorithm to be able to fit curved lines.

---

[4] In the mathematical subfield of numerical analysis, a B-spline, or basis spline, is a spline function that has minimal support with respect to a given degree, smoothness, and domain partition.

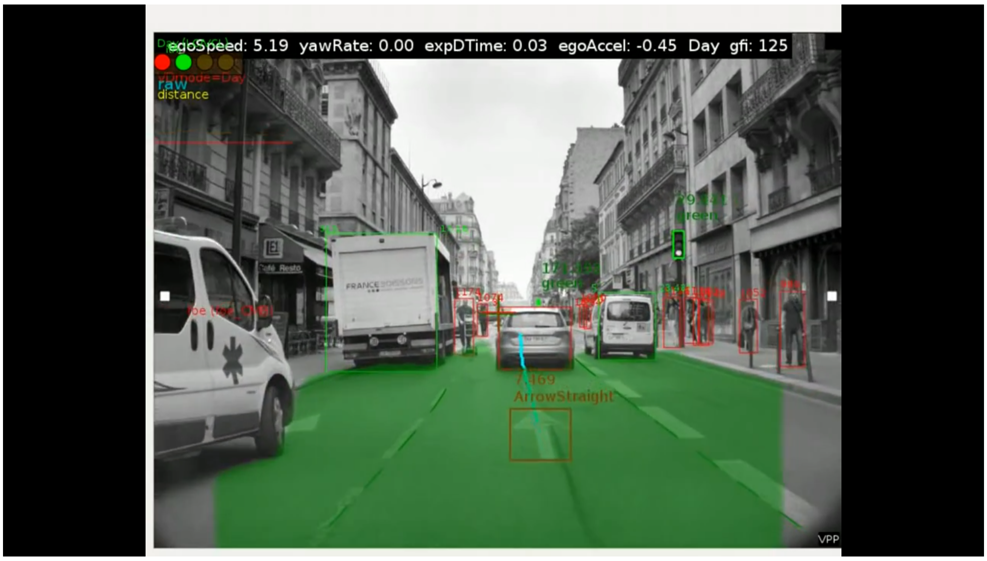

egoSpeed: 5.19 yawRate: 0.00 expDTime: 0.03 egoAccel: -0.45 Day gfi: 125
raw
distance
ArrowStraight
VPP

Figure 1.1: Mobileye algorithm output

1.2 iOnRoad

iOnRoad is an application for iOS and Android. It features lane departure warning as well as forward collision warning and headway monitoring. It also adds some useful features like video recording and car locator.

Advantage of the iOnRoad is its low cost. It can also be easily installed in any car by attaching the smartphone to the windshield, not requiring additional configuration. The trade off is a limitation of the smartphone's computational power. The frame rate is considerably smaller than the mobileyes even when tested on iPhone 5S.

The algorithms used are also less complex than the ones used in Mobileye. The road lane recognition algorithm doesn't fit curved lanes and sometimes have problems to re-adjust after the lane was changed.

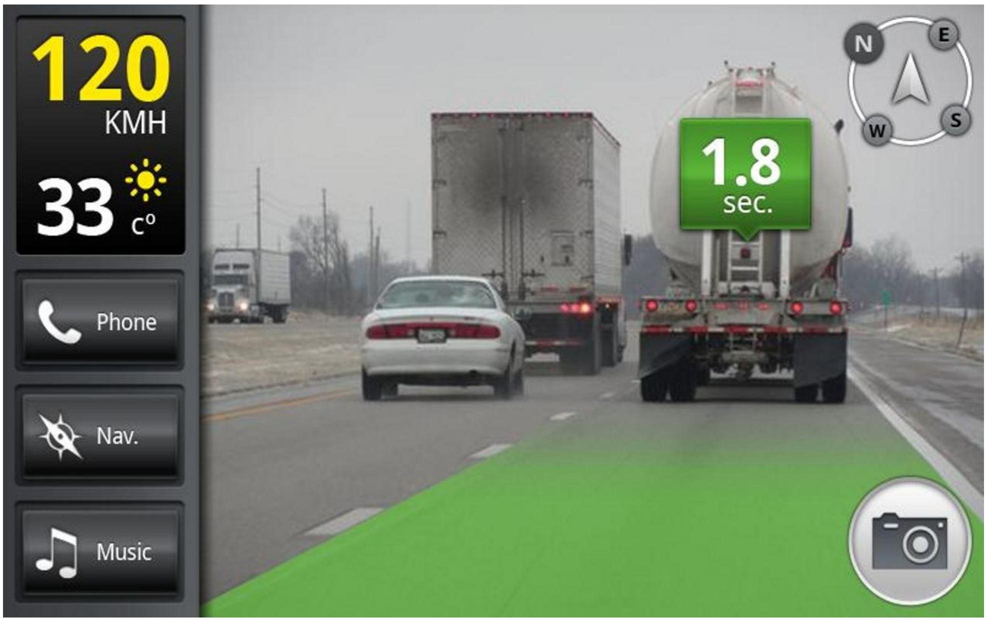

120
KMH
33
c°
Phone
Nav.
Music
1.8
sec.
N
E
W
S

Figure 1.2: iOnRoad application

Chapter 2

Analysis and Design

2.1 Requirements

The chosen techniques used in the algorithm are based on the given requirements. The main goal is to be able to recognize a lane departure and issue a warning. In order for it to be possible the algorithm should:

- Recognize lanes on roads with lane markings.
- Be able to recognize both full and dashed lines
- Be able to track the current lane over time
- Switch lane when the driver is changing the lane

2.2 Modules and techniques

There are many approaches to the road lane recognition. Even though they vary significantly, according to a recent survey [6] the steps common in all algorithms can be described in following diagram 2.1. Even though the modules described does not have to be all present in the algorithm most of them can be mapped as its subsystem. The data flows from the low-level algorithms to the top. There can also be feedback from the higher level modules to the bottom ones to improve the result. E.g. *Image to World* module, once it's extracted, can be useful in feature extraction process.

2.3 Image Pre-processing

The purpose of image pre-processing is to get rid of the noise in the image. It can also correct exposure in case of bad illumination, so the image can be prepared for a next step which is the feature extraction.

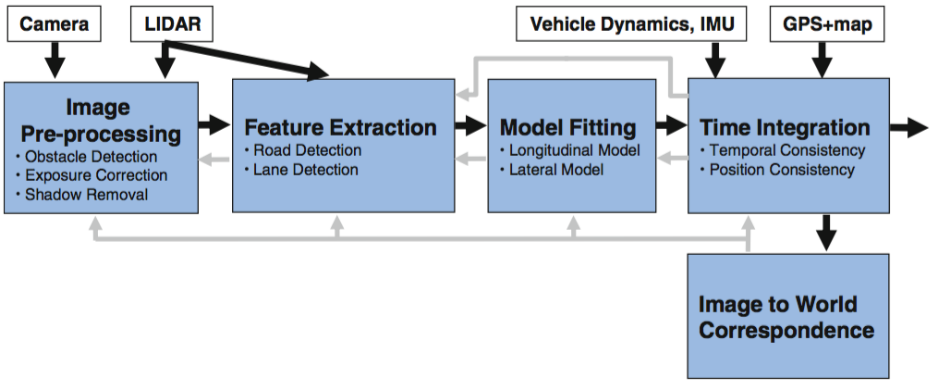

Camera
LIDAR
Vehicle Dynamics, IMU
GPS+map
Image Pre-processing
• Obstacle Detection
• Exposure Correction
• Shadow Removal
Feature Extraction
• Road Detection
• Lane Detection
Model Fitting
• Longitudinal Model
• Lateral Model
Time Integration
• Temporal Consistency
• Position Consistency
Image to World Correspondence

Figure 2.1: Road Lane algorithm diagram

Because the goal is to extract lane markings there need to be a way of separating them in the image. There are few assumptions that can be used for such purpose. The lane markings usually have a contrast color compared to the road. They also have distinct width and lengthy shape. The result of the image pre-processing module is a binary image that should ideally highlight markings in white pixels and discard the road and other features in black.

2.3.1 Color threshold

The assumption that the markings are of a distinct color can be applied. The pixels that are similar to the color within a given ϵ range are used and the rest is discarded. This can produce good results if the threshold color is chosen well.This is, however, naive solution as the illumination condition can change the color of the markings. The markings in many countries can also vary in color (e.g. white and orange).

2.3.2 Fixed intensity threshold

The image can be converted into the gray scale and then compared by its pixel intensity (usually 0-255) with a fixed threshold. Again as in *Color thresholding* the result depends on the choice of the constant. This method is also vulnerable to illumination changes.

2.3.3 Adaptive Intensity threshold

The method is different from the *Fixed intensity thresholding* in the choice of the thresholding parameter. Instead of fixed value, it is computed as a mean of neighbouring pixels. This partially solves the illumination change problem but create a lot of false positive pixels. That can be solved by additional noise removal. The Erosion, Gaussian filter or Median filter can be used to remove the false positive pixels.

2.3.4 Marking width threshold

The method is based on the Marking Detector in [7]. The idea is to use the low-high-low intensity profile of the row in the image to make the threshold more precise and to get rid of the false positives. Each pixel can be described by a formula that is shown in Figure 2.3.4.

$$y_i = 2x_i - (x_{i-\tau} + x_{i+\tau}) - |x_{i-\tau} - x_{i+\tau}|$$

Figure 2.2: Marking width formula

where x_i is the pixel intensity and τ is the presumed marking width. The last term in the function is used to lower intensity of the pixel that has big difference between intensity of the left and right pixels. This way the marking of certain width will be highlighted.

To counter the illumination effects the row is normalized by the rows maximum intensity.

2.3.5 Conclusion

The *Marking width threshold* was selected as the image pre-processing algorithm. The main reason was high reliability and low number of false positive pixels. The width of the lane is in perspective changing with increasing distance which can be solved by implementing feedback from *Image to World* module.

2.4 Feature extraction

In the context of the feature extraction there are two terms often mentioned. Road boundary recognition and lane recognition. These might seem like a similar problems but are considerably different. Road boundaries does not have to be always man-made and can vary significantly. From curbs in cities, grass in the country to the snow in the north teritories. The road also doesn't necessarily have to be paved. This makes the problem much more wide than the lane marking recognition. For that reason this thesis focus solely on the marked roads.

There are many different methods to extract lane markings from the image. Some of them are listed below.

2.4.1 Prerequisites

2.4.1.1 Inverse Perspective Mapping

Some of the papers introduce the Inverse Perspective Mapping (IPM)[8] as the first step. Given the *Image to World* model it can transform input image to a bird-eye view. The points on screen are translated into the real world coordinates and then projected back into x,z space. That can be simplified by creating an affine transformation that will convert a pixel from original image to newly created top image. The cars will be distorted in the image but the road would be as seen from the air when the model is set correctly. This technique can make it easier for some of the algorithms to find the markings as they don't have to deal with the perspective distortion effect.

This technique has however certain trade-offs. If the model is not computed properly, the image will become distorted and detecting of the lines more problematic. It can be caused by wrong installment of the camera that have adjusted pitch or yaw. It is also expected for the road to be flat. In case there is a slope with higher angle (e.g. when car is approaching a hill) the inverse perspective will produce distorted result. This can be, however, mitigated by advanced method for *Image to World* calibration and 3D reconstruction. *IPM* also come with increased computational cost and partial loss of resolution. An example of inverse mapped image is shown in Figure 2.3.

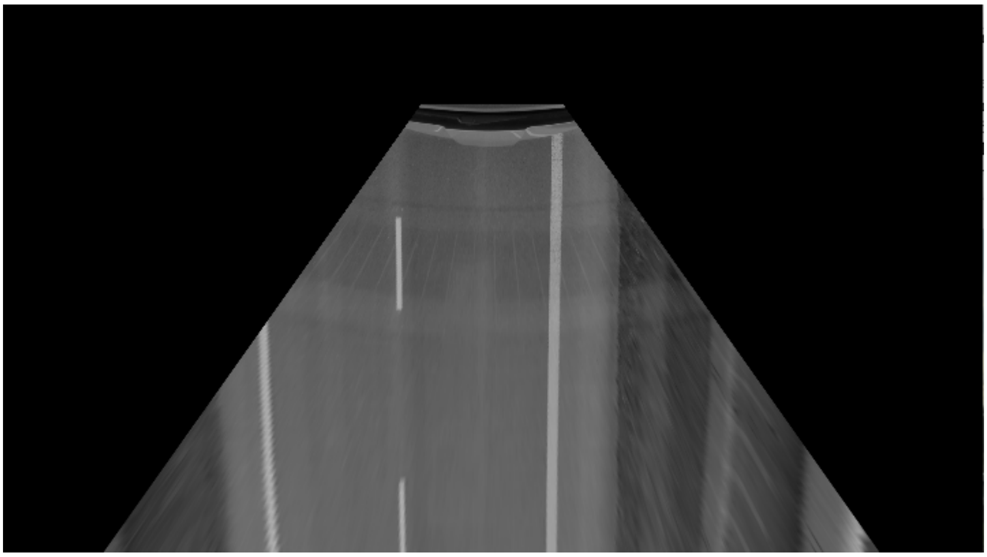


Figure 2.3: Inverse perspective mapping

2.4.1.2 Canny edge detector

Some of the mentioned algorithms need a binary image that describes the edges in the original image as the input. The most used method is the Canny edge detector. It was developed by John F. Canny in 1986 [9]. The process consists of following steps.

Noise reduction Use the Gaussian filter[5] to remove the noise in the image. The kernel size should be carefully selected so the edges are still detectable.

Intensity gradient The intensity gradient need to be computed for every pixel in the image. The image is first converted to a gray-scale image. The Sobel operator [10] is often used in relation with the *Canny edge detector*. The operator uses 3x3 kernel which is applied on every pixel of the image by multiplying locally similar entries and summing (operation known as convolution). It is done once in the vertical and once in the horizontal direction as seen in Figure 2.4 and 2.5. The G_x and G_y returns higher value for neighbouring pixels that differ in their intensities. This way the edges can be highlighted.

$$G_x = \begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix} * A$$

Figure 2.4: Horizontal kernel convolution

$$G_y = \begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix} * A$$

Figure 2.5: Vertical kernel convultion

At each point of the image an approximation of the gradient can be calculated by combining both horizontal and vertical gradients in Figure 2.6. The direction of each pixel can be also computed as shown in Figure 2.7

$$G = \sqrt{G_x^2 + G_y^2}$$

Figure 2.6: Sobel gradient

$$\mathbf{\Theta} = \operatorname{atan2}\left(\mathbf{G}_y, \mathbf{G}_x\right)$$

Figure 2.7: Sobel direction

---

[5] Gaussian filter is a filter whose impulse response is a Gaussian function (or an approximation to it). Gaussian filters have the properties of having no overshoot to a step function input while minimizing the rise and fall time.

Edge thinning The output of the Sobel is a gradient image with different values marking the edge strength. The desired image would, however, have single pixel line in the place of real edges. This way the edge detecting also become resolution independent because high resolution image would display thicker edges. Non-maximum edge suppression is applied to thin the edges. In principle it compares the edge strength of a pixel with its negative and positive gradient neighbour. If the value is largest in the pixel's direction it is preserved. Otherwise it is suppressed. In some implementations more of discrete directions (0°, 90°, 135°, 45°) are used to be able to detect diagonal edges better.

Threshold Hysteresis Instead of a single threshold a double threshold is used. The high threshold and low threshold can be chosen as a parameter of the algorithm. Every value above the high threshold is considered an edge, everything below low threshold is discarded. The rest of values is either preserved or discarded based on their connection to the pixels already classified as edges. This can be achieved by a blob analysis [6] that is exploring its 8-connected neighborhood pixels.

The resulting edges can be seen in Figure 2.9.

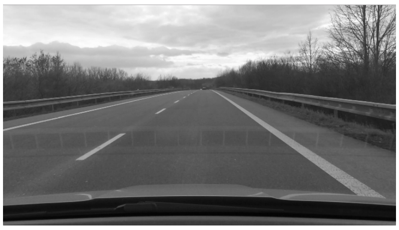


Figure 2.8: Original road image

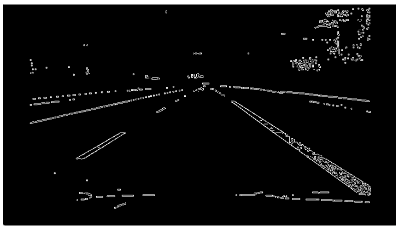


Figure 2.9: Canny Edges

2.4.2 Particle filtering

The Particle filter also known as or Sequential Monte Carlo (SMC) can be used as the feature extraction algorithm. The term "particle filters" was first coined in 1996 by Del Moral [11]. The filtering is often used for estimate of the internal state of system based on partial observations. The algorithm consists of three steps:

Initialisation The algorithm distributes given number of samples (particles) based on a given proposal distribution $q(x)$.

---

[6] Blob analysis is an algorithmic application of graph theory where subsets of connected components are uniquely labeled based on a given heuristic.

Importance sampling In importance sampling, the target distribution $p(x)$ is approximated, using samples drawn from a proposal distribution $q(x)$. For each particle the weight is calculated based on importance weight formula. The crucial idea is to update the particles so they will approximate the distribution in the next time step. The update is given by a process model. After that, weights are normalised to sum up to one. The process is then repeated in every iteration.

Resampling In practice, every update iteration leads to the degeneracy problem. Only small part of the weights have significant weight and rest have very small weights. This can be solved by resampling the particles from the set in accordance to the weights.

Particle filter for feature extraction In [12] updated version of the algorithm is used. First, the IPM is applied to mitigate effects of the perspective distortion. In initiation phase the particles are uniformly distributed in the bottom of the image. The particles are weighted according to a given weighting function which takes in account multiple hypotheses. This is needed as there can be more than one lane marking present in the image.

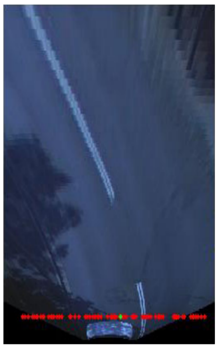


Figure 2.10: Initiation

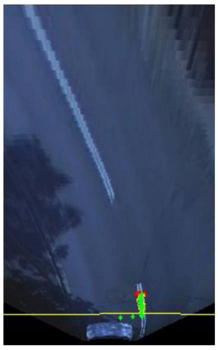


Figure 2.11: Progress

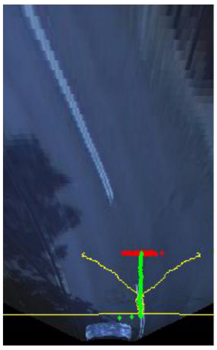


Figure 2.12: Low certainty

The particle filter is updated in relation to time axis which is replaced with the y axis. That is, particles are not being observed in time but in the bottom-up progress in one frame. The process model is set to define how the lanes move between updates in the image. Using defined Markov model the algorithm describes likelihood of the line to go straight, turn left or right. This allows to track curved markings in the image.

Additional observation model can be applied to improve robustness of the algorithm. This can use edge strength as well as color of the marking. The uncertainty of the estimates can be derived from the variance in the sample set. If the samples spread over a larger area the uncertainty rises. The result of the algorithm can be seen in figure 2.12. Red color highlights the particles, green are the estimates and yellow shows the level of uncertainty.

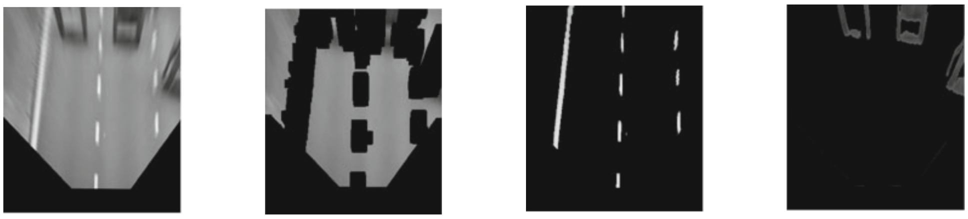
The advantage of the algorithm is ability to detect non-straight markings and its robustness. The use of inverse perspective mapping for the first step of the algorithm makes assumption about the position and rotation of the mounted camera. In addition laser was used to correct for uneven road surface and the changing pitch of the vehicle. That is, in order for algorithm to work without the careful installation and additional laser sensor changes would have to be made.

2.4.3 Recursive Bayesian segmentation

In [7] a robust algorithm to determine whether any pixel is a road, lane marking or an object (car) is used. It is based on Bayesian model that assigns likelihood models for each class. Likelihood models are described as parametric functions, according to the expected properties of the considered image features with respect to the defined classes. The pavement is described as monolithic surface with low intensity variance, markings as near-vertical bright stripes and objects as dark regions with lower intensity than the road.

The parameters for the likelihood function are estimated using Expectation Maximization algorithm [13]. This allows for the model to change based on the illumination. This however causes problems when abrupt changes occur e.g. when the vehicle is entering a tunel. A control mechanism that disable updating of parameters is implemented to mitigate these problems.

The method uses a robust algorithm which, apart from lane markings, can also detect road boundaries and other vehicles. The complexity of its implementation is quite high as it, apart from the Bayesian framework, also needs additional line (or curve) fitting to produce actual marking lines. It also uses IPM with all its advantages and disadvantages.



IPM Pavement Markings Objects

Figure 2.13: Output of Bayesian segmentation

2.4.4 Hough transform

Hough transform is based on 1962 patent of Paul Hough [14]. The algorithm takes binary image indicating the edges as an input. The *Canny Edge Detector* is most commonly used. The original version of the algorithm is used to identify lines in the image. Later there has been improvements in order to be able recognize other shapes like circles or ellipses. The originally proposed method used the equation in Figure 2.14 to describe a line where m is the slope of the line and b is the y-intercept.

$$y = mx + b$$

Figure 2.14: Original Hough line representation

This however allowed for unbounded space for vertical lines and change was proposed [15] to the equation that can be seen in Figure 2.15.

$$\rho = x \cos \theta + y \sin \theta$$

Figure 2.15: Hough line representation

Where θ is the angle of the line and ρ is the perpendicular distance from the origin. A line with $\rho = 10$ and $\theta = 45°$ is displayed on 2.16

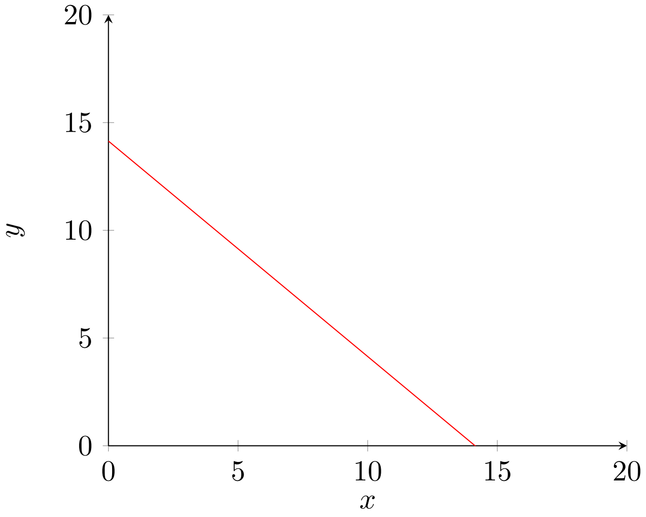


Figure 2.16: ρ and θ line representation

For each point a number of lines can be plotted going through it with diffent θ. A ρ is then computed using equation in 2.14. This is repeated for each point. The resulting graph is called a Hough graph. Example of the graph with two points can be seen in 2.17. The red is a ρ function of a point [2,2] and the blue of the a point [3,3]. Their intersection marks the θ and ρ of their connecting line.

A two-dimensional array called accumulator is used for collecting the results for each point. Every point votes on a different line as they intersect with

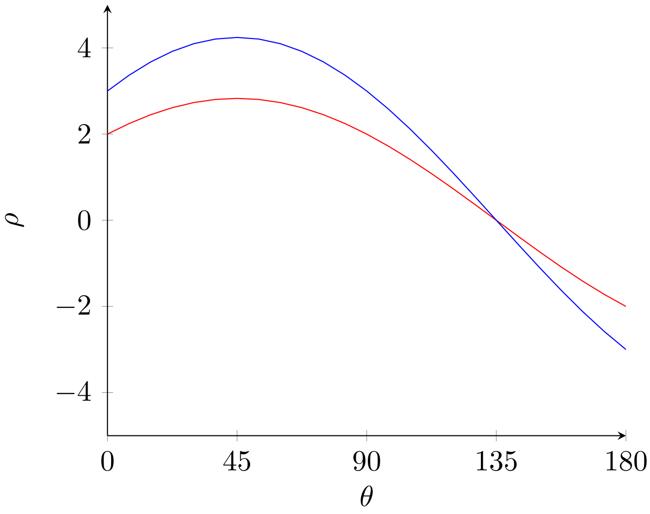
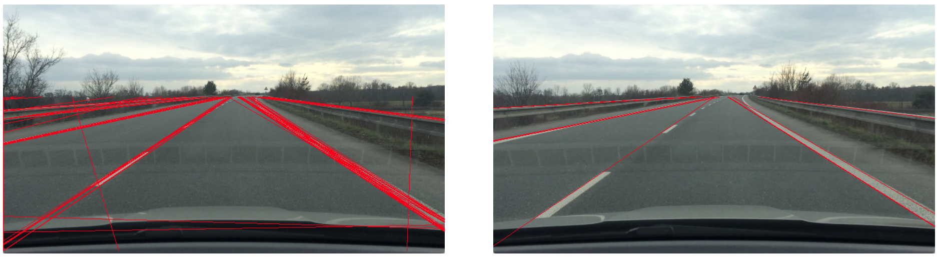


Figure 2.17: Hough space graph

each other in Hough space. After the process the accumulator is thresholded and local maximum is extracted. This matrix of (ρ, θ) are the found lines.

Probabilistic Hough Transform Probabilistic Hough Transform is just a variant of Standard Hough Transform. Instead of transforming all M points it selects only a subset of m points to be transformed to the *Hough space* where $m < M$. This will reduce the complexity of the voting stage. The value of m has to be carefully selected to be able detect all features. It usually depends on what minimum length lines we want to be able to detect. In the Figure 2.18 result of Probabilistic Hough Transform is displayed for high and low threshold.



Hough with threshold 20 Hough with threshold 100

Figure 2.18: Output of Hough transform

2.4.5 Conclusion

The *Hough Transform* has been chosen as the Feature Extraction algorithm. It does not require IPM and works reliably in the scenario of the road with marked lines. It does not detect curved lines in the distance. However, the markings close to the vehicle are almost straight even when the road is significantly curved. For *Lane Departure Warning* only the close distance lane marking position is necessary to determine car's position within the lane. *Hough transform* also produces line segments. Even though additional model fitting is necessary, it is less complex than in other methods described. At last, the *Hough Transform* is well known and used algorithm and there is plenty of optimized implementations available as well as of the *Canny Edge Detector*.

2.5 Model fitting

The module responsible for converting the features obtained from the feature extraction module into actual lanes is called the Model fitting module. The choice of the module heavily depends on the output of the feature extraction module. Since the Hough Transform was selected as the feature extraction module, a model should be fitted using set of line segments that have been extracted from the image.

2.5.1 Line filtering

The obtained lines have to be filtered to avoid false positives. There is a number of different methods that can be used for the filtering.

2.5.1.1 Direction

Lines can be filtered based on their direction. An assumption that the vehicle is moving parallel to the lane markings can be used. The lines can be filtered by given range $(a, b) = \{x \in \mathbb{R} \,|\, a, b > 0°\}, a, b < 180°$. In the Figure 2.19 the example of the filter can be seen. The lines that are within range are shown in green color. Notice that the far left and far right markings have not been included.

2.5.1.2 Vanishing point

In perspective, vanishing point (v_x, v_y) is a point where set of parallel lines intersect.It is a point of much interest in computer vision field as it corresponds to a 3-dimensional point in space. If it is correctly detected it should be located on the horizon in the middle of the road. The lines could then be compared by the distance at the intersection with the $y = v_y$ line. The vanishing point can be set statically in the center of the image or detected. The reliable detection

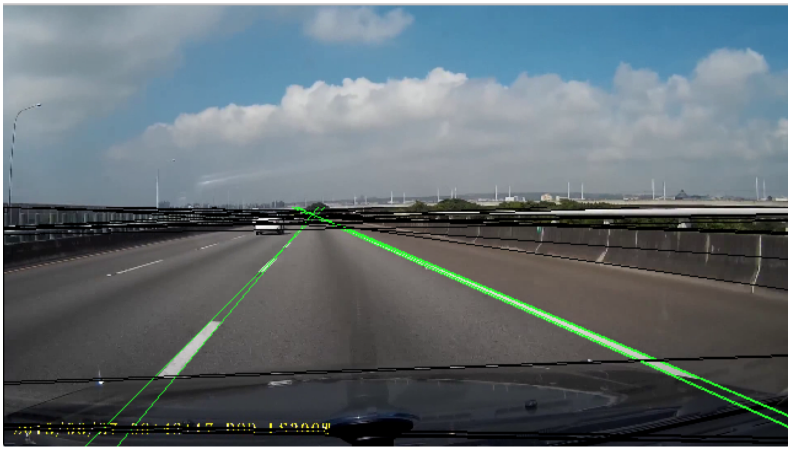


Figure 2.19: Direction line filtering

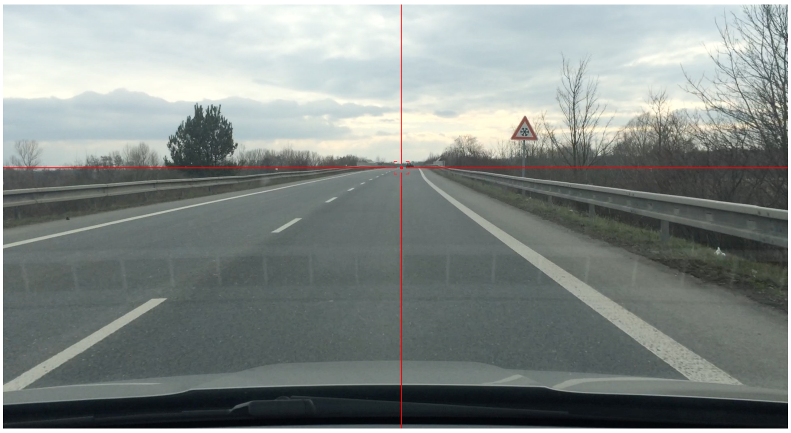


Figure 2.20: Vanishing point

of the vanishing point is however not a trivial problem. The example of the vanishing point is shown in 2.20.

2.5.1.3 Direction in the real world

The lane angle changes with the distance to the car in the x axis because of the perspective distortion. To mitigate this effect an *Image To World* model can be used to transform lanes to real-world space. With the real world model the lanes can be filtered to the ones that are parallel to the z-axis within given deviation. In the Figure 2.21 an image where all four lane markings are filtered is shown.

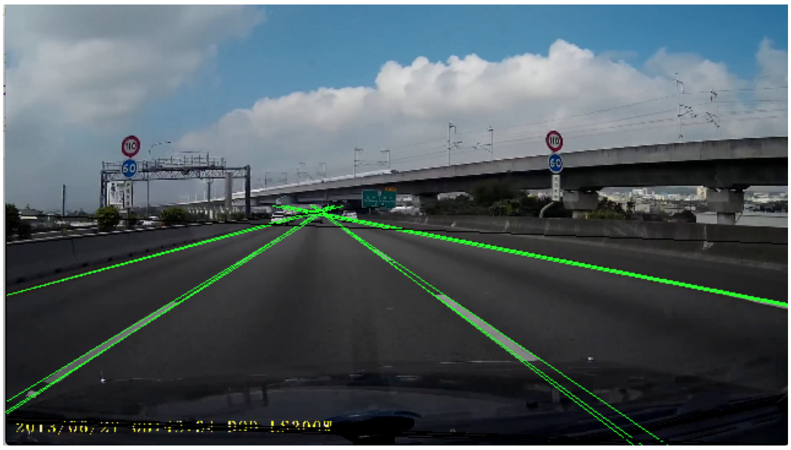


Figure 2.21: Real world direction line filtering

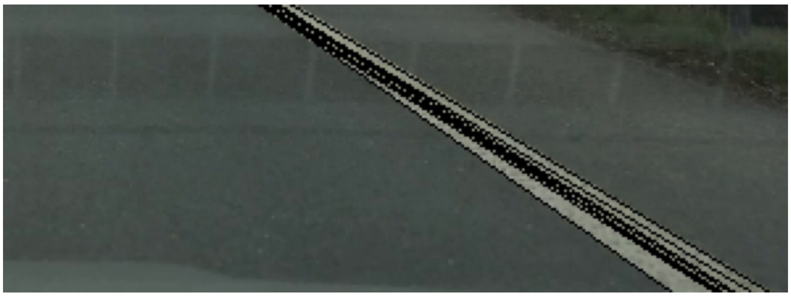


Figure 2.22: Marking with multiple lines

2.5.2 Line merging

There are often multiple lines found for single marking. Example of such marking can be seen in 2.22. It can be due to the marking bad quality, special type (such as double markings) or other causes. To simplify the model and lower further computational cost the most similar markings are merged together. The similarity of lines is computed based on the formula in Figure 2.1.

$$\delta = |f(s_1, \vec{n_a}, p_b) - f(s_1, \vec{n_b}, p_a)| + |f(s_2, \vec{n_a}, p_b) - f(s_2, \vec{n_b}, p_a)|, \tag{2.1}$$

$$f(y, \vec{n}, p) = (\vec{n_x} * p_x + \vec{n_y} * p_y - \vec{n_y} * y)/\vec{n_x} \tag{2.2}$$

Where $\vec{n_a}, \vec{n_b}$ being lines normal vectors, $p_a p_b$ points lying on the lines and s_1, s_2 a segment range for which the similarity should be measured. The lines with δ lower than certain threshold are then merged together.

2.5.3 Marking center detection

To be sure that the found line is a lane marking, additional step is needed. Otherwise road barriers, sideline of cars, shadows and other objects could be mistaken for the markings [16]. Additionally center of each lane marking can be obtained. This is useful as either the left or right edge of the same marking is often detected and creates noise in line measurements.

Lines are first sampled at given points from the top to the bottom along them. An one-dimensional array of pixel intensities is extracted from the gray-scale image where *Threshold filter* was previously applied. The pixels in the left and right direction from the line at sample point are saved to the array. Close up image of single sample can be seen in 2.23. In Figure 2.24 a graph of extracted intensities is shown. The width of the marking at a given point is then obtained using *Image to World* model. This will make sure that the width of more distant markings will be appropriately adjusted. New 1-D vector of intensities is then constructed. The graph represent and ideal profile of a lane marking and is called a template graph or data set. The 1-D vector is given a width of the lane marking l_w and margin on both sides $l_w/2$ wide . The resulting profile is illustrated in Figure 2.25.

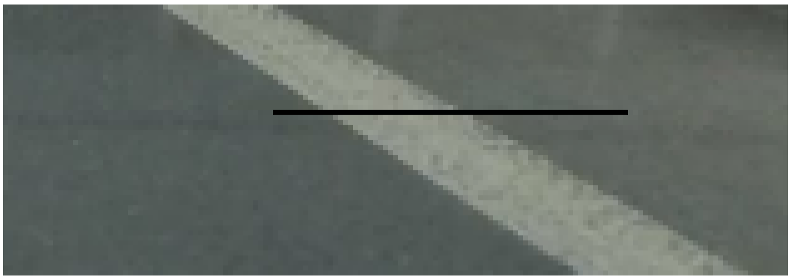


Figure 2.23: Lane marking 1-D sample

Cross correlation In signal processing, cross-correlation is a measure of similarity of two series as a function of the lag of one relative to the other. This is also known as a sliding dot product or sliding inner-product.

The formula defining the cross-correlation [17] is shown in Figure 2.3 where T is the smaller *template data-set* and I is the source data-set. The result is stored in the result matrix R. In this case a normalized version is more beneficial as the actual intensities can differ in various illumination. The formula used is shown in 2.4.

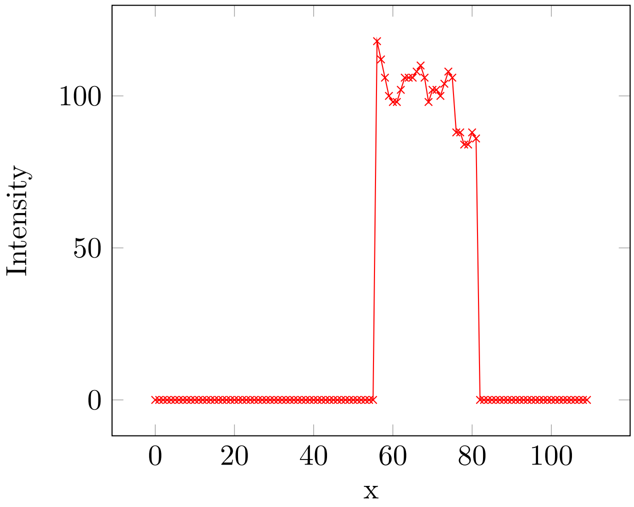


Figure 2.24: Marking intensities

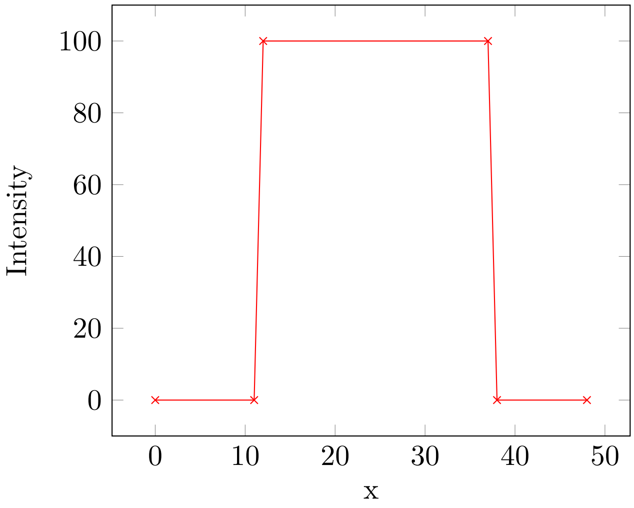


Figure 2.25: Marking profile

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x', y+y')) \tag{2.3}$$

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x', y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x', y+y')^2}} \tag{2.4}$$

Matching To identify the matching area, the template is compared against the source. For every pixel cross-correlation with the template image is computed. This will produce a matrix of correlation coefficients for each pixel. The maximum c_{max}, c_x for maximum coefficient and its location is then found. Finally the center of the line can be computed by $c_x + l_w/2$. The result can be seen in Figure 2.26

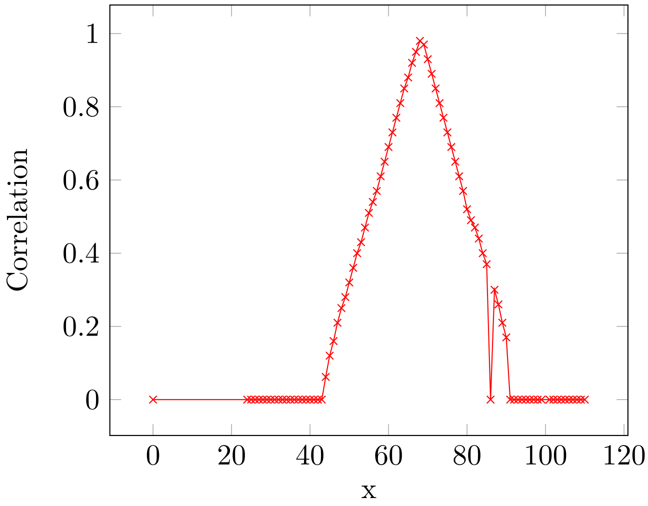


Figure 2.26: Result of cross-correlation

Thresholding By applying threshold on the maximum coefficient it is possible to eliminate false positives. This will help determine whether the line is genuine lane marking or different kind of object

Line fitting In the previous step a list of line centers is created for each line as illustrated in Figure 2.5.3.

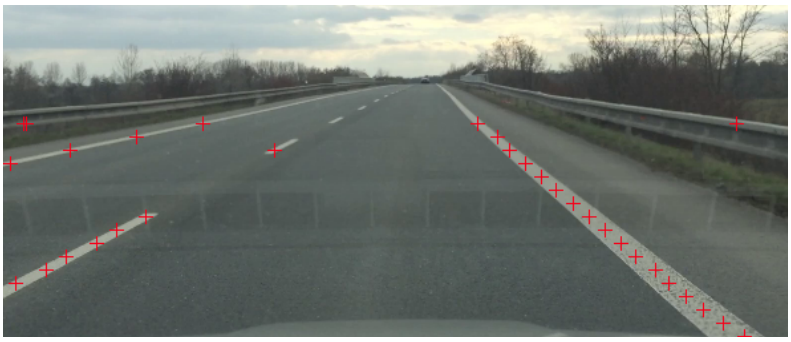


Figure 2.27: Markings center points

A line needs to be fitted through the points. If the points would be connected directly the line would have jagged shape. A typical solution for a given problem - fitting a line through set of points - is the Least square estimator [18]. Consider over-determined system in Figure 2.28.

$$\sum_{j=1}^{n} X_{ij}\beta_j = y_i, \ (i = 1, 2, \dots, m),$$

$$\mathbf{X}\boldsymbol{\beta} = \mathbf{y},$$

Figure 2.28: Linear least squares

Such systems usually do not have a solution (only when the points are in a perfect line) but the resulting line can be estimated using linear regression. The goal is to find the coefficients $\boldsymbol{\beta}$ which fit the equations "best," in the sense of solving the minimization problem as seen in Figure 2.5. S basically defines the problem as a square of the vertical distance from a data point to the given line function. Example of a line fitted with vertical distance (displayed as a blue line) is shown in 2.29. There are more ways how to calculate the minimum distance. An example of a line fitted based the perpendicular distance can be seen in Figure 2.30.

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} S(\boldsymbol{\beta}), \tag{2.5}$$

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{m} \left| y_i - \sum_{j=1}^{n} X_{ij}\beta_j \right|^2 = \left\| \mathbf{y} - \mathbf{X}\boldsymbol{\beta} \right\|^2. \tag{2.6}$$

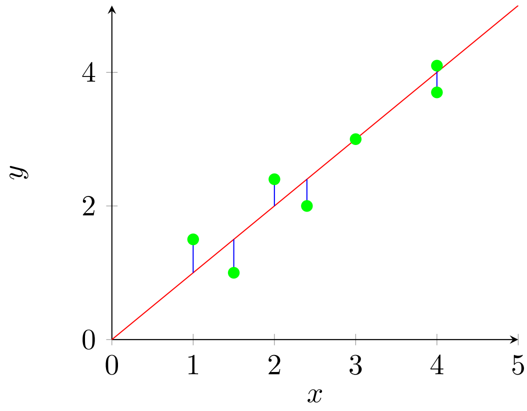


Figure 2.29: Example of a line fitted using vertical distance

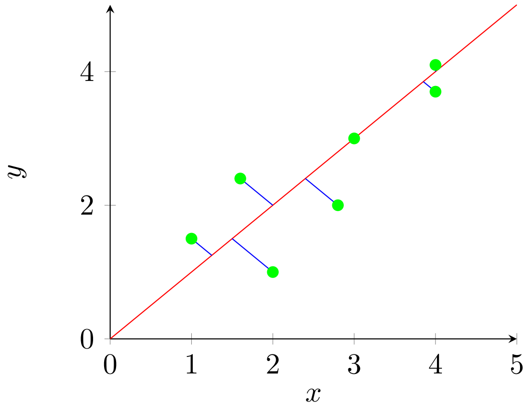


Figure 2.30: Example of a line fitted using perpendicular distance

The solution is unique given that the rows of matrix are linearly independent. Finally the coefficient vector is created in Figure 2.31

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

Figure 2.31: Coefficient vector for Least Squares

In a typical implementation, the algorithm iteratively fits the line using the weighted least-squares algorithm. After each iteration the weights w_i are adjusted to be inversely proportional to *distance function* [19]. In Figure 2.5.3 a line fitted through the sample points using *Least squares* is shown. After all the lines are fitted it is given a confidence measure which is a simple division of successfully sampled points and total number of points sampled for each line. This can give serve as base for recognizing dashed markings or rejecting lines with low confidence.

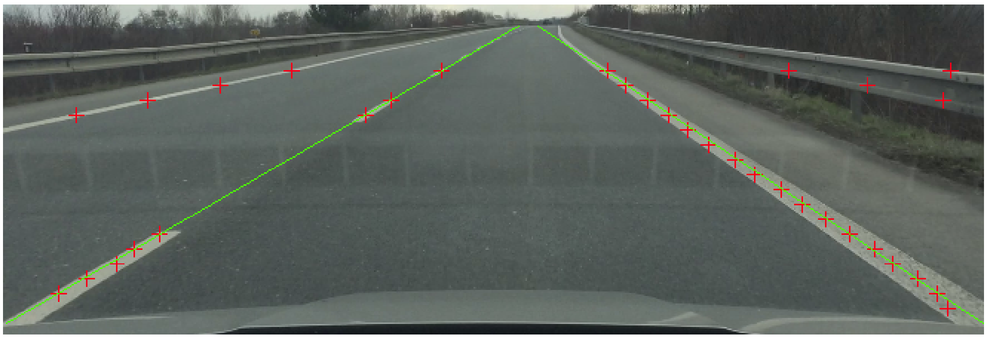


Figure 2.32: Markings fitted line

2.5.4 Current lane fitting

From the set of lines that have been found for a marking the current lane can be constructed. The image can be split in half and closest lines from left and right can be set as lane boundary. This works well for set of continuous full lines. For dashed lines, the features are not extracted in every frame and the detection would become erratic. In the multi-lane scenario lane would switch from the next lane to the current one back and forward.

Time integration module described in 2.6 is very helpful in keeping track of the lane width. This allows the algorithm to reject lines in case there is a large discrepancy between the line predicted from the *Time Integration* module. The lines that are closer to the predicted line are also scored higher so there is higher probability they will be selected. If there is only one line detected, the other side can be extrapolated based on the known lane width from the previous state.

Some of the algorithms [20] use a strong assumptions about the lane width to detect them. This algorithm does not make such assumption. It allows for the width to be changed slowly, but abrupt changes that could be caused by erroneous detection are discarded.

2.6 Time integration

To improve accuracy of the detection, to reject erroneous detections and to possibly reduce computational cost an information obtained from previous frames can be used. Detection accuracy is improved by predicting the detection and smoothing the result over time. By supplying good initialization of the model parameters the result can be also improved.

2.6.1 Particle Filter

A *Particle filter* (described in 2.4.2) can be used for lane tracking. In [21] the lane is represented as a sequence of points along its left and right boundary. Efficient lane tracking algorithm uses previously detected lane boundary points, adjusts them according to the vehicle's motion model, and then offsets them according to values obtained from the image.

2.6.2 Kalman Filter

Kalman filter [22] is an optimal estimator. It can be used to infer parameters of interest from indirect, inaccurate and uncertain observations. Because of its recursive nature, measurements can be processed as they arrive. Compared to the *Particle Filter* it is relatively simple and comes with low computational cost. The Kalman filter is a best estimator for linear systems and the motion model of the car can be in this case described as a linear system.

The formula in Figure 2.33 describes how current state x_k derives from the previous state x_{k-1}. The F_k is the state transition matrix, B_k is the control matrix, u_k is the control vector and w_k is the process noise.

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k$$

Figure 2.33: Current state in Kalman filter

State vector The current state of the system is saved in a vector called the state vector. There are more ways to track a state of the current lane. The left and right lines can be tracked in the image space using the state vector $x_k = (\rho_l, \theta_l, \rho_r, \theta_r)^T$ where ρ is the distance from the origin and θ is the line angle. This would, however, lead to a tracking lines individually so the lane geometry could not be compelled. Additionally, tracking in the image space does not deal well with perspective distortion.

More beneficial representation of the state vector is $x_k = (c_k, w_k, \dot{c_k}, \dot{w_k})^T$ [7]. Where x_k is the lane center and w_k is the lane width in real space. $\dot{c_k}, \dot{w_k}$ are the relative velocities. This allows the filter to track width of the lane which should remain mostly constant and its relative position to car. The lane can also be tracked even when only one marking is recognized as the c_k can be extrapolated based on previously measured width.

Transition matrix The state transition matrix applies each parameter at the time $k-1$ on the system state at the time k. It basically describes how should the system transition from one state to another. In the context of the lane tracking the Constant Velocity Model is assumed. The Constant Velocity Model [23] is calculated from the dynamics of the Ego-motion [24].

$$\begin{aligned} c_{k+1} &= c_k + \dot{c_k}.\delta_t \\ w_{k+1} &= w_k + \dot{w_k}.\delta_t \\ \dot{c_{k+1}} &= \dot{c_k} \\ \dot{w_{k+1}} &= \dot{w_k} \end{aligned} \tag{2.7}$$

where δ_t is the time step between the filter steps. The set of equations yields the transition matrix in Figure 2.34.

Control vector and control matrix The control vector contains any control inputs (steering angle, throttle setting, braking force) and control matrix is used to apply the effect of each control input parameter in the vector on

$$\begin{pmatrix} 1 & 0 & \delta_t & 0 \\ 0 & 1 & 0 & \delta_t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 2.34: Transition matrix

the state vector. There is no information about car control currently available. The control matrix can be, however, used to alter estimation in case of the lane change. When the car approaches the edge of the lane c_k is altered to transition the prediction for the destination lane. The control matrix is applied on the c_k as can be seen in Figure 2.35.

$$B_k = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 2.35: Control matrix

The control vector is supplied based on condition in Figure 2.36.

$$u_k = \begin{cases} w_k & \text{if } x < -\frac{w_k}{2} \\ -w_k & \text{if } x > \frac{w_k}{2} \\ 0, & \text{otherwise} \end{cases}$$

Figure 2.36: Control vector condition

Measurement The measurement vector c_k, w_k is the lane center position and its width. The measurement is made according to the formula in Figure 2.37 where H_k is a matrix mapping the state vector parameters into the measurement domain. The v_k is then a measurement noise and it is similar in its structure to the *process noise*. In this case the H_k is an identity matrix.

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

Figure 2.37: Measurement

The *Kalman filter* calculation consists of two stages: prediction and measurement update.

Prediction The priory state is predicted in the Figure 2.8 based on the transition and control matrix and the previous state. Its covariance matrix $\mathbf{P}_{k|k-1}$ is predicted in Figure 2.9 where Q_k is the covariance matrix of process noise. The priory state can be used to focus feature extraction algorithm to a certain part of the image and thus save computational cost.

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k\mathbf{u}_k \tag{2.8}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1|k-1}\mathbf{F}_k^{\mathrm{T}} + \mathbf{Q}_k \tag{2.9}$$

Measurement update In the update step the current measurement is combined with the *priory state* to refine the state estimate. The innovation residual y_k and covariance S_k is described in 2.10 and 2.11 where R_k is observational noise covariance matrix. The Kalman gain K_k described in 2.12 is a function of the relative certainty of the measurements and current state estimate. With a high gain, the filter places more weight on the measurements and with low gain the filter places more weight on its predictions. This way the estimation changes based on how much they were successful in the past. In 2.13 and 2.14 the updated (posteriori) state estimate and covariance $P_{k|k}$ is described.

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1} \tag{2.10}$$

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k \tag{2.11}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{2.12}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \tag{2.13}$$

$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{2.14}$$

2.7 Image to World Correspondence

The understanding of the geometrical relation between the image and the road is very useful in many parts of the Road Lane Recognition algorithm. In *IPM* it is used to transform the original image to a bird-eye view of the road. It can also be used to track the lanes in real world coordinates instead of the image coordinates. Another use of the real-world model is for *ROI* estimation as the horizon can be derived from it.

2.7.1 Homogeneous coordinates

For the representation of the point in the real-world space homogeneous coordinates are used. The homogeneous coordinates use $\mathbb{R}^{n+1}$ vector to represent a point in $\mathbb{R}^n$ space. The conversion between homogeneous and euclidean coordinates is given in Figure 2.38.

$$\begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_n \\ w \end{pmatrix} \leftrightarrow \begin{pmatrix} x_1/w \\ x_2/w \\ ... \\ x_n/w \end{pmatrix}$$

Figure 2.38: Homogeneous coordinates

where $\leftrightarrow$ symbol indicates that both vectors occupy the same point in space. The homogeneous coordinate vector is unaffected in scale if multiplied by a non-zero constant. The coordinates become useful when used with *affine transformations*. The homogeneous coordinates can be transformed by left multiplying with the transformation matrix.

2.7.2 Affine transformation

Affine transformation is a function which preserves points, straight lines and planes. The set of parallel lines will be still parallel after an affine transformation. It is defined as composition of two functions linear transformation and translation in Figure 2.39. That can be simplified by using the Augmented matrix in Figure 2.40.

$$\vec{y} = f(\vec{x}) = A\vec{x} + \vec{b}.$$

Figure 2.39: Affine transformation

$$\begin{bmatrix} \vec{y} \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & A & & \vec{b} \\ 0 & \ldots & 0 & 1 \end{array} \right] \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}$$

Figure 2.40: Augmented matrix

For the the real-world model we will be using translation, scale and rotation transformations. Examples of such transformations in $\mathbb{R}^2$ are in Figures 2.41, 2.42 and 2.43.

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 2.41: Translation

$$\begin{bmatrix} w & 0 & 1 \\ 0 & h & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 2.42: Scale

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 2.43: Rotation

2.7.3 3D Projection

The goal of 3D projection is to transform a 3D point in from $\mathbb{R}^3$ to the 2D space $\mathbb{R}^2$ that represents the image screen. In the context of lane recognition the road and its lanes are the objects that need to projected. In Figures 2.44, 2.45 a road in 3D space and its projection is illustrated.

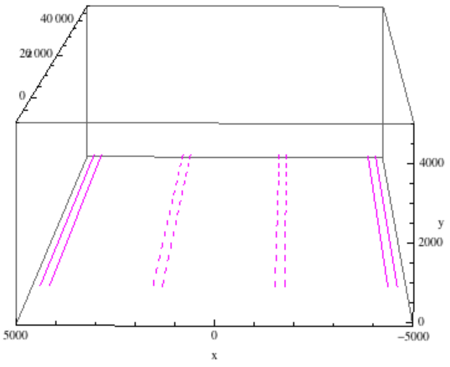


Figure 2.44: Road in 3D space

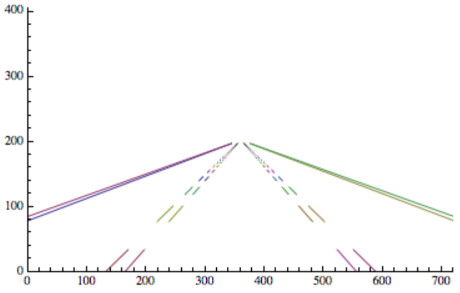


Figure 2.45: Projected road

2.7.3.1 Orthographic projection

Orthographic projection ignores the fact that the objects appear smaller as the distance grows bigger. It is a projection of a single view onto a drawing surface in which the lines of projection are perpendicular to the drawing surface. It can be used for birds-eye view, profile or cross section. But because we work with camera input we need to be able to describe the road as it would be recorded photographically.

2.7.3.2 Weak perspective projection

Weak perspective projection is based on orthographic projection. It has, however, a scaling factor defined to scale the objects that are closer to appear bigger. This approximation is performing reasonably well when the field of view is small and the depth of the object is small compared to distance to the camera. This is however not the case of the markings that will be long and stretching from the front of the camera to the horizon.

2.7.3.3 Perspective projection

To accommodate the true perspective in the scene a Perspective projection can be used. The projection converts 3D point (d_x, d_y, d_z) to a 2D point (p_x, p_y). The x and y coordinates are divided by its z component and multiplied by a *focal length* f of the camera in 2.46. The objects that are closer to the origin will also appear bigger in the projected image. This can also be achieved by multiplying the homogeneous coordinates with a projection matrix described in Figure 2.47. To normalise resulting coordinates p_x, p_y they have to be divided by the w component.

$$p_x = d_x/d_z * f$$
$$p_y = d_y/d_z * f$$

Figure 2.46: Perspective projection

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{pmatrix}$$
$$\begin{pmatrix} p_x \\ p_y \\ p_z \\ w \end{pmatrix} = P \begin{pmatrix} d_x \\ d_y \\ d_z \\ 1 \end{pmatrix}$$

Figure 2.47: Perspective projection matrix

2.7.4 Pinhole camera model

The simple perspective projection would be sufficient if the camera would be located at the origin with zero rotation in relation to all axes. In reality the camera would be typically positioned about a meter above the road and could be rotated along the y and x axis. The pinhole model can be used for mathematical description of the camera in space. The model is used in various scenarios in computer graphics where camera needs to be manipulated. The model can be broken down into the extrinsic and intrinsic part.

Extrinsic The extrinsic part describes camera's position and rotation. In Figure 2.48, 2.49 a top and side view of the camera on the road is illustrated. (x_c, y_c, z_c) describes the camera position in space. The rotation of the camera is described by yaw (γ) and pitch (θ). In Figure 2.50 an extrinsic matrix of the camera is shown. It is a multiplication of camera translation, yaw-rotation and pitch-rotation matrices.

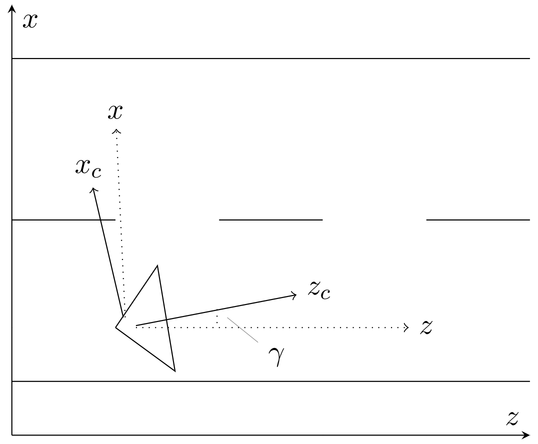


Figure 2.48: Camera top view

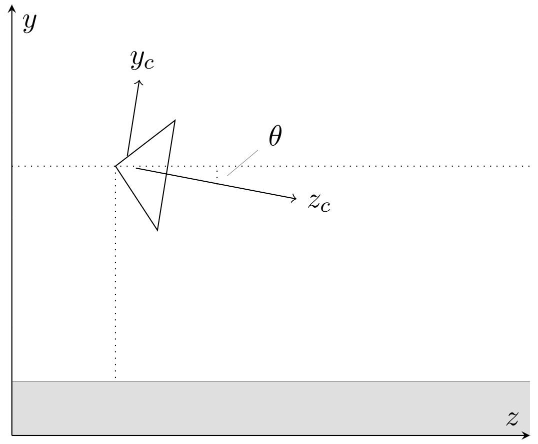


Figure 2.49: Camera side view

$$T = \begin{pmatrix} 1 & 0 & 0 & -cx \\ 0 & 1 & 0 & -cy \\ 0 & 0 & 1 & -cz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & Cos(\gamma) & -Sin(\gamma) & 0 \\ 0 & Sin(\gamma) & Cos(\gamma) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} Cos(\theta)) & 0 & Sin(\theta)) & 0 \\ 0 & 1 & 0 & 0 \\ -Sin(\theta) & 0 & Cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$E = T.R$$

Figure 2.50: Extrinsic matrix

Intrinsic The intrinsic model describes the geometry of the inside of the camera. It is illustrated in Figure 2.51. The image plane is the 2D image we would see as the projection of the 3D world. The point P is projected as point Q on the image plane. The origin O is where the camera aperture is located. The aperture is imagined as an infinitely small point. That is, the pinhole camera model does not take in account any lenses used or any distortion in the image. The 3D plane that intersects the x and y axis is called a *principal plane* or front of a camera. The focal length f is a distance between the *image plane* and the *principal plane*. The world is projected through the origin O. The projection line illustrated in blue color connects 3D point P and projected point Q. The projection line always comes through the origin. The resulting image is therefore inverted and have to be rotated by 180° which corresponds to how the real-world pinhole camera works.

The intrinsic matrix is described in 2.7.4. The W, H is the size of the screen in pixels and w, h is the size of the sensor in millimeters. The origin is translated to the center of the screen because that is where the real aperture is located. The image have to be flipped along the y-axis because in most of computer image formats the Y position is counted from the top to the bottom. The image is also scaled so it will correspond to the real-world coordinates.

Projection with camera pinhole model It can be noticed that the intrinsic and extrinsic matrix have different dimensions. That is because the extrinsic matrix is applied pre-projection and intrinsic post-projection. The projected 3D point in homogeneous coordinates is first multiplied by the extrinsic matrix E and projection matrix P. The resulting 2D point is then multiplied by the intrinsic matrix I.

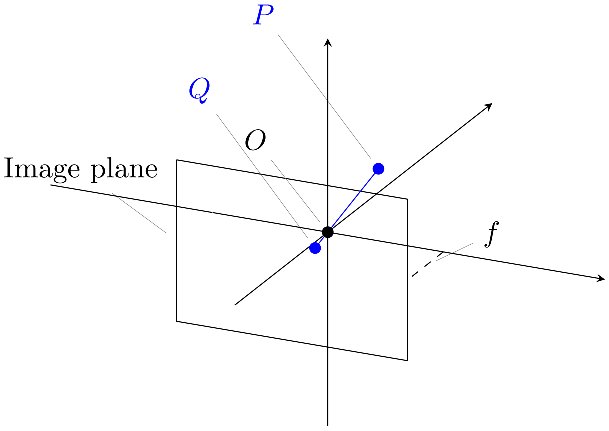


Figure 2.51: Intrinsic pinhole camera model

$$I = \begin{pmatrix} W/w & 0 & W/2 \\ 0 & -H/h & H/2 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 2.52: Intrinsic matrix

2.7.5 Inverse perspective projection

For the actual application the inverse of the projection is needed. In the original perspective projection (in our case the camera capture of the road) there is an information lost about the z-coordinate of captured objects.

We can use an assumption that the objects lie on the road plane. That is true for the lane markings. The way to do it is to imagine *projection line* and road plane at $y = 0$. The point where they intersect is the projected point in space. The plane can be defined by a normal vector $\vec{n} = (0, 1, 0)$, a point lying on it $r = (0, 0, 0)$ and equation in Figure 2.15. The projection line can be defined in parametric form in Figure 2.16. It comes through the origin $(0, 0, 0)$ and the projected point p_i. Finally intersection is found in Figure 2.17 and can be further substituted to the form in Figure 2.18

$$\vec{n} \cdot (p_o - r) = 0 \tag{2.15}$$

$$p_i t = 0 \tag{2.16}$$

$$\vec{n} \cdot (p_i t - r) = 0 \tag{2.17}$$

$$p_o = \frac{n \cdot r}{n \cdot p_i} p_i \tag{2.18}$$

This can be translated in linear transformation as the inverse projection matrix. The matrix is multiplied by (p_x, p_y) with f as its z coordinate.

$$\begin{pmatrix} dx \\ dy \\ dz \\ w \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{n_x}{\mathbf{n \cdot r}} & \frac{n_y}{\mathbf{n \cdot r}} & \frac{n_z}{\mathbf{n \cdot r}} & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ f \\ 1 \end{pmatrix}$$

Because the plane should not be moving with the camera its position and rotation should be adjusted. It is done by adjusting planes normal vector $\vec{n}$ and point r as described in Figure 2.7.5. The *extrinsic rotational matrix* R and *extrinsic translation matrix* T is used for the adjustment.

$$\vec{n} = R.(0, 1, 0)$$

$$r = T.(0, 0, 0)$$

Figure 2.53: Plane translation and rotation

CHAPTER 3

Implementation and testing

As part of the thesis I developed a prototype of the Vision-based driver assistant. The goal was to implement a reliable lane recognition and tracking algorithm and on top of that a simple interface that will display current lane and issue warning when driver approaches the lane boundary. The implementation detail of the prototype are discussed in the next section.

3.1 Computer vision framework

Computer vision is a growing field of computer science and there are multiple libraries to choose from. It is possible to develop the prototype without a *CV* framework. It would give a developer more control over the algorithms used. However, *CV* frameworks usually offer highly optimized versions of the algorithms. To write a comparably fast algorithms on my own would be very time consuming. Thus, a *CV* framework have been selected. There are multiple parameters that are desired:

- To have vast array of algorithms already implemented that are commonly used in image processing such as Edge detection, Hough Transform, filtering, template matching etc.
- To contain functions to manipulate video from different sources (file, video-camera).
- To be able to display higher GUI such as text, lines and polygons on top of the video and controls such track-bars, buttons etc..
- To provide basic mathematical entities and operators such as matrices, vectors, normalization, matrix multiplication etc..
- To be highly optimized

- To run on number of devices and platforms including Windows and UNIX based systems, mobile platforms and possibly embedded systems[7].

3.1.1 VXL

VXL (the Vision-something-Libraries) is a collection of C++ libraries designed for computer vision research and implementation [25]. The library provides support for numerics, imaging, geometry, streaming, basic templates and utilities. It provides a good base for further development of own algorithms. There isn't, however, much of higher algorithms available or are not well documented. The library is implemented in C++.

3.1.2 ccv

The ccv is minimalist and well organised Computer vision library [26]. It offers a handful of useful algorithms such as the Real-Time Object Detection, Face detection, Pedestrian detection and many more. Each algorithm is mostly based on a single paper. It solves some specific problems but does not provide the basic building blocks for own implementations. It does perform well in the context of mobile applications by caching potentially redundant operations such as matrix multiplications, image pyramid generation, color space conversion etc.. while maintaining a clean function interface.

3.1.3 libCVD

libCVD is C++ library designed to be easy to use and portable for fast video saving, loading and display [27]. The library is designed in a loosely-coupled manner, so that parts can be used easily in isolation if the whole library is not required. It offers compatibility with a wide array of platforms. There is also number of algorithms implemented but it is lower compared to *OpenCV*. For example, there is an implementation of *Canny Edge Detector*, but no *Hough Transform.* The library is still actively developed. Compared to *OpenCV* which currently has 563 contributors the team behind libCVD is just 5 people.

3.1.4 OpenCV

OpenCV is an open source library for a computer vision [28]. It contains vast array of algorithms related to *CV* from preprocessing algorithms (Canny, Hough) to more advanced object detection, structure from motion algorithms, filters and many more.

OpenCV was designed for computational efficiency and with a strong focus on real-time applications. According to the website it has more than 2500 optimized algorithms, 47 thousand people of user community and estimated number of downloads exceeding 9 million. It is by far the most popular library in the CV field and it is used by majority of CV researchers. It is written in C++ and has API's for multiple languages.

---

[7] An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints

3.1.5 Conclusion

OpenCV was selected as the *CV* framework. The main reason was its wide adoption and number of algorithms for image processing and also other algorithms that might become useful when new features are added in future.

3.2 Implementation

There are several choices of languages in which it is possible to write code with OpenCV. There is the C++ which is a language the OpenCV is written in, but also less comprehensive C API. There are also bindings for Python and Java plus number of wrappers in other minor languages. Especially python is very popular in the field of computer vision. It provides simple API and is used to quickly implement new algorithms and adjust them. The performance is not much an issue as the OpenCV-Python provides simply a wrapper to C++. But when a custom method is implemented in python the performance can be lowered. The python code is also not simply portable to mobile devices. For that reasons the prototype is written in C++. The OpenCV 2.4.9 is used.

The typical implementation in OpenCV C++ is a single function algorithm divided to function blocks by comments. That can be beneficial for performance reasons, but I found it very hard to orient in. Instead, the code is divided into classes based on single responsibility principle [29]. Dependency injection pattern is used for managing dependencies. In following subsections parts of the prototype are described.

3.2.1 The Basic Image Container

In C++ version of the OpenCV, a structure `cv::Mat` is used for storing the data of single frame from the input video. The advantage of the `cv::Mat` is that there is no need for the developer to manage the memory. Mat consist of two data parts: the matrix header (containing the information about the size of the matrix, the type of the matrix, address, etc...) and a pointer to the matrix containing the pixel values. The types `cv::Mat` and `cv::Mat&` can be then used almost interchangeably. The structure can also be used for simple cropping without copying the actual pixel values. The copying should be avoided as it is an expensive operation.

3.2.2 Image capture

`cv::VideoCapture` module is used for handling the video input. It can receive video from number of resources including source files of various formats, web-cameras or phone video-camera. Additionally I wrote a wrapper class `VideoInput.h` that resizes the image to a bounding rect and provides other useful functions. The output is in form of matrix `cv::Mat`, with rows and columns representing the position of pixels in the original image counted from the top left corner.

3.2.3 Image pre-processing

The video is first converted to grayscale matrix using `cvtColor`. The Marking width filter 2.3.4 is then applied to highlight the markings in `MarkingWidthFilter.h`.

3.2.4 Feature Extraction and Model Fitting

The feature extraction and model fitting is provided by the class `FeatureExtraction.h`. First a `cv::Canny` is used for edge detection. The mean value μ of the grayscale image is calculated and the upper and lower threshold are set according to:

$$t_{high} = \alpha * \mu; t_{low} = \beta * \mu; \alpha > \beta$$

where α and β are constants based on empirical observation. This way the canny edge detector can adjust to the illumination changes.

On the output of the Canny a Probabilistic Hough Transform `cv::HoughLinesP` is applied. ρ is set to 1 and θ to 1°. The output of the algorithm are line segments. Next step is to transform them to the real space using `ImageToWorld.h` module. The resulting lines are then filtered and lines that are not near-perpendicular to the x-axis are rejected. The lines are also filtered based on their similarity to neighbouring lines. This way there is a smaller number of lines sampled in the next step and computational cost is lowered. Even if the proper edge of the marking was merged with a non-edge, the sampling process will find the center of the marking anyway.

The sampling first divides the line to a number of samples along its y-axis and fills a small `Mat` with one dimensional array of the sample point neighbouring pixels. The template `Mat` is also created with presumed width of the lane at the sample point. The `cv::matchTemplate` algorithm is then called on the one dimensional array with a template `Mat`. The `CV_TM_CCORR_NORMED` method is selected to compute cross-correlation. The result is a matrix of correlation coefficients. `cv::minMaxLoc` is used to find the maximum coefficient which is saved to a vector of center points.

The center points are then fitted to a line with `cv::fitLine` algorithm using least squares distance method. The sampled lines are collected and again filtered based on their confidence and angle. The current lane is fitted from them according to the lane fitting algorithm described in 2.5.4 and saved as a structure from `Lane.h`.

3.2.5 Lane Tracking

The `cv::KalmanFilter` is used for tracking the current lane. The state `Mat` holds lane center position x_k, width w_k and is initially set to arbitrary initiation value (e.g. (0, 3m)). Additionally information whether the left and line markings are dashed are saved in the state. Transition matrix and control matrix are set according to 2.6.2. Before the *feature extraction* step a `predict` method is called to obtain a lane prediction. Additionally a control vector is supplied in the predict step to update estimate in case the lane was changed. The prediction is used to better estimate lane position. After the current lane has been extracted `correct` method is called to do a measurement update.

3.2.6 Image to world

The image to world is perhaps the most complex module. An OpenCV implementation exists for a projection matrix `cv::sfm::projectionFromKRt` in Structure from Motion module. There is, however, no inverse perspective implementation that would fit scenario of the road plane and I found it would be better to implement the whole model myself. To be sure that the model is correct I first prototyped it in Mathematica[8] and then used the matrices for the C++ implementation. The `ImageToWorld.h` is given a camera position, yaw, pitch ,cameras screen size, sensor size and focal length. The matrices are generated and cached. They are only re-cached if one of the parameters change. The point translation is then a simple matrix multiplication of the point homogeneous coordinates. The `toWorld` method first multiply the coordinates of 2D Point extracted from the image by the inverted *intrinsic matrix*, inverted *perspective projection matrix* and then inverted *extrinsic matrix.* The `toImage` first multiply 3D point of real space coordinates by the *extrinsic matrix*, *perspective projection matrix* and *intrinsic matrix.*

3.2.7 Inverse Perspective Mapping

The IPM is implemented as a part of the prototype. Simple library [30] has been used to compute the affine transformation needed to translate the original video to birds-eye view based on the *ImageToWorld* module. The code is included in `InversePerspectiveMapping.h`. The IPM itself was not actually used as input for the *Feature extraction* module, because the ImageToWorld was developed later and previously developed Feature extraction algorithm would have to be significantly changed. But the IPM was added to the project anyway as it may become useful in future development.

---

[8] Mathematica is a symbolic mathematical computation program, sometimes called a computer algebra program, used in many scientific, engineering, mathematical, and computing fields.

3.2.8 Renderer

To debug outputs of algorithms and display the result on screen a class `Renderer.h` is used. The instance can be passed to any of the modules. Lines, points or text can be projected on top of the original image. For that purpose OpenCV drawing functions are used. The interface is designed to be simplistic and not to distract user from driving. The current lane is represented by a single dashed green line with a text information about the car's deviation from the center of the lane. The prototype also features a debug mode where a grid representing a road plane is displayed for easier configuration.

3.2.9 Settings

The `SettingsWindow.h` is used to update algorithm presets in real time. This is especially useful for calibrating the *ImageToWorld* module. The debug mode can be also turned on and off using the controls. The OpenCV highgui trackbars were used to provide a simple slider interface to the user.

3.2.10 Lane departure warning

The output of the lane recognition algorithm is used to determine width w_k and center of the lane x_k in relation to the car position. The lane center deviation in percent can be calculated as

$$d_k = |x_k/w_k * 100|$$

The lane departure warning is issued once it exceeds given threshold. The warning is visual and auditory to make sure the driver will notice it.

3.2.11 Mobile application

The RoadAssistant application in iOS was created to explore uses of the algorithm in the context of mobile applications. OpenCV provides framework that can be linked natively to the iOS Xcode[9]. project. The camera of the smartphone can be accessed using `CvVideoCamera` from the OpenCV `cap_ios.h` module. The `Mat` of the current frame is extracted and used by the Lane Recognition Algorithm.

The main issue of scaling down the application to the smartphone is the performance. There are several ways to optimize the algorithm.

---

[9] Xcode is Integrated development environment for the iOS and MacOS development provided by Apple

Region of Interest Based on the *Image to World* module, a specific region of interest, from front of the car to distance about 10m is selected as the Region of Interest. The detection algorithm is then applied only on parts of the image within this distance. Anything further is also not very reliable source of lane markings as it can be obstructed by passing cars.

Frame rate reduction Typical smartphone video frame rate is 29.97 frames per second. The algorithm can reliably work with less than that. The OpenCV camera input make it easier as the algorithm is not fed a captured video but takes new frame from camera as soon as the last one was processed. This way the FPS is reduced automatically.

Hough transform The *Hough transform* is the most time-consuming step of the algorithm. Once the first estimation of the lane is made by the *Hough transform*, the *Line sampling* algorithm can be used along the lane prediction made by *Kalman filter* instead. This optimization, however, lead to significant decrease in the algorithm accuracy and wasn't used in the end as the application performed well even without it (see 3.3).

3.3 Results

The prototype have been tested on various footage taken either on highway or country roads both in Czech Republic and Taiwan. The tested scenarios are mostly roads with visible lane marking on both sides. The evaluation of the algorithm is problematic, due to the lack of accepted test protocols, performance metrics as there is no ground truth[10] information available. Most of papers in the field of road lane recognition field then choose their own metric [6].

Because the lane departure feature depends solely on the proper lane detection, the error of the detected center-line x_k, and width w_k was selected as the measure in this thesis. The current lane is first manually found and saved every 30 frames (or 1 second). A measuring program was created on top of the road assistant that takes a mouse as an input. By setting up the *Image to World* model the program allows to measure the lanes in the same domain as the algorithm. Left and right lane boundary is then marked by mouse-clicking on the line position. The process takes a lot of time and can hardly be applied on a larger scale test. It is, however, currently the only available way to found a ground truth.

The x_k and w_k is compared to the algorithm output and from the difference is computed the error. The resulting figure is an average of the errors found in each frame. The scenarios consist of various footage taken at different time with different cameras. Part of them were taken by the author of the thesis with an iPhone 5S camera and part were provided by NTUT Department of Computer Science with unspecified camera. The Figure 3.1 shows the errors in highway and country road scenario.

---

[10] Ground truth is a term used in various fields to refer to information provided by direct observation as opposed to information provided by inference

Parameter	640x480	320x240
x_k average error	18,0 cm	23,4 cm
w_k average error	12,4 cm	16,2 cm
Frames sampled	171	
Time sampled	5130 s	

Table 3.1: Error in the highway and country scenario

In more complex urban environment (villages, urban roads) the detection is more difficult because one or both of the markings are often missing. Although the algorithm was not designed for this scenario, as long as at least one marking is present, it performs reasonably well. The error of detection is higher and is shown in the Figure 3.2.

Parameter	640x480	320x240
x_k average error	45,7 cm	48,3 cm
w_k average error	48,3 cm	54,2 cm
Frames sampled	82	
Time sampled	2460 s	

Table 3.2: Error in the urban scenario

The performance of the prototype differs with the desktop and mobile version and also resolution that is used for capturing the video. In the table 3.3 the average FPS per device is shown for a sample video. The performance drops rapidly with increasing resolution, especially in the mobile phone. But for the resolution of 352x288 the application recognize lane in near-real time. The tests have shown that the accuracy has decreased with input in smaller resolution. However, the difference is minimal and the lane departure warning is still issued accurately.

Device	Resolution	FPS
MacBook Pro 2.6 GHz Intel Core i5	640x360	15.4
	320x240	29.3
iPhone 5S	640x480	8.1
	352x288	25.6

Table 3.3: The performance in FPS

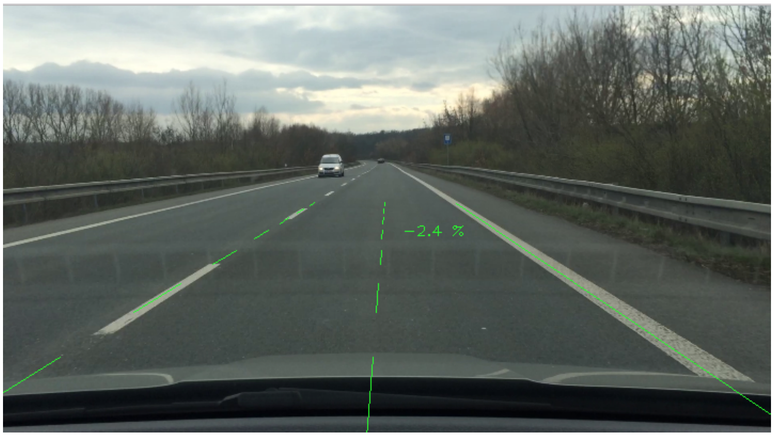


Figure 3.1: Example of successfully detected lane in the country road

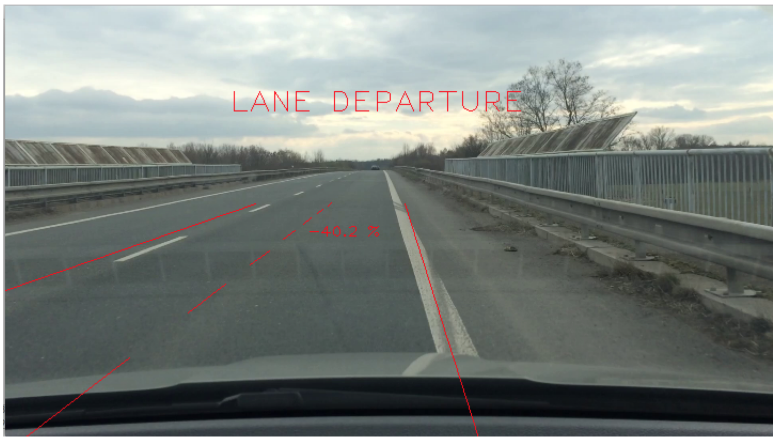

LANE DEPARTURE
−40.2 %

Figure 3.2: Lane departure warning

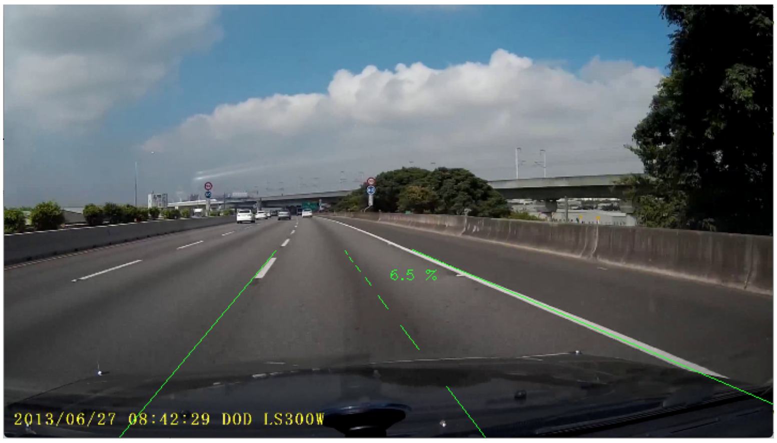

6.5 %
2013/06/27 08:42:29 DOD LS300W

Figure 3.3: Detection in highway multi-lane scenario

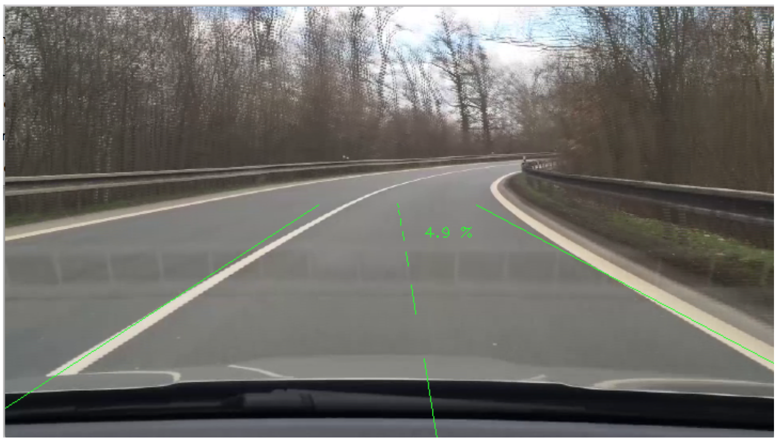


Figure 3.4: Detection of curved marking

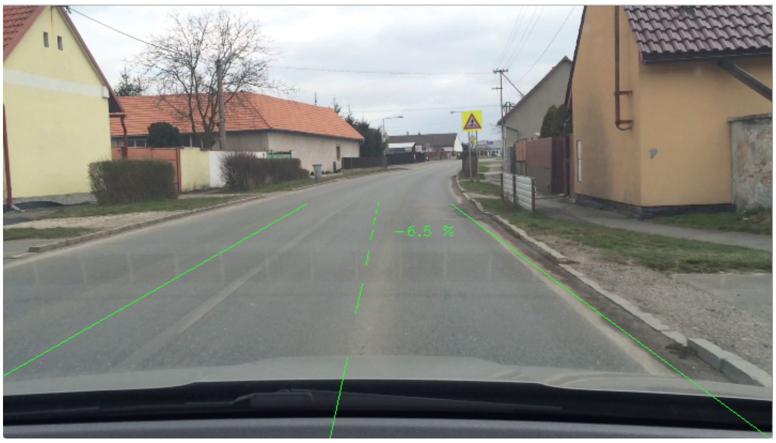


Figure 3.5: Example of erroneous detection caused by faded left marking

Chapter 4

Conclusion

The goal of this thesis was to create a prototype of a vision-based assistant. The underlying task of developing reliable road lane recognition algorithm turned out to be the main portion of the thesis. Even though the problem is well known, there is still small number of actual implementations that work in real-time available. The usual practice is to test the algorithm in higher languages such as Matlab [11] where efficiency is not a primary concern.

When I chose this problem to be my thesis I didn't have any background in the computer vision. The study of the different methods and algorithms took most of the time when I was working on the thesis. There were many times when I tried other techniques that are not even mentioned in the text because they didn't perform sufficiently well.

The prototype created works very well in the highway scenario and on the country road with clearly marked lane markings. The measured center-line accuracy is well under 25 cm. In the curves algorithm detects lane with good accuracy even though it is based on detection of straight lines. The dashed marking detection was also challenging as the markings are not detectable in every frame and time integration have to be used. The Kalman filter was effectively used for such situations and also to smoothen the lane output. Its control matrix was used to handle change of the lane in simple but effective way. The lane departure warning is reliable once the lane is detected and displays the information for the driver in a simple interface.

A mobile application was created based on the prototype. Using lower resolution it performs well even in real-time. That allows for the algorithm to be used on various portable devices. Because it is developed in C++ it is not only limited for iOS but can be used on nearly any other mobile platform and even some embedded systems.

At last, the prototype provides good base platform for another practical application in the field. For example the key part of the prototype - *Image To*

---

[11] MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language

World model have many uses in the road-related scenarios. The information about creating the model is scarce and to create the *Inverse Perspective Projection* was surprisingly complicated. The hope is, this thesis could be used in future works where similar problems could be avoided.

4.1 Possible future work

There is a large number of possible improvements that are equal to the large scale of the problem. In the following sections some of the features that could be added are described.

4.1.1 Car and pedestrian detection

The typical feature in other assistants is the car and pedestrian detection. The system can issue a warning once the car is too close to the obstacle. It can be done by detecting Haar-like features of the obstacle and tracking it in time. The *IPM* and *Image To World* that is already implemented would significantly simplify the problem.

4.1.2 Traffic sign detection

Similarly a traffic sign detector can be added to e.g. warn the driver about exceeding the speed limit. The detector is usually based on a cascade of support vector machine (SVM) classifiers that had been trained for specific traffic sign classes.

4.1.3 Vanishing point detection

There is a number of applications that can take advantage of the vanishing point position. For example, it can be used for the Image To World Model calibration. So far, the model has to be manually re-calibrated every time the camera position, yaw or pitch changes. Based on the vanishing point position the parameters can be computed. The detection itself is based on the output of the *Hough Transform.* Each two line segments are combined and their intersection is found. Some version of the RANSAC[12] algorithm can be then used to determine the vanishing point of the whole road.

4.1.4 Dashed line detection

So far the dashed line detection is based on the *Marking center detection* algorithm that gives certain confidence for a given line. In future it would be good if the lane departure warning was issued only when a full line was crossed.

---

[12] Random sample consensus (RANSAC) is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers.

That would mean to make the dashed line detection algorithm more reliable. More advanced method based on pattern matching could be developed.

4.1.5 Road boundary detection

Another algorithm for the current lane recognition could be developed. This would allow for the algorithm to work in the urban scenarios where the lane markings are not clearly visible. There are equally large number of possible algorithms in existence: based on Bayesian segmentation, curb detection, road color detection and extraction etc...

4.1.6 Curved line fitting

The lines are now fitted with a simple parametric equation of a line as it is sufficient for the lane departure recognition. It could be improved by fitting a B-spline instead. That way the lane could be estimated more accurately on curvy roads.

Bibliography

[1] Paul, A.; Boyle, L. N.; Tippin, J.; et al. Variability of driving performance during microsleeps. In *Proceedings of the Third International Driving Symposium on Human Factors in Driver Assessment, Training and Vehicle Design*, 2005, pp. 433–440.

[2] Hall, S. Elon Musk says that the LIDAR Google uses in its self-driving car ‘doesn't make sense in a car context. 2015, [cited 2016-03-21]. Available from: `http://9to5google.com/2015/10/16/elon-musk-says-that-the-lidar-google-uses-in-its-self-driving-car-doesnt-make-sense-in-a-car-context/`

[3] Model S Software Version 7.0. [cited 2016-03-21]. Available from: `https://www.teslamotors.com/presskit/autopilot`

[4] Mobileye. [cited 2016-03-21]. Available from: `http://us.mobileye.com/`

[5] Malisiewicz, T. Mobileye's quest to put Deep Learning inside every new car. [cited 2016-03-21]. Available from: `http://www.computervisionblog.com/2015/03/mobileyes-quest-to-put-deep-learning.html`

[6] Hillel, A. B.; Lerner, R.; Levi, D.; et al. Recent progress in road and lane detection: a survey. *Machine Vision and Applications*, volume 25, no. 3, 2014: pp. 727–745.

[7] Nieto, M.; Laborda, J. A.; Salgado, L. Road environment modeling using robust perspective analysis and recursive Bayesian segmentation. *Machine Vision and Applications*, volume 22, no. 6, 2011: pp. 927–945.

[8] Muad, A. M.; Hussain, A.; Samad, S. A.; et al. Implementation of inverse perspective mapping algorithm for the development of an automatic lane tracking system. In *TENCON 2004. 2004 IEEE Region 10 Conference*, IEEE, 2004, pp. 207–210.

[9] Canny, J. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, , no. 6, 1986: pp. 679–698.

[10] Kanopoulos, N.; Vasanthavada, N.; Baker, R. L. Design of an image edge detection filter using the Sobel operator. *Solid-State Circuits, IEEE Journal of*, volume 23, no. 2, 1988: pp. 358–367.

[11] Del Moral, P. Non-linear filtering: interacting particle resolution. *Markov processes and related fields*, volume 2, no. 4, 1996: pp. 555–581.

[12] Sehestedt, S.; Kodagoda, S.; Alempijevic, A.; et al. Efficient Lane Detection and Tracking in Urban Environments. In *EMCR*, Citeseer, 2007.

[13] Dempster, A. P.; Laird, N. M.; Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1977: pp. 1–38.

[14] Hough, P. V. Method and means for recognizing complex patterns. Technical report, 1962.

[15] Duda, R. O.; Hart, P. E. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, volume 15, no. 1, 1972: pp. 11–15.

[16] Borkar, A.; Hayes, M.; Smith, M. T. Robust lane detection and tracking with ransac and Kalman filter. In *ICIP*, 2009, pp. 3261–3264.

[17] OpenCV Template Matching. [cited 2016-04-20]. Available from: `http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html`

[18] Lawson, C. L.; Hanson, R. J. *Solving least squares problems*, volume 161. SIAM, 1974.

[19] OpenCV Structural Analysis and Shape Descriptors. [cited 2016-04-21]. Available from: `http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html`

[20] Huang, A. S.; Moore, D.; Antone, M.; et al. Finding multiple lanes in urban road networks with vision and lidar. *Autonomous Robots*, volume 26, no. 2-3, 2009: pp. 103–122.

[21] Jiang, R.; Klette, R.; Vaudrey, T.; et al. New lane model and distance transform for lane detection and tracking. In *Computer Analysis of Images and Patterns*, Springer, 2009, pp. 1044–1052.

[22] Kalman, R. E. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, volume 82, no. 1, 1960: pp. 35–45.

[23] Kalman Filter for a Constant Velocity (CV) Model in Python. [cited 2016-04-23]. Available from: `http://nbviewer.jupyter.org/github/balzer82/Kalman/blob/master/Kalman-Filter-CV.ipynb`

[24] Irani, M.; Rousso, B.; Peleg, S. Recovery of ego-motion using image stabilization. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, IEEE, 1994, pp. 454–460.

[25] VXL - C++ Libraries for Computer Vision. [cited 2016-04-24]. Available from: `http://vxl.sourceforge.net/`

[26] ccv - a modern computer vision library. [cited 2016-04-24]. Available from: `http://libccv.org/`

[27] libCVD - computer vision library. [cited 2016-04-24]. Available from: `http://www.edwardrosten.com/cvd/`

[28] OpenCV. [cited 2016-04-24]. Available from: `http://opencv.org/`

[29] Martin, R. C. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[30] Nieto, M. Source code Inverse Perspective Mapping C++, OpenCV. [cited 2016-04-25]. Available from: `https://marcosnietoblog.wordpress.com/2014/02/22/source-code-inverse-perspective-mapping-c-opencv/`

Appendix A

Installation

A.0.1 Desktop

- Download and install OpenCV library. The process differ with the operation system used. For more information about how to install the library visit the OpenCV website.
- Install latest version of cmake
- In the `code` directory call `cmake .` and `make`
- Run `./build/RoadAssistant -v VIDEO_PATH`
- The camera settings can be adjusted via the settings window or by providing executable with additional parameters. See `./RoadAssistant -h` for more info

A.0.2 iOS Mobile Application

Computer running the OS X operation system is required for the iOS Mobile Application installation. Additionally, a device with iOS 9.3+ is needed.

- Install XCode IDE
- Open `RoadAssistant_iOS.xcodeproj`
- Add your Apple ID to the XCode in preferences. The account does not have be a developer account.
- In the project editor set your account (Personal Team) as a team for the target. If the Xcode displays an error, click Fix issues to create new provisioning profile.
- Run the project with your device connected and selected.

- The application starts in a setup mode where the user is required to align the displayed grid with the road plane. The camera's position, yaw and pitch can be set using a slider in the top of the screen.
- Once the grid is aligned, tap the start button.

Appendix **B**

Contents of CD

- `readme.txt` ... the file with CD contents description
- `measurements` ... the measurement data from the results section
- `code` ... code of the prototype
  - `build` ... executable
  - `src` ... source codes
  - `library` ... libraries
  - `desktop` ... Xcode project for desktop version
  - `iOS` ... Xcode project for iOS version
- `src` ... the LaTeX source code files of the thesis
- `text` ... the thesis text directory
  - `DP_Kohout_Tomas_2016.pdf` ... the Diploma thesis in PDF format
  - `DP_Kohout_Tomas_2016.ps` ... the Diploma thesis in PS format