Semi-Automatic Parameterization of Relevant Video Regions in Surveillance Images

Benjamin Fellner

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2014

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, September 29, 2014

Benjamin Fellner

Contents

Kurzfassung

Straßenerkennung und das Analysieren des Bildinhaltes sind wesentliche Voraussetzungen für unterschiedliche Fahrerassistenzsysteme, welche heutzutage in allen modernen Autos eingebaut werden. Beide Erfordernisse sind große Forschungsgebiete im *Computer Vision* Segment mit vielen verschiedenen Anwendungsgebieten, welche alle ihre eigenen Anforderungen an die Straßenerkennung an sich stellen. Die meisten Veröffentlichungen beschäftigen sich mit einem direkt auf dem Auto angebrachten Kamerasystem.

Dieses System unterscheidet sich in vielen Bereichen im Vergleich zur hier behandelten Aufgabenstellung. Das definierte Ziel dieser Problemstellung umfasst individuelle Fahrbahnspur- und Straßenerkennung auf Bildern, welche innerhalb eines Tunnels erstellt wurden. Die Daten kommen hierbei von mehreren Videokameras, die in den Tunnels installiert sind. Diese können an unterschiedlichen Stellen, z. B. an der Decke oder der Wand, positioniert werden. Die verschiedenen Tunnels, gemeinsam mit den unterschiedlichen Aufbauten, erschweren die Straßenerkennung erheblich.

In dieser Arbeit werden zusätzlich die verwandten Arbeiten thematisiert und die unterschiedlichen Methoden getestet, welche sich mehr oder weniger für eine Segmentierung eignen. Weiters wird ein neuartiger, schrittweise durchgeführter *Image Warping* Algorithmus für Straßenerkennung vorgestellt. Dieser wird in mehrere Subprobleme unterteilt, welche einzeln aufgeschlüsselt und am Schluss zusammengefügt werden.

Abschließend wird im letzten Kapitel das dazugehörige Projekt beschrieben, welches für einen Industriepartner erstellt wurde.

Abstract

Road understanding and street detection are essential requirements for advanced driver assistance systems. This are active fields of research in computer vision with many different application areas. All of these diverse areas have various requirements for the street and lane detection step. Most of the publications focus on visual systems, installed on the car itself.

This setup differs in many ways, compared to the specified problem, which is elaborated here. The goal is to develop a system, which performs street and lane detection on road images taken inside a tunnel. The input source are multiple video cameras, which are mounted on various positions inside the tunnel. They can be on the ceiling or on any side of the wall. The diversity of the environment intensifies the detection problem.

This thesis elaborates the related work with different approaches, which are more or less suitable for the given task, leading to a new incremental image warping algorithm for street and lane detection. This approach is divided into several sub-problems, which are all worked out individually and put together at the end.

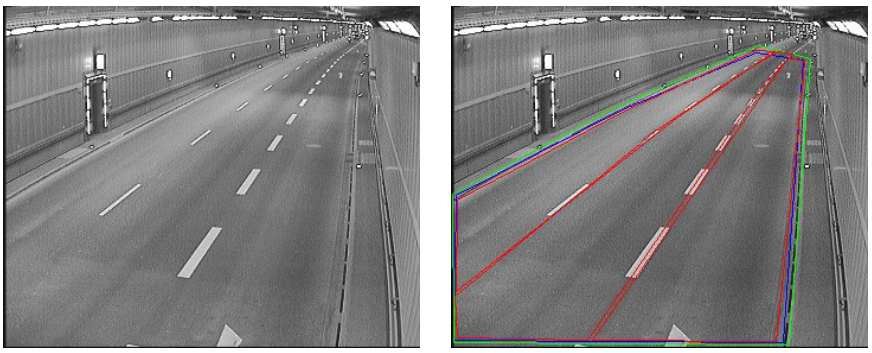
Finally an associated project was done with an industrial partner, which is covered in the implementation chapter and completes the theoretical work.

Chapter 1

Introduction

Understanding the content of an image is essential for any kind of surveillance system. In this master thesis the main focus is on road and lane detection in traffic surveillance videos. This is a crucial part for the further analysis and automatic event detection done by an industrial partner. The associated thesis project was done in cooperation with this company and will be used in their next program release. Some images are partially obscured in order to fulfill the signed non disclosure agreement.

The main task is to improve the parameterization of a security camera, which is most of the time inside a motorway tunnel. Multiple polygon structures should be created, which represent the road with its corresponding lanes, see Figure 1.1. Up to this point, drawing the different polygons onto the video stream was done by hand and took quite some time. The fact that



(a) (b)

Figure 1.1: The desired goal. Given is an image, which shows a tunnel scene without any cars (a). The program should find the road (green) with its corresponding lanes (red). The output should be a polygon, which can be used in other parts of the program. Additionally the result should be drawn onto the video stream (b).

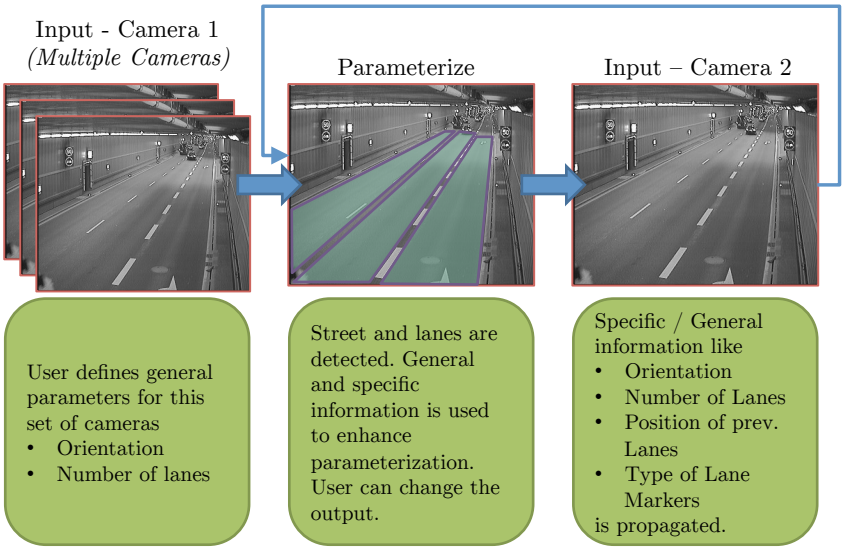


Figure 1.2: The desired architecture of the parameterization algorithm. The main focus of this thesis will be on the street and lane detection and therefore the parameterization of one camera.

numerous video cameras must be parameterized for one surveillance project intensifies the workload.

Now the main goal is to develop an algorithm, which improves this parameterization process by making it faster[1] and more convenient for the operator. Taking into account that multiple cameras are filming the same tunnel, the previous information should be passed along to the next input image in the current sequence in order to enhance the detection algorithm, see Figure 1.2.

1.1 Problem definition

Although we as humans can immediately see the road with its corresponding lanes, it is not an easy task to do this semi-automatically. There are numerous factors, which impede the detection process. These problems can be categorized into *Environmental*, the surroundings of the street, *Self-induced*, the street and lane markings itself and *Input*, the image quality. Environmental problems are:

- Most of the times the tunnel itself describes a curve due to different security reasons, see Figure 1.3 (a).
- Dashed lane markings do not have the same distance to each other.

[1] Achievable with fewer mouse clicks.

- Road-like structures appear on the tunnel walls.
- The road is divided due to a motorway exit, which has a completely different appearance, see Figure 1.3 (b).
- Street bays can be installed in the tunnel and therefore appear in some images, see Figure 1.3 (c).
- The detection itself should work on the outside as well.
- The entrance and the exit of a tunnel change the whole appearance completely, see Figure 1.3 (d).

Self-induced problems apply to the street with its lane markings. These come into play especially if the algorithm uses these markings for the lane detection process. Some of them are:

- Lane markings appear with different sizes, shapes and color values.
- Sometimes only a curbside is installed and therefore the side marks of the road does not exist.
- Different types of dashed lane markings can appear in one image.
- The surface of the road itself does not have a unique structure, which would make it suitable for some segmentation methods.
- The surface of the road is inconsistent throughout the whole tunnel due to abrasion and wet street. The whole surface can change and new structures can appear.
- Other road markings like arrows or labels can appear anywhere.

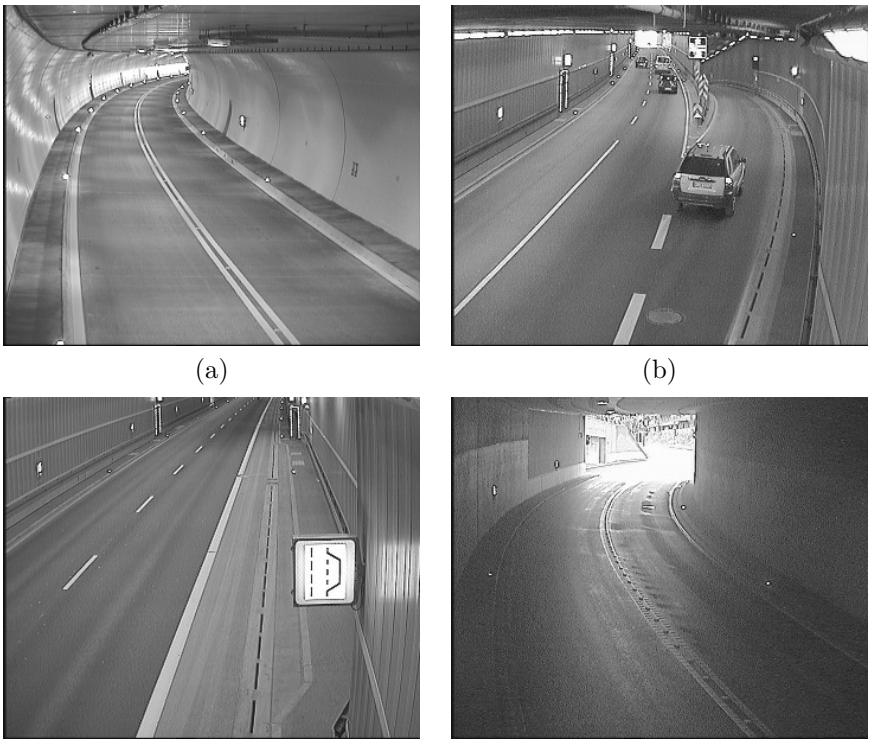
Input problems apply to the given input image. These can vary greatly based on the used camera. Some of them are:

- Reflections on the road, caused by wet asphalt, imitate lane markings.
- The image quality itself is not good and has a very small resolution (352×288).
- Straight lines in the real environment do not appear as straight lines in the image due to the different cameras, which are not calibrated.
- In the rear part of the street (the upper part of the image) the lane markings are only some pixels big.
- Clipping can occur, therefore the street markings itself can start in the middle of the image.

These problems must be kept in mind when creating the different approaches. In Figure 1.3 some of them are displayed.

1.2 Expected end result

The optimal solution would be that the road detection algorithm should work on every image regardless of the road type, see Figure 1.4. This is not



(a) (b) (c) (d)

Figure 1.3: Detection challenges. Most of the time the tunnel describes a curve (a). A motorway exit changes the appearance completely (b). Street bays can occur (c). An exit of a tunnel with very bad lighting conditions (d).

expected, because it is very hard to achieve and greatly depends on the used detection algorithm.

However, the algorithm should be able to detect the road with its corresponding lanes in an image with good visible lane markings. The user should have the possibility to modify the resulting polygon structure. This should be doable by either setting different starting points for the detection algorithm or by changing the polygon directly. Additionally the desired polygon structure with the surrounding street polygon, which includes all the individual lane polygons, should be created in a finishing step.

Nevertheless this work focuses on road and lane detection including an overview of related work in Chapter 2 with alternative approaches explained in Chapter 3. Road marker detection is described in Chapter 4 and the Chapters 5 - 7 explain the incremental image warping algorithm, which is the main part of this work. Results are shown in Chapter 8 and the implementation is outlined in Chapter 9. The thesis will be completed by the conclusion and future work in Chapter 10.

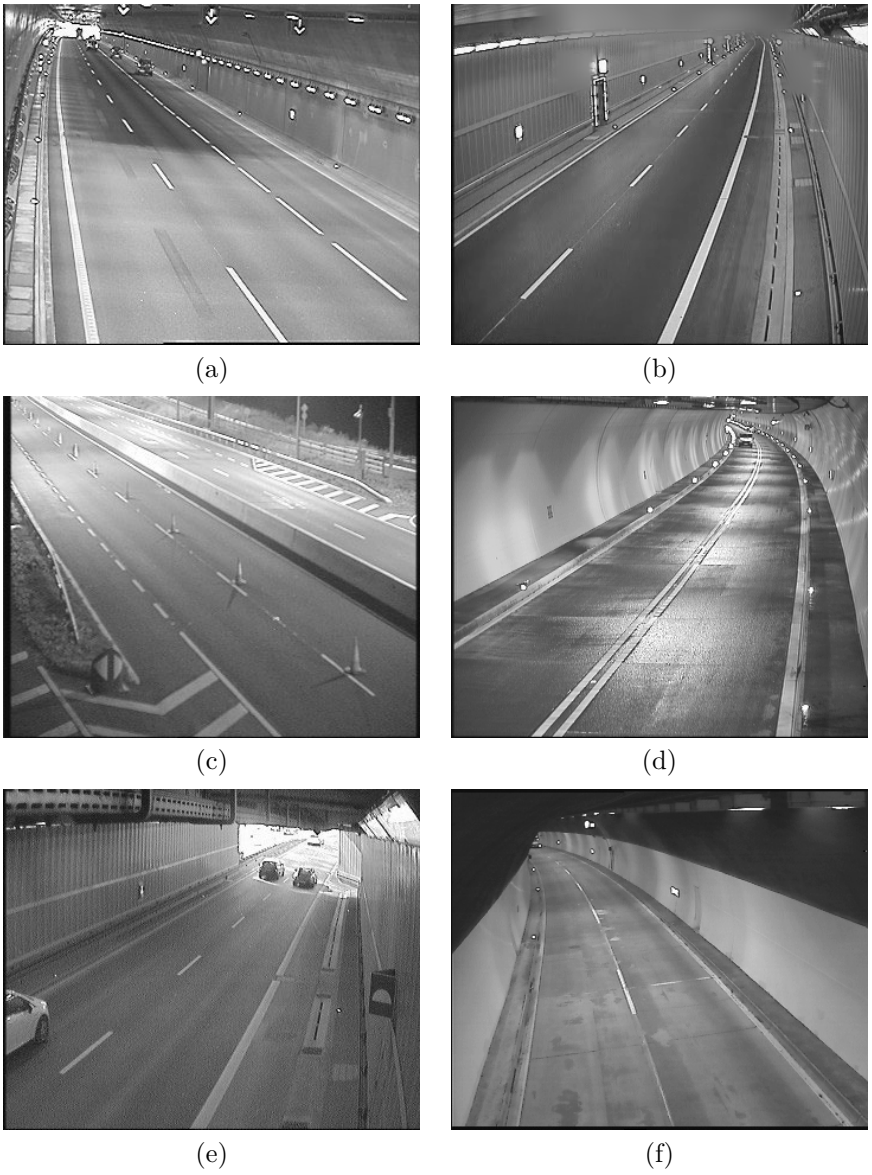


Figure 1.4: A subset of different road types. A road with three lanes and different dashed lane markings (a); a road with two lanes and clutter (b); an outside scene with cones on the street (c); a tunnel entrance with a wet street (d); an exit of a tunnel (e) and a worn road with dirty markings (f).

Chapter 2

Related work

In a modern car numerous systems support the driver in different ways and they are getting smarter and more reliable with every new edition. It goes from lane change assistance over accident prevention to parking aid. The problem of visual road and lane detection is crucial for these systems, although many other sensors are used additionally. Driver assistance systems, which can reduce the risk of car accidents are getting more and more reliable. These systems can either alert the driver in dangerous situations or engage actively in the driving process.

However, full autonomous driving in urban and rural environments is still not fully achieved yet. It is a huge field with many published contributions. Very good surveys were published in 2006 [18] and 2012 [1], which give a good overview over this interesting topic. The following chapter is a brief summary of the expected features and the different approaches. A more accurate comparison of alternative approaches with their pros and cons is given in Chapter 3.

2.1 Automotive features

Current and future automotive features have their different lane and road understanding demands. At this time some of them work quite reliable, but others, like full autonomous driving, have their limitations. The following driver assistance systems were summarized by [1] and provide a good overview:

- Lane Departure Warning (LDW) [10] issues warnings for near lane departure events.
- Adaptive Cruise Control (ACC) follows the nearest vehicle in the current lane with a safe headway distance.
- A lane keeping system returns the car to the lane center when unsignaled lane departure occurs.

(a) (b)

Figure 2.1: DARPA Urban Challenge. A typical vehicle in the DARPA Urban Challenge [31]. Multiple computers (a) evaluate and combine the data of the different sensors (b) [24].

- A lane centering system keeps the car in the middle of the lane.
- A lane change assistant system changes the lane autonomously on demand.
- A turn assistant system turns on driver or automatic navigation demand autonomously.
- Full autonomous driving for paved roads in cities and on highways.
- Full autonomous driving for cross country driving in non-paved areas.

Nowadays full autonomous driving is possible, but has its limitations, especially for cross country driving [15]. In an urban environment such a system would need to achieve road and lane perception with obstacle detection, based on the infrastructure provided for human drivers. This type of computational scene understanding with a low error tolerance, is beyond the reach of current perceptual systems.

A lot of research efforts were made for the DARPA Grand Challenge (2005)[1] [30] and the Urban Challenge (2007)[2] [31]. However, this can not be compared to a vehicle, which performs autonomous driving by using only one camera. In these challenges a typical vehicle has multiple *Light Detection And Ranging* (LIDAR), geographic information systems (GIS), global position system (GPS), internal measurement units (IMU) and the computing power of a dozen computers [24], see Figure 2.1.

2.2 Generic system

Numerous approaches can achieve lane detection and road segmentation. In [1] a generic system (see Figure 2.2) was presented, which can be applied to almost all of the published techniques. This system decomposes the detection

[1] Wikipedia site (http://en.wikipedia.org/wiki/DARPA_Grand_Challenge_(2005)).
[2] Wikipedia site (http://en.wikipedia.org/wiki/DARPA_Grand_Challenge_(2007)).

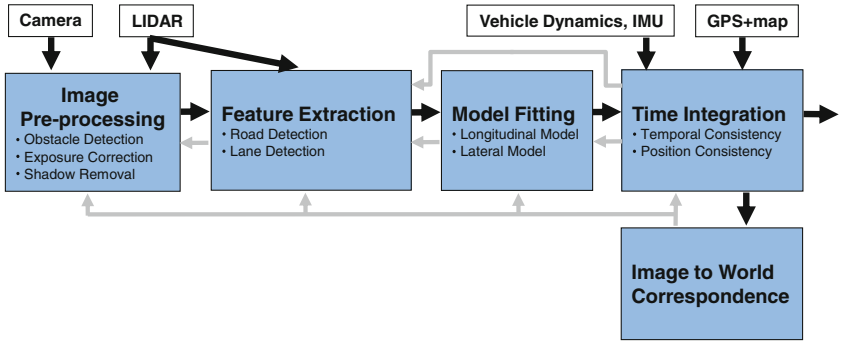


Figure 2.2: Architecture of a generic system for road detection. Input for the different modules is the camera image, *Light Detection And Ranging* (LIDAR), vehicle dynamics and internal measurement units (IMU). Image taken from [1].

process into five modules, image pre-processing, feature extraction, model fitting, time integration and image to world correspondence. Depending on the requirements, some of the models can be ignored. In a way the developed algorithm proposed in Chapter 5 can be applied to this system as well. The input can be from a simple camera image, stereo imaging, *Light Detection And Ranging* (LIDAR), vehicle dynamics and internal measurement units (IMU). Stereo imaging as well as the LIDAR system can be used for obstacle identification and curb detection [12].

2.2.1 Image pre-processing

Several operations can be applied to the image in order to remove image artifacts and to enhance the desired features. This step can also reduce errors introduced by illumination-related effects, shadows on the road, image noise and many more.

For the proposed incremental image warping algorithm (Chapter 5) a smoothing filter and edge detection were applied. The road marker detection algorithm (Chapter 4) performs region labeling on the input image.

2.2.2 Feature extraction

Image features like lane markings and road boundaries are detected in this step of the overall process. *Ridgeness* and RANSAC [17] (see Figure 2.3) are used as well as thresholding [25]. In general the appearance of lane boundary features can vary greatly. The size, the color and the distance to each other can not be assumed to be constant. This appearance variety is worked out in [18].

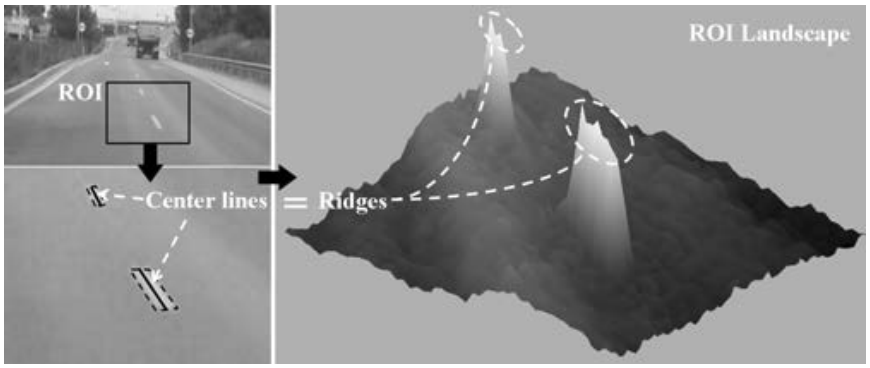
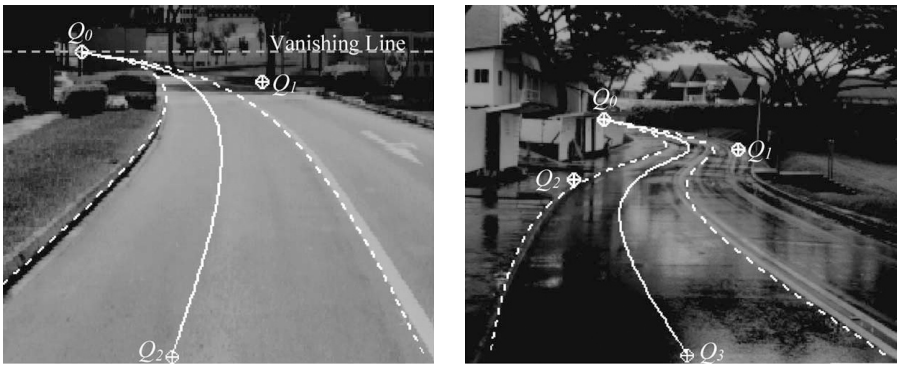


Figure 2.3: Extracted road features. The road markings are extracted into features, these features are then further processed. Image taken from [17].



(a) (b)

Figure 2.4: B-Snake lane model. A lane model using 3 control points where Q_0 is on the previously detected vanishing line (a). Another model using 4 control points (b). Image taken from [26].

For the road marker detection (Chapter 4) feature extraction was done by using the Niblack [19] algorithm, which performs adaptive thresholding on an image, see Figure 4.4.

2.2.3 Model fitting

Usually a geometric model is fitted to the extracted visual features. In some cases B-Snakes [26], see Figure 2.4 are used as well as other geometrical models [28]. Normally Random Sampling Consensus (RANSAC) [14] is used for model fitting for all model types. In addition image features like the position of the vanishing point and the vanishing line are used in order to enhance the fitting process.

In the proposed incremental image warping algorithm a model is de-

signed as a combination of many transformation parameters, see Chapter 5. It is fitted step by step, analyzing every horizontal image line.

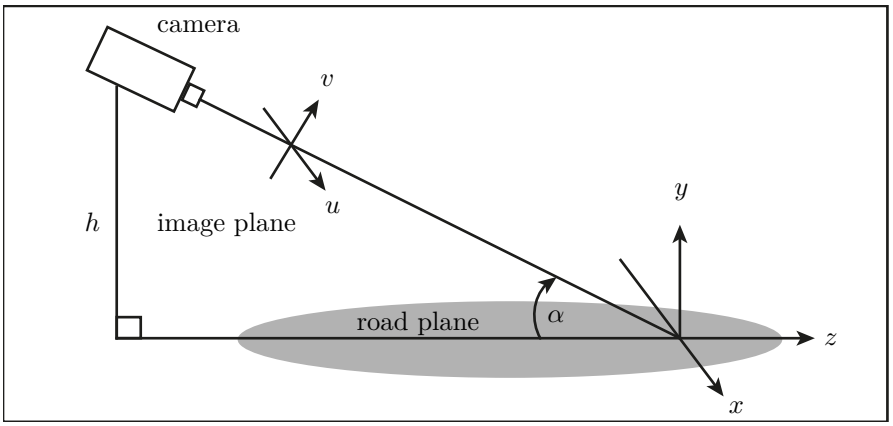
2.2.4 Time integration

Propagating the already processed information from one frame to another can benefit the road and lane detection in the next step, see Figure 1.2. In order for this to work it requires great similarity between the individual time steps. Time integration is not used for the developed algorithm, because the similarity constraint is not fulfilled in a satisfying way.

2.2.5 Image to world correspondence

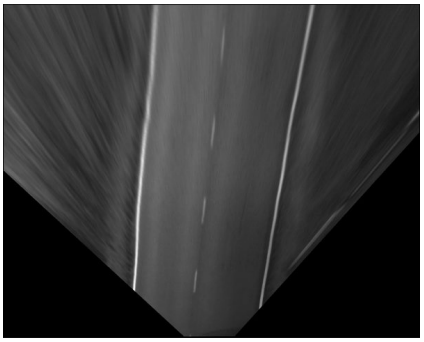
The process of finding straight and parallel lines really improves if the distorted camera image is transformed into a perspective-free view, see Figure 2.5. With the transformed image methods like statistical Hough transform and particle filter [16] can then be used to find these lines. Nevertheless such a perspective-free view can be very beneficial in almost every other stage of the road detection algorithm.

However, the camera position in correlation to the road plane must be known, see Figure 2.5 (a). This is not the case for the given problem scope. The parameters could be estimated, but due to the huge case diversity and road property deviations, see Section 1.1, the result would be inaccurate and not supportive.



(a)

(b)



(c)

Figure 2.5: Transformation of the camera image. With the known position of the camera in relation to the road plane (a), a transformation matrix can be defined, which transforms the original image (b) into the rectified one (c). Image (b) and (c) taken from [16].

Chapter 3

Alternative approaches

During the progress of the associated thesis project, several processing procedures were implemented and tested in order to achieve a good segmentation. Some of them have a high possibility for a good road segmentation, but others will fail completely due to the given problem. All of the implemented approaches, including road marker detection, see Chapter 4 and the final incremental image warping algorithm, see Chapter 5, work directly on the 2D image plane without any information of the 3D geometry. This was done because the environment, the cameras itself, the view angle and the position of the camera are different in every project.

However, this does not mean that no 3D geometry information can be gained by analyzing the result. For example the vanishing point algorithm, see Section 3.5 and the detected road markings can provide a clue about the overall environment. Nevertheless some methods like the Hough transform, see Section 3.4, would not benefit from a known image to world transformation due to the fact that curved roads remain curved in a transformed image, see Figure 3.1. Additionally some model fitting methods, see Section 2.2.3, require a vanishing point as an anchor for the used models and can therefore not work with the transformed image.

So the following chapter gives an overview about the several processing algorithms that are tested and implemented in order to achieve a good segmentation. See Table 3.1 - 3.3 for a short description, advantages and disadvantages of the different methods.

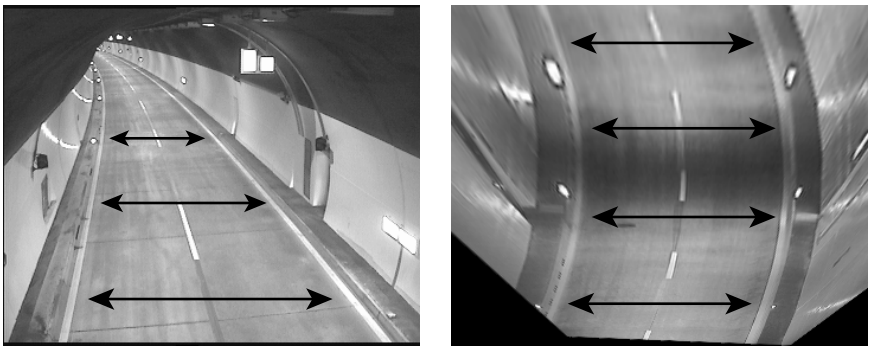
3.1 Color segmentation

Road segmentation based on the color range of the street surface is not successful due to several factors, see Figure 3.2. Some of them are, that

- the color range is very limited or even restricted to black and white,
- the walls of the tunnel have the same color and
- unbalanced lighting produces light spots on the roadway.

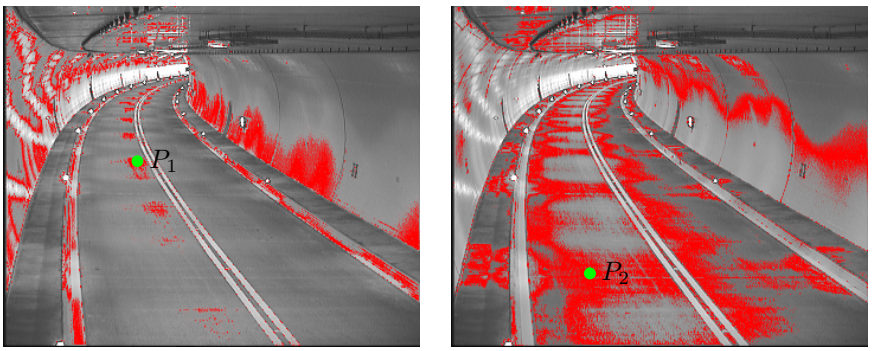
Table 3.1: Different segmentation approaches. These methods use segmentation in order to achieve road and lane detection.

Methods	Pros / Cons
Color Segmentation: The road should be segmented based on a defined color range. One color sample of the road is taken and a specific range around that color is set, see Section 3.1.	• Simple approach. • Color range can be tuned to handle uneven lighting. • Different color spaces can be used.
	• Very limited for grayscale images. • Greatly depends on the road surroundings. • Does not work in tunnels were the walls have the same color.
Texture-based segmentation: The road should be segmented based on a defined texture. Usually the textures in road images are strongly anisotropic with a dominant orientation and can be grouped together, see Section 3.2.	• Unsupervised segmentation. • Can compensate for uneven lighting.
	• Greatly depends on the road surroundings. • Individual lanes are not distinguishable. • Sidewalks and other structures can lead to a false segmentation. • Correcting an incorrect result can be difficult.
Watershed segmentation: The road with its lanes should be segmented using the watershed algorithm, see Section 3.3.	• Few input is required. • It is simple to use. • The image is divided in useable areas.
	• Dashed lane markings propose a problem. • Segments can bleed into each other. • Analyzing and fixing the result can be hard.



(a) (b)

Figure 3.1: Transformed image. The original image (a) shows the expected image distortion. The width of the street is getting smaller and smaller while advancing to the vanishing point. In (b) a part of the transformed image is shown. The width remains the same throughout the image.



(a) (b)

Figure 3.2: Segmentation based on color range. The points P_1 in (a) and P_2 in (b) indicate the position of the color sample. The red area shows the color range given the sample points. This approach will not be successful due to several factors like unbalanced lighting and limited color range of the grayscale image.

So in this scenario color segmentation fails completely due to the given circumstances, but this simple technique can be successful on other task settings. Like in [11] color based segmentation is used in order to detect lane boundaries of a road, using a region of interest and an auto threshold value.

3.2 Texture-based segmentation

Texture-based segmentation can deliver good results, especially on road images in rural areas without any lane markings, see Figure 3.3. Other methods

Table 3.2: Different approaches for image feature extraction. These methods can provide additional information about specific road and lane features, but they do not deliver a complete segmentation.

Methods	Pros / Cons
Hough transform: The street lines should be segmented with the Hough transform, see Section 3.4.	• Straight lines are detected. • Dashed lane markings in a straight line can be detected.
	• Curved lines can not be handled. • Many lines are produced that have nothing to do with the street. • A correlation between the set of lines and the street needs to be done.
Vanishing point detection: The vanishing point (VP) of the street should be detected, see Section 3.5.	• The VP provides useful information. • A road model can use the VP as an anchor.
	• There is no individual VP for curved roads. • For those the image needs to be divided into sections. • These individual results need to be merged.

that rely on those markings or on a color difference between the surface and the environment would fail here. Usually the textures in road images are strongly anisotropic with a dominant orientation. Based on these features and a proper classification a good segmentation can be achieved [27]. Additionally other image features like vanishing point estimation can be done by grouping the dominant orientations of the road texture together [22].

3.3 Watershed segmentation

Watershed Segmentation [3] can produce quite good results, see Figure 3.4 (b), but needs a lot of initial input, which must be known in advance. Different factors make the segmentation more difficult. The following road properties,

Table 3.3: Methods to solve the problem. The algorithms used for these methods were implemented during the thesis project and extensively tested. Especially the incremental image warping algorithm produces the best results.

Methods	Pros / Cons
Road marker detection: The road markings of the street should be detected and merged together, see Chapter 4.	• Dashed lane markings can be merged. • The Lane orientation can be calculated.
	• Hard to decide which region is a dashed lane marker. • A high enough resolution is required in order to detect markers robustly throughout the image.
Incremental image warping: The algorithm should straighten and rectify an image of a curved road without any prior knowledge about the content itself, see Chapter 5.	• Very few user input is required. • Can even work unsupervised. • A correct result provides a complete rectification and • a detailed road model.
	• Dashed lane markings can propose a problem. • Other structures can hinder the warping algorithm. • Good visible markings and strong edges are required.

like

- the amount of road lanes,
- the initial position and
- the initial orientation

must be known in advance. If thats not the case, the algorithm must be designed to estimate these parameters. However, the different appearances of various roads can make this a hard task. Additionally the output needs to be evaluated by a system, which can determine

- whether the solution is correct,

Figure 3.3: Gravel road image. Texture-based segmentation can be quite successful in this case.

- which region is a lane and which not,
- where does the lane end and
- which parts need to be removed.

Understanding the output of the watershed algorithm and identifying and removing false parts is not trivial. Additionally streets with dashed lane markings, see Figure 3.4 (c), represent a problem for the segmentation. These images need to be pre-processed in order to improve the result. This can be done by the proposed lane merging algorithm, see Section 4.4.

In [2] a fast watershed transformation was introduced, which achieves road segmentation and obstacle detection on a video stream of a moving car. Here the connection of the ground markings is done by a temporal filter applied on consecutive images and the initial input for the watershed algorithm is gained from the previous image.

3.4 Hough transform

The Hough transform [13] can be used to find the best straight lines in one image, see Figure 3.5. The problem with this approach is that the parameters of the algorithm vary greatly for different images. This is because in this case the Hough transform is applied to a canny edge image [8], which has of course its own parameter set. The threshold value, which defines how long a line should be, depends heavily on the image content. If this value is set very low, see Figure 3.5 (d), then a lot of short straight lines are found. These lines must then be set in relation to each other in order to eliminate all the

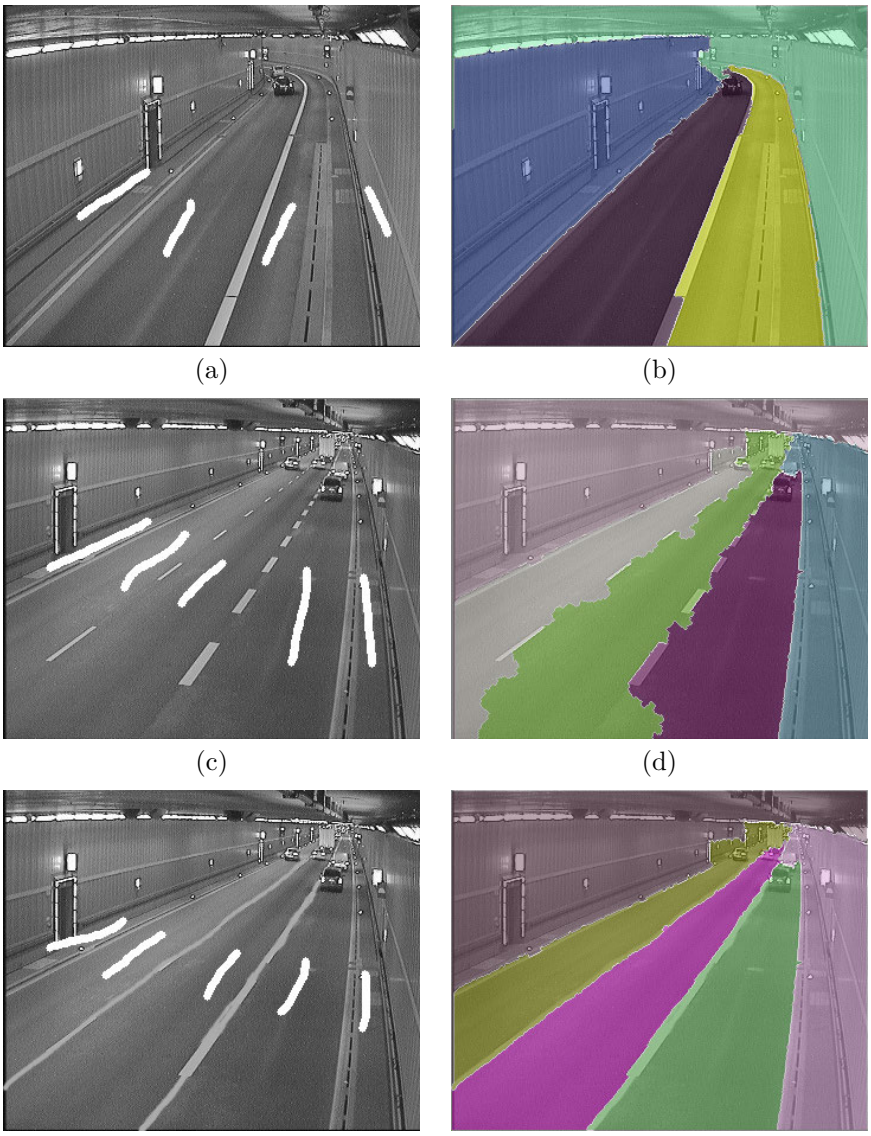
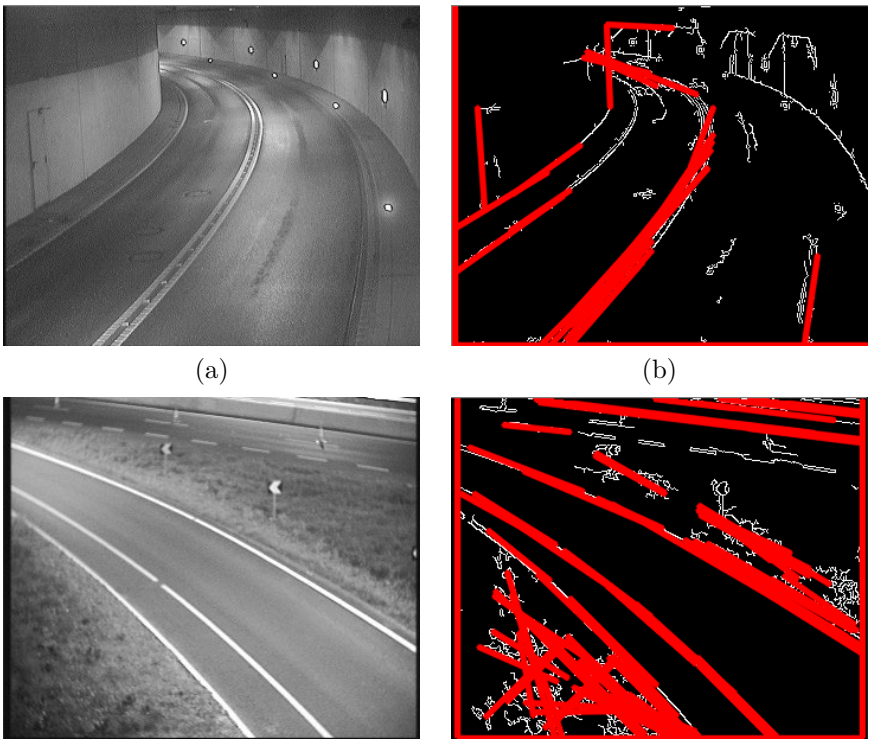


Figure 3.4: Watershed segmentation examples. The input (white strokes) of the algorithm was user generated. In order to do this automatically, road properties like the orientation and the amount of lanes must be known in advance. Street lanes with clear and solid boundaries (a) can be detected quite good (b). Problems occur when dashed lane markings are present (c). This produces, depending on the marker distance, inaccurate results (d). Merging these individual markers (e) really improves the result of the segmentation (f).



(a) (b)

(c) (d)

Figure 3.5: Hough Transform. A really strong curved road (a). The result of the canny edge detection and Hough transform (b), which does not work in this case. Another example (c), which delivers better results (d). However, after the Hough transform all the different lines must be put in relation to each other in order to eliminate all the false positives and to detect the road.

false positives and to detect the road itself.

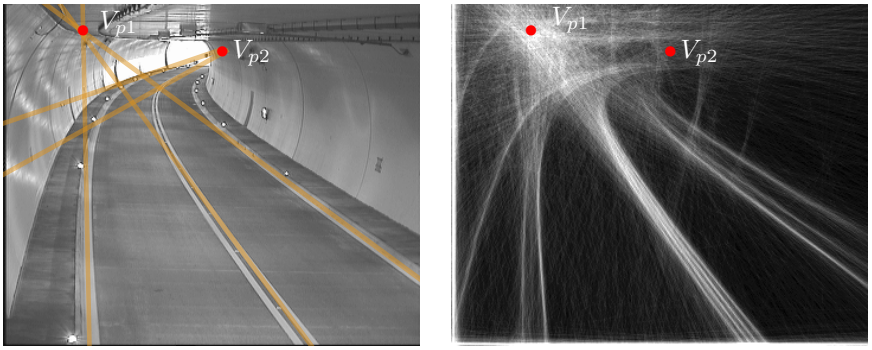
However, road segmentation based only on the Hough transform is not practicable, but it is heavily used for other purposes throughout the literature.

3.5 Vanishing point detection

The vanishing point in one image provides additional helpful information about the 3D environment of the scene. Given a valid detection, it shows where the street is converging to. This can be useful in order to fit a specific model of the road where the vanishing point can serve as an anchor point in the fitting process.

However, the vanishing point can be outside of the image ore may not be present at all. This must be considered in the algorithms, which use it.

Figure 3.6: Vanishing Point of a straight road. The straight visible image lines make the detection of the vanishing point V_p (red dot) very accurate and easy. Image taken from [33].



(a) (b)

Figure 3.7: Vanishing Point of a curved road. The original image with a few straight lines drawn into it resulting in two approximate positions of V_{p1} and V_{p2} (a). Result of the proposed algorithm with the original positions (b). It is clearly visible, that a unique vanishing point for the whole image cannot be found.

3.5.1 General detection

Normally the vanishing point is found by the intersection of the visible straight lines in one image, see Figure 3.6. Detecting the vanishing point on a curved road is not trivial, because no clear intersection of the different straight lines is given, see Figure 3.7. In [26] an algorithm was presented

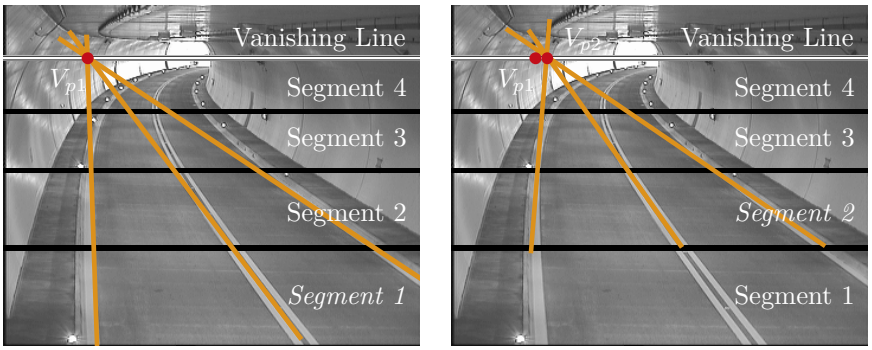


Figure 3.8: Vanishing Point of image segments. All points lie on the vanishing line, which has to be defined first. The vanishing point V_{p1} for the first segment is calculated (a). If no point is found for a segment then the vanishing point from the previous segment is duplicated. The vanishing point V_{p2} for the second segment is found and placed on the vanishing line (b).

that deals with that problem. It consists of the following steps.

1. The image is divided into a small number of segments.
2. A vanishing line is defined where all the following points are located.
3. For every segment an individual vanishing point is located.

The process is illustrated in Figure 3.8.

3.5.2 Detection in image space

In [9] two algorithms were proposed to find vanishing points in natural images. They are not limited to work on road images alone. The second approach, which works directly in the image plane, was implemented and tested. The image is used as an accumulation space for a particular version of the Hough transform. The algorithm is composed of the following steps, which are illustrated in Figure 3.9.

Input: The input for this algorithm is the original grayscale image $\mathbf{I}$ of size $M \times N$. The resulting accumulated image $\mathbf{A}$ has the same size.

1. An edge detection operation is performed on the original image $\mathbf{I}$. This is done by using a convolution with a 3×3 kernel. The proposed kernels, H_x in x- and H_y in y-direction are defined as,

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad H_y = \begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}. \tag{3.1}$$

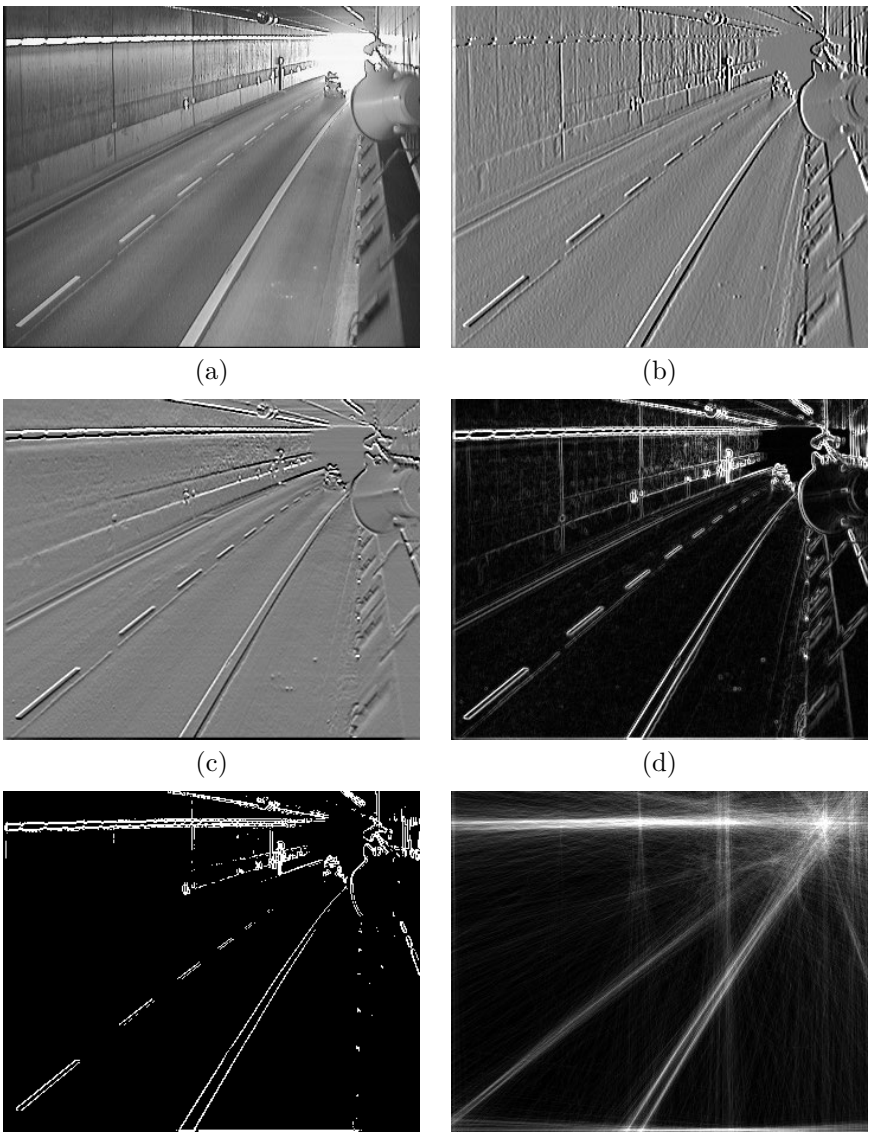


Figure 3.9: Vanishing point detection algorithm example. The original grayscale image $\mathbf{I}$ (a). The image convolved with kernel H_x resulting in $\mathbf{D}_x = \mathbf{I} * H_x$ an edge image in x-direction (b). The same image convolved with kernel H_y resulting in $\mathbf{D}_y = \mathbf{I} * H_y$ an edge image in y-direction (c). The normalized absolute image $\mathbf{N}$ of both edge images (d). Applying a threshold t, which depends on the image, results in the binary image (e). The accumulated image $\mathbf{A}$ (f).

These kernels are very simple and were exchanged with

$$H_x = \frac{1}{32} \cdot \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \quad \text{and} \quad H_y = \frac{1}{32} \cdot \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}, \tag{3.2}$$

to achieve better results, see [7]. So the edge images in x-direction $\mathbf{D}_x$ and in y-direction $\mathbf{D}_y$ are defined as,

$$\mathbf{D}_x = \mathbf{I} * H_x \quad \text{and} \quad \mathbf{D}_y = \mathbf{I} * H_y. \tag{3.3}$$

This is illustrated in Figure 3.9 (b) and (c).

2. The normalized image $\mathbf{N}$ is the summation of both edge images $\mathbf{D}_x$ and $\mathbf{D}_y$, divided by the maximum image value $\mathbf{N}_{\max}$. It is defined as

$$\mathbf{N} = \frac{|D_x| + |D_y|}{\mathbf{N}_{\max}} \quad \text{with} \quad \mathbf{N}_{\max} = \max_{(x,y)\in\mathbb{N}} \mathbf{N}(x,y). \tag{3.4}$$

This is illustrated in Figure 3.9 (d).

3. In this optional step a binary image $\mathbf{B}$ is created by using a defined threshold value, see Figure 3.9 (e).
4. For every point $p(x,y) \in \mathbf{B}$ the orientation $\phi(x,y)$ is calculated by

$$\phi(x,y) = \arctan \frac{|D_y(x,y)|}{|D_x(x,y)|}\,. \tag{3.5}$$

With the given slope $\phi(x,y)$ a line passing through $p(x,y)$ can be added to the accumulated image $\mathbf{A}$, see Figure 3.9 (f).

5. Now a suitable maximum detection method should calculate the position of the vanishing point $V_p(x,y)$ in $\mathbf{A}$.

The calculation of binary image $\mathbf{B}$ is not necessary for the algorithm and can be skipped. This procedure is described in Alg. 3.1.

3.5.3 Problems and conclusion

Different problems with the proposed algorithm occur very quickly.

- The vanishing point of an image of a curved road can not be detected exactly, see Figure 3.7.
- The threshold value t depends on the image content and must be set separately, see Figure 3.9 (e).
- The vanishing point can lie outside the image window. In that case the parameter space $\mathbf{A}$ must be extended. An automatic event detection of this must be developed.
- Multiple maxima can occur in the parameter space $\mathbf{A}$.

Algorithm 3.1: Simplified vanishing point detection algorithm. Given a grayscale image this algorithm calculates the probable vanishing point.

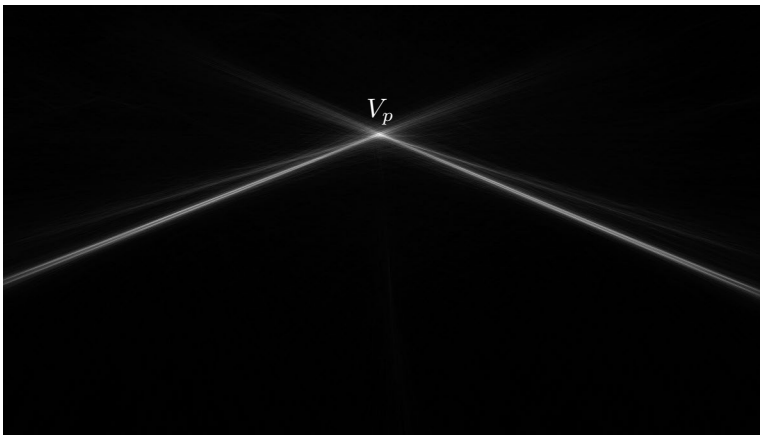
1: VP Detection($\mathbf{I}, t$)
The image $\mathbf{I}$ of size $M \times N$ and the threshold t are given.
The algorithm returns the position of the vanishing point $V_p(x, y)$.
2: $\mathbf{D}_x = \mathbf{I} * H_x$ ▷ see Eqn. 3.2 and Figure 3.9 (b)
3: $\mathbf{D}_y = \mathbf{I} * H_y$ ▷ see Eqn. 3.2 and Figure 3.9 (c)
4: $\mathbf{N}_{\max} \leftarrow \max_{(x,y)\in\mathbb{N}} \mathbf{N}(x, y)$
5: $\mathbf{N} \leftarrow \frac{\mathbf{D}_x + \mathbf{D}_y}{\mathbf{N}_{\max}}$ ▷ see Eqn. 3.4 and Figure 3.9 (d)
6: $\mathbf{A} \leftarrow ()$ ▷ accumulated image of size $M \times N$
7: **for all** $p(x, y) \in \mathbf{N}$ **do** ▷ for every pixel p in $\mathbf{N}$
8: **if** $\mathbf{N}(x, y) > t$ **then** ▷ if value > threshold
9: $\phi(x, y) \leftarrow \arctan \frac{|D_y(x,y)|}{|D_x(x,y)|}$
10: $\mathbf{A} \leftarrow$ CalcAndAddLine($\phi(x, y), \mathbf{A}$)
11: **end if**
12: **end for**
13: $V_p(x, y) \leftarrow$ DetectMaximum($\mathbf{A}$)
14: **return** V
15: **end**

16: CalcAndAddLine($\phi(x, y), \mathbf{A}$)
The orientation $\phi(x, y)$ at position x, y as well as
the current accumulated image $\mathbf{A}$ of size $M \times N$ are given.
A line L of length N, the height of the image $\mathbf{A}$, is created and
added to the accumulated image $\mathbf{A}$.
The process is illustrated in Figure 3.9 (f).
The updated accumulated image $\mathbf{A}$ is returned.
17: $x_1 \leftarrow x + N \cdot \cos(\phi)$
18: $y_1 \leftarrow y + N \cdot \sin(\phi)$
19: $x_2 \leftarrow x + N \cdot \cos(-\phi)$
20: $y_2 \leftarrow y + N \cdot \sin(-\phi)$
21: $L \leftarrow$ CreateLine(x_1, y_1, x_2, y_2) ▷ A new line is created.
22: $\mathbf{A} \leftarrow$ DrawLineInImage($L, \mathbf{A}$) ▷ L is added to the image $\mathbf{A}$.
23: **return** $\mathbf{A}$
24: **end**

25: DetectMaximum($\mathbf{A}$)
The accumulated image $\mathbf{A}$ of size $M \times N$ is given.
Different methods can be used. In [9] $\mathbf{A}$ is averaged
by an 11×11 kernel H around each pixel.
The position x, y of the maximum in $\mathbf{A}$ is returned.
26: $\mathbf{A} \leftarrow \mathbf{A} * H$
27: **return** $\max_{(x,y)\in\mathbb{N}} \mathbf{A}(x, y)$
28: **end**

(a)

(b)



(c)

Figure 3.10: Vanishing point detection result. The original image converted to grayscale (a). The normalized edge image **N** (b). The accumulated image **A** with the detected vanishing point V_p (c). A specific threshold t was set in order to eliminate weak edges, see Alg. 3.1.

Nevertheless the algorithm can produce quite accurate results, see Figure 3.10. As a pre-processing step the input image can be smoothed in order to improve the edge detection step. Additionally the calculation of the orientation ϕ may be omitted if another line drawing algorithm is used, which does not need an orientation in degrees. If the exact position of the vanishing point can not be calculated, then the algorithm can provide a hint of the street orientation, which can be used for other purposes.

Chapter 4

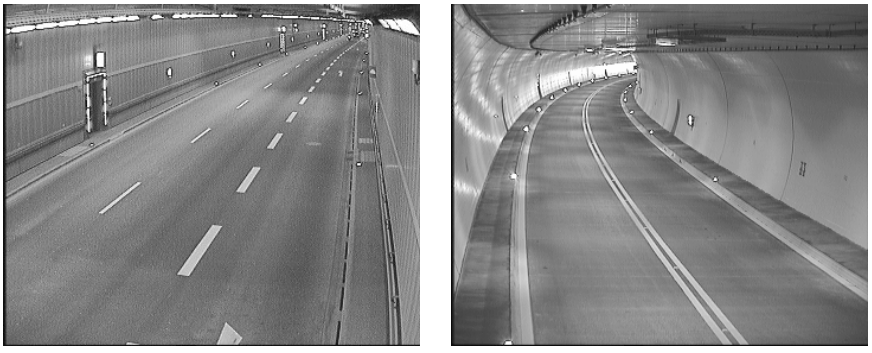
Road marker detection

The basic idea of the road marker detection algorithm is that dashed road markings should be detected and in another step merged together. The focus lies on these interrupted markers, because they stand out in the image, see Figure 4.1. If they are present and correctly detected, they can provide useful information about the road properties. This includes

- the number of lanes,
- the position and
- the orientation of the street.

In addition to the watershed algorithm, dashed lane markers want to be merged in order to get a better segmentation, see Section 3.3.

However, the main problem is to distinguish a marker from another similar structure in the image in a robust way. This is not a trivial task due to several challenges that will be explained in this chapter.



(a) (b)

Figure 4.1: Road markings. Good visible dashed road markings, but without any border lines (a). A curved tunnel with good visible continuous lane markings (b).

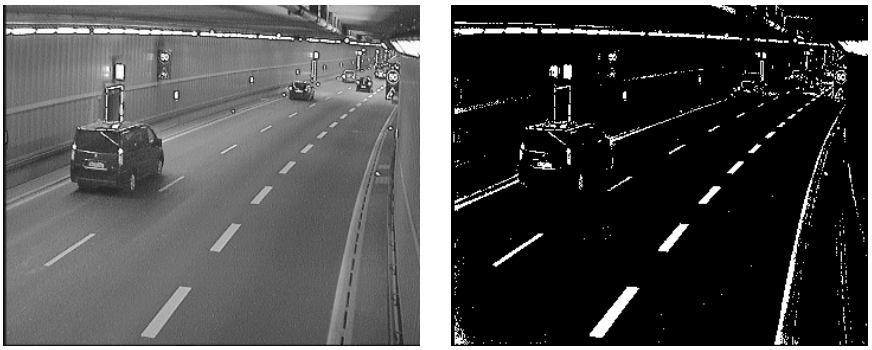
(a) (b) (c) (d)

Figure 4.2: Comparison of global threshold values t. The original image (a), is being thresholded with a factor $t = 100$ (b), $t = 140$ (c) and $t = 160$ (d). Obviously this is not a good solution, because the threshold value is very sensitive and vary greatly for different images. Additionally features get lost, because its a global threshold value.

4.1 Thresholding

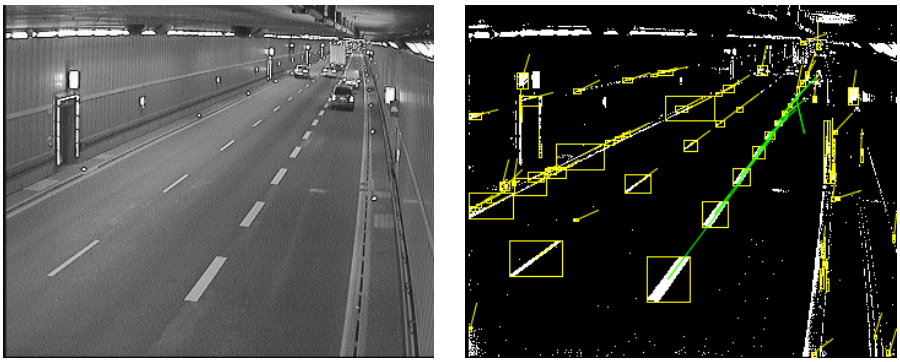
Normally the road markings are white and good visible, so the first approach will be to detect them via thresholding. However, a global thresholding method isn't useful for this case, see Figure 4.2. Using an adaptive threshold algorithm like *Niblack* [19], produces better results. It can retrieve more information, because of its adaptive local behavior and the parameters of the algorithm are more robust. This means that they can be used for a large variety of images. An example is shown in Figure 4.3 with the implementation deployed in a *Java Archive* of [29]. Additionally the implementation is described in [5].

However, different circumstances like wet streets, abrasion and soiled markings make this problem more difficult. This issue intensifies if dashed lane markings want to be clearly detected and merged.



(a) (b)

Figure 4.3: Adaptive threshold algorithm. The parameters of the *Niblack* algorithm are radius $r = 30$, $\kappa = 0.8$, $d_{\min} = 15$ and the region type is Gaussian. Normally the *Niblack* algorithm does not provide a Gaussian distribution, but this was added in addition by [5]. These parameters can be applied without change to different images, achieving a similar result.

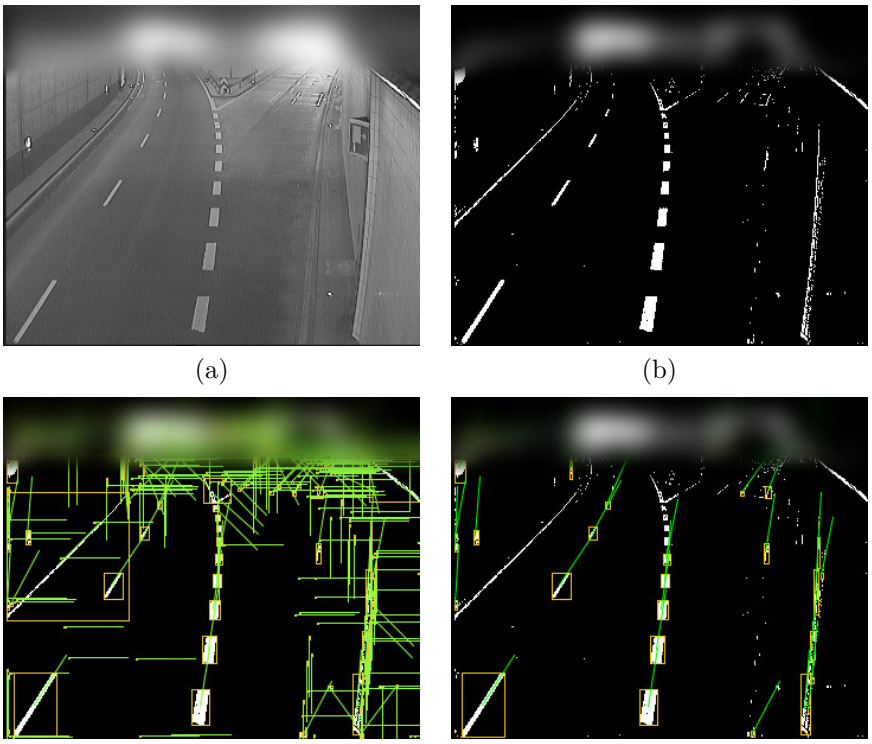


(a) (b)

Figure 4.4: Extracted road features. First the original picture (a) is processed with an adaptive thresholder, then the road markings are detected via region labeling (b). Note that unwanted regions are already eliminated.

4.2 Region labeling

On the result of the thresholding step, which is a binary image $\mathbf{B}$, region labeling is applied. Some pre-processing steps like closing or other morphological operators [34] could be performed on $\mathbf{B}$, but they are not necessary, because unwanted regions can be discarded later. Then the size, orientation and eccentricity can be calculated for every region, see Figure 4.4.



(a) (b) (c) (d)

Figure 4.5: Dashed lane marker detection example 1. On the original image **I** (a) an adaptive threshold algorithm is applied, resulting in **B** (b). Now for every found region, combined in a set $\mathbf{S} = \{r_1, r_2, \ldots, r_n\}$, the orientation and its bounding box is calculated and drawn (c). Based on their size and orientation many unwanted regions can be discarded (d), resulting in the final set $\mathbf{S}'$. The images were made partially obscured due to the confidentiality agreement.

4.3 Detection algorithm

The following section describes the steps of the road marker detection algorithm. Additionally the procedure is illustrated in Figure 4.5.

Input: The input for this algorithm is the original grayscale image **I**, the values for the size constraint $s_{\min}$ and $s_{\max}$ and the values for the orientation constraint $\phi_{\min}$ and $\phi_{\max}$.

1. The original grayscale image **I** is processed with an adaptive thresholding algorithm, which results in a binary image **B**.
2. Pre-processing steps, in this case morphological operators [34] like erosion, dilation, opening and closing, can be applied on **B** in order to remove image errors.

3. Region labeling[1] is performed on $\mathbf{B}$, resulting in a set $\mathbf{S} = \{r_1, r_2, \ldots, r_n\}$ of binary regions r_i.
4. For every binary region $r_i \in \mathbf{S}$ the size s_i, the orientation ϕ_i, the bounding box, the center, the central moments and the eccentricity are calculated.
5. Every region is discarded, which does not meet the given requirements. This includes the region size s_i, which has to be in the defined range of $s_{\min} < s_i < s_{\max}$ and its orientation $\phi_{\min} < \phi_i < \phi_{\max}$, but of course it is not limited to that. Note that the orientation can only be used, if the angle of the road is approximately known, which can be achieved by the vanishing point calculation, see Section 3.5. The size constraint discards regions, which are either too small or too big. This results in an adjusted binary region set $\mathbf{S}'$.
6. Now the remaining regions $r_i \in \mathbf{S}'$ can be merged together, based on their orientation. This can be done by searching for another region, which lies in the direction of the region itself, see Section 4.4.

Nevertheless the size and orientation constraint must be set individually for every image, because the dashed lane markings appear in different shapes and sizes. At first the size range can be set just to eliminate all the very small and very large regions. In addition to that, the size constraint must be flexible depending on where you are in the image. This is necessary, because the size of the individual markers decrease in the upper part of the picture. In this region a marker can be very small, but in the lower part of the image a region of this size must be rejected, otherwise the constraint would be useless. False detection can occur if the range values are not set correctly, see Figure 4.6.

4.4 Merging algorithm

An image with successfully merged markers and therefore the already known orientation can be the input for the watershed algorithm, in order to achieve a better segmentation, see Section 3.3. The following section describes the steps of the road marker merging algorithm. Additionally the procedure is illustrated in Figure 4.7.

Input: The inputs for this algorithm are the original grayscale image $\mathbf{I}$, the values for the size constraint $s_{\min}$ and $s_{\max}$ and the values for the orientation constraint $\phi_{\min}$ and $\phi_{\max}$. Additionally the start region R_s in step 2 is defined manually or from another merging pass.

1. The first steps, including the adaptive thresholding process, region labeling and the region parameter calculation, must be performed as

[1] Depth first labeling was used, but this has no effect on the result.

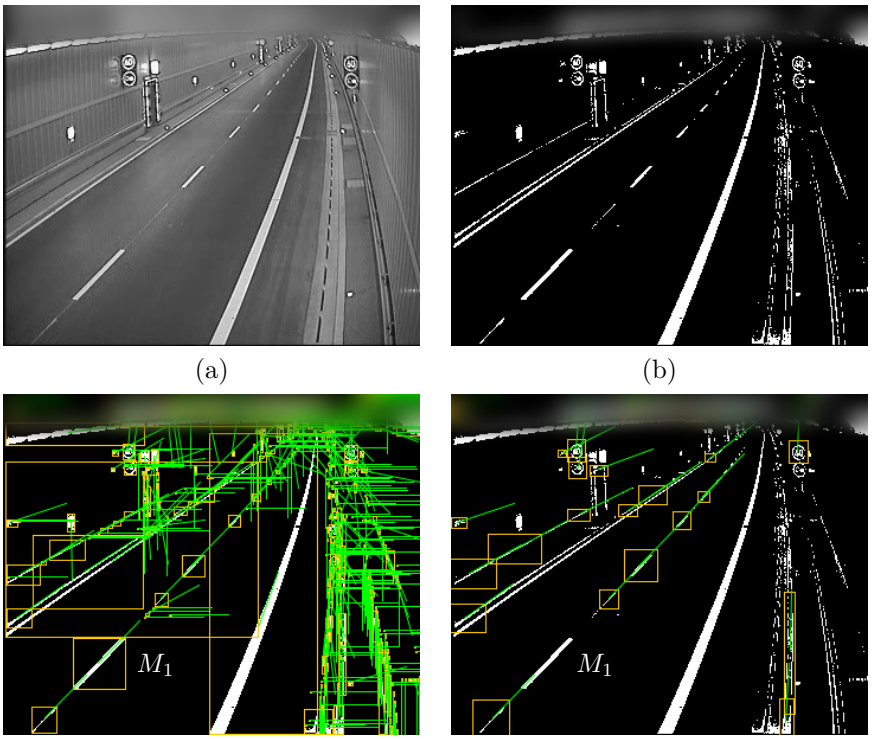


Figure 4.6: Dashed lane marker detection example 2. As before on the original picture (a), an adaptive threshold algorithm is applied (b). In (c) every detected region is drawn and in (d) unwanted regions are eliminated. Here the elimination based on the region size discards the good visible marker M_1, which should not happen. So the range values need to be tuned for every image, which is not optimal, see Section 4.4. The images were made partially obscured due to the confidentiality agreement.

well. Then based on its size and orientation every region is discarded, which does not meet the constraint parameters. These steps are described in Section 4.3 and in Figure 4.5.

2. A start region R_s is defined manually or from a previous merging pass.

3. The next region is searched within a specific length l in direction $\mathbf{d}$ of R_s. This is done by calculating a *Bresenham* line $\mathbf{L}$ from P_1, the center of R_s, to P_2, which can be calculated by

$$P_2 = P_1 + l \cdot \mathbf{d}, \tag{4.1}$$

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + l \cdot \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}. \tag{4.2}$$

Seen as a vector addition Eqn. 4.1 results in Eqn. 4.2.

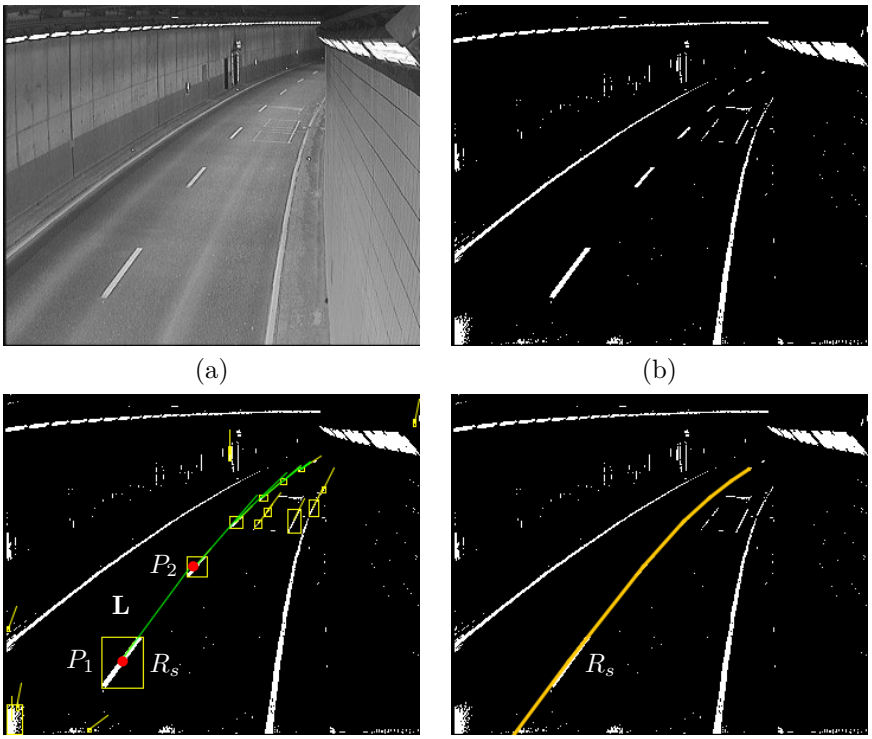
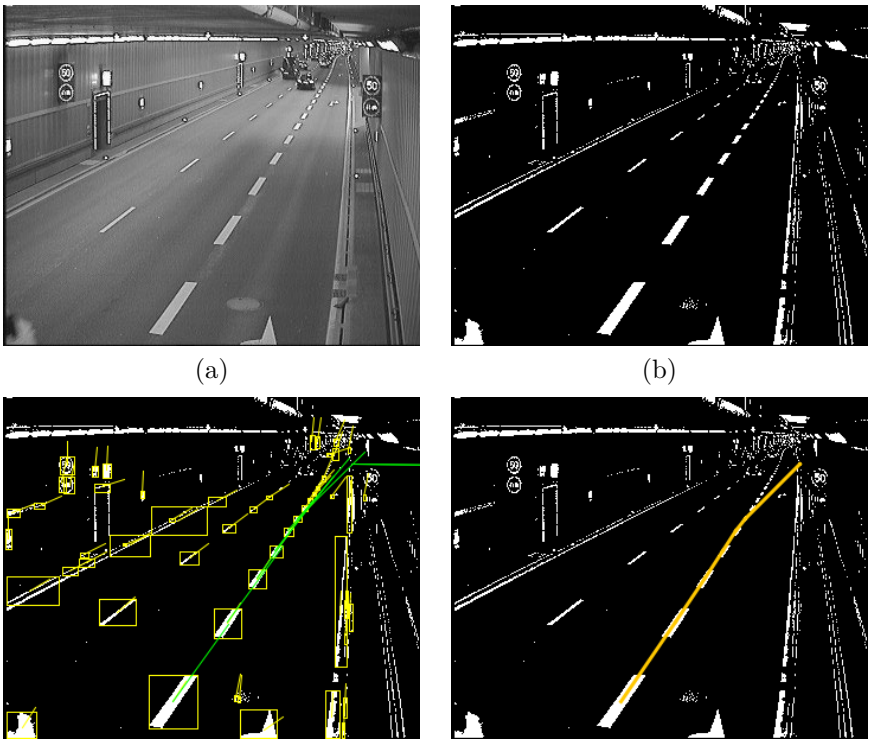


Figure 4.7: Dashed lane marker merge. The original image (a) is thresholded with an adaptive algorithm (b). Only the regions which satisfy the orientation and size constraint remain (c). A start region R_S is defined and based on it the next one is searched using a *Bresenham* line **L** from P_1 to P_2 which can be calculated by Eqn. 4.1. If another region is found, it will be connected with the first one (d). This is done multiple times.

4. If the *Bresenham* line **L** hits a bounding box from another region, see Figure 4.7 (c), then the procedure is repeated starting from step 2. However, the bounding box greatly depends on the orientation of the region. It would be better to use a specific perimeter around the center in order to check if the region is hit or not. Of course there are many other solutions to this problem and this is just one of them.

The algorithm can produce quite good results in the lower part of the image where the dashed lane markers are still large enough. However, in the upper part of the image the markers get smaller and smaller resulting in an almost quadratic region. For this regions the orientation becomes more and more unstable, meaning the merged line can drift off, see Figure 4.8. This is due to the small image resolution of 352×288, which is simply not suitable for a robust detection in this region.



(a) (b)

(a) (b)

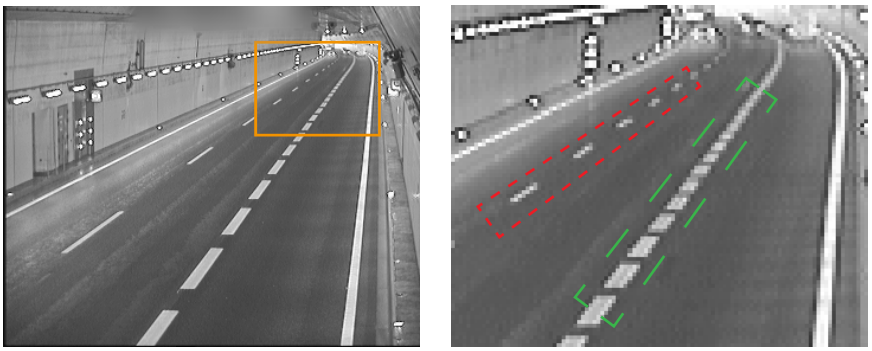
Figure 4.8: Dashed lane marker merge. The original image (a) is thresholded with an adaptive algorithm (b). Only the regions which satisfy the orientation and size constraint remain (c). The results (d) show how the algorithm can drift off the correct path due to other regions, which are very near to each other. Due to the small image resolution of 352×288 the orientations of the regions in this area become more and more undefinable.

4.5 Challenges

As already seen in Figure 4.8 (d) problems occur very quickly. The main difficulties are

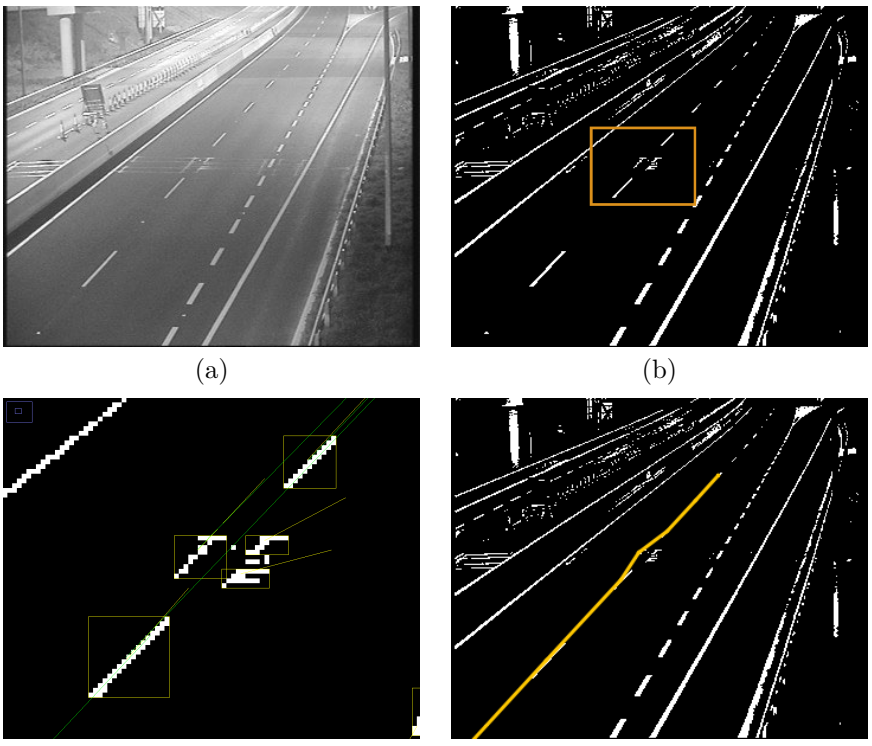
- the small resolution and therefore the small binary region size,
- the elimination of unwanted regions and
- the merging of regions in the upper part of the image.

Getting these problems under control is not that easy, see Figure 4.9. Unwanted regions, caused by reflections on wet streets or by dirty roads, can not be eliminated automatically, see Figure 4.10. Cars on the street produce unwanted regions as well, see Figure 4.11 (b), but the algorithm can still be successful in this case.



(a) (b)

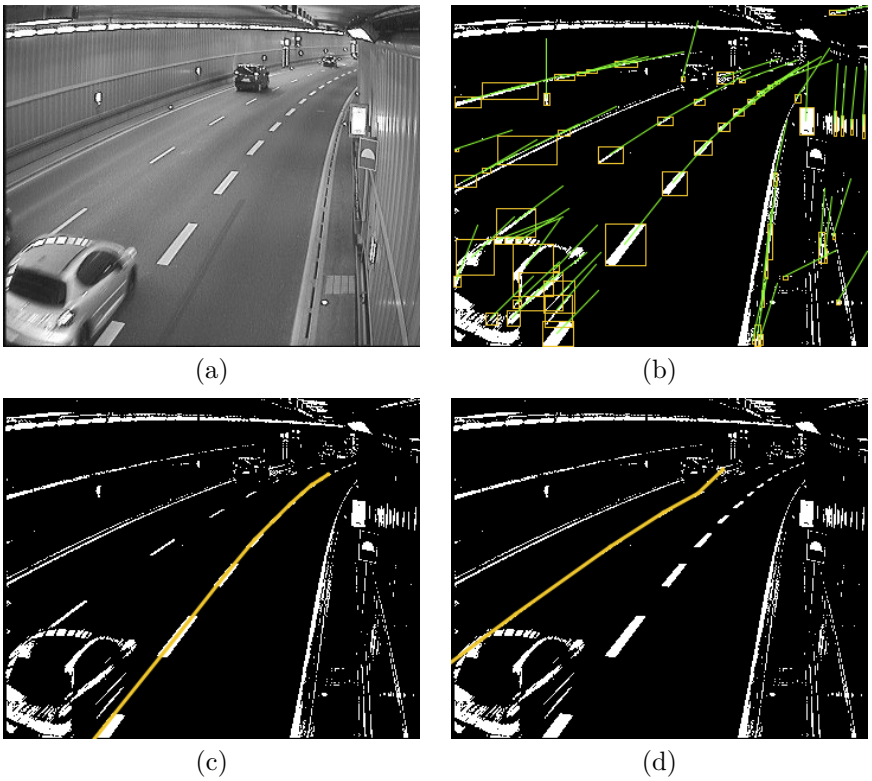
Figure 4.9: Marker merge problems 1. The normal image with the orange rectangle (a), which represents the enlarged area (b). In this part of the image the regions are very small and can merge together, which is a problem for the merging algorithm.



(a) (b)

(c) (d)

Figure 4.10: Marker merge problems 2. The original image (a) is thresholded with an adaptive algorithm (b). The enlarged area (c) shows the falsely detected regions, which are very similar to a normal dashed lane marker. The result of the marker merge algorithm (d).



(a) (b)

(c) (d)

Figure 4.11: Dashed lane marker merge result with cars. The original image with moving cars (a). Unwanted regions are produced by the car in the bottom left corner (b). The result of two individual merging steps is shown in (c) and (d). Starting with the right region the algorithm can still produce good results in this case.

Chapter 5

Incremental image warping principle

This concept is based on the idea, that the algorithm should straighten and rectify an image of a curved road without any prior knowledge about the content itself. In this rectified image the problem of detecting the road, which can be curved in any way, with its individual lanes is reduced to a simple problem of finding vertical lines. The incremental image warping is done by a stepwise transformation of every single horizontal image line in order to match the previous one as close as possible, see Figure 5.1. A general warping example on an actual input image is shown in Figure 5.2. The transformation is done by scaling and shifting all the individual image

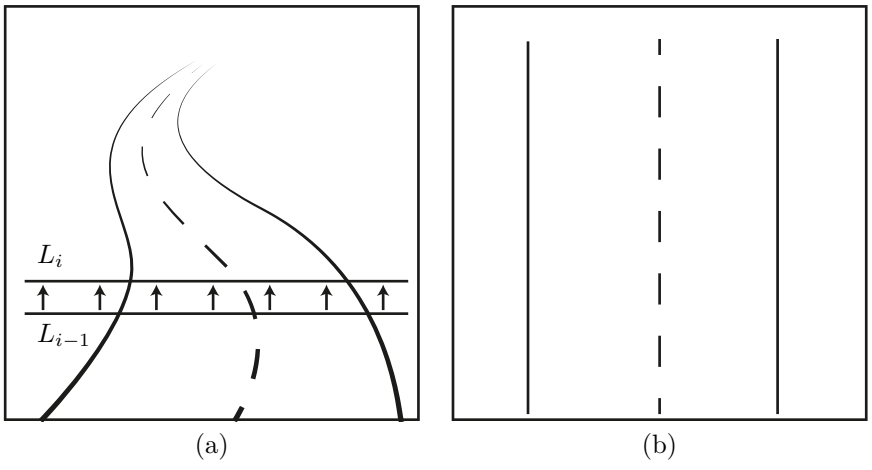
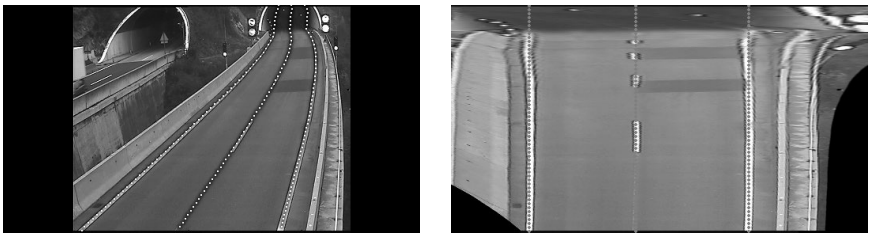


Figure 5.1: Incremental image warping. Every image line L_i of original image (a) is transformed to match the previous line L_{i-1} as good as possible. The optimal result (b) is a rectified and straightened image where all curved lane markings become vertical ones.



(a) (b)

Figure 5.2: General warping example. The original image (a) is transformed into a rectified one (b). In this transformed space all curved lane markings become straight ones. In (b) straight lines correspond to the curved road in (a).

lines individually in order to match the previous line as good as possible, see Section 5.2. This step rectifies the image using the lane markings and other image structures, but does not analyze the image itself. So the algorithm knows nothing of an overall road or lane model, but creates one in the end.

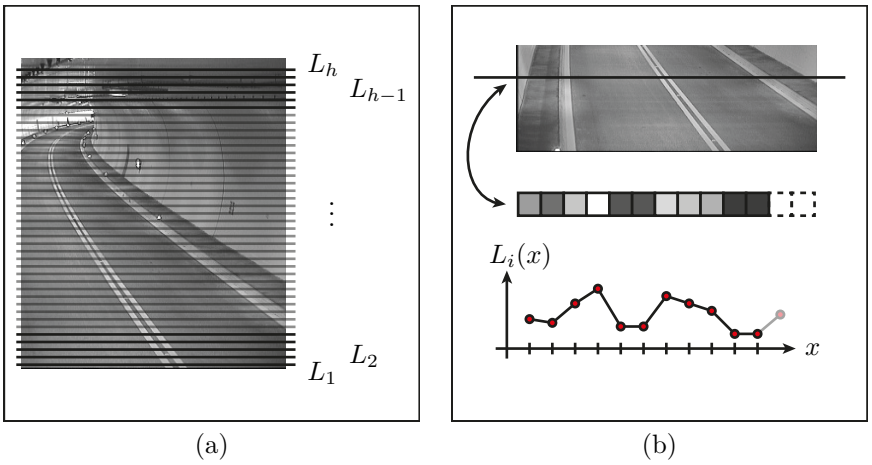
Resulting road model: Input is a grayscale image $\mathbf{I}$ of size $w \times h$ with its width w and height h. This image is divided into individual image lines, resulting in $L_1, L_2, \ldots, L_h$ lines. Every image line L_i has the same width w. Now every line L_i is transformed with a specific scale s_i and shift t_i in order to match the previous line L_{i-1} as good as possible. So every image line starting from L_2 has two parameters resulting in a road model, which has $n = (h-1) \cdot 2$ parameters.

5.1 One image line as a discrete 1D-signal

One image line L_i can be seen as a discrete 1D-Signal with a specific frequency. This frequency corresponds to the given sample rate. The camera already sampled the real world scene, so there is one sample point for every pixel position, see Figure 5.3.

Linear interpolation: Linear interpolation is used as an interpolation method to get intensity values on floating point positions and to gain a continuous signal, see Figure 5.3 (b). This method is sufficient for this case, however, more advanced methods like cubic interpolation or other variations can also be used. *Target To Source* mapping should be used in order to ensure that no holes exist in the transformed image, see Section 5.2.

Line scaling: Scaling one image line can be seen as scaling a 1D-Signal with a certain frequency. This frequency increases during downscaling and



(a) (b)

Figure 5.3: Image line as a 1D-signal. The grayscale image $\mathbf{I}$ of size $w \times h$, with its width w and height h, is divided into individual image lines, resulting in $L_1, L_2, \ldots, L_h$ lines (a). For every image line L_i, which has the same width w, linear interpolation is used in order to gain a continuous signal (b).

decreases during upscaling. Errors occur if the signal is scaled down, but the sample rate maintains the same. This would violate the *Nyquist–Shannon sampling theorem*[1]. However, this is not the case if the incremental image warping algorithm starts from the bottom of the image. Then every line is upscaled due to the road, which is converging to a vanishing point, see Figure 5.5.

Line shifting: Shifting the image line does affect the position of the original sample points with their intensity values. Linear interpolation needs to be applied again, if the displacement is in the decimal area and the intensity values on the original sample positions need to be known. Thats the case if two signals need to be compared, see Chapter 6.

Original signal: Repeated shifting and scaling, as it occurs in the incremental image warping algorithm, should not affect the original signal. The sample rate, the transformation values and the comparison area are defined in the comparison step, see Chapter 6. Now the original signal is transformed and sampled using linear interpolation at the defined sample rate over the comparison area.

[1] Wikipedia site (http://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem).

5.2 Transformation of a single image line

To transform one image line L_i by a scale factor s and a displacement factor t, every line position x needs to be transformed by a transformation matrix $T_{s,t}$, which is specified as

$$T_{s,t} = \begin{pmatrix} s & t \\ 0 & 1 \end{pmatrix}. \tag{5.1}$$

This specifies the transformation

$$\begin{pmatrix} x' \\ 1 \end{pmatrix} = \begin{pmatrix} s & t \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ 1 \end{pmatrix} \tag{5.2}$$

of one image line $L_i(x)$ from normal space to transformed space. The value of the transformed line $L'_i(x)$ at position x can be calculated from the original one by

$$L'_i(x) = L_i(x \cdot s + t). \tag{5.3}$$

The inverse transformation $T'_{s,t}$ is specified as

$$T'_{s,t} = \begin{pmatrix} \frac{1}{s} & -\frac{t}{s} \\ 0 & 1 \end{pmatrix}. \tag{5.4}$$

This specifies the transformation

$$\begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{s} & -\frac{t}{s} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x' \\ 1 \end{pmatrix} \tag{5.5}$$

of one image line $L'_i(x)$ from transformed space to normal space. The value of the original line L_i at position x can be calculated from the transformed one by

$$L_i(x) = L'_i\left(\frac{x' - t}{s}\right). \tag{5.6}$$

Now with Eqn. 5.3 and Eqn. 5.6 the complete transformation in both directions is specified.

5.3 Algorithms

In Alg. 5.1 the basic steps are shown in order to transform the given image $\mathbf{I}$ into the rectified one $\mathbf{I}'$. Especially the step of finding the best transformation values (Alg. 5.1 line 3) can be done in different ways. Chapter 6 describes what is needed to define the optimization problem and Chapter 7 describes what is needed to solve this problem.

Algorithm 5.1: Basic line matching algorithm. This algorithm describes the basic abstract steps in order to transform the given image $\mathbf{I}$ into the rectified one $\mathbf{I}'$.

```
1:  LineMatching(I)
      The given image I is transformed into I'.
      The current image line number is defined as i and the
      absolute transformation values as s̄_i and t̄_i, see Section 5.4.
      The algorithm returns the transformed image I'.
2:    for all L_i ∈ I do                    ▷ for every image line L_i ∈ I
3:      s_i, t_i ← FindBestValues(L_i, L_{i-1})
4:      s̄_i ← ∏_{k=1}^{i} s_k                          ▷ absolute scale value
5:      t̄_i ← t̄_{i-1} · s_i + t_i                      ▷ absolute shift value
6:      I' ← TransformAndAdd(L_i, s̄_i, t̄_i)          ▷ draw in the image I'
7:    end for
8:    return I'
9:  end
```

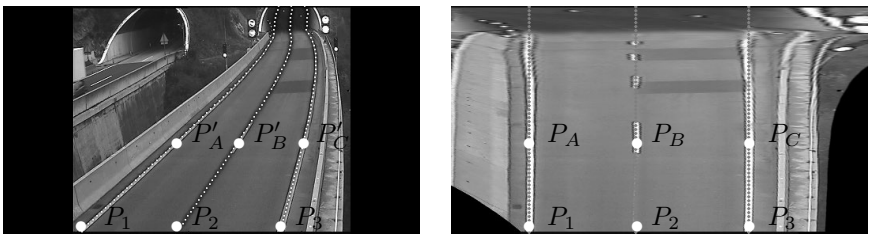
```
10: FindBestValues(L_i, L_{i-1})
      The two image lines L_1 and L_{i-1} are given.
      Chapter 6 describes how to compare them and Chapter 7 how to
      define the abstract optimization problem OP.
      The algorithm returns the best scale s and shift t values,
      this is described in Section 7.4.
11:   OP ← ⟨L_i, L_{i-1}⟩                    ▷ The abstract OP is defined
12:   return s, t ← SolveTheOtimizationProblem(OP)
13: end
```

5.3.1 Incremental image warping

Input and Output: Given is the grayscale input image $\mathbf{I}$ of size $w \times h$ with its width w and height h. The procedure returns the straightened and rectified image $\mathbf{I}'$.

1. The input image $\mathbf{I}$ is divided into individual image lines $L_1, L_2, \ldots, L_h$. Every image line L_i has the same width w.
2. Every line L_i is transformed with a specific scale s_i and shift t_i in order to match the previous line L_{i-1} as good as possible. This is described in Chapter 6 and Chapter 7.
3. Every transformed line L'_i is calculated using linear interpolation and is drawn into the transformed image $\mathbf{I}'$. The width of $\mathbf{I}'$ can be greater than the original one.

Now that the rectified image $\mathbf{I}'$ and all the parameters for every image line L_i are found, the actual road model is defined.



(a) (b)

Figure 5.4: Road detection example. The original image (a) is transformed into a rectified one (b). In this transformed space all curved lane markings become straight ones. The starting points P_1, P_2 and P_3 are the same in (a) and in (b). Straight lines in (b) correspond to the curved road in (a). The points P_A, P_B and P_C in (b) still have the same x-value as P_1, P_2 and P_3 and correspond to the transformed points P'_A, P'_B and P'_C in (a).

5.3.2 Road detection

Given a street image $\mathbf{I}'$, which is completely rectified and straightened, the road detection only needs the x-values of the start position, see Figure 5.4. These can either be obtained automatically or they are given by the user.

Input and Output: Given is the rectified image $\mathbf{I}'$ of size $w' \times h$ with its width w', which may be different than the original one, and its height h and all the transformation values[2] for every image line L_i, resulting in a road model, which has $n = (h-1) \cdot 2$ parameters.

1. The x-values of the start position of the different lanes are defined either automatically or by the user.
2. For every image line L_i the transformation parameters are used in order to transform the x-values from the transformed space into the original one. So the road model, which has $n = (h-1) \cdot 2$ parameters, defines the shape of the whole road on every image position.

To summarize, given a correct and complete rectified image only the start position of the different lanes needs to be known in order to obtain a correct road detection.

5.4 Relative and absolute transformation

In the matching algorithm the scale s_i and the displacement factor t_i are evaluated for every line L_i individually. These two values specify the relative transformation from one image line L_i to its previous one L_{i-1}. In order to get the absolute transformation, these values must be summed up in

[2] All the scale $s_1, s_2, \ldots, s_h$ and shift values $t_1, t_2, \ldots, t_h$.

a specific way. The first image line L_0 has no predecessor and therefore no transformation values. The position x for the next image line $L_1(x)$ is transformed by s_1 and t_1, so $L'_1(x)$ is defined as

$$L'_1(x) = L_1(x \cdot s_1 + t_1). \tag{5.7}$$

The next image line $L_2(x)$ is transformed again by s_2 and t_2, so $L'_2(x)$ is defined as

$$L'_2(x) = L_2(x \cdot s_2 + t_2). \tag{5.8}$$

The image line L_1 was already transformed. In order to perform this transformation to L_2 it can be written as

$$L''_2(x) = L_2((x \cdot s_1 + t_1) \cdot s_2 + t_2) \tag{5.9}$$

$$= L_2(x \cdot s_1 \cdot s_2 + t_1 \cdot s_2 + t_2). \tag{5.10}$$

So L'_i represents the transformed image line based on the predecessor and L''_i represents the absolute transformed line. Now the absolute values for the scale $\bar{s}_n$ and the shift factor $\bar{t}_n$ can be defined as

$$\bar{s}_n = \prod_{k=1}^{n} s_k \quad \text{and} \quad \bar{t}_n = \bar{t}_{n-1} \cdot s_n + t_n, \tag{5.11}$$

resulting in

$$L''_2(x) = L_2(x \cdot \bar{s}_2 + \bar{t}_2). \tag{5.12}$$

So every line pair has relative and absolute transformation values, this is illustrated in Figure 5.5.

5.5 Correlation between normal and transformed space

The correlation between the normal image $\mathbf{I}$ of size $w \times h$ with its width w and height h and the transformed one $\mathbf{I}'$ of size $w' \times h$ with its width w', which may be different than the original one, and height h, is given by the resulting transformation model, which has $n = (h-1) \cdot 2$ parameters. This model describes the transformation for every image position. An example is shown in Figure 5.6.

5.6 Results

The algorithm was tested with different images, see also Chapter 8. It worked quite well for images with a good lane marker quality, see Figure 5.2 and for images where no clipping occurs, see Figure 5.7. Images where clipping occurs in the lower part of the image and only dashed road markings are

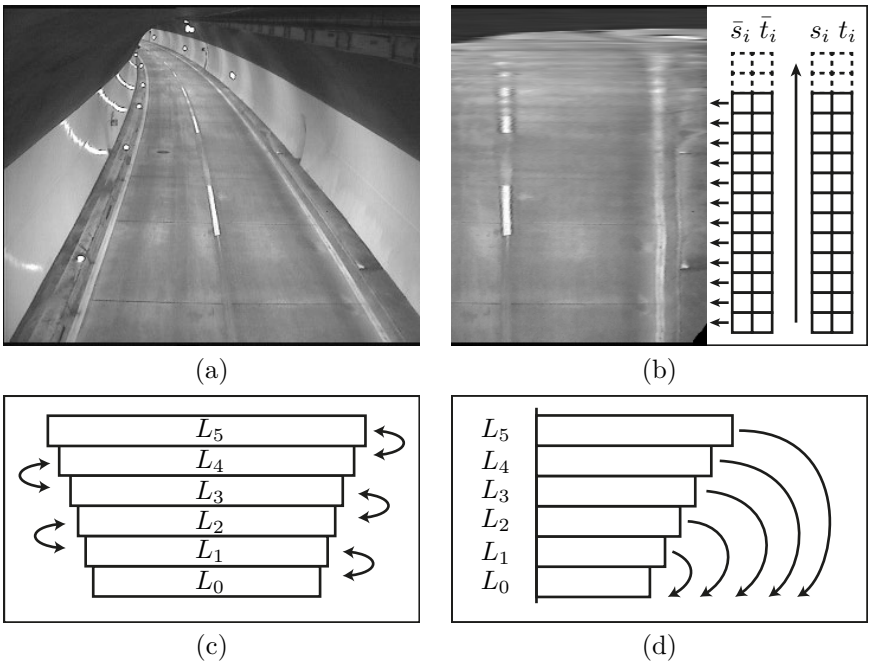


Figure 5.5: Relative and absolute values. The original image (a) is transformed into a rectified one (b). Every single image line L_i is transformed by its relative values s_i and t_i to match the previous line L_{i-1} as good as possible (c). Based on the previous transformations $s_{i-1}, \ldots, s_1$ and $t_{i-1}, \ldots, t_1$, the absolute values $\bar{s}_i$ and $\bar{t}_i$ can be calculated by Eqn. 5.11 for every image line (d).

present propose a problem, see Figure 5.8. Some of them can be handled by a good designed score function, see Chapter 6 and a good optimization problem solving method, see Chapter 7.

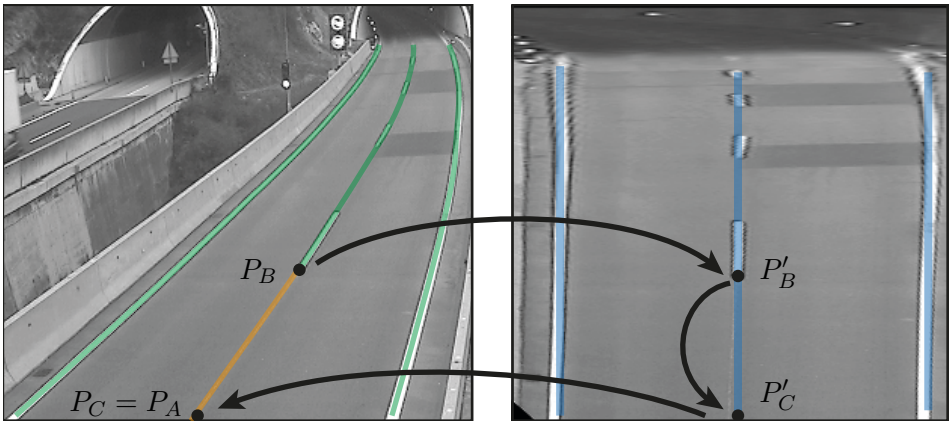
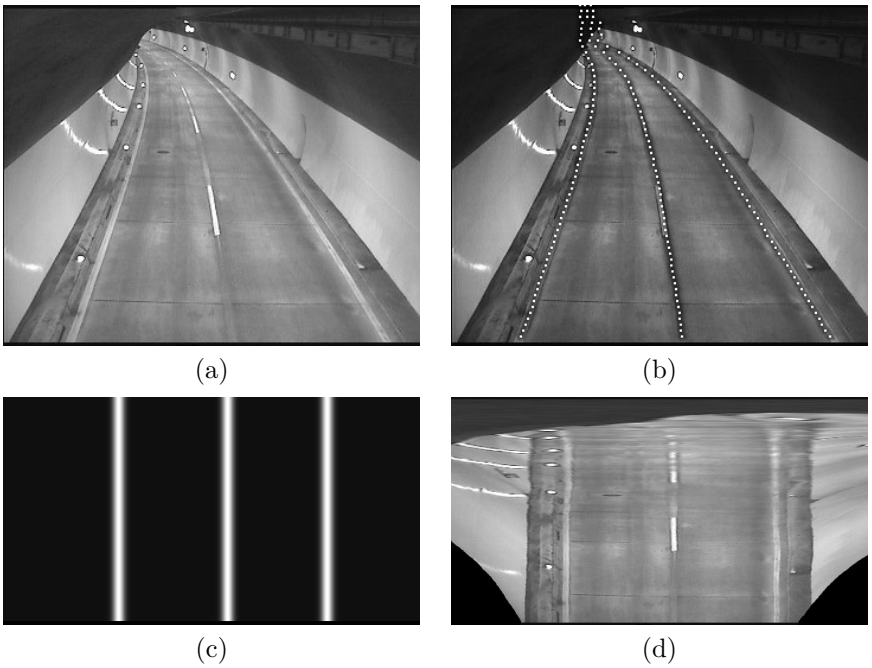


Figure 5.6: Correlation between normal and transformed space. The image $\mathbf{I}$ in normal space (a) is transformed into the rectified one $\mathbf{I}'$ (b) by the proposed incremental image warping algorithm. Given the starting point of the middle line P_B the corresponding starting point P_A, which is the direct extension of the middle line, can be calculated very easily. At first $P_B = (x, y)$ is transformed into $P'_B(x', y)$ given the transformation on the corresponding image line y. Note that this transformation does not change the y-value. Then in order to get the desired point P'_C the y-value is set to 0, resulting in $P'_C = (x', 0)$. Finally P'_C must be transformed into P_C, which is located in normal space. This is especially easy for image line $y = 0$, because no transformation must be performed, resulting in $P'_C = P_C = P_A$. A complex line in (a) becomes one point in (b).



(a) (b)

(c) (d)

Figure 5.7: Incremental image warping example. The original image (a) with the result (b). The reference line image, see Chapter 6, which has now three lines (c), with the transformed image (d). Note that (c) and (d) are wider. If the position of the middle lane is known in advance, it can improve the matching algorithm. It is possible to detect the middle in the transformed image as well.

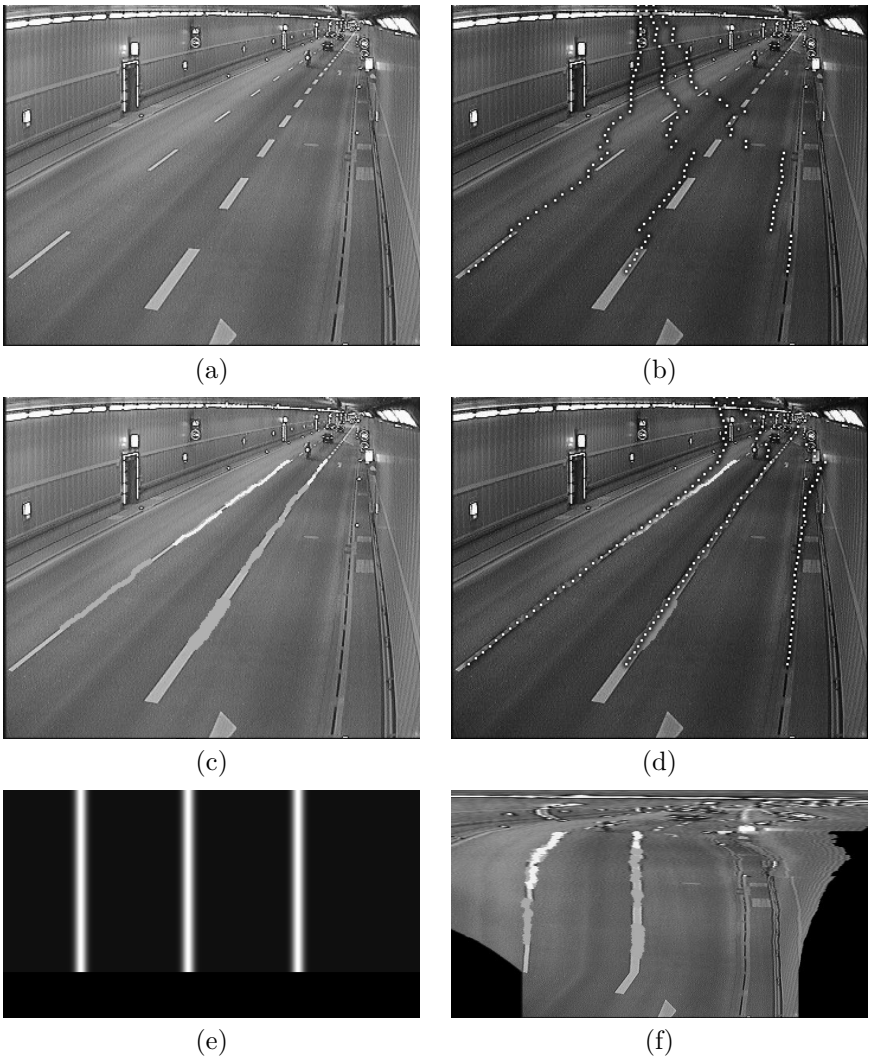


Figure 5.8: Incremental image warping challenges. The original image (a) with the result (b). At this state the algorithm can't work quite well due to the lack of good visible road markings. The original image with painted road markings (c) with the result (d). Now the result is better, because the dashed lane markings are merged together. Note that the right boundary marking is still missing. The reference line image (e) with the transformed image (f).

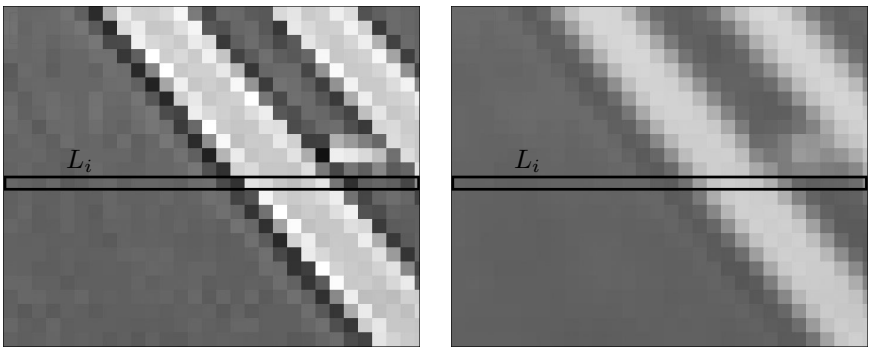
Chapter 6

Comparing two image lines

The incremental image warping algorithm is based on the transformation of individual image lines in order to match another line as good as possible. In order to evaluate this, a score function F needs to be defined that

- can compare two lines, L_1 and L_2, with each other,
- should provide a local minimum if these two lines are similar,
- should be able to restrict the comparison to a specific area and
- should be able to adjust the sample rate.

This can be done by a simple L_2 norm between two image lines over a specified range $[0, \ldots, w]$ and with a defined sample rate. A smoothing filter is applied as a pre-processing step in order to remove image noise and other image errors, see Figure 6.1. However, smoothing the whole image with the



(a) (b)

Figure 6.1: Smoothing the image. Taking a close look at one of the input images reveals that the images are very noisy and the cameras already apply a sharpening filter as a pre-processing step (a). One part of the image line L_i is shown in both images. Since the score function F only compares two image lines with each other, a smoothing filter is applied in order to enhance the matching algorithm (b).

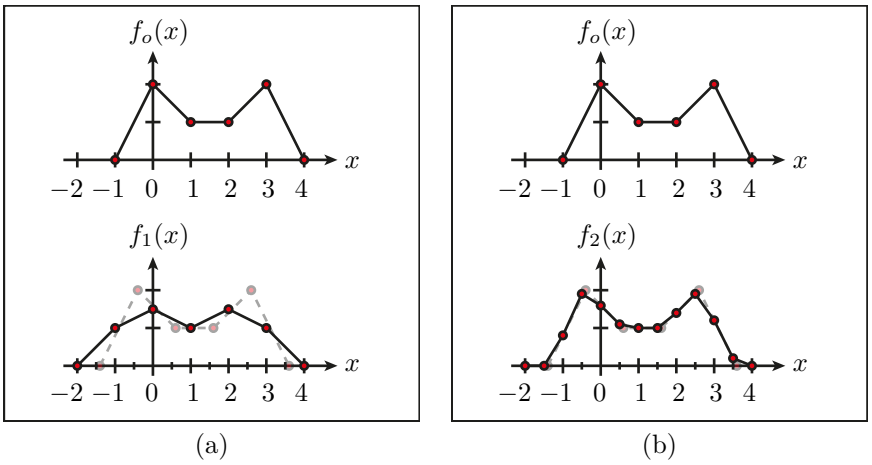


Figure 6.2: Sampling step width. The original discrete signal $f_o(x)$ is transformed by $T_{s,t} = (1, -0.4)$, which means that the original scale remains the same, but the signal is shifted by $t = -0.4$. In (a) the original sample step width of $w = 1$ remains the same, resulting in a function $f_1(x)$ with a huge divergence from the original one. In (b) the amount of samples was doubled and therefore the step width reduced to $w = 0.5$. The resulting function $f_2(x)$ is very similar to $f_o(x)$. So an oversampling of the transformed function is needed in order to find the optimal transformation T_{opt}.

same strength destroys useful image information in the upper part of the image. The strength of the filter should be diminished based on the image position, see Section 8.1.3 and Figure 8.9. Increasing the sample rate provides a more accurate score value for transformations with sub-pixel accuracy, see Figure 6.2.

6.1 L_2 norm score

The score function F can be defined as

$$F = \sum_{x=0}^{w} \left[L_1(x) - L_2(x)\right]^2, \tag{6.1}$$

by using a simple L_2 norm of two image lines over a specified range $[0, \ldots, w]$. To use this norm in the incremental image warping algorithm, the score function F becomes dependent on the scale s and the shift t, resulting in

$$F(s,t) = \sum_{x=0}^{w} \left[L_1(x) - L_2'(x)\right]^2 \mapsto \min. \tag{6.2}$$

The image line L_2 will be transformed by s and t, resulting in L_2', using Eqn. 5.3. Now this score function calculates one score value for one transformation $T(s,t)$. Using the brute-force matcher (Section 7.4.1), a visualization

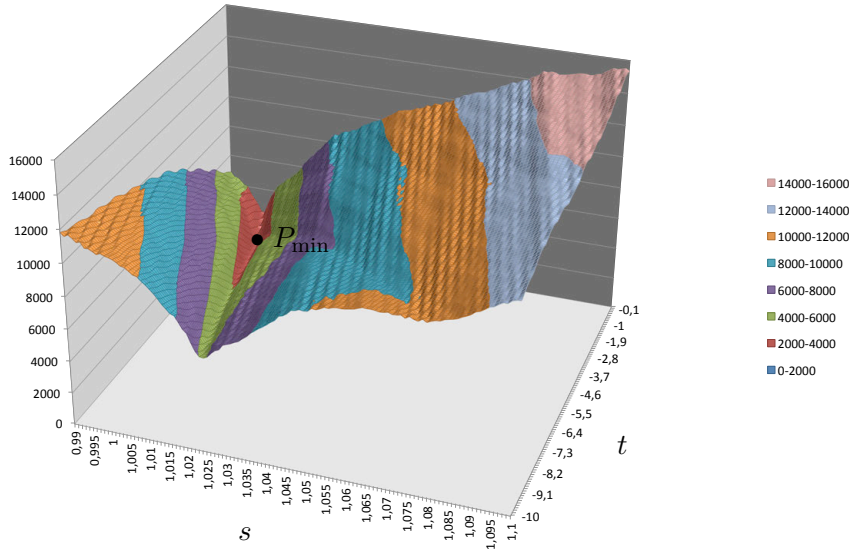


Figure 6.3: Score function visualization. The line score function $F(s,t)$, see Eqn. 6.2, is shown in a specific range around the local minimum $P_{\min}$. The scale value s is in the range of $s = [0.99, 1.1]$ and the shift value t is in $t = [-0.1, -10]$.

of $F(s,t)$ in a specific area around the local minimum $P_{\min}$ can be created, see Figure 6.3.

6.2 L_2 norm score with a weighting line

This score function is based on a simple L_2 norm between two image lines, multiplied with the weighting line at every position. So $F(s,t)$ is specified as

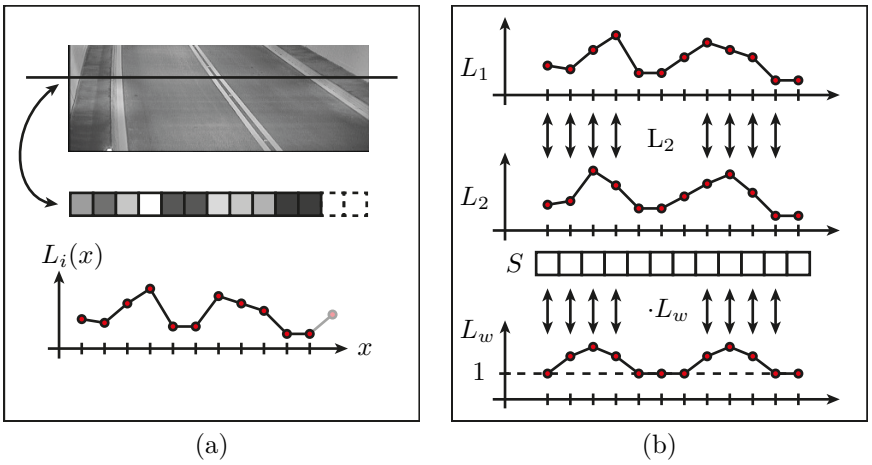
$$F(s,t) = \sum_{x=0}^{w} \left[L_1(x) - L_2'(x)\right]^2 \cdot L_w(x) \mapsto \min . \tag{6.3}$$

The weighting line L_w must have the same width as L_1 and L_2', see Figure 6.4.

6.2.1 Weighting line on positions

This weighting line L_w is used as a location based weighting function for the calculation of the matching score. The Gaussian function $g(x)$ with its variance σ and its shift value μ is defined as

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} . \tag{6.4}$$



(a) (b)

Figure 6.4: Score calculation with a weight function. Two image lines, L_1 and L_2, are retrieved from the original image (a). The score of these two lines can be calculated using a L_2 norm combined with a weighting function L_w, see Eqn. 6.3. L_w simply weights the result S from the L_2 norm on specific positions (b).

Eqn. 6.4 represents the normalized Gaussian, which has

$$\int_{-\infty}^{\infty} g(x) = 1 \tag{6.5}$$

as its property. A factor s was added to scale the Gaussian

$$g'(x) = g(x) \cdot s \tag{6.6}$$

and is therefore increasing or decreasing the impact of the weighting function. The weighting line $L_w(x)$, which is initialized on every position with 1 and has a given width of w, is now defined as the accumulated sum of all scaled Gaussians $g'_x(x)$

$$L_w(x) = \sum_{x=0}^{w} \left[1 + g'_1(x) + g'_2(x) + \ldots + g'_n(x)\right] \tag{6.7}$$

for every line position x, see Figure 6.5. A successful rectification is shown in Figure 6.6. Nevertheless problems can occur, if the continuous road markings are being clipped in the upper part of the image, see Figure 6.7.

6.2.2 Weighting line from the image

The weighting line L_w can also be retrieved directly from the image content. The idea is that lane markings appear in the image as bright lines, see Figure 6.8. So for a fixed amount of image lines n, the image content in

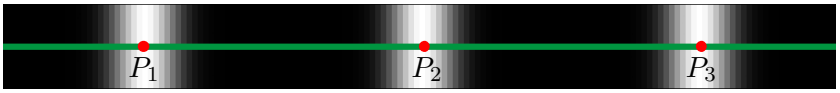


Figure 6.5: Weighting line zoom. This image is a zoomed image of the weighting line $L_w(x)$, which represents a location based weighting function for the calculation of the comparison score, see Eqn. 6.3. Normally the line would have a height of 1 pixel, which is shown by the green line. For a better visualization several lines are drawn above each other. The red dots mark the positions P_1, P_2 and P_3, where the centers of the three individual Gaussian functions $g_1'(x), g_2'(x)$ and $g_3'(x)$ are. The line is created using Eqn. 6.7. Black represents a function value of 1.

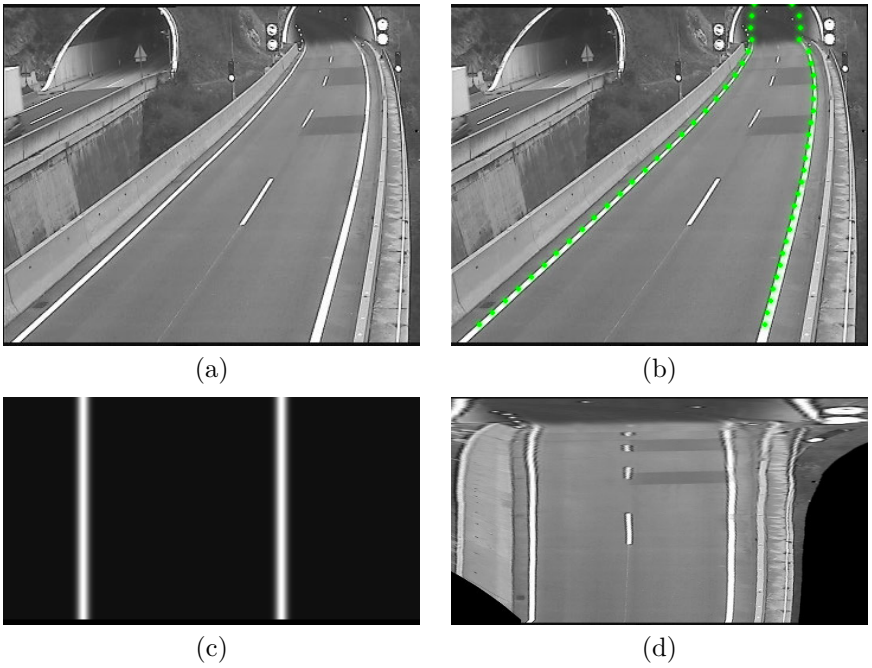
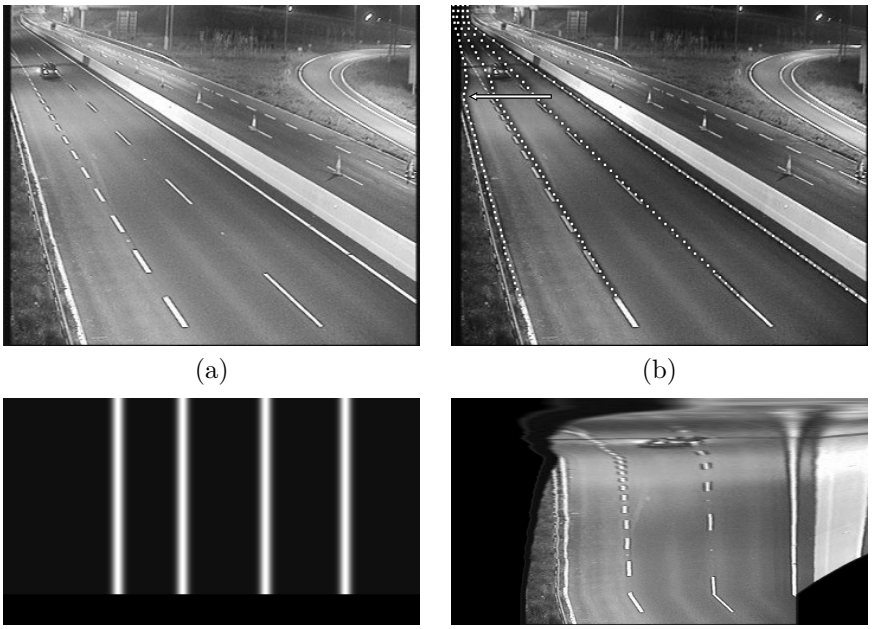


Figure 6.6: Weighting line example. The original image (a) with the result (b). The weighting line image (c) with the transformed image (d). Note that (c) and (d) are wider. The detection of the middle lane in (d) can be done afterwards.

the transformed space, see Figure 6.8 (c), is summed up and normalized. This results in different weighting lines $L_{w,i}(x)$ for every image line i, see Figure 6.8 (b). These values are then used for the general score calculation, see Eqn. 6.3.

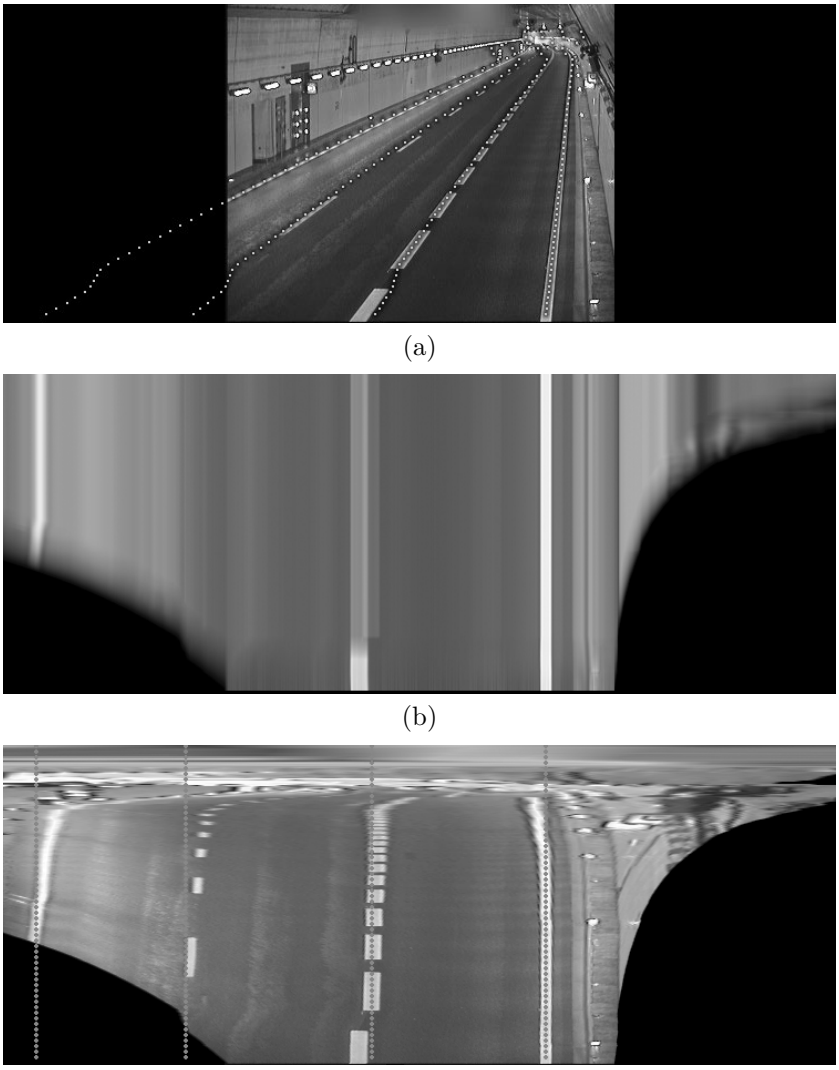
However, this method is very error-prone and is not usable if other bright structures are in the image. Nevertheless quite good results can also be produced, see Figure 6.9.



(a) (b)

(c) (d)

Figure 6.7: Weighting line problem. The original image (a) with the result (b). The weighting line image, which has now four lines (c) with the transformed image (d). The algorithm worked quite well up to the point where the left continuous road line disappears (white arrow). The deviation before that is caused by the lanes, which become physically narrower. This proposes a problem for the algorithm, which uses a fixed width for the different lanes.



(a)

(b)

(c)

Figure 6.8: Weighting line from the image example 1. The original image with additional width and the result on top (a). The visualization of every individual weighting line $L_{w,i}(x)$ for every image line i (b). The transformed result (c).

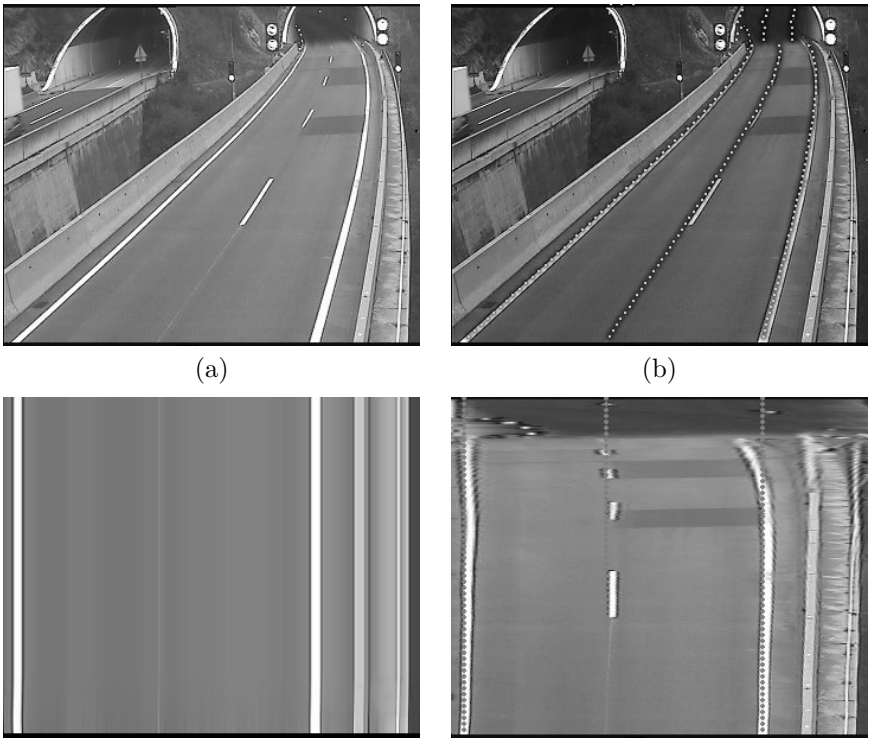
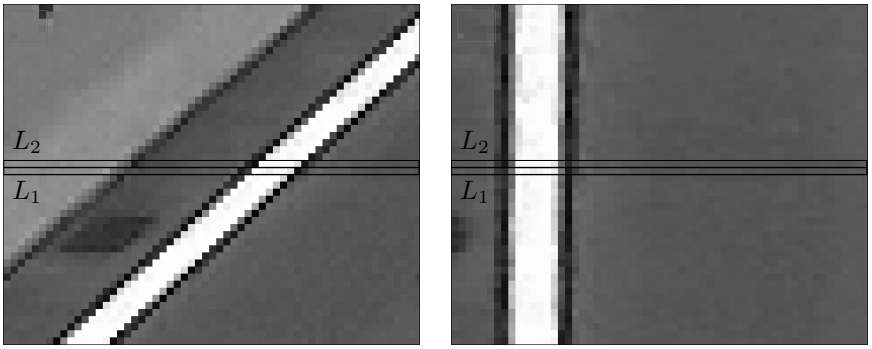


Figure 6.9: Weighting line from the image example 2. The original image (a) with the detection result (b). A fixed amount of image lines n in the transformed space (d) is summed up and normalized resulting in an image of different weighting lines $L_{w,i}(x)$ (c). The summation stops after a fixed amount of image lines, so that the resulting weighting line does not change anymore, leading to a more robust incremental image warping. This behavior also causes the disappearance of the dashed lane markers in (c).

Chapter 7

Image line matching as an optimization problem

To complete the incremental image warping algorithm an optimization problem needs to be defined that uses the score function $F(s,t)$ in order to find the best transformation values that match two adjacent lines as good as possible, see Figure 7.1. Now a solver needs to be created that uses the score function and the image lines and combines it with other useful information.



(a) (b)

Figure 7.1: Image line matching as an optimization problem. The original image (a) and the result of the incremental image warping algorithm (b). The score function $F(s,t)$ defines the score value given L_1, L_2, the scale s and the shift t, see Chapter 6. Now the best transformation values need to be found in order to transform the image lines from (a) to (b).

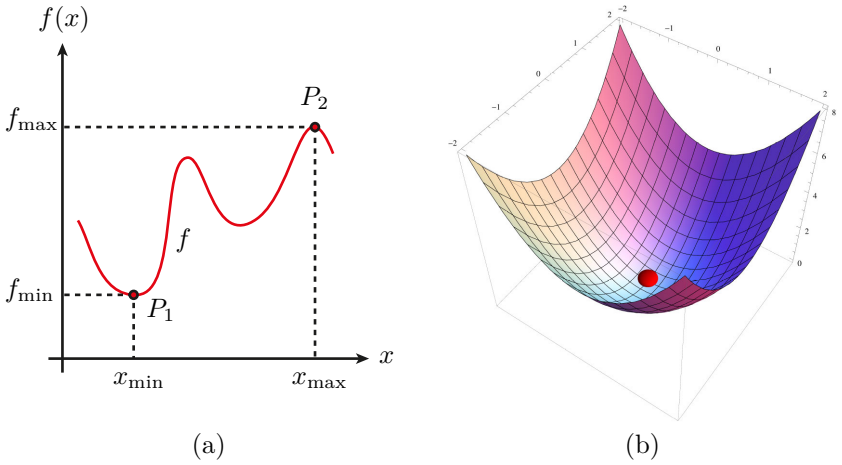


Figure 7.2: Minimization examples. In the 1D-case P_1 marks the minimum of the plotted function $f(P_1) = f_{\min}$ (a). In the 2D-case the red dot marks the position of the minimum (b).

7.1 General definition

The general definition of the optimization problem can be defined by finding the minimum of an unknown function $f(x_1, x_2 \dots x_n)$ with parameters $x_1, x_2 \dots x_n$

$$f(x_1, x_2 \dots x_n) \mapsto \min . \tag{7.1}$$

So the parameters are sought in order to minimize this function. In Figure 7.2 two examples are shown.

Now a specific optimization problem can be created, which uses the line score function $F(s,t)$, see Eqn. 6.3, in combination with other desired properties, see Figure 7.3. It can be defined to enhance or to restrict the line matching algorithm. Enhancement can be done by predicting a better initial guess and restriction by penalizing offset values for s and t. However, it is possible to define the optimization process in many other ways to enforce the desired behavior of the overall result.

7.2 Local problem definition

In order to transform the original image into a rectified version, as already seen in Chapter 5, the line matching algorithm performs a stepwise transformation of every single horizontal image line, see Figure 5.1. Therefore a new optimization problem must be defined for every image line pair L_1 and L_2 individually.

The actual matching is done in the transformed image space. This means that the previous line is the already transformed line L_2' and the next line L_1

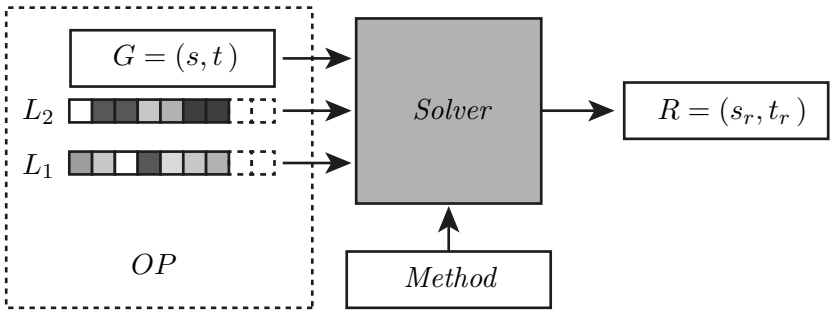


Figure 7.3: Flowchart of the optimization process. The input for the *Solver* on the left side is the initial guess $G = (s, t)$ with the start values for the scale s and the shift t and the two image lines L_1 and L_2. The *Solver* can be implemented as a more or less intelligent brute-force algorithm or by using a state of the art optimizer. The result $R = (s_r, t_r)$ returns the best scale and shift values in order to match the two image lines together. This behavior must be specified in the *Method*, which can be seen as another input for the *Solver*.

is still in normal space. Therefore the initial guess $G = \left(\bar{s}, \bar{t}\right)$ is the summed version of all the scale and shift values, see Section 5.4 and Figure 7.4. So the optimization problem

$$OP \mathrel{\widehat{=}} \left\langle G, L_1, L_2' \right\rangle \tag{7.2}$$

has the initial guess G and both image lines L_1 and L_2' as input.

7.2.1 Prediction enhancement

The prediction enhancement is calculated from the previous transformation values $\bar{s}_p$ and $\bar{t}_p$ and the current ones $\bar{s}$ and $\bar{t}$. The difference between those values

$$d_s = \bar{s} - \bar{s}_p \quad \text{and} \quad d_t = \bar{t} - \bar{t}_p \tag{7.3}$$

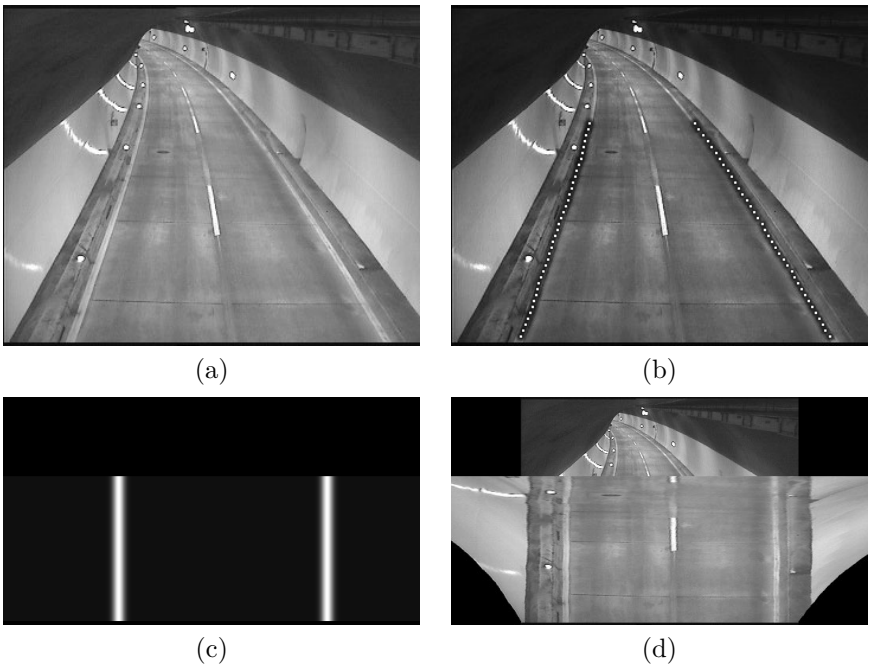
can then be used to update the initial guess G. A better way to do this, is to calculate a running average with the previous differences $d_{s,p}$ and $d_{t,p}$. So with

$$\bar{d}_s = \frac{d_s + d_{s,p}}{2} \quad \text{and} \quad \bar{d}_t = \frac{d_t + d_{t,p}}{2}\,, \tag{7.4}$$

the initial guess $G = \left(\bar{s} + \bar{d}_s, \bar{t} + \bar{d}_t\right)$ can be updated. The optimization problem

$$OP \mathrel{\widehat{=}} \left\langle G, L_1, L_2' \right\rangle \tag{7.5}$$

has the updated initial guess G and both image lines L_1 and L_2' as input.



(a) (b) (c) (d)

Figure 7.4: Line matching example. The original image (a) with the partial result (b). The reference line (c) with the partial transformed image (d). Note that (c) and (d) are wider. The algorithm stopped intentionally at a given position to see the huge difference between the next normal image line and the previously transformed one.

7.2.2 Line offset restriction

The line offset restriction can be created by penalizing the differences between the current and the previous optimization values. So the optimization problem

$$OP \mathrel{\widehat{=}} \langle G, L_1, L_2', \bar{s}_p, \bar{t}_p \rangle \tag{7.6}$$

has the initial value G, both image lines L_1 and L_2' and the previous values $\bar{s}_p$ and $\bar{t}_p$ as input. The score function for the optimizer is updated to use the line score function F, the difference between the scale d_s and between the shift value d_t. So the final score value

$$V = \alpha \cdot F + \beta \cdot d_s + \gamma \cdot d_t \tag{7.7}$$

is the summation of all the individual scores F, d_s and d_t weighted by α, β and γ. To sum these values up and weight them in a convenient way F, d_s and d_t should be normalized, see Figure 7.5.

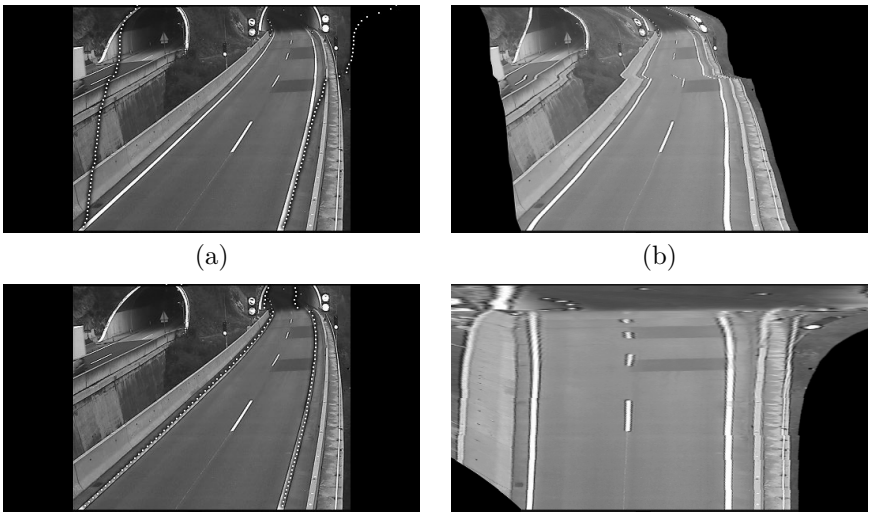


Figure 7.5: Offset restriction example. The offset restriction should prevent huge differences in the transformation values from one line pair to the other, resulting in a visible jump. If the restriction has too much influence, it becomes too rigid and a correct rectification is not possible anymore (b). The result is shown in (a). The same score function is used, but the offset restriction has less influence (d). In this case the result is quite accurate (c).

7.3 Global problem definition

Incremental image warping can also be done in a global way, which has many advantages and some disadvantages. The same problem solver can be used, only the data and the initial guess change. Given is an image $\mathbf{I}$ with a specific height h and width w. The initial guess $G = (s_1, t_1, s_2, t_2, \ldots, s_{h-1}, t_{h-1})$ has a scale s_i and a shift value t_i for every single image line pair, resulting in a length of $(h-1) \cdot 2$ values. The optimization problem

$$OP \mathrel{\widehat{=}} \langle G, \mathbf{I} \rangle \tag{7.8}$$

has now the initial guess G and the complete image $\mathbf{I}$ as input. Now the optimizer needs to find a local minimum in this huge space.

The drawback of this approach is this huge search space, which has a dimension of $D = (h-1) \cdot 2$. Therefore the runtime can be quite long, greatly depending on the size of the image. Nevertheless this approach has many advantages, because

- the whole image information both in transformed and original space and
- the relationship between all transformation values are available.

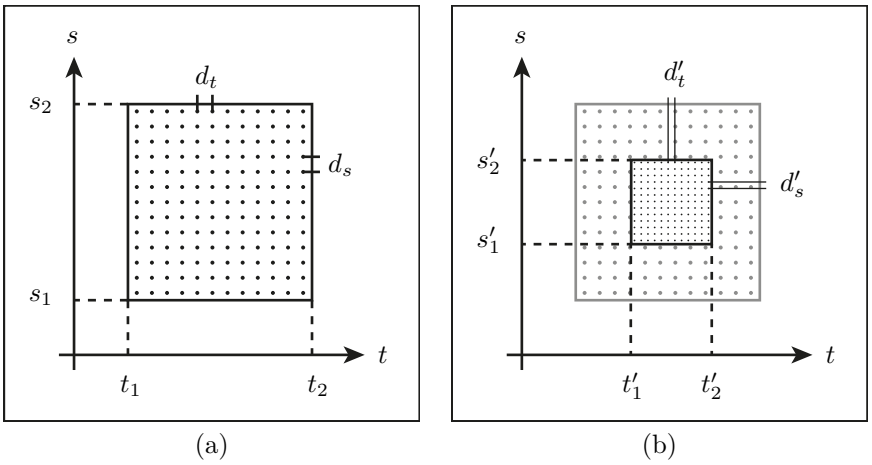


Figure 7.6: Brute-force algorithm. The first pass of the brute-force algorithm covers an area from s_1 to s_2 and from t_1 to t_2 (a). The taken samples and the resulting shift d_s and scale d_t step width are indicated by the black dots. The range values can be updated in order to cover a smaller area more precisely (b). Now s'_1, s'_2, t'_1, t'_2, d'_s and d'_t represent the updated values.

Now a score function can be defined, which can deal with the individual transformation values of every image pair and the global rectification criterium.

7.4 Solving the optimization problem

Now that the optimization problem with its object function is defined, it can be solved in different ways.

7.4.1 Brute-force method

The first approach that did come in mind was to solve the optimization problem with a more or less intelligent brute-force algorithm. The idea of this algorithm is simply to try out every combination of scale and shift values in a defined range, see Figure 7.6 (a).

Runtime: An important factor is the individual step width of each value. This has a huge impact on the runtime of the algorithm. In order to achieve a more precise result, a second pass of the brute-force algorithm can be carried out. The range values can be updated and the step width can be decreased in order to achieve a more accurate result at a better runtime, see Figure 7.6 (b).

7.4.2 Derivative-free methods

These methods only require the objective function values, but no derivative information. This fits very well for the defined optimization problem with its object function. This is because the derivatives can only be estimated and not calculated for the defined function. A review of derivative-free optimization algorithms and a comparison of the software implementations are provided in [23].

The corresponding thesis project uses the *Apache Commons Math* [32] implementation of

- the Powell optimizer [20],
- the BOBYQA optimizer [21] and
- the CMA-ES optimizer [35]

in order to solve the optimization problem. All of them deliver almost identical results and are easy to use, see Chapter 9.

Used methods: The *Powell optimization* is an efficient method for finding the minimum of a function $F(\mathbf{x})$ of several variables $\mathbf{x}$, without having to calculate derivatives of this function. A subsequent method of this optimization technique, which was published in 1964, was the *BOBYQA optimization* published in 2009. This method finds the minimum of a function $F(\mathbf{x})$ with bounds $\mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$ on the variables. This was very useful in order to restrict the transformation $T_{s,t}$ of the individual image lines.

The *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) is a stochastic derivative-free method for numerical optimization of non-linear or non-convex continuous optimization problems, see Figure 7.7. The implementation of this method was also tested and functioned as well.

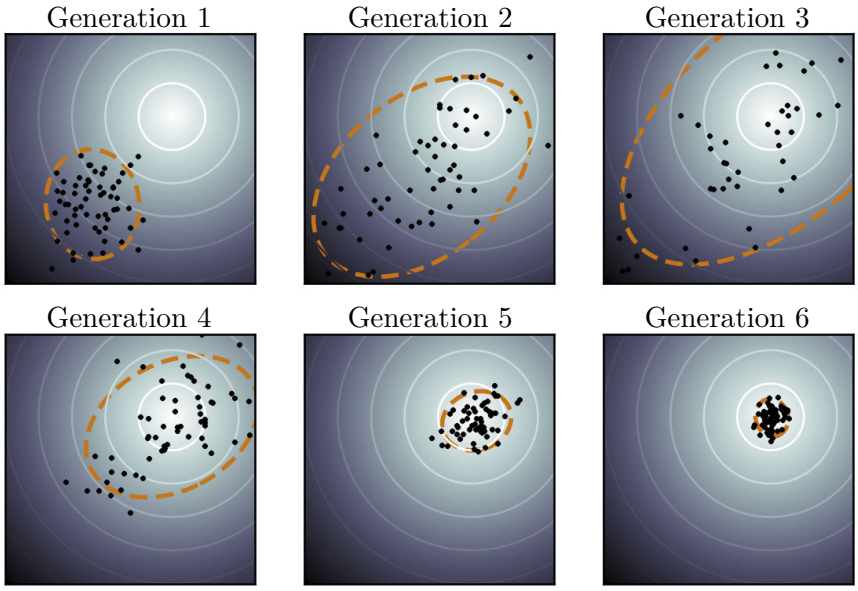


Figure 7.7: CMA-ES optimization example. Given is a simple two-dimensional problem with one global optimum. The amount of samples is the same in every generation. The function is evaluated for every sample position (black dots) of the population. The dotted line shows the distribution of the samples. The population is propagated to the next generation and concentrates over the global optimum within a few generations. Image taken from [35].

Chapter 8

Results and challenges

This chapter summarizes the results and challenges of the incremental image warping algorithm, see Figure 8.1-8.6. Two charts display the transformation values for all image lines for one input image, see Figure 8.2 (e) and Figure 8.3 (e). Analyzing these charts can give additional information about the quality of the warping result. Sharp jumps in the slope of the shift or the scale curve should not occur, because then the warping result will be incorrect.

A step by step incremental image warping example, using only one and two starting points, is shown in Figure 8.5 and the same example, using five starting points, is shown in Figure 8.6. Finally the different challenges and solutions are described in Section 8.1.

8.1 Challenges and solutions

This section summarizes some of the challenges and problems that occurred during the development of the incremental image warping algorithm.

8.1.1 Other image structures

The environment around the street that should be detected can change in numerous ways. A uniform surrounding proposes no problem for the algorithm, but other image structures with sharp edges pointing in another direction than the street do. In Figure 8.7 some examples are shown. This can be handled by weighting the line score function greatly in comparison to the surrounding and therefore setting the evaluation window exactly on the street, see Section 6.2.

8.1.2 Image errors and black lines

Image errors can appear on every input image and are caused by the used cameras. They often occur as black lines or individual pixel errors produced

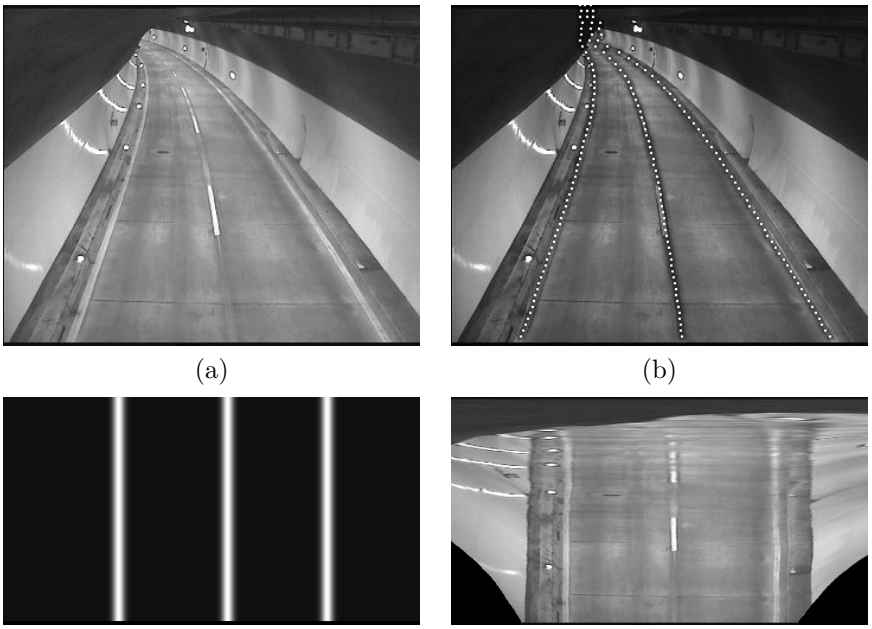


Figure 8.1: Incremental image warping result 1. The original image (a) with the result (b). The reference line image, which has now three lines (c), with the transformed image (d). Note that (c) and (d) are wider. If the position of the middle lane is known in advance, it can improve the matching algorithm. It would be possible to detect the middle in the transformed image as well.

by dirty lenses. These errors must be considered in different ways. At first the black lines in the bottom of the image must be detected and skipped for the line warping algorithm, see Figure 8.8 (a). This simple task can be done in several ways. The individual pixel errors, see Figure 8.8 (b), can be dealt with by blurring the input image, see Section 8.1.3.

8.1.3 Camera image and smoothing

The quality of the input image itself varies greatly due to the different camera models that are being used. The algorithm itself works with image lines with a height of 1 pixel and is therefore very sensitive to noise and image errors. Some cameras already apply some pre-processing steps like sharpening to the output video stream, which can be problematic for fitting dashed lane markers. One solution to this problem is to blur the image. However, applying the same blur strength to the whole image is counterproductive, because this will destroy image information like road markers in the upper part of the input image, which are already very small in size, see Figure 8.9.

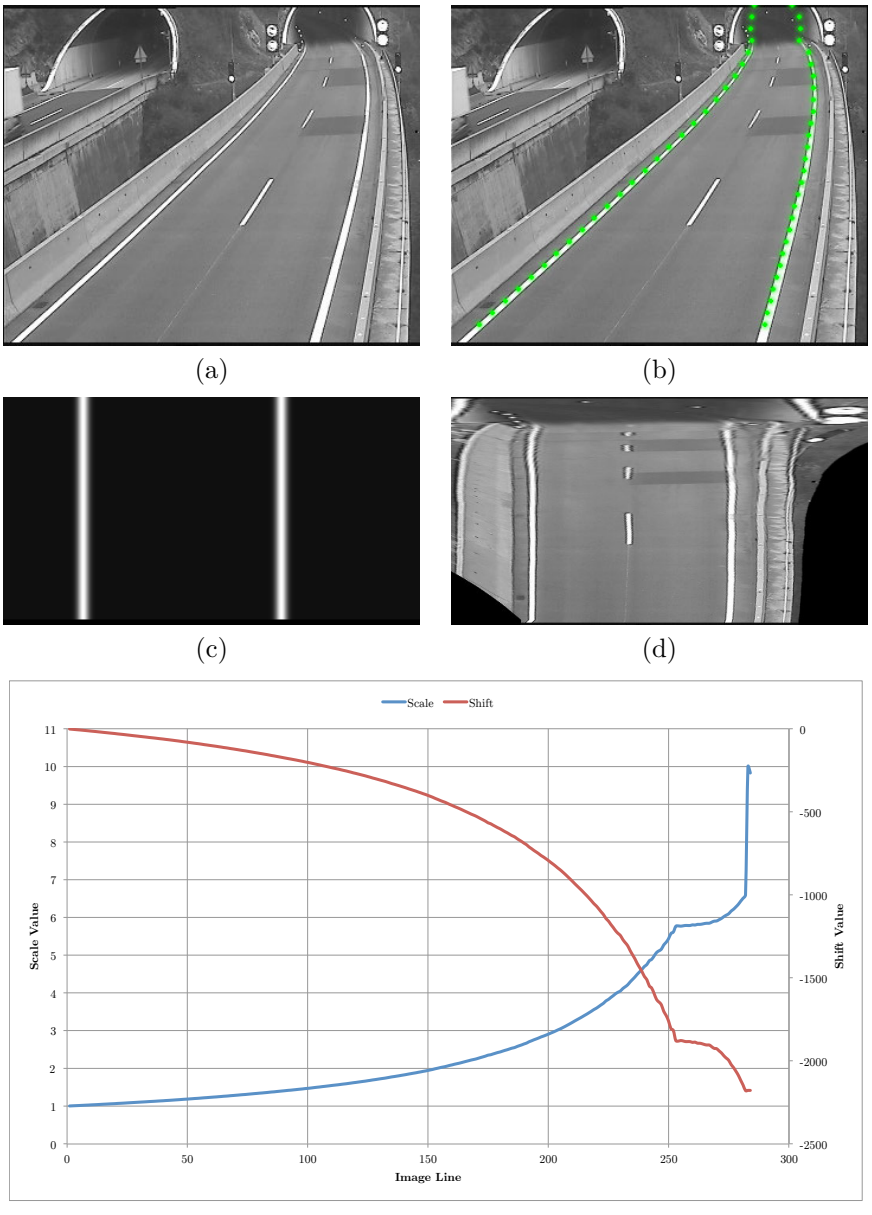
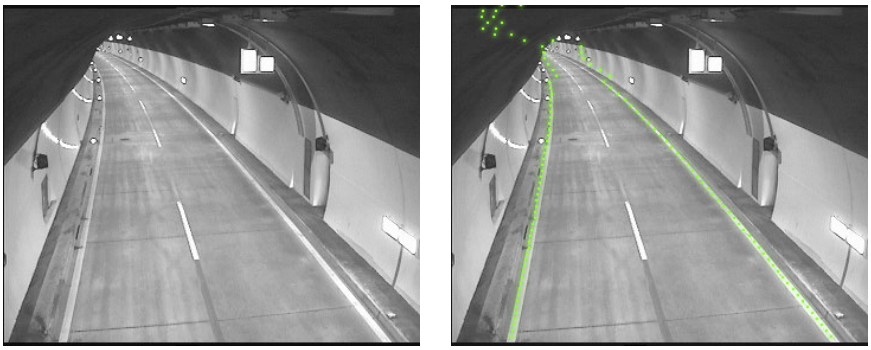
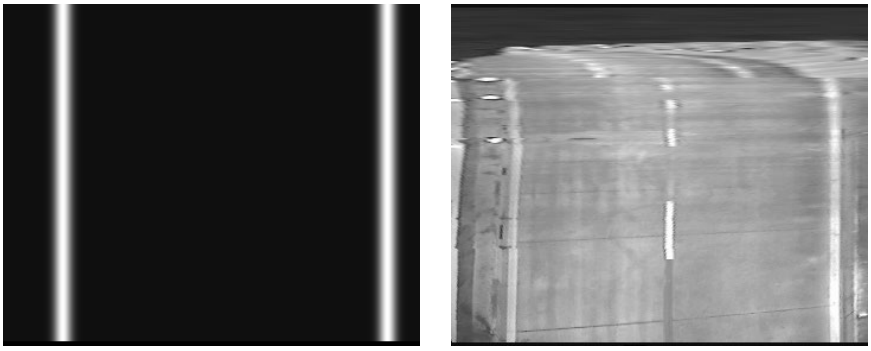


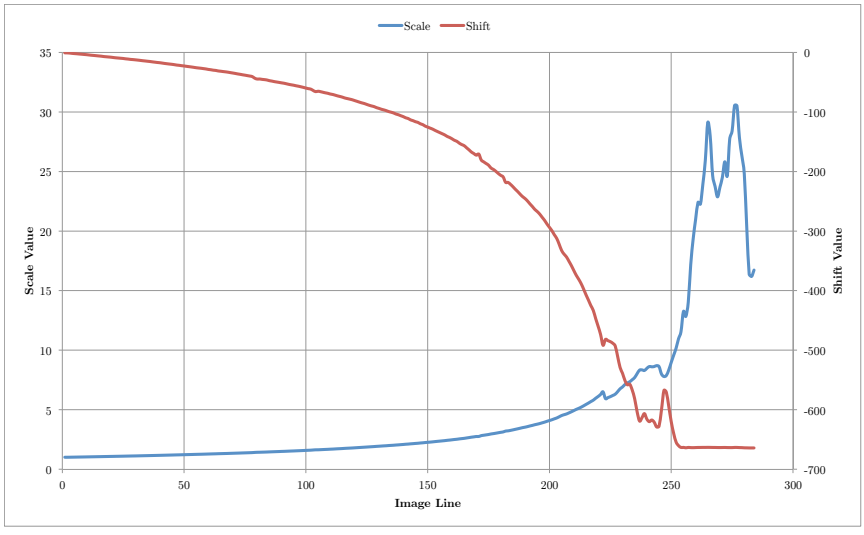
Figure 8.2: Incremental image warping result 2. The original image (a) with the result (b). The weighting line image (c) with the transformed image (d). Note that (c) and (d) are wider. The detection of the middle lane in (d) can be done afterwards. The resulting transformation values are shown in (e). The scale values, starting with $s = 1$, are printed on the primary vertical axis. The shift values, starting with $t = 0$, are printed on the secondary vertical axis. The x-axis shows the image line number. At approximately 252 the street is lost and can not be matched anymore, resulting in the value jump.



(a) (b)

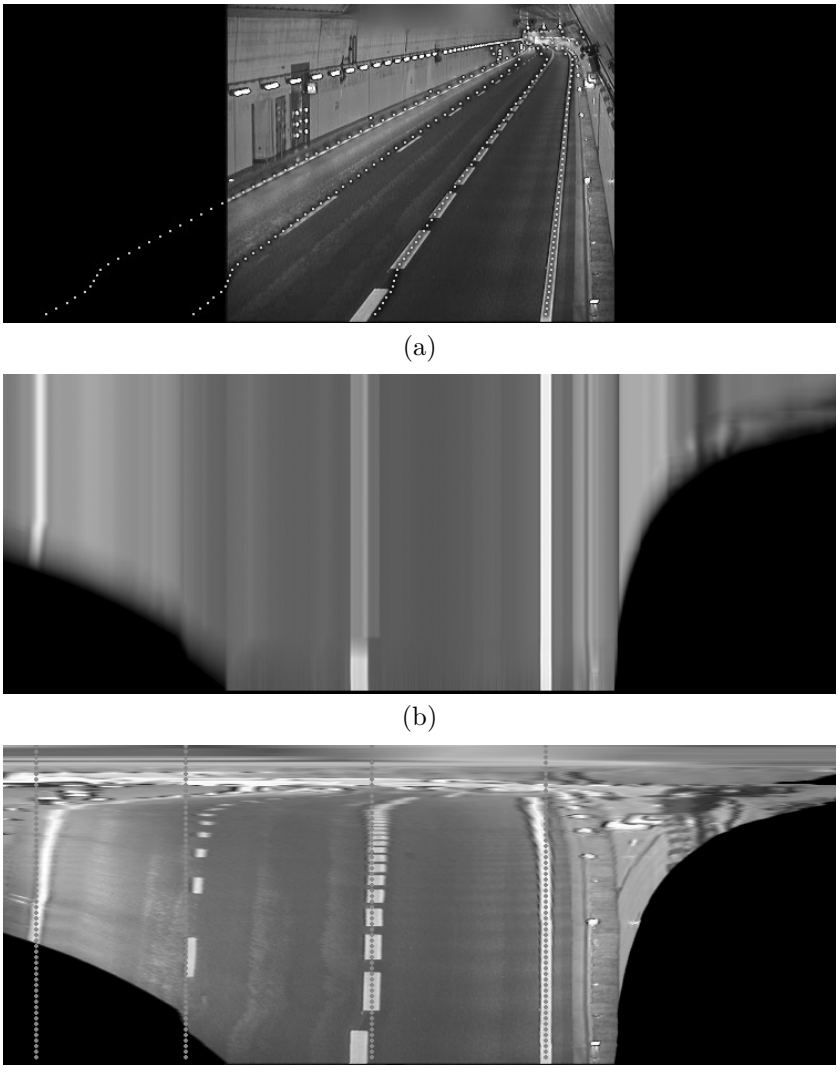


(c) (d)



(e)

Figure 8.3: Incremental image warping result 3. The original image (a) with the result (b). The weighting line image (c) with the transformed image (d). The resulting transformation values are shown in (e). The scale values, starting with $s = 1$, are printed on the primary vertical axis. The shift values, starting with $t = 0$, are printed on the secondary vertical axis. The x-axis shows the image line number. At approximately 240 the street is lost and can not be matched anymore, resulting in the value jump.



(a)

(b)

(c)

Figure 8.4: Weighting line from image. The original image with additional width and the result on top (a). The visualization of every individual weighting line $L_{w,i}(u)$ for every image line i (b). The transformed result (c).

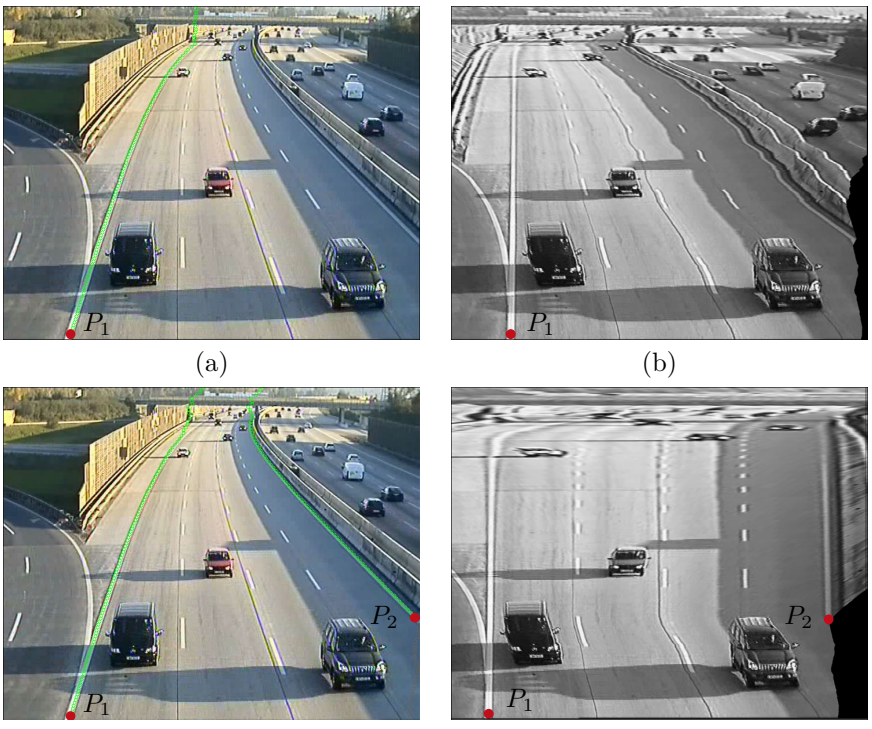


Figure 8.5: Step by step incremental image warping example 1. In (a) only the left street line, with the starting point P_1, was selected and successfully detected. The result shows a rectification of only this line, the other ones were not considered (b). In (c) the right street line, with the starting point P_2, was added to the selection. The street is correctly rectified after the image line of point P_2 (d). In Figure 8.6 the complete rectification is shown.

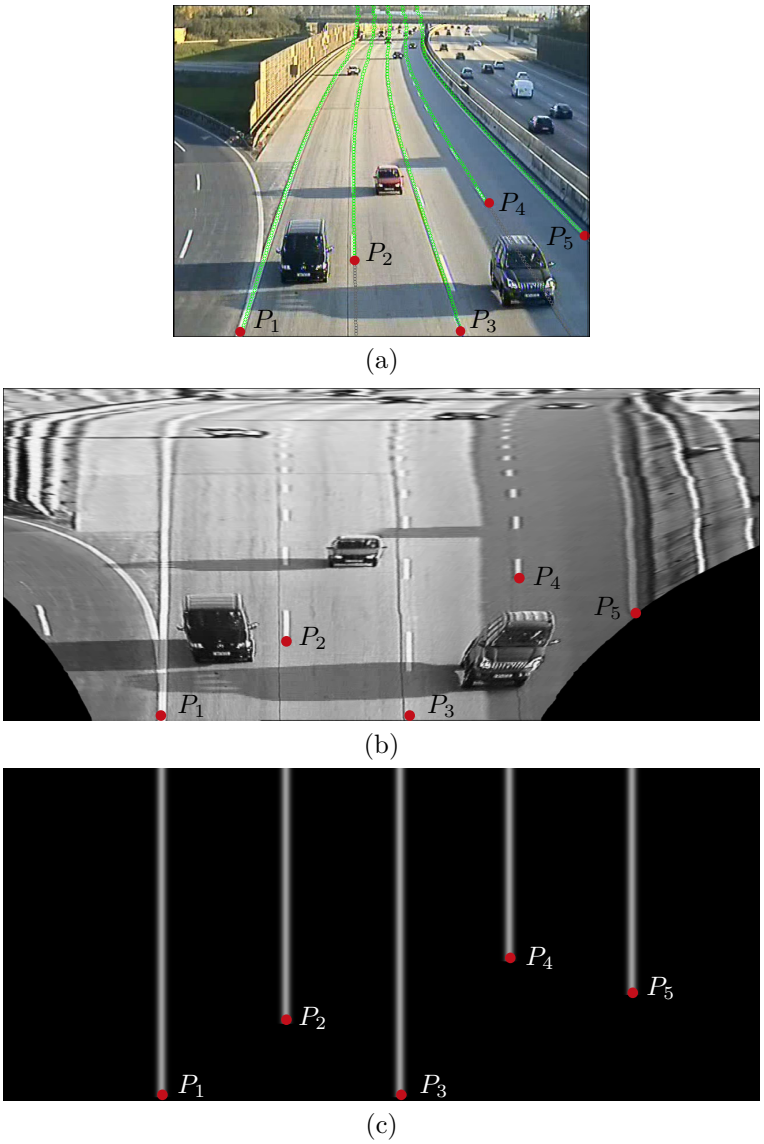
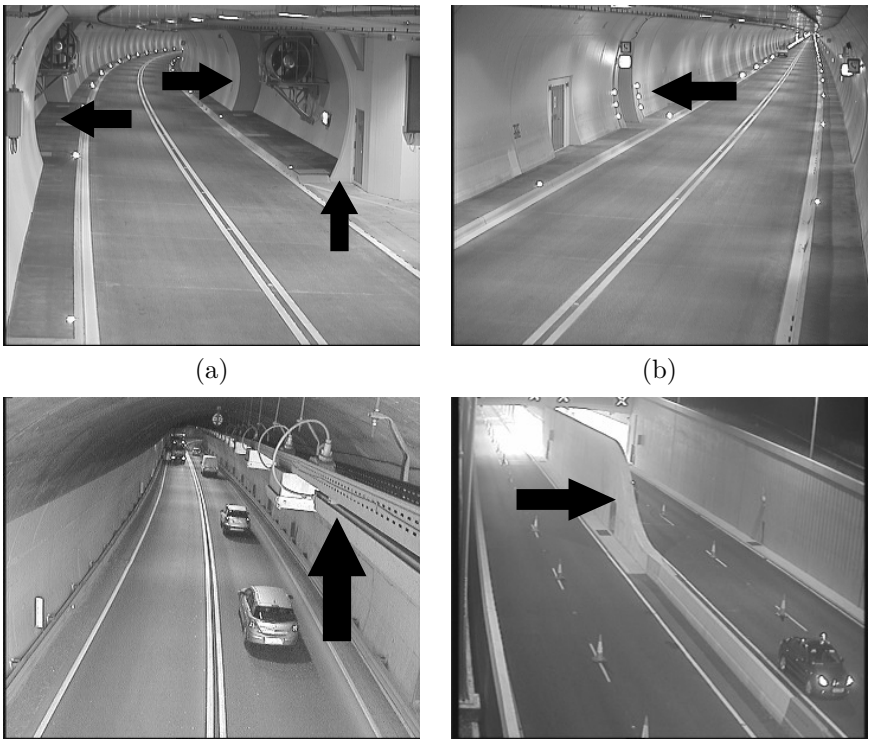


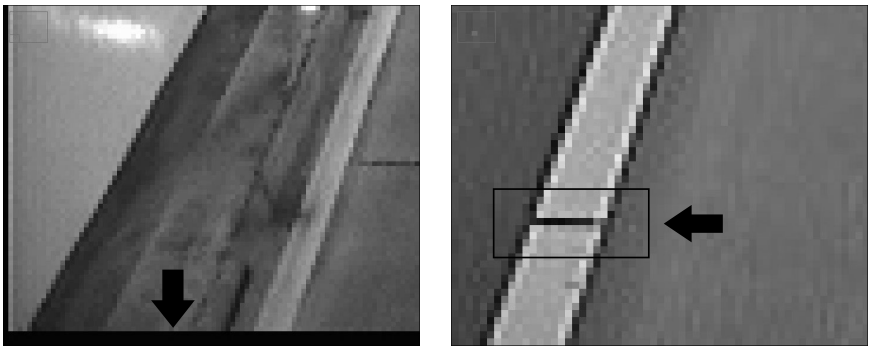
Figure 8.6: Incremental image warping example 2. In (a) the five street lines were selected and successfully detected. The red dots indicate the starting points P_1, P_2, P_3, P_4 and P_5 of the weighting line. The result shows the fully rectified street (b). The associated weighting lines (c). A partly rectification of the same image is shown in Figure 8.5.



(a) (b)

(c) (d)

Figure 8.7: Other image structures. Dents in tunnel walls (a), emergency escape doors (b), ceiling lights (c) and structures not belonging to the road propose a problem for the image warping algorithm.



(a) (b)

Figure 8.8: Image errors. Black lines often appear on the bottom of the image (a). Individual pixel errors can occur throughout the image (b). These errors must be dealt with in order for the line warping algorithm to work.

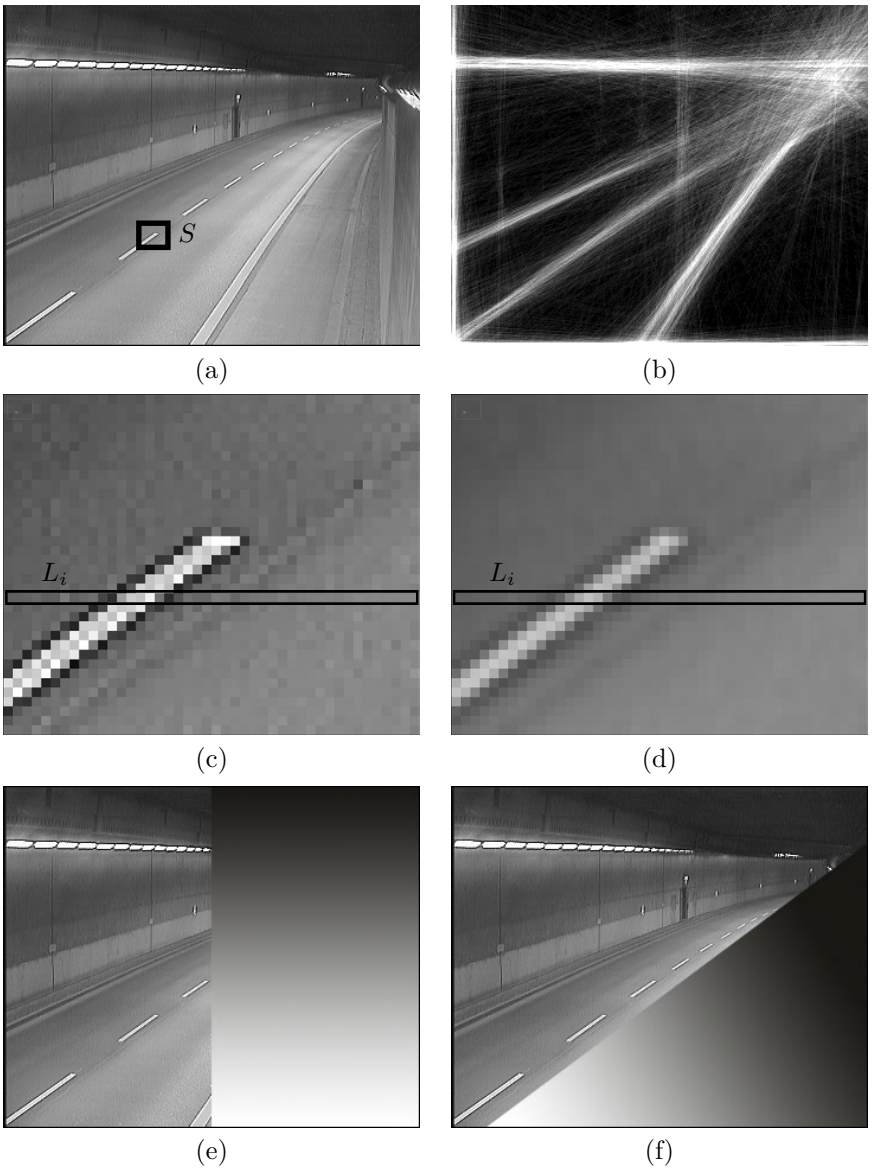


Figure 8.9: Camera image errors. The errors are not clearly noticeable in the original camera image (a). The black square S in (a) indicates the image section in (c) and (d). In the enlarged image (c) the noise and sharpening effects are clearly visible. The sharp steps at the end of one lane marker propose a problem for the algorithm. A Gaussian blur with a radius of $\sigma = 0.7$ reduces this problem greatly (d). In (c) and (d) the black rectangle represents a part of one image line L_i. However, applying the same blurring filter with the same strength over the whole image, destroys image information in the upper part. Therefore the strength of the filter must be diminished, based on the position. This can be done in a very simple way from top to bottom, indicated by the gradient in (e). To improve this non optimal solution, the vanishing point algorithm (Section 3.5) can be used to find the approximate position of the vanishing point (b) and use this to update the angle of the gradient (f).

Chapter 9

Implementation

The project was divided into two parts. At first the development, prototyping and testing was done in Java with ImageJ[1]. The entire thesis project was done with this setup, because the environment is known and prototyping is easy. As usual in ImageJ all the initial algorithms need to be put in the `default` package. In Figure 9.1 the class diagram is illustrated, including all the created and used java packages and some classes.

The second part was at the end of the thesis project. The most useful algorithms were converted into a C++, in combination with OpenCV[2] version. This was then implemented into the existing program structure of the industrial partner. Due to the confidentiality agreement this part is not explained in this thesis.

Implemented algorithms The content of the `default` package is broken down into Table 9.1, which describes the implemented incremental image warping algorithms and into Table 9.2, which describes the other implemented algorithms.

9.1 Bresenham drawing and ellipse voting

Different versions of the Bresenham algorithm [4] were implemented. It is possible to draw

- lines from one point to another,
- ellipses, full or divided into sections and
- circles, full or divided into sections.

Additionally these helper classes can return a list of the individual pixels. This is needed in the *Marker Merge* and the *Ellipse Crawl* algorithm.

[1] ImageJ homepage (http://imagej.nih.gov/ij/).
[2] OpenCV homepage (http://opencv.org/).

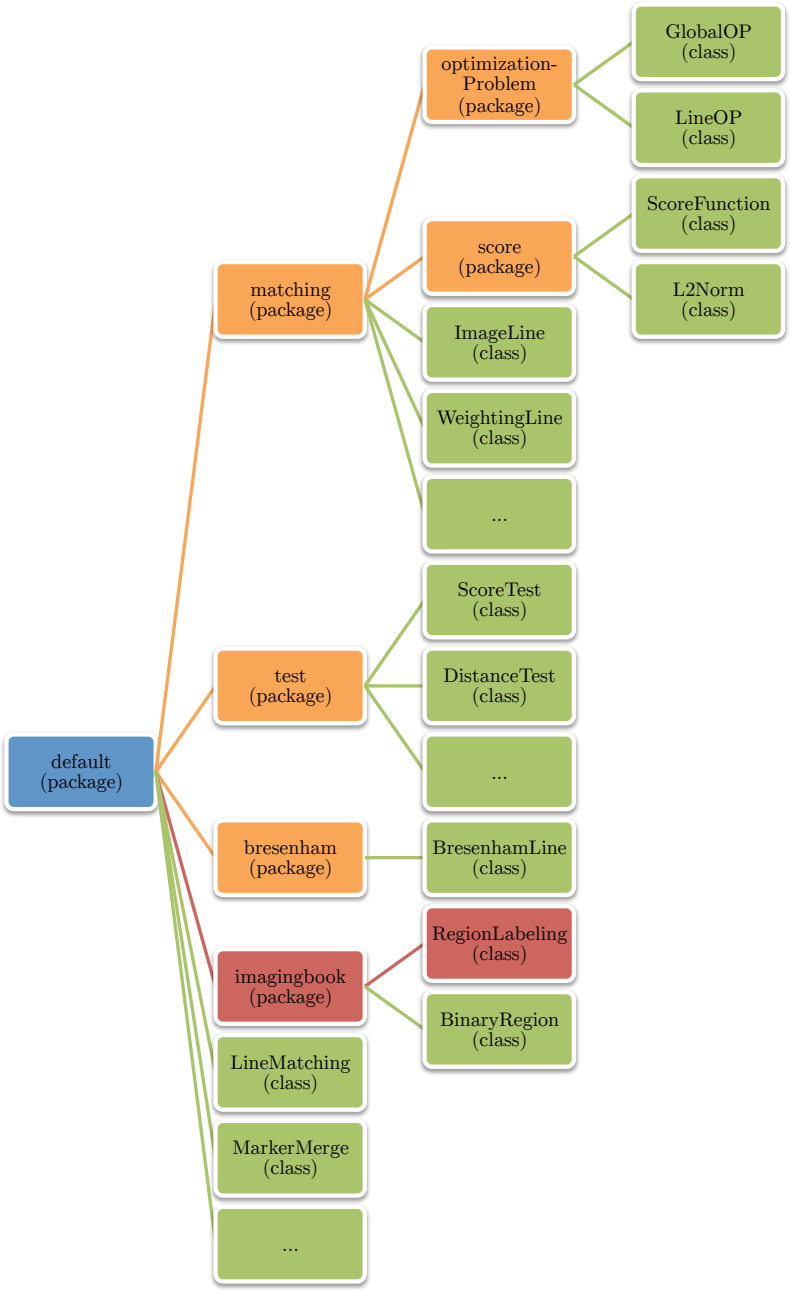


Figure 9.1: Class diagram of the java implementation. The structure is divided into the `default` package, the `matching` package, which is divided into sub-packages, see Section 9.3, the `test` package, see Section 9.5, the `bresenham` package and the external `imagingbook` package [29], see Section 9.2.

Table 9.1: Overview of the implemented incremental image warping algorithms. All of the following ImageJ Plugins perform incremental image warping on a given image, using different approaches.

Java Class	Description
`LineMatching`	This plugin uses the brute-force algorithm, which only uses the L_2 norm as a score function.
`LineMatching_Ref`	This plugin uses the brute-force algorithm, which uses the reference line approach.
`LineMatching_Weight`	This plugin uses the brute-force algorithm, which uses the weighting line approach.
`LineMatching_Powell`	This plugin solves the optimization problem using a Powell Optimizer, which only uses the L_2 norm as a score function.
`LineMatching_Powell_Ref`	This plugin solves the optimization problem using a Powell Optimizer, which uses the reference line approach.
`LineMatching_Powell_Weight`	This plugin solves the optimization problem using a Powell Optimizer, which uses the weighting line approach.

Ellipse crawling algorithm

This algorithm uses one quarter of an ellipse with a specific length to width ratio and tries to find the part, which fits best on a continuous lane marker. The following steps are performed:

Input: Given is the grayscale input image $\mathbf{I}$.

1. On the original picture $\mathbf{I}$ a thresholding step is performed, see Figure 9.2 (b).
2. Based on the given starting point P_S multiple quarters of ellipses $E_1, E_2, \ldots, E_n$ are calculated with different length to width ratios, see Figure 9.2 (c).
3. This ellipse bundle $\mathbf{B} = \langle E_1, E_2, \ldots, E_n \rangle$ is then rotated in order to cover all the lane marker orientations, resulting in $\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_m$.
4. Every point on every ellipse E_i in every ellipse bundle $\mathbf{B}_i$ is checked if a lane marker is hit or not, resulting in one vote score for every ellipse.
5. The ellipse part with the most score is then chosen in order to propagate the starting point P_S, the procedure is repeated starting from 2.

Table 9.2: Overview of the other implemented algorithms.

Java Class	Description
`Draw_Regions`	This plugin draws all the found binary regions into another image.
`Marker_Merge`	This plugin performs the marker merge algorithm, see Section 4.4, starting with a user defined marker.
`Automatic-_Marker_Merge`	This plugin performs the marker merge algorithm, see Section 4.4, starting with an automatically detected start region.
`VP_Detection`	This plugin performs vanishing point detection based on the described algorithm, see Section 3.5.
`Log_ImageLine`	This plugin takes two image lines out of a given picture and logs the results of the score function.
`Ellipse_Vote`	This plugin votes on different rotated Bresenham ellipses, in order to follow a continuous lane marking, see Section 9.1.
`Test_Cases`	In addition to the JUnit tests, see Section 9.5, different test cases, like image line and weighting line test, are implemented as ImageJ plugins.

9.2 Binary regions implementation

The base of the binary region implementation, which includes region labeling, was taken from the *Computer Vision* lecture, which was based on [6]. The calculation of the central moments, the eccentricity and the orientation was added. Additionally a depth first labeling, which has an 8 neighborhood connectivity, was implemented.

9.3 Matching (Java package)

This package includes all the needed classes in order to perform incremental image warping on a given image. It includes the `BruteForceMatcher`, the `ImageLine` class, the `WeightingLine` class, the `ReferenceLine` class and the `MatchingHelper` class. Additionally two other packages, the `optimization-Problem` and the `score` package, are used in order to bundle the remaining classes.

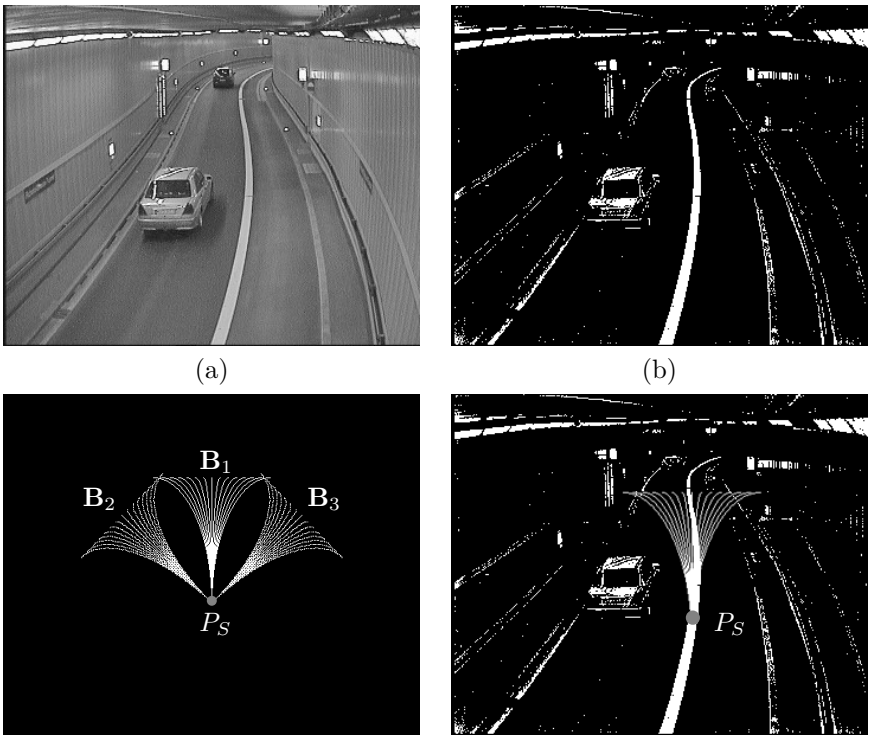


Figure 9.2: Ellipse crawling algorithm. The original image (a) with the result of the thresholding step (b). One quarter of an ellipse is drawn multiple times with different length to width ratios. This ellipse bundle $\mathbf{B}_1$ is then rotated. In (c) this is visualized at two stages $\mathbf{B}_2$ and $\mathbf{B}_3$. Given a starting point P_S the best ellipse part is chosen, which has the most votes on a line marker (d).

`ImageLine` (Java class)

A class was needed that is independent from the original image. Now scaling and shifting can be done without any boundaries. This class stores the following data:

- The original line,
- the current scale value and
- the current shift value.

The line itself can be scaled and shifted multiple times. No calculation is done at this point, only the scale and shift values are updated and stored. Now the whole transformed line, which can be longer or shorter, or only the visible part can be fetched. The latter method additionally saves calculation time, because only the required part is calculated.

The interpolation method is currently only a simple linear interpolation. This is done via a target to source mapping, which calculates for every position x of the transformed line $L'(x)$ a interpolated value at position x' of the original line $L(x')$, see Eqn. 5.3.

`WeightingLine` (Java class)

This class can generate a weighting line, see Section 6.2.1, with various amounts of points. At any point a Gaussian is calculated and accumulated into the weighting line, see Figure 6.5. Additionally the width of the line and the properties of the Gauss function can be set.

`ReferenceLine` (Java class)

This class calculates a weighting line directly from a reference image, see Section 6.2.2. Different summation variants are implemented as well as a function to normalize the resulting line.

`MatchingHelper` (Java class)

This helper class has different methods, which are useable in order to test different parts of the algorithm individually. There are methods for scaling, shifting and multiplying one image line and for getting an interpolated value on any position.

`BruteForceMatcher` (Java class)

The first idea that comes in mind in order to solve the local problem definition, see Section 7.2, is trying out every combination of scale and shift values in a defined range. The result was the brute-force matcher, which has

- the scale range,
- the shift range,
- the scale step width and
- the shift step width

as properties, see Section 7.4.1. For the two transformation values two nested loops are needed, which use these range and step width values.

`TransformationPair` (Java class)

The transformation pair simply holds the shift and scale values in a Java object.

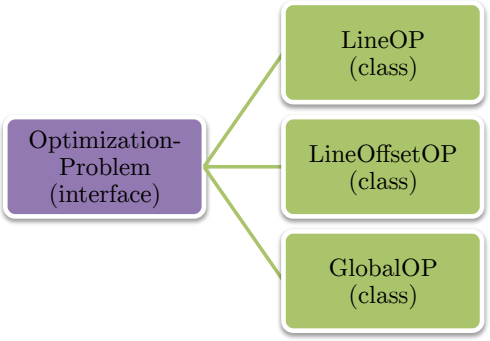


Figure 9.3: Hierarchy of the optimization problem classes.

9.3.1 Optimization Problem (Java package)

The hierarchy of the optimization problem classes is shown in Figure 9.3. All of these classes use the score function package (Section 9.3.2) to calculate the actual score value between two image lines. This architecture was created in order to gain flexibility, because now the actual score functions can be exchanged and are decoupled from the optimization problem.

`OptimizationProblem` (Java interface)

The interface of the optimization problem itself is very simple, because only one function is needed.

```
public interface OptimizationProblem {
    public ObjectiveFunction getObjectFunction();
}
```

The method declaration and the type `ObjectiveFunction` come from the *Apache Commons Math* [32] implementation of the Powell Optimizer [20]. Up to this point the library was not needed. The `BruteForceMatcher` imports this interface and therefore the *Apache Commons Math* library, in order to achieve a consistent call structure, but it could work without it.

This `ObjectiveFunction` can be implemented as an anonymous class, which has again only one method that needs to be defined.

```
public ObjectiveFunction getObjectFunction() {
  return new ObjectiveFunction(new MultivariateFunction() {
      public double value(double[] point) {
          //Score calculation
        return 0;
      }
    });
  }
```

This `value(...)` function is needed for the specific score calculation.

`LineOptimizaionProblem` (Java class)

This class is designed to return a score value for the given scale and shift values. The following steps are needed in order to define a specific optimization problem.

1. The class needs to implement the `OptimizationProblem` interface.

```
public class LineOptimizationProblem implements
    OptimizationProblem {
    //...
}
```

2. A constructor can be defined, which holds all the needed information.

```
public LineOptimizationProblem(ImageLine line1, ImageLine line2,
    ScoreFunction scoreFunction) {
  //Constructor
}
```

   In this case the two image lines and the used score function are passed.

3. As already seen, the `ObjectiveFunction` can be implemented as an anonymous class.

```
public ObjectiveFunction getObjectFunction() {
  return new ObjectiveFunction(new MultivariateFunction() {
    public double value(double[] point) {
      //Score calculation
      return 0;
    }
  });
}
```

4. The input for the `double value(double[] point)` function represents in this specific case the scale and shift values. So a score can be calculated based on this values.

```
public double value(double[] point) {
  line1.setScaleValue(point[0]);
  line1.setShiftValue(point[1]);
  return scoreFunction.getScore(line1, line2);;
}
```

   The use of the `scoreFunction` adds additional flexibility, because it can be exchanged. However, the score value could be calculated directly in this method.

`LineOffsetOP` (Java class)

The line offset optimization problem is very similar to the normal line optimization problem. In this case the deviation from other scale and shift values is used for the score calculation, see Section 7.2.2.

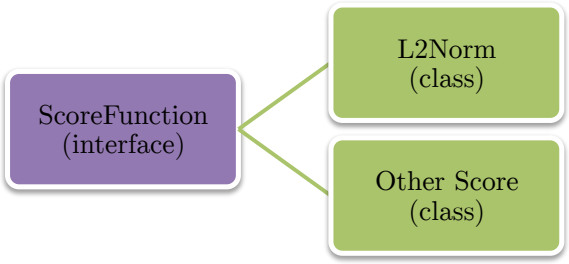


Figure 9.4: Hierarchy of the score function classes.

```
public double value(double[] point) {
    //Score calculation
  return WEIGHT_LINE * scoreValLine +
         WEIGHT_SCALE_OFFSET * scoreValScaleOffset +
         WEIGHT_SHIFT_OFFSET * scoreValShiftOffset;
}
```

So the weighted normal score with the weighted offset of the scale and shift value is returned.

`GlobalOptimizationProblem` (Java class)

The global optimization problem implements the same object function with the same

```
public double value(double[] point) {
    //Score calculation
  return 0;
}
```

`value(...)` method. However, given the whole image with a specific height h, the `double[] point` array has now a length of $l = (h-1) \cdot 2$, see Section 7.3. This of course changes the score calculation inside this method.

9.3.2 Score (Java package)

The hierarchy of the score function classes is shown in Figure 9.4. The implemented classes are used for calculating the actual score value between two image lines.

`ScoreFunction` (Java interface)

The following methods are needed in order to create a specific score function class.

```
public interface ScoreFunction {
```

```
  public void setRefLine(double[] refLine);
  public void setWindow(double[] window);
  public void setNormalize(boolean normalize);
  public double getScore(ImageLine line1, ImageLine line2);
}
```

The only important one, which needs to be implemented, is the

```
public double getScore(ImageLine line1, ImageLine line2);
```

method, which returns the score value given both image lines. Setting the weighting line or the window range can be ignored, if they are not needed as well as the `normalize` boolean.

`L2Norm` (Java class)

This class is designed to calculate a L_2 norm score given two image lines. In the constructor

```
public L2Norm(double stepWidth) {
  //initialization
}
```

the step width s_w can be set. By using $s_w < 1$ oversampling is achieved during the comparison step, see Figure 6.2. Additionally the methods

```
public void setWindow(double[] window) {
  //initialization
}
```

and

```
public void setRefLine(double[] refLine) {
  //initialization
}
```

are implemented in order to set the window range and a weighting line, see Figure 6.4. Now the

```
public double getScore(ImageLine line1, ImageLine line2) {
    double scoreVal = 0;
  //calculation
  return scoreVal;
}
```

method calculates the score value v by

$$v = \sum_{x=a}^{b} [L_1(x) - L_2(x)]^2 \cdot L_w(x) \tag{9.1}$$

given the two image lines L_1 and L_2, the defined window range $[a, b]$ and the weighting line L_w, see Section 6.2.

9.4 Application examples

If a new ImageJ Plugin is created, it only needs classes from the matching package (Section 9.3) and the external apache commons math library [32]. The desired optimization problem and score function can be imported from the corresponding sub packages.

9.4.1 Image warping using the local method

This example shows how all the different classes work together in an abstract way in order to perform incremental image warping, using a local problem definition.

1. All the objects that do not change during the process are defined at the beginning.

   ```
   WeightingLine weightingLine = new ...;
   ScoreFunction scoreFunction = new ...;
   MultivariateOptimizer optimizer = new PowellOptimizer(...);
   ```

   This includes the `MultivariateOptimizer`, which can be a Powell optimizer or another derivative-free optimization method, see Section 7.4.2.

2. Every single image line is matched with the previous one, resulting in the best transformation values.

   ```
   for (...) {
       OptimizationProblem optimizationProblem = new LocalOP(...);
       pointValuePair = optimizer.optimize(
                           new MaxEval(10000),
                           optimizationProblem.getObjectFunction(),
                           GoalType.MINIMIZE,
                           initialGuess);
   }
   ```

   So for every image line pair a new local optimization problem needs to be created, which is then solved by the used optimizer. The `pointValuePair` holds the best scale and shift values, which are the result of the optimization step. The optimizer can be initialized with different parameters like the maximum evaluation steps, the current object function, the goal type and the initial guess, see [32].

3. The current image line is transformed and drawn into the new transformed image.

9.4.2 Image warping using the global method

Creating an ImageJ Plugin, which performs incremental image warping using the global method, the same basic steps as the local method are needed. Since the input for this method is the whole image, the surrounding loop is not needed anymore.

```
ScoreFunction scoreFunction = new ...;
MultivariateOptimizer optimizer = new PowellOptimizer(...);
OptimizationProblem optimizationProblem = new GlobalOP(...);
pointValuePair =  optimizer.optimize(...);
```

The new global optimization problem has now the whole image and the initial guess for every single image line pair as input, see Section 7.3. The optimization step

```
pointValuePair = optimizer.optimize(
                    new MaxEval(10000),
                    optimizationProblem.getObjectFunction(),
                    GoalType.MINIMIZE,
                    initialGuess);
```

does not change at all. Only the initial guess and the resulting point value pair have much more parameters. Given an image with a specific height h and width w and a scale s_i and shift t_i value for every single image line pair, the `initialGuess` and the `pointValuePair` have $(h-1)\cdot 2$ parameters, see Section 7.3.

9.5 Testing

Testing the algorithm is an important factor in the development process. The complexity is very high and many different parts of the algorithm need to work together. Finding an error afterwards, which occurred somewhere in a sub-module is hard. It must be ensured that all the different parts of the program work correctly on their own.

Tests as ImageJ Plugins: Some tests need to be done visually or by an own application with user input and realtime interaction. This can be achieved easily by creating ImageJ Plugins for the individual tests. The possibility of realtime interaction using sliders or buttons is provided by the framework. Some tests and helper classes were created in order to calculate different image parameters or to visually test the scaling and shifting of image lines. One example is the weighting line, which was tested visually, see Figure 9.5 and by a JUnit test case.

Tests with JUnit: When using a testing framework, in this case JUnit[3], many advantages arise. JUnit is designed to write repeatable tests, which should not fail throughout the development process. A high coverage of the code using multiple test cases is very beneficial. By extending and running all of the test cases on a regular basis, errors are found at an early implementation stage. Some of the implemented test cases are shown in Figure 9.6.

[3] JUnit homepage (http://junit.org).

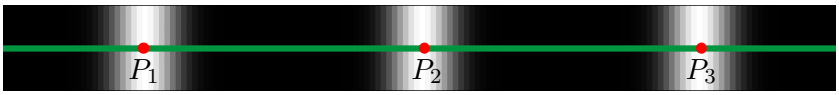


Figure 9.5: Weighting line test. This image was created using a specific weighting line test. Normally the line would have a height of 1 pixel, which is shown by the green line, see Section 6.2.1. For a better visualization several lines are drawn above each other. The red dots mark the position P_1, P_2 and P_3, where the centers of the three individual Gaussian functions $g_1'(x), g_2'(x)$ and $g_3'(x)$ are. The line is created using Eqn. 6.7. Black represents a function value of 1.

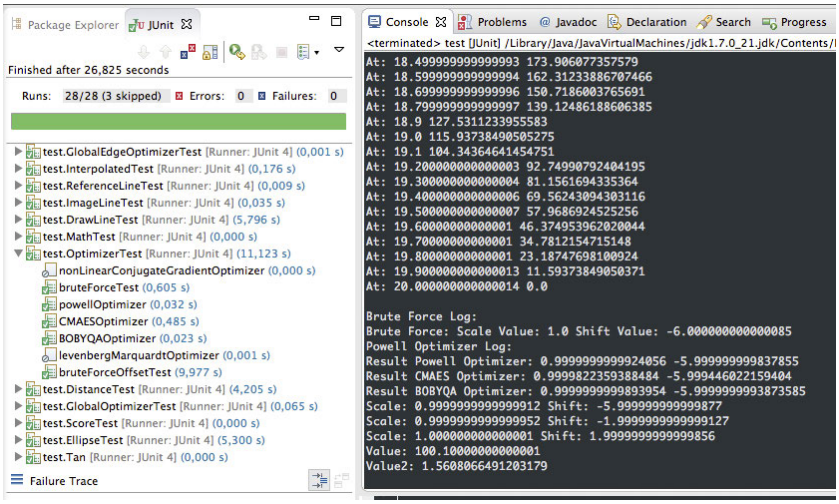


Figure 9.6: Screenshot of a JUnit test case run. The expanded JUnit test case `test.OpitimzerTest` shows some of the tested optimizers with their runtime. In this small scale scenario the brute-force method took 0.6, the CMAES optimizer 0.49, the Powell optimizer 0.032 and the BOBYQA optimizer 0.023 seconds. This of course only applies to this specific test with the used optimization parameters.

Chapter 10

Conclusion and future work

Many challenges come with the problem addressed in this thesis due to the changing setup and input data. Different approaches were tested (Chapter 3) in order to achieve a good segmentation. Some of them can not be used, but others can produce additional image information, which can be used otherwise.

The vanishing point detection algorithm in image space (Section 3.5) is a simple approach, but it is also limited when it comes to a robust detection. For example, given a curved road image it can only provide an indication of the position of the vanishing point. So this approach can work in some cases, but is to sensitive on small image edges due to the nature of the algorithm.

The road marker detection and merging algorithm (Chapter 4) can produce quite good results, but are also very sensitive on image errors and other image content that mimic the shape of a road marker. Nevertheless it can improve other segmentation methods like the watershed algorithm.

The incremental image warping algorithm (Chapter 5-7) can straighten and rectify an image of a curved road without any prior knowledge about the content itself. The results provide a good segmentation with minimal user input. The runtime of the algorithm is great as well due to the used optimizer. This makes this approach operational for the industrial partner of the thesis project. Nevertheless there is room for future work.

Finding the starting point automatically: The starting points could be found automatically, making this approach completely autonomous. This could be done by pre-processing the image and calculating a score, based on the points with their corresponding weighting lane.

Defining a better score function: The L_2 norm score provides good results, but it is possible that a more sophisticated score can enhance the line matching, making it more robust.

Analyzing the chart data: A lot of information could be gained from the transformation chart data in Figure 8.2 (e) and Figure 8.3 (e). The warping result could be analyzed and errors could be fixed.

Global method improvement: The global problem definition could be changed to reduce computation time. Not every single image line must be calculated. The whole image is given as an input for the optimization step, therefore image lines in a certain step width could be matched. This would shorten the runtime. In between a suitable interpolation method can be used in order to gain all the transformation values.

Based on the performed research and analysis of the different approaches, the incremental image warping algorithm seems like a good candidate for road and lane detection for this given task. It currently produces good results and there is the possibility of improvements in the future to make it more robust and reliable.

References

Literature

[1] Aharon Bar Hillel et al. “Recent progress in road and lane detection: a survey”. In: *Machine Vision and Applications* (Feb. 2012) (cit. on pp. 6–8).

[2] S. Beucher and M. Bilodeau. “Road segmentation and obstacle detection by a fast watershed transformation”. In: *Proceedings of the Intelligent Vehicles ’94 Symposium*. Paris, France, Oct. 1994, pp. 296–301 (cit. on p. 17).

[3] S. Beucher and F. Meyer. “The morphological approach to segmentation: the watershed transformation. Mathematical morphology in image processing.” In: *Optical Engineering* 34 (1993), pp. 433–481 (cit. on p. 15).

[4] J. E. Bresenham. “Algorithm for Computer Control of a Digital Plotter”. In: *IBM Systems Journal* 4.1 (Mar. 1965), pp. 25–30 (cit. on p. 73).

[5] Wilhelm Burger and Mark J. Burge. *Principles of Digital Image Processing – Advanced Methods (Vol. 3)*. Undergraduate Topics in Computer Science. London: Springer Publishing Company, Incorporated, 2013. URL: www.imagingbook.com (cit. on pp. 28, 29).

[6] Wilhelm Burger and Mark J. Burge. *Principles of Digital Image Processing – Core Algorithms (Vol. 2)*. Undergraduate Topics in Computer Science. London: Springer Publishing Company, Incorporated, 2009. URL: www.imagingbook.com (cit. on p. 76).

[7] Wilhelm Burger and Mark J. Burge. *Principles of Digital Image Processing – Fundamental Techniques (Vol. 1)*. Undergraduate Topics in Computer Science. London: Springer Publishing Company, Incorporated, 2009. URL: www.imagingbook.com (cit. on p. 23).

[8] John Canny. “A Computational Approach to Edge Detection”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* PAMI-8.6 (Nov. 1986), pp. 679–698 (cit. on p. 17).

[9] Virginio Cantoni et al. “Vanishing point detection: representation analysis and new approaches”. In: *11th International Conference on Image Analysis and Processing, 2001.* Palermo, Italy, Sept. 2001, pp. 90–94 (cit. on pp. 21, 24).

[10] Hsu-Yung Cheng et al. “Lane Detection With Moving Vehicles in the Traffic Scenes”. In: *IEEE Transactions on Intelligent Transportation Systems* 7.4 (Dec. 2006), pp. 571–582 (cit. on p. 6).

[11] Kuo-Yu Chiu and Sheng-Fuu Lin. “Lane detection using color-based segmentation”. In: *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*. Vienna, Austria, Sept. 2005, pp. 706–711 (cit. on p. 14).

[12] Radu Danescu and Sergiu Nedevschi. “Probabilistic Lane Tracking in Difficult Road Scenarios Using Stereovision”. In: *IEEE Transactions on Intelligent Transportation Systems* 10.2 (June 2009), pp. 272–282 (cit. on p. 8).

[13] Richard O. Duda and Peter E. Hart. “Use of the Hough Transformation to Detect Lines and Curves in Pictures”. In: *Communications of the ACM* 15.1 (Jan. 1972), pp. 11–15 (cit. on p. 17).

[14] Martin A. Fischler and Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Communications of the ACM* 24.6 (June 1981), pp. 381–395 (cit. on p. 9).

[15] Hui Kong, Jean-Yves Audibert, and Jean Ponce. “General Road Detection From a Single Image”. In: *IEEE Transactions on Image Processing* 19.8 (Aug. 2010), pp. 2211–20 (cit. on p. 7).

[16] Guoliang Liu, Florentin Worgotter, and Irene Markelic. “Combining Statistical Hough Transform and Particle Filter for robust Lane Detection and Tracking”. In: *Intelligent Vehicles Symposium (IV), 2010 IEEE*. San Diego, CA, USA, June 2010, pp. 993–997 (cit. on pp. 10, 11).

[17] A. Lopez et al. “Detection of lane markings based on ridgeness and RANSAC”. In: *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*. Vienna, Austria, Sept. 2005, pp. 433–738 (cit. on pp. 8, 9).

[18] J.C. McCall and M.M. Trivedi. “Video-Based Lane Estimation and Tracking for Driver Assistance: Survey, System, and Evaluation”. In: *IEEE Transactions on Intelligent Transportation Systems* 7.1 (Mar. 2006), pp. 20–37 (cit. on pp. 6, 8).

[19] Wayne Niblack. *An Introduction to Digital Image Processing*. Birkeroed, Denmark: Strandberg Publishing Company, 1985 (cit. on pp. 9, 28).

[20] M.J.D. Powell. “An efficient method for finding the minimum of a function of several variables without calculating derivatives”. In: *The Computer Journal* 7 (1964), pp. 155–162 (cit. on pp. 62, 79).

[21] M.J.D. Powell. “The BOBYQA algorithm for bound constrained optimization without derivatives”. In: *University of Cambridge NA Report DAMTP 2009/NA06* (Aug. 2009) (cit. on p. 62).

[22] Christopher Rasmussen. “Grouping dominant orientations for ill-structured road following”. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 1. Washington, DC, USA, June 2004, pp. 470–477 (cit. on p. 15).

[23] Luis Miguel Rios and Nikolaos V. Sahinidis. “Derivative-free optimization: a review of algorithms and comparison of software implementations”. In: *Journal of Global Optimization* 56.3 (July 2012), pp. 1247–1293 (cit. on p. 62).

[24] Christopher Urmson et al. “Autonomous driving in urban environments: Boss and the urban challenge”. In: *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I* 25.8 (2008), pp. 425–466 (cit. on p. 7).

[25] Huan Wang and SL Shao. “Lane Markers Detection based on Consecutive Threshold Segmentation”. In: *Journal of Information and Computing Science* 6.3 (2011), pp. 207–212 (cit. on p. 8).

[26] Yue Wang, Eam Khwang Teoh, and Dinggang Shen. “Lane detection and tracking using B-Snake”. In: *Image and Vision Computing* 22.4 (Apr. 2004), pp. 269–280 (cit. on pp. 9, 20).

[27] Jinyou Zhang and Hans-Hellmut Nagel. “Texture-based segmentation of road images”. In: *Proceedings of the Intelligent Vehicles ’94 Symposium*. Paris, France, Oct. 1994, pp. 260–265 (cit. on p. 15).

[28] Shengyan Zhou et al. “A novel lane detection based on geometrical model and Gabor filter”. In: *Intelligent Vehicles Symposium (IV), 2010 IEEE*. San Diego, CA, USA, June 2010, pp. 59–64 (cit. on p. 9).

Online sources

[29] Wilhelm Burger. 2014. URL: http://imagingbook.com/ (cit. on pp. 28, 74).

[30] DARPA. 2014. URL: http://archive.darpa.mil/grandchallenge05/index.html (cit. on p. 7).

[31] DARPA. 2014. URL: http://archive.darpa.mil/grandchallenge/index.html (cit. on p. 7).

[32] Commons Math Developers. *Apache Commons Math, Release 3.2*. The Apache Software Foundation. 2014. URL: http://commons.apache.org/proper/commons-math/ (cit. on pp. 62, 79, 83).

[33] wallsave. 2014. URL: http://www.wallsave.com/wallpaper/4200x2800/roads-pictures-road-4781613.html (cit. on p. 20).

[34] Wikipedia. 2014. URL: http://en.wikipedia.org/wiki/Mathematical_morphology (cit. on pp. 29, 30).

[35] Wikipedia. 2014. URL: http://en.wikipedia.org/wiki/CMA-ES (cit. on pp. 62, 63).