CHALMERS
UNIVERSITY OF TECHNOLOGY

Automatic View Compensation

Master's thesis in Communication Engineering

RICKARD DAHL

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018:NN

Automatic View Compensation

RICKARD DAHL

Department of Electrical Engineering
Division of Signal processing and Biomedical engineering
Computer vision and medical image analysis research group
Chalmers University of Technology
Gothenburg, Sweden 2018

Automatic View Compensation
RICKARD DAHL

Supervisor: Carl Toft, Department of Electrical Engineering
Examiner: Carl Olsson, Department of Electrical Engineering

Master's Thesis 2018:NN
Department of Electrical Engineering
Division of Signal processing and Biomedical engineering
Computer vision and medical image analysis research group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Automatic View Compensation
RICKARD DAHL
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Controlling the luminance in the tunnel threshold zone is a challenging task. On one hand the luminance in the threshold zone needs to be high enough for a driver experiencing significant glare to be able to detect a possible obstacle located in the relatively dark threshold zone and stop in time to avoid an accident. On the other hand there are demands to keep the luminance as low as possible to reduce the energy consumption.

In order to control the luminance in the threshold zone, the glare experienced by the driver needs to be estimated. This is done by measuring the equivalent veiling luminance L_{seq}. Ideally L_{seq} would be measured at the stopping distance, $1.5m$ above the road surface and in the middle of the road with the camera centered on the tunnel opening. In practice however, the camera often needs to be mounted on a height of $5 - 7m$ to avoid staining and vandalism and it sometimes also needs to be placed on the side of the road. This camera positioning leads to a significant L_{seq} error.

In this thesis a view transformation method aimed at automatically reducing the L_{seq} error is presented. The method is based on Depth Image Based Rendering, which requires a camera pose for a virtual view as well as the depth map for the original view. Depth maps of a scene are rarely available however and for this reason a monocular depth map estimation method is proposed. The depth estimation method is based on detecting Maximally Stable Extremal Regions and classifying the regions according to detected line segments.

The proposed view transformation method has been evaluated on a small number of scenes and the error reduction is as high as $40 - 85\%$. There are however several issues that require future improvements, including: a long computation time, high occurrence of errors in the estimated depth maps and errors in the centering of the diagram used for the L_{seq} computation. Nevertheless, the proposed approach can with a few minor adjustments be implemented in the luminance measurement system and be used in practice.

Keywords: Luminance, threshold zone, glare, L_{seq}, view transformation, Depth Image Based Rendering, depth map estimation.

Acknowledgements

To begin with, I am thankful for all the help and support I've received from my examiner Carl Olsson and from my supervisor Carl Toft.
I would also like to thank the employees at *Cipherstone Technologies AB* for helping in making this thesis a reality and for contributing to a friendly and welcoming atmosphere at the workplace. First I would like to thank Kenneth Jonsson, the CEO of *Cipherstone Technologies AB*, without whom the thesis work would not be possible. Kenneth has not only been very helpful in helping me to understand the project and how it relates to the product/system as a whole but has also been kind and helpful in many other ways. I also owe my deepest gratitude to Suryanarayana Murthy Muddala for providing invaluable expertise in computer vision, for giving me important advice with regards to how to prioritize the different parts of the thesis work and for helping me to stay on track. Furthermore I would like to thank David Tingdahl for sharing his expertise on BRDFs (Bidirectional Reflectance Distribution Functions). Last but not least, I want to thank Chanadrashekhar Nasurade, the embedded software programmer at *Cipherstone Technologies AB*. Although his field of expertise is not computer vision, he has contributed greatly by helping me with the measurement gathering part of the work and has also contributed to a positive atmosphere in the office.

Rickard Dahl, Gothenburg, June 2018

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background

Intelligent Transport Systems (ITS) can be defined as the use of various technologies such as wireless communication and computer vision/image analysis to among other things monitor traffic, improve traffic safety and reduce congestion [1]. Applications include monitoring the number of vehicles on a road and varying of the speed limits depending on the traffic conditions or weather conditions. Another important application of ITS is automatic control of tunnel lighting.

The most recent recommendations for tunnel lighting were presented by the International Commission on Illumination (Commission internationale de l'éclairage (or CIE for short)) in the CIE 88:2004 report [2]. According to the recommendations, a tunnel can be divided into several regions as shown in Figure 1.1. The threshold zone is an especially complicated region for lighting control. On one hand the driver might be perceiving significant glare due to sunlight (thus reducing visibility) and on the other hand the driver needs to be able to discern a possible object (such as another vehicle) located in the threshold zone and stop in time to avoid an accident. Furthermore, it takes some time for the eyes to adjust when going from a bright environment to a dark one which means that the light level transition cannot be too abrupt. Finally, there is also a desire to keep the light level as low as possible to reduce energy consumption and thus reduce costs of the lighting and the impact on the environment. For these reasons it is important to automatically control the luminance in the threshold zone L_{th}.

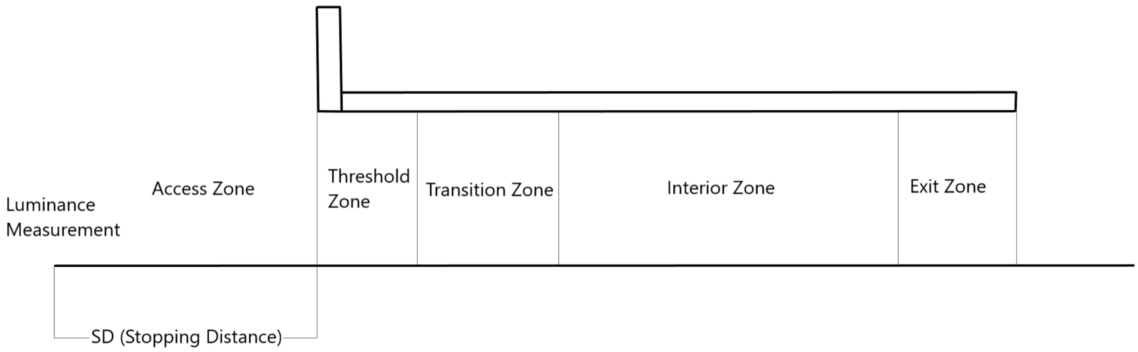


Figure 1.1: Overview of different tunnel regions. The threshold region is at located at the start of the tunnel. The luminance L_{th} in the threshold zone is controlled based on the equivalent veiling luminance L_{seq} measured at the stopping distance from the tunnel.

According to the recommendations in [2], the Equivalent veiling luminance L_{seq} (or alternatively the L_{20} value) needs to be measured in order to determine the threshold luminance L_{th}. L_{seq} should be measured at the stopping distance (the distance that is required for the vehicle to stop, defined as the sum of the reaction distance and the braking distance) corresponding to the speed limit before the tunnel. The camera should be placed around 1.5 meters above the road (corresponding to the average driver position) and in the middle lane of the same side of the road as the tunnel opening. The exact details for how L_{seq} is then computed are given in Section 3.7.1.

In practice however, this camera positioning is difficult (if not impossible) to achieve. To avoid staining and vandalism, the luminance measurement device (for instance a camera) needs to be mounted at a significant height above the road (usually 5-7 meters). In some cases the device (camera) can not be mounted above the middle of the road and instead needs to be placed on the side of the road. In order for the device (camera) to then be centered on the tunnel opening, the device (camera) needs to be rotated, thus also introducing an angle error. To reduce costs the current infrastructure (traffic poles) often needs to be reused. As such the real distance to the tunnel might differ from the stopping distance. This difference can however be easily compensated for (as long as it is small enough).

To investigate the error sources and practical considerations for this kind of camera installation for L_{seq} (or L_{20}) measurement, a research project [3] was conducted by Cipherstone Technologies AB in cooperation with SP Technical Research Insitutute of Sweden and the University of Gothenburg. The L_{seq} measurements were carried out from April 2015 to the end of that year on the Gnistäng Tunnel site in Gothenburg (the location is shown in [4]). The camera was placed on a pole approximately 68 meters from the tunnel opening (which is close to the stopping distance), near the right side of the road and approximately 7 meters above the road. The camera was then centered on the tunnel opening as shown in Figure 1.2.

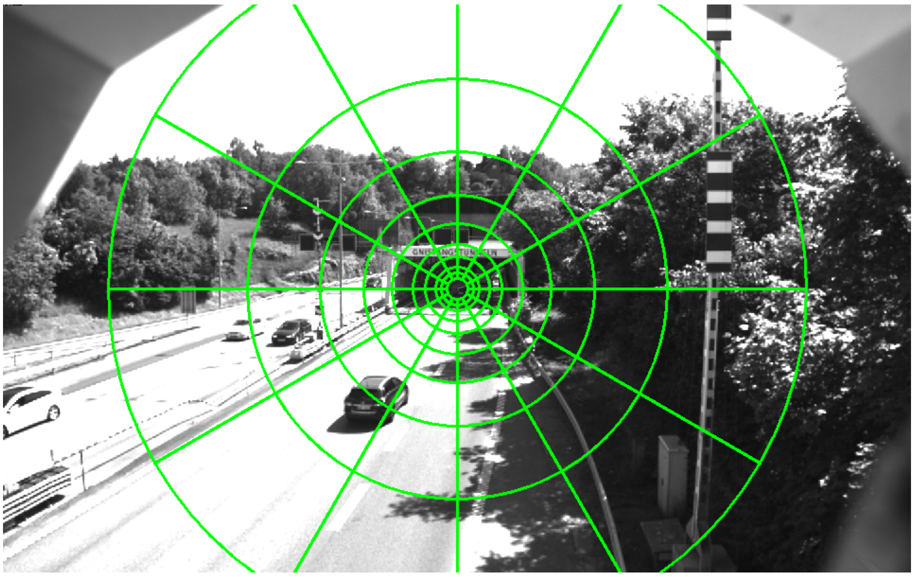


Figure 1.2: Camera view used when computing L_{seq} at the Gnistäng Tunnel scene. The camera is centered on the tunnel opening. Note that the L_{seq} pattern is shown here. The L_{seq} computation is described in detail in Section 3.7.1.

Some of the key findings of the research project [3] include:

- The impact of traffic on the luminance estimation is significant [3]. In the research project a comparison was made between luminance measurements for images with traffic and corresponding images without traffic. The traffic was filtered out from the original images using traffic compensation. It was shown that usually the relative error is around 8-10 percent but can at high traffic levels increase to even 70 percent or more. Therefore, Cipherstone Technologies AB currently filter out the traffic from the images before measuring the luminance.
- In [3] it was noted that in order to center the camera on the tunnel from the mounting position (5 to 7 meters above the road surface), the camera rotation angle relative to the road surface (x-axis rotation or pitch) becomes 4°-10° [3]. It was then shown that in the angle range of interest the luminance value for asphalt has a strong angle-dependency and furthermore it was shown that the luminance value also varies depending on the type of asphalt and depending on whether the asphalt is dry or wet. This phenomenon is best described by Bidirectional Reflection Distribution Functions (BRDFs). BRDFs describe how much light is reflected in a given direction $\mathbf{v}_r$ for a surface of a particular material, based on the light level at an incoming direction $\mathbf{v}_i$ [5] [6].
- As shown in Figure 1.3 a failure to center the L_{seq} region on the tunnel opening can lead to a significant luminance error [3].
- Finally, as shown in Figure 1.4, there is a significant difference between the parts of the scene that can be seen from the practical camera position (5 to 7 meters above the road and sometimes also not centered to be above the middle lane) and the ideal camera position (1.5 meters above the road, centered on the middle lane) [3]. Because of this difference there is an error introduced in the L_{seq} measurement.

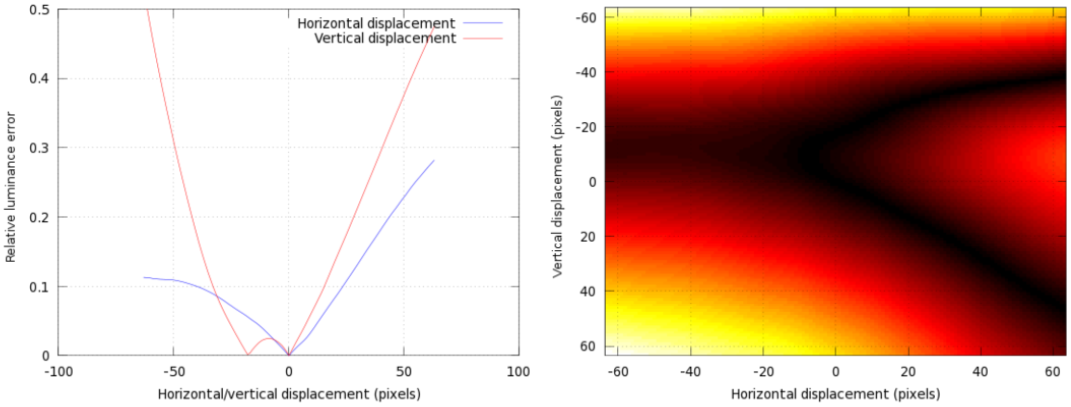


Figure 1.3: Relative luminance error as a function of horizontal or vertical displacement in pixels. The figure is taken from [3]. Permission to use this and other figures from the research project was granted by Kenneth Jonsson (one of the authors).

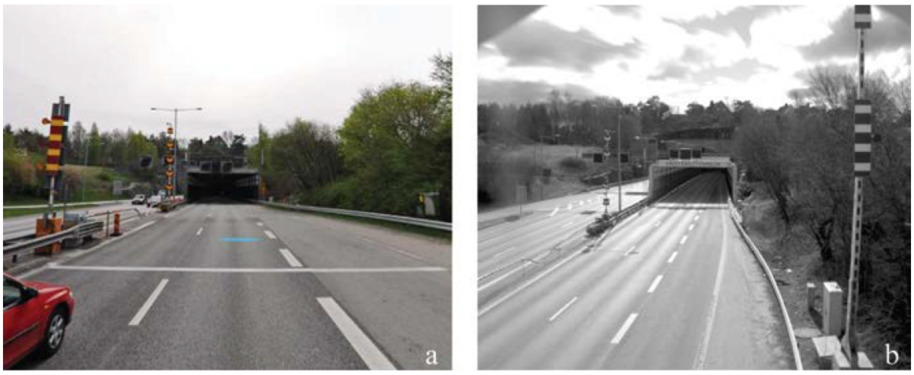


Figure 1.4: a) shows the street view (2m above the road centered in the middle of the right lane) and b) shows the view on the right side of the road and approximately 7m above the road. The figure is taken from [3].

After the research project, an experiment was conducted at *Cipherstone Technologies AB* to measure the error between L_{seq} at the camera position and the L_{seq} at the the ideal L_{seq} measurement position. As shown in figures 1.5 and 1.6 the relative error between L_{seq} measured at the practical position and L_{seq} measured at the ideal position was around 43.2% (note however that the photos were taken approximately 30 minutes apart).

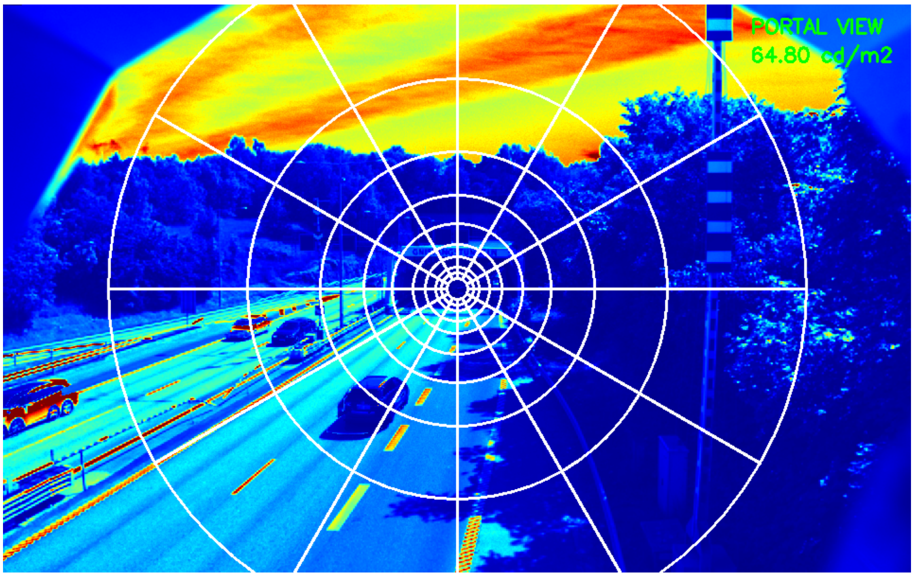


Figure 1.5: L_{seq} measured from the camera mounting position. The image has been color-coded using the jet colormap. Blue corresponds to low luminance values and red to corresponds to high luminance values.

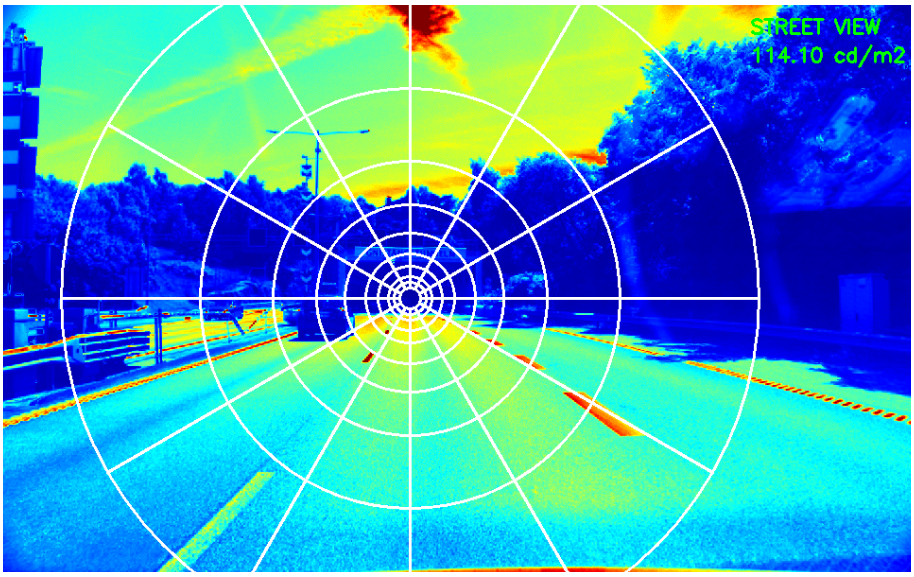


Figure 1.6: L_{seq} measured from the ideal position (driver's view). This figure uses the same color coding as Figure 1.5.

A further investigation was then done at *Cipherstone Technologies AB* to see if the L_{seq} error can be reduced by warping the original view down to a view similar to the driver's view. This was done by approximating the scene by a small number of planes and then for each plane applying a homography to warp that plane (refer to Section 3.1.2 for a theoretical background of homographies). This type of warped image is shown in 1.7. As can noted from the figure, the relative luminance error was reduced to 25.5%. It can also be noted that the image contains major artifacts (distortions).

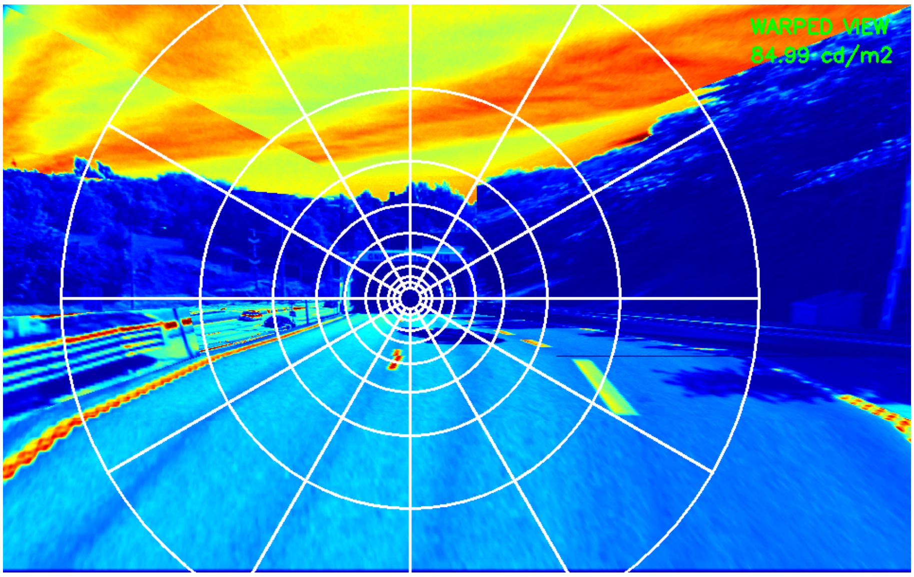


Figure 1.7: L_{seq} in the warped view. The image at the camera mounting position has been warped by approximating the image by a small number of planes and applying homographies to warp the planes. This figure uses the same color coding as Figure 1.5.

The work presented in this report was done at *Cipherstone Technologies AB* and builds upon the research project [3] and the subsequent experiments carried out by *Cipherstone Technologies AB*.

1.2 Purpose

The purpose of the report is to describe a method developed for automatically warping images captured from a camera view above the road, down to the ideal position on the street level and to evaluate the L_{seq} error reductions for this method. The method is based on Depth Image Based Rendering (DIBR), which is a warping method that requires a depth map. To this end, the report also presents a somewhat novel (albeit not yet fully developed) method of depth estimation for a tunnel or road scene from a single camera view. The depth estimation method is based on finding potentially useful image regions and classifying them according to detected line segments within these regions. In order to evaluate the quality of the proposed depth map estimation method, a comparison is made with manually created reference depth maps and the effects of the depth maps on the warped view and the value of L_{seq} are studied.

1.3 Limitations

Originally one of the goals of the project was also to study the Bidirectional Reflection Distribution Function (BRDF) for asphalt. The study of this problem would include gathering measurement data at sites with significant proportions of asphalt and to estimate the luminance for asphalt as a function of height or angle relative to road surface. This would also include investigating how to handle dry road surfaces and how to handle wet road surfaces. However, due to time constraints this part of the project did not get completed.

1.4 Overview

The rest of the report is divided as follows: Chapter 2 Related Works gives an overview of related work in various relevant subjects, including projective geometry, 3D image warping, depth map estimation and segmentation. For more in-depth information refer to the sources cited. Chapter 3 Theory describes the general theory behind the different techniques/algorithms used in the implementation of the work. Among other things it includes a short introduction to some of the topics in projective geometry, a summary of various segmentation methods that have been used and a description of how the Equivalent veiling luminance ($Lseq$) is computed. For a detailed description of the how these methods are used in practice for the implementation refer to Chapter 4 Implementation. This chapter describes the whole pipeline, from input images to output $Lseq$ value for the warped view. Chapter 5 Data Gathering describes the hardware, software and methods used for data gathering. The data gathered comes in two forms: Photos of different scenes and various distance and height measurements in that scene. The results are presented and analyzed in Chapter 6. Other than a comparison of $Lseq$ for different views (view above road, road level view and warped view) there is also a comparison between the proposed depth map estimation method and manually created reference depth maps. Finally, Chapter 7 contains a discussion about the results, suggestions for future work and more.

2
Related Works

Projective Geometry is one of the fundamental topics in Computer Vision. A well-regarded introduction to how Projective Geometry can be applied to Computer Vision is given in [7]. The book covers key topics in Single View Geometry such as vanishing points, vanishing lines, homographies, single view metrology and camera geometry, and also key topics in Multiple View Geometry such as epipolar geometry, warping between different camera poses (geometrical 3D warping) and 3D reconstruction from multiple camera views. There is a rich body of research focusing on various of the topics covered in the book.

One of the key topics of the thesis is generation of a virtual view from a single image. A theoretical description of virtual view generation is given in [7]. A survey of various techniques for generating virtual views is presented in [8]. In general the techniques for generating a virtual view of a scene can be divided into three categories [9]:

- Image Based Rendering (IBR) [9]. The new view is generated using two or more existing views of the scene.
- Reconstructing a 3D model of the scene and viewing the 3D model from the desired position and with the desired rotation [9].
- Depth-Image Based Rendering (DIBR) [9]. The view is generated using the texture/color information from an image and the depth information provided in the form of a depth map.

Image Based Rendering has mostly been ignored in this thesis since only one view/image of the scene is available.

There are numerous methods for 3D reconstruction from one or more views of the scene [7] [5]. Once again, the focus has been put mostly on the methods that require only one image/view. In [10], Criminsi *et. al* describe how to make affine 3D measurements in a single projective view of a scene and use these measurements to reconstruct a 3D model. Their method requires the computation of a vanishing line for a plane and the computation of a vertical vanishing point. Another method of 3D reconstruction from a single projective image is presented by Wang *et. al* in [11]. They present a way to recover both the internal parameters of the camera and the camera pose relative to a world coordinate system. A 3D model can then be constructed by approximating the scene by planar patches. These and other similar methods primarily work well for structured (man-made) environments. Furthermore, the methods usually require some user interaction (or at least prior knowledge). In [11] for instance, specification of three pairs of line segments which have equal lengths is required. Finally, for these methods to be useful there has to be a way to easily and automatically find the correct camera pose (position and orientation) and the 3D models have to be in the correct perspective projection view.

Depth-Image Based Rendering is the virtual view generation method used in the implementation. The fundamental part of any DIBR technique is 3D image warping (also known as geometrical 3D warping) and it is described in numerous sources, including [7], [9], [12] and [13]. A very common application of DIBR is the generation of stereoscopic images or video for 3DTV [12] [13] [14]. The two key issues with DIBR is that the warped views contain artifacts in the form of holes and ghosting (edge color spilling over to adjacent pixels) and that a depth map is required for the warping. Hole-filling (inpainting) and depth map generation are two major research topics on their own.

There are many different approaches to inpainting ranging from simple interpolations to advanced hole-filling combining several different techniques (including the use of the depth map) to generate a warped view with as few artifacts as possible. In [15], Telea describes a relatively fast and efficient method of inpainting technique based on the Fast Marching Method. Although the method is applied to filling of scratches in photographs, it can also successfully be applied to hole-filling in a warped image. Fehn [12] presents a whole pipeline for 3DTV based on DIBR. He combines linear interpolation and Gaussian smoothing of the depth map with existing hole-filling techniques. The most advanced of the inpainting approaches mentioned here is given by Muddala *et. al* in [13]. Although their approach produces excellent results, it is quite complicated (it requires many different steps such as boundary extraction, foreground-background classification and block matching) and very slow.

As previously mentioned one of the problems that one needs to solve when using DIBR is the acquisition of a depth map. One of the more reliable methods of acquiring a depth map is by using laser-based sensors [16] [17]. This approach however is pretty costly [17] and the resulting depth maps generally have low resolutions [18] (not to mention that requiring extra equipment can be cumbersome). Depth maps can instead be acquired using cameras and computer vision/image analysis techniques.

Depth maps are commonly created using multiple cameras but this approach is both complex and costly [18]. It requires accurate calibration of the cameras and the estimation of the relative poses (differences in position and orientation) of the cameras. The least expensive and least complex approach to estimating depth from multiple cameras is to use of a stereo camera. A stereo camera consists of two lenses with the same orientation (rotation) but separated by a small horizontal distance. The depth map is then created by first estimating the disparity between different views (disparity is inversely proportional to the depth and is one of the cues used by humans in depth perception), which is done by finding correspondences (corresponding image coordinates).

Depth maps can also be estimated from a single image, this is known as monocular depth and it relies on monocular depth cues such as motion, defocus, focus and geometry [18]. Due to the assumption of a static scene there is no need to consider depth map estimation from motion.

Depth estimation from defocus relies on the fact that distant objects appear blurrier in an image compared to closer objects. An example of a method using the defocus cue for depth estimation is presented by Zhuo and Sim in [19]. In their approach they first re-blur the input image using a Gaussian filter and then they compute the ratio of gradients between the original image and the blurred image. From this they then estimate the defocus amount at each part in the image. An issue with defocus-based depth estimation methods is the requirement that the background in an image has to be blurry to some extent [18].

An example of a depth estimation method using the focus cue is described by Haque *et al.* in [18]. They first estimating the focus at each pixel in the image using 2D Gaussian-Hermite moments and then apply Bayesian matting on the sparse focus map obtained.

Both defocus- and focus-based methods *seem* to work pretty well for color images but not for grayscale images. Whether this is actually the case or not has not been investigated in this thesis. In either case, the remaining option is depth estimation from geometry. An important benefit of depth from geometry is that it is for the most part independent of the type of input image (color or grayscale).

There exist numerous approaches to monocular depth estimation from geometry. These methods usually combine estimation of geometric entities such as vanishing points and vanishing lines with some form of segmentation.

Zhao *et al.* [20] describe a simple geometric depth map generation method based on finding the vanishing point in the scene and finding the lines that intersect at the vanishing point. The depth is then estimated using only a few planes. They also combine this method with a depth from motion method to track moving objects. A somewhat similar method is presented by Battiato *et al.* in [21]. This method combines a *geometrical depth map* based on a detected vanishing point with a *qualititave depth map* obtained by color segmentation and classification of regions into sky, mountain and other regions. Similarily to in [20], the vanishing point is used as basis for generating gradient depth planes, although in [21] the location of the vanishing point is used to classify the geometric type of the scene. In [22], Fan *et al.* also detect a vanishing point and classify the geometrical scene type accordingly. They then combine the depth map generated by gradient depth planes with an image labelling method based on the grayscale intensities in the image.

An approach quite different from the ones mentioned above is described by Jung *et al.* in [23]. They first create a staircase depth map by tracing the lines in an edge detection image. Then the depth map is refined using edge preserving smoothing. Another quite different approach is presented in [17] by Kazmi *et al.*. Their method can be used to compute a depth map for a road viewed from a camera located at a significant height above the road. Based on the lane lines in the image they derive a 1D projective transformation for the depth computation. In [24], a scene-independent depth estimation method based on field of view is presented. It requires beforehand knowledge of the camera height and the field of view of the camera but it can be used to fairly accurately compute overall distances (depths), heights and widths in the image. The method alone however cannot be used to create a decent depth map since it does not take into account the scene structure (walls, objects, sky etc.).

More advanced geometry-based depth map estimation methods tend to rely on some form of supervised machine learning. Saxena *et al.* [16] use Markov Random Fields

(MRF) to learn the local plane parameters. Their approach not only estimates the depth but can also be used for 3D reconstruction of a scene. Hoiem *et al.* use a Conditional Random Field (CRF) model to segment the image using hierarchical segmentation and generate two different depth estimates: the minimum depth estimate and the maximum depth estimate. They then use these depth estimates to find the occlusion boundaries in the image (the boundaries between different occluding objects). Zhang and Yan [26] also use machine learning for hierarchical segmentation and use it to label objects as ground, sky or standing object. Each standing object is further classified by the number of vanishing points (no vanishing point, vanishing point on one side and vanishing points on both sides). The output of their method is both a depth map and the recovery of the occlusion boundaries in the image. In [9], Zhang and Yan build upon [26] and present an entire DIBR pipeline consisting of segmentation and region labelling, depth map estimation, warping and inpainting. The segmentation method consists of watershed segmentation followed by a feature-based hierarchical segmentation process. Based on this, the regions are labelled as ground, sky or standing objects. The depth estimation makes use of vanishing points in a similar way as in [26].

The depth map estimation method in the thesis combines several of the ideas mentioned above but does so in a somewhat novel way. The works related to the specific methods used for the depth estimation are summarized below.

Lines are one of the key image features, providing important geometrical information about an image. The classic method for line detection which is still used to this day was developed by Duda and Hart [27]. In their method (The Hough (Line) Transform) lines are represented according to the distance to the origin (radius) and orientation (angle). Lines are detected through voting scheme in a 2D array with the radius on one axis and the angle on the other axis. One of the drawbacks of the Hough Transform is that it is fairly slow in general. The Progressive Probabilistic Hough Transform (PPHT) is a method presented by Matas and Kittler [28] aimed at addressing this problem. Unlike the Hough Transform, it detects line segments rather than lines. Instead of going through all (or most) pixels in the image (which is the case in [27]), the method selects pixels randomly and then a line voting scheme is used only for the selected pixels. A problem with both [27] and [28] is that these methods require tuning of several parameters (not to mention the parameter tuning required for the edge detection image that is the input to these methods). One line segment detection method that solves most of the problems associated with the Hough Transform and probabilistic variants of the Hough Transform is presented by von Gioi *et al.* in [29]. The line segment detector (LSD) method not only provides accurate results with a low number of false detections but it is also linear-time and requires no tuning of parameters.

Vanishing points are another important source of geometric information about a scene and are important in various applications including camera calibration and depth map estimation (as noted in the preceding paragraphs). There is a wide body of research on topic of estimation of vanishing points. Chappero *et al.* [30] combine the results of three different methods to detect vanishing points: Hough Transform, Gradient-based method and Mean-Shift Segmentation followed by Hough Transform. Each of the methods result in a voting matrix (which shows the distribution of where a vanishing point is most likely to be located) and the voting matrices are then weighted together to determine the vanishing point. Kong *et al.* [31] use a voting scheme based on confidence-weighted Gabor filters to detect a vanishing point and then use a vanishing point constrained edge detection method to find the boundaries of a road. Another vanishing point detection method focused on road images is presented by Wu *et al.* in [32]. Line segments are first detected using the LSD method, followed by removal of lines that are most likely irrelevant (nearly horizontal or nearly vertical lines for instance). The line segments are then weighted according to their lengths and orientations. Finally a voting scheme is employed to find the vanishing point. The methods mentioned so far can be used only to find a single vanishing point in the image. The method developed by Nieto and Saldago [33] can however be used to find multiple vanishing points simultaneously. They use an orientation-based error function (for detected line segments or for gradients) and combine robust methods such as MSAC or MLESAC with non-linear optimization.

Segmentation is the grouping of an image into (hopefully) meaningful regions and is one of the key topics in computer vision/image analysis [37]. Thresholding is the simplest segmentation method but has a major drawback: A threshold has to be selected. There are however methods for automatic threshold selection (based on the image histogram for instance) such as the one described by Zack *et al.* in [34]. Watershed segmentation is one of the common types of segmentation. Meyer [35] describes a version of watershed segmentation based on image markers (initial labels for regions in the image). GrabCut [36] (developed by Rother *et al.*) is a background-foreground segmentation based on Gaussian Mixture Models (GMMs) and energy minimization. It is accurate in general but requires some user input (for instance a few pixels marked as background and a few pixels marked as foreground). The Hierarchical Feature Selection (HFS) segmentation method [37] (developed by Cheng *et al.*) is an advanced but fast segmentation method. The method uses learned Support Vector Machine (SVM) features and combines Simple Linear Iterative Clustering (SLIC) with Efficient Graph Based (EGB) segmentation.

A related area of computer vision/image analysis is blob detection. Blob detection is used to find regions in an image that are (hopefully) interesting in some way. The Maximally Stable Extremal Regions (MSER) method [38] was proposed by Matas *et al.* for the purpose of matching correspondences between stereo image pairs. The first step in the algorithm is to threshold the image for all possible grayscale levels. The regions in different threshold images are then either expanded or kept unchanged depending on the change in the region area. The final stable regions in the image are the blobs/regions detected by MSER. Nistér and Stewénius [39] present a linear-time MSER algorithm.

The final topic of this section is the luminance estimation for threshold zone in a tunnel. A guide for tunnel lighting, including the method for estimating the threshold zone luminance L_{th} based on calculating either the L_{seq} or the L_{20} value in an image is provided in [2] by Commission Internationale de l'Éclairage (CIE). In [3], a research study was done by Jonsson *et al.* to investigate different sources of error in the calculation of L_{seq} for a camera centered at the tunnel opening mounted 7 meters above the road plane and on the side of the road. One of the conclusions of the report was that the main source of error in the L_{seq} calculation is due to the camera pose (camera position and orientation) difference between the ideal camera pose for measuring L_{seq} and the camera pose used in practice.

3
Theory

This chapter presents some of the theory that is important for understanding the implementation of the project. How this theory is used in practice is described in Chapter 4. Note that some of the sections in this chapter are fairly brief. The main reason for this is that they describe techniques that already have implementations in the OpenCV library (which was used in the project) and thus have not been implemented (programmed) as part of the project. Therefore, a thorough understanding of these specific techniques is not required to understand the rest of the report. Refer to the sources for more detailed information about the techniques used. At any rate, before the theory is presented, it is important to take note of some basic conventions used in this report.

Throughout the report (unless stated otherwise), vectors are represented in column form $\mathbf{x} = (x_1, x_2, \cdots)^T$ and matrices are thus post-multiplied, i.e. $\mathbf{b} = A\mathbf{x}$. Another thing to note is the image coordinate system shown in Figure 3.1. The origin of the coordinate system is at the upper-left corner in the image and the y-axis is pointing down rather than up. For an image of size $W \times H$, the bottom right coordinate is $(x, y) = (W - 1, H - 1)$. Finally, how the 3D coordinate system is defined is usually clear from the context. In most cases however, it is assumed that the Z-axis is pointing in the same direction as the camera and that the X- and Y-axes represent horizontal and vertical displacement respectively.

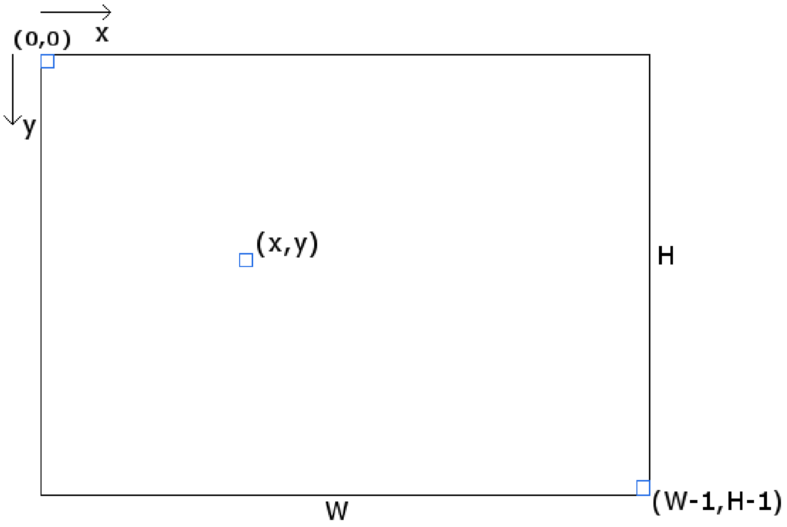


Figure 3.1: Image coordinate system. Note that y-axis is pointing down.

3.1 Projective Geometry

As previously mentioned in Chapter 2, projective geometry is one of the cornerstones of computer vision. It is defined as the "branch of mathematics that deals with the relationships between geometric figures and the images, or mappings, that result from projecting them onto another surface" [41]. A key reason why projective geometry is so important in computer vision is because images are a result of projections of the observed 3-dimensional world into a 2-dimensional image plane [7]. By analyzing geometric concepts using projective geometry a mathematical description of the relationships between the 3-dimesional world and a 2-dimensional image can be obtained.

Projective geometry can be viewed as an extension of Euclidean geometry [7]. Although Euclidean geometry satisfactorily describes fundamental geometric concepts such as angles and shapes of objects, it fails to adequately describe various other geometric concepts. One such example is the intersection of two parallel lines. While all other types of line intersections (in two dimensions) can easily be handled in Euclidean geometry, the intersection of two parallel lines is a special case. Two parallel lines are said to meet at a point at infinity (also referred to as an "ideal point") but such a point is not defined in Euclidean geometry. Projective geometry expands upon Euclidean geometry by also including points at infinity/ideal points.

Points in projective geometry are notated using homogeneous coordinates (also known as projective coordinates) [7]. In contrast to Cartesian coordinates, which are defined by ordered n-tuples (where n is the number of dimensions), homogeneous coordinates are defined by ordered (n+1)-tuples. Thus in the one-dimensional projective space $\mathbb{P}^1$ a coordinate point is defined as $\mathbf{x} = (x, w)^T$. Similarly, in $\mathbb{P}^2$ the homogeneous coordinates are defined as $\mathbf{x} = (x, y, w)^T$ and in $\mathbb{P}^3$ as $\mathbf{X} = (X, Y, Z, W)^T$, and so on.

Homogeneous coordinates have some special properties. Firstly, a non-zero multiple of a homogeneous coordinate is still the same homogeneous coordinate [7]. For instance $(2x, 2y, 2)^T$ is the same coordinate as $(x, y, 1)^T$. More generally $(kx, ky, k)^T$ is considered to be the same coordinate point as $(x, y, 1)^T$. Secondly, points at infinity (or ideal points) are represented as $(x, y, 0)^T$.

A homogeneous coordinate can be converted into a Cartesian (inhomogeneous) coordinate by first dividing all the entries in the (n+1)-tuple by the value in the last entry and then only keeping the first n entries in the tuple. For instance an Cartesian/inhomogeneous coordinate in $\mathbb{R}^2$ can be obtained as follows: $\tilde{\mathbf{x}} = (x/w, y/w)^T$. A Cartesian coordinate on the other hand is converted to a homogeneous coordinate by extending the n-tuple to an (n+1)-tuple where the last entry is either 1 or 0 (depending on if the point is finite or if the point is at infinity). For instance a Cartesian coordinate $\tilde{\mathbf{x}} = (x, y)^T$ can be converted to a homogeneous coordinate $\mathbf{x} = (x, y, 1)^T$

The use of homogeneous coordinates is fundamental to many (if not most) of theorems and results in projective geometry and often results in more eloquent mathematical descriptions of geometric objects and relationships. One key example that is used several times in the implementation of the project is the relationship between 2-dimensional lines and points.

The most general description of a line in two dimensions is given by $ax+by+c = 0$ [7]. In homogeneous coordinates a two-dimensional line is notated as $\mathbf{l} = (a, b, c)^T$. The intersection point of any two two-dimensional lines $\mathbf{l}_1$ and $\mathbf{l}_2$ is given as $\mathbf{x} = \mathbf{l}_1 \times \mathbf{l}_2$, regardless if the lines are parallel or not. Similarly, a two-dimensional line passing through two homogeneous points $\mathbf{x}_1$ and $\mathbf{x}_2$ is computed as $\mathbf{l} = \mathbf{x}_1 \times \mathbf{x}_2$.

Planes are another key geometric entity in projective geometry [7]. A plane in 3-space is given by

$$\pi_1 X + \pi_2 Y + \pi_3 Z + \pi_4 = 0 \tag{3.1}$$

and is represented using homogeneous coordinates as

$$\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3, \pi_4)^T. \tag{3.2}$$

Equation (3.1) can be homogenized and written as follows

$$\pi_1 X_1 + \pi_2 X_2 + \pi_3 X_3 + \pi_4 X_4 = 0, \tag{3.3}$$

or in a concise form as

$$\boldsymbol{\pi}^T \mathbf{X} = 0, \tag{3.4}$$

where X has been replaced by X_1/X_4, Y by X_2/X_4 and Z by X_3/X_4.

The plane equation can also be written in Euclidean form [7] as follows

$$\mathbf{n} \cdot \tilde{\mathbf{X}} + d = 0, \tag{3.5}$$

where $\mathbf{n} = (\pi_1, \pi_2, \pi_3)^T$ is the plane normal, $\tilde{\mathbf{X}} = (X, Y, Z)^T$ is the inhomogeneous 3D coordinate, $d = \pi_4$ and $X_4 = 1$. The distance from the origin of the coordinate system to the plane is then defined as $D = d/\|\mathbf{n}\|$.

A special type of plane that is among other things useful when analyzing virtual view generation (see Section 3.1.4) is the plane at infinity π_∞ [7]. It is defined as $\boldsymbol{\pi}_\infty = (0, 0, 0, 1)^T$ and it contains all ray directions $\mathbf{d} = (X_1, X_2, X_3, 0)^T$.

Projective geometry as applied to computer vision can be divided into several parts: Single View Geometry, Two-View Geometry, Three-View Geometry and n-View Geometry [7]. The main differences between these topics is the number of camera views available. Single View Geometry is most important in the context of this thesis because as previously mentioned in the introduction only one camera view is available in the final product. However, some basic understanding of Two-View Geometry is also important because it gives inslight into how two camera views are related and how a camera view can be can be warped to desired virtual view and why simple 2D transformations (including homographies) are not enough for general view transformations. The main topics of interest in this overview of projective geometry are: Vanishing Points, Image Transformations, Camera Geometry and Virtual View Generation.

3.1.1 Vanishing Points

One of the characteristic features of projecting the 3-dimensional world into an image (perspective projection) is that objects that have seemingly infinite extent may have finite extent in the image [7]. This can be seen by observing parallel lines (for instance the rails of a train track shown in Figure 3.2) and noting that in the distance the lines seem to converge to a single point. It turns out that this type of point, referred to as a Vanishing Point, is a very important geometric concept in computer vision since it provides much information about the geometric structure in structured environments [33]. Vanishing points can for instance be used for single view metrology, estimation of the calibration matrix K (see Section 3.1.3 for the definition of K) from a single view and even as part of monocular depth map estimation.

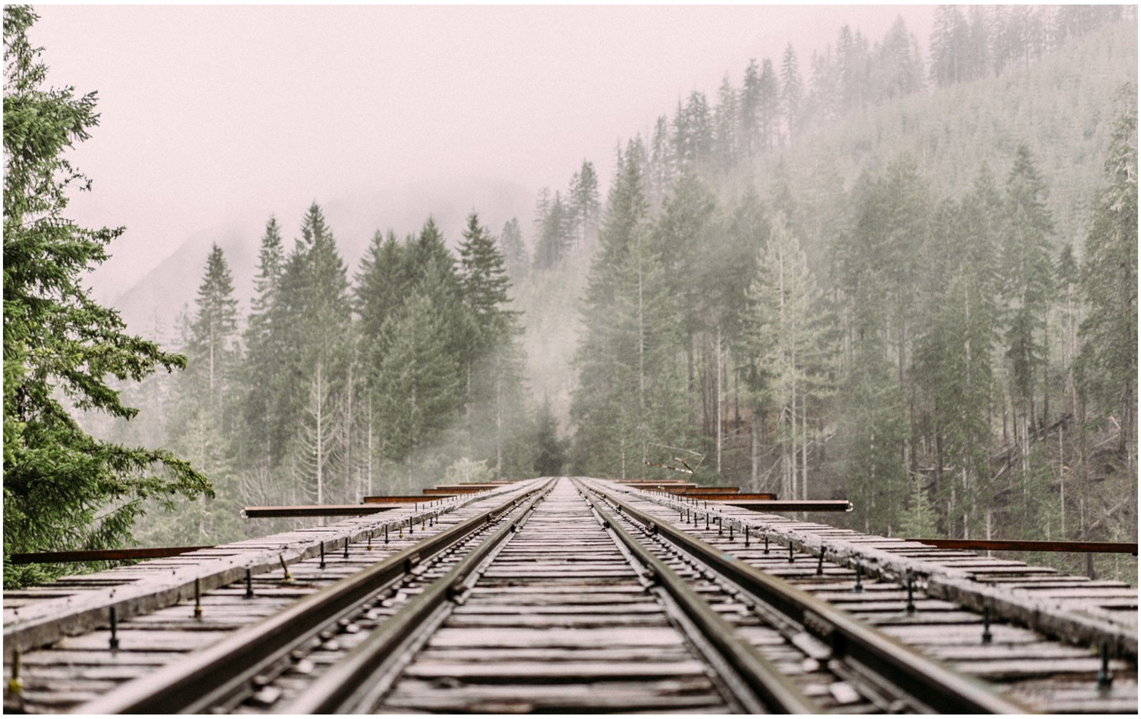


Figure 3.2: An example of a case where a vanishing point can easily be determined. Note that the train tracks, despite being parallel, seem to converge to a single point. This is the vanishing point. This image was downloaded from from https://www.maxpixel.net/Train-Tracks-Railroad-Tracks-Railway-Pine-Trees-1245906 and was shared under the Creative Commons Zero (CC0) license.

3.1.2 Image Transformations

In the most general terms a 2D image transformation is a mapping from image coordinates in the source image to image coordinates in the destination image [40]. In other words a mapping

$$T : \mathbb{R}^2 \to \mathbb{R}^2 \tag{3.6}$$

is applied to each image coordinate $\mathbf{x} = (x, y)^T$:

$$(x', y')^T = T(x, y). \tag{3.7}$$

Although there exist many non-linear 2D image transformations, the transformations of interest in this case are linear transformations of homogeneous coordinates. This type of transformation, known as a projectivity (and sometimes also called a homography or a projective transform) [7], is an invertible mapping

$$h: \ \mathbb{P}^2 \rightarrow \mathbb{P}^2 \tag{3.8}$$

from an input homogeneous coordinate $\mathbf{x}$ to an output homogeneous coordinate $\mathbf{x}'$ such that

$$\mathbf{x}' = h(\mathbf{x}) = H\mathbf{x}, \tag{3.9}$$

where H is a general non-singular 3×3 matrix, commonly referred as a homography matrix. Equation (3.9) can also be written as

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}. \tag{3.10}$$

H has up to 9 non-zero entries but it has only the eight independent ratios since multiplication of the matrix with a scale factor results in the same transformation [7]. The most important specializations of the projective transform and their geometric properties are described in the following paragraphs (described in order from the most specialized to the most generalized transformation).

Isometries. Isometries are one of the simplest specializations of the projective transform and are defined as follows

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \epsilon \cos\theta & -\sin\theta & t_x \\ \epsilon \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3.11}$$

where θ is the angle of rotation of the 2D image plane, t_x and t_y are the translations of the x and y coordinates respectively. When $\epsilon = 1$ the transformation is composed solely of a translation and rotation. On the other hand if $\epsilon = -1$ the transformation is also composed of a reflection. The main geometric properties preserved by the transformation (also known as invariants) are areas, lengths and angles.

Similarity transformations. Similarity transformations (also called similarities) are defined as follows [7]

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{3.12}$$

The main difference between isometries and similarities is the added scale factor s which represents isotropic (uniform) scaling [7]. A more concise way to write equation (3.12) is as follows

$$\mathbf{x}' = H_S\mathbf{x} = \begin{bmatrix} sR & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}, \tag{3.13}$$

where R is a 2×2 rotation matrix, $\mathbf{t}$ is a 1×2 translation vector and $\mathbf{0}^T$ is a 2×1 zero vector. Neither of the three operations performed by the matrix (rotation, translation and isotropic scaling) changes the shape of geometric objects [7]. The scaling does not change angles but it does change the area and length of objects. The ratio of two areas or the ratio of two lengths are however invariants (preserved by the transformation).

Affine transformations. The next generalization is given by affine transformations, defined as [7]

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.14}$$

and can be written more compactly as

$$\mathbf{x}' = H_A\mathbf{x} = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{x}, \tag{3.15}$$

where A is any non-singular 2×2 matrix. Except for translation, rotation and scaling, affine transformations can also deform objects by non-isotropic scaling in the x and y directions and by shearing. Because the shape of objects is changed by a general affine transformation, angles and ratio of lengths are no longer invariants. Three important invariants preserved by affine transformations are parallel lines (parallel lines are mapped to other parallel lines), ratio of lengths of parallel line segments and ratio of areas.

Projective transformations. The final specialization and the most general linear 2D transformation operating on homogeneous coordinates is the projective transformation [7]. It was previously stated in equation (3.10) and can be written in a more compact form as

$$\mathbf{x}' = H_P\mathbf{x} = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^T & v \end{bmatrix} \mathbf{x}, \tag{3.16}$$

where $\mathbf{v} = (v_1, v_2)^T$ represents the projective part of the transformation and v is a scalar. As mentioned before, only the ratio of the eight independent elements is of importance and the transformation thus has eight degrees of freedom (dof).

The homography matrix defining the projective transformation can be computed by four point point pairs (point correspondences) under the condition that no three points are collinear (are on the same line) [7]. To see why this is the same note that equation (3.10) can be re-written in inhomogeneous form as

$$x' = \frac{x_1'}{x_3'} = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \tag{3.17}$$

$$y' = \frac{x_2'}{x_3'} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}. \tag{3.18}$$

Multiplying both equations by the denominators gives the following

$$x'(h_{31}x + h_{32}y + h_{33}) = h_{11}x + h_{12}y + h_{13}, \tag{3.19}$$

$$y'(h_{31}x + h_{32}y + h_{33}) = h_{21}x + h_{22}y + h_{23}. \tag{3.20}$$

It can be noted that these two equations are linear in the elements of H and since there are eight degrees of freedom, 4 point correspondences are required.

3D Projective Transformations

Projective transformations in 3 dimensions are defined in an analogous fashion. A projectivity in 3 dimensions is an invertible mapping [7]

$$h:\ \mathbb{P}^3 \to \mathbb{P}^3, \tag{3.21}$$

such that

$$\mathbf{X}' = h(\mathbf{X}) = H\mathbf{X}, \tag{3.22}$$

where $\mathbf{X}$ is the input 1×4 homogeneous coordinate, H is a general non-singular 4×4 matrix and $\mathbf{X}'$ is the output homogeneous coordinate [7]. The specializations of the projectivity are listed below.

An Euclidean transformation matrix is defined as

$$\begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}, \tag{3.23}$$

where $\mathbf{0}$ is a 1×3 zero vector, $\mathbf{t} = (t_X, t_Y, t_Z)^T$ is a 3-dimensional translation vector and R is a 3D rotation matrix of size 3×3 [7].

The next generalization is a similarity, defined as follows

$$\begin{bmatrix} sR & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}, \tag{3.24}$$

where s is a unifrom scaling factor [7].

Next follows the affine transformation

$$\begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}, \tag{3.25}$$

where A is an arbitrary non-singular 3×3 matrix [7].

Finally, a general 3-dimensional projective transform can be written as

$$\begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^T & \upsilon \end{bmatrix}, \tag{3.26}$$

where $\mathbf{v}$ is a general 1×3 vector and υ is a scalar [7].

It is worth noting that the rotation matrix R for a 3D projective transformation is significantly more complicated to define. There are in fact several different ways to represent a rotation in 3 dimensions, including through Euler angles, exponential twist and quaternions [5]. In this report Euler angles are used to represent 3D rotations.

Using the Euler angle representation a 3D rotation (around the origin) can be completely determined by three angles: pitch θ_x, yaw θ_y and roll θ_z, corresponding to rotation angles around the X-axis, Y-axis and Z-axis respectively [6]. The three angles can be converted to a rotation matrix as follows

$$R = R_z R_y R_x, \tag{3.27}$$

where R_x, R_y and R_z are the rotation matrices for rotations around the X-axis, Y-axis and Z-axis respectively and are defined in

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix}, \tag{3.28}$$

$$R_y = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \tag{3.29}$$

and

$$R_z = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.30}$$

3.1.3 Camera Geometry

A camera is an object that projects the 3D world into a 2D image [7]. Although there exist several types of cameras, the most common type is central projection, given in its' most general form by the following equation:

$$\mathbf{x} = P\mathbf{X}, \tag{3.31}$$

where $\mathbf{x}$ is a homogeneous 2D image coordinate, P is a 3×4 matrix referred to as the camera matrix and $\mathbf{X}$ is a homogeneous 3D world coordinate [7]. The camera matrix can be decomposed as follows

$$P = K[R|\mathbf{t}] = KR[I| - \tilde{\mathbf{C}}], \tag{3.32}$$

where R is a 3×3 rotation matrix, $\tilde{\mathbf{C}}$ is the camera origin (camera centre) in inhomogeneous 3D coordinates, $\mathbf{t} = -R\tilde{\mathbf{C}}$, I is the identity matrix and K is a 3×3 matrix known as the calibration matrix and contains the intrinsic parameters of the camera [7]. Both the rotation R and the camera origin $\tilde{\mathbf{C}}$ (the combination of both is known as the camera pose) are set relative to a world coordinate system. It is common practice to set one of the cameras at the origin of the world coordinate system with the camera pointing in the Z direction (no rotation). This is known as the canonical position and the camera matrix is in this case given by

$$P = K[I|\mathbf{0}], \tag{3.33}$$

where $R = I$ and $\tilde{\mathbf{C}} = \mathbf{0}$ is the 3×1 zero vector [7]. The poses (positions and rotations) for the other cameras are then set relative to this camera.

As previously mentioned, K is the calibration matrix and contains the intrinsic parameters of the camera [7]. The simplest form of the calibration matrix is defined as follows

$$K = \begin{bmatrix} f & & \\ & f & \\ & & 1 \end{bmatrix}, \tag{3.34}$$

where f is the focal length of the camera in pixels (f can also be given as in a metric unit (usually millimeters)) [7]. The focal length is defined as the distance between the camera centre and the image plane in the direction of the camera Z-axis (known as the principal axis). The image plane is given in Euclidean coordinates as $Z = f$. The point at which the principal axis (camera Z-axis) intersects with the image plane is known as the principal point $\tilde{\mathbf{x}}_0 = (x_0, y_0)^T$ [7]. In the calibration matrix given in equation (3.34) it is assumed that the origin of the image coordinate system is at the principal point. However, in practice this is rarely the case. The following generalization takes this discrepancy into account

$$K = \begin{bmatrix} f & & x_0 \\ & f & y_0 \\ & & 1 \end{bmatrix}. \tag{3.35}$$

Equation (3.35) assumes that the pixels in the camera are square-shaped, but in practice many cameras have non-square pixels (due to flaws in the camera sensor) [7]. In those cases the camera calibration matrix can be generalized as follows:

$$K = \begin{bmatrix} f_x & & x_0 \\ & f_y & y_0 \\ & & 1 \end{bmatrix}, \tag{3.36}$$

where f_x and f_y are the focal lengths in pixel units in the x and y directions respectively. The focal lengths can be calculated from the focal length in metric units f_m (the metric distance between the camera centre and the image plane) as follows: $f_x = f_m m_x$ and $f_y = f_m m_y$ where m_x and m_y are the the numbers of pixels per metric unit in the x and y directions respectively.
The calibration matrix is given in its' most general form by

$$K = \begin{bmatrix} f_x & s & x_0 \\ & f_y & y_0 \\ & & 1 \end{bmatrix}, \tag{3.37}$$

where s is the skew parameter [7]. $s \neq 0$ means that the x- and y-axes of the camera are not perpendicular. Non-zero skew only occurs in unusual cases such as when taking a photo of a photo.
Note that the parameters in the calibration matrix K are defined in pixel units. It is however possible to convert these parameters to metric units. For instance, the focal length f in pixel units can be converted to the focal length metric units f_m either by

$$f_m = \frac{f}{m}, \tag{3.38}$$

where m is the number of pixels per metric unit (usually millimeter) or by

$$f_m = f s_p, \tag{3.39}$$

where s_p is the pixel size (usually in micrometers).

The camera matrix has many unique properties that will not be listed here (refer to [7] for more theoretical background) but a property of key importance is back-projection, in other words the projection of a 2D image coordinate $\mathbf{x}$ to 3D using the camera matrix P [7]. Before defining the back-projection it is important to note that all depth (Z-coordinate) information is lost when a 3D coordinate $\mathbf{X}$ is forward projected using equation (3.31). Therefore, a simple back-projection does not retrieve the full 3D coordinate $\mathbf{X}$. Instead, an image coordinate $\mathbf{x}$ is back-projected to a ray, which can be written as the join of two points on the ray from the camera centre $\mathbf{C}$ to the 3D coordinate $\mathbf{X}$ as follows

$$\mathbf{X}(\lambda) = P^{+}\mathbf{x} + \lambda\mathbf{C}, \tag{3.40}$$

where P^{+} is the pseudo-inverse of the camera matrix. The pseudo-inverse is defined as $P^{+} = P^{T}(PP^{T})^{-1}$ and has the property that $PP^{+} = I$ where I is the identity matrix. To confirm that $P^{+}\mathbf{x}$ is a point on the ray note that $P(P^{+}\mathbf{x}) = I\mathbf{x} = \mathbf{x}$. Thus, $P^{+}\mathbf{x}$ is forward-projected to $\mathbf{x}$ as expected.

3.1.4 Virtual View Generation

Virtual view generation is the process of generating an arbitrary new view of a scene from an existing image [7]. This can be done by first defining a camera matrix for the second view relative to the first view. A common way to do this is to set the first camera at the origin with no rotation, in other words setting $\mathbf{C} = \mathbf{0}$ and $R = I$ as described in Section 3.1.3. The first camera matrix can then be defined as $P = K[I|\mathbf{0}]$ and the second one (corresponding to the virtual view) can be defined as $P' = K'[R|\mathbf{t}]$. When generating a virtual view it is usually assumed that the same camera is used for the virtual view as for the original view, thus one can usually set $K' = K$.

Assuming that $\mathbf{x} = (x, y, 1)^T$ is the normalized homogeneous coordinate in the first (original) view, then from

$$\mathbf{x} = P\mathbf{X} = K[I|\mathbf{0}]\mathbf{X} \tag{3.41}$$

the inhomogeneous world 3D coordinate is given as $\tilde{\mathbf{X}} = (X, Y, Z)^T = ZK^{-1}\mathbf{x}$ [7]. The 3D-coordinate $\mathbf{X}$ can then be forward projected using the camera matrix for the warped view P' as follows [7]

$$\mathbf{x}' = P'\mathbf{X} = K'[R|\mathbf{t}]\mathbf{X} = K'RK^{-1}\mathbf{x} + K'\mathbf{t}/Z. \tag{3.42}$$

It can be noted that the first term in equation (3.42) takes account of the rotation and the change of the internal parameters and that it only requires knowledge of the image position in the original view [7]. The second term on the other hand takes account of the camera translation and depends on the depth of the point Z but does not depend on the image coordinate in the original view. Thus, a general transformation between two arbitrary camera views requires not only knowledge of the camera matrices but also the depth information, which in practice means that in most cases the depth information is required to perform the transformation. But it can also be noted that an exception to this rule is if there is no translation between the cameras ($\mathbf{t} = \mathbf{0}^T$) or when $Z = \infty$. A more formal description of this exception is given below.

It is shown in [7] that the images of points $\mathbf{X}_\pi$ on a plane π are related by a projective transformation H as follows

$$\mathbf{x}' = H\mathbf{x} = K'(R - \mathbf{t}\mathbf{n}^T/d)K^{-1}\mathbf{x}, \tag{3.43}$$

where $\mathbf{n}$ is the plane normal and d is the orthogonal distance from the first camera to the plane [7].

A particularly important projective transformation of this kind is the infinite homography H_∞, which is induced by the plane at infinity π_∞ (π_∞ was defined in Section 3.1) [7]. This homography can be derived by the following limiting process

$$H_\infty = \lim_{d \to \infty} H = \lim_{d \to \infty} K'(R - \mathbf{t}\mathbf{n}^T/d)K^{-1} = K'RK^{-1}. \tag{3.44}$$

Equation (3.42) can thus be re-written as follows [7]

$$\mathbf{x}' = K'RK^{-1}\mathbf{x} + K'\mathbf{t}/Z = H_\infty\mathbf{x} + K'\mathbf{t}/Z. \tag{3.45}$$

The importance of this is result is two-fold: Firstly, if there is no translation between the cameras, i.e. $\mathbf{t} = \mathbf{0}$, no explicit depth information Z is needed and the views are related by a simple homography [7]. Secondly, points far away (for instance the sky) can be approximated as having $Z = \infty$ and thus can be seen as points on π_∞ (because of the inverse depth $\frac{1}{Z}$, the error will be negligible). This makes sense intuitively considering that points closer to the camera appear to be more affected by translation than points further away.

An alternative way to define warping between two arbitrary camera views, which does not require that one of the cameras is set at the origin, is given in [13]. Instead of defining the camera matrix P as a 3×4 matrix, it can be defined as a 3-dimensional 4×4 homography matrix as follows

$$P = KI_{3\times4}\begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}, \tag{3.46}$$

where K is the 3×3 calibration matrix, R is the 3×3 rotation matrix, $\mathbf{t} = -R\tilde{\mathbf{C}}$ as before, $I_{3\times4}$ is the 3×4 identity matrix and $\mathbf{0}$ is a 1×3 zero-vector. The main difference between this representation of the camera matrix and the previous one defined in Section 3.1.3 is that one row of has been added to make it a 4×4 matrix and thus a 3-dimensional homography matrix. According to [13] the homography matrix is invertible and given by

$$P^{-1} = \begin{pmatrix} R^{-1}K^{-1} & -R^{-1}\mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}. \tag{3.47}$$

The overall transformation from 2D coordinates in the first view $\mathbf{x}$ to 2D coordinates in the second view $\mathbf{x}'$ is given by [13]

$$Z'\mathbf{x}' = ZP'P^{-1}(\mathbf{x}), \tag{3.48}$$

where Z is the depth value in the original view, Z' is the depth value in the warped view and P and P' are the 4×4 camera matrices for the original view and the virtual view respectively.

3.2 Inpainting

As described in Section 3.1.4, in general an arbitrary new view of the scene can be generated provided that the depth information (Z-values) is known. In practice this requires both the original image (also referred to as the texture) I providing the color information and a depth map D which is an image of indices where each index corresponds to a depth value (see Figure 3.3 for an example). The process of generating a virtual view from an image and a depth map is referred to as Depth Image Based Rendering (DIBR) [13].

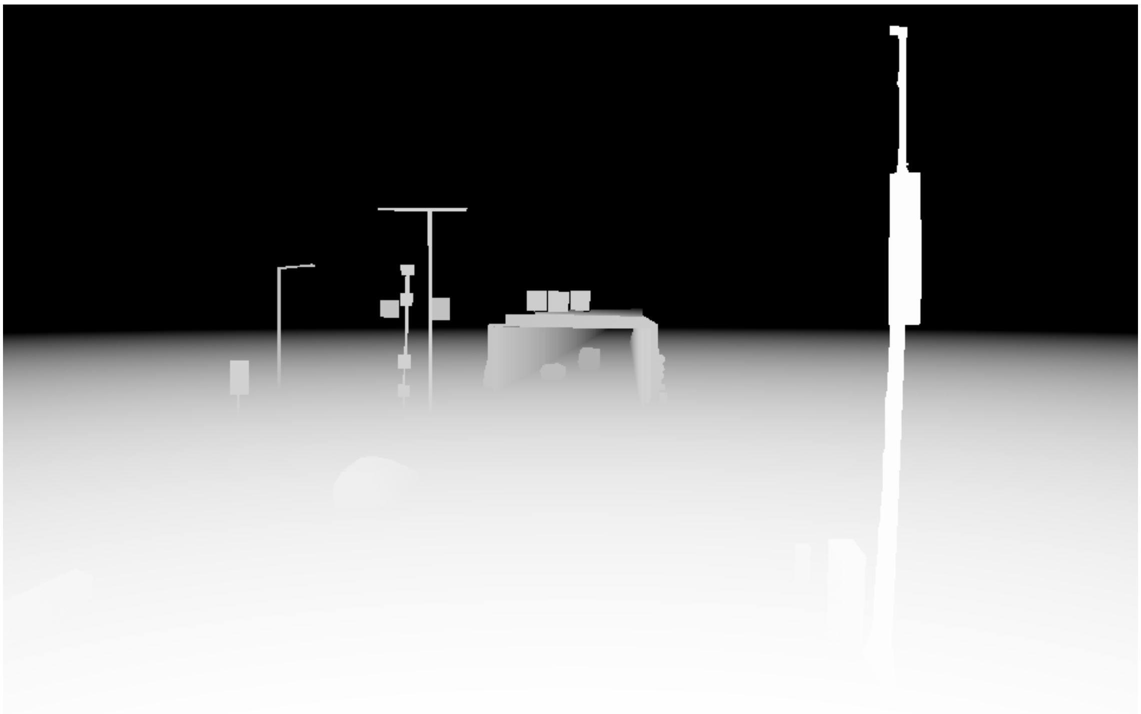


Figure 3.3: Example of a depth map. Note that this depth map was manually defined in GNU Image Manipulation Program (GIMP) and only defines the depth for some of the objects in the scene.

There are two types of artifacts produced by DIBR: ghosting and holes [13]. Ghosting is a result of misalignment of the texture (color information) and the depth map (which is a consequence of inaccuracies in the depth map). It is visible in the image in the form of color information at the edges being spilled over to regions with discontinuities in the depth. This is mostly a minor issue and can be glossed over

for the most part. Holes on the other hand are a more severe type of artifact which significantly degrades the quality of the warped image.
Holes can be classified into cracks, disocclusions and out-of-field areas [13]. Cracks are small holes which are usually 1-2 pixels wide and occur due to rounding of warped floating-point image coordinates to the nearest integer values. Disocclusions on the other hand are caused by significant differences in depth values between neighboring objects and tend to cause much larger holes in the warped image. Finally, out-of-field areas occur at the boundaries of the warped image and are caused by the lack of color (and depth) information at those regions in the original image.
The process of filling holes in an image is known as inpainting, it is not only applied in DIBR but has also other applications such as scratch removal [5]. The main use for inpainting in the thesis is for a more decent visualization of the warped view. After all what is most important in the thesis is a more accurate estimation of the luminance value which can be obtained by warping the original view rather than a good visualization of the warped view (as shown in Section 6.3, the influence of inpainting on the L_{seq} value is minimal). Consequently, only one method of inpainting is used in the implementation, namely "An Image Inpainting Technique Based on the Fast Marching Method" by Telea. Refer to the source [15] for a detailed description of the technique.

3.3 Line and Line Segment Detection

As previously mentioned in Chapter 2, lines are an important source of geometric information about a scene. This is especially true for structured environments (i.e. environments with many man-made structures). Furthermore, line segments can also be used to approximate the shapes of curved objects. For these reasons there is a rich body of literature covering the topic of line detection or line segment detection. The method of line segment detection chosen in the implementation is "LSD: A Fast Line-Segement Detector with a False Detection Control", presented by von Gioi *et al.* in [29].

3.3.1 Line Segment Detector (LSD)

The Line Segment Detector (LSD) method is a linear-time method for line segment detection [29]. It yields accurate results, with only a small number of false detections and unlike traditional line or line segment detection methods such as Hough Transform [27] and Probablistic Hough Transform [28] it requires no parameter tuning.
The first step in the LSD algorithm/method is to apply a region growing algorithm on the image [29]. Initially every pixel in the image is marked as not visited. For a pixel marked as not visited, a region growing algorithm is applied. Adjacent pixels that have approximately the same gradient orientation (within an angle threshold τ) are added to the line-support region. Once there are no longer any adjacent pixels to add based on the threshold then the algorithm terminates. The pixels included in the region are marked as visited and are not used as starting points for the region growing algorithm.

In the next step, each region is approximated using a rotated rectangle. This rectangle is defined by the center point, the orientation, the length and the width [29]. The center point is assigned by computing the center of mass and the orientation is defined by the first inertia axis. The length and width of the rectangle are then selected in such a way that the rectangle covers the whole line-support region.
The final step in the LSD method is a validation step [29]. To determine whether a line segment is valid, the Number of False Alarms (NFA) is first computed based on a binomial distribution. If the NFA is smaller than a threshold ϵ then the line segment is added to a list of valid line segments, otherwise the line segment is rejected and the pixels in the region are marked as not valid. The not valid pixels cannot be used as starting points in the region growing algorithm but can be included in regions with other starting points. For further details about the method refer to [29].

3.4 Segmentation

Segmentation is one of the key areas of and most widely studied problems in image analysis/computer vision [5] and is related to cluster analysis in statistics. It is the problem of dividing an image into a number of (hopefully) meaningful regions. Some segmentation techniques divide the image into just two regions, sometimes referred to as background/foreground segmentation. Other segmentation techniques instead divide the image into an arbitrary number of regions. The complexity of the segmentation techniques vary from simple thresholding to advanced segmentation based on supervised learning techniques.
The following subsections briefly describe the segmentation techniques used in the project, namely Thresholding, GrabCut and Hierarchical Feature Selection (HFS). For more detailed information about the techniques refer to the sources.

3.4.1 Thresholding

The simplest method of segmentation is thresholding. Thresholding is defined as follows:

$$T(x,y) = \begin{cases} M, & \text{if } I(x,y) > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{3.49}$$

x is the image column coordinate and y is the image column coordinate, $I(x,y)$ is the current pixel value, $T(x,y)$ is the output pixel value, ϵ is a threshold (pixel value) and $M \neq 0$, usually $2^n - 1$ where n is the number of bits per pixel.
There are several drawbacks with thresholding but one of the primary issues is the difficulty in selecting a correct (good enough) threshold value ϵ. There are however several methods for automatically selecting the threshold based on the histogram (pixel value distribution) of the image [43] [44]. The triangle method [34] (as it's commonly referred to) is one such method.
In the triangle method, the first step is to find the peak (highest value) of the histogram of the grayscale image [34] [43]. The next step is to find the farthest end of the histogram, defined as the bin corresponding to either the darkest or brightest

pixel value in the image depending on which one of these two bins is furthest away from the peak in the histogram. A line between the peak of and this bin is then created. The threshold M is then set as the pixel value that maximizes the distance between this line and the histogram. An illustration of this is shown in Figure 3.4.

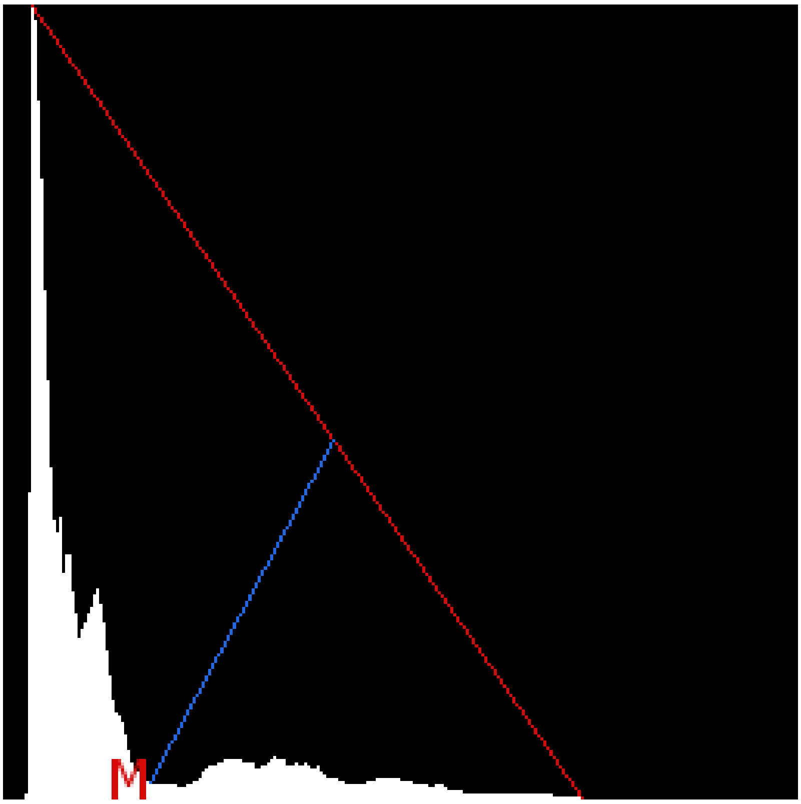


Figure 3.4: Histogram and the triangle threshold M. Note that this is a dummy example. The actual threshold and lines may differ in reality. Refer to [34] for a better illustration.

3.4.2 GrabCut

GrabCut is a semi-interactive method of segmenting color images into foreground and background for color images [36]. The method requires only minor user interaction in the form of using a rectangle to delineating the region to be segmented. In some cases however the result can be improved using additional user interaction in the form of pixels marked as foreground or background.

The method can be summarized as follows: For each of the two labels, foreground and background, a Gaussian Mixture Model (GMM) is created [36]. Each Gaussian mixture has full-covariance and K components. The image region is then segmented using iterative energy minimization which improves the initial foreground/background labeling of the pixels, eventually leading to convergence (i.e. the GMMs do not change after more iterations). For more details about the technique refer to [36].

3.4.3 Hierarchical Feature Selection (HFS)

Hierarchical Feature Selection (HFS) segementation is the most advanced segmentation technique used in the implementation of the project. Unlike the previous segmention methods mentioned, HFS segmentation segments the image into an arbitrary number of regions/classes.

In the first step of the algorithm the image is divided into superpixels using Simple Linear Iterative Clustering (SLIC) [37] [45]. Each superpixel represents a small image region with roughly similar color and texture.

Then in the next step, a graph node is assigned for each superpixel [37] [45]. The weights of the edges (connections between nodes) are computed based on feature weights learned using a Support Vector Machine (SVM) model. Efficient Graph Based image segmentation (EGB) is then applied on the graph to merge different regions. The result of the EGB is then post-processed by merging regions that are smaller than a certain area into a neighboring region.

EGB can be repeated one or more times on the result to get a more coarse segmentation [37] [45].

3.5 Maximally Stable Extremal Regions (MSER)

One of the key parts in the implementation of the thesis is the detection of potentially useful regions in the image. This is done using the Maximally Stable Extremal Regions (MSER) region detector. The following analogy is useful in describing the idea behind MSERs [38]:

Imagine viewing all possible thresholdings of a grayscale image starting from the lowest threshold $t = 0$ up to $t = 255$ [38]. The first image $I_t = 0$ will appear entirely white. As the threshold is increased, black regions start appearing and growing. These correspond to local intensity minima. Eventually the minima will merge and in the end the last thresholded image will be entirely black. The connected components (mergings of different regions) obtained through this process are referred to as the set of all maximal regions. The set of all minimal regions can be obtained by reversing the process, i.e. starting from the highest threshold $t = 255$ and decreasing the threshold until $t = 0$ is reached. A MSER is then defined as a region for which the region area A does not change significantly over a range of thresholds Δ.

More formally, the variation q for a region R at a specific threshold t is defined as [38]

$$q_t = (A_{R_{t+\Delta}} - A_{R_{t-\Delta}})/A_{R_t}. \tag{3.50}$$

The region is then considered maximally stable if it satisfies $q_t > q_{t-1}$ and $q_t < q_{t+1}$, i.e. if it is a local minimum [38].

For more details refer to the original paper [38].

3.6 Field of view

Field of view (FOV) is defined as the set of angles (horizontal and vertical) that determine to what extent a scene (part of the 3D world) is visible from a particular point and in a particular direction [46]. Humans for instance have a binocular field of view of around 140 degrees. In many cases, the vertical field of view α_v differs from the horizontal field of view α_h.

Figure 3.5 shows the horizontal field of view α_h is related to the camera sensor width W and focal length f [5] [6]. From the figure it can be noted that

$$\tan\frac{\alpha_h}{2} = \frac{W}{2f}. \tag{3.51}$$

From this it follows that α_h can be calculated as

$$\alpha_h = 2\arctan\frac{W}{2f}. \tag{3.52}$$

The corresponding equation for the vertical field of view α_v is

$$\alpha_v = 2\arctan\frac{H}{2f}, \tag{3.53}$$

where H is the height of the camera sensor.

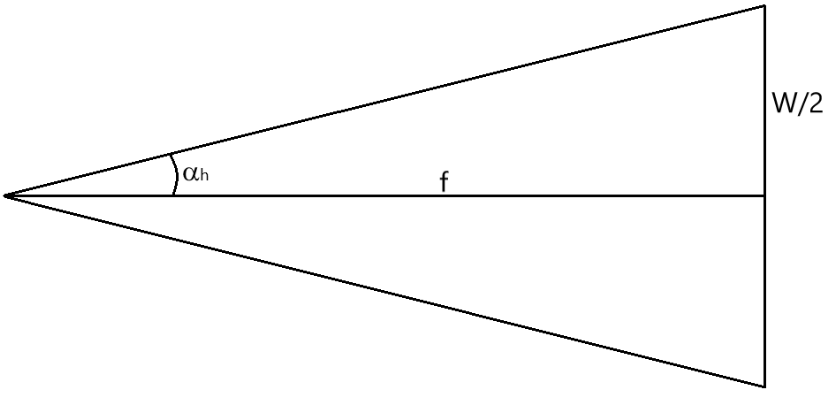


Figure 3.5: Horizontal field of view α_h. f is the focal length and W is the width of the camera sensor.

3.6.1 Depth from Field of View

The field of view is an important component in the depth estimation method proposed by Salih and Malik in [24]. Along with the vertical and horizontal field of views α_v and α_h, the method also requires knowledge of the camera height h and the camera rotation angle relative to the ground θ as defined in equation (3.54), where θ_x is the camera X-axis rotation (or pitch). Using these inputs and a couple of trigonometric relationships it is shown that the 3D coordinates $\tilde{\mathbf{X}} = (X, Y, Z)$ on the ground plane can be recovered. It is also possible to recover the heights and widths of objects in the scene but this is of less interest for the application. Note however that the method does not take into account occlusions. In other words, objects on the ground plane blocking the view (for instance vertical objects such as signs) are treated as being part of the plane and are assigned the same depth as if they would be part of the plane. Therefore, it is important to estimate the depth for objects located on the plane separately.

$$\theta = 90^\circ - \theta_x \tag{3.54}$$

The basic geometry for the camera setup is shown in 3.6. In the Figure it is shown that the part of the scene viewed by the camera is defined by the vertical and horizontal fields of view α_v and α_h.

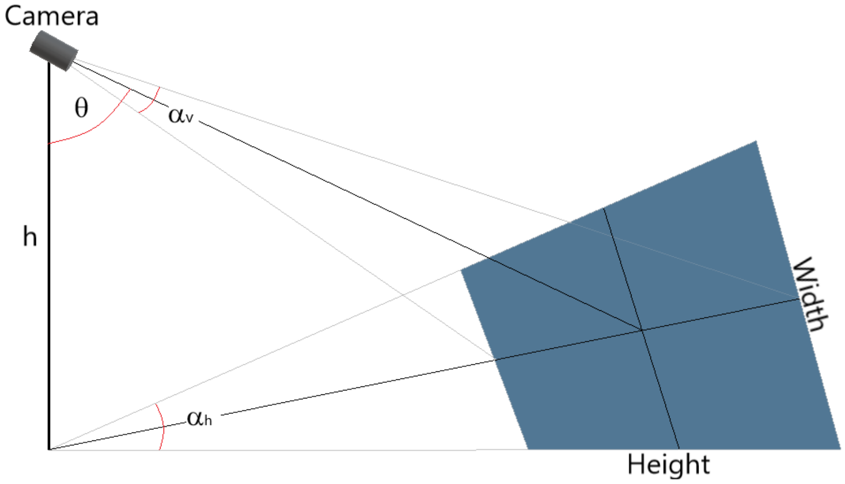


Figure 3.6: The basic camera setup. Here the blue region represents the part of the scene that is visible in the camera.

Now, consider an object located at $\tilde{\mathbf{x}} = (i, j)^T$ in the image and its' corresponding 3D coordinate $\tilde{\mathbf{X}} = (X, Y, Z)^T$. The trigonometric relationships for this are shown in Figure 3.7.

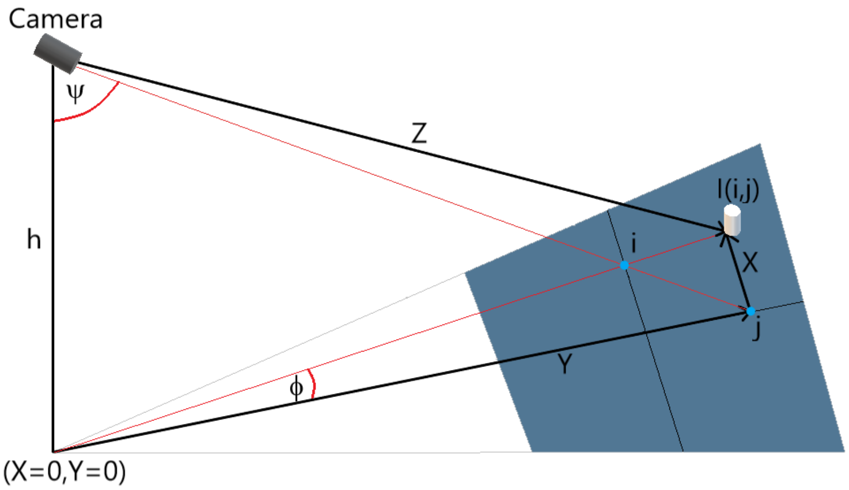


Figure 3.7: Trigonometric relationships between an object located at $\tilde{\mathbf{x}} = (i, j)^T$ in the image and the corresponding 3D coordinate $\tilde{\mathbf{X}} = (X, Y, Z)^T$.

The vertical angle ψ and the rotation angle ϕ are defined in equations (3.55) and (3.56) respectively. Here H is the image height and W is the image width. The angular steps (due to one pixel displacement) in the vertical and horizontal directions are given by $\frac{\alpha_v}{H}$ and $\frac{\alpha_h}{W}$ respectively. Assuming that the point $(X, Y)^T = (0, 0)^T$ is located directly beneath the camera, the 3D coordinate $(X, Y, Z)^T$ can then be computed using equations (3.57)-(3.59). For more details refer to the original paper.

$$\psi = \theta + \left(\frac{H}{2} - j\right)\left(\frac{\alpha_v}{H}\right) \tag{3.55}$$

$$\phi = \left(i - \frac{W}{2}\right)\left(\frac{\alpha_h}{W}\right) \tag{3.56}$$

$$Y = h \tan \psi \tag{3.57}$$

$$X = Y \tan \phi \tag{3.58}$$

$$Z = \sqrt{h^2 + Y^2 + X^2} \tag{3.59}$$

3.7 Luminance

Luminance is defined as the luminous intensity I_v per unit surface area A [47]. This can be written mathematically as shown in equation (3.60). Luminance is measured in candela per square meters or cd/m^2.

$$L_v = \frac{dI_v}{dA} \tag{3.60}$$

A camera can be viewed as a non-ideal lux meter (device for measuring luminance) since it maps luminance values into pixel intensities [48]. An image captured by a typical camera only covers a small dynamic range (range of luminances, given as a ratio of the highest luminance value with the lowest luminance value), for instance 100 : 1. For a real-life scene however, the dynamic range can be as high as $10^6 : 1$. A method to significantly increase the dynamic range of a luminance measurement with a camera is to create High Dynamic Range (HDR) images.

The basic idea of HDR imaging is to photograph a scene at several different exposure times (refer to Section 5.1 for an explanation of exposure times) [48]. Longer exposure times give brighter images and vice versa. The pixel values (at specific exposure times) however still need to be mapped to actual luminance values. This is done by measuring the camera calibration curve, for instance using an integrating sphere or a Munsell ColorChecker Chart [5].

For the specific cameras used in the product, the calibration curve can be approximated by four linear regions at different exposure times. At a specific pixel coordinate (x, y), the luminance $L(x, y)$ is computed as shown in equation (3.61). Here $I(x, y)$ is the pixel value in exposure image number i and k_i and m_i are the parameters defining the linear approximation.

$$L(x, y) = k_i \cdot I(x, y) + m_i \tag{3.61}$$

The complete luminance computation algorithm is listed in algorithm 1. Here the order of images is from the brightest (with longest exposure time) to the darkest (with shortest exposure time). For each pixel (x, y) the algorithm selects the brightest possible image I_i that fulfills $I_i(x, y) \leq \tau$ where $\tau = 255 \cdot t$. Here t is a scale factor, usually 0.9. In other words, the algorithm selects the pixel from the brightest image possible for which the pixel value is not saturated or close to saturated. Based on the chosen image I_i, the luminance $L(x, y)$ is calculated using equation (3.61). Note also that the boolean *foundExp* is set to true to prevent the luminance calculation to be repeated for darker images.

Algorithm 1 Grouping of depth regions into depth region sets

```
function LUMINANCEIMAGE(InputImages)
    Initialize L as an image of size $W \times H$ with values 0.0
    for $y = 0$ to $y = H - 1$ do
        for $x = 0$ to $x = W - 1$ do
            foundExp=false
            for $i = 0$ to $length(InputImages)$ do
                if foundExp=false && $InputImages[i] \leq \tau$ then
                    $L(x, y) = k_i \cdot InputImages[i](x, y) + m_i$
                    foundExp=true
                end if
            end for
        end for
    end for
end function
```

The result is a matrix/image containing the luminance value, which cannot be easily visualized. However, the pixel values chosen from the different exposures can be visualized as shown in Figure 3.8.

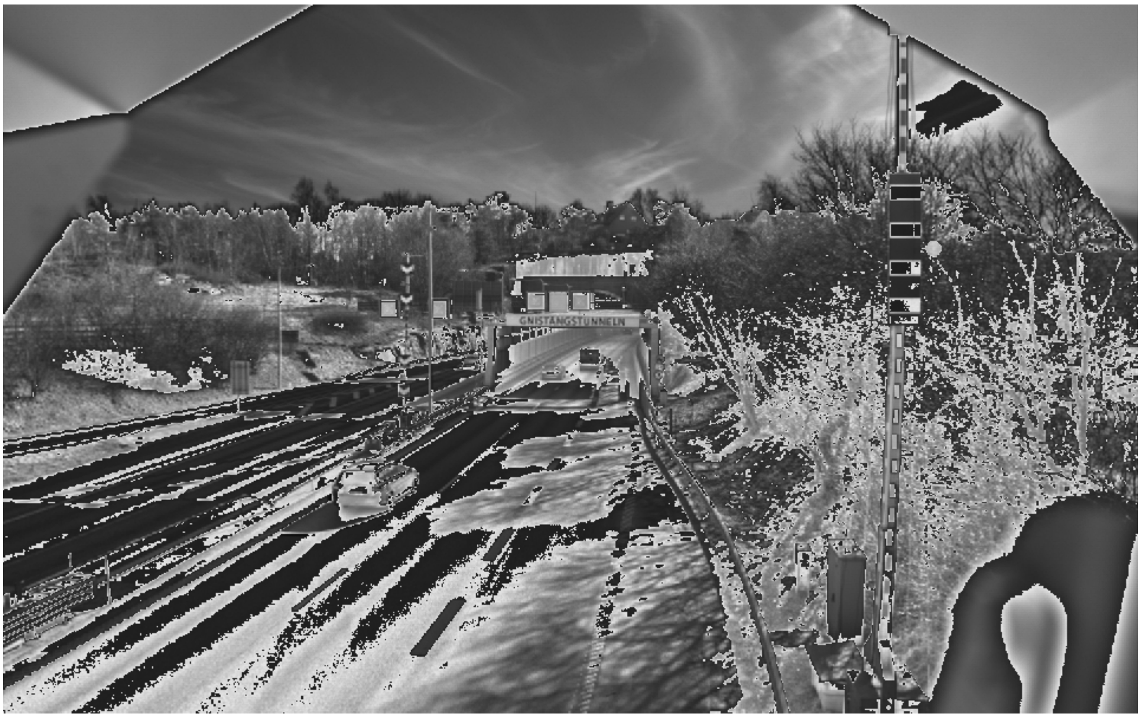


Figure 3.8: Image showing the pixel values chosen by algorithm 1.

3.7.1 Equivalent Veiling Luminance

According to the CIE88:2004 guide, the luminance L_{th} in the threshold zone (start of the tunnel) should be adjusted in such a way that the driver (at the stopping distance away from the tunnel) is able to perceive any obstacles located at the tunnel entrance [2]. The visibility of an object is determined by its' perceived contrast $C_{perceived}$, defined according to the guide as

$$C_{perceived} = \frac{L_{o,p} - L_{r,p}}{L_{r,p}}, \tag{3.62}$$

where $L_{o,p}$ is the perceived luminance for the object and $L_{r,p}$ is the perceived luminance of the road surface near the obejct [2]. The definitions of $L_{o,p}$ and $L_{r,p}$ are given in equations (3.63) and (3.64). Here $L_{o,intrinsic}$ is the intrinsic luminance for the object, $L_{r,intrinsic}$ is the intrinsic luminance for road surface, L_{atm} is the luminance contribution from the atmosphere, L_{ws} is the luminance contribution of the windscreen, τ_{atm} is a factor representing atmospheric losses, τ_{ws} is a factor representing the luminance loss due to the windscreen and L_{seq} is the equivalent veiling luminance. From the equations it can be noted that the intrinsic luminances are affected both by atmospheric transmission losses and losses due to the windscreen and that the atmospheric luminance contribution is affected by transmission through the windscreen.

$$L_{o,p} = \tau_{ws} \cdot \tau_{atm} \cdot L_{o,intrinsic} + \tau_{ws} \cdot L_{atm} + L_{ws} + L_{seq} \tag{3.63}$$

$$L_{r,p} = \tau_{ws} \cdot \tau_{atm} \cdot L_{r,intrinsic} + \tau_{ws} \cdot L_{atm} + L_{ws} + L_{seq} \tag{3.64}$$

The equivalent veiling luminance L_{seq} is the total estimated luminance contribution from outside the 2° view cone of sharp vision [2]. Light sources outside the 2° cone are partially scattered in the eye, thus disturbing the perception of L_o and L_r. Although there exist special devices for measuring L_{seq}, it is also possible to calculate L_{seq} using Holiday-Stiles formula [2]. The idea is to use a polar diagram consisting of $12 \times 9 = 108$ zones as shown in Figure 3.9. Each zone is scaled in such a way that for a given average luminance it produces equal amounts of stray light in the eye. Or stated another way: Zones further away from the centre need to be larger in order to produce to produce the same amount of stray light compared to zones closer to the centre. The specific radial angles defining the different rings in the diagram are listed in Table 3.1. Given the radial angle ϕ, the vertical field of view α_v and the image height H, the radius r of a ring is computed using equations (3.65) and (3.66).

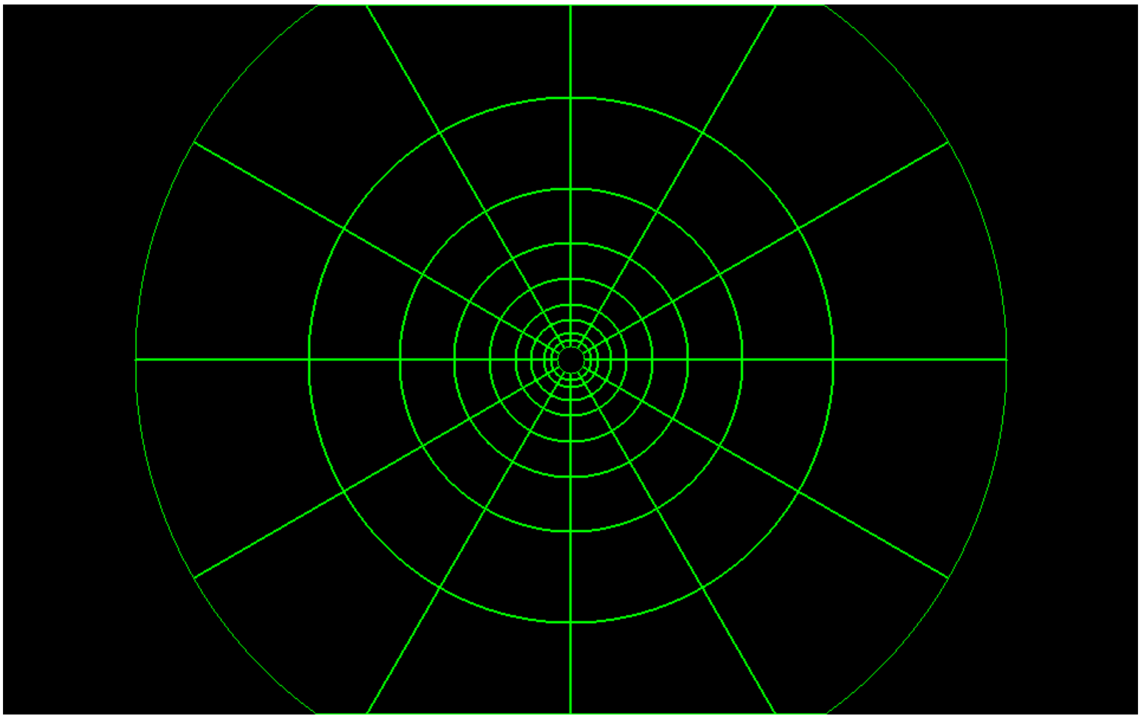


Figure 3.9: Diagram for L_{seq} computation. Each zone in the diagram produces an equal amount of stray light for a given average luminance.

Ring	Centre	1	2	3	4	5	6	7	8	9
Angle of opening	2.0°	3.0°	4.0°	5.8°	8.0°	11.6°	16.6°	24.0°	36.0°	56.8°

Table 3.1: Radial angles in the L_{seq} diagram.

$$x = \frac{H/2}{\tan(\alpha_v/2)} \tag{3.65}$$

$$r = \tan(\phi/2) \cdot x. \tag{3.66}$$

The luminance contribution for a zone is computed as

$$L_{ije} = (\tau_{ws} \cdot L_{ij}) + L_{ws}, \tag{3.67}$$

where L_{ij} is the average luminance in that zone. The total luminance or L_{seq} is then calculated by summing the luminances for all the zones as follows:

$$L_{seq} = 5.1 \cdot 10^{-4} \sum_{i,j} L_{ije}. \tag{3.68}$$

Note that the 2° circle in the middle is not included in the L_{seq} computation [2]. Once L_{seq} has been computed, the threshold luminance L_{th} is computed according to [2] as

$$L_{th} = \frac{L_m}{\frac{1}{C_m}\left(\frac{\rho}{\pi q_c} - 1\right) - 1}, \tag{3.69}$$

where

$$L_m = \frac{\tau_{ws} \cdot L_{atm} + L_{ws} + L_{seq}}{\tau_{ws} \cdot \tau_{atm}}. \tag{3.70}$$

Here C_m is the desired minimum perceived contrast, usually around 28%. ρ is an reflection factor for a reference object (obstacle) in the tunnel. It is usually assumed that the reference object is a cube with a $0.2m$ long side, yielding diffuse reflections and with $\rho = 0.2$. Finally, $q_c = L/E_v$ is the contrast revealing coefficient, defined as the ratio of the road surface luminance L and the vertical illuminance E_v. The vertical illuminance is defined according to [2] as the illuminance "at a particular location at a height of $0.1m$ above road surface, in a plane facing and at right angles to the direction of oncoming traffic".

4
Implementation

This chapter describes the overall pipeline of the implementation, from input images to output L_{seq} value for the resulting warped view. Essentially, the implementation consists of four main parts as shown Figure 4.1. First, the depth map for the scene is estimated. This is the most complicated part of the implementation and is summarized below and described in detail in Sections 4.1 to 4.10. Once the depth map has been created, the original images are warped to a new view using the principle of Depth Image Based Rendering (DIBR). First the camera geometry for the scene is defined as described in Section 4.11. Then the image is warped using the estimated depth map and the camera poses as described in Section 4.12. The warped view contains holes and can thus be inpainted as described in Section 4.12.1 but this an optional step, which is useful for visualization but does not yield improved L_{seq} results (refer to Section 6.3 for the analysis). In the final step, the L_{seq} value is computed. This is done by first creating the L_{seq} diagram in the warped view as described in section 4.13 and then computing the L_{seq} value in the warped view, which is described in Section 4.14.

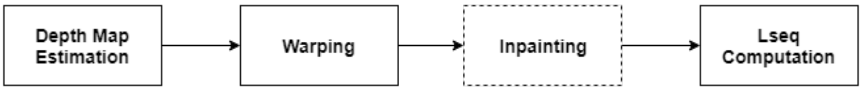


Figure 4.1: The four main steps in the implementation. Note that the inpainting step is optional.

As mentioned above, the depth map estimation is the most complicated of the four main parts in the implementation. Once the input images have been loaded (Section 4.1), two types of features are extracted from the images: Line segments using the Line Segment Detector (LSD) method (Section 4.2) and Maximally Stable Extremal Regions (MSERs), refer to Section 4.4.

The implementation differs slightly depending on whether or not the scene contains a tunnel. For the product/application only tunnel scenes are relevant but the L_{seq} error reduction has also been evaluated on scenes without tunnels. If the scene has a tunnel, the tunnel opening is first detected using the bounding boxes for the MSER regions as described in 4.4.1. If instead the scene is without a tunnel, the vanishing point of the scene is detected from the detected line segments as described in Section 4.3. Either the tunnel opening or the vanishing point is used to divide the scene in the vertical direction: Above and below the tunnel opening or above and below the end of the road. This is then used as basis for detecting the sky and road parts in the image as described in Section 4.5. The sky and road regions are detected in order to reduce the number of false vertical region detections. It could after all be the case that a MSER is classified as vertical despite being part of the road plane or the sky, which is undesirable.

The next step in the depth map estimation is region classification using the detected line segments as described in Section 4.6. Once a region has been classified as vertical, the shape of the region is refined using the Grabcut Foreground/Background segmentation algorithm. This is described in Section 4.7. The tunnel outline and tunnel walls are detected separately using the detected tunnel opening as basis (section 4.8). The final step before the actual depth map creation is to potentially group together detected depth regions as described in Section 4.9.

Once all the depth regions in the image has been detected the remaining part of the depth map estimation is to create the depth map. First a depth map based on the field of view method proposed by Salih and Malik in [24] is created as described in Section 4.10.1. The method gives the depth for the road plane. Then the tunnel walls are added to the depth map as described in Section 4.10.2. Finally, the other detected depth regions are added to the depth map (Section 4.10.3).

The project was implemented using the OpenCV C++ library. For the sake of brevity not all of the OpenCV methods/algorithms used in the implementation have been described in the previous chapter. This includes for instance contour finding, bounding box extraction and polygon approximations of contours. For more details about various OpenCV functions, methods and classes refer to the OpenCV documentation [49].

The images used in this chapter are from the Gnistängstunneln scene which was the scene for the research study mentioned in the introduction [3]. It represents the ideal type of scene for the application.

4.1 Input

Using the HDR mode, the camera takes series of four monochrome (grayscale) photos of size $W \times H = 960 \times 600$, where each photo has a different exposure time. Therefore, the input to the system are four photos rather than just one photo. This is taken advantage of in the implementation by extracting different types of information from the images with the different exposure times. As described in Section 5.1, a shorter exposure time gives a darker image. An example of a series of images with differing exposures is shown in Figures 4.2 to 4.5.

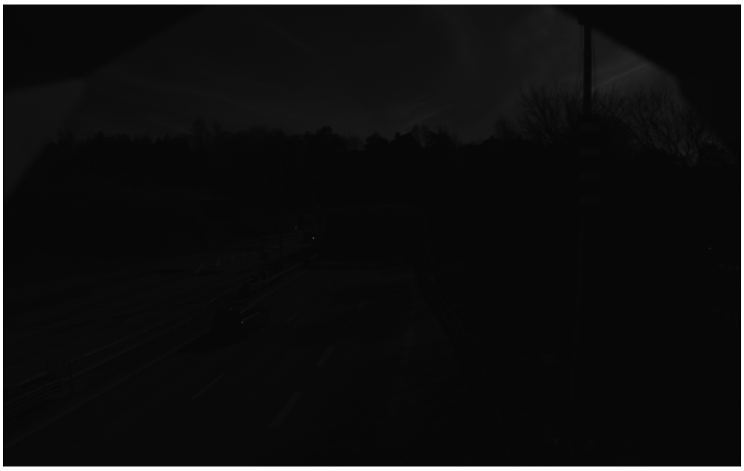


Figure 4.2: The image with the shortest exposure time.

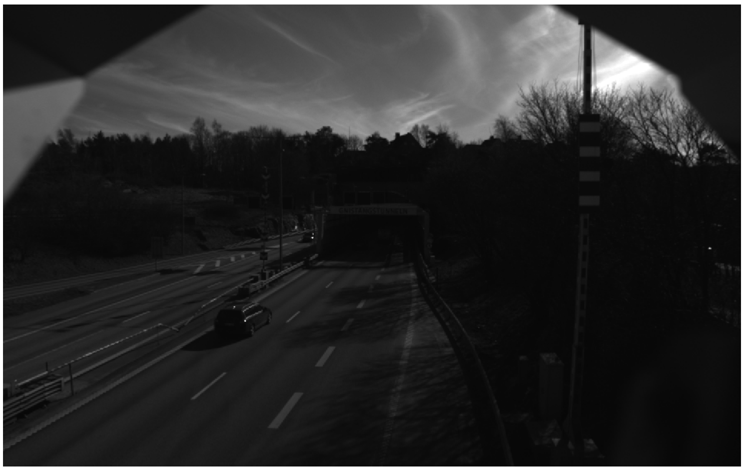


Figure 4.3: The image with the next shortest exposure time.

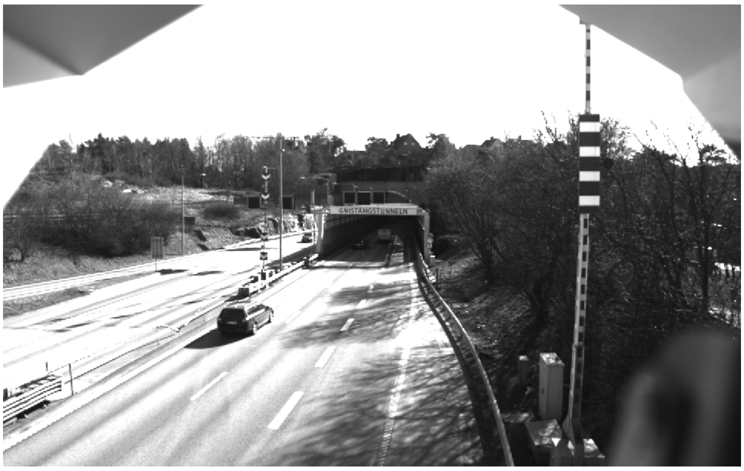


Figure 4.4: The image with the next longest exposure time.

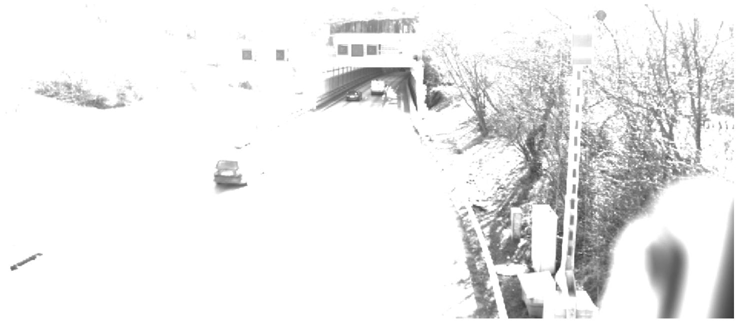
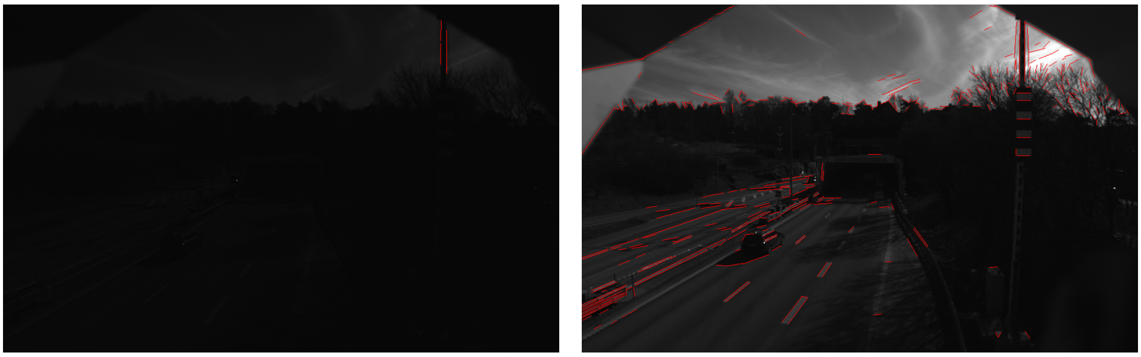


Figure 4.5: The image with the longest exposure time.

4.2 Line segment detection

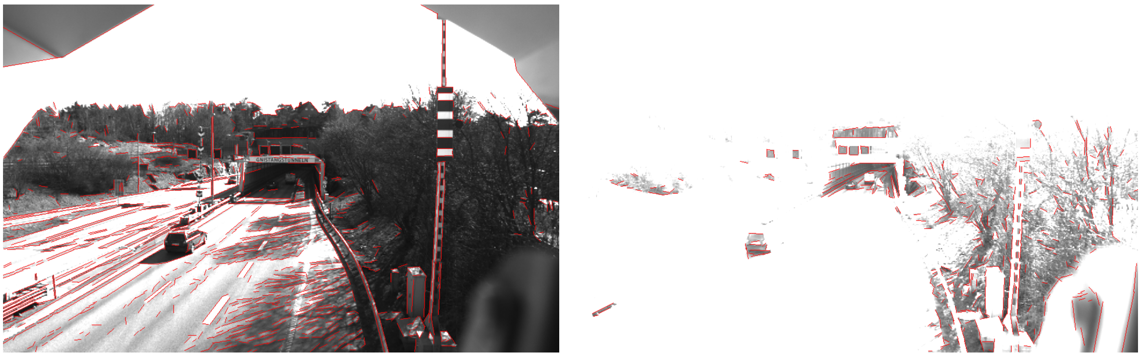
Once the four images have been loaded, the next step in the implementation is to detect the line segments in the images. This is done using the OpenCV implementation of the LSD Line Segment Detector [50] (refer to Section 3.3.1 for a theoretical background). The only parameter that can be controlled in the OpenCV implementation is the level of refinement: either there is no refinement, standard refinement or advanced refinement. A lower level of refinement gives more line segments but the line segments are in general less reliable (false detections become more common). Consequently, advanced line segment refinement is used in the implementation. Figures 4.6 and 4.7 show the line segments detected for the different images.



(a) Line segments for darkest exposure image.

(b) Line segments for next darkest exposure image.

Figure 4.6: Line segments detected for the two darkest images.



(a) Line segments for the next brighest exposure image.

(b) Line segments for the brightest exposure image.

Figure 4.7: Line segments detected for the two brighest images.

The detected line segments are used in several parts of the implementation including vanishing point estimation and region classification (see Section 4.6).

4.3 Vanishing point estimation

Vanishing points are used in the implementation for two different purposes depending on whether or not the road scene contains a tunnel. For a scene with a tunnel, a vanishing point is used to estimate the depth of the tunnel walls as described in Section 4.8.2. For a scene without a tunnel on the other hand, a vanishing point is estimated to roughly determine the y-coordinate at which the road ends. This is later used to detect the road and sky regions in the image as described in Section 4.5.

The implementation uses the source code [51] for the MSAC vanishing point estimation method proposed by Nieto and Saldago in [33]. Without tweaking the core of the source code there are only two parameters that can be changed: The maximum number of line segments used for the vanishing point estimation and the mode for the vanishing point estimation, either *LS* (Least Squares, which is a modified version of the approach proposed by Košecká and Zhang in [42]) or *NIETO* (the approach proposed by Nieto and Saldao in [33]). In general it can be noted that a relatively large number of maximum line segments together with the *LS* method tend to yield the most reliable results. However, the result is also highly dependent on the line segments used as input for this method. For instance when estimating the vanishing point for a road scene viewed roughly in the direction of the traffic it is a good idea to remove the vertical or nearly vertical line segments and the horizontal or nearly horizontal line segments since these are likely to be outliers.

Figure 4.8 shows the estimated vanishing point for the Gnistängstunneln scene. The line segments from all of the exposure images have been combined into a single container but the vertical or nearly vertical and the horizontal or nearly horizontal lines have been removed. Note that this is the approach used for estimating the vanishing point for scenes without tunnels as mentioned above.

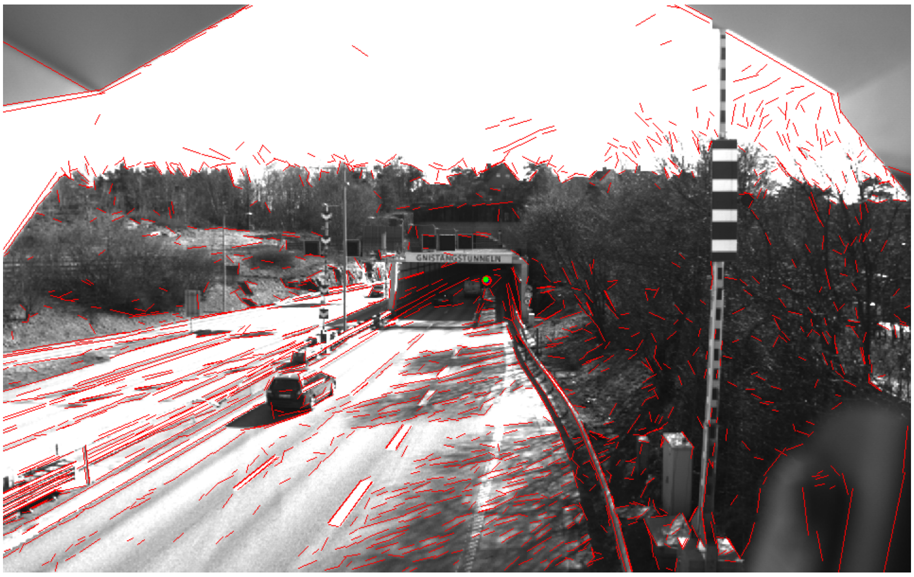


Figure 4.8: Estimated vanishing point. The line segments from the different exposure images were first combined into a single container. Then the vertical or nearly vertical line segments and the horizontal or nearly horizontal line segments were removed. The estimation is based on the remaining line segments.

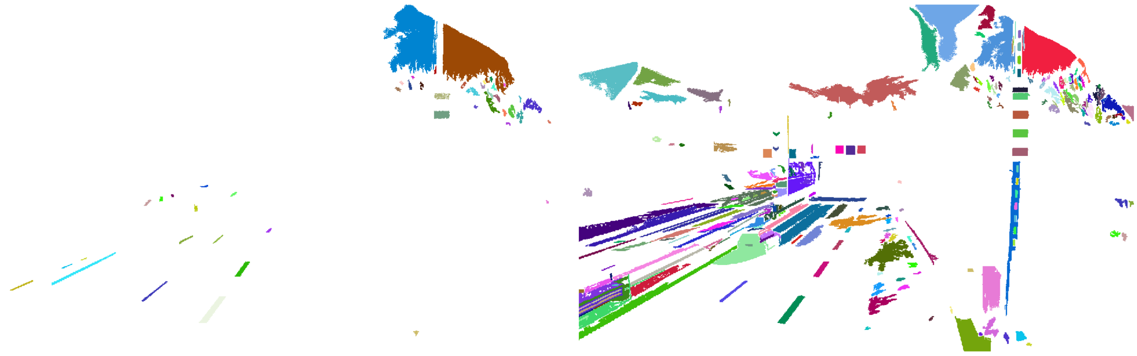
4.4 Maximally Stable Extremal Regions (MSER)

The next step in the implementation is the detection of Maximally Stable Extremal Regions (MSER) in the image (refer to Section 3.5 for a theoretical background). This is done for all four exposure images since more regions can be detected this way. In the OpenCV implementation of MSER there is a total of four parameters that need to be set to extract regions in a grayscale image [52]:

1. Δ is the number of gray levels for which a region needs to be stable to be considered a maximally stable region (see Section 3.5 for the definition). A higher Δ gives a lower number of regions.
2. The minimum area, A_{min}. MSER regions with areas smaller than A_{min} are removed.
3. The maximum area, A_{max}. MSER regions with areas larger than A_{max} are removed.
4. Maximum variation v_{max}. MSER regions with variation larger than v_{max} are removed. In the OpenCV MSER implementation the variation is defined as $v_t = (A_{R_t} - A_{R_{t-\Delta}})/A_{R_{t-\Delta}}$ where t is the threshold, R is a region and A is the region area. Note that this definition of variation differs from the one defined in Section 3.5. A lower v_{max} gives a lower number of regions.

The major weakness of using MSER for region detection is the need to set these four parameters. It becomes especially apparent when there are four images to adjust the parameters for. There are however a few simple guidelines. For the darkest image there are barely any regions detected so the settings should be as low as possible, in other words $\Delta = 1$, a low A_{min}, a high A_{max} and v_{max} close to 1. As the images get brighter it seems to be a good idea to increase Δ and to decrease v_{max}. If the

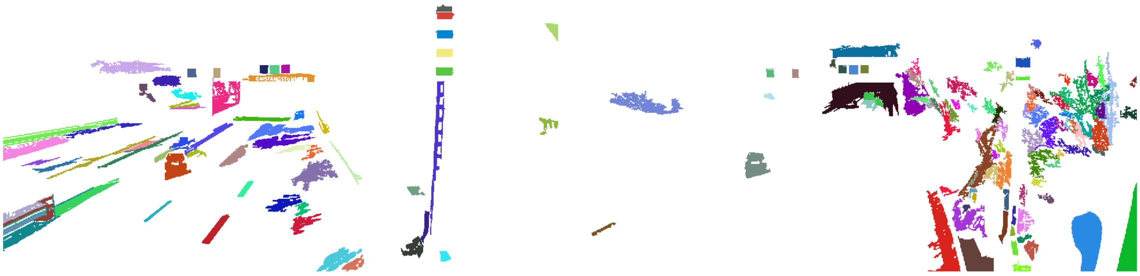
scene has many objects of different sizes it is a good idea to have a high range of A_{min} to A_{max} but this can on the other hand lead to a larger number of irrelevant regions being detected. In the end, the specific settings are quite dependent on the scene geometry and the scene brightness.
The detected MSER for the Gnistängstunneln scene are shown in Figures 4.9 and 4.10.



(a) MSER for the darkest exposure image. The parameters are $\Delta = 1$, $A_{min} = 20$, $A_{max} = 10000$ and $v_{max} = 0.99$.

(b) MSER for the next darkest exposure image. The parameters are $\Delta = 1$, $A_{min} = 30$, $A_{max} = 5000$ and $v_{max} = 0.4$.

Figure 4.9: MSER for the two darkest exposure images shown on a white background. Note that only the outer MSER are shown.



(a) MSER for the next brightest exposure image. The parameters are $\Delta = 5$, $A_{min} = 200$, $A_{max} = 2000$ and $v_{max} = 0.25$.

(b) MSER for the brightest exposure image. The parameters are $\Delta = 5$, $A_{min} = 100$, $A_{max} = 4000$ and $v_{max} = 0.6$.

Figure 4.10: MSER for the two brightest exposure images shown on a white background. Note that only the outer MSER are shown.

The MSER can be approximated using several shapes such as bounding rectangles, rotated rectangles, bounding circles and polygons. For the purpose of depth estimation the MSER are approximated by bounding rectangles. Figure 4.11 shows the bounding rectangles for the MSER for the next darkest exposure image.

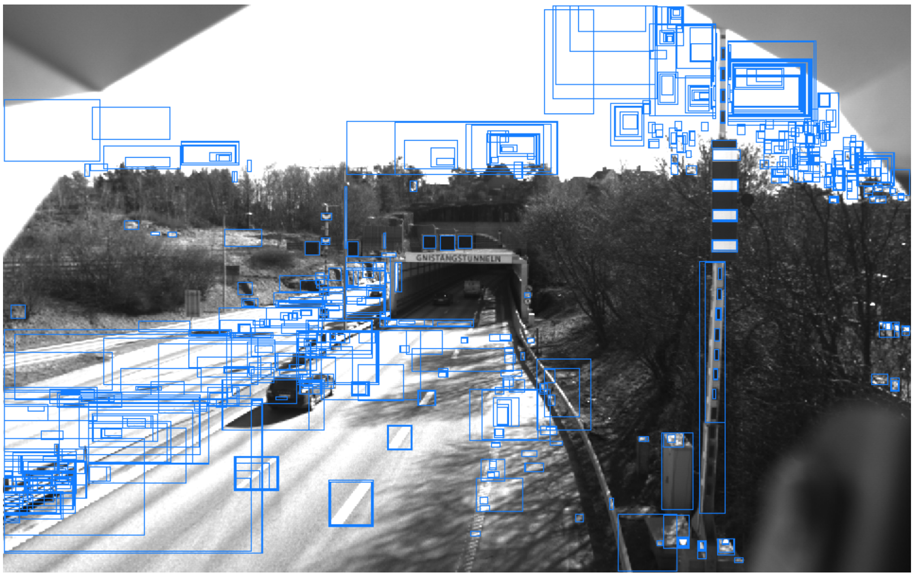


Figure 4.11: Bounding rectangle approximations of the MSER in the next darkest exposure image.

It can be noted that Figure 4.11 contains many inner MSER regions. These inner regions are redundant for the purpose of depth estimation and are therefore removed. Figure 4.12 shows the remaining bounding rectangles for the MSER for the next darkest exposure image after removal of the inner bounding rectangles.

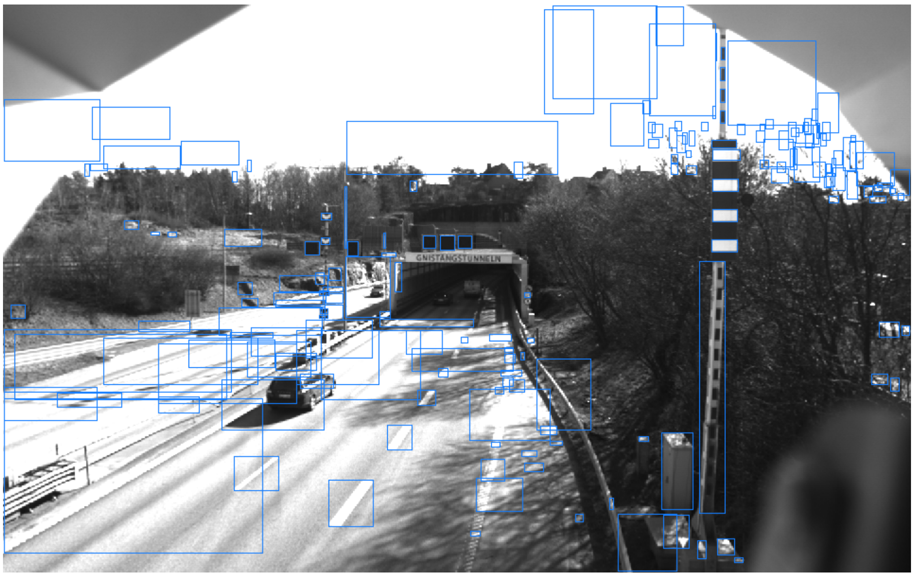


Figure 4.12: Bounding rectangle approximations of the MSER in the next darkest exposure image after removal of the inner bounding rectangles.

4.4.1 Finding the tunnel opening

As can be seen in Figure 4.10, the MSER for the tunnel opening can be obtained from the brightest exposure image. This makes sense considering that the brightest exposure image is for the most part oversaturated (white) with the exception of dark regions such as the tunnel opening (refer back to Figure 4.5). As shown in Figure 4.13, the tunnel opening is well approximated by a bounding rectangle if the tunnel opening is rectangular. When the shape of the tunnel opening is not rectangular the shape estimation becomes more complicated. Depending on the quality of the detected MSERs it could be possible to estimate the shape of the tunnel using a convex polygon. However, the implementation currently only contains the estimation of the tunnel opening shape for rectangular tunnel openings.

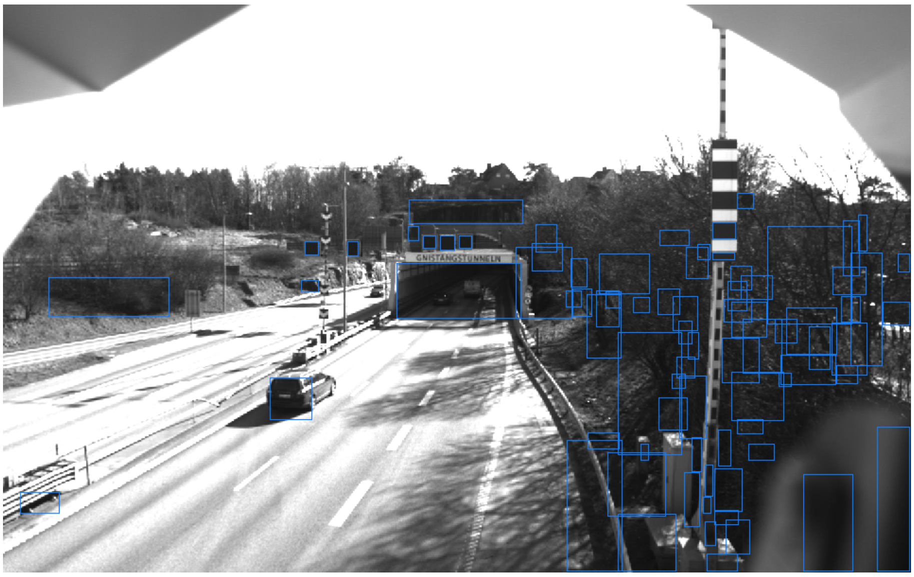


Figure 4.13: Bounding rectangle approximations of the MSER in the brightest exposure image after removal of the inner bounding rectangles.

Although the tunnel opening has been approximated by a shape such as a bounding rectangle it is not yet clear which of the bounding rectangles belongs to the tunnel opening. However, it is known that the camera is centered on the tunnel opening and therefore a reasonable assumption is that the tunnel opening contains the midpoint in the image $\tilde{\mathbf{x}}_{mid} = (W/2, H/2)$ where W is the image width and H is the image height. The correct bounding rectangle can then easily be found by going through the list of all bounding rectangles and finding the bounding rectangle containing $\tilde{\mathbf{x}}_{mid}$ (note that this requires the inner bounding rectangles to be filtered out as described in Section 4.4).

Finding the tunnel opening is important for several reasons. Firstly, it is useful for depth estimation since the shape approximation of the tunnel can be used together with a detected vanishing point inside the tunnel to find the tunnel walls as described in Section 4.8.2 and by knowing the tunnel opening location and shape it is also possible to find the outer outline of the tunnel as described in Section 4.8.1. Secondly, it provides another important hint about the geometry of the scene since it can be assumed that the road is located below the tunnel opening and that the sky is located above the tunnel opening. This is useful for detecting the road and sky regions in the scene as described in Section 4.5. Finally, the line segments contained inside the tunnel can be excluded from the rest of the depth estimation pipeline.

4.5 Road and Sky Detection

This section describes how to find the road and sky regions in the scene. The purpose of this step is to reduce the number of false detections of potential vertical depth regions in the image by removing the MSER from the sky and removing line segments from the road. After all, the road itself can be approximated as a single plane and does not contain any vertical objects other than vehicles (which as mentioned in Section 1.1 are compensated for in the application anyway) and the sky is very far away and can be thought of as belonging to the plane at infinity as described in Section 3.1.4.

The process of finding the road and sky regions in the image can be divided into three steps: Segmentation using Hierarchical Feature Selection (HFS), Thresholding using the Triangle threshold and matching of segmented regions with the thresholded regions.

4.5.1 Hierarchical Feature Selection (HFS) Segmentation

The Hierarchical Feature Selection (HFS) method is used for a general segmentation of the image. Just as the other segmentation methods used it is part of the OpenCV library [53]. As described in Section 3.4.3, the segmentation method has two steps: First the image is divided into a number of superpixels using Simple Iterative Linear Clustering (SLIC) and then two iterations of Efficient Graph Based segmentation (EGB) are used.

The implementation of HFS requires setting 7 different parameters:

1. *slicSpixelSize*. The size of the superpixels when initializing SLIC is approximately $slicSpixelSize \times slicSpixelSize$.
2. *numSlicIter*, the number of iterations for SLIC.
3. *spatialWeight* is the third parameter required for SLIC. A higher value results in a segmentation with more local consistency.
4. *segEgbThresholdI* is used in the second stage of the algorithm (first EGB stage). It is a threshold used on the edge weights when merging adjacent nodes. A higher value tends to give more regions.
5. *minRegionSizeI*, is another parameter used in the first EGB stage. Regions with less than *minRegionSizeI* are merged into adjacent regions.
6. *segEgbThresholdII* serves the same purpose as *segEgbThresholdI* but for the second EGB stage.
7. *minRegionSizeII* serves the same purpose as *minRegionSizeI* but for the second EGB stage.

Given that HFS requires 7 parameters, it is hard to find any simple rules describing what values might be useful.

Figure 4.14 shows the HFS segmentation of the Gnistängstunneln scene for the next brightest exposure image. As can be seen from the figure, the road and the sky regions are segmented quite accurately.

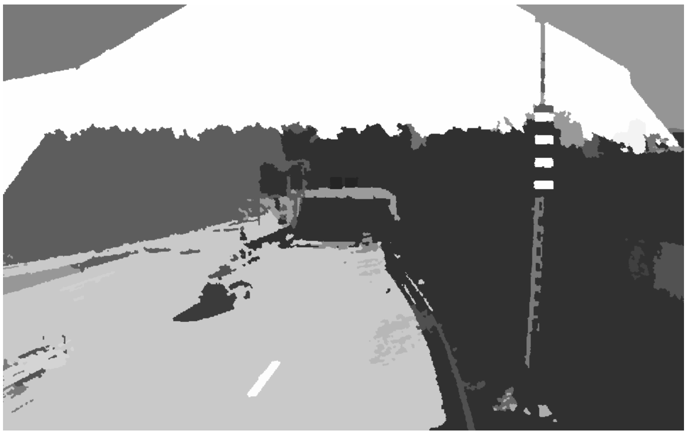


Figure 4.14: HFS segmentation of the Gnistängtunneln scene for the next brightest exposure image. The parameters used are: $slicSpixelSize = 20$, $numSlicIter = 5$, $spatialWeight = 1.0$, $segEgbThresholdI = 0.08$, $minRegionSizeI = 250$, $segEgbThresholdII = 0.28$ and $minRegionSizeII = 400$.

Once the image has been segmented, masks are created for each segmented region and the segmented regions are sorted by the number white pixels (value 255) contained in the region, from largest number of pixels to the smallest number of pixels. This is useful later when extracting the road and sky regions from the image as described in Section 4.5.3.

4.5.2 Thresholding

One of the interesting things in either the darkest or the next darkest exposure image is that both the road lines and the sky clearly stands out from the other parts in the image. Either the darkest or the next darkest exposure image is therefore thresholded. The threshold is calculated automatically using the Triangle method described in Section 3.4.1. The result is shown in Figure 4.15.

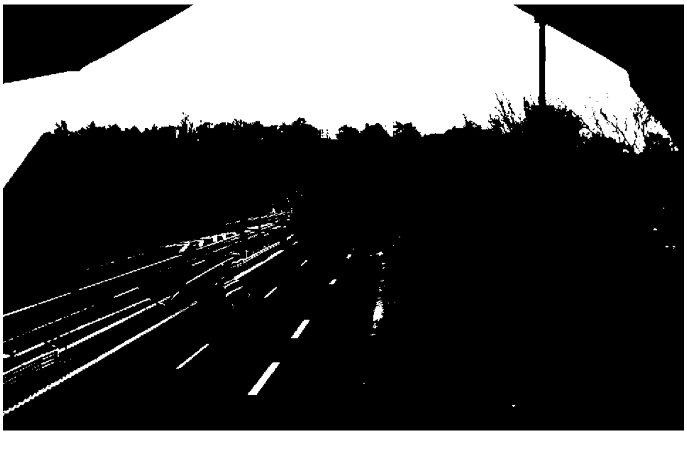
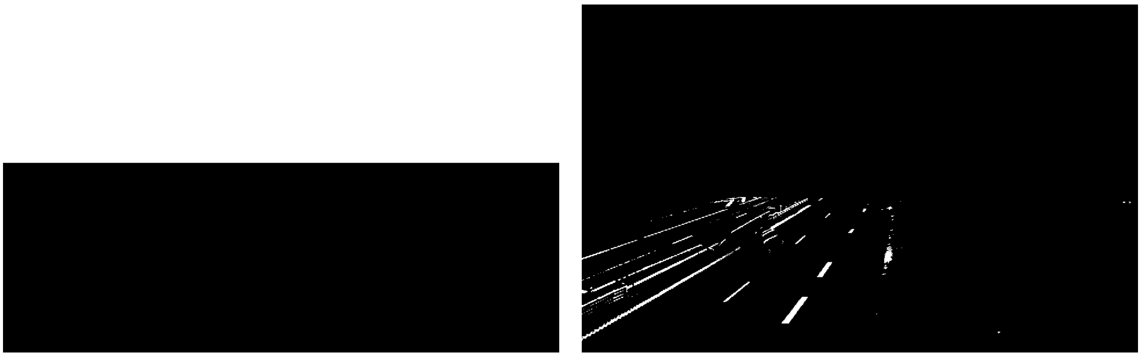


Figure 4.15: Thresholding of the next darkest exposure image. The threshold is computed using the Triangle method.

4.5.3 Finding the road and sky regions

Either the vanishing point or the tunnel bounding rectangle can be used to divide the scene into two parts based on the y-coordinates. The sky is assumed to be located above the vanishing point or the tunnel bounding rectangle while the road is assumed to be located below the vanishing point or the tunnel bounding rectangle. Based on this assumption two masks are created. The mask used for the sky detection I_{upper} consists of a black background with a white rectangle drawn above either the vanishing point y-coordinate or the upper bounding rectangle y-coordinate. The mask for the road detection I_{lower} on the other hand contains only the thresholded parts below the vanishing point or below the tunnel bounding rectangle. Both of these masks are shown in Figure 4.16.



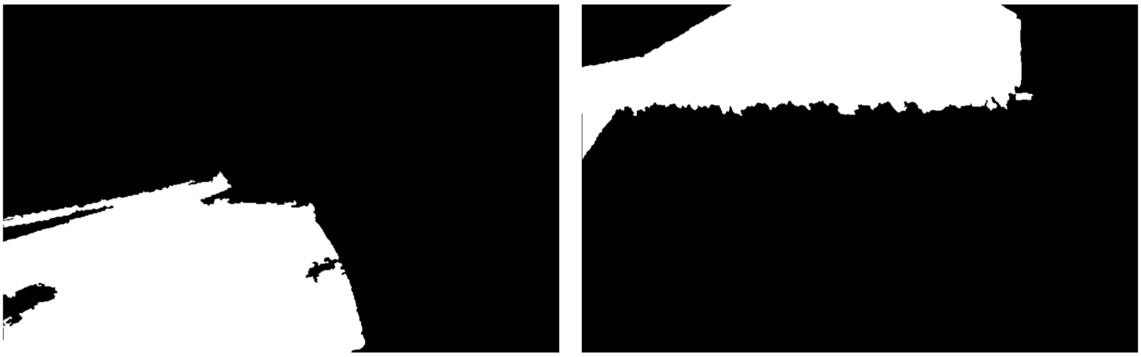
(a) Mask used for the sky region detection. It consists of a black image with a white rectangle drawn above the vanishing point y-coordinate or the upper tunnel bounding rectangle y-coordinate.

(b) Mask used for road detection. It was created by drawing a black rectangle in the thresholded image. The black rectangle is located above either the vanishing point y-coordinate or the lower tunnel bounding rectangle y-coordinate.

Figure 4.16: (a) shows the mask used for the detection of the sky region and (b) shows the mask used for the detection of the road region.

After the creation of the two masks, the next and final step is to find the HFS region masks with the highest numbers of overlapping pixels. This is done using the & (AND) operation, which results in masks where only overlapping (white) pixels remain. To detect the sky region the upper mask I_{upper} is ANDed with the HFS segmentation region masks $I_{reg,i}$. The region mask $I_{reg,i}$ for which the highest number of white pixels remain after the AND operation is assumed to be the mask for the sky region. The road region is detected in a similar manner but using I_{lower} instead of I_{upper}.

The resulting road and sky region masks are shown in Figure 4.17. Note that in this case that the sky region mask does not contain the whole sky region due to the pole in the scene separating the sky into two parts as was shown in Figure 4.14. In this case a better approximation of the sky region could be obtained by simply keeping the part of the thresholded image shown in Figure 4.15 that is above the tunnel bounding rectangle. However, in general this method of finding the sky region based on the HFS provides much better results.



(a) Road region mask. **(b)** Sky region mask.

Figure 4.17: Road region and sky region masks.

4.6 Region classification

As previously mentioned, the sky and road regions in the image are detected in the first place to reduce the number of false vertical depth region detections. This is done by removing the MSER within the sky region mask and by removing the line segments within the road region mask and within the tunnel bounding rectangle. Alternatively, either the MSER or the line segments could be removed from both regions. In either case, the remaining MSER are assumed to be potential regions that may have vertical depth (or be considered standing object using the terminology from [9]), such as for instance lamp posts, signs, poles and buildings. In the current implementation only the MSER from the next darkest exposure image are used for the purpose of region classification and subsequent region refinement and depth map estimation.

The proposed approach to classify the remaining MSER into different geometric classes is based on the line segments within the region bounding rectangles. More specifically, the line segments are placed into different bins depending on their angles, forming what is here referred to as a line histogram.

4.6.1 Line histograms

A line histogram is defined in this report as the line segment magnitude (length) distribution as a function of the line segment orientations (angles). For each line segment, the orientation (angle) and the magnitude (length) is computed. The resulting magnitude is added to the bin corresponding to the orientation.

The computation of a line histogram is summarized in Algorithm 2. Here *vec* is the line segment container. Each line segment is defined by the two end points $\tilde{\mathbf{x}}_0 = (x_0, y_0)^T$ and $\tilde{\mathbf{x}}_1 = (x_1, y_1)^T$ and has a orientation $-90 \leq \theta \leq 90$ (note that the y-axis is pointing down in the image coordinate system and as a consequence, the line orientations are flipped compared to the familiar Euclidean coordinate system). Once the angle has been computed the corresponding bin index *binIndex* is assigned based on the angle range for each bin *binDist* (note that vertical or near vertical line segments are assigned bin index 0). Finally, the line segment magnitude is computed using the familiar Euclidean 2-dimensional distance formula and is added to the assigned bin.

Algorithm 2 Line Histogram

```
for $i = 0$ to $i = length(vec) - 1$ do
    $\theta = \arctan \frac{(y_1 - y_0)}{(x_1 - x_0)}$
    if $(\theta \geq -90$ && $\theta \leq -90 + binDist/2)$ ||
$(\theta \geq 90 - binDist/2$ && $\theta \leq 90)$ then
        $binIndex = 0$
    else
        $binIndex = ((\theta + 90 - binDist/2)/binDist) + 1$
    end if
    magnitude=$\sqrt{(y_1 - y_0)^2 + (x_1 - x_0)^2}$
    lineHistogram[binIndex]=lineHistogram[binIndex] + magnitude
end for
```

Figure 4.18 shows the line segments within a bounding rectangle for one of the detected MSER and Figure 4.19 shows the corresponding line histogram. Each bin in the line histogram has an angle range of 10° and there is thus a total of 18 bins. Bin 0 corresponds to vertical or near vertical ($-90 \leq \theta \leq -85$ or $85 \leq \theta \leq 90$) line segments, bin 1 to line segments with orientations $-85 \leq \theta \leq -75$, bin 2 to line segments with orientations $-75 \leq \theta \leq -65$, and so on. The total magnitude for the horizontal line segments ($-5 \leq \theta \leq 5$) is shown in bin 9. In the line histogram shown in Figure 4.19 there is a large peak at bin 0 meaning that vertical or near vertical line segments contribute to most of the magnitude, which in turn implies that it probably is a vertical region.

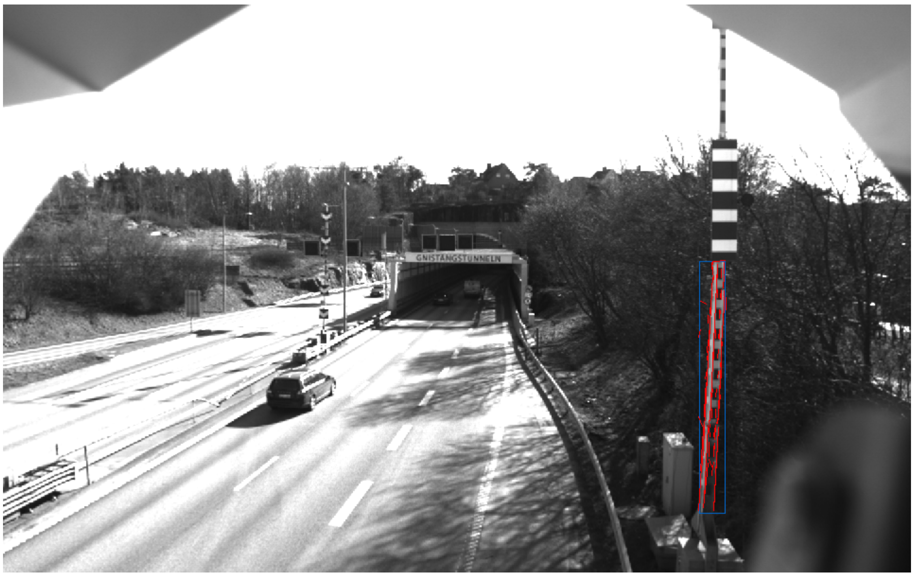


Figure 4.18: Bounding rectangle for one of the detected MSER and the corresponding line segments.

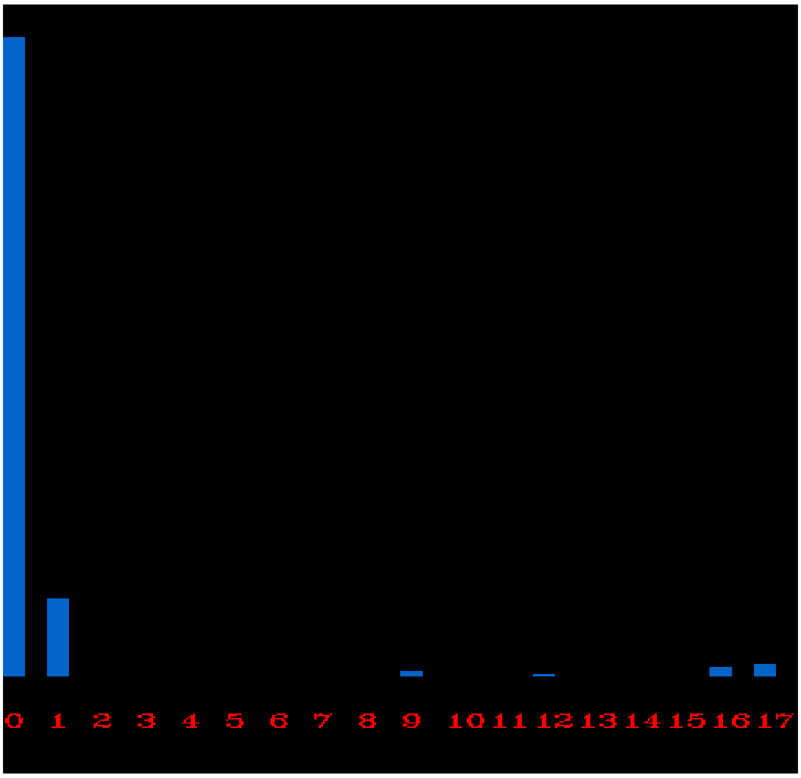


Figure 4.19: The line histogram for the line segments shown in Figure 4.18. Clearly, the vertical or near vertical line segments contribute with most of the magnitude, implying that it probably is a vertical region.

It could be argued that an alternative approach would be to simply count the line segments in each orientation bin instead of summing the magnitudes. However, it could be the case that there are only a few long line segments with a specific orientation but many more short line segments, leading to a result that is far less robust and more difficult to interpret. Anyway, the number of line segments in each bin is also stored in the current implementation and is used along with the line histogram to classify the regions as described below.

4.6.2 Classification of line histograms

Once the line histogram has been computed, the next step is to classify the region. Currently, a region can be classified as one of four classes: *Useless*, *Ambiguous*, *Vertical* and *Other*. The classification procedure is presented in Algorithm 3 and can be described as follows:
First the total number of line segments in the histogram is counted. If the number of line segments n_{lines} is 1 then the region is classified as *Useless* because a single line segment provides very unreliable information. After this, the number of bins with non-zero values is counted. If the number of bins with non-zero values N_{nz} is larger than half of the total number of bins, i.e. if $N_{nz} > N_{tot}/2$, then the region is classified as *Ambiguous*. The reasoning behind this is that if there are line segments with many different orientations it is hard to make use of this information to make a solid judgment regarding how the depth should be assigned. This usually

happens for objects that have ambiguous depth such as trees. In the final step, the line histogram indices are sorted by the bin magnitudes in decreasing order. Therefore, the first element in the sorted container is the index for the peak in the histogram. The region is then classified as *Vertical* if either the peak is at index 0 (corresponding to vertical or near vertical line segments) or if the peak is at index $N_{bins}/2$ (corresponding to horizontal or near horizontal line segments) and there is at least one vertical or nearly vertical line segment (i.e. bin 0 has a value $\neq 0$). Otherwise the region is classified as classified as *Other*.

Algorithm 3 Line Histogram Classification

```
function CLASSIFYLINEHIST(lineHist)
    lhClass=Other
    if $n_{lines} \leq 1$ then
        lhClass=Useless
        return lhClass
    end if
    if $N_{nz} > N_{tot}/2$ then
        lhClass=Ambiguous
        return lhClass
    end if
    indices=sortIndices(lineHist)
    if $indices[0] = 0 \; || \; (indices[0] = N_{bins}/2 \; \&\& \; lineHist[0] \neq 0)$ then
        lhClass=Vertical
    end if
    return lhClass
end function
```

The classification method described above is in some ways arbitrary and there is certainly a lot of room for experimentation and improvements. Currently, only regions classified as *Vertical* are considered as valid for the depth estimation.

4.7 Region refinement

The regions classified as *Vertical* are treated as regions with vertical depth (or standing objects). However, the MSER estimations of the regions are usually somewhat inaccurate. This can be seen in Figures 4.9 and 4.10. In the MSER images for the next darkest and next brightest exposures, it can be clearly seen that the lower right part of the large pole on the right side of the image is missing. The proposed approach to improve the region estimations is to use GrabCut foreground-background segmentation (refer back to Section 3.4.2 for a theoretical background) for all the exposure images and to select the best matching region approximation based on the best matching line histogram.

As mentioned in Section 3.4.2 the GrabCut segmentation method is semi-interactive and requires some prior knowledge. In OpenCV this prior knowledge can be provided in two ways to the GrabCut algorithm [54]. The first and simplest way is to input a rectangle to delimitate the region of interest. Only the part of the image inside the rectangle is segmented into foreground and background. The other method is to input an explicit mask with some pixels marked as background (value 0), some pixels marked as foreground (value 1) and the other pixels marked either as probably background (value 2) or probably foreground (value 3). This second method is more accurate than the rectangle method but the drawback is that more prior information is required. However, as described below, there is already enough prior information available in the implementation to use the second method.

The GrabCut mask for a region classified as *Vertical* is created in the following way: First the bounding rectangle for the MSER is enlarged both in the x- and y-directions by a number of pixels ϵ_x and ϵ_y respectively. Initially the bounding rectangle was defined by the upper left corner coordinate $\tilde{\mathbf{x}}_{up,left} = (x_0, y_0)^T$ and the lower right corner coordinate $\tilde{\mathbf{x}}_{down,right} = (x_1, y_1)^T = (x_0 + W - 1, y_0 + H - 1)^T$ where W is the width of the rectangle and H the height of the rectangle. After the enlargement the rectangle is defined by the corner points $\tilde{\mathbf{x}}'_{up,left} = (x_0 - \epsilon_x, y_0 - \epsilon_y)$ and $\tilde{\mathbf{x}}'_{down,right} = (x_1 + \epsilon_x, y_1 + \epsilon_y)$. Once the new rectangle has been created, a new mask of the same size as the rectangle is created and filled with value 3 (probably foreground). Then, the background (value 0) is drawn on the mask in the form of two rectangles on the left and right ends of the mask. The background rectangles have width ϵ_x and the same height as the mask. Finally, the MSER points are drawn in the mask with value 1 (foreground). A visualization of the GrabCut mask for the pole MSER is shown in Figure 4.20.

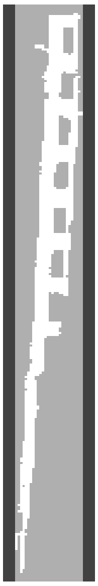


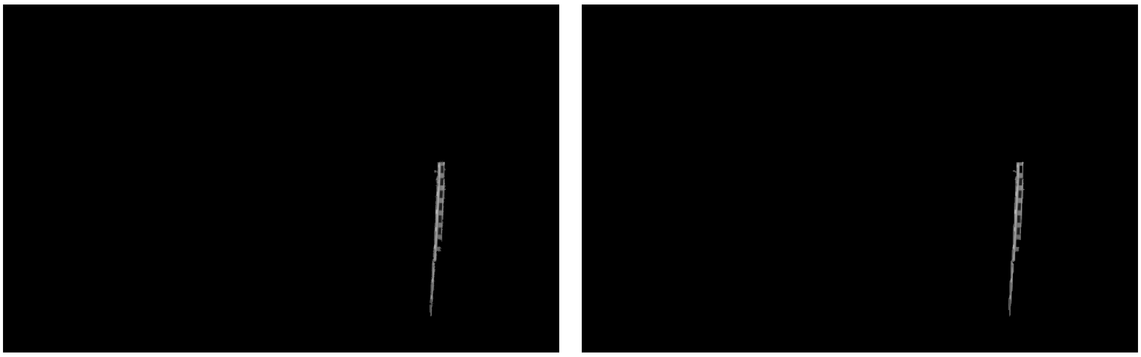
Figure 4.20: GrabCut mask for the pole MSER detected in the next darkest exposure image. Here the background is assigned grayscale value 65, the foreground is assigned grayscale value 255 and the probably foreground region is assigned grayscale value 175. The reason why the uppermost and lowermost parts of the mask are not assigned as background is because it is assumed that the object may actually be taller than the MSER.

After the GrabCut mask has been created, GrabCut segmentation is done for all of the exposure images resulting in slightly different region approximations as shown in

Figures 4.21 and 4.22. To determine which of the four region approximations is "optimal", the differences between the line histogram for the MSER bounding rectangle L_{mser} and the line histogram for the line segments within the region approximations $L_{reg,j}$ are calculated as

$$\delta_j = \sum_{i=0}^{N-1} abs(L_{mser}[i] - L_{reg,j}[i]) \tag{4.1}$$

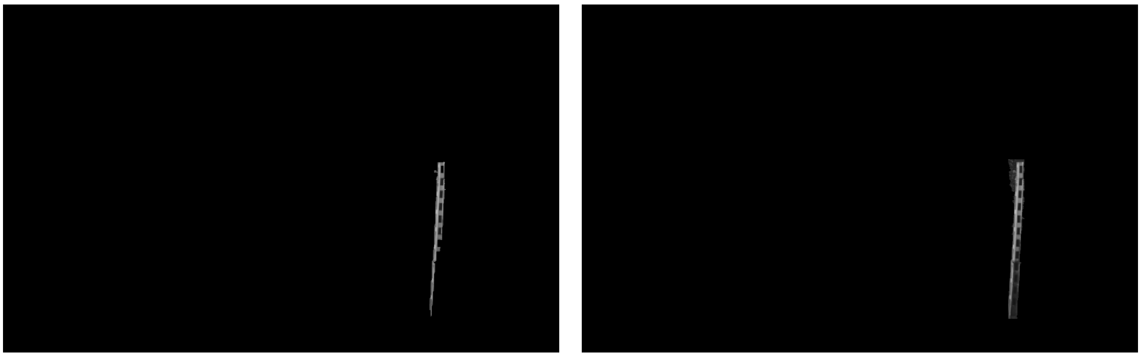
where N is the number of bins and j is the exposure image number. The region approximation which has the lowest error δ_j is considered to be the best region approximation and is later used in the depth map. In this case the best region approximation was obtained from the brightest exposure image. The corresponding mask is shown in Figure 4.23. It can be noted that the GrabCut approximation chosen contains some parts of the background (trees). A way to improve the result would be to instead approximate the MSER by rotated rectangles and then create GrabCut masks based on rotated rectangles instead of regular bounding rectangles. This has however not been implemented yet.



(a) GrabCut for the darkest exposure image.

(b) GrabCut for the next darkest exposure image.

Figure 4.21: Regions approximated by using GrabCut in the darkest and next darkest exposure images. The number of iterations for the GrabCut algorithm is 10.



(a) GrabCut for the next brightest exposure image.

(b) GrabCut for the brightest exposure image.

Figure 4.22: Regions approximated by using GrabCut in the next brightest and brightest exposure images. The number of iterations for the GrabCut algorithm is 10.

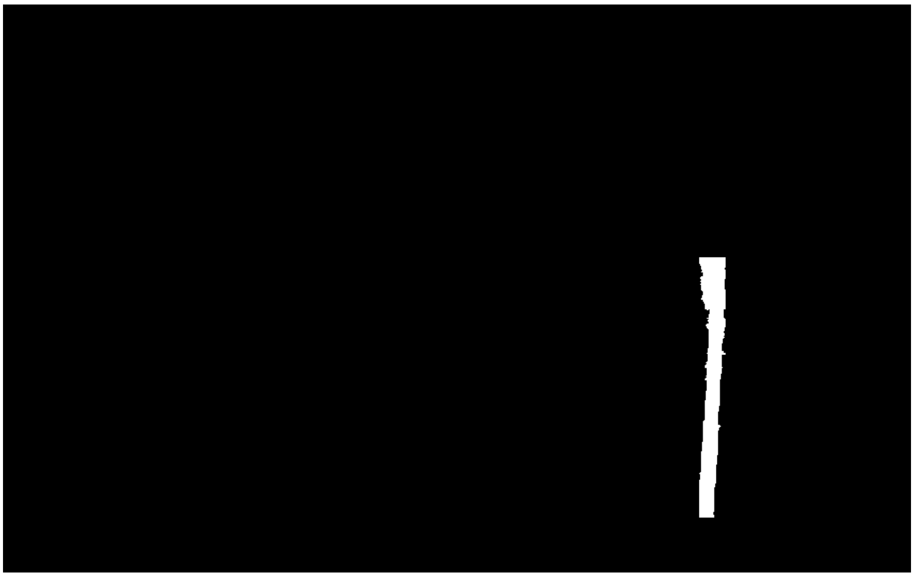


Figure 4.23: Depth region mask obtained from GrabCut.

4.8 Finding the tunnel outline and tunnel walls

In the implementation there are two regions in the image that are treated separately from the other depth regions, namely the tunnel outline and the tunnel walls. These are treated separately because the tunnel opening has previously been located as described in Section 4.4.1 and there is more geometric information available when compared to the other depth regions.

4.8.1 Finding the tunnel outline shape

Since the rectangle approximating the tunnel opening has been detected it can be assumed that the tunnel outline (outer part of the tunnel opening) is located left of, right of and above the rectangle (and that the road is located below the rectangle). Therefore, based on this assumption a GrabCut mask, such as the one shown in Figure 4.24, is created. The first step in creating this mask is to expand the tunnel bounding rectangle significantly in the left, right and upper directions and then to create a mask with the new expanded rectangle size and initialize it with value 3 (probably foreground). After this, the region defined by the tunnel rectangle is painted with value 0 (background). The background is also painted on the edges of the mask with a thickness t_{outer}. Finally, the foreground (value 1) is painted in the mask in the form of three strips of thickness t_{inner} around the tunnel bounding rectangle.

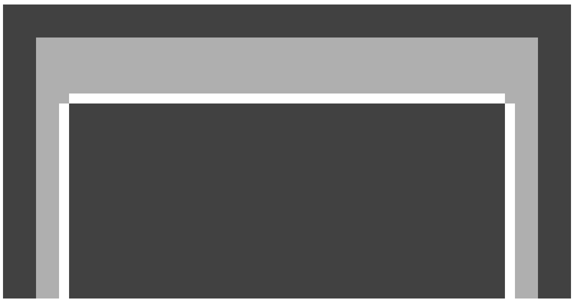


Figure 4.24: Visualization of the GrabCut mask for tunnel outline approximation.

Just as in Section 4.7, the GrabCut segmentation is done for all of the exposure images and the best tunnel outline is chosen based on the lowest error. In this case however there is no reference line histogram available. Therefore, the error function has been defined in a different way:

$$\epsilon_j = \sum_{i=0}^{N-1} L_{reg,j}[i] - s_v - s_h. \tag{4.2}$$

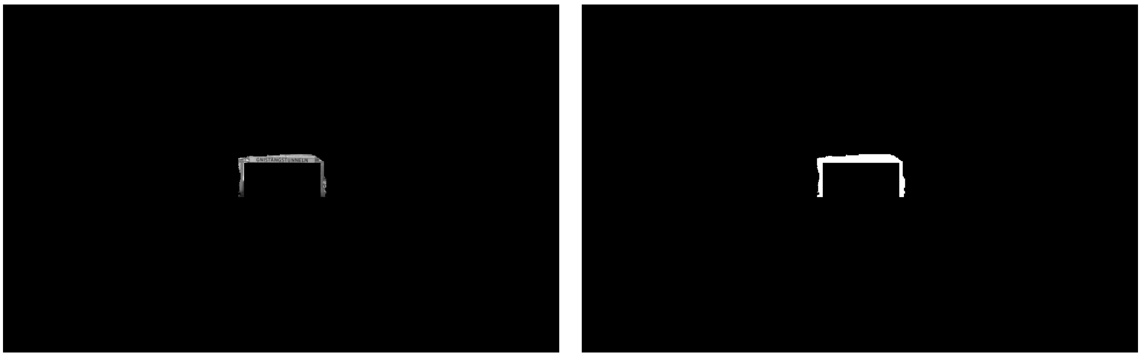
where j is the exposure image number, $L_{reg,j}$ is the line histogram for the current region approximation, N is the number of bins and s_v and s_h are sums defined as

$$s_v = L[N/2-1] + L[N/2] + L[N/2+1] \tag{4.3}$$

and

$$s_h = L[0] + L[1] + L[N-1] + L[N-2]. \tag{4.4}$$

The reasoning behind this error function is that a tunnel outline with a rectangular shape mostly contains vertical or near vertical and horizontal or near horizontal line segments. Thus a good approximation of the shape for a rectangular tunnel outline should have a line histogram with high magnitudes for the bins corresponding to vertical or near vertical line segments and horizontal or near horizontal line segments, resulting in a small error ϵ_j. The tunnel outline approximation with the lowest error ϵ_j is shown in Figure 4.25 along with the depth region mask.



(a) Best tunnel outline approximation (from the next brightest exposure image).

(b) The corresponding depth region mask.

Figure 4.25: Best tunnel outline approximation and the corresponding depth region mask. The number of iterations in the GrabCut segmentation algorithm is 7.

4.8.2 Finding the tunnel walls

The approach taken to find the tunnel walls differs significantly from the approaches used to find the vertical depth regions and the tunnel outline. Rather than relying on line histogram, the method is based on the vanishing point.

In the first step, the vanishing point in the tunnel $\mathbf{x}_{vp}$ is estimated using the MSAC method. The line segments inputted to the MSAC method are taken from the brightest exposure image. Only the line segments that are within the tunnel bounding rectangle and have orientations within the ranges $-85° \leq \theta \leq -10°$ and $10° \leq \theta \leq 85°$ are used for the estimation.

Once the vanishing point has been estimated, the lines outlining the left and right walls are computed as

$$\mathbf{l}_{left} = \mathbf{x}_{ll} \times \mathbf{x}_{vp} \tag{4.5}$$

and

$$\mathbf{l}_{right} = \mathbf{x}_{lr} \times \mathbf{x}_{vp}, \tag{4.6}$$

where $\mathbf{x}_{ll}$ is the lower left corner of the rectangle and $\mathbf{x}_{lr}$ is the lower right corner of the rectangle. The point of intersection of the lines with the upper rectangle line $l_{r,up}$ are defined as

$$\mathbf{x}_l = \mathbf{l}_{r,up} \times \mathbf{l}_{left} \tag{4.7}$$

and

$$\mathbf{x}_r = \mathbf{l}_{r,up} \times \mathbf{l}_{right}, \tag{4.8}$$

where $\mathbf{l}_{r,up}$ is computed as

$$\mathbf{l}_{r,up} = \mathbf{x}_{ul} \times \mathbf{x}_{ur}. \tag{4.9}$$

Here, $\mathbf{x}_{ul}$ and $\mathbf{x}_{ur}$ are the upper left and upper right corners of the rectangle. The left and right walls can then be defined as polygons with three inhomogeneous coordinates (i.e. triangles) (see equations (4.10) and (4.11)).

$$p_{left} = [\tilde{\mathbf{x}}_{ll},\ \tilde{\mathbf{x}}_{l},\ \tilde{\mathbf{x}}_{ul}] \tag{4.10}$$

$$p_{left} = [\tilde{\mathbf{x}}_{lr},\ \tilde{\mathbf{x}}_{r},\ \tilde{\mathbf{x}}_{ur}] \tag{4.11}$$

The resulting polygon approximations of the walls are shown in Figure 4.26.

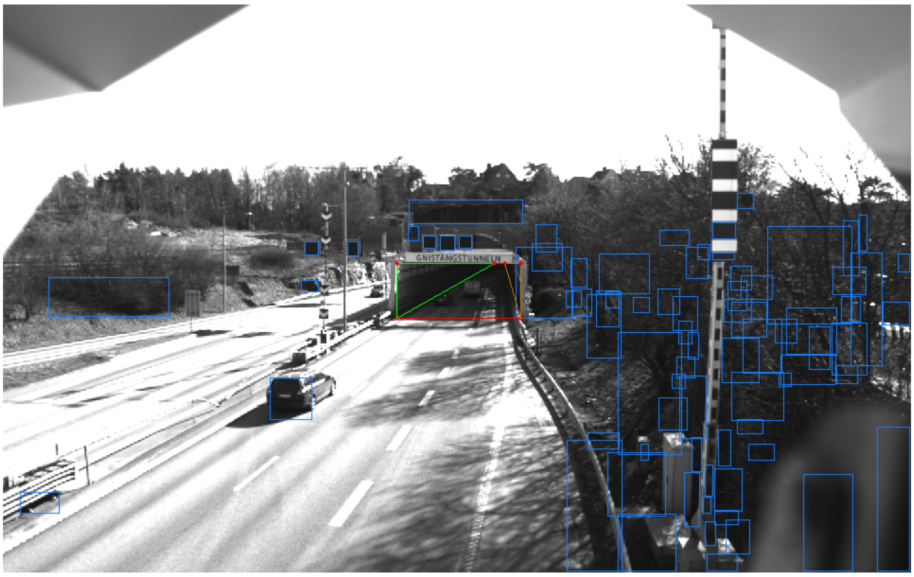


Figure 4.26: Approximation of the tunnel opening. The tunnel bounding rectangle is shown in red and the left and right tunnel walls are approximated as triangles. The polygon coordinates are also drawn.

4.9 Grouping and refinement of depth regions

The obtained depth regions so far (excluding the tunnel walls which are handled separately) are shown in Figure 4.27. From the figure it can be seen that depth regions that in actuality have the same depth (such as the different parts of the pole on the right side) or very similar depth (such as the tunnel outline and the signs above the tunnel) are in different depth regions. This is a problem because as described in Section 4.10, the depth for a vertical object is sampled directly from below the bounding rectangle of that region. Thus for instance if a part of the pole is treated as a different region it will be considered to have a different depth than the other parts. To avoid this situation the depth regions are grouped together vertically as described in Algorithm 4

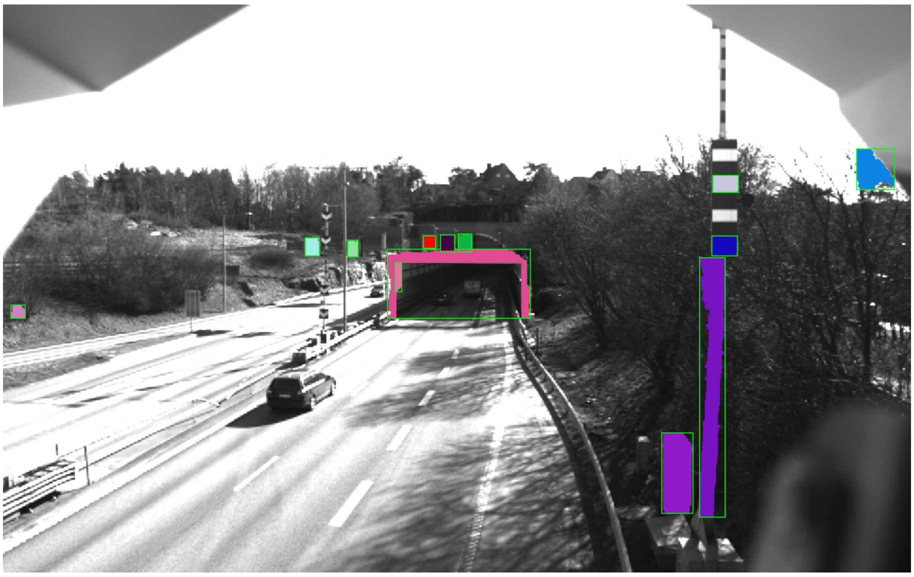


Figure 4.27: Depth regions obtained after region classification and region mask creation and after the tunnel outline approximation. The region bounding boxes are also shown.

The algorithm begins with the creation of an empty container for the depth region sets (depth region groupings). After this the *setIds* container is initialized to -1. This container keeps track of the set number a specific depth region belongs to and when the value is -1 it means that the depth region does not belong to any set.

After the initialization the function loops through all the depth regions, which have prior to the function call been sorted from bottom to top based on the bottom y-values of the bounding rectangles. If the current depth region is not part of a set yet, i.e. if $setIds[i] = -1$, then a new depth set containing the current depth region is added to the container with the depth sets *depthSets*. The region is then given the current set ID *setId*. After that the current set ID is incremented. In this way the next depth set will be given a new set ID. In the inner loop a search is done to find matching depth regions. In the first if-statement a check is done to see if the depth region does not belong to a depth set yet and if the regions are close enough vertically, i.e. if the top y-coordinate for bounding rectangle i is not further than ϵ_y away from the bottom y-coordinate for bounding rectangle j. Inside the if-statement two boolean variables are evaluated to determine if there also is any overlap between the regions in the x-direction. If conditions in the inner if-statement are fulfilled then there is both overlap in the vertical and horizontal directions. In that case depth region j is added to the depth region set and the set number is set to that of region i.

The result of the algorithm is illustrated in Figure 4.28. It can be noted that the depth regions for the pole have been grouped together and that the signs over the tunnel entrance are in the same depth set as the tunnel outline. The results however are not yet entirely satisfactory considering the large gap that can be seen between two of the depth regions on the pole. There is also a depth region inside the tunnel which should not be there.

Algorithm 4 Grouping of depth regions into depth region sets

```
function findDepthSets(depthRegs, $\epsilon_y$, $\epsilon_x$)
    Initialize depthSets container as empty
    Initialize setIds container to -1
    $setId = 0$
    for $i = 0$ to $i = length(depthRegs) - 1$ do
        if $setIds[i] = -1$ then
            Add new depth set containing DepthRegs[i] to depthSets
            setIds[i]=setId
            setId=setId+1
        end if
        curr=depthRegs[i].boundingRect
        for $j = i + 1$ to $j = length(depthRegs) - 1$ do
            next=depthRegs[j].boundingRect
            if $setIds[j] = -1$ && $curr.y_{top} + \epsilon_y \geq next.y_{bot}$ then
                nextRegThinner=
                $(next.x_{left} \geq curr.x_{left}$ && $next.x_{left} \leq curr.x_{right})$ ||
                $(next.x_{right} \geq curr.x_{left}$ && $next.x_{right} \leq curr.x_{right})$
                nextRegWider= $next.x_{left} \geq curr.x_{left} - \epsilon_x$
                && $next.x_{right} \leq curr.x_{right} + \epsilon_x$
                if nextRegThinner || nextRegWider then
                    setIds[j]=setIds[i]
                    add depthRegs[j] to depthSets[setIds[i]]
                end if
            end if
        end for
    end for
end function
```

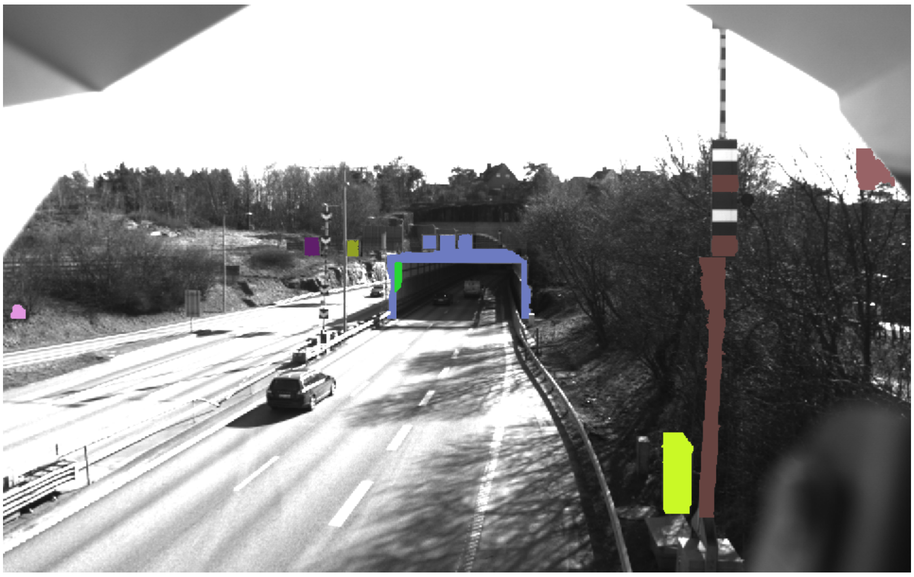


Figure 4.28: Depth regions after grouping into depth region sets.

The depth region inside the tunnel can easily be removed based on the tunnel bounding rectangle but the closing of gaps in sets of depth regions is slightly more complicated and involves adding rectangular depth regions in-between the existing depth regions where there exists a gap. For the sake of brevity the exact details are left out. The result of the refinements is shown in Figure 4.29.

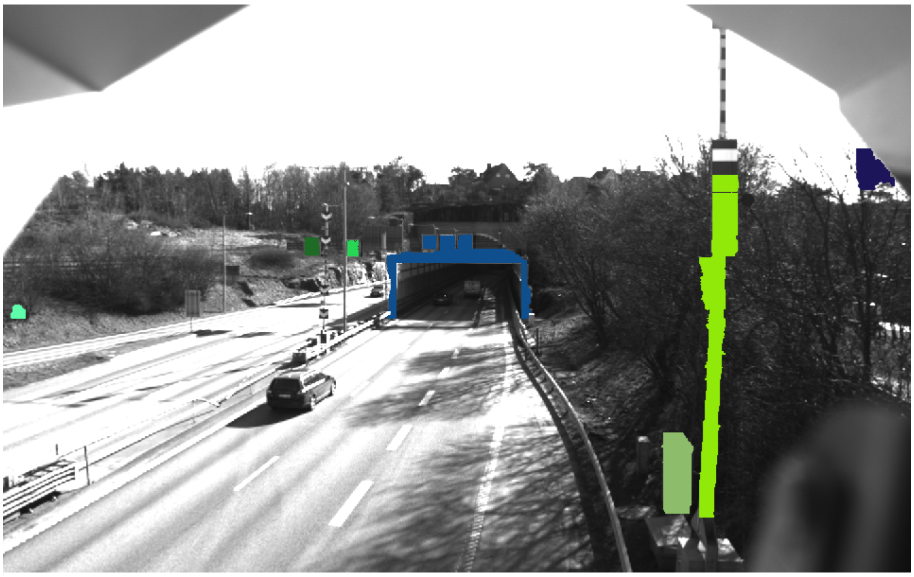


Figure 4.29: Depth region sets after refinement.

4.10 Depth Map Estimation

Once all the depth regions in the image have been detected it is finally time to actually create the depth map. First, a depth map is created based on the field of view method described in Section 3.6.1. As previously mentioned this depth map only measures the depth on the plane (in this case the road) and does not take into account that objects located on the plane have other depth values. Next, the depth for the tunnel walls is added to the depth map. In the final step, all of the detected vertical depth regions are added.

4.10.1 Depth Map based on Field of View

As mentioned in Section 3.6.1, the field of view based method of depth estimation requires four parameters: the vertical field of view α_v, the horizontal field of view α_h, the height of the camera h and the camera angle relative to the road θ. As is mentioned in Section 5.1, the vertical and horizontal fields of view are properties of the camera and lens and for the specific camera and lens. The parameters h and θ on the other hand vary depending on the scene. The camera height h is assumed to be given while the camera angle θ can be calculated from θ_x (the pitch angle) which is either assumed to be given or can be computed as described in Section 4.11.

The depth (Z-value) for an image coordinate $\tilde{\mathbf{x}} = (x, y)^T$ is computed using equations (3.55) to (3.59). It would be possible to directly compute the Z-values when warping but the main issue would be that the depth value would only valid for points on the plane (points on the road) and be invalid in all other cases (such as for vertical objects). Instead, the depth values (Z-values) are stored in a lookup table T of length N and the depth map D stores the indices to the lookup table (0 to $N-1$) for all image coordinates. Thus, the depth value Z is obtained from the look-up table as follows

$$Z = T[D[x, y]]. \tag{4.12}$$

The value at index i in the look-up table is calculated as

$$T[i] = Z_{far} - i\frac{Z_{far} - Z_{near}}{N-1}, \tag{4.13}$$

where Z_{far} is a user-defined maximum depth for the scene and Z_{near} is the nearest depth value in the image. Z_{near} is computed using the field of view depth estimation method for $\tilde{\mathbf{x}}_{near} = (W/2, H-1)^T$ where W is the image width and H is the image height.

The depth map index at $\tilde{\mathbf{x}} = (x, y)^T$ is computed from the depth Z using equation (4.14) as follows:

$$D[x, y] = \begin{cases} 0, & \text{if } Z > Z_{far} \text{ or } Y < 0 \\ (N-1)\frac{Z_{far}-Z}{Z_{far}-Z_{near}}, & \text{otherwise} \end{cases}. \tag{4.14}$$

Checking for $Y < 0$ is necessary because for some unknown reason the field of view method can give negative Y values, resulting in an incorrect depth map.

The field of view based depth map for the Gnistängstunneln scene is shown in Figure 4.30. Darker pixels represent points further away from the camera. The pixels with value 0 (black) have depth $Z = Z_{far}$ and can be thought of as points on the plane at infinity.

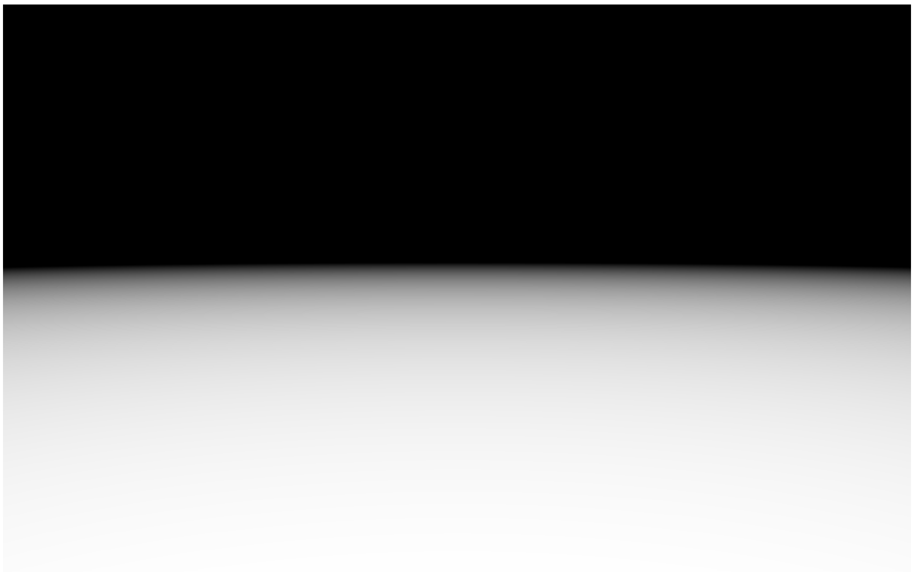


Figure 4.30: Field of view based depth map. $N = 256$ and $Z_{far} = 250$ in this case.

4.10.2 Adding the tunnel walls

As previously mentioned in Section 4.8.2, the left and right tunnel walls are represented by two polygons p_{left} and p_{right} respectively. For convenience the definitions of the polygons are repeated below. They are

$$p_{left} = [\tilde{\mathbf{x}}_{ll},\ \tilde{\mathbf{x}}_{l},\ \tilde{\mathbf{x}}_{ul}] \tag{4.15}$$

and

$$p_{left} = [\tilde{\mathbf{x}}_{lr},\ \tilde{\mathbf{x}}_{r},\ \tilde{\mathbf{x}}_{ur}]. \tag{4.16}$$

Here $\tilde{\mathbf{x}}_{ll}$, $\tilde{\mathbf{x}}_{lr}$, $\tilde{\mathbf{x}}_{ul}$ and $\tilde{\mathbf{x}}_{ur}$ are the corners of the tunnel bounding rectangle (lower left, lower right, upper left and upper right respectively). $\tilde{\mathbf{x}}_{l}$ and $\tilde{\mathbf{x}}_{r}$ are the intersections of the upper rectangle line with the left and right wall lines respectively.

The depth for a tunnel wall is computed along the line segment between the first and second points in the polygon. This is the line segment delineating the boundary between the road and the tunnel wall. For each point $(x_i, y_i)^T$ on the line, the depth value at one unit below, i.e. at $(x_i, y_i + 1)^T$, is sampled. A vertical line segment with the sampled depth $D[x_i, y_i + 1]$ is then drawn from $(x_i, y_i)^T$ to the upper line of the bounding rectangle. The result of adding the depth for both of the walls is shown in Figure 4.31.

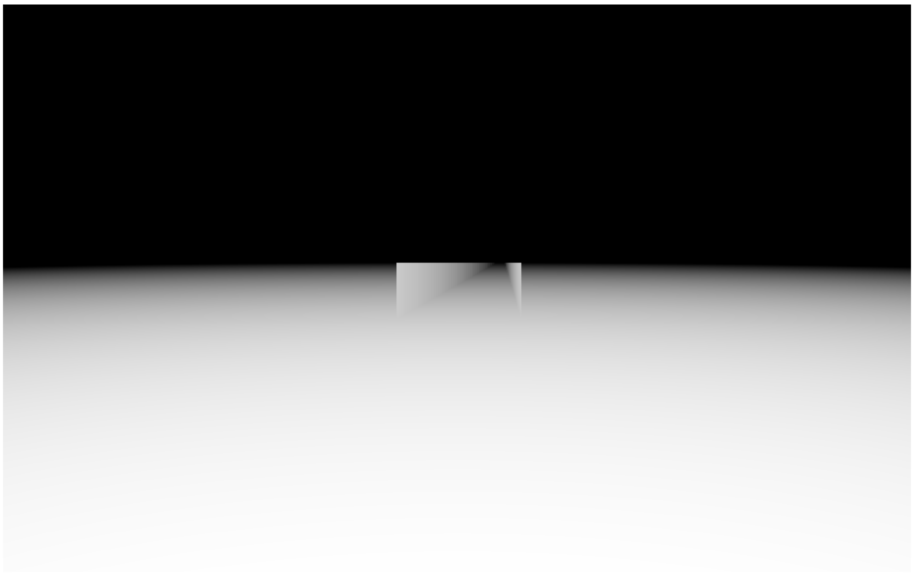


Figure 4.31: Depth map with the tunnel walls included.

4.10.3 Adding vertical depth regions

In the final step of the depth map creation, the regions with vertical depth are added. The depth for a vertical depth region set is computed along the lower line of the set bounding rectangle r. For each point $(x_i, y_{bottom})^T$ on the line, the depth index value at one unit below is sampled from the depth map D. This depth value is then assigned for all the points directly above (same x-coordinate) that are within the bounding rectangle and for which the mask M is defined (i.e. where it has value 255 or white). The procedure is summarized in Algorithm 5 and the complete depth map is shown in Figure 4.32.

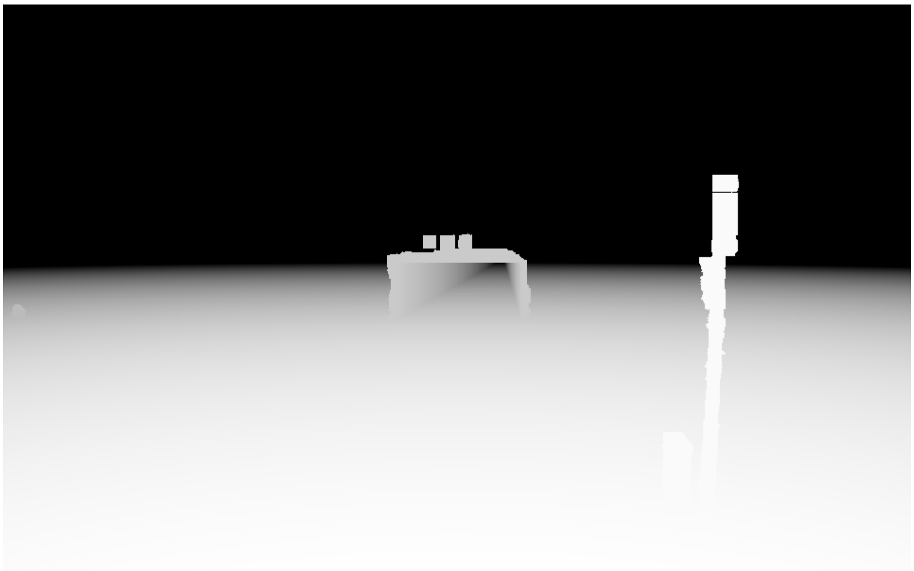


Figure 4.32: Complete depth map with all of the depth regions included.

Algorithm 5 Drawing of vertical depth regions

```
function addDepthRegions(D, depthSets)
    for $i = 0$ to $i = length(depthSets) - 1$ do
        r=depthSets.getRect()
        M=depthSets.getMask()
        $x_{start} = r.x$
        $x_{end} = r.x + r.width - 1$
        $y_{bottom} = r.y + r.height - 1$
        $y_{top} = r.y$
        $y_{sample} = y_{bottom} + 1$
        for $x = x_{start}$ to $x = x_{end}$ do
            $depth = D[y_{sample}, x]$
            for $y = y_{top}$ to $y = y_{bottom}$ do
                if M[x,y]=255 then
                    D[x,y]=depth
                end if
            end for
        end for
    end for
end function
```

4.11 The Camera Geometry

As described in Section 3.1.4, the generation of a virtual view requires not only a depth map but also two camera matrices P for P' for the original and virtual positions respectively. In the implementation, the original camera view is set at origin and points in the Z direction. Thus, the camera matrix is defined as

$$P = K[I|\mathbf{0}], \tag{4.17}$$

where K is the calibration matrix estimated through calibration (refer to Section 5.2 for the estimated calibration matrix), I is the identity matrix and $\mathbf{0}$ is the 3×1 zero vector.

The camera matrix for the virtual view P' is then defined relative to P as

$$P' = K[R|\mathbf{t}] = KR[I| - \tilde{\mathbf{C}}] \tag{4.18}$$

where $\tilde{\mathbf{C}} = (c_x, c_y, c_z)^T$ is the position of the virtual view relative to the origin and R is the rotation matrix. As described in Section 3.1.2, the rotation matrix can be defined by three angles θ_x (pitch), θ_y (yaw) and θ_z (roll). There are thus in total 6 parameters (c_x, c_y, c_z, θ_x, θ_y and θ_z) defining the difference in position and orientation between the two views. The functions of the parameters are summarized in Table 4.1.

Parameter	Function	Positive value	Negative value
c_x	X-axis translation	Translation right	Translation left
c_y	Y-axis translation	Translation down	Translation up
c_z	Z-axis translation	Translation forward (zoom in)	Translation backward (zoom out)
θ_x	Rotation around the X-axis	Rotation of the view upwards	Rotation of the view downwards
θ_y	Rotation around the Y-axis	Rotation of the view right	Rotation of the view left
θ_z	Rotation around the Z-axis (same as 2D image rotation)	Clockwise rotation	Counter-clockwise rotation

Table 4.1: Summary of the parameters for defining the virtual camera view position and orientation.

How these parameters are obtained may differ from case to case. However, if the distance to the middle of the road (x-axis distance to the middle of the tunnel) d_x, the height of the camera above the road surface h, the tunnel height H and the distance to the tunnel D are given, then c_x, c_y, θ_x and θ_y can easily be calculated. Figures 4.33 and 4.34 show the geometry for a typical tunnel scene. In this case the camera is assumed to be on the right side of the road, which means that $c_x = -d_x$. If the camera would be on the left side of the road then $c_x = d_x$. The desired height over the road surface for the virtual view is $h_{virt} = 1.5$ which means that $c_y = h - h_{virt}$. In Figures 4.33 and 4.34 it is assumed that the camera that the camera is centered on the tunnel opening. Therefore, the angles θ_x and θ_y can be calculated using

$$\theta_x = \arctan \frac{h - H/2}{D} \tag{4.19}$$

and

$$\theta_y = \arctan \frac{d_x}{D} \tag{4.20}$$

respectively. In this case the angles are positive but for a camera placed on the left side of the road, θ_y would be negative. If the camera is mounted parallel to the road plane then the roll θ_z can be assumed to be negligible. Usually it can also be assumed that there is no Z-axis translation, i.e. $c_z = 0$.

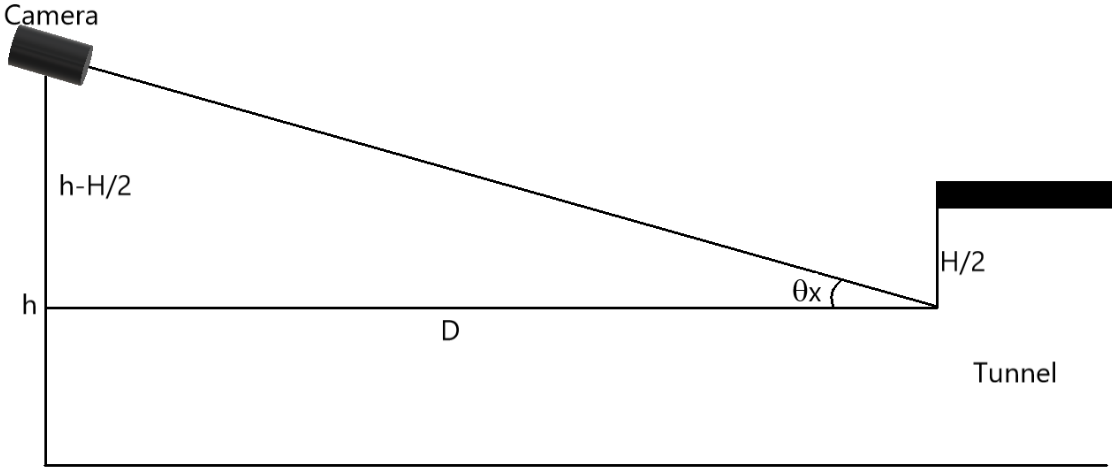


Figure 4.33: Geometry for a typical tunnel scene viewed from the side. The computation of the pitch θ_x is listed in equation (4.19).

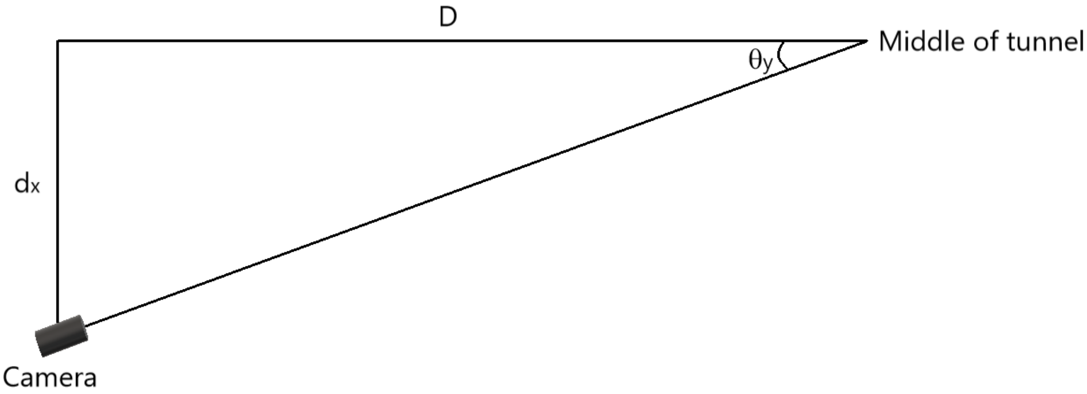


Figure 4.34: Geometry for a typical tunnel scene viewed from above. The computation of the yaw θ_y is listed in equation (4.20).

4.12 Warping

Once the depth map has been estimated and the camera matrices have been defined, the original view of the scene is warped using the warping procedure in Algorithm 6. The algorithm only includes the relevant parts of the warping function. In the actual implementation however there are several masks created, some of which are important for the inpainting and others for the luminance estimation. These will be described later.

The function takes in five parameters: the 4×4 camera matrices P and P' described in Section 3.1.4, the image to be warped I, the depth map D and the lookup table T. Both I and D are of size $W \times H$

Since the size of the warped image is not known beforehand, the purpose of the first double for-loop is to find the minimum and maximum warped coordinates (x_{min}, y_{min}) and (x_{max}, y_{max}). At the start of the inner loop, the depth is obtained by first retrieving the depth map value at the current coordinate $\tilde{\mathbf{x}} = (j, i)^T$ and then using this value as the index to the lookup table. Afterwards, the projected coordinates $\tilde{\mathbf{x}}' = (x, y)^T$ are obtained by calling the *ProjectedCoordinates* function. The function computes the warped coordinates as

$$Z'\mathbf{x}' = ZP'P^{-1}(\mathbf{x}), \tag{4.21}$$

This is done in two steps: first the image coordinate $\mathbf{x} = (j, i, 1)^T$ is back-projected to $(X, Y)^T$ using the the depth Z and the inverse of the projection matrix for the original view, i.e. P^{-1}. Then, the 3D coordinate $\mathbf{X} = (X, Y, Z, 1)^T$ is projected to $\mathbf{x}' = (x', y', w)^T$ using the camera matrix for the virtual view P'. Finally, the inhomogeneous coordinates are obtained as $\tilde{\mathbf{x}}' = (x, y)^T = (x'/w, y'/w)^T$. Once the projected coordinate point has been obtained, the maximum and minimum warped coordinates are updated if needed. To avoid re-doing the warping computation in the second loop, the projected coordinates x and y are stored in the matrices $projX$ and $projY$ respectively.

After the first double for-loop, the warped image is initialized as an empty (black) image of size $\delta_x \times \delta_y$, where $\delta_x = x_{max} - x_{min}$ and $\delta_y = y_{max} - y_{min}$. Furthermore, the matrix Z_{true} of size $\delta_x \times \delta_y$ is also created. This matrix is used in the second double for-loop to keep track of the lowest depth at the warped coordinate $\tilde{\mathbf{x}}'$. It could after all be the case that two or more coordinates $\tilde{\mathbf{x}}$ with different depths Z are warped to the same point $\tilde{\mathbf{x}}'$. In that situation only the closest point (i.e. with the lowest depth) should be drawn in the warped image.

At the start of the second loop, the depth value Z is retrieved similarly to how it was done in the first loop. Then the warped coordinates x and y are retrieved from the matrices $projX$ and $projY$. The warped coordinates x and y are then shifted by x_{min} and y_{min} respectively. This is required to keep the warped coordinates x and y within the ranges of the defined coordinates in the image, i.e. $0 \leq x \leq \delta_x - 1$ and $0 \leq y \leq \delta_y - 1$. It could after all be the case that the warped coordinates are negative or that the minimum coordinates x_{min} or y_{min} are larger than 0. An extra check is then done to make sure that the coordinates indeed are within defined coordinate range for the warped image. In the final part of the loop, the value in the original image I at coordinate $(j, i)^T$ is copied to the warped image I_w at the warped coordinate $(x, y)^T$ and the depth value stored in Z_{true} is updated. If the depth value at Z_{true} has not yet been assigned then it is simply given the value Z. Otherwise, if the current depth value Z is smaller than the value already stored in Z_{true}, then the stored depth value is updated to Z and a new pixel value is assigned at $(x, y)^T$ in the warped image. As mentioned above this is needed in order to only show the closest pixel at each coordinate $(x, y)^T$ in the warped image (the points further away are occluded).

The result of warping using this method is shown in Figure 4.35. In this case it was assumed that $\tilde{\mathbf{c}} = (c_x, c_y, c_z)^T = (-7.5, 5.5, 0.0)^T$ and that $\boldsymbol{\theta} = (\theta_x, \theta_y, \theta_z)^T = (3.78°, 6.29°, 0.0°)^T$. The distance between the camera and the middle of road was estimated from measurement in a map to be around $d_x = 7.5m$. In the intro-

Algorithm 6 Depth-Image Based Warping

```
function WARP(P, P', I, D, T)
    Initialize projX to an empty matrix of size $W \times H$ with values 0.0
    Initialize projY to an empty matrix of size $W \times H$ with values 0.0
    $x_{min} = \infty$
    $x_{max} = -\infty$
    $y_{max} = \infty$
    $y_{min} = -\infty$
    for $i = 0$ to $i = H - 1$ do
        for $j = 0$ to $W - 1$ do
            d=D[j, i]
            Z=T[d]
            (x, y) = ProjectedCoordinates(P, P', j, i, Z)
            if $x < x_{min}$ then
                $x_{min} = x$
            end if
            if $x > x_{max}$ then
                $x_{max} = x$
            end if
            if $y < y_{min}$ then
                $y_{min} = y$
            end if
            if $y > y_{max}$ then
                $y_{max} = y$
            end if
            projX[i,j]=x
            projY[i,j]=y
        end for
    end for
    $\delta_x = x_{max} - x_{min}$
    $\delta_y = y_{max} - y_{min}$
    Initialize $I_w$ to image of size $\delta_x \times \delta_y$ with values 0
    Initialize $Z_{true}$ to a matrix of size $\delta_x \times \delta_y$ with values 0.0
end function
```

```
function WARP (CONTINUATION)(P, P', I, D, T)
    for $i = 0$ to $i = H - 1$ do
        for $j = 0$ to $W - 1$ do
            d=D[i,j]
            Z=T[d]
            x=projX[i,j]
            y=projY[i,j]
            $x = x - x_{min}$
            $y = y - y_{min}$
            if $x \geq 0$ && $x < \delta_x$ && $y \geq 0$ && $y < \delta_y$ then
                if $Z_{true}[x, y] = 0.0$ then
                    $Z_{true}[x, y] = Z$
                    $I_w[x, y] = I[j, i]$
                else
                    if $Z < Z_{true}[x, y]$ then
                        $Z_{true}[x, y] = Z$
                        $I_w[x, y] = I[j, i]$
                    end if
                end if
            end if
        end for
    end for
end function
```

duction it was mentioned that the distance of the camera to Gnistängstunneln is approximately $D = 68m$ and the height of the camera is approximately $h = 7m$. Furthermore, it is known that the height of the tunnel is around $H = 5m$. Using these relations it is possible to recover both the virtual camera position and the pitch and yaw angles (see equations (4.19) and (4.20)).

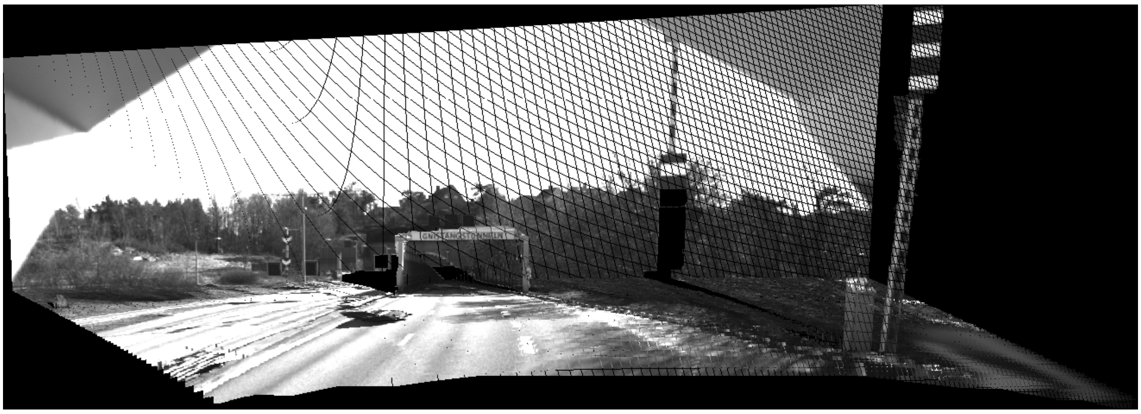


Figure 4.35: Warped image of the Gnistängstunneln scene. The parameters $\tilde{\mathbf{c}} = (-7.5, 5.5, 0.0)^T$ and $\theta = (3.78°, 6.29°, 0.0°)^T$ were used in this case.

4.12.1 Inpainting

As can be seen in Figure 4.35, the warped image contains many holes where there are no pixel values defined. The small holes are caused by rounding of the warped coordinates into the nearest integers and the larger holes are disocclusions caused by large differences in the depth value. The holes in the image can closed with inpainting. This step is not strictly necessary for the pipeline as a whole but can certainly improve the visualization of the result.

In OpenCV there are two built-in inpainting methods [55]: A Navier-Stokes based inpainting method and The Fast Marching Based inpainting method [15]. The inpainting function requires two parameters: the mask to defining the points to be inpainted and the radius around each point defining the size of the pixel neighborhood to be considered when inpainting. The method by which the inpainting mask for the warped image is defined is described below:

First, the inpainting mask M_I is initialized as a completely white image of the same size as the warped image. Then, the mask is assigned the value 0 (black) for all warped coordinates (all points that are defined in the in the warped image). The mask so far is shown in Figure 4.36. It can be seen that the boundaries in the inpainting mask are white and this is clearly undesirable since the boundaries should not be inpainted. To remove the white regions at the boundaries in the inpainting mask M_I, the so called outline mask M_O is created to delineate the defined boundaries in the warped image.

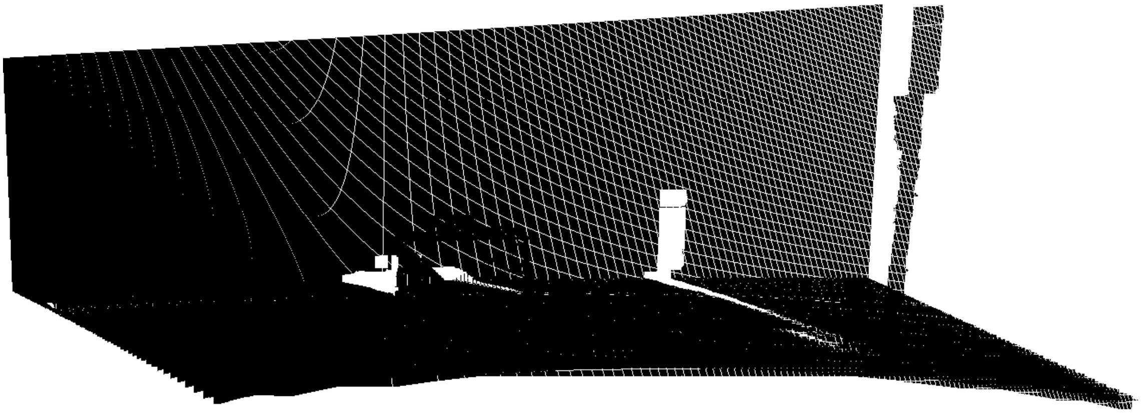


Figure 4.36: The incomplete inpainting mask after filling the it with zeros at the defined pixel values in the warped image.

The first step in creating the outline mask M_O is to threshold the warped image such that the pixels with non-zero values in the warped image are assigned the value 255 (white). Next, 5 iterations of a closing morphological operation are applied on the threshold mask M_T to close most of the gaps in the mask (refer to [56] for a short summary of different morphological operations). After this, the largest contour in the mask is found and this contour is drawn on the empty (black) outline mask image as white (255). The resulting outline mask is shown in Figure 4.37.

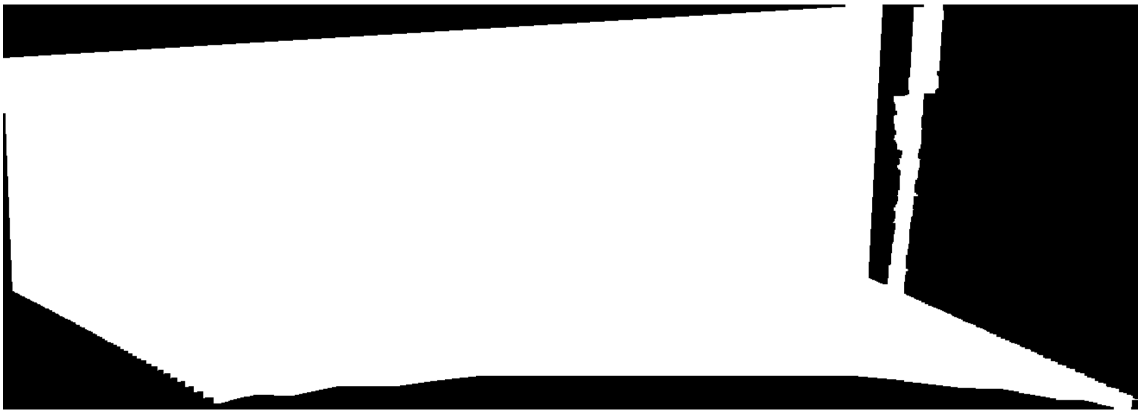


Figure 4.37: Mask showing the outline of the warped image region.

In the final step, the coordinates in M_I for which M_O has pixel value 0 are filled with with black color. The resulting inpainting mask is shown in Figure 4.38.

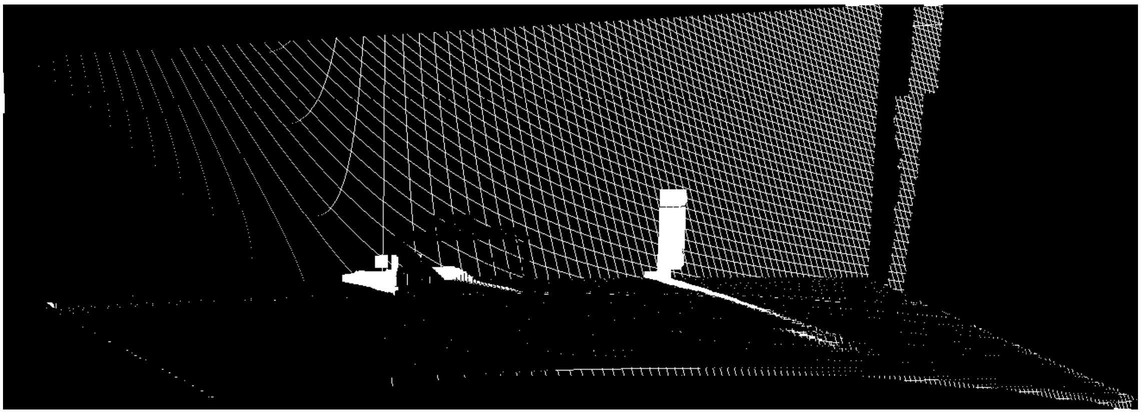


Figure 4.38: Final inpainting mask.

The result of inpainting the warped image is shown in Figure 4.39. Note that the result is good enough for the small holes but there are major artifacts for the large holes.

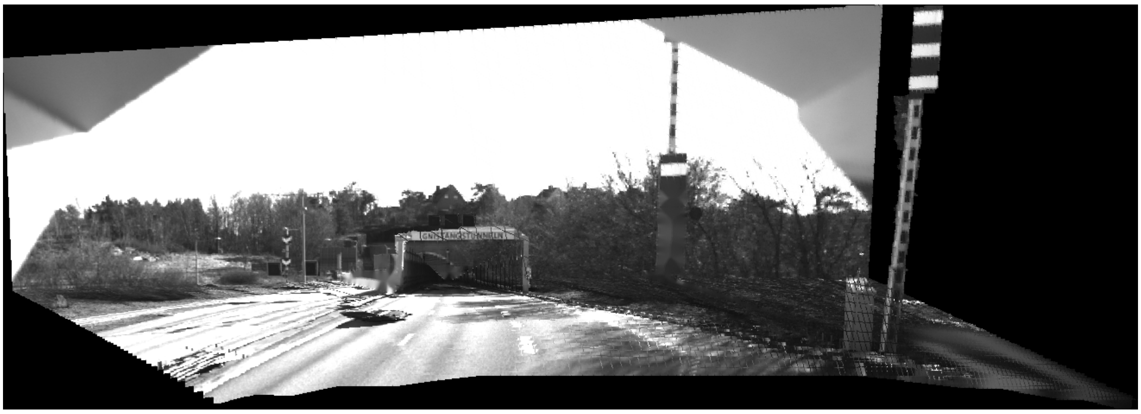


Figure 4.39: The inpainted warped image. The radius was set to 3 in this case.

4.13 Defining the Equivalent Veiling Luminance region

The theory behind the Equivalent Veiling Luminance (L_{seq}) estimation is described in Section 3.7. For images which have not been warped, the theory can be applied without any significant changes to compute the luminance. However, warped images require some extra considerations. The main difficulty with estimating the luminance in the warped image is that the image size differs from the original image size. The warped image shown in Figure 4.39 is of size $W_w \times H_w = 1324 \times 478$ while the original image is of size $W \times H = 960 \times 600$. The warped image is thus both shorter and wider and thus has a different aspect ratio (ratio of width and height). Furthermore, the vertical distance from the middle of the tunnel to the bottom of the image $\Delta_{y,bot} = y_{bot} - y_{tunnel}$ usually differs from the vertical distance from the middle of the tunnel to the top of the image $\Delta_{y,top} = y_{tunnel} - y_{top}$. Although a mathematically robust way to define the L_{seq} region in the warped image may exist, in the implementation an empirical method is used.
First note that the L_{seq} region in the warped image should be defined in such a way that it roughly approximates the L_{seq} diagram for the street view. Also note that the radius for the L_{seq} region in unwarped views is calculated based on the image height and that the region is circular. In the warped view it is not possible (or rather not desirable) to define a circular region in the same way. Instead, the region consists of two semi-ellipses of different sizes. The pattern is shown in Figure 4.40 and the method by which it was created is described below.

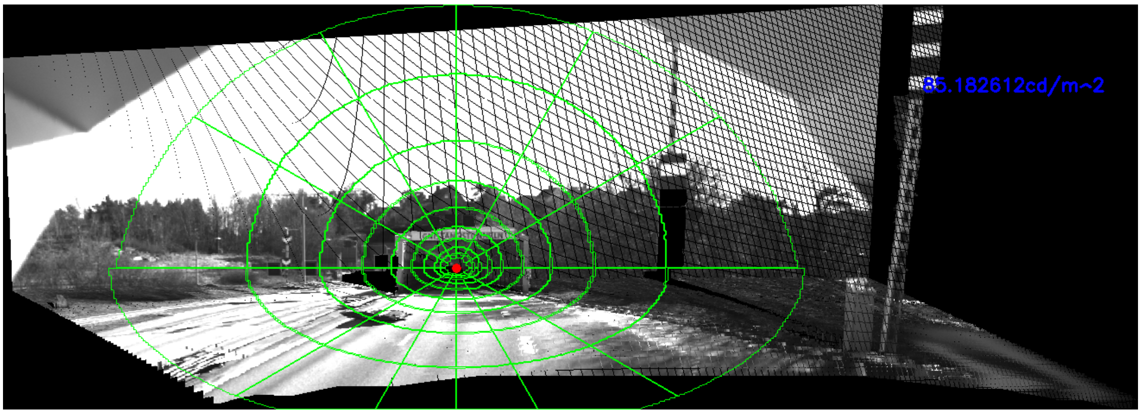


Figure 4.40: L_{seq} diagram shown in the warped view of the next brightest exposure image.

The first step in creating the L_{seq} diagram for the warped view is to estimate the height and width of the main region in the warped image. The main region is defined as the region in the image containing most of the pixels. In Figure 4.40 the main region has roughly the same height and width as the image, but in some cases, objects with vertical depth may be warped outside of the main region (especially if the depth for a region has been estimated incorrectly). In either case, the outline of the main region in the image is approximated from the outline mask M_O shown in Figure 4.37.

The width W_{reg} is simply found by approximating the region by a bounding rectangle and by setting the width of the region to the same width as the rectangle. The procedure for finding the region height is on the other hand slightly more complicated. First, the contour for the outline mask M_O is roughly approximated by a polygon. Then, between each pairs of polygon coordinates a line segment is created. The algorithm then finds the lines directly below and directly above the warped tunnel center coordinate. Then, the algorithm finds the highest point $\tilde{\mathbf{x}}_{top} = (x_{top}, y_{top})^T$ on the line segment above and the lowest point $\tilde{\mathbf{x}}_{bot} = (x_{bot}, y_{bot})^T$ on the line segment below. These two points are assumed to be the highest and lowest points in the image region and the height is found as the difference in the y-values, i.e. $H_{reg} = y_{bot} - y_{top}$.

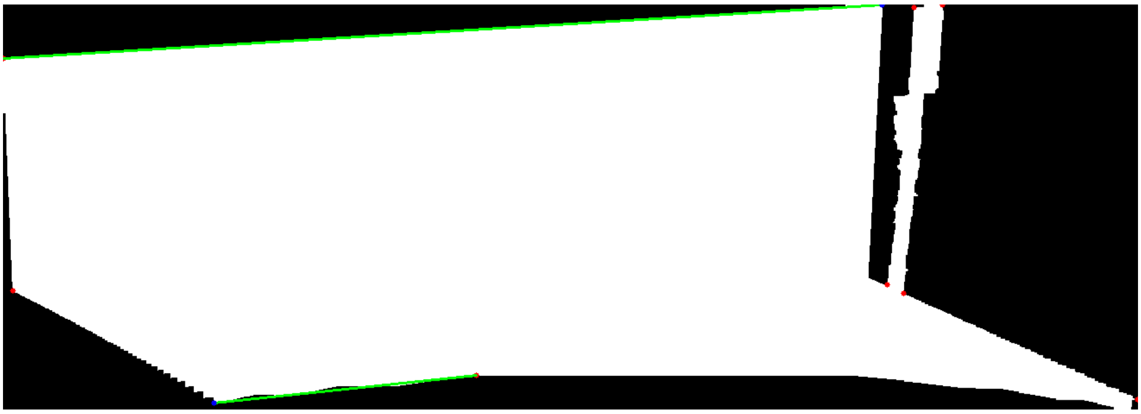
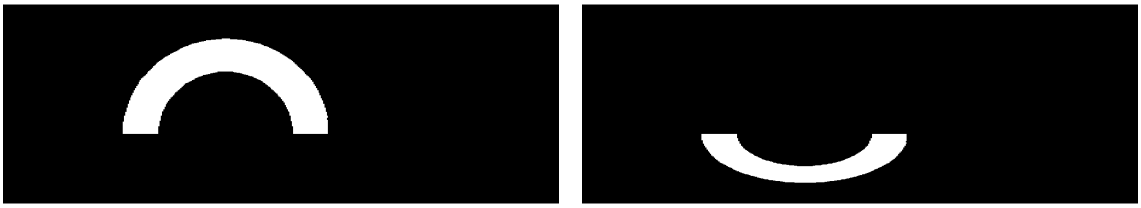


Figure 4.41: A visualization of the height estimation procedure. The red dots show the polygon coordinates and the blue dots show the points $\tilde{\mathbf{x}}_{top}$ and $\tilde{\mathbf{x}}_{bot}$.

Once the height and width of the main region in the warped image has been estimated, the next step is to create the masks defining the different subregions in the L_{seq} diagram. For this purpose, half-circle masks are used. The inner and outer radius of each half-circle mask is calculated using equations (3.65) and (3.66) as described in Section 3.7.1. In this case the region height H_{reg} is used rather than the image height H. Then, the upper and lower half-circle masks are scaled. The upper half-circle mask is scaled by $\frac{y_{tun}-y_{top}}{H_{reg}/2}$ in the y-direction and by W_{reg}/H_{reg} in the x-direction. The lower half-circle mask on the other hand, is scaled by $\frac{y_{bot}-y_{tun}}{H_{reg}/2}$ in the y-direction and by W_{reg}/H_{reg} in the x-direction. The reason why the scaling in the y-direction is different for the two masks is because the vertical distance between the top of the region and the center of the tunnel may differ from the vertical distance between the center of tunnel and the bottom of the region. The x-scaling is defined somewhat arbitrarily but it makes intuitive sense considering that it is almost the same as the aspect ratio for the warped image.

After the scaling, the masks have an elliptic shape as shown in Figures 4.42.



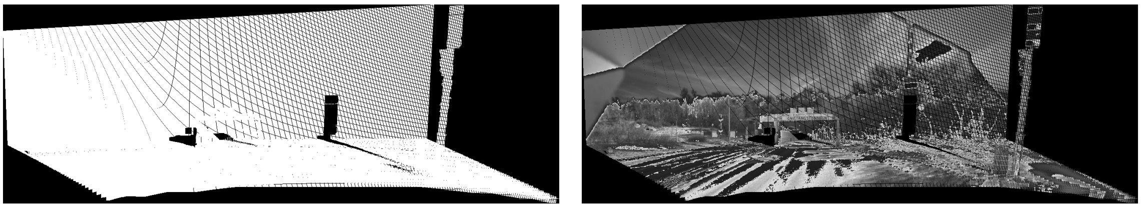
(a) Upper half-elliptic mask.

(b) Lower half-elliptic mask.

Figure 4.42: Two of the half-elliptic masks used when defining the subregions in the L_{seq} diagram.

4.14 Equivalent Veiling Luminance computation

Once the L_{seq} region for the warped view has been defined it is fairly straightforward to compute the L_{seq} value for the warped view. The brightest exposure image is first warped and the computed warped coordinates are re-used to warp the other exposure images. Finally, the luminance image is computed using the same method was described in Section 3.7 with the exception that only the points at which the warped image has defined values are included. Figure 4.43 shows the mask with the defined regions in the warped image and the visualization of the luminance image. The computed luminance value is shown in Figure 4.40.



(a) Mask showing the regions in the warped image that have defined pixel values.

(b) Visualization of the luminance image for the warped view.

Figure 4.43: Mask with defined regions in the warped image and the visualization of the luminance image for the warped view.

5

Data Gathering

The primary purpose of this chapter is to describe practical methods used in the development work and for data gathering. Section 5.1 describes the hardware (cameras) and software used in the project along with some of the different camera settings. In Section 5.2, the method used to calibrate the cameras is presented. The methods used in the measurement gathering (gathering of photos) is described in Section 5.3.

5.1 Camera hardware and software

A camera setup used during the development work consists of four components: A USB 3.0 monochrome (grayscale) machine vision camera, a wide-angle lens, a tripod and an optical filter. The optical filter halves the amount of light reaching the camera and is required for correct measurements of L_{seq}.

Unlike consumer cameras, the camera used does not have a Graphical User Interface (GUI) and is not powered by a battery. To power the camera and to control the camera settings it needs to be connected to a computer through a USB 3.0 port. The camera settings are mainly controlled through the FlyCapture software [57]. FlyCapture can be used to control several important settings such as the number of frames per second (fps), the brightness value and the exposure value. The exposure value setting together with the fps value control how long time the camera is exposed to light when capturing a photograph (known as the exposure duration or shutter time), which partially determines the brightness of the photo (a longer duration gives a brighter image and vice versa). An important mode available for the camera is HDR (High Dynamic Range). Using the HDR mode, the camera captures groups of 4 images where each image has an independent exposure value setting. This setting is essential for luminance measurements.

The focal length of a camera is determined by the lens [5]. In most cases (at least for consumer cameras) the camera lens is built-in and cannot be changed. In this case however the lens is separate from the camera sensor and consequently there is more flexibility in choosing a proper lens for the application.

The lens used for the development of the project has two important settings: focus and f-number, both of which are controlled by wheels on the lens. Focus determines the distance at which points in the image are the sharpest. Points that are not in focus tend to be blurrier. For the purposes of the measurement gathering the focus was set to ∞ since the camera needs to be focused on objects far away (such as a tunnel opening at the stopping distance). The second wheel on the lens controls the f-number, which is defined as

$$N = \frac{f}{d}, \tag{5.1}$$

where f is the focal length and d is the lens diameter (also known as the aperture) [5].

The common way to write the f-number is in the $f/\#$ format where f is the focal length and # is the aperture explicitly given as a number (usually in millimeters), for instance $f/1.4$, $f/2$, $f/2.8$ etc [5]. A decrease of the f-number by a factor of $\sqrt{2}$ has roughly the same effect as the an increase of the exposure duration by a factor of 2. Consequently, the f-number is another setting that controls the brightness of the image.

For this particular combination of camera and lens, the vertical field of view is approximately $\alpha_v = 47.4°$ and the horizontal field of view is approximately $\alpha_h = 72°$.

5.2 Camera Calibration

As mentioned in Section 3.1.3, the internal parameters of the camera are contained in the calibration matrix K [7]. The process of estimating K is known as camera calibration. Although there exist various approaches for estimating the calibration matrix based on for instance vanishing points, a very common method is to use a checkerboard pattern as shown in Figure 5.1. The checkerboard pattern was printed out on an A4 paper and taped on a flat solid surface. The checkerboard pattern consists of 10×7 squares (9×6 corners) where each square has $S = 2.6cm = 0.026m$ long sides.

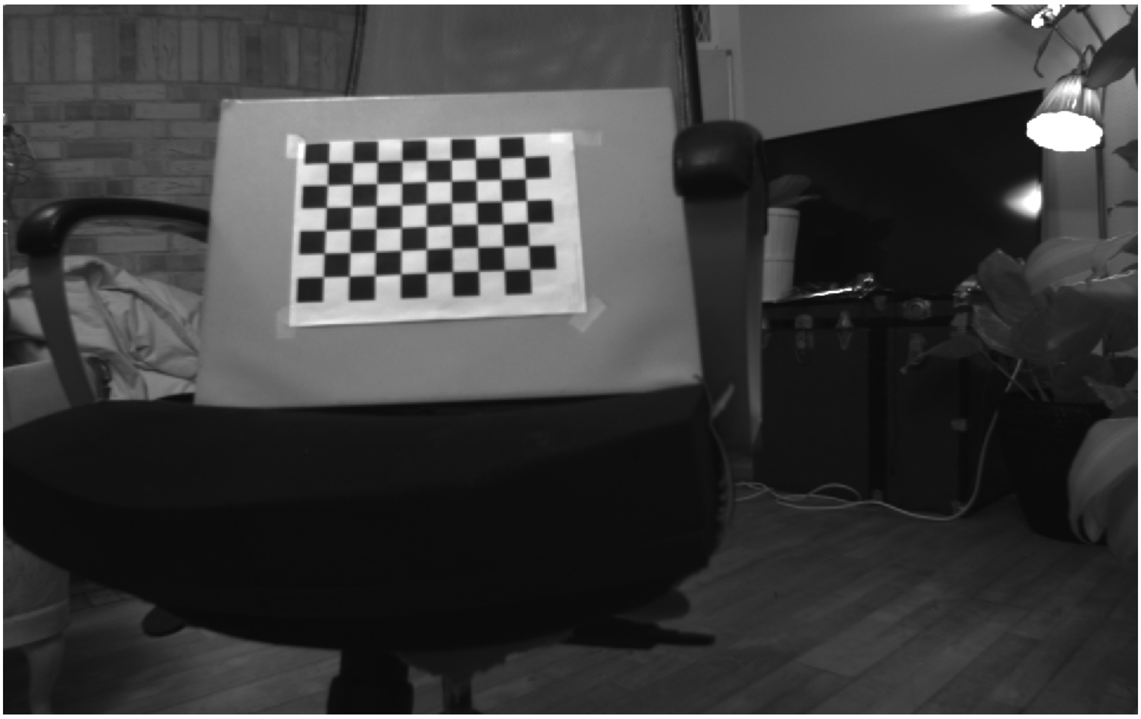


Figure 5.1: A checkerboard pattern attached to a solid flat surface.

The calibration was done using the *calibration.cpp* OpenCV sample code (see [58] for a thorough explanation of the code). This program requires a small number of images of the same calibration (checkerboard) pattern at different positions or with different orientations [58]. In theory only 2 images would be enough to calibrate the camera. Due to noise however, 10 or more images are recommended. Overall a total of 15 photos of the checkerboard pattern were captured, although some of the images were not good enough and were thus not included in the calibration.

The first step in the calibration is finding the inner corners of the checkerboard as shown in Figure 5.2 [58]. Then, the checkerboard points $\tilde{\mathbf{x}}$ are converted into 3D coordinates as $\tilde{\mathbf{X}} = (X, Y, Z)^T = (jS, iS, 0)^T$ where j is the corner number in the x-direction, i is the corner number in the y-direction and S is the length of one side of a square. Z is set to zero because the corners are located on the same plane and thus do not have a different depth (the calibration pattern surface is flat). Finally, the image coordinates and the corresponding 3D coordinates for the different images are used as input to the *calibrateCamera* OpenCV function [59].

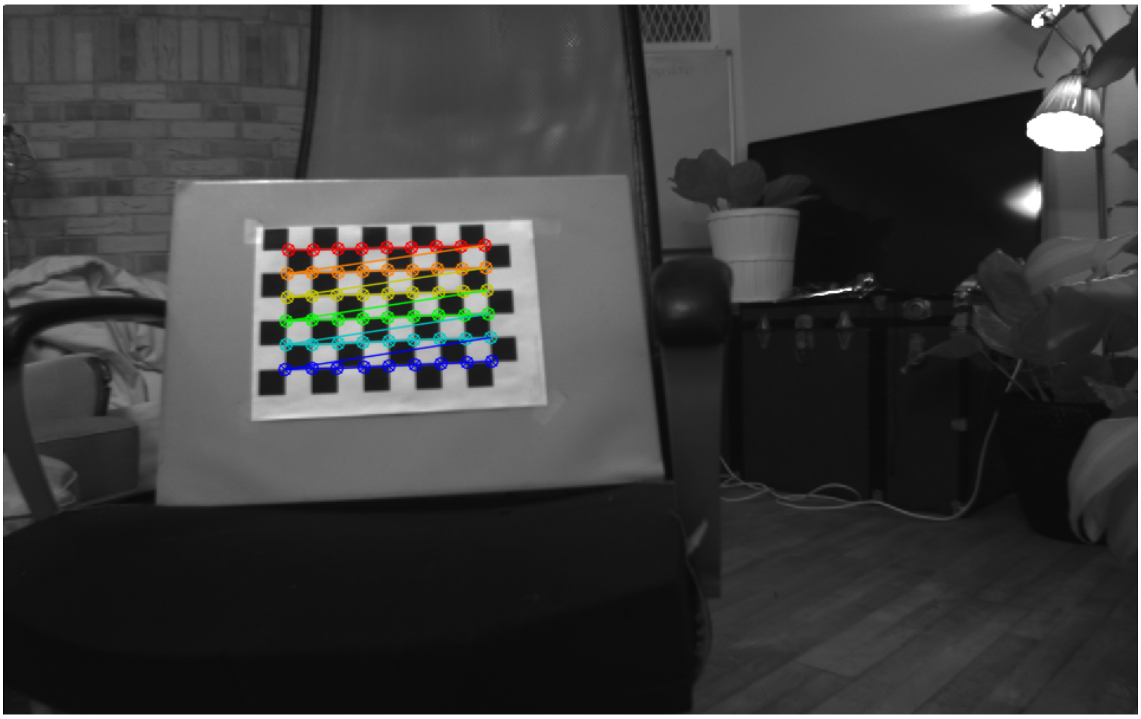


Figure 5.2: Detected corners in the checkerboard image.

The code sample was only slightly modified to allow for the assumption of non-square pixels, i.e. $f_x \neq f_y$. It was assumed that there is no skew, i.e. $s = 0$. This resulted in the following calibration matrix for one of camera used:

$$K = \begin{bmatrix} f_x & 0 & x_0 \\ & f_y & y_0 \\ & & 1 \end{bmatrix} = \begin{bmatrix} 685.2 & 0 & 476.8 \\ 0 & 685.9 & 308.8 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

There are a few interesting points to note about the estimated calibration matrix. Firstly, the principal point $\tilde{\mathbf{x}}_0 = (x_0, y_0)^T$ is roughly at the image center

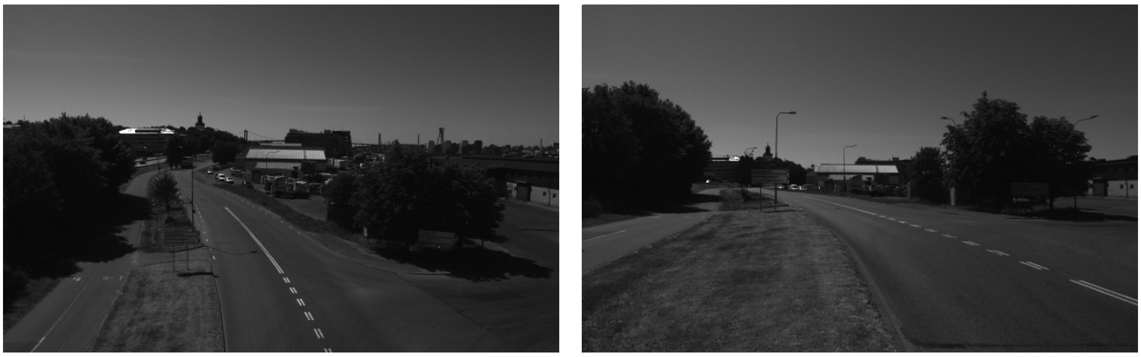
$\tilde{\mathbf{x}}_{mid} = (W/2, H/2)^T = (480, 300)^T$. This is typically the case for most cameras [5]. Furthermore, note that the ratio of the focal lengths is $a = \frac{f_y}{f_x} \approx 1.0009$, which means that the pixels are almost square. Finally, the focal length (assuming $f = f_x \approx f_y$) is approximately $f_m = fs_p = 685.2 \cdot 5.86 \cdot 10^{-5}m = 4.02mm$.

5.3 Measurement Gathering

An important but difficult part of the project was to gather relevant measurements needed to evaluate the result. Ideally the L_{seq} measurements would be gathered at a scene with a tunnel, with one camera viewing the tunnel from $5-7m$ above the road and possibly also with a horizontal displacement relative to the middle of the road in the direction of traffic and the other camera simultaneously capturing photos of the tunnel $1.5m$ above the road in the middle of the road in the direction of the traffic. Both cameras would be placed at the stopping distance from the tunnel. However, in practice such measurements are costly, difficult and impractical. There are only a few major road tunnels available in the Göteborg area and only a few of them can be photographed from above by a pedestrian. None of them however can be photographed at road level by a pedestrian (photographing the tunnel from a car is possible, albeit difficult). The closest example to a successful measurement of this kind for L_{seq} was done at the Gnistängstunneln site mentioned in Section 1.1 but even in that case the measurement was not ideal due to the 30 minute time difference between the measurements. To do an experiment of similar quality for other tunnel scenes would possibly require new installations. On the other hand, the camera system is going to be installed at new tunnel sites in the near future, which means that such data may become available for future comparisons.

In practice, the measurements were almost exclusively gathered at road scenes without tunnels. Where possible, images of the scene both from above the road and at road level were taken. In most cases however, only the view from above was easily available.

Most of the measurements were done for the purpose of the evaluation of the L_{seq} error reduction. This required both a view from above the road (usually at a height of more than 5 meters and with a pitch angle between 3 and 10 degrees) and a view at road level (ideally in the middle of the road but if not possible (due to traffic) then on the side of the road). The images were first gathered above the road and then at the road level. In both cases the cameras were to the extent possible centered on the same point in the scene. The images were captured when the scene was mostly free from traffic to avoid traffic affecting the L_{seq} values. Even though there was a time difference of several minutes between the photos of the scene above the road and the photos of the scene at road level, it can safely be assumed that this does not have a significant impact on the L_{seq} values as the weather conditions had not changed in any significant way. An example of a view from above and the corresponding road level view is shown in Figure 5.3.



(a) View of the scene above the road. (b) View at road level.

Figure 5.3: Two views of the Fiskholmsmotet scene (shown in [61]). The road level view is approximately directly below the view above the road. Both views are roughly centered on a truck in the scene.

Several metric measurements were also gathered for each view of a scene. The Nikon Forestry Pro laser measurement device [60] shown in Figure 5.4 was used for this purpose. It was primarily used for measuring the height of the camera above the road and sometimes also the horizontal distance to the point on which the camera was centered. In general, distances in a scene can be measured from some online maps, for instance at [62] or [63]. The pitch angle was estimated by reading the pitch axis scale on the tripod (both the pitch and yaw can be changed for the tripod and the roll can be changed by adjusting the legs of the tripod).

Figure 5.4: Laser measurement device Nikon Forestry Pro.

6
Results

In this chapter, the results of the proposed method to reduce the L_{seq} error are evaluated and analyzed. The chapter is divided into several sections as follows: In Section 6.1, an relative (percental) metric for the reduction of the L_{seq} error is proposed. The main results are presented in Section 6.2. This section contains a comparison between the L_{seq} errors for views above the road, views at road level and warped view. In Section 6.3, the effects of inpainting on the result are discussed. The influence of the depth map on the L_{seq} error reduction is investigated in Section 6.4. This section includes a comparison between reference depth maps manually created in The GNU Image Manipulation Program (GIMP) and the depth maps created automatically in the implementation. Finally, in Section 6.5, the computational performance of the proposed method is analyzed and discussed. The results presented in this chapter are further discussed in a broader way in Chapter 7.

6.1 Error Reduction Metric

In this section the metrics used in the evaluation of the L_{seq} error reduction are presented. The metrics are relative (percental) rather than absolute.
The first two metrics

$$E_{cam} = \frac{abs(L_{street} - L_{cam})}{L_{street}} \tag{6.1}$$

and

$$E_{warped} = \frac{abs(L_{street} - L_{warped})}{L_{street}} \tag{6.2}$$

measure the the percental error for the view from above the road and the warped view respectively. Here, L_{cam}, L_{street} and L_{warped} are the luminance values for the view above the road, for the road level view and for the warped view respectively. Based on these two metrics, the relative error is defined as

$$E_{rel} = \frac{E_{warped}}{E_{cam}}. \tag{6.3}$$

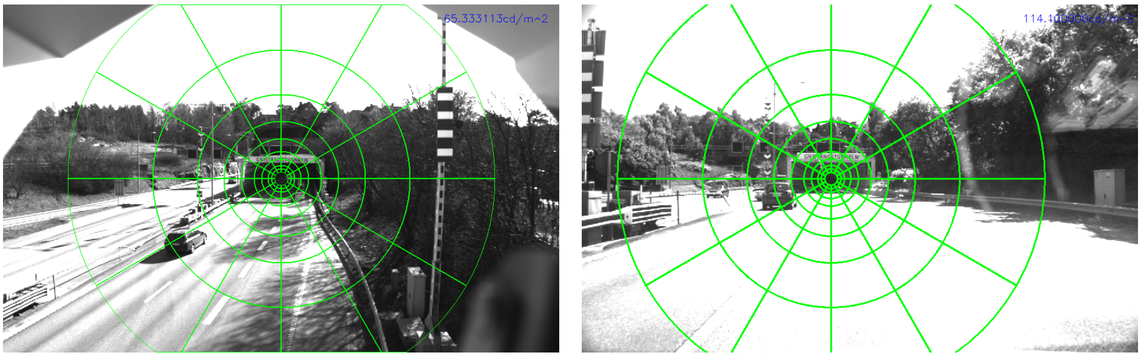
Finally, the percental error reduction is then given as

$$R = 1 - E_{rel}. \tag{6.4}$$

6.2 Reduction of Equivalent Veiling Luminance Error

The proposed error reduction metric is evaluated at five scenes, namely at the Gnistängstunneln scene (see [4]), at two locations in the Fiskholmsmotet scene (see [61]) and at two locations in the Lindholmen Stairs scene (see [64]). Ideally the metric should be evaluated at more scenes (and at different weather conditions) but this is the data that is currently available. Since the Gnistängstunneln scene represents the ideal scene for the application, it is the scene that is analyzed most thoroughly in the subsequent sections.

The two views of the Gnistängstunneln scene are shown in Figure 6.1 and the warped view is shown in Figure 6.2. Computing the errors gives $E_{cam} \approx 0.427 = 42.7\%$ and $E_{warped} \approx 0.253 = 25.3\%$. Thus, the resulting error reduction is $R \approx 0.407 = 40.7\%$. The resulting error of the homography-based warping approach mentioned in Section 1.1 and shown in Figure 1.7 is $E_{warped,hom} \approx 0.255 = 25.5\%$ and the error reduction is thus $R_{hom} \approx 0.403 = 40.3\%$. Thus, there is only a minor difference between the two methods but the homography-based warping method has severe artifacts/distortions. The original source code for the homography-based warping method is missing and as a consequence the method will not be evaluated on the other data. In any case, note that the L_{seq} diagram in both warped views differs somewhat compared to the L_{seq} diagram in the street view. Either way, in both warped views there is a higher proportion of road and sky, which is probably the main cause of the error reduction. The error could probably be reduced further by taking into account the Bidirectional Reflectance Distribution Function (BRDF) for asphalt.



(a) View of the Gnistängstunneln scene above the road. Here the height is $h \approx 7m$

(b) View of the Gnistängstunneln scene at road level.

Figure 6.1: Two views of the Gnistängstunneln scene. For the view above the road, $L_{seq} \approx 65.33cd/m^2$, and for the road level view, $L_{seq} \approx 114.10cd/m^2$. Note that there is a large time difference between the two images, which may overestimate or underestimate the differences in L_{seq}.

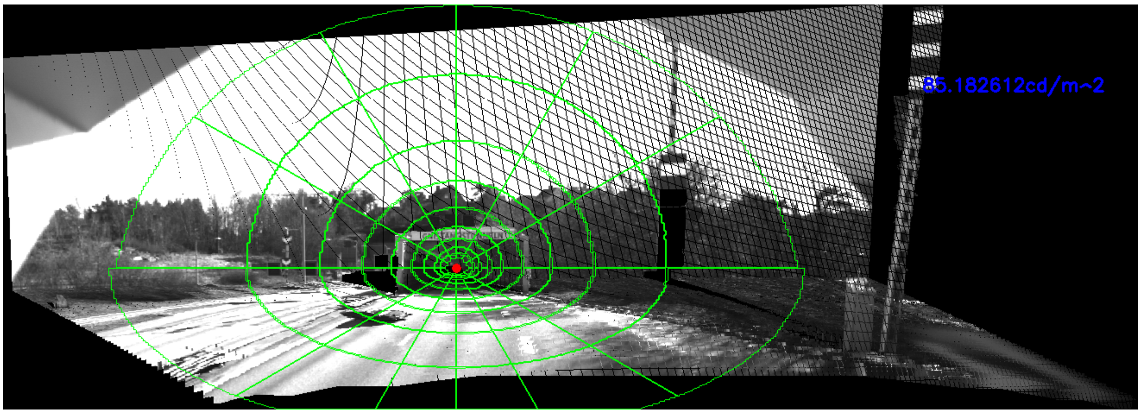
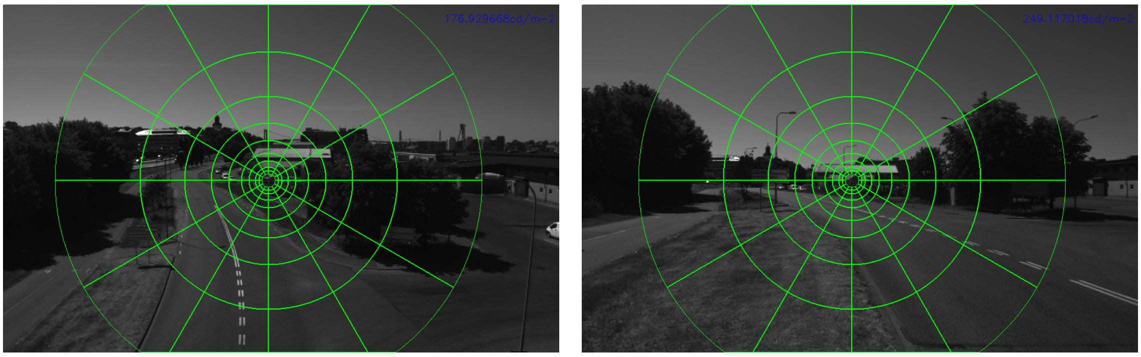


Figure 6.2: L_{seq} in warped view for the Gnistängstunneln scene. Here, $\tilde{\mathbf{c}} = (t_x, t_y, t_z)^T = (-7.5, 5.5, 0.0)^T$ and $\boldsymbol{\theta} = (\theta_x, \theta_y, \theta_z)^T = (3.78°, 6.29°, 0.0°)^T$. The measured luminance is $L_{seq} \approx 85.18 cd/m^2$.

The second scene in the evaluation is referred here to as Fiskholmsmotet Mid. In this case the images were first captured above the road, slightly right of the middle and then at road level on the left side. The two views are shown in Figure 6.3 and the corresponding warped view is shown in Figure 6.4. In this case the errors are $E_{cam} \approx 0.289 = 28.9\%$ and $E_{warped} \approx 0.061 = 6.1\%$. The error reduction is thus $R \approx 0.788 = 78.8\%$. Note that in this case the L_{seq} value for the warped view is overestimated. This is mainly because the L_{seq} region is not centered correctly. The centering of the L_{seq} region in the warped view depends on the depth value used when warping the point at which the L_{seq} diagram is centered in the original view. The depth value is set to the distance to the center point D. Here $D = 73$ was used (which is close to the actual distance). However, by using for instance $D = 100$, the L_{seq} region is closer to being centered correctly as shown in Figure 6.5. Note that in this case the L_{seq} value is not overestimated but the error reduction has decreased to $R_{D=100} = 0.547 = 54.7\%$. A probable cause of this error is that the field of view depth map does not take into account occlusions. Therefore, although the distance to the truck on which the L_{seq} diagram is centered on may be $D = 73$, the depth value that should be used could instead be the absolute distance on the ground plane not considering occlusions (i.e. the distance assuming there would not be a truck in the scene). The error in the centering could also be caused by errors in the depth map (for instance if the pitch angle θ_x is underestimated then the Z values in the scene are overestimated). In summary, it is important to center the diagram correctly in the warped view to avoid overestimating the L_{seq} value.



(a) View above the road ($h \approx 9.2m$) in the Fiskholmsmotet Mid scene.

(b) View at road level in the Fiskholmsmotet Mid scene.

Figure 6.3: Two views of the Fiskholmsmotet Mid scene. For the view above the road, the luminance is $L_{seq} \approx 176.93cd/m^2$ and for the road level view the luminance is $L_{seq} \approx 249.12cd/m^2$.

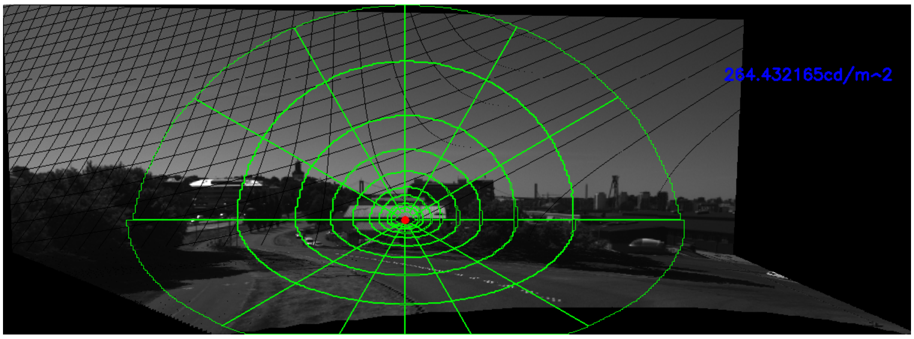


Figure 6.4: L_{seq} in the warped view for the Fiskholmsmotet Mid scene. Here the translation and rotation was approximated to be, $\tilde{\mathbf{c}} = (t_x, t_y, t_z)^T = (-11.0, 7.6, 0.0)^T$ and $\boldsymbol{\theta} = (\theta_x, \theta_y, \theta_z)^T = (3.5°, -2.0°, 0.0°)^T$. The measured luminance is $L_{seq} \approx 264.43cd/m^2$. In this case the luminance value is overestimated compared to the road level luminance value which is $L_{seq} \approx 249.12cd/m^2$.

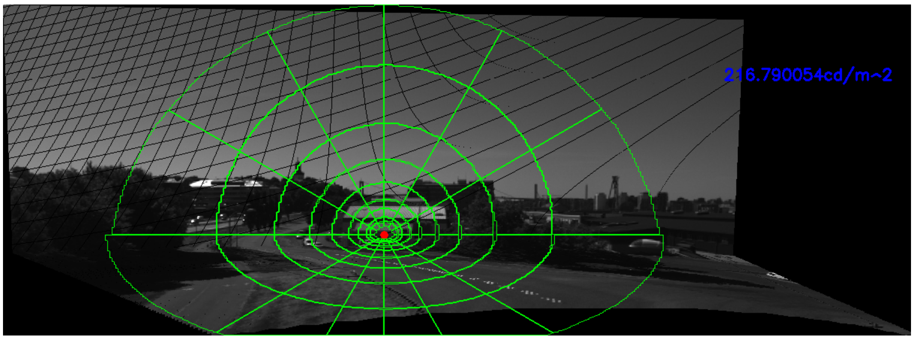
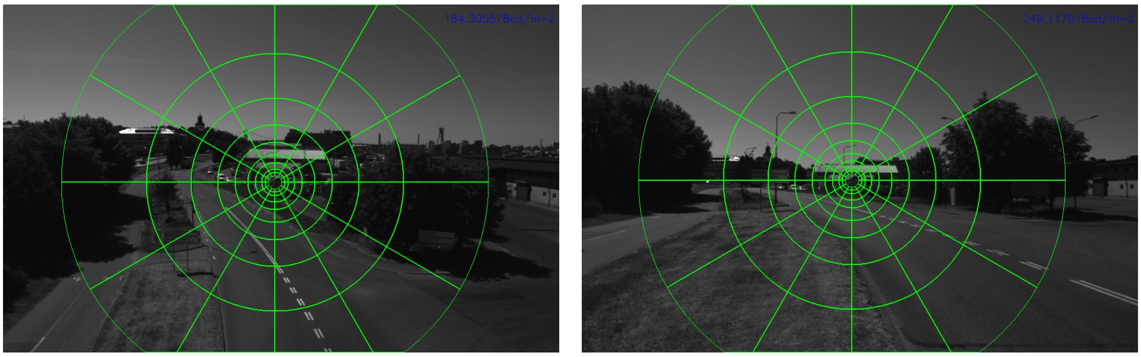


Figure 6.5: L_{seq} in the warped view for the Fiskholmsmotet Mid scene when setting $D = 100$ instead of $D = 73$. The measured luminance is $L_{seq} \approx 216.79cd/m^2$.

The third scene analyzed in this section is the Fiskholmsmotet Left scene. As shown in Figure 6.6, the images used in this comparison mostly differ by vertical translation; the view above the road is almost directly above the view at road level. The warped view is shown in Figure 6.7 (note that $D = 100$ was used in this case). Here the errors are $E_{cam} \approx 0.260 = 26.0\%$ and $E_{warped} \approx 0.106 = 10.6\%$. The error reduction is thus $R \approx 0.594 = 59.4\%$.



(a) View of the Fiskholmsmotet Left scene from above the road (at $h \approx 9.2m$).

(b) View at road level in the Fiskholmsmotet Left scene.

Figure 6.6: The two views of the *Fiskholmsmotet Left* scene. For the view above the road, the luminance is $L_{seq} \approx 184.31cd/m^2$ and for the road level view the luminance is $L_{seq} \approx 249.12cd/m^2$.

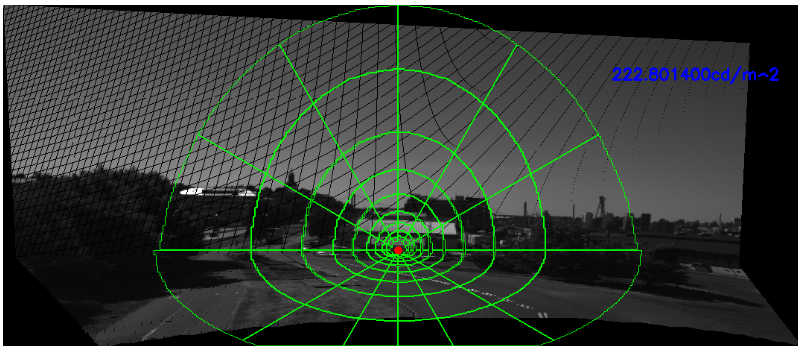
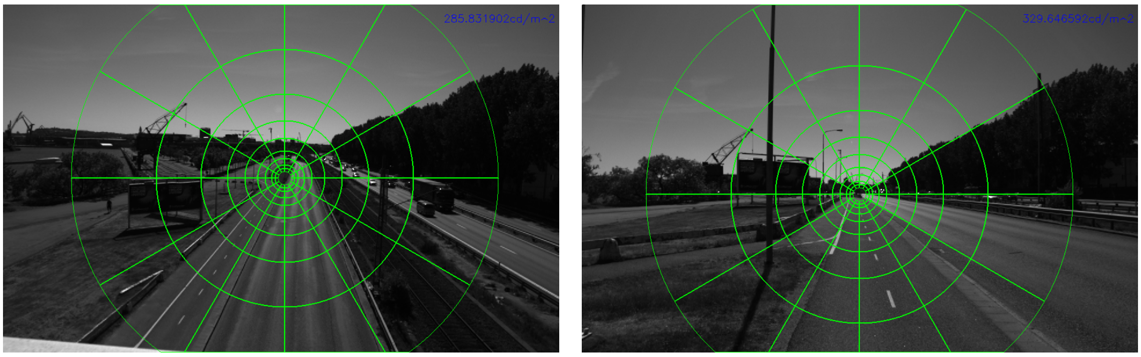


Figure 6.7: L_{seq} in the warped view for the Fiskholmsmotet Left scene. Here the translation and rotation is approximately $\tilde{\mathbf{c}} = (t_x, t_y, t_z)^T = (-4.0, 7.6, 0.0)^T$ and $\boldsymbol{\theta} = (\theta_x, \theta_y, \theta_z)^T = (4.0°, -5.0°, 0.0°)^T$. The measured luminance is $L_{seq} \approx 222.80cd/m^2$.

The fourth scene included in this evaluation is the Lindholmen West scene, shown in Figure 6.8 and the corresponding warped view is shown in Figure 6.9. Here, $E_{cam} \approx 0.133 = 13.3\%$, $E_{warped} \approx 0.059 = 5.9\%$ and $R \approx 0.554 = 55.4\%$. Note that the centering of the L_{seq} diagram in the warped view is slightly wrong but despite this, the L_{seq} diagram approximates the L_{seq} diagram for the road level view quite well.



(a) View of the Lindholmen West scene from above the road. The height above the road surface is $h \approx 8.4m$.

(b) View at road level in the Lindholmen West scene.

Figure 6.8: The two views of the Lindholmen West scene. For the view above the road, the luminance is $L_{seq} \approx 285.83cd/m^2$ and for the road level view the luminance is $L_{seq} \approx 329.64cd/m^2$.

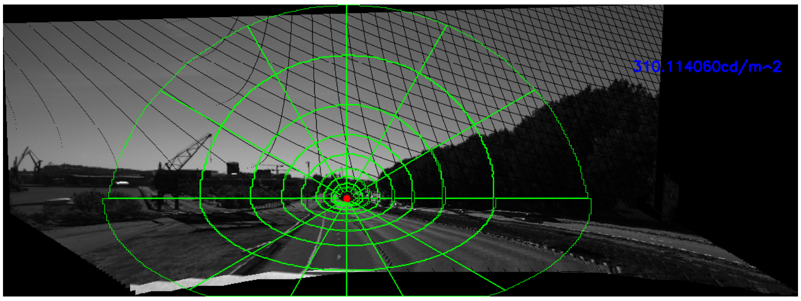
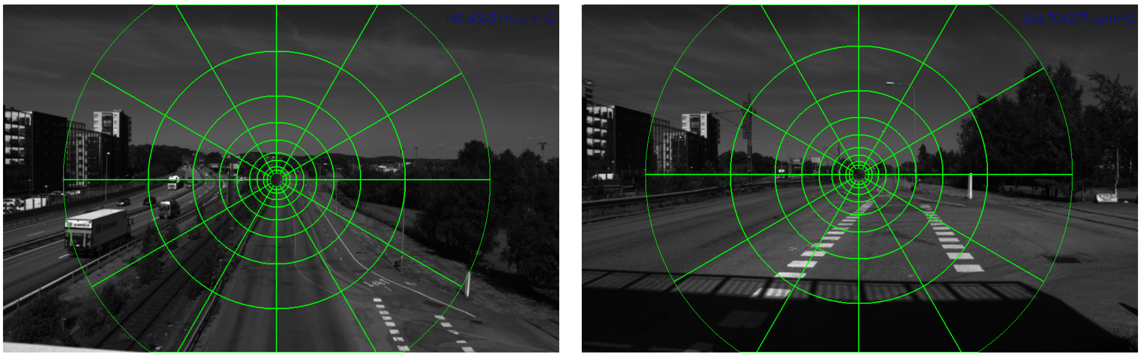


Figure 6.9: L_{seq} for the warped view of the Lindholmen West scene. Here the translation and rotation is approximately $\tilde{\mathbf{c}} = (t_x, t_y, t_z)^T = (-7.2, 6.8, 0.0)^T$ and $\boldsymbol{\theta} = (\theta_x, \theta_y, \theta_z)^T = (3.5°, 3.0°, 0.0°)^T$ respectively. The measured luminance is $L_{seq} \approx 310.11cd/m^2$.

Finally, the result is also presented for the Lindholmen East scene. The two original views of the scene are shown in Figure 6.10 and the result after warping is shown in Figure 6.11. In this case, the errors are $E_{cam} \approx 0.133 = 13.3\%$ and $E_{warped} \approx 0.019 = 1.9\%$. The error reduction is as high as $R \approx 0.852 = 85.2\%$. Note however that the warped view contains some artifacts due to errors in the estimated depth map.



(a) View of the Lindholmen East scene from above the road. In this case the height above the road surface is $h \approx 8.4m$.

(b) View at road level in the Lindholmen West scene.

Figure 6.10: Two views of the Lindholmen East scene. The luminance values are $L_{seq} \approx 195.44cd/m^2$ and $L_{seq} \approx 224.70cd/m^2$ for the view above the road and the view at road level respectively.

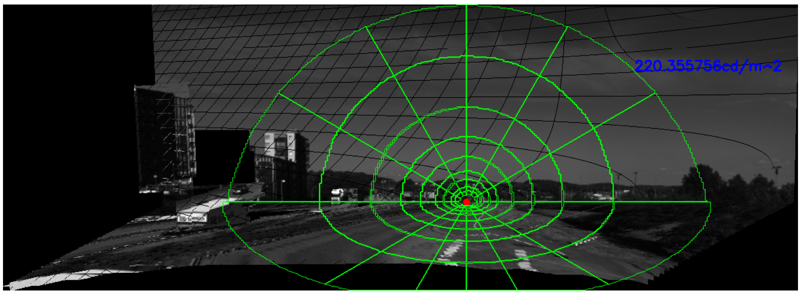


Figure 6.11: L_{seq} for the warped view of the Lindholmen East scene. The translation is approximately $\tilde{\mathbf{c}} = (t_x, t_y, t_z)^T = (8.5, 6.8, 0.0)^T$ and the rotation is approximately $\boldsymbol{\theta} = (\theta_x, \theta_y, \theta_z)^T = (3.0°, -1.5°, -0.5°)^T$ respectively. The measured luminance is $L_{seq} \approx 220.36cd/m^2$.

The results of the comparison are summarized in Table 6.1.

Scene	L_{cam}	L_{street}	L_{warped}	E_{cam}	E_{warped}	R
Gnistängstunneln	65.33	114.10	85.18	42.7%	25.3%	40.7%
Fiskholmsmotet Mid	176.93	249.12	264.43	28.9%	6.1%	78.8%
Fiskholmsmotet Left	184.31	249.12	222.80	26.0%	10.6%	59.4%
Lindholmen West	285.83	329.64	310.11	13.3%	5.9%	55.4%
Lindholmen East	195.44	224.70	220.36	13.3%	1.9%	85.2%

Table 6.1: Summary of the L_{seq} values (in cd/m^2) and the corresponding error values and error reduction metric for different scenes.

6.3 Effect of Inpainting

As previously mentioned, inpainting is an optional step in the pipeline. It is mostly a way to improve the visual quality. However, it is of interest to also investigate how inpainting affects the L_{seq} error reduction and if it is worth the effort. The effect of inpainting on the L_{seq} value is thus analyzed for the scenes presented in Section 6.2. The inpainted versions of the warped views for the Gnistängstunneln scene and the scenes at Fiskholmsmotet and Lindholmen are shown in figures 6.12 to 6.16. A comparison of the L_{seq} values and the error reduction metrics R for the inpainted warped views and the warped views with no inpainting is presented in Table 6.2. Overall it can be noted that the L_{seq} values are very similar and thus that the rates of error reduction are about the same. In those cases where the result has improved, it is an improvement of $\leq 1\%$. Considering the extra computational cost of inpainting (which is discussed in Section 6.5), it is certainly not worth the effort.

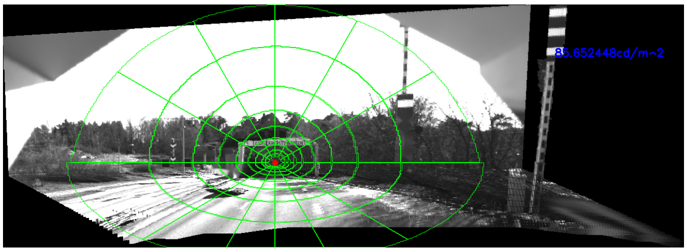


Figure 6.12: Inpainted version of the warped view at the Gnistängstunneln scene. Here the measured luminance is $L_{seq} \approx 85.65 cd/m^2$.

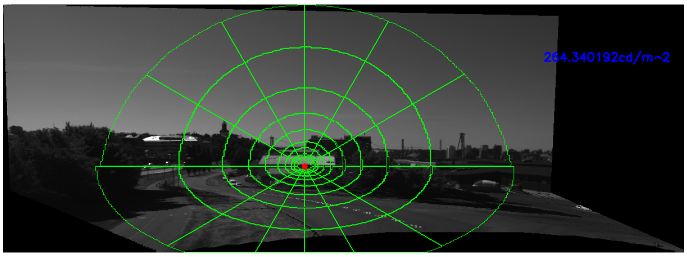


Figure 6.13: Inpainted version of the warped view at the Fiskholmsmotet Mid scene. For the inpainted image the measured luminance is $L_{seq} \approx 264.34 cd/m^2$.

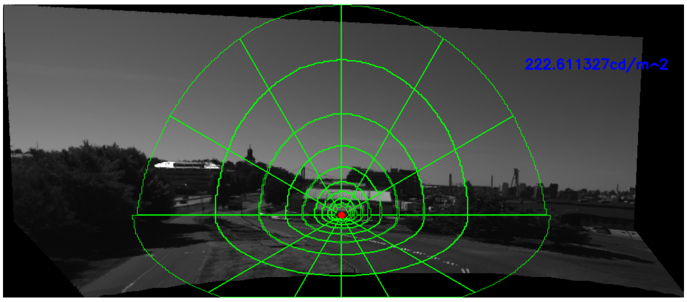


Figure 6.14: Inpainted version of the warped view at the Fiskholmsmotet Left scene. The measured luminance is $L_{seq} \approx 222.61 cd/m^2$.

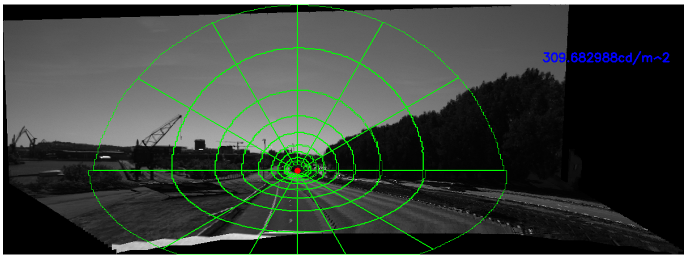


Figure 6.15: Inpainted version of the warped view at the Lindholmen West scene. In this case the measured luminance is $L_{seq} \approx 309.68 cd/m^2$.

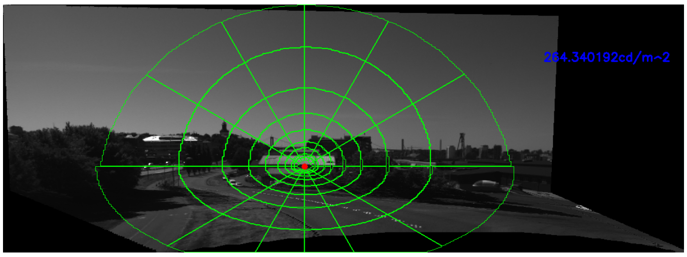


Figure 6.16: Inpainted version of the warped view at the Lindholmen East scene. $L_{seq} \approx 220.34 cd/m^2$ in this case.

Scene	$L_{no\ inpaint}$	$L_{inpaint}$	$R_{no\ inpaint}$	$R_{inpaint}$
Gnistängstunneln	85.18	85.65	40.7%	41.7%
Fiskholmsmotet Mid	264.43	264.34	78.8%	78.9%
Fiskholmsmotet Left	222.80	222.61	59.4%	59.1%
Lindholmen West	310.11	309.68	55.4%	54.4%
Lindholmen East	220.36	220.34	85.2%	85.0%

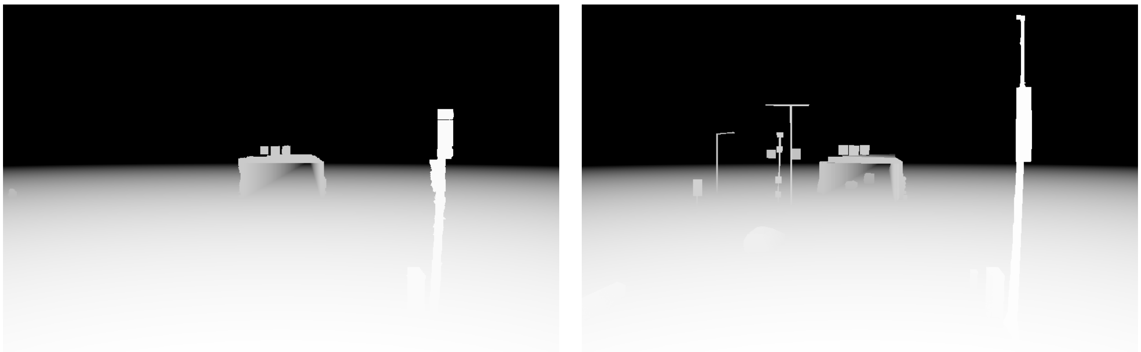
Table 6.2: Comparison of L_{seq} values and rates of error reduction for the warped views with or without inpainting. The inpainting radius chosen was 3.

6.4 Effect of depth map

The most demanding and complicated part of the implementation is the depth map estimation. It is therefore important to analyze the effect that a depth map has on the warped view and the resulting error reduction. This is done by comparing the estimated depth map with a reference depth map for some scenes and also by changing the resolution of the depth map (the number of distinct depth levels). As previously mentioned, the reference depth maps were created manually in GIMP. This was done based on the field of view depth map. The reference depth maps do not contain the correct depth for all of the objects in the scene as the depth can in some cases be quite difficult to define manually, besides, the proposed depth estimation method is not intended to recover the depth of objects such as trees or terrain that have more ambiguous depth.

6.4.1 Comparison with reference depth maps

A comparison of the proposed depth map resulting from the proposed depth estimation method and the reference depth map for the Gnistängstunneln scene is shown in Figure 6.17. From the figure it can be noted that the proposed depth estimation method fails to recover the depth of several vertical objects in the scene such as the lamps and the sign on the left side of the image. Furthermore, the depth for the pole was not correctly recovered using the proposed method.



(a) The depth map created for the Gnistängstunneln scene using the proposed depth estimation method.

(b) Reference depth map for the Gnistängstunneln scene.

Figure 6.17: The depth maps for the Gnistängstunneln scene. $Z_{far} = 250$ for the two depth maps.

The warped view using the reference depth map is shown in Figure 6.18. In this case the error reduction has increased from $R \approx 40.7\%$ to $R \approx 45.8\%$. Note however that this reduction of the error is probably mostly due to the larger L_{seq} region in the this image compared to the warped image based on the estimated depth map. This implies that the current method of finding the L_{seq} region in the warped image needs to be replaced by a better one. Regardless, the increase in the reduction of the L_{seq} error is relatively minor.

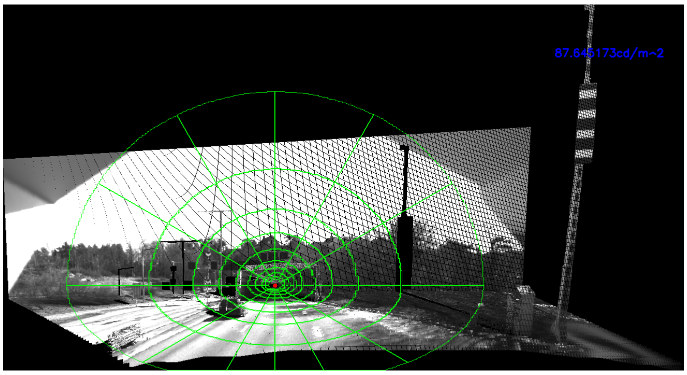
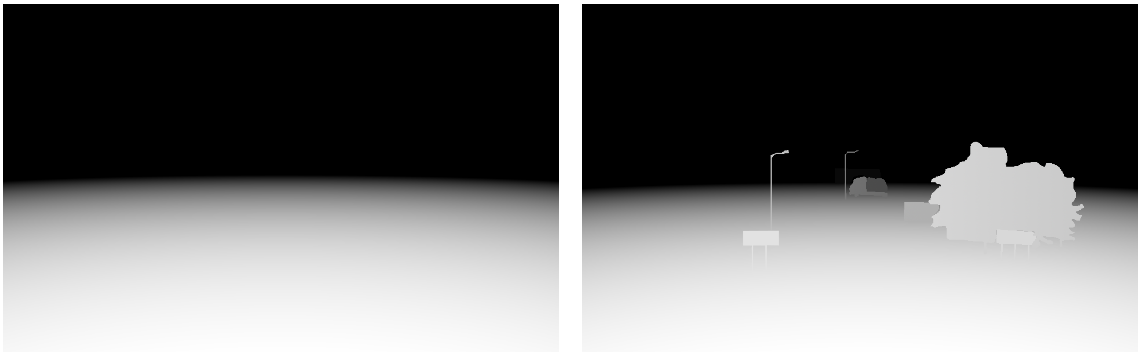


Figure 6.18: Warped view of the Gnistängstunneln scene that results when using the reference depth map. Here the luminance is $L_{seq} \approx 87.65cd/m^2$

A reference depth map is also available for the Fiskholmsmotet Left scene. The two depth maps (estimated and reference) are shown in Figure 6.19. It can be noted that the proposed depth estimation method does not find any vertical depth region. This is because the MSER parameters were set in such a way as to avoid false detections. It is probably possible to tweak the parameters in order to obtain a better depth map estimation but the need to tweak (many) parameters highlights a weakness in the method.



(a) The depth map created for the Fiskholmsmotet Left scene using the proposed depth estimation method.

(b) Reference depth map for the Fiskholmsmotet Left scene.

Figure 6.19: The depth maps for the Fiskholmsmotet Left scene. In this case $Z_{far} = 150$.

As shown in Figure 6.20, the estimated L_{seq} value using the reference depth map is slightly lower than the L_{seq} value shown in Figure 6.7. The L_{seq} error reduction is $R_{true,warped} \approx 42.7\%$ when using the reference depth map compared to $R_{warped} \approx 59.4\%$ when using the estimated depth map. Although the proposed depth estimation method yields a better result in this case, it might not always be the case. It is still a good idea to use an as accurate depth map as possible to get a more accurate view transformation.

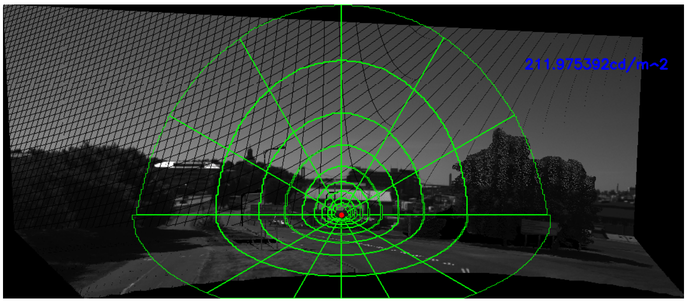


Figure 6.20: Warped view of the Fiskholmsmotet Left scene using the reference depth map. Here $L_{seq} \approx 211.98 cd/m^2$.

The final scene for which a reference depth map is available is the Lundbytunneln scene (see [65]). For this scene only the view above the road shown in Figure 6.21 is available and thus no comparison of the L_{seq} values or the error reduction can be done. However, it is still important to compare the depth maps considering that this scene contains a tunnel and is thus the type of scene that is important for the application. As shown in Figure 6.22 there is a major difference in quality between the estimated depth map and the reference depth map. The proposed depth estimation method has several flaws in this case.

The first flaw is that the method works poorly when it comes to finding large structural elements in the scene. In this case there are walls on both sides of the road, the tunnel opening is directly cut into the mountain and there is terrain and vegetation at a significant height above the road surface. The second flaw is that the the method does not yet handle the case when there are two tunnel openings and two clearly separated road regions. Finally, the tunnel opening shape is a combination of a rectangular part and a semi-circular part but the current implementation only works for rectangular tunnel shapes. For this specific scene some user interaction can be useful. For instance, each wall can be defined by a polygon with four points and the depth can be drawn using vertical lines as was done for when defining the depth for the tunnel walls (refer back to Section 4.8.2).

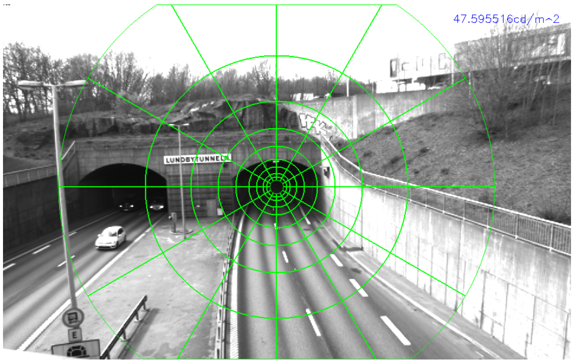
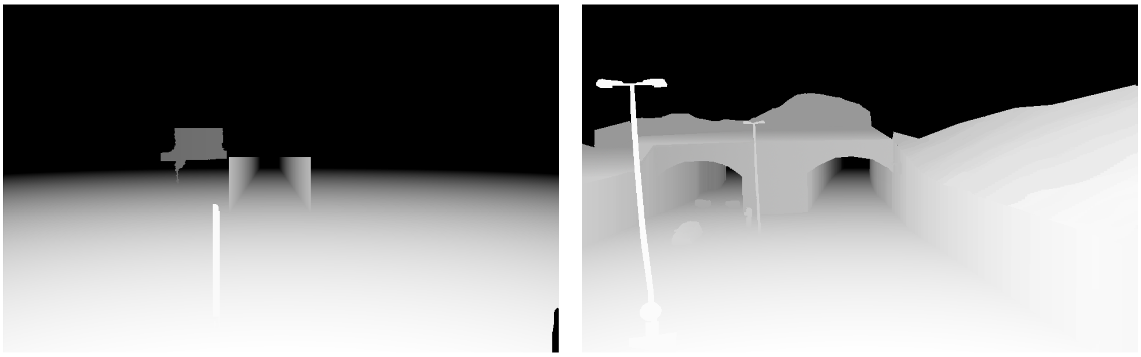


Figure 6.21: View of Lundbytunneln above the road. The measured luminance is $L_{seq} \approx 47.60 cd/m^2$



(a) The depth map created for the Lundbytunneln scene using the proposed depth estimation method.

(b) Reference depth map for the Lundbytunneln scene.

Figure 6.22: The depth maps for the Lundbytunneln scene. In this case $Z_{far} = 150$.

The resulting warped views are shown in figures 6.23 and 6.24. There is a major difference in quality between the two warped views. Although the road level L_{seq} value is unknown, it is much more probable that the warped view using the reference depth map gives a closer result (or at least the overestimation of the L_{seq} value can be avoided). After all, the L_{seq} diagram in Figure 6.23 is centered above the tunnel opening (due to the faulty depth estimation) instead of being centered at a point inside the tunnel opening as in Figure 6.24.

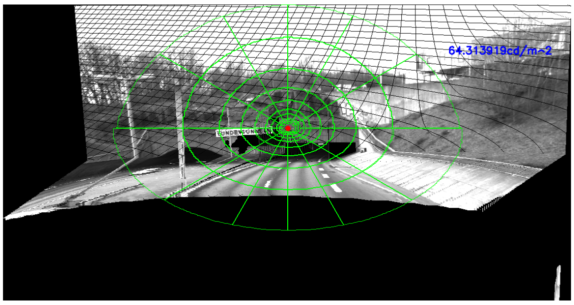


Figure 6.23: Warped view of Lundbytunneln (using estimated depth map). Here, $\tilde{\mathbf{c}} = (t_x, t_y, t_z)^T = (6.8, 7.7, 0.0)^T$ and $\boldsymbol{\theta} = (\theta_x, \theta_y, \theta_z)^T = (5.0°, -1.5°, -1.0°)^T$. The measured luminance is $L_{seq} \approx 64.31 cd/m^2$.

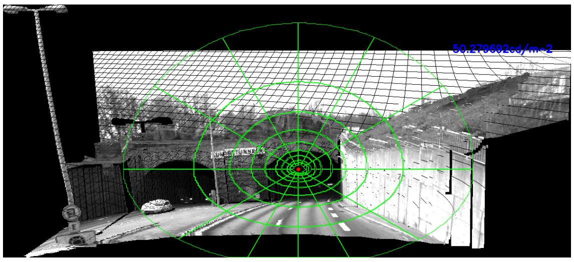


Figure 6.24: Warped view of Lundbytunneln (using the reference depth map). The measured luminance is $L_{seq} \approx 50.28 cd/m^2$.

6.4.2 Changing the resolution of the depth map

The warped views presented so far all use depth maps with $N = 2^8 = 256$ unique values stored in the look-up table. This is a convenient value to use considering that a depth map typically uses 8 bits and 8-bit images are easy to visualize. It is entirely possible to use a higher depth resolution (higher value of N) but the depth map needs to be stored in a 16-bit format, thus doubling the storage requirement. As shown in Table 6.3, this is generally not worth it. In most cases a higher N gives a worse result.

N	L_{seq}	R
256	85.18	40.7%
512	84.78	39.9%
1024	85.47	41.3%
2048	84.74	39.8%

Table 6.3: Influence of depth resolution N on L_{seq} and on the error reduction R.

6.5 Computational Performance

Another important aspect in the evaluation of the implementation is the computational performance. Currently, the camera used in the final product runs at 70 fps, which means it captures series of four images with different exposure times 17.5 times per second. The luminance is computed for each series of four images. Under the assumption that the computer in the system only spends its' computational power on warping the view and evaluating the L_{seq} value in the warped view, the available computation time is $1s/17.5 \approx 0.057s = 5.7ms$. However, since the system uses traffic compensation it means that a depth map for a scene can be computed just once and be re-used when warping the images. Furthermore, as discussed in chapter 7, an alternative is to warp back the L_{seq} diagram defined in the warped view back to the original view. In any case, the computational performance for the current implementation is evaluated and analyzed in this section.

In total the time required to run the program for the Gnistängstunneln scene is approximately $4.630s$. Note however that the program contains many redundant parts and has not been optimized. What is more interesting than absolute value (which will differ depending on the computer hardware anyway) is the computation time for specific parts of the implementation.

The computational time (both in seconds and percentage of total time) for most parts of the program are listed in Table 6.4 (for a detailed description of the different parts in the implementation refer back to chapter 4). In total the parts listed correspond to 90.93% of the program running time. The depth map estimation takes in total around $3.671s$ and corresponds to 79.29% of the running time. Once the depth map for the scene has been estimated, the warping and the L_{seq} computation in the warped image (including defining the L_{seq} region in the warped image) takes $0.36s = 36ms$. This is only 7.7% of the computational time. Although the time required for warping, defining the L_{seq} region and computing the luminance exceeds the desired $5.7ms$ (of course, the computation time will differ when using the hardware in the product), the computation can be simplified by warping back the luminance region to the original view as discussed in chapter 7.

Also note that the values presented in Table 6.4 are for the case of no inpainting. When inpainting is used the computation time increases to around $40s$, which combined with the very minor difference in L_{seq} value makes inpainting entirely redundant.

Part of the implementation	Computation time (s)	Computation time (%)
Luminance for original and road view	0.174	3.76%
Line segment detection	0.272	5.88%
MSAC	0.050	1.08%
MSER and bounding rectangle filtering	0.594	12.85%
HFS	0.642	13.87%
Sky & Road Region Detection	0.101	2.19%
Finding vertical depth regions	1.609	34.74%
Finding tunnel outline	0.280	6.05%
Depth sets and refinement	0.015	0.33%
Finding tunnel walls	0.034	0.73%
Field of view depth	0.050	1.07%
Adding tunnel wall depth	0.005	0.10%
Adding depth for other regions	0.019	0.42%
Warping first image and creation of masks	0.053	1.14%
Warping other images	0.039	0.84%
Warped luminance image	0.024	0.52%
L_{seq} region in warped image	0.243	5.33%
L_{seq} computation in warped image	0.001	0.03%
Total	4.630	100%

Table 6.4: Computation time for different parts in the code.

7 Discussion and Conclusion

In this report a method of L_{seq} error reduction for tunnel scenes is proposed. The proposed method uses Depth Image Based Rendering to warp the original view of the scene above the road, down to a virtual road level view of the scene. Warping using Depth Image Based Rendering requires two main components: the camera pose for the virtual view and a depth map for the original view. The camera pose can be obtained by measurements in the scene but the depth map is usually far harder to obtain. Since a depth map for a specific scene is rarely available, a monocular depth estimation method based on depth from field of view and vertical region detection has also been proposed and evaluated.

The method was evaluated on different scenes and the L_{seq} error reduction was more than 40%. However, the proposed approach has several flaws, including occasional major errors in the depth map estimation (compared to ground truth depth maps), flaws in the L_{seq} diagram in warped views and a long computation time. It is therefore important to discuss possible ways to improve the current implementation and also alternative approaches to solving the problem.

7.1 Improvements in the depth map estimation

The proposed depth estimation method has several flaws, including the need to tune many parameters, sensitivity to false detections, failure to recover more structural scenes and computational complexity. Several potential ways to improve the depth map estimation have already been mentioned in the previous chapters.

Currently only the MSERs detected in the next darkest exposure image are used when finding potential vertical depth regions in the scene. By combining the MSERs detected in all of the images it would be possible not only to detect more vertical depth regions but also to reduce the number of false detections.

The proposed region classification method based on line histograms is also flawed since it fails to detect many of the regions with vertical depth and has a high number of misclassifications. More experimentation can be done to improve the region classification. The misclassifications are however partially also caused by incorrect segmentation of the scene using HFS, leading to incorrect road region and sky masks. This can be partially solved using fine-tuning of HFS and MSER parameters but both of these methods are difficult to tune since they require setting many parameters.

Once a region has been classified as "Vertical" it is segmented into foreground and background using GrabCut. Currently a GrabCut mask with rectangular shape is defined for each vertical region. However, in some cases the region is far better approximated by a rotated rectangle. Creating a GrabCut mask based on a rotated rectangle would lead to a better approximation of the region shape. An even better way to improve the approximation of the region shape would be to trace the region outline using the line segments in the region.

Another source of error in the depth map is incorrect grouping of depth regions into depth sets. It would probably make much more sense to group depth regions together based on possible connecting line segments rather than a threshold.

As mentioned in Section 6.4.1, the proposed depth estimation method fails to detect large structural objects in the scene, for instance long walls or a tunnel openings directly cut into a mountain. It would be possible to detect larger objects in the scene by increasing the maximum area A_{max} for the MSER algorithm but that may lead to new potential MSER that can be misclassified as vertical. Furthermore, the current region classification system tends to fail to classify large regions as vertical because they contain many more line segments (with many different orientations) compared to small regions. It is entirely possible that combining the line histogram classification with vanishing point detection would lead to a better result. After all, a vertical vanishing point would most likely indicate that the region is vertical. In any case, if a failure to detect large structural elements in a scene occurs it may be a good idea to manually create polygons approximating the regions. The depth can then by drawn using vertical depth lines similarly to how was done when drawing the tunnel wall depth in Section 4.10.2.

Another flaw in the current depth estimation method is that the tunnel wall depth is only estimated correctly under the assumption that the tunnel shape is rectangular. In reality however there are many tunnels with non-rectangular shapes. This issue can be solved by first detecting the correct tunnel shape and then implementing algorithms to find the tunnel walls for different shapes.

Finally, the current method assumes that there is only one tunnel opening visible in the scene but this is not always the case. An easy solution to the problem is to find the correct bounding rectangle (or other shape) for the second tunnel opening. This however requires knowledge of the location of the second tunnel opening in the image.

There may be some ways to reduce the computation time required for the depth map estimation. As shown in Section 6.5, roughly one third of the computational cost is dedicated to finding potential regions with vertical depth. The most computationally demanding method/algorithm in this part of the code is GrabCut. GrabCut could however be replaced (at the cost of a slightly worse region shape approximation) by another foreground/background segmentation method such as Watershed Segmentation (which is significantly faster). Another way to reduce the computation time is to store the line segments in a grid structure. Currently, to find the line segments within each MSER requires searching through a list of all line segments. However, if the line segments would be stored in a grid instead, then the search would be limited to only the parts of the grid where the MSER is located. This would greatly speed up the search. Finally, the computation time required for HFS can be greatly reduced by using the GPU/CUDA implementation of the algorithm rather than the CPU implementation.

7.2 Reducing the computation time

Currently, the computation time required for the L_{seq} estimation is many times more than what is available in the product. Even if the performance requirements would be slightly reduced by computing L_{seq} less often and thus dedicating more time to the computation, there still would not be enough time to run the whole implementation. However, as previously mentioned, the system uses traffic compensation. This means that in most practical cases the scene can be assumed to be static (unless the camera is moved of course). The assumption of a static scene implies that the depth map for a scene can be created once and used each time the image needs to be warped. Furthermore, under the assumption of a constant depth map and that the view is warped to the same camera pose every time, the computation can be further simplified. After all, each warped image will have the same size as the previous one and each time the image will be warped in the same way (have the same warped coordinates as the previous image). Moreover, the mask defining pixels with valid values in the warped image can be created once and re-used each time. The L_{seq} diagram can also be pre-computed and applied on the warped images. Finally, there is also the possibility to warp back the L_{seq} diagram to the original camera view and calculating the luminance value in the original view. After this warped back L_{seq} diagram has been computed, it can be stored and re-used. Thus in summary, the computation time can be greatly reduced by pre-computation and it should be entirely possible to do computation of the L_{seq} value within the assigned time slot. Pre-computation also implies greater flexibility in choosing a depth estimation method and in defining a more correct L_{seq} diagram in the warped view (for instance re-centering the diagram if it is centered incorrectly).

7.3 Future Work

As mentioned in Section 7.1, there are many ways to improve the proposed depth map estimation method. However, an alternative is to implement an already existing depth map estimation method. As mentioned in Chapter 2, it is probably best to use a geometrical depth estimation method. A suggestion is to use the line-tracing depth estimation method proposed by Jung *et al.*. If supervised machine learning is a possibility then the methods/algorithms proposed by Zhang and Yan in [26] or [9] could be implemented. These two methods yield very good depth estimation results. Since the depth map would be pre-computed there is no need to worry about the computational performance for the method chosen.

In Section 7.2 it was mentioned that the ultimate reduction of computation time can be achieved by computing the L_{seq} value in the original view using the warped back L_{seq} diagram. This method of measuring the L_{seq} value has not been implemented yet but it would not require a lot of work. It can be done by using the warped depth map and applying an inverse warping transformation in two steps as follows: Back-projecting the warped 2D coordinates for the L_{seq} region masks using the warped depth and the inverse of the camera matrix for the warped view, i.e. P'^{-1}, and then forward projecting the 3D coordinates (X', Y', Z') using the original camera matrix

P. Note that before warping back the L_{seq} diagram it might be a good idea to find a better and more reliable method to define it in the warped view or to re-center the diagram manually.
An obvious direction for future work is to gather more measurement data and evaluate the proposed approach on this data. Ideally the data should be gathered at more scenes, specifically at scenes where there is an easy way to obtain both a view of the scene from above the road and a view at road level. Since the view of the scene from road level is usually hard to obtain for a pedestrian it might be a good idea to instead take images of the scene in a car similarly to how it was done for the Gnistängstunneln scene. This approach has the added benefit that the effect of the windscreen on the L_{seq} measurement can be included. Ideally the measurements should be done for tunnels, which is not always possible. However, since the luminance measurement product will be installed at a few tunnel sites in the near future, such measurements may become available. So far all of the measurements presented were gathered when the weather was sunny. However, it is important to study the effectiveness of the proposed method for different weather conditions such as partially cloudy, entirely cloudy, wet/rainy weather, wet road but sunny and so on. It may be the case for instance that there is not a large difference in L_{seq} values between the views when it is cloudy and that the proposed method in that case does not yield a significant reduction of the error.
The final direction of future work is to study the Bidirectional Reflectance Distribution Function (BRDF) for asphalt, both in the case of dry asphalt and in the case of wet asphalt. This was originally going to be a part of the project but there was not enough time to study this problem in detail. The study could involve estimating the BRDF as a function of height or angle and then applying the BRDF to the luminance computation to obtain a higher reduction of the L_{seq} error.

7.4 Conclusion

In conclusion, the proposed warping method in its' current implementation reduces the L_{seq} error significantly. The error reduction is close to the error reduction obtained using ground truth depth maps. Even though the implementation has many flaws, it is perfectly viable for implementation in the luminance measurement system using pre-computation. The best way to reduce the L_{seq} error further is probably to apply the BRDF for asphalt on the luminance measurement. Using better depth maps only reduces the error slightly but on the other hand wrong parameters and false detection of depth regions may degrade the result of the proposed depth estimation method. It is therefore recommended to switch to a more robust and more reliable depth estimation method or alternatively allow for some user interaction to improve the estimated depth maps.

Bibliography

[1] *ITS JPO Website Home Page* [Online]. Available: https://www.its.dot.gov/about/faqs.htm

[2] W. Adrian *et al.*, "Guide for the Lighting of Road Tunnels and Underpasses,", CIE, 2004.

[3] K. Jonsson *et al.*, "Final report - Traffic compensated luminance estimation," Cipherstone Technologies AB, SP and University of Gothenburg, Göteborg, Västra Götaland, Final Rep., 2015.

[4] (2017, June). *Göteborg, Västra Götalands län - Google Maps* [Online]. Available: https://www.google.com/maps/@57.6763654,11.8954671,3a,57.6y,180.22h,79.31t/data=!3m6!1e1!3m4!1sNrfQF0uwq0iS-9snXyg3Hg!2e0!7i13312!8i6656

[5] R. Szeliski, *Computer Vision Algorithms and Applications Szeliski.* London, UK: Springer, 2011.

[6] T. Akenine-Möller, *Real-Time Rendering*, 3rd ed. Miami: CRC Press, 2008.

[7] R. Hartley and A. Zisserman, *Multiple View Geometry in computer vision*, 2nd ed. Cambridge, UK: Cambridge University Press, 2003.

[8] H. Shum and S. B. Kang, "Review of image-based rendering techniques," in *Proc. SPIE 4067, Visual Communications and Image Processing 2000*, 2000 ©SPIE. doi: 10.1117/12.386541

[9] S. Zhang and S. Yan, "View Transformation based on a Single Outdoor Image, "*IJARS*, vol. 10, no. 5, pp. 225-233, May, 2013.

[10] A. Criminsi *et al.*, "Single View Metrology," *IJCV*, vol. 40, no. 2, pp. 123-148, Nov., 2000.

[11] G. Wang *et al.*, "Camera Calibration and 3D reconstruction from a single view based on scene constraints," *IVC*, vol. 23, no. 1, pp. 311-323, Mar., 2005.

[12] C. Fehn, "Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV," in *Proc. SPIE 5291, Stereoscopic Displays and Virtual Reality Systems XI*, San Jose, CA, 2004, pp. 92-104.

[13] S. M. Muddala *et al.*, "Virtual view synthesis using layered depth image generation and depth-based inpainting for filling disocclusions and translucent disocclusions," *JVCIR*, vol. 38, pp. 351-366, July, 2016.

[14] Y. Zhao *et al.*, "An Overview of 3D-TV System Using Depth-Image-Based Rendering," in *3D-TV System with Depth-Image-Based Rendering Architectures, Techniques and Challenges.* New York: Springer, 2013, ch. 1, pp. 3-35.

[15] A. Telea, "An Image Inpainting Technique Based on the Fast Marching Method," in *JGT*, vol. 9, no. 1, pp. 23-34, Apr., 2004.

[16] A. Saxena *et al.*, "Learning 3-D Scene Structure from a Single Still Image," in *2007 IEEE 11th Int. Conf. Computer Vision.*, Rio de Janeiro, 2007, pp. 1-8.

[17] S. M. Kazmi *et al.*, "Exploiting a scene calibration mechanism for depth estimation," in *2012 3rd Int. Conf. Image Processing Theory, Tools and Applications.*, Piscataway, NJ, 2012, pp. 421-425.

[18] S. Haque *et al.*, "Gaussian-Hermite moment-based depth estimation from single still image for stereo vision," *JVCIR*, vol. 41, pp. 281-295, Nov., 2016.

[19] S. Zhuo and T. Sim, "Defocus map estimation from a single image," *Pattern Recognition*, vol. 44, no. 9, pp. 1852-1858, Sept. 2011.

[20] D. Zhao *et al.*, "Depth map extraction based on geometry," in *2012 Proc. IEEE Southeastcon.*, Orlando, FL, 2012, pp. 1-5.

[21] S. Battiato *et al.*, "3D stereoscopic image pairs by depth-map generation," in *Proc. 2nd International Symposium 3D Data Processing.*, Thessaloniki, 2004, pp. 124-131

[22] Y. Fan *et al.*, "A 2D-to-3D Image Conversion System Using Block Slope Pattern Based Vanishing Point Detection Technology," in *2012 International Symposium on Computer, Consumer and Control.*, Taichung, 2012, pp. 321-324.

[23] Y. J. Jung *et al.*, "A novel 2D-to-3D conversion technique based on relative height-depth cue," in *Proc. SPIE 7237, Stereoscopic Displays and Applications XX.*, 2009 ©SPIE. doi: 10.1117/12.806058

[24] Y. Salih and A. S. Malik, "Depth and Geometry from A Single 2D Image Using Triangulation," in *2012 IEEE International Conference on Multimedia and Expo Workshops.*, Melbourne, VIC, 2012, pp. 511-515.

[25] D. Hoiem *et al.*, "Recovering Occlusion Boundaries from an Image," *IJCV*, vol. 91, no. 3, pp. 328–346, Feb., 2011.

[26] S. Zhang and S. Yan, "Depth estimation and occlusion boundary recovery from a single outdoor image," *Opt Eng*, vol. 51, no. 8, Aug., 2012.

[27] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11-15, Jan., 1972.

[28] J. Matas *et al.*, "Robust Detection of Lines Using the Progressive Probabilistic Hough Transform," *CVIU*, vol. 78, no. 1, pp. 119-137, Apr., 2000.

[29] R. G. von Gioi *et al.*, "LSD: A Fast Line Segment Detector with a False Detection Control," *TPAMI*, vol. 32, no. 4, pp. 722-732, Apr., 2010.

[30] E. J. Chappero *et al.*, "Vanishing Point estimation from monocular images".

[31] H. Kong *et al.*, "Vanishing point detection for road detection," in *2009 IEEE Conf. Computer Vision and Pattern Recognition.*, Miami, FL, 2009, pp. 96-103.

[32] Z. Wu *et al.*, "A Novel Line Space Voting Method for Vanishing-Point Detection of General Road Images," vol. 16, no. 7, pp. 948-960, June, 2016.

[33] M. Nieto and L. Salgado, "Real-time robust estimation of vanishing points through nonlinear optimization," in *Proc. SPIE 7724, Real-Time Image and Video Processing 2010.*, Brussels, 2010, pp. 772402-7724014.

[34] G. W. Zack *et al.*, "Automatic measurement of sister chromatid exchange frequency," *JHC*, vol. 25, no. 7, pp. 741-753, July, 1977.

[35] F. Meyer, "Color image segmentation," in *Int. Conf. Image Processing and its Applications.*, Maastricht, 1992, pp. 303-306.

[36] C. Rother *et al.*, ""GrabCut": interactive foreground extraction using iterated graph cuts," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 309-314, Aug., 2004.

[37] M. Cheng *et al.*, "HFS: Hierarchical feature selection for efficient image segmentation," in *European Conference Computer Vision.*, Amsterdam, 2016, pp. 867-882.

[38] J. Matas *et al.*, "Robust wide-baseline stereo from maximally stable extremal regions," *IVC*, vol. 22, no. 10, pp. 761-767, Sept., 2004.

[39] D. Nistér and H. Stewénius, "Linear time maximally stable extremal regions," in *ECCV '08 Proc. 10th European Conference Computer Vision*, Marseille, 2008, pp. 183-196.

[40] W. Burger and M. J. Burge, "Geometric Operations," in *Digital Image Processing An Algorithmic Introduction Using Java*, 2nd ed. London, UK: Springer, 2016, ch. 21, pp. 513-514.

[41] B. Artman. (2018, May 11). *Projective geometry | Britannica.com* [Online]. Available: https://www.britannica.com/science/projective-geometry

[42] J. Košecká and W. Zhang, "Video compass," *LNCS*, vol. 2353, pp. 476-490, Jan., 2002.

[43] K. Sharkey and R. Beare. (2012, Janunary 27). *Histogram-based Thresholding | The Kitware Blog* [Online]. Available: https://blog.kitware.com/histogram-based-thresholding/

[44] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *JEI*, vol. 13, no. 1, pp. 146-168, Mar., 2004.

[45] (2018, June 7). *OpenCV: Hierarchical Feature Selection for Efficient Image Segmentation* [Online]. Available:
https://docs.opencv.org/trunk/dc/d29/group___hfs.html

[46] D. Chandler and R. Munday, *A Dictionary of Media and Communication*, 2nd ed. Oxford, UK: OUP, 2016.

[47] R. J. Koshel. (2014). *Luminance - AccessScience from McGraw-Hill Education* [Online]. Available:
https://www-accessscience-com.proxy.lib.chalmers.se/content/391200

[48] M. Safdar *et al.*, "Obtaining Absolute Scene Luminance using HDR Imaging," *LNEE*, vol. 369, pp. 133-138, Jan., 2016.

[49] (2018, May 21). *OpenCV API Reference - OpenCV 2.4.13.6 documentation* [Online]. Avaiable: https://www.docs.opencv.org/2.4/modules/refman.html

[50] (2018, May 25). *OpenCV: cv::LineSegmentDetector Class Reference* [Online]. Available: https://docs.opencv.org/trunk/db/d73/classcv_1_1LineSegmentD etector.html

[51] M. Nieto. (2012, March 31). *Vanishing point detection C++ source code | Marcos Nieto's Blog* [Online]. Available:
https://marcosnietoblog.wordpress.com/2012/03/31/vanishing-point-detection-c-source-code/

[52] (2018, May 22). *OpenCV: cv::MSER Class Reference* [Online]. Available: https://docs.opencv.org/trunk/d3/d28/classcv_1_1MSER.html

[53] (2018, Feb 23). *OpenCV: cv::hfs::HfsSegment Class Reference* [Online]. Available: https://docs.opencv.org/3.4.1/d2/de0/classcv_1_1hfs_1_1HfsSegment.html

[54] (2018, May 26). *Miscellaneous Image Transformations - OpenCV 2.4.13.6 documentation* [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneou0s_transformations.html?highlight=grabcut#grabcut

[55] (2018, May 29). *Inpainting - OpenCV 2.4.13.6 documentation* [Online]. Available: https://docs.opencv.org/2.4/modules/photo/doc/inpainting.html

[56] (2018, May 30). *Image Filtering - OpenCV 2.4.13.6 documentation* [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=morphologyex#morphologyex

[57] *FLIR Downloads* [Online]. Available:
https://eu.ptgrey.com/support/downloads/10984/

[58] (2018, May 31). *Camera calibration With OpenCV - OpenCV 2.4.13.6 documentation* [Online]. Available:
https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

[59] (2018, May 31). *Camera Calibration and 3D Reconstruction - OpenCV 2.4.13.6 documentation* [Online]. Available:
https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#calibratecamera

[60] (2018). *Nikon | Sport Optics | Forestry Pro* [Online]. Available:
http://imaging.nikon.com/lineup/sportoptics/laser/forestrypro/index.htm

[61] (2017, June). *Fiskhamnsgatan - Google Maps* [Online]. Available:
https://www.google.com/maps/@57.6991065,11.9288457,3a,75y,58.65h,86.56t/data=!3m6!1e1!3m4!1snLFiOz0iEe7XjmySkXuFxQ!2e0!7i13312!8i6656

[62] *Kartor, vägbeskrivningar, flygfoton, sjökort & mycket mer på eniro.se* [Online]. Available: https://kartor.eniro.se

[63] *Interaktiv karta - hitta.se* [Online]. Available: https://www.hitta.se/kartan

[64] (2017, July). *Lundby Hamngata - Google Maps* [Online]. Available:
https://www.google.com/maps/@57.7172471,11.9519538,3a,75y,44.81h,88.65t/data=!3m6!1e1!3m4!1snttz9ssjaraDDYu3lhb3rA!2e0!7i13312!8i6656

[65] (2017, May). *Lundbytunneln - Google Maps* [Online]. Available:
https://www.google.com/maps/@57.7103876,11.923793,3a,75y,291.87h,80.03t/data=!3m6!1e1!3m4!1sw_Q0ZnpYAag1cJiWU5E7ng!2e0!7i13312!8i6656