Applications of Machine Vision

Quality Control, Cancer Detection and Traffic Surveillance

Hanna Källén

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118
SE-221 00 Lund
Sweden

http://www.maths.lth.se/

Doctoral Theses in Mathematical Sciences 2016:1
ISSN 1404-0034

ISBN 978-91-7623-701-4 (print)
ISBN 978-91-7623-702-1 (digital)
LUTFMA-1055-2016

Printed in Sweden by MEDIA-TRYCK AB, Lund 2016

Preface

In this thesis, image analysis and computer vision has been used to assist in real-world problems. The problems that are studied in the thesis differs a lot, but in all cases image analysis can be used to either reduce the amount of manually work or to increase accuracy.

The work in this thesis is based on the following papers:

- Hanna Källén, Håkan Ardö and Olof Enqvist,"Tracking and Reconstruction of Vehicles for Accurate Position Estimation", *IEEE Workshop on Applications of Computer Vision (WACV)*, Hawaii, 2011.
- Hanna Källén, Anders Heyden, Kalle Åström and Per Lindh, "Measurement of Bitumen Coverage of Stones for Road Building, Based on Digital Image Analysis", *IEEE Workshop on Applications of Computer Vision (WACV)*, Breckenridge, 2012.
- Hanna Källén, Anders Heyden and Per Lindh, "Measuring Bitumen Coverage of Stones using a Turntable and Specular Reflections", *International Conference on Computer Vision Theory and Applications (VISAPP)*, Barcelona, 2013.
- Hanna Källén, Anders Heyden and Per Lindh, "Estimation of Grain Size in Asphalt Samples using Digital Image Analysis", *Conference on Applications of Digital Image Processing XXXVII*, San Diego, 2014.
- Hanna Källén, Jesper Molin, Anders Heyden, Claes Lundström and Kalle Åström, "Towards Grading Gleason Score using Generically Trained Deep Convolutional Neural Networks", Accepted to *IEEE International Symposium on Biomedical Imaging (ISBI)*, Prague, 2016.

- Hanna Källén, Anders Heyden, Kalle Åström and Per Lindh, "Measuring and Evaluating Bitumen Coverage of Stones using two Different Digital Image Analysis Methods", *Measurement*, 2016.

Acknowledgements

First of all I would like to thank my main supervisor Anders Heyden and my co-supervisors Kalle Åström and Per Lindh for all valuable help during these years at the Centre of Mathematical Sciences.

I also would like to thank all co-authors on the papers, Olof Enqvist and Håkan Ardö at the math department and Jesper Molin and Claes Lundström at Sectra in Linköping.

Furthermore, I would like to thank all my co-workers at the Centre of Mathematical Sciences, and especially the members of the vision group. I would like to give a special thanks to my former room mates for many years, Zaiyide, Kerstin and Håkan. Even though the office could be crowded at some times, I stilled enjoyed your company. I also would like to give a special thanks to Matilda for many interesting discussions during the years, fruitful or not, and to Carl-Gustav for helping me out whenever my computer broke down, with or without my help. I also would like to thank the staff at Peab in Helsingborg for suppling materials when I needed it.

Finally, I would like to thank all friends and family for everything else. A special thank to Linda Vallenhag who was the best study mate ever during my engineering studies. Many were the days you saved my space in the study center during the exam weeks. For that I am forever grateful.

This work was founded by SBUF (12943) and Vinnova (grant number 2013-03906 and 2009-00252).

Mora, 2016-02-27

Contents

Chapter 1

Introduction

During the last decades, image analysis has become an important tool in various applications. The increase in computer power has led to that more advances algorithms can be developed. In this thesis image analysis has been used in three different applications; quality control of asphalt, cancer detection in histopathological images and traffic surveillance.

In the first part of the thesis image analysis has been used to assist researchers at asphalt laboratories. Asphalt consists of a mixture of stones of different sizes and a binder called bitumen. Mainly two problems have been studied; the affinity between stones and bitumen and grain size distribution in asphalt samples. The affinity between bitumen and stone is very important for the durability of the pavement and is traditionally measured by the rolling bottle test. First stones are covered in bitumen, then they are put in a glass bottle that is put on a bottle rolling machine. The rolling causes some of the bitumen to get teared of, the more that is left the better. Today the degree of bitumen coverage is estimated manually by a few laboratory assistant and the results often differ between different laboratory assistants and different laboratories. A goal in this thesis is to replace the manual evaluation in the method by automatic image analysis to make this test more accurate and objective. It is a quite easy problem to solve when the color difference between the stones and the black bitumen is fairly clear, but when the stones are very dark it is difficult to see any difference between stones and bitumen.

Another quality control is to estimate the size distribution of stones in asphalt samples to see if it follows the recipe for the asphalt. This is done today by dissolving the sample in highly toxic substances. Then the size distribution is estimated by measure the percentage of stones that passes a certain sieve size. To avoid toxic substances, as preferred by the European Union, we have analysed cross section of the samples with image analysis to estimate the size distribution.

In the second part, histopathological images have been studied. The patholo-

gist have to analyze a large amount of prostatic biopsies. In the biopsies, cancerous tissue has to be separated from benign tissue. Also the cancerous tissue is separated into three different grades of cancer. This is very time-consuming and a goal in this thesis is to assist the pathologist in their diagnostics by giving a suggestion of segmentation that the pathologist can verify or adjust. For this, features from pre-trained deep neural networks have been used to classify the images. These features were used to train classifiers to classify the image into the four different classed.

The third and last part of the thesis deals with tracking and 3D-reconstruction of vehicles. To measure how safe certain roads and intersections are one can predict the number of accidents that will happen by observing certain events, conflicts, during a short time period. This is done by letting trained personnel study video of the intersection, which is time-consuming and expensive. In this thesis, 3D-reconstructions of vehicles have been done from videos, with the 3D-models it is possible to accurate estimate positions of the vehicles. A system to track interesting points between images in a movie sequence and then sort out which points that belong to which vehicle has been developed. These points are then used to estimate camera orientations and to build models of the vehicles. Also the vehicles pose has been estimated in all frames, to get their positions. The positions could then be used to detect conflicts in the intersection.

Thesis Overview

The thesis has been split into three different parts. The first part concerns applications of image analysis in asphalt research, to assist in the laboratory work performed in the laboratories. The second part is about finding cancerous tissue in pathological images. The third part concerns traffic surveillance and 3D-reconstructions of vehicles.

Chapter 2 This chapter provides an introduction to segmentation of images, and present some of the most common segmentation algorithms.

Chapter 3 Asphalt consists of a mixture of stones of different size and a binder called bitumen. This chapter describes a method to automatically estimate the degree of bitumen coverage of stones that are partly covered in bitumen. This is an important test to ensure that the pavement last as long as possible.

Chapter 4 When asphalt is produced, the size distribution of the stones are defined by a recipe for the asphalt. Another important quality control is to estimate the size distribution of stones in asphalt samples, and compare this with the recipe. This chapter provides an algorithm for analyzing slices of asphalt by image analysis.

Chapter 5 This chapter provides theoretical background to some important machine learning techniques, that are used in the following chapter.

Chapter 6 In this chapter a system to automatically classify microscopy images of prostatic tissue into different grades of cancer is presented.

Chapter 7 This chapter derives the camera model that is used to project points in space to an image. Also an algorithm to track a point in an image sequence is described.

Chapter 8 A system that builds 3D models of vehicles in an intersection using a video of the intersection is presented. These models can be used to estimate the position of the vehicles in the intersection for every frame in the video.

Author Contributions

The per-paper contributions of the author are as follows:

- Hanna Källén, Håkan Ardö and Olof Enqvist,“Tracking and Reconstruction of Vehicles for Accurate Position Estimation", *IEEE Workshop on Applications of Computer Vision (WACV)*, Hawaii, 2011.

  I implemented the algorithms, performed all experiments and wrote the paper with help from Olof.

- Hanna Källén, Anders Heyden, Kalle Åström and Per Lindh, “Measurement of Bitumen Coverage of Stones for Road Building, Based on Digital Image Analysis”, *IEEE Workshop on Applications of Computer Vision (WACV)*, Breckenridge, 2012.

  The idea to use image analysis to assist in the laboratory work came from Per. He also developed the setup for the experiments and provided the

images. Kalle and I developed the method together, I did most of the coding and wrote most of the paper.

- Hanna Källén, Anders Heyden and Per Lindh, "Measuring Bitumen Coverage of Stones using a Turntable and Specular Reflections", *International Conference on Computer Vision Theory and Applications (VISAPP)*, Barcelona, 2013.

  The idea to use a turntable to achieve specular reflection in the bitumen came from me. I was also involved in photographing the stones and wrote all the code and most of the paper.

- Hanna Källén, Anders Heyden and Per Lindh, "Estimation of Grain Size in Asphalt Samples using Digital Image Analysis", *Conference on Applications of Digital Image Processing XXXVII*, San Diego, 2014.

  The idea to use image analysis to analyze the grain size estimation came from Per. I implemented the fast marching algorithm with some input from Anders, and the rest of the image analysis algorithms myself. I also wrote the paper, apart from a small part in the introduction.

- Hanna Källén, Jesper Molin, Anders Heyden, Claes Lundström and Kalle Åström, "Towards Grading Gleason Score using Generically Trained Deep Convolutional Neural Networks", Accepted to *IEEE International Symposium on Biomedical Imaging (ISBI)*, Prague, 2016.

  Kalle came up with the idea to use generically trained networks to classify prostatic tissue. I performed the experiments and wrote most of the paper with help from Jesper.

- Hanna Källén, Anders Heyden, Kalle Åström and Per Lindh, "Measuring and Evaluating Bitumen Coverage of Stones using two Different Digital Image Analysis Methods", *Measurement*, 2016.

  This is a continuation on the papers presented at WACV 2012 and VISAPP 2013. I came up with the idea to use the results in WACV to evaluate a modified version of the method in VISAPP. I performed all experiments, did all the coding and wrote the paper with some input from Kalle and Anders.

Part I

Quality Control

Chapter 2

Preliminaries on Segmentation

A common problem in image analysis and in this thesis is segmentation of images. With segmentation we mean to separate the image into different regions, or segments, where pixels within a region belong to the same object or are similar in some other way. Since segmentation is a very common problem there exists a large number of different segmentation techniques. Most of the methods deal with the two class problem of foreground and background segmentation, where the interesting part of the image is in the foreground and the not so interesting part in the background. In this chapter, three different segmentation methods will be presented; the very basic thresholding, segmentation by graph-cuts and fast marching. Some morphological operations to manipulate the segments will also be presented.

2.1 Morphology

Morphological operations [31] are used to shrink or expand shapes or to smoothen the boundaries of an object. Some of the most common operations are erosion, dilation, opening and closing.

Morphological operations are often used on binary images but could also be extended to gray scale images. In this brief introduction, only morphological operation on binary images will be discussed. All the morphological operations are performed with two sets, one of them is the image and the other one is a small set called structuring element. The structuring element could for example be a square, a cross or a disc.

2.1.1 Erosion and Dilation

Erosion is used to shrink objects by removing elements at, or close to, the boundary. It is defined by

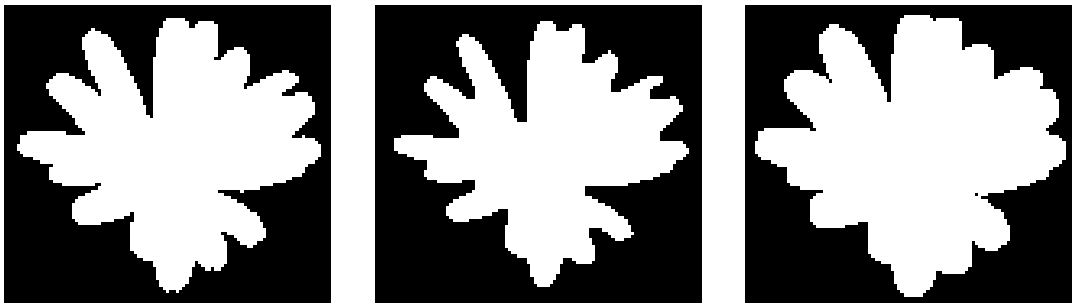
$$A \ominus B = \{x \in X | x + b \in A \text{ for every } b \in B\}, \tag{2.1}$$

where A is the object and B a symmetric structuring element. Here X is the whole image.

Dilation is the opposite of erosion and used to expand the object by adding elements close to the boundary. Dilation is defined as

$$A \oplus B = \{x \in X | x = a + b \text{ for some } a \in A \text{ and } b \in B\}. \tag{2.2}$$

Figure 2.1 shows an example of erosion and dilation on the object seen in Figure 2.1a. The structuring element used in both the erosion and dilation is in this case is a 3×3 square. The image after erosion can be seen in Figure 2.1b and the image after dilation in Figure 2.1c.



(a) Original image. (b) After erosion. (c) After dilation.

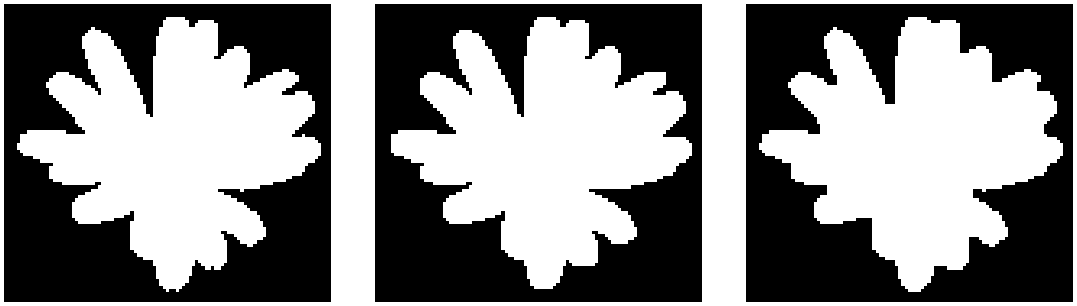
Figure 2.1: Erosion and dilation performed with the same structuring element on the same object.

2.1.2 Opening and Closing

The opening of a set is defined as erosion with a structuring element followed by dilation with the same structuring element on the resulting object. The closing is the opposite, dilation followed by erosion. Both opening and closing will cause some smoothing of the contour of the object. The opening will remove small out-sticking elements and remove small islands but will not affect deep valleys.

Closing affects the valleys and fills small holes but will not affect out-sticking objects.

Figure 2.2 shows opening and closing with the same square structuring element on the image shown in Figure 2.2a. The opening can be seen in Figure 2.2b and the closing in Figure 2.2c.



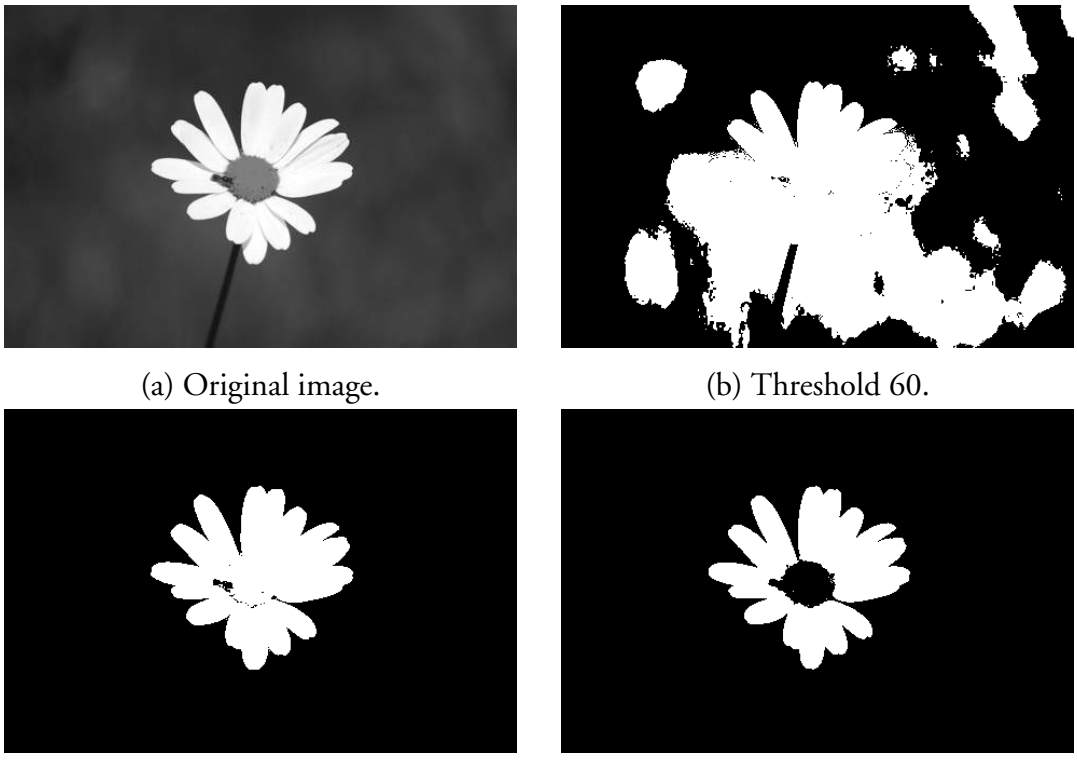
(a) Original image. (b) After opening. (c) After closing.

Figure 2.2: Opening and Closing performed on the same object.

2.2 Thresholding

A simple way to segment an image into foreground and background is by thresholding the image. The most common way is to convert the image into a grayscale image and threshold on the intensities. Pixels with intensities below the threshold are labeled as background and pixels above the threshold are labeled as foreground or vice versa. It is also possible to threshold on some other property in the image, such as color difference. Thresholding works well in those cases where there is a clear intensity or color difference between the foreground and background, but it is very sensitive for the threshold chosen.

Figure 2.3 shows segmentation of a grayscale image with three different thresholds. The original image is seen in Figure 2.3a, the threshold in Figure 2.3b is too low and does not give any satisfying segmentation of the image. The threshold in Figure 2.3d is chosen too high while the threshold in Figure 2.3c is a good choice of threshold.



(a) Original image. (b) Threshold 60.

(c) Threshold 90. (d) Threshold 140.

Figure 2.3: Segmentation of an image with three different thresholds.

2.3 Graph-Cuts

With thresholding, the spatial information of the pixels are not used, two pixels close to each other are equally likely to be assigned to the same class as two pixels far away from each other. In many cases, this spatial information may be more important than pixel values. One widely used segmentation algorithm that uses both pixel values and spatial information is graph-cuts [11, 12, 49]. The idea is to express the problem in terms of minimizing an energy function. This function is represented by a graph, where nodes are the variables with edges between them. Minimizing the function will be the same as calculating the maximum flow in the graph if all weights on the edges are non-negative, which there are many algorithms for.

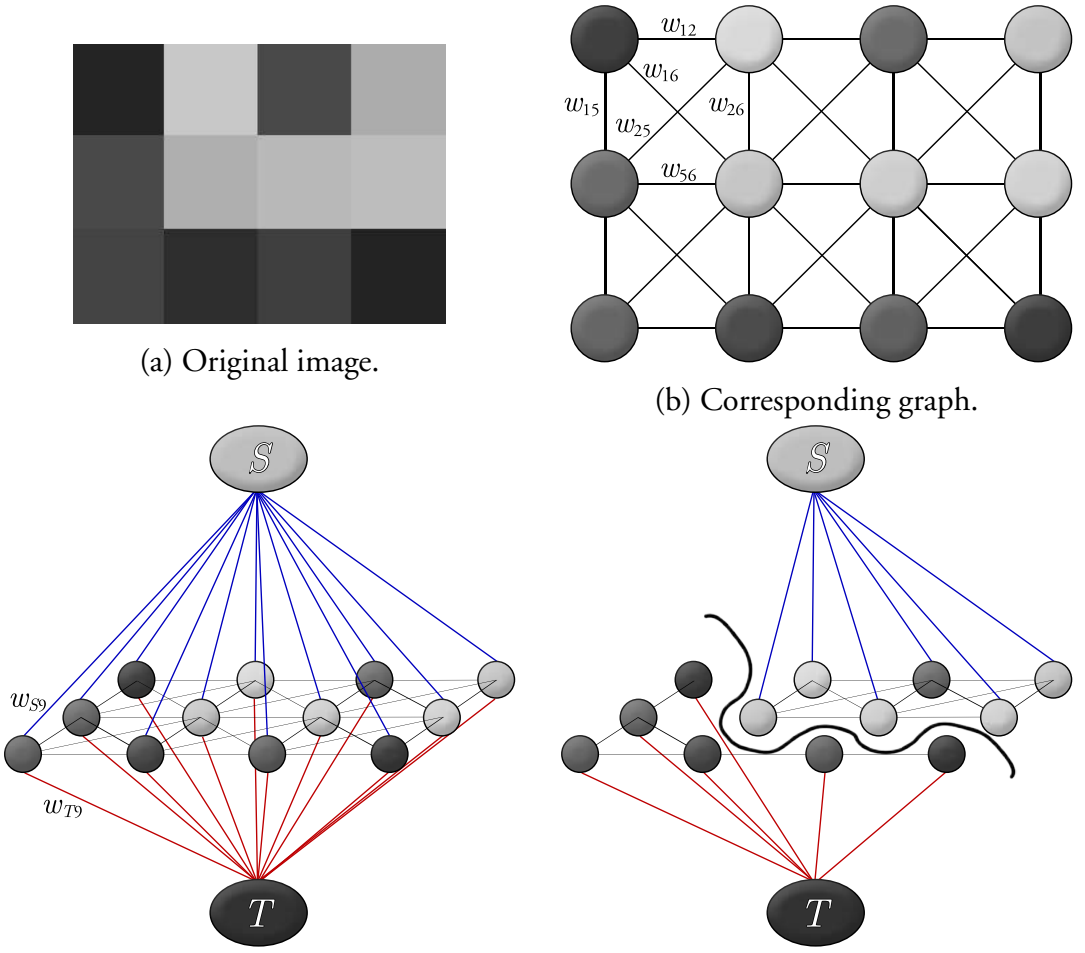
We want to find labels, f_p, for all pixels, p, in the best possible way. These labels are in this case foreground or background. We want to do this in a way that pixels close to each other are more likely to be assigned to the same label. The solution to this problem is to minimize an energy function consisting of a data part and a regularization part, on the form

$$E(f) = \underbrace{\sum_{i \in \mathcal{P}} D_i(f_i)}_{\text{data part}} + \underbrace{\sum_{i,j \in \mathcal{N}} w_{ij}(f_i, f_j)}_{\text{regularization part}}, \tag{2.3}$$

where D_i is a term that typically measures how well label f_i fit the data and $\mathcal{P}$ is the set of all pixels. The term w_{ij} describes how hard we should punish if two neighboring pixels have different labels and the set $\mathcal{N}$ is the set of all interacting pixels, neighbors. The terms w_{ij} can either be set individually for each pair of pixels or to a constant, same for all pairs.

An illustration of the graph-cut method is shown in Figure 2.4. Figure 2.4a shows a small image of 3×4 pixels. The corresponding graph can be seen in Figure 2.4b. The pixels are connected in a 8-neighborhood, meaning that pixels not at the border has 8 neighbors. The nodes in the graph correspond to the pixels in the image and they are connected to their neighbors by the edges shown in the illustration. Every edge has a weight associated, denoted by w_{ij}, which is the weight between pixel i and pixel j.

Then all nodes are connected to a foreground and background node called source, S, and sink, T. This can be seen in Figure 2.4c. The weight of the edge



(a) Original image.

(b) Corresponding graph.

(c) Foreground and background node.

(d) Separated graph.

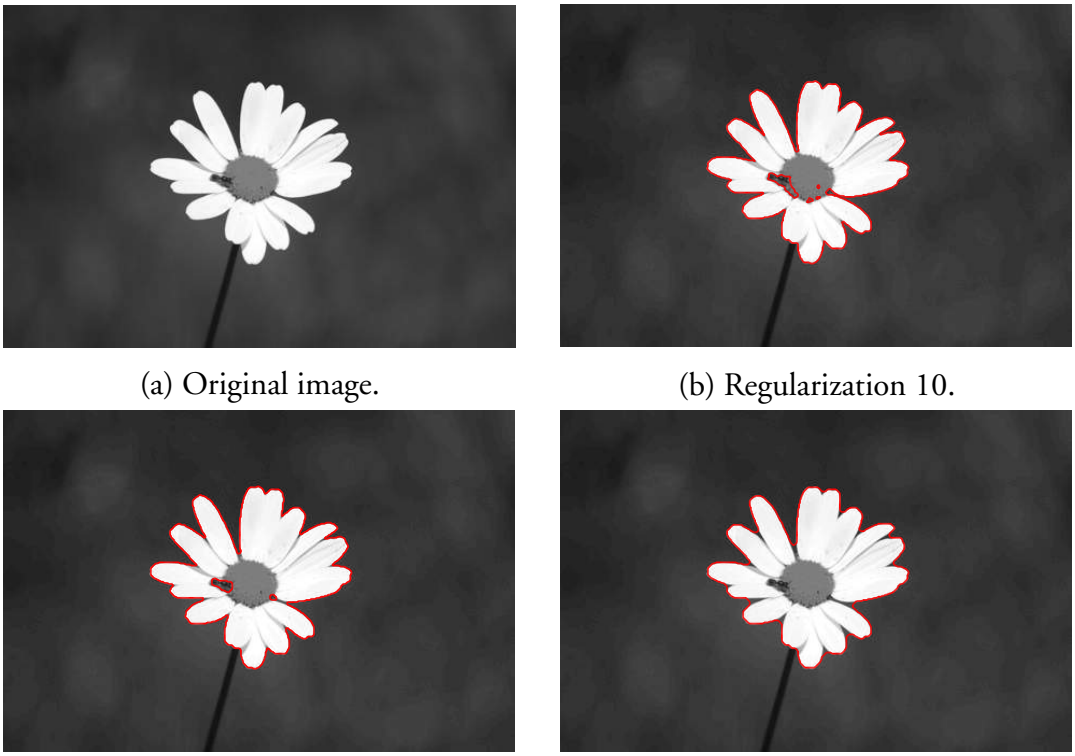
Figure 2.4: Illustration of a graph.

from the foreground node, S, to pixel i is denoted by w_{Si} and usually depends on the intensity difference between the foreground node and the pixel. In the same way the weight for the edge between the background node and a pixel is denoted w_{Ti}. Those will correspond to the term D_i in Equation 2.3. Then an optimization algorithm is used to find the best possible cut, meaning the cut where the total cost is as low as possible. The cut must be done in such a way that all pixels in the end have an edge to either the foreground or the background node, not both, as illustrated in Figure 2.4d. There cannot exist any edges from one side of the graph to the other. The cost for cutting one edge depends on the weight, high weight means high cost, and the total cost is the sum of the weight for all edges that must be cut. Figure 2.4d shows the separated graph with the

cut marked with a black line. The pixels that afterwards are connected to the foreground node are classified as foreground and the others as background.

2.3.1 Segmentation of an Image using Graph-Cut

Figure 2.5 shows an example of a segmentation done by graph-cuts for three different values on the regularization term. The weights between the pixels are set to a constant, different in the different images, and the weights to the foreground and background nodes are set to be the intensity difference between the pixel and the foreground and background node respectively. The original image can be seen in Figure 2.5a and the result of the segmentation can be seen in Figures 2.5b, 2.5c and 2.5d.



(a) Original image.

(b) Regularization 10.

(c) Regularization 25.

(d) Regularization 100.

Figure 2.5: Segmentation of an image by the graph-cut method with three different regularization terms. The red lines show the border between foreground and background.

2.4 Fast Marching

The fast marching algorithm was presented by James A. Sethian in [79] and [80]. It is a numerical technique that follows the evolution of an interface. Figure 2.6 shows such an interface expanding in the directions of the arrows.

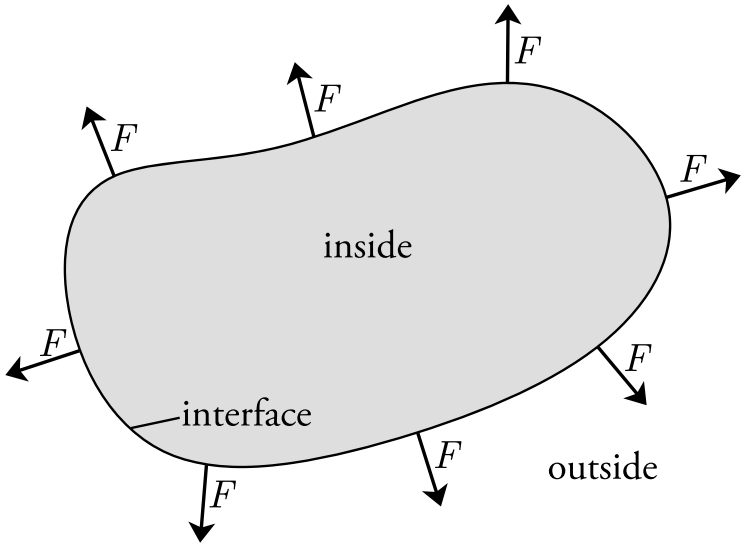


Figure 2.6: An interface expanding in the direction of the arrows.

In the fast marching method, we start with some initial boundary and let this curve expand according to some speed function, F. Which speed function to use depends on the application but it is always positive so that the interface only can expand and not shrink. In this thesis we will use the fast marching method for segmenting images and therefore we will start the curve somewhere where we are certain it is background and let the curve propagate from this. After a while we will stop the propagation and let the pixels inside the interface be set to background and the pixels that have not been reached by the curve to foreground. It is also possible to start in the foreground and expand towards background.

2.4.1 The Propagation Equation

The goal for the algorithm is to compute how the curve propagates and at what time the curve reaches all points. The time when the curve reach a point is called the arrival time of the point, denoted by $T(x)$, where x is the point. Consider the curve and the point x outside the curve in Figure 2.7. We start by assuming that

the curve only moves in one dimension and that T and F are one-dimensional functions.

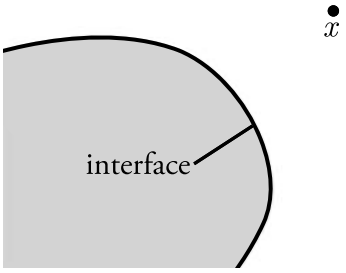


Figure 2.7: An interface that will reach the point outside denoted by x at time t.

The time when the curve reach the point x is denoted by t. By definition this is the arrival time, $T(x)$, for the point. This gives us

$$t = T(x), \tag{2.4}$$

if we take the derivative with respect to t we get

$$1 = \frac{dT}{dt} = \frac{dT}{dx}\frac{dx}{dt}, \tag{2.5}$$

where $\frac{dx}{dt}$ is nothing else than the speed of the interface at point x, which is F. Using this we can write (2.5) as

$$1 = F\frac{dT}{dx}. \tag{2.6}$$

In multiple dimensions this becomes the Eikonal equation

$$|\nabla T|F = 1, \tag{2.7}$$

where T is the arrival time for the curve and F is the speed function for the curve.

2.4.2 Algorithm

To numerically solve the Eikonal equation, we discretize the space by a finite grid, in the case of images it is already discretized and the grid points are the pixels. The grid points are then put in one of the following classes: *Known*, *Trial* or

Far. *Known* consists of the grid points on the boundary of the interface or points already passed by the curve. These points have already been assigned with an arrival time. *Trial* consists of the points that are neighbors to the boundary and not in *Known*. For these points a temporary arrival time can be computed, and the points are also put in a min-heap to easily and efficiently find the element with the smallest arrival time. *Far* consists of all other points.

The min-heap is a data structure, a tree, organized in a way that the top element is always the smallest one. In this case the smallest means that it has the smallest arrival time. This makes it very efficient to find the smallest arrival time without having to search through all elements.

In each step in the algorithm, the point with the smallest arrival time is chosen, this point is added to *Known* and removed from *Trial*. For neighbors to the points that are in *Far*, a temporary arrival time is computed and the point is added to *Trial*. For neighbors in *Trial*, the temporary arrival times are updated. This continues until all points are investigated or when the smallest of the temporary arrival times reaches some threshold.

Fast Marching Algorithm

1. Initialize, add the points at the initial boundary to *Known*, calculate the temporary arrival time for the neighbors, not in *Known*, according to Section 2.4.3, add them to *Trial* and to the heap.
2. Take out the first element from the heap which is the point with the smallest arrival time, add it to *Known* and remove it from *Trial*.
3. For all neighbors not in *Known*: update arrival time according to Section 2.4.3, add the ones not already in *Trial* to *Trial* and to the heap.
   While updating the arrival times, also update the heap.
4. Repeat 2-3 until the heap is empty or until the smallest of the arrival times is larger than some threshold.

2.4.3 Updating the Arrival Times

To find the arrival function we want to find a solution for the Eikonal equation derived earlier

$$|\nabla T|F = 1.$$

We solve this differential equation numerically by using the following updating scheme

$$\left(\begin{array}{l} \max(D_{ij}^{-x}T, 0)^2 + \min(D_{ij}^{+x}T, 0)^2 \\ \qquad + \max(D_{ij}^{-y}T, 0)^2 + \min(-D_{ij}^{+y}T, 0)^2 \end{array} \right)^{1/2} = \frac{1}{F_{ij}}, \tag{2.8}$$

where F_{ij} is the speed at the point (i, j), D_{ij}^{-x} is the one sided derivative in the negative x-direction defined by $D_{ij}^{-x} = \frac{T(x)-T(x-h)}{h}$. In the same way D_{ij}^{+x} is the one sided derivative in the positive x-direction defined by $D_{ij}^{+x} = \frac{T(x+h)-T(x)}{h}$. D_{ij}^{-y} and D_{ij}^{+y} are defined in the same way.

If we use a slightly different approximation of the gradient, introduced in [76], we get the more convenient upwind scheme

$$\left(\begin{array}{l} \max(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0)^2 \\ \qquad + \max(D_{ij}^{-y}T, -D_{ij}^{+y}T, 0)^2 \end{array} \right)^{1/2} = \frac{1}{F_{ij}}. \tag{2.9}$$

In this thesis, we will use the fast marching algorithm for segmenting images. In this case the grid points are the pixels in the images and the derivatives can be computed more easily. If we call the arrival time we want to calculate T and the arrival times for the neighbors as shown in Figure 2.8, we can calculate the derivatives as

$$\begin{aligned} D_{ij}^{-x}T &= T - a, \\ D_{ij}^{+x}T &= b - T, \\ D_{ij}^{-y}T &= T - c, \\ D_{ij}^{+y}T &= d - T, \end{aligned} \tag{2.10}$$

where a, b, c and d are the arrival times for the neighbors of the interesting pixel.

If we assume that the arrival times a and b are both known we can have one of the following cases:

- Both a and b are smaller than T, $T > a$, $T > b$

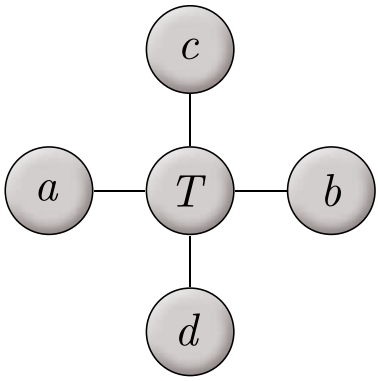


Figure 2.8: The arrival times for the current grid point and its neighbors. The arrival time we want to update is denoted by T and the arrival times for the neighbors are denoted a, b, c and d.

- a is smaller than T but b is larger, $T > a$, $T \leq b$
- b is smaller than T but a is larger, $T \leq a$, $T > b$
- Both a and b are larger than T, $T \leq a$, $T \leq b$.

For the first case we get

$$\max(T - a, -(b - T), 0) = \max(T - a, T - b, 0) = T - \min(a, b).$$

For the second case we get

$$\max(T - a, T - b, 0) = T - a,$$

but since we also know that $b > a$ this will be the same thing as writing

$$\max(T - a, T - b, 0) = T - \min(a, b).$$

The same thing applies for the third case. For the last case we get

$$\max(T - a, T - b, 0) = 0.$$

Then we can reduce the four cases above to just two. In similar way we get two cases when c and d are known.

The equation in (2.9) will look different depending on the size of T. Therefore we have to solve it for the three following cases:

- $T > \min(a, b)$ and $T > \min(c, d)$
- $T > \min(a, b)$ and $T \leq \min(c, d)$
- $T \leq \min(a, b)$ and $T > \min(c, d)$.

The fourth case $T \leq \min(a, b)$ and $T \leq \min(c, d)$ can never occur since T has to always increase and therefore has to be larger than the arrival time for at least one of its neighbors. Which one to use is not known in advance since T is not known.

In the first case when $T > \min(a, b)$ and $T > \min(c, d)$ we get

$$\begin{aligned}
&\max(T-a, T-b, 0)^2 + \max(T-c, T-d, 0)^2 \\
&= (T - \min(a, b))^2 + (T - \min(c, d))^2 \\
&= 2T^2 - 2(\min(a, b) + \min(c, d))T + \min(a, b)^2 + \min(c, d)^2.
\end{aligned}$$

And we can rewrite (2.9) as

$$T^2 - (\min(a, b) + \min(c, d))T + \frac{1}{2}\left(\min(a, b)^2 + \min(c, d)^2 + \frac{1}{F^2}\right). \tag{2.11}$$

This is a quadratic equation with two solutions and we choose the larger of the two. The solution will be real as long as $|\min(a, b) - \min(c, d)| \geq \frac{\sqrt{2}}{F}$. If this does not hold it is not a valid solution and we ignore this case.

For the second case when $T > \min(a, b)$ and $T \leq \min(c, d)$ we instead get

$$\max(T-a, T-b, 0)^2 + \max(T-c, T-d, 0)^2 = (T - \min(a, b))^2 + 0,$$

and we can rewrite (2.9) to

$$T = \frac{1}{F} + \min(a, b). \tag{2.12}$$

In the third case we get in the same way

$$T = \frac{1}{F} + \min(c, d). \tag{2.13}$$

These three cases will yield three different arrival times for the pixel. We are interested in the largest possible solution. For the first case the arrival time must

satisfy $T > \min(a, b)$ and $T > \min(c, d)$. If it does not, we are not in the first case anymore and the arrival times need to be computed according to the other cases.

Since F is always greater than zero it is enough to just look at one of the cases two and three. If we choose $T = \frac{1}{F} + \max(\min(a, b), \min(c, d))$ the arrival time will automatically be greater than both $\min(a, b)$ and $\min(c, d)$, violating the constraints.

By this we can reduce the two latter cases to one, namely

$$T = \frac{1}{F} + \min(a, b, c, d). \tag{2.14}$$

We can only have two possible outcomes for the solution of this equation, either T is between $\min(a, b)$ and $\min(c, d)$ or greater than both, in the latter case the constrains are violate and the arrival time must be calculated according to (2.11).

In the end we just have to calculate the arrival times according to (2.11). If the solution gets real and it does not violate the constraints this will be the largest of the solutions. Otherwise we calculate the arrival times according to (2.14).

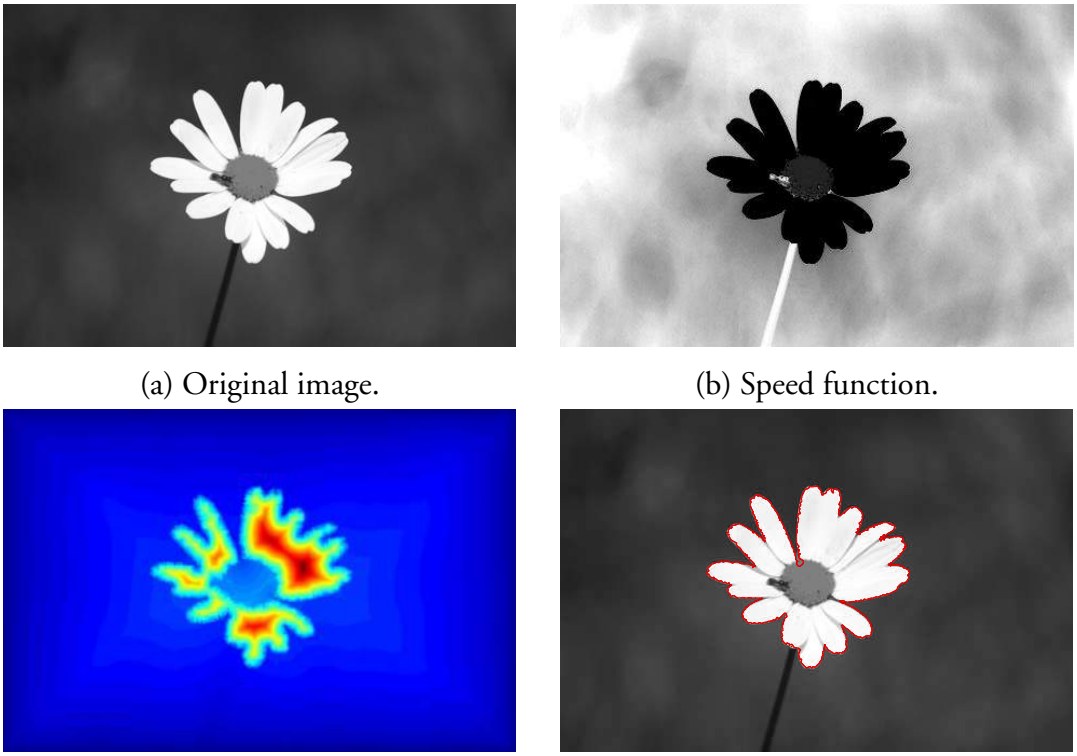
Usually the arrival times for all the neighbors are not known, but this will not make any substantial difference in the calculations. If we know one of a and b we just set $\min(a, b)$ to the one we know. In the same way we set $\min(c, d)$ to the one we know of c and d. Setting this we can calculate the arrival times exactly as before. If we do not know any of a and b we have to know at least one of c and d, in this case we calculate the arrival time according to (2.14) setting $\min(a, b, c, d) = \min(c, d)$.

2.4.4 Segmentation of an Image using Fast Marching

Figure 2.9 shows an example when fast marching was used to segment an image into foreground and background. The original image can be seen in Figure 2.9a. This image is then transformed to a speed image shown in Figure 2.9b where white means high value and black low. The speed function has to be chosen depending on the application. In this case we have a dark background and a bright foreground so we choose a speed function that will give high values for dark pixels and low values for brighter pixels. The speed function is given by

$$F(x, y) = \frac{1}{1 + e^{I(x,y)/v}}, \tag{2.15}$$

where $I(x, y)$ is the intensity of the given point and v determine the steepness of the function.



(a) Original image. (b) Speed function.

(c) Arrival times. (d) Segmentation.

Figure 2.9: Segmentation of an image using the fast marching method.

Then we initialize the fast marching to start at the edges of the image expanding inwards to the center. The arrival times can be seen in Figure 2.9c, blue means small arrival times and red and yellow color means larger arrival times. We get the final segmentation shown in Figure 2.9d by thresholding on the arrival times with some suitable threshold. The red line shows the border between foreground and background.

Chapter 3
Improving the Rolling Bottle Test

Asphalt or asphalt concrete is the most used pavement for paved roads all over the world and is made of a mixture of stones and a petroleum-based material called bitumen. The bitumen works as a glue and makes the stones stick together. In order to prevent that stones at the surface get loose, the affinity between the stones and the bitumen needs to be as good as possible. If stones start to detach from the pavement, it will eventually have to be repaired or redone and this is a very expensive process. One way to investigate the affinity is by the rolling bottle method.

3.1 Rolling Bottle Method

The rolling bottle method, [2], is a fast and simple way to investigate the affinity between stones and bitumen. First the stones are heated and completely covered in hot bitumen, and the stones are put to rest for a while. Then the bitumen covered stones and a glass rod are put in a glass bottle that is filled with distilled water. The water has a temperature of 5°C to prevent that the stones stick together.

The bottle with the stones is then put on a bottle rolling machine like the one shown in Figure 3.1. When rolling the bottle some of the bitumen will get teared off the stones, the glass rod in the bottle will prevent the stones from sticking together.

After rolling for six hours the degree of bitumen coverage is estimated. This is usually done using manual inspection, where laboratory assistants compare the stones to reference images with known bitumen coverage. There are also written guidelines to assist in this manual estimation. Usually this is done independently by two different observers.

Figure 3.1: A bottle rolling machine.

When the degree of bitumen coverage is estimated, the bottle with the stones are put back on the machine to roll for some more hours before the degree of bitumen coverage is estimated again. This is repeated a few times to see how the degree of bitumen coverage is changing with rolling time.

The main drawback with this method as it is done today is that the estimation of the degree of bitumen coverage is made manually by visually inspecting a small number of stones. This is a very difficult estimation problem. The result will often differ a lot between different observers and laboratories. Therefore it would be very useful to have a computer system to analyze the stones. This will make the method less subjective since the same computer program can be used in different laboratories.

3.2 Previous Work

In [64], an algorithm for trying to estimate the degree of bitumen coverage using image analysis has been developed. In the proposed method, a cyan-colored background for easy segmentation of the background has been used. To avoid sparkles and reflections in the image a cyan-colored truncated cone, with the camera in one of the bases, is used. To classify pixels either as stones or bitumen, a principal component analysis was implemented. Using the first component the images were thresholded and pixels below the threshold were classified as bitumen. In the paper, no evaluation of the method were performed and it is not possible to see how well it performs.

A more advanced method for estimating the degree of bitumen coverage was suggested in [85]. To avoid reflections in the bitumen surface, the stones are put in a crystallization dish where they were covered with distilled water. A plastic

cylinder was put around the aggregates and illuminated from outside to ensure diffuse lighting to prevent shadows to occur. A probability based segmentation method was used for segmenting the images. To train parameters in the classifier, reference images on the background, the raw aggregates and aggregates completely covered in bitumen were used.

A somewhat similar method was proposed in [52]. Like in the previous mentioned article the bitumen covered stones were put on a plate filled with de-ionized water, which was placed on a green background. Pictures were taken with a camera straight above the stones and the images were transformed to the YUV color space. Then the background was removed from the images by thresholding in the different channels in the YUV color space. The same approach was used to segment bitumen from stone. These two methods use very specific experimental setup and it could be difficult to reproduce the experiments in different laboratories.

Another method to estimate the degree of bitumen coverage was proposed by us in [45]. In this method we used series of images of stones, where the images in the series have different exposure times. Two reference series were used to get information of how a typical bitumen or stone pixel looks like. The pixels in the reference images were clustered so that a stone pixel is represented by some cluster centers and a bitumen pixel is represented with some other cluster centers. To estimate the degree of bitumen coverage in a series of images with partly covered stones, the images was segmented using a graph-cut method. The weights in the graph from a pixel to the foreground and background node depend on the distance to the closest bitumen and stone clusters. This is almost the same as using the minus log likelihood using a Gaussian Mixture Model, [62].

All the above methods rely on some color difference between the stone material and the bitumen. These methods are not suitable when analyzing darker stones, where the color difference is very small. There has also been a few attempts, [47, 66], to develop methods that do not depend on the color of the stones.

In [66], a different approach, based on the differences in the surface characteristics in bitumen and stone, is used. Bitumen and stone reflect light in different ways. In stone, diffuse reflection occurs and in bitumen specular reflection dominates. In the article, the stones have been scanned with two line lasers, perpendicular to each other. The idea is that the light that hits the stone surfaces is scattered away in all directions, where some rays will reach the camera. If the

light hits a bitumen surface then all light is reflected away in the same direction, away from the camera. This results in an image with very low intensities where it is bitumen and higher intensities where it is stone. While scanning with the lasers one gets a series of images, these images are then combined to a single gray scale image. By looking at the histogram of this image the authors claim that they can distinguish between bitumen, stone and background. The authors also claim that the color of the stones are of no significance for this method, but only one color has been tested in the article and there is no evaluation of the result. Using lasers as light source is also an expensive solution and it might not be useful for many laboratories.

Another method that uses specularities in the bitumen was presented by us in [47]. To get as much specularities in the bitumen covered part of the stones as possible, the stones were placed on a turntable that was illuminated from one side with natural light. By turning the turntable a little between images different sides of the stones were illuminated. After that all images are registered to each other, the difference of the highest and the lowest intensity is measured for all pixels. Two reference series, one of stones without bitumen and one completely covered in bitumen, are used to build histograms over the differences. The histograms are then used to create a probability function that tells what the probability is that a pixel is bitumen given the intensity difference for the pixel. The degree of bitumen coverage is estimated by summarizing these probabilities for all foreground pixels and dividing it by the total number of foreground pixels.

3.2.1 Contributions

This chapter is mainly based on the work in our article [46]. The goal is to find a method that works just as good on dark stones as it does on brighter stones. It is very difficult even for an experienced human to manually estimate the degree of bitumen coverage, especially on darker stones, and different laboratory assistants often get different answers. This makes it very hard to get any reliable ground truth.

The proposed method in this chapter is a continuation of the method in [47], which detect the specular reflections in the image and therefore is independent on which color the stones have. To evaluate the result, a somewhat simplified version of the method presented in [45] has been used. This method uses the color of the stone and is not suitable for darker stones. That method was used as evaluation method since it first segments the bitumen from the stones. Therefore the result

can be analyzed to see if it is reliable, as long as lighter stones are used on which it can clearly be seen where there is bitumen. The results have also been compared with manual estimations.

The two methods presented differ in many important aspects. The reference method estimates the degree of bitumen coverage by segmenting the image into bitumen and stone. The degree of bitumen coverage is estimated by counting the number of pixels that were classified as bitumen and stone respectively. The second method, the general method, uses the amount of specular reflections that occur in the image to estimate the degree of bitumen coverage. This works since the specular reflections only occurs in the bitumen covered part of the stones. The segmentation method in the reference method only works if there is a significant color difference between the stones and bitumen, where the general method only uses specularities and therefore the color of the stones does not matter.

3.3 General Method

The idea behind this method is to use specular reflections, that only occurs in the bitumen covered part of the stones, to estimate the degree of bitumen coverage. To detect specular reflections, several images with light from different directions are needed so that reflections occur in one or more of the images but not in the others. The most practical way to achieve this is to put the stones on a turntable that is illuminated from one side. Between images the turntable is rotated a few degrees so that the stones will be illuminated from other angles. With this setup, the images first need to be registered to each other so that they have the same orientation before the images can be analyzed. After this, the stones are segmented from the background and last, the degree of bitumen coverage is estimated by analyzing the amount of specular reflections in the images.

3.3.1 Experimental Setup

To capture images with light from many different directions, the setup shown in Figure 3.2 is used. The stones are placed on the turntable, illuminated from the side by a light source that emits light in a quarter of a circle. In this way the stones are illuminated from the top and from the side at the same time. The camera is faced so that the optical axis of the camera coincides with the rotation axis of the turntable. By rotating the turntable somewhat between the images, many pictures with light from different directions are produced.

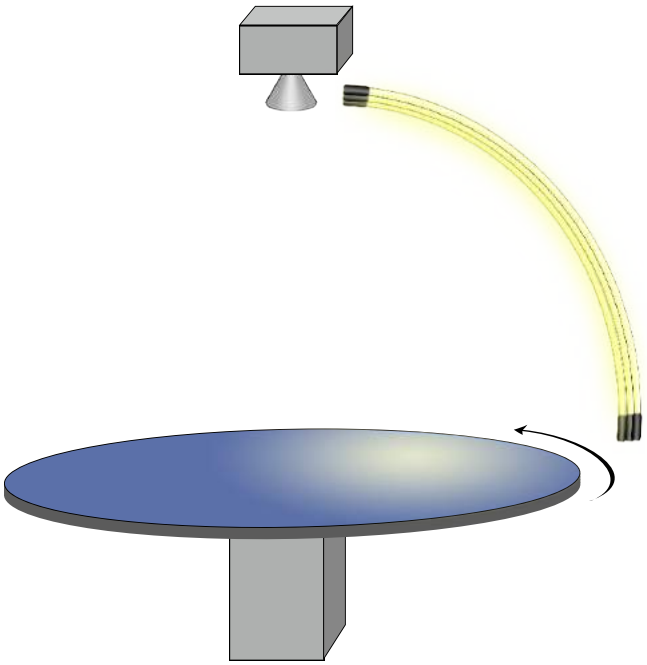


Figure 3.2: The experimental setup that is used for this method. There is a camera placed straight above the center of a turntable so that its optical axis coincides with the rotation axis of the turntable. There is also a light source placed on one side of the turntable which illuminates one side of the stones. When the turntable is rotated a few degrees, other sides of the stones will be illuminated.

3.3.2 Registration of the Images

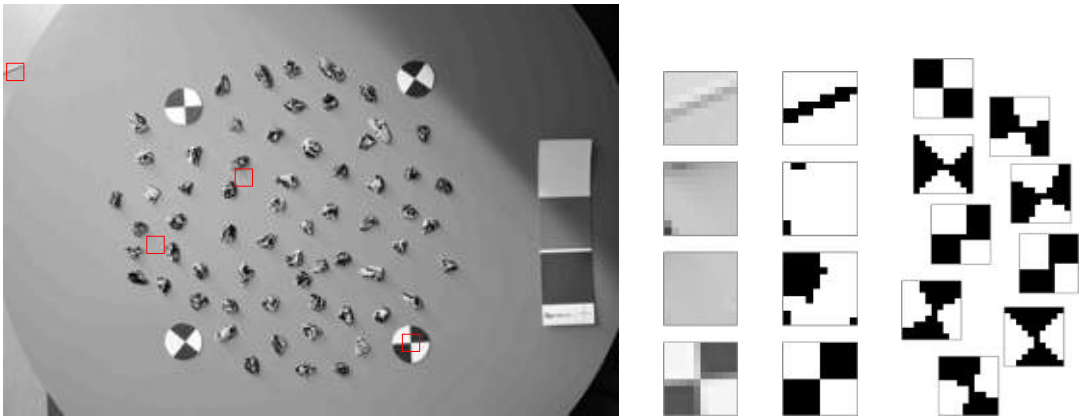
To be able to analyze the images they need to be registered to each other. The first image is used as a reference image and the others are transformed to correspond to that. This is done by extracting four key points in all images and use these points to calculate a homography from all images to the reference image. Four markers with the shape of small checker patterns are put on the turntable as these key points.

Detection of the Checker Patterns

To find the four checker patterns in the image, a sliding window detector was used. Each window is a small part from the image that is 10×10 pixels. These small image patches are converted to binary images by setting all pixels that have higher intensities than the mean intensity in the patch to white and the rest to

black. An image, in which the four checker patterns should be found, can be seen in Figure 3.3a. Four semi-random selected patches are marked with red squares and close-ups of these patches can be seen in Figure 3.3b. Figure 3.3c shows these patches after conversion to binary images.

The binary patch is then compared to a stack of binary checker patterns rotated in different ways, some patterns from the stack are shown in Figure 3.3d. For all images in the stack the differences between the binary image and the patterns are computed and the smallest difference is stored. Then, the four checker patterns are found as the four best locations in the image, where the difference between the binary patch and the best fit pattern is small.



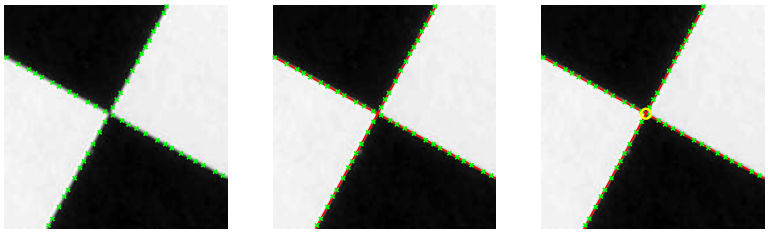
(a) (b) (c) (d)

Figure 3.3: Detection of the checker patterns. (a) the downscaled grayscale image with four patches marked with red squares, (b) close-ups of the four patches, (c) the patches converted to binary images, (d) the patterns for comparison.

Finding the Key Points

The key points are chosen as the intersection point for the four squares in the checker pattern. To find this point, points at the boundary between the white and the black areas are extracted. Here one could use sub-pixel methods, [4, 5], for improving the precision in the edge detection and thus in the precision of the key points. The small images with these points marked can be seen in Figure 3.4a as green stars. Then a RANSAC, [28], inspired algorithm was used to fit two perpendicular lines to the points, these lines are shown in red lines in Figure 3.4b. First, one line was fitted, then the points that were classified as inliers

were removed and a new line to the remaining points was fitted. The key point was found by finding the intersection between these two lines, shown as a yellow circle in Fig 3.4c.



(a) (b) (c)

Figure 3.4: Finding the intersection point in a small image of a checker pattern. (a) the detected points marked with green stars, (b) the two fitted lines in red lines, (c) the intersection point marked with a yellow circle.

Transformation of the Images

When images share the same camera center, which all images do in our setup, there exists a homography between the two images. The homography can be described by a 3×3 matrix, $\boldsymbol{H}$, that satisfies

$$\lambda \boldsymbol{x} = \boldsymbol{H}\boldsymbol{y}, \tag{3.1}$$

where $\boldsymbol{x}$ is the point in the reference image, $\boldsymbol{y}$ is the point in the image that should be transformed, $\boldsymbol{H}$ is the homography between the images and λ is a scaling factor. The points $\boldsymbol{x}$ and $\boldsymbol{y}$ are expressed in homogenous coordinates.

To estimate the homography, at least four corresponding key points are needed. The corresponding points needed is now found but in a random order. To know which point in an image in the series that correspond to a point in the reference image, the positions in the previous image in the series are stored and the order of the new points that minimizes the total distance between the new points and the points in the previous image is chosen. This works as long as the turntable is not rotated too much between two consecutive images.

Once the homography between an image and the reference image is known, it can be used to transform all points to their position in the reference image. This procedure is repeated for all the images in the series. While doing this, a mask that will be white for pixels that are inside all images and black otherwise is created.

Between two consecutive images in the series the turntable is rotated around 10°. Figure 3.5 shows some examples of those images, between two consecutive images in the figure the turntable has been rotated around 90°.

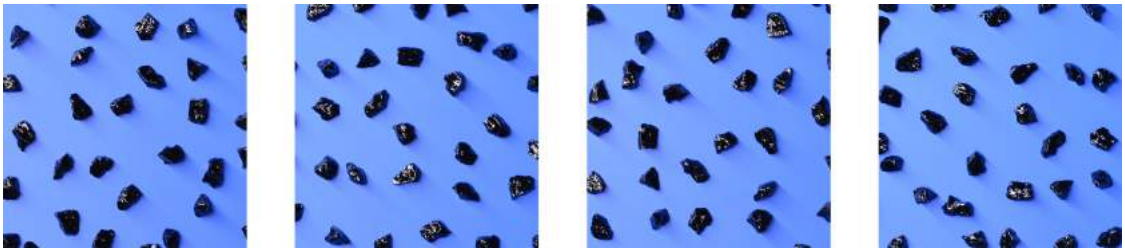


Figure 3.5: Four of the original images obtained by the experimental setup used in this experiment. The images are rotated relative each other and the light comes from the same direction in all images.

The transformed images can be seen in Fig 3.6, now the images are rotated in the same way but the light looks as it comes from different directions.

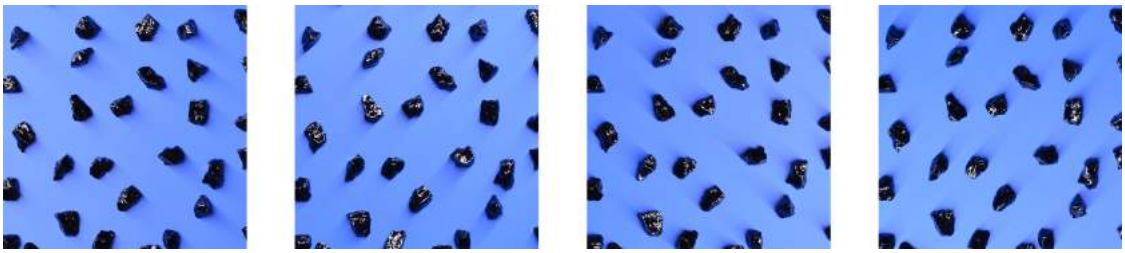


Figure 3.6: The images after transformation. Now the images are registered to each other and the light comes from different directions.

3.3.3 Segmentation of the Stones

To later be able to estimate the degree of bitumen coverage for the stones, not considering the background, the stones need to be segmented from the background. Since the shadows are quite sharp in all images, a mean over all the transformed images is calculated, this will result in an image where the shadows get smooth and almost disappear. The resulting image can be seen in Figure 3.7. This is the image that is used for segmenting the stones from the background.

The segmentation is done with a probability based segmentation method. A reference image in which a segmentation is known is used to create a three-dimensional histogram, in which the color of the background pixels are stored.

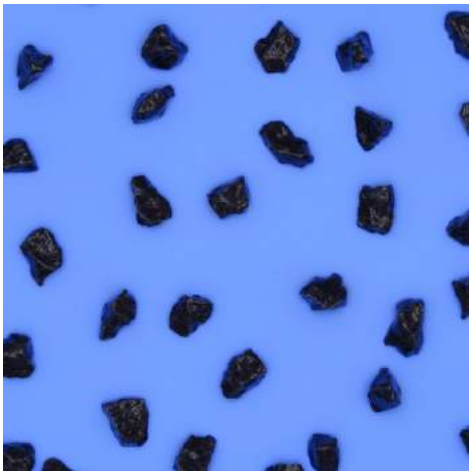


Figure 3.7: The mean image of all the registered images in the series. Here the shadows are less sharp than in the original images.

Afterwards some smoothening of the histogram was performed by convolution with a Gaussian kernel. For a histogram of size $128 \times 128 \times 128$ a kernel with $\sigma = \frac{20}{3}$ was used. A histogram for the foreground pixels was also created, but since it is unknown what it looks like, this histogram is set to be constant. From these histograms the probability that a pixel of a certain color is background can be computed. The probability matrix that a pixel is background is computed by

$$P_{bg} = \frac{H_{bg}}{H_{fg} + H_{bg}}, \tag{3.2}$$

where P_{bg} is the probability matrix, H_{bg} is the histogram for the background pixels and H_{fg} is the histogram for the foreground pixels. This technique of estimating the probability density function is similar to the Parzen-Rosenblatt window estimation technique, [70, 74].

Once the probability matrix is computed, the probability that a pixel is background for all the pixels in a new image can be computed. When this is done all pixels for which the probability for being background is more than 50 % are classified as background and the rest as foreground. The segmentation is then refined by first remove very small segments and then reduce the foreground segments by some pixels by binary erosion to ensure that no background pixels will be classified as foreground. The final segmentation can be seen in Figure 3.8. The border between the foreground and background are marked with red lines.

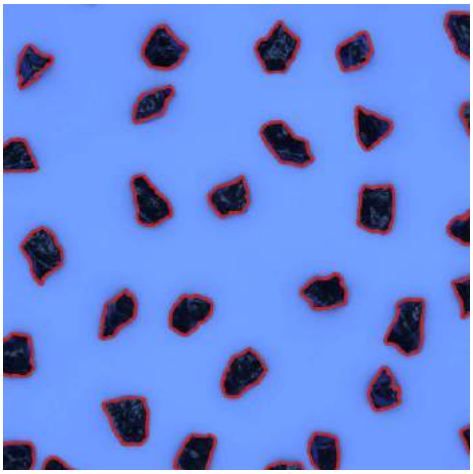
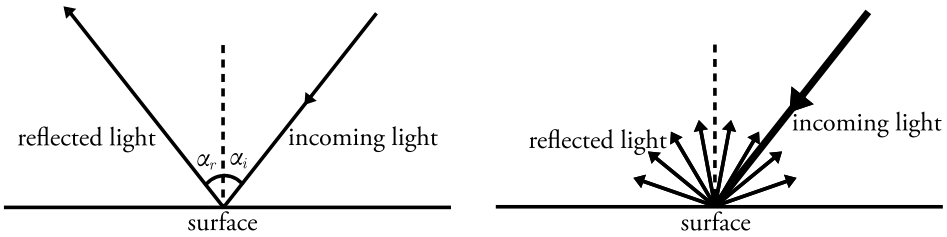


Figure 3.8: The segmentation result. The red lines show the border between foreground and background.

3.3.4 Estimation of the Degree of Bitumen Coverage

When a ray of light hits a surface one of the following things can happen; the light gets absorbed by the object, the light is refracted and continues to travel through the object or the light is reflected. It is the reflected light that is visible. There are mainly two types of reflections; specular reflection and diffuse reflection. Specular reflection happens when the surface is very smooth, in this case all of the light will be reflected in the same direction. This can be seen in Figure 3.9a, the angle between the normal of the surface and the incoming light is denoted by α_i, and the angle between the normal and the reflected light is denoted by α_r, these two angles are always equal, $\alpha_i = \alpha_r$. Diffuse reflections happens in uneven surfaces, in this case light will get scattered away in all possible directions. This is shown in Figure 3.9b. The light that hits the camera sensor will be visible in the image.



(a) Specular reflection (b) Diffuse reflection

Figure 3.9: Reflection in two different types of surfaces.

Bitumen and uncovered stones have different surface characteristics. Bitumen reflects light in the specular way, depending of the angle of the incoming light the reflected light will either hit the camera sensor, resulting in a bright spot in the image or miss the camera sensor resulting in a black spot. The uncovered stones reflect light in the diffuse way. In this case the direction of the incoming light does not matter since it will be scattered away in all directions anyway.

To estimate the degree of bitumen coverage, the amount of specular reflections in the image is investigated. A specular reflection can be detected when a pixel is very bright in one or more images in the series and much darker in other images. To find specular reflections, the difference between the highest intensity for a pixel through the whole series of images and the lowest intensity for the same pixel is calculated. The difference image is shown in Figure 3.10; white means high difference and black low. The stones in the image are completely cov-

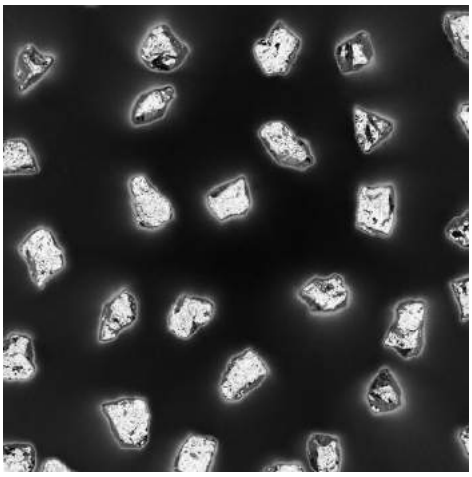


Figure 3.10: The difference between the highest intensity in all images and the lowest intensity. White means high difference and black low.

ered in bitumen. In the ideal case specular reflections occur everywhere but that is not physically possible, it depends on the orientations of the surface patches. If the distance to the camera is assumed to be much larger than the height of the stones, specular reflections cannot occur for surfaces that have larger angle than 45° between the surface and the turntable. This is illustrated in Figure 3.11. The figure shows two rays from a light source, one of them hits the top surface of the stone and is reflected towards the camera. The other ray hits the other side of the stone, this ray is reflected away from the camera and will not be visible in the image.

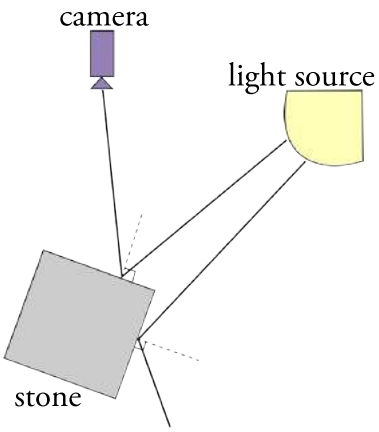


Figure 3.11

To compensate for this, a reference series with stones that are completely covered in bitumen is used and the amount of specular reflections for those stones is estimated. A specular reflection is detected when the difference between the highest and the lowest intensity is higher than some threshold. To find a good value for the threshold an image of stones completely covered in bitumen and an image of stones without any bitumen were used. By varying the threshold and detecting specular reflections an ROC curve can be achieved. True positives is defined to be pixels in the difference image of stones completely covered in bitumen with pixels values higher than the threshold and true negative is defined to be pixels in the difference image of stones not covered at all that has a lower pixel value than the threshold. In both images the background is first removed. The resulting ROC curve can be seen in Figure 3.12. To have as high sensitivity as possible and keeping specificity higher than 0.99, a threshold on 150 seems to be the best choice. This point is marked with a red point in the plot.

The amount of specular reflections is calculated as

$$a = \frac{\#\text{ foreground pixels with difference} \geq 150}{\#\text{ foreground pixels}}. \tag{3.3}$$

The amount of specular reflection in the image with stones completely covered in bitumen is denoted by a_0, then for all other images of stones partly covered, the degree of bitumen coverage is estimated as

$$doc = \frac{a}{a_0}, \tag{3.4}$$

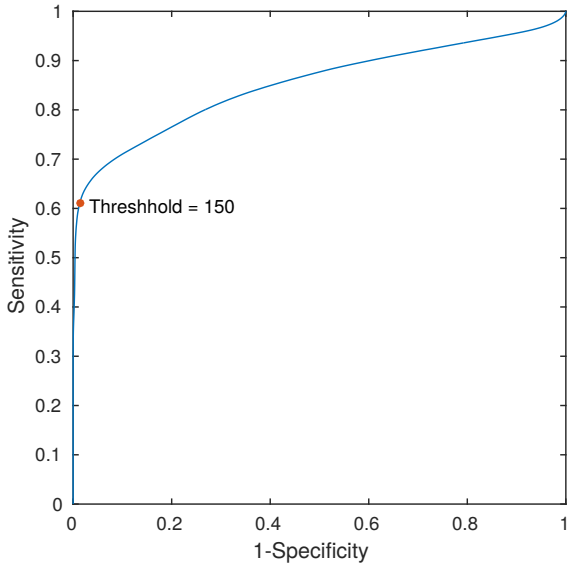


Figure 3.12: ROC curve while changing the threshold, the chosen threshold is marked with a red dot at the curve.

where doc is the degree of bitumen coverage, a is the amount of specular reflections in the current image and a_0 the amount of specular reflections in the image of stones completely covered in bitumen.

3.4 Reference Method

In this method, the images are segmented into background, stones and bitumen. In order to have a correct segmentation it is important to avoid specular reflections as much as possible. Then, the segmentation result is used to compute the degree of bitumen coverage. Two reference images were used, one of stones that is completely covered in bitumen and one of stones not covered at all. The reference images are used to get knowledge of how a typical bitumen and stone pixel look like. When this is known, the degree of bitumen coverage for some test images of stones partly covered in bitumen can be estimated. To do that, the stones are first segmented from the background. For each stone a small graph is created and the bitumen is segmented from the stone using a graph-cut algorithm. Once the bitumen is segmented from the stones, the degree of bitumen coverage can be estimated by counting the number of bitumen and stone pixels.

3.4.1 Preparations

The images used in this method comes from the same experimental setup that was used in Section 3.3. Instead of using the difference images, an image was created by using the lowest intensity for the pixel in the series of images. In this way, specular reflections are avoided in the images. Since very bright stones were used it will still be a clear color difference between the stones and the bitumen.

Two reference images and a number of test images are used as input to the method. The reference images are used to get knowledge of how a typical stone or bitumen pixel look like in the images, there is one depicting stones that are not covered at all and one depicting stones completely covered in bitumen. For the test images, the degree of bitumen coverage should be calculated. Some examples of reference images and test images can be seen in Figure 3.13.

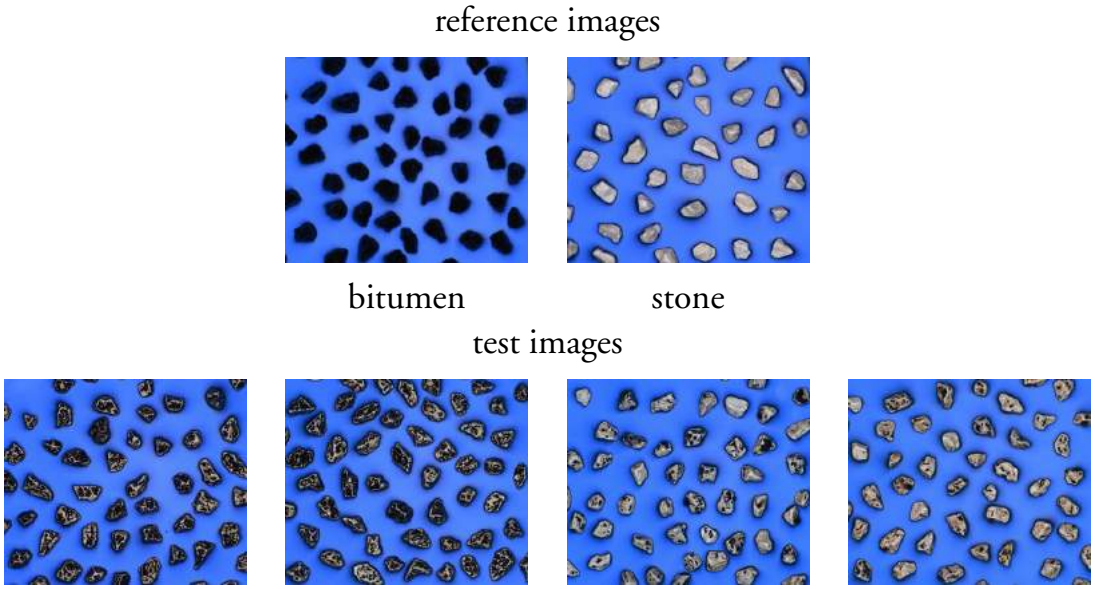


Figure 3.13: Some examples of reference images and test images. One image of stones completely covered in bitumen and one image of stones not covered at all was used as reference images. The test images are images of stones partly covered in bitumen for which the degree of bitumen coverage should be estimated.

Intensity Adjustment

To be able to compare intensities in different images to each other, they need to have the same intensity on the background. To ensure this, a reference stick was

put in the images with three different fields of white, light gray and dark gray. These fields can be used to adjust the intensities in the images to correspond to a reference image. To find the fields in the image a search for large homogeneous areas, areas without edges, was performed. Figure 3.14 shows a plot for the intensities in one channel for the three fields for two images. The intensities for the image that should be adjusted is on the x-axis and the intensities for the reference image is on the y-axis. A linear function $y = kx + m$ was fitted to the points using the least square method.

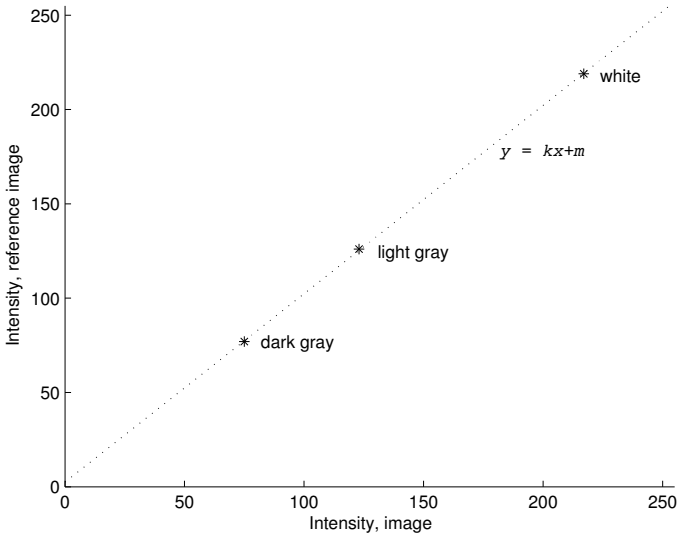


Figure 3.14: Intensities in one channel for a reference image and the image that should be adjusted. The intensities for the three fields for the image that should be adjusted is on the x-axis and the intensities for the reference image is on the y-axis. A linear function with slope k and constant term m is fitted the the points.

The intensities in one channel can be adjusted by computing

$$I_{new} = kI_{old} + m, \tag{3.5}$$

where I_{new} is the intensity after adjustment, k is the slope of the linear function, I_{old} is the intensity for the image before adjustment and m is the constant term for the function. This adjust one channel in the image, the other two are adjusted in the same way.

Clustering

To find out how a typical stone or bitumen pixel looks like, the two reference images were used to cluster the foreground pixels into different clusters where pixels that belongs to the same cluster are similar to each other. First the background has to be removed so that only stone or bitumen pixels are clustered. The segmentation obtained by the segmentation in Section 3.3 was used.

The clustering is done using a k-means clustering algorithm that is described in [61]. The clusters are represented by its cluster center that is the mean value of all the pixels in the cluster. Figure 3.15 shows the process of cluster the pixels for a small image of one stone. The original image is shown in Figure 3.15a. After that the background is removed, the image in Figure 3.15b was obtained, the pixels belonging to the foreground are then clustered into five different cluster, shown in Figure 3.15c. The image in Figure 3.15d shows which pixel in the image that belongs to which cluster.

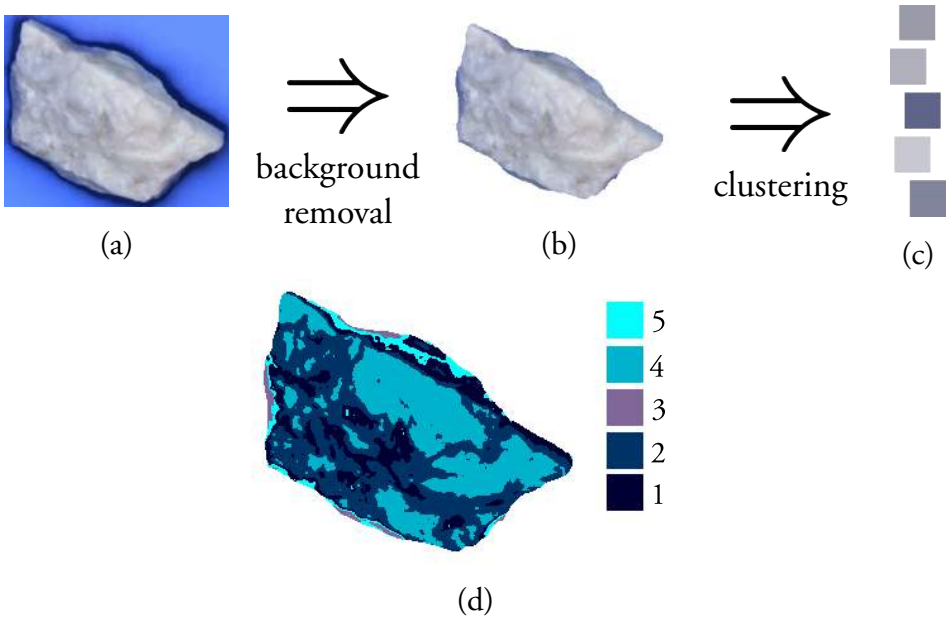


Figure 3.15: The clustering process for one stone. (a) the original image, (b) the image with the background removed, (c) the five clusters and (d) shows which pixels that belong to which cluster.

To investigate how the number of clusters affect the segmentation result, ground truth for a small number of stones were created. After removing the background from the reference images they were clustered into clusters, varying

the number of clusters. The same number of clusters was used for bitumen and stone pixels. Using this clusters the bitumen was segmented from the stones and the Jaccard score between the segmentation and the ground truth was calculated. The Jaccard score for two segmentations is defined to be the quotient between the intersection of the segmentations and the union of them. The score is between 0 and 1, where 1 is a perfect segmentation according to the ground truth. The initialization of the k-means algorithm is done by randomly place cluster centers, therefore the clusters will differ somewhat running k-means more than once, this will also affect the segmentation result. To evaluate the number of clusters the experiments were performed five times with different cluster centers. The result of this is shown in Figure 3.16, where the blue lines shows the Jaccard score for an individual trial and the thicker black line is the mean over the five experiment. As can be seen the difference is very small for everything between 2 and 7 clusters. A number in between, 5, is chosen to reduce processing time but still make sure that the method will still work on stones with higher variance in appearance.

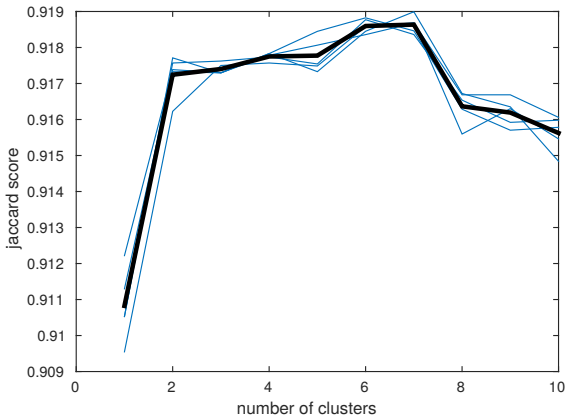


Figure 3.16: Jaccard score for different number of clusters used to cluster the bitumen and stone pixels.

3.4.2 Segmenting Bitumen from Stone

To segment the bitumen covered part of the stone from the uncovered parts, a graph-cut method [11, 12, 49, 10] was used. When segmenting with the graph-cut method one represent the image as a graph where the nodes represent the pixels in the image. The nodes are connected with edges that have certain weights; the weights can be the same for all pair of pixels or be set individually for each pair. All the nodes are then also connected to a foreground node, called S-node, and a

background node, called T-node. These edges have weights that typically depend on the intensity difference between the pixel and the foreground and background node respectively. The segmentation is obtained by cutting edges in the graph so that there does not exist a path from the S-node to the T-node and set the pixels connected to the S-node to foreground and the rest to background.

Calculating the Weights

The bitumen covered part of the stone are set to foreground and the rest to background. To set the weights between pixels and the foreground node, the distance to the cluster centers for the bitumen pixels computed in Section 3.4.1 is used. This will measure how similar a pixel in the image is to a bitumen pixel. The distance from a pixel to a cluster center is computed by

$$d_{ik} = \|\boldsymbol{p}_i - \boldsymbol{p}_k\|_2, \tag{3.6}$$

where d_{ik} is the distance from pixel i to cluster center k, $\boldsymbol{p}_i$ is the tree dimensional pixel value for pixel i and $\boldsymbol{p}_k$ is the vector that describe the cluster center for cluster k.

A pixel is more likely to be bitumen if the pixel lies closer to a bitumen cluster than to a stone cluster. If a bitumen cluster is very close and the closest stone cluster is far away the possibility for being bitumen is much higher than if the closest bitumen cluster and the closest stone cluster are equally far away. Therefore, the weight from a pixel i to the S-node, denoted by w_{iS} is set to be the normalized distance

$$w_{iS} = \frac{d_{ib}}{d_{ib} + d_{is}}, \tag{3.7}$$

where d_{ib} is the distance from pixel i to the closest bitumen cluster and d_{is} is the distance from the pixel to the closest stone cluster.

The weight from pixel i to the T-node, w_{iT}, is then set to

$$w_{iT} = 1 - w_{iS}. \tag{3.8}$$

The weights between two neighboring pixels depends on the color difference between the pixels and is set to

$$w_{ij} = e^{-d_{ij}^2/2\sigma^2}, \tag{3.9}$$

where w_{ij} is the weight on the edge between pixel i and pixel j, σ is a constant that controls how fast the exponential function decreases. The term d_{ij} is the normalized color difference between pixel i and pixel j and is calculated as

$$d_{ij} = \frac{\|\boldsymbol{p}_i - \boldsymbol{p}_j\|_2}{255\sqrt{3}}, \tag{3.10}$$

where $\boldsymbol{p}_i$ is the color for pixel i and $\boldsymbol{p}_j$ is the color for pixel j. By this normalization we get that $0 \leq d_{ij} \leq 1$.

To find a good value for σ, ground truth was created for a few stones by manually segmenting bitumen from stone. For these stones segmentation with graph-cut was performed with different values on σ and the Jaccard score between those segmentations and the ground truth were calculated. A plot of the Jaccard scores is shown in Figure 3.17. As can be seen in the plot there is a very small difference in the Jaccard scores for $0 \leq \sigma \leq 0.03$. The value $\sigma = 0.02$ was chosen and by looking at the segmentation result for a larger set of stones without ground truth, it was confirmed that it was indeed a suitable value.

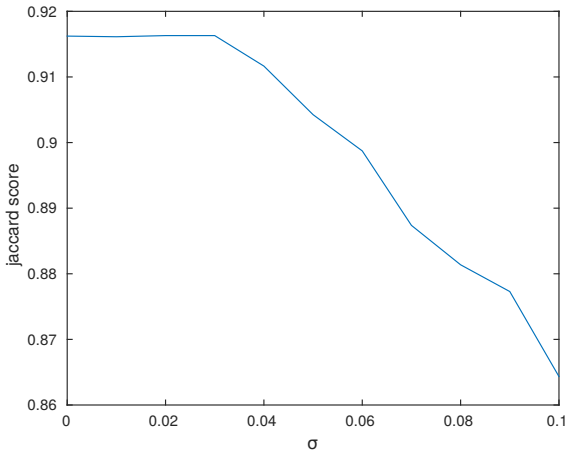


Figure 3.17: Jaccard score for different values of σ, the optimal value should be somewhere between 0 and 0.03.

3.4.3 Estimation of the Degree of Bitumen Coverage

Once the segmentation is done, the degree of bitumen coverage is calculated by simply count the number of pixels that are classified as bitumen and divide it by the total number of foreground pixels. The degree of bitumen coverage is

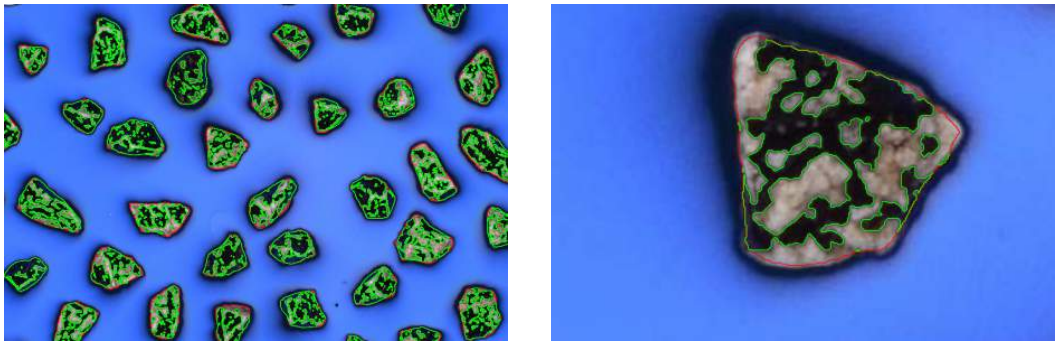
calculated by

$$doc = \frac{\text{\# bitumen pixels}}{\text{\# foreground pixels}}. \tag{3.11}$$

3.5 Experiments

Four bottles with stones were analyzed with the two image analysis methods, two of the bottles were rolled for six hours and the other two were rolled for 24 hours. For each bottle three series of images were taken, between the series the stones were collected and replaced on the turntable. There are also a manual estimation from three laboratory assistants for each bottle.

The segmentation result, using the reference method, for the first trial for the four bottles can be seen in Figure 3.18 - 3.21, the red lines show the border between the foreground and the background and the green lines show the border between bitumen and stone. The images to the right are close-ups of some stones in the corresponding left image.

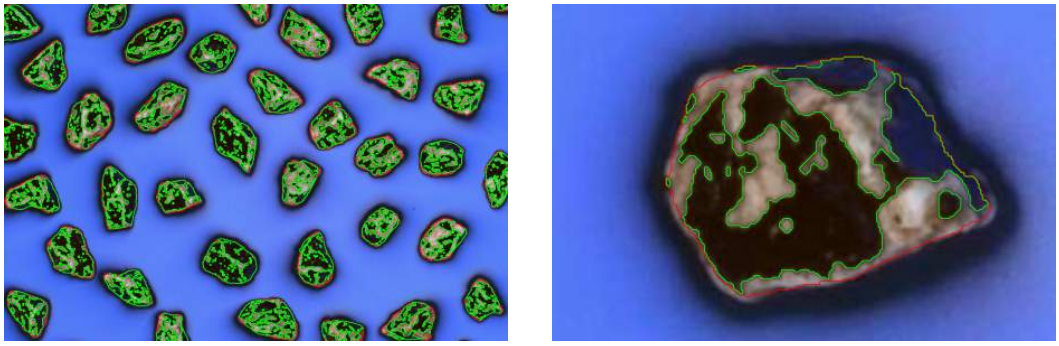


(a) (b)

Figure 3.18: The segmentation result for the first bottle. The red lines show the border between foreground and background and the green lines show the border between bitumen and stone. (b) is a close-up of one stone in (a).

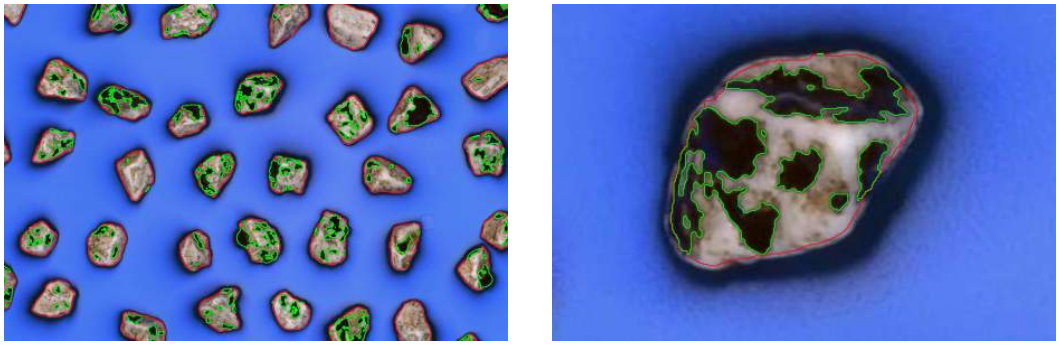
As can be seen in the images the green lines fit very well to what is expected to be the border between bitumen and stone. This implies that the segmentation is very good and that the estimated degree of bitumen coverage should be close to the true value.

The estimated degree of bitumen coverage can be seen in Table 3.1. The table shows the estimated vales for both image analysis methods and the values



(a) (b)

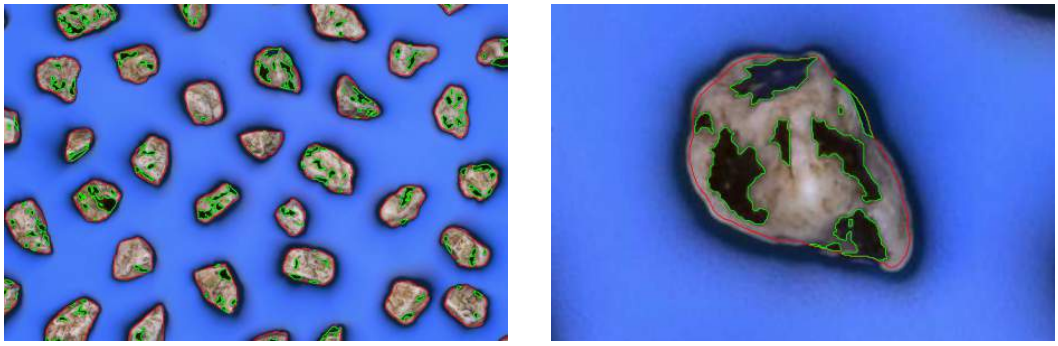
Figure 3.19: The segmentation result for the second bottle. The red lines show the border between foreground and background and the green lines show the border between bitumen and stone. (b) is a close-up of one stone in (a).



(a) (b)

Figure 3.20: The segmentation result for the third bottle. The red lines show the border between foreground and background and the green lines show the border between bitumen and stone. (b) is a close-up of one stone in (a).

estimated by three different laboratory assistants. For the image analysis methods there are three series of pictures with the stones places differently for each bottle, while there is only one measurement per laboratory assistant. As described in the rolling bottle test standard, the manual estimation is always performed by at least two laboratory assistants, the result of the test will be the mean over the measurements of the assistants rounded to the closest 5 %. For this evaluation however, it is also of interest to see the variation between different assistants and different placing of the stones, therefore all numbers are presented in the table below.



(a) (b)

Figure 3.21: The segmentation result for the fourth bottle. The red lines show the border between foreground and background and the green lines show the border between bitumen and stone. (b) is a close-up of one stone in (a).

Table 3.1: The result of the two image analysis methods and manual estimations of three laboratory assistants.

trial	ref. method	gen. method	ass 1	ass 2	ass 3
PA_1	58.1	45.5			
PA_2	60.6	48.4	80	70	80
PA_3	58.2	46.4			
P5_1	53.8	42.7			
P5_2	57.2	49.2	75	70	80
P5_3	58.4	52.1			
P4_1	18.1	14.3			
P4_2	20.1	19.3	15	15	15
P4_3	18.7	17.4			
P1_1	15.9	12.7			
P1_2	18.0	13.5	18	15	15
P1_3	17.1	14.1			

The manual estimations are not the ground truth and the degree of bitumen coverage differs in many cases between the laboratory assistants. The estimations are done according to some predefined guidelines but it is still difficult to estimate the percentage by hand. Since the segmentation we get from the reference method can be visualized and it agrees well with what is expected to be bitumen and stone, the estimated value for the degree of bitumen coverage can be assumed to be very

close to the true value and those values are used as the truth. It can also be seen in the table that the general method seems to underestimate the degree of bitumen coverage a bit, but the underestimation is quite consistent for all the trials.

A simplified version of Table 3.1 is shown in Table 3.2. The table shows the mean value of the different trials for all bottles for both image analysis methods and by manual estimations, all rounded to the closest 5 %.

Table 3.2: The mean values of the result of the two image analysis methods and manual estimations of three laboratory assistants rounded to the closest 5 %.

trial	ref. method	gen. method	manual
PA_1	60	45	75
P5_2	55	50	75
P4_2	20	15	15
P1_2	15	15	15

The quotient between the values estimated by the two image analysis methods can be computed. Doing that, it can be noticed that quotient does not differ significantly between the different trials. To find an optimal value, k, to multiply the values estimated by the general method we have minimized the function

$$\min_k \sum_{i=1}^{n} \left(1 - k\frac{y_i}{x_i}\right)^2, \tag{3.12}$$

where x_i are the estimated coverage by using the reference method and y_i is the estimated values for the general method.

Minimizing this gives

$$k^* = \frac{\sum_{i=1}^{n} y_i/x_i}{\sum_{i=1}^{n} (y_i/x_i)^2}. \tag{3.13}$$

To cross-validate our results two trials per bottle has been used to estimate the optimal quotient for those values and tested on the rest. This was performed for all three possibilities to choose two trials for the optimization. The result of this can be seen in Table 3.3, in the table the red values are the ones not used in the optimization but the ones that was tested on. The relative mean error was

calculated as

$$e = \frac{1}{n}\sum_{i=1}^{n}\frac{|k^* y_i - x_i|}{k^* y_i}, \tag{3.14}$$

where k^* is the optimal quotient obtain by (3.13) and x_i and y_i are the estimated values for the reference method and the general method respectively.

Table 3.3: Cross-validation of the optimizations results and the relative mean errors.

trial	ref. method	1 and 2	1 and 3	2 and 3
PA_1	58.1	55.4	54.8	53.1
PA_2	60.6	59.0	58.4	56.5
PA_3	58.2	56.4	55.9	54.1
P5_1	53.8	52.0	51.5	49.8
P5_2	57.2	59.9	59.4	57.4
P5_3	58.4	63.4	62.8	60.8
P4_1	18.1	17.4	17.3	16.7
P4_2	20.1	23.5	23.3	22.5
P4_3	18.7	21.2	21.0	20.3
P1_1	15.9	15.4	15.3	14.8
P1_2	18.0	16.4	16.3	15.7
P1_3	17.1	17.1	17.0	16.4
e		0.0584	0.0617	0.0751

As can be seen in the table, the two methods do not differ significantly after it has been compensated with the quotient, k, and there is no significant difference depending on which trials used for optimization. This result implies that some stones with clear difference between stone and bitumen can be used to calibrate the general method to get very close to the true degree of bitumen coverage.

3.6 Conclusions

A method to estimate the degree of bitumen coverage of stones partly covered in bitumen that does not depend on the color of the stones has been presented. The results have been compared with the result of a reliable method using stones of

light rock type and the results are consistent with those. Lighter stones for which the degree of bitumen coverage can be estimated using other methods can be used to calibrate the variables in the method to give result that are very close to the true coverage. Even without the calibration, the method can be used to compare different stone materials to each other. The result is objective and will not change depending on which laboratory that use the program.

Chapter 4

Grain Size Distribution in Asphalt Samples

One of the quality controls of hot mix asphalt (HMA) is to check if the grain size distribution in the asphalt follows the recipe of the mixture. This check could be performed on HMA directly from the plant or on samples that are drilled from the finished pavement. The quality check includes determination of particle size distribution via sieving. The test consists of the determination of the particle size distribution of the aggregates in the bituminous mixture by sieving and weighing. A granulometric analysis of the aggregate is performed after binder extraction. The binder extraction is often performed with methylenchloride (dichloromethane). Methylenchloride is toxic and the European Union has decided to reduce the usage of methylenchloride.

After the binder distraction the material is sieved in order to determine the particle size distribution. The method proposed in this chapter is an alternative to binder extraction in combination with sieving. The asphalt sample to be analyzed is sliced into a few slices. Then each two-dimensional slice is analyzed by image analysis methods to find the size distribution for all slices. The work in this chapter is based on the work in our article [48].

4.1 Related Work

There are a few attempts to estimate the grain size distribution from two-dimensional cross-sections of asphalt samples. The article [17] combines a canny edge detection with thresholding to find an initial segmentation of the image. They use the watershed algorithm to deal with the problem of grains being too close to each other and therefore segmented as one segment. The watershed algorithm is performed on a distance image, the binary image of the foreground after applying

boolean distance to it. In this way they separate segments with concavities along the contours.

In [83] they use the L*a*b color space to segment the grains from the bitumen background. After converting to the L*a*b color space they easily find a good threshold and then they perform thresholding. The same problem as mentioned in the earlier article with undersegmented segments they also find and solve it in the same way with the watershed algorithm.

Both articles compare to the true aggregate gradation and both show good correlations. However, it is unclear how they from the image segmentation calculates the size distribution.

4.2 Finding the Stones

To estimate the size distribution of the stones in one slice of the sample we first segment the image by using the fast marching method. In the resulting segmentation it often happens that stones lying too close to each other belongs to the same foreground segment even if they are not supposed to. In order to get correct size estimations we need to separate these segments. This is done by applying some morphological operations.

4.2.1 Segmentation by Fast Marching

To segment the stones from the background the fast marching method described in Section 2.4 was used. To know at what speed the curve should advance at a certain point we need a speed function. We choose to use a logistic function defined by

$$F(x,y) = \frac{1}{1 + e^{I(x,y)/v}}, \tag{4.1}$$

where $F(x,y)$ is the speed at point, (x,y), $I(x,y)$ is the intensity at the same point in the input image and v is a scaling factor that controls the steepness of the function. The function for three different values on v can be seen in Figure 4.1.

The speed function is then applied to the original image converted to grayscale. The original image and the speed image can be seen in Figure 4.2.

The fast marching algorithm produces the arrival times for every grid point, pixel in this case. The image is then segmented by thresholding the arrival times.

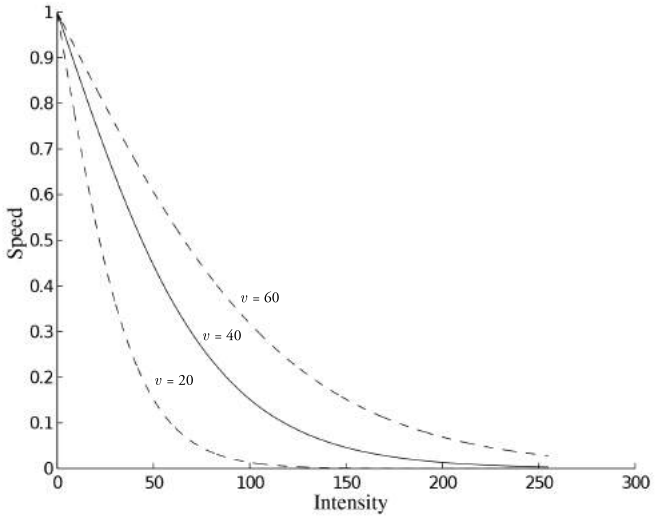


Figure 4.1: The speed function used with three different values on the steepness factor v. On the y-axis is the speed and on the x-axis is the pixel intensity.

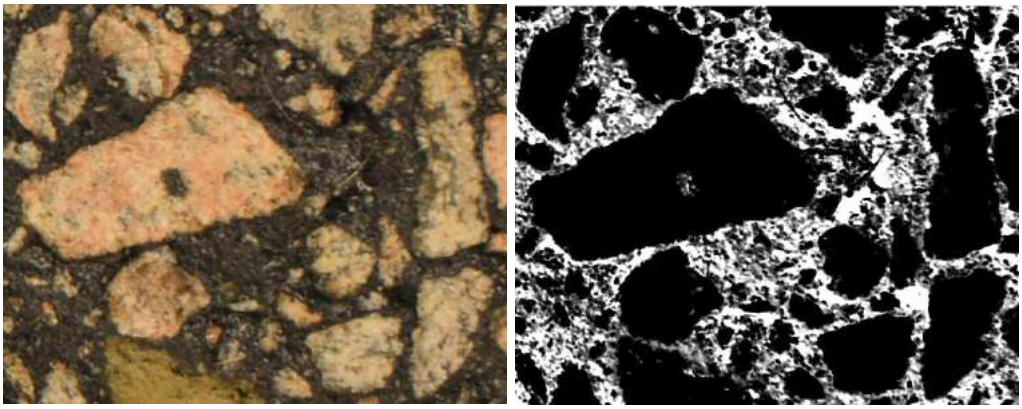


Figure 4.2: The original image to the left and the speed image to the right.

Pixels with arrival times smaller than the threshold are set to background and the rest to foreground.

4.2.2 Refining the Segmentation

Sometimes when stones are very close to each other in the sample it is not enough bitumen between them, and the fast marching algorithm have a hard time separating those stones into different segments. Separating the segments would require a high threshold, but using a too high threshold would make it impossible to find

the smaller stones in the sample. Therefore we choose a lower threshold and try to deal with the undersegmented segments in another way. Figure 4.3 shows such a segment.

Figure 4.3: A segment with strange shape that probably should be two different segments. The red line shows the border of the segment.

For this purpose we use the morphological methods erosion and dilation in a clever way. First we perform binary erosion on the segments. This will, if the segments have concavities, cause the segment to eventually fall apart into two or more segments. If the size of the segment, after performing erosion repeatedly without the segment falling apart, is less than half of the original size, we assume that the segment represent one stone and use the original segment. When it does fall apart, we continue the process on the new smaller segments keeping count on how many times we perform the erosion. Afterwards we perform dilation the same number of times that we performed erosion until the last separation of the segment. In this way all non-convex segments get separated in two or more new segments. Some smoothing of the contour will also occur. This will not cause any problems for the overall method since we are not interested in the shape of the segment itself, only in the size of it. Both erosion and dilation are performed with a disc of radius 4 pixels as structuring element.

Separation of segments

For all segments:
    until the segment is too small:
        perform binary erosion
        if separated:
            run the algorithm again on the new segments
    dilate if the segment was separated earlier

This will in some cases cause some overlap between different segments. Since we do not want that, we find these intersections of segments, and assign all those pixels to belong to either of the overlapping segments. This is done by first subtracting the overlap from all the interesting segments and then iteratively alternate between first dilate one of the two segment, assign the pixels in the intersection between the overlap and the dilated segment to that segment and then perform the same thing with the other segment. This continues until all points in the overlap are assigned to one of the segments.

After applying this algorithm to the segments in Figure 4.3 we get the result shown in Figure 4.4. The image shows the original segment with the border of the new ones drawn with red contours.

4.3 Size Estimation

To estimate the size of the stones we use the method described in [84], slightly adjusted, to find the rectangles that fit the segments best. The best-fit rectangle is the rectangle with the smallest width that the segment fit into.

The method consists of two main steps, first the orientation of the segment is estimated using the least-second moment method. Then the rectangle is fit using the Multiple Ferret method. The idea is to find the orientation by finding the line through the object that minimizes the total moment. Figure 4.5 shows a segment with such a line through the center of mass of the segment. This line has the direction $(\cos\theta, \sin\theta)$, where θ is the angle between the line and the x-axis. We translate the segment so that the origin coincides with the center of mass of the segment.

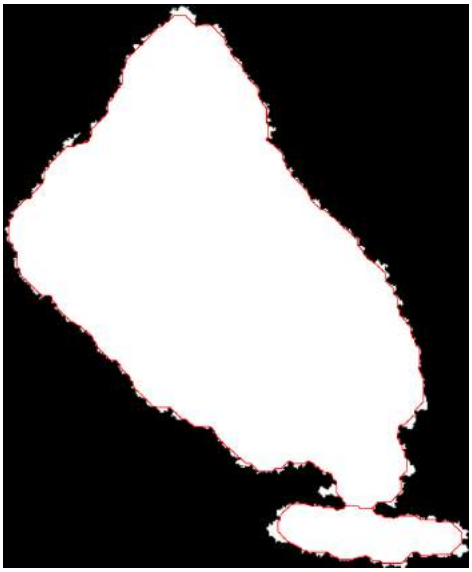


Figure 4.4: The segment after separation to two new smaller segments. The red lines indicates where the borders of the separated segments are.

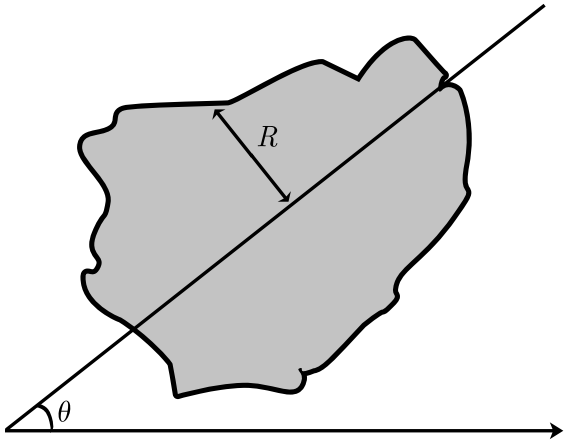


Figure 4.5: The main rotation axis.

The total moment is expressed by the integral

$$E = \iint_{\Omega} R^2 \, \mathrm{d}x\mathrm{d}y, \tag{4.2}$$

where R is the perpendicular distance from the point (x, y) to the line we seek and Ω is the integration area, the segment we want to find the orientation of.

The distance, R, for a point $\boldsymbol{x} = (x, y)$ can be achieved by taking the dot

product between the vector from the origin to the point and the normalized normal to the line. The normal is given by $(-\sin\theta, \cos\theta)$. Then we can write the integral in equation (4.2) as

$$E = \iint_{\Omega} (-x\sin\theta \ + y\cos\theta)^2 \ \mathrm{d}x\mathrm{d}y$$
$$= \iint_{\Omega} x^2\sin^2\theta - 2xy\sin\theta\cos\theta + y^2\cos^2\theta \,\mathrm{d}x\mathrm{d}y. \quad (4.3)$$

By introducing

$$I_{xx} = \iint_{\Omega} x^2 \,\mathrm{d}x\mathrm{d}y,$$
$$I_{xy} = \iint_{\Omega} xy \,\mathrm{d}x\mathrm{d}y \text{ and}$$
$$I_{yy} = \iint_{\Omega} y^2 \,\mathrm{d}x\mathrm{d}y,$$

we can write this as

$$E = \begin{pmatrix} -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{pmatrix} \begin{pmatrix} -\sin\theta \\ \cos\theta \end{pmatrix}. \quad (4.4)$$

Minimizing this is equivalent to minimizing the quadratic form $\boldsymbol{w}^T \boldsymbol{I} \boldsymbol{w}$, subject to $|\boldsymbol{w}| = 1$, where

$$\boldsymbol{I} = \begin{pmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{pmatrix} \text{ and } \boldsymbol{w} = \begin{pmatrix} -\sin\theta \\ \cos\theta \end{pmatrix}.$$

We seek the direction, $\boldsymbol{u}$, that minimizes this quadratic form. This can be found as the eigenvector that corresponds to the smallest eigenvalue of the matrix $\boldsymbol{I}$. To fit the box around the segment we also need a vector $\boldsymbol{v}$ that is perpendicular to $\boldsymbol{u}$. Since $\boldsymbol{I}$ is symmetric, the two eigenvectors will be orthogonal, therefore we find the vector $\boldsymbol{v}$ by taking the other eigenvector of $\boldsymbol{I}$.

If we know the orientation of the object we can find a box around the segment by using dot products. We start by extracting all boundary points of the segment. For all these boundary points, $\boldsymbol{x} = (x, y)$, we calculate the dot product between

the point vector and the vector $\boldsymbol{u}$ we calculated before as the direction of the main rotation axis.

By taking the dot products between a point vector, $\boldsymbol{x}$, and the vector $\boldsymbol{u}$ we get the length of the orthogonal projection of $\boldsymbol{x}$ on $\boldsymbol{u}$. If we store the points that gives the highest and lowest value of the dot product we get the points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ shown in Figure 4.6. The points together with the direction $\boldsymbol{v}$ gives us the two lines l_1 and l_2 that both are tangents to the segment, where the vector $\boldsymbol{v}$ is calculated as mentioned before.

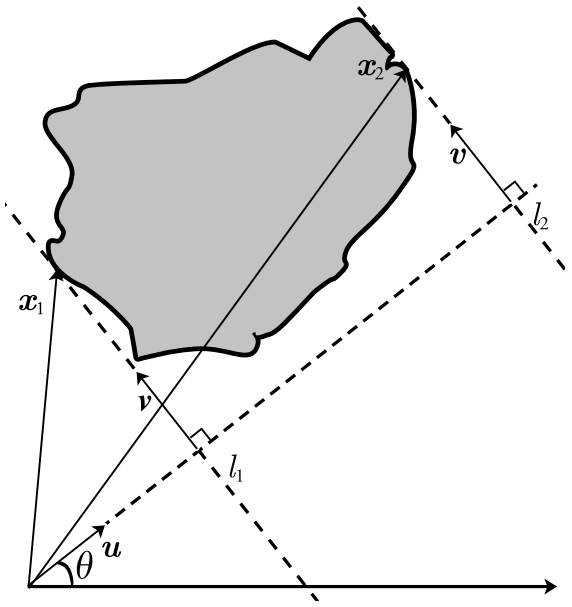


Figure 4.6: The points that gives the smallest and largest dot product between itself and $\boldsymbol{u}$. The vector $\boldsymbol{v}$ is perpendicular to $\boldsymbol{u}$ and shows the direction of the lines l_1 and l_2.

In the same manner we can calculate the dot product between the boundary points and the vector $\boldsymbol{v}$, find the maximum and the minimum and receive the lines l_3 and l_4, shown in Figure 4.7. The corners of the rectangle are then achieved by finding the four intersections of the lines.

The orientation of the segment that we get from minimizing the total moment will just give us an approximative estimation of the angle of the rectangle. To find the rectangle with the smallest width that the segment fit into we try a few directions around $\boldsymbol{u}$ and $\boldsymbol{v}$ by rotating those vectors with some small positive and negative angles. For all those directions we calculate a rectangle and then we choose the one with the smallest width.

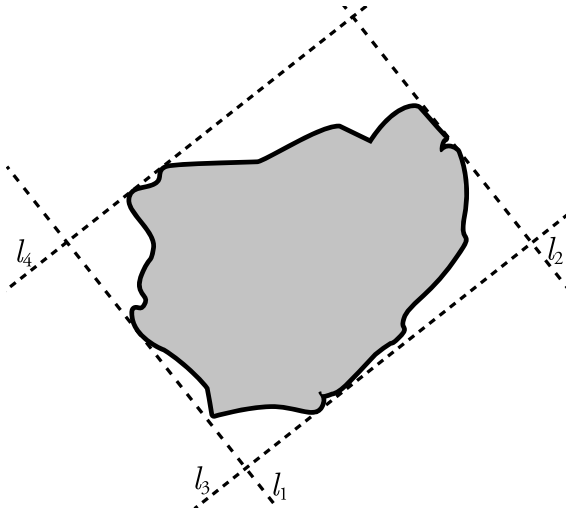


Figure 4.7: The four lines that surround the segment.

Figure 4.8 shows the best fit rectangle for some segments, the red lines show the rectangles using the angle estimated by the total moment method and the cyan colored lines show the rectangles after optimization.

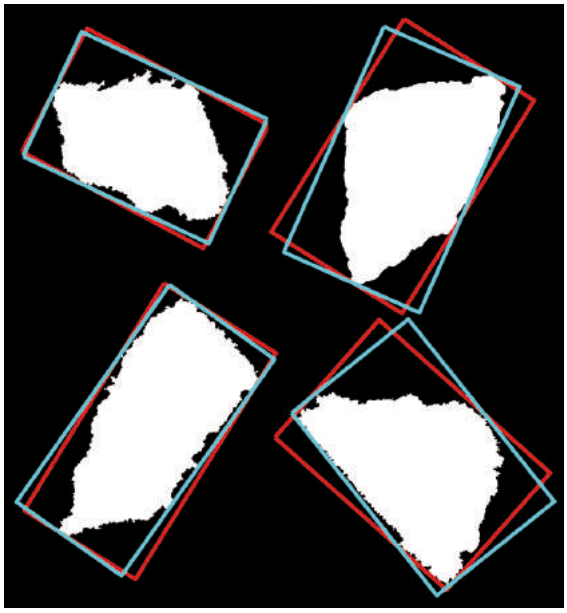


Figure 4.8: Four segments with the best fit rectangle in cyan colored lines and the rectangle obtained with the angle estimated by the total moment method shown in red lines.

4.3.1 Estimation of the Size Distribution

To estimate the size distribution of the stones in the asphalt sample we want to estimate the weight percentage of stones that passes a certain sieve size. First we estimate how many millimeters one pixel corresponds to by choosing two points in the image we know the true distance in millimeters for and compare this distance with the distance in pixels. Then for each sieve size we calculate the total area of the segments that have width smaller or equal to the current sieve size and divide it by the total area of all segments. The percentage of weight, S_s, that passes sieve size s is calculated by

$$S_s = 100\frac{A_s}{A_T}, \tag{4.5}$$

where A_s is the total area of the stones passing sieve with size s and A_T is the total area of the stones in the cross section.

Due to limitations of the camera we can not see stones that are too small, therefore we have to adjust our curve somewhat. We assume that we should be able to find stones bigger than 0.5 mm but that it could be hard to find stones much smaller. Therefore we omit stones smaller than 0.5 mm in our analyze by subtracting the percentage of passing sieve with size 0.5 mm and renormalize. The new passing percentage is calculated by

$$\hat{S}_s = 100\frac{S_s - S_{0.5}}{100 - S_{0.5}}. \tag{4.6}$$

4.4 Results

Figure 4.9 shows an asphalt sample with the best-fit rectangle for the detected stones in cyan-colored lines. The right image shows a close up of some part of the left image. By looking in the image we see that the boxes seem to fit what we would expect is the stones in the sample quite well. All the visible stones in this sample also seem to be surrounded by a box indicating that the method works well to find the stones in the two dimensional image.

We have analyzed four slices of an asphalt sample and compared to the given recipe of the asphalt. The sample was also analyzed in the standard way by dissolving the samples in methylenchloride followed by sieving and weighing. Both of the later curves were renormalized by omitting the smaller sizes as described

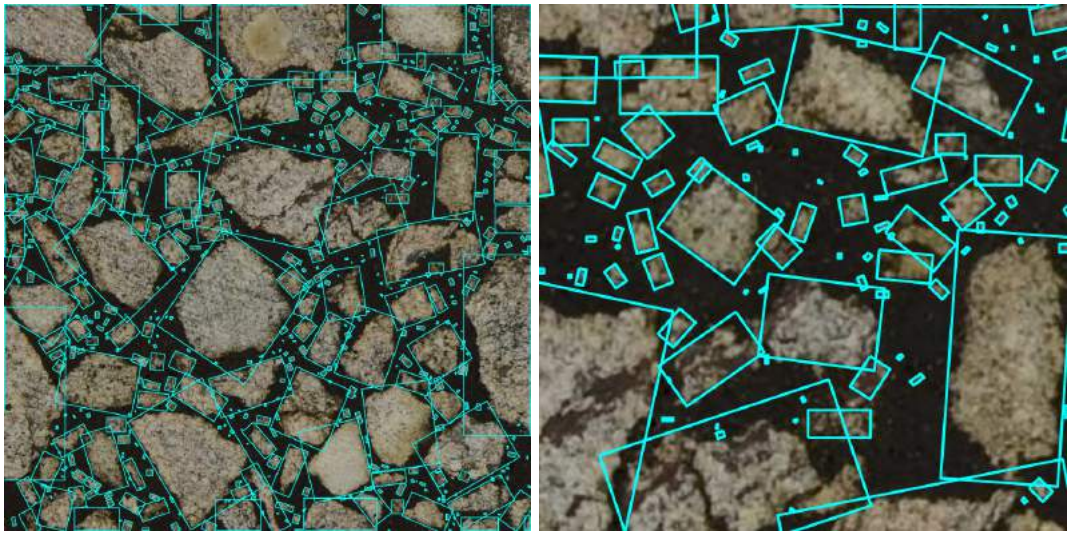


Figure 4.9: Original image with the best-fit rectangle marked with cyan colored lines, the right image is a close up of a part of the left image.

earlier. Figure 4.10 shows the size distribution of the grains. On the y-axis is the percentage of stone mass passing the sieve for different sieve sizes on the x-axis. The black curve shows the distribution according to the recipe of the asphalt, the dashed black curve shows the size distribution according to the analyze done by the standard method and the red curve shows the size distribution given by the image analysis method described in the chapter. The curve is a mean over the four slices. The numbers can be seen in Table 4.1.

Table 4.1: Percentage of passing for the recipe and the two analysis methods.

percentage passing	sieve size		
	2 mm	8 mm	16 mm
recipe	21.0 %	66.7 %	97.5 %
standard method	18.1 %	63.9 %	98.8 %
image analysis	18.7 %	66.6 %	100.0 %

As can be seen in the figure the result from our method follows both the recipe and the analyze well.

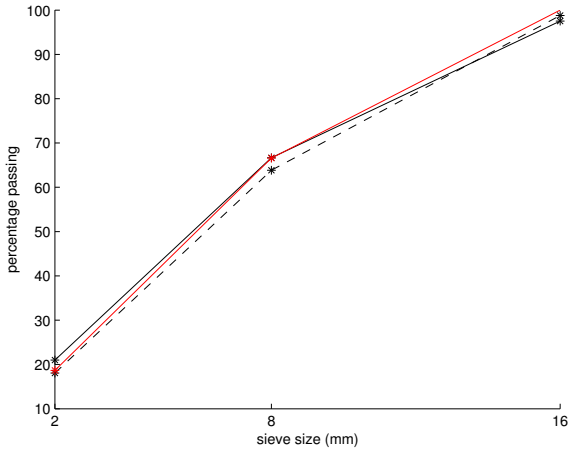


Figure 4.10: Size distribution of the stones in the asphalt sample. The sieve size in millimeters is on the x-axis and the percentage passing at a given sieve size is on the y-axis. The black curve shows the recipe, the dashed black curve shows the result from the standard analysis and the red curve the result from the image analysis.

4.5 Conclusions and Future Work

The method shows promising result with rectangles that fit the stones in the sample very well. Also the calculated size distribution is close to both the recipe of the asphalt and the result from analysis done by the standard way. However the method have limitations in finding the smallest of the stones, in order to do that we need cameras with better resolutions such as microscopes. Also some more experiments have to be done to evaluate the method a bit more.

Part II

Cancer Detection

Chapter 5
Preliminaries on Machine Learning

A very common problem in both image analysis and other areas is classification of data. Classification is the problem of identifying which category a certain observation belongs to. The algorithm that assign the data to different classes is called a classifier.

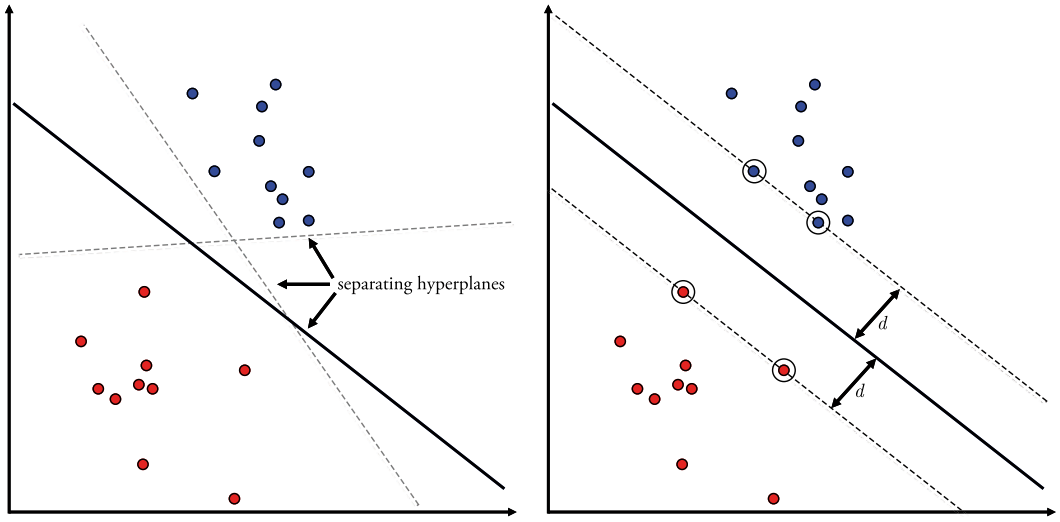
The training of the classifier is an example of supervised learning. In supervised learning the algorithm is trained by use of some training data that consists of data points, or observations, with a label associated. If the training data do not have any labels associated we instead use unsupervised learning. An example of unsupervised learning is clustering, for example by the well known k-means algorithm [61], where the data is grouped into different groups where similar data points belong to the same group. This chapter aims to give some introduction on supervised learning and to present some classifications methods. The first two methods; support vector machines [21] and random forest [14] use feature vectors as input, where the features often are hand-crafted to fit the problem. The third method, deep learning [3], uses images as input and features are extracted from the images. The features are extracted by performing a series of convolutions on the image, where the convolution kernels are trained using the training data.

5.1 Support Vector Machines

Support vector machines [21] is a binary classification algorithm that tries to separate data in an n-dimensional space with an $(n-1)$-dimensional hyperplane. After finding the hyperplane that separates the training data in the best way, a new data point can be classified by looking at what side of the hyperplane the point is.

5.1.1 Optimal Separating Hyperplane

First we assume that the data points are well separated, that means that we can separate the data points with a hyperplane without errors. Figure 5.1a shows an example of well separated data in two dimensions. The red points belong to one class, and the blue points belong to the other class. These two groups can be separated by different lines, three examples of separating lines are shown in the figure. The dashed lines separates the groups but not in an optimal way, line shown in solid black is the optimal way to separate the points, this is called the optimal separating hyperplane.



(a) A few separating hyperplanes. (b) Hyperplane with margins.

Figure 5.1: Example of the optimal separating hyperplane for two-dimensional data. The red points belong to one class and the blue points to another class. (a) shows some possible separating hyperplanes with the optimal separating hyperplane marked with a solid black line. (b) shows the optimal separating hyperplane with margins marked with dashed lines. The support vectors that is the points closest to the plane is marked with circles, the distance from them to the separating hyperplane is denoted by d.

The reason for that the line shown in solid black is the optimal separating line is because the margin to the points are larger than for all other lines. These margins are illustrated in Figure 5.1b and are the distance, denoted by d, between the solid black line and the dashed lines. The points marked with circles are the points closest to the optimal hyperplane, these points are called support vectors.

To find the optimal hyperplane we have to find the hyperplane that maximizes

the distance d. We can write a hyperplane as $\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$, where $\boldsymbol{w}$ is a vector of coefficients, $\boldsymbol{x}$ are points of the hyperplane and b is a constant.

By labeling the training data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ with labels $y_1, \ldots, y_m$, where y_i is either 1 or -1 depending on which class $\boldsymbol{x}_i$ belongs, a hyperplane can be found so that all training points satisfy

$$\begin{cases} \boldsymbol{w} \cdot \boldsymbol{x}_i + b \geq 1 & \text{if } y_i = 1, \\ \boldsymbol{w} \cdot \boldsymbol{x}_i + b \leq -1 & \text{if } y_i = -1, \end{cases} \tag{5.1}$$

where y_i is the label for point $\boldsymbol{x}_i$. These inequalities can also be written as one according to

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 \quad \forall i. \tag{5.2}$$

By definition the support vectors satisfy $y_i(\boldsymbol{w}_0 \cdot \boldsymbol{x}_i + b_0) = 1$, this can be achieved by scaling of the parameters $\boldsymbol{w}$ and b.

The distance, d, between the optimal hyperplane and the supporting vectors is given by

$$d(\boldsymbol{w}, b) = \frac{1}{2}\left(\min_{y=1} \frac{\boldsymbol{x} \cdot \boldsymbol{w}}{|\boldsymbol{w}|} - \max_{y=-1} \frac{\boldsymbol{x} \cdot \boldsymbol{w}}{|\boldsymbol{w}|}\right). \tag{5.3}$$

With the optimal hyperplane, $\boldsymbol{w}_0 \cdot \boldsymbol{x} + b_0 = 0$, this distance can be written as

$$d(\boldsymbol{w}_0, b_0) = \frac{1}{|\boldsymbol{w}_0|} = \frac{1}{\sqrt{\boldsymbol{w}_0 \cdot \boldsymbol{w}_0}}. \tag{5.4}$$

Hence we can find the optimal hyperplane by solving the following optimization problem

$$\begin{cases} \text{minimize} & \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} \\ \text{subject to} & y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 \quad \forall i. \end{cases} \tag{5.5}$$

This is a quadratic minimization problem with linear constraint. Using the standard optimization method with Lagrange multipliers we get the Lagrangian function

$$L_P(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w} \cdot \boldsymbol{w} - \sum_{i=1}^{m} \alpha_i(y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1), \tag{5.6}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$ is a vector of non-negative Lagrange multipliers corresponding to the constraints in (5.2) and m is the number of observations.

At its minima the gradient of L_P is zero, therefore by taking the derivative of L_P with respect to $\boldsymbol{w}$ and b we get the following constraints

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i \boldsymbol{x}_i, \tag{5.7}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0. \tag{5.8}$$

Plugging these conditions into the Lagrangian we get the dual problem

$$L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i} \cdot \boldsymbol{x_j}. \tag{5.9}$$

The maximum of this dual function is the same as the minimum for the primal function in (5.6). Hence, the problem in (5.5) is equivalent to the problem

$$\begin{cases} \text{maximize} & \boldsymbol{\alpha 1} - \frac{1}{2} \boldsymbol{\alpha}^T Y X X^T Y \boldsymbol{\alpha}, \\ \text{subject to} & \alpha_i \geq 0 \quad \forall i, \\ & \sum \alpha_i y_i = 0, \end{cases} \tag{5.10}$$

where $\mathbf{1} = (1, \ldots, 1)^T$ is an m-dimensional unit vector, where m is the number of observations, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$, $Y = \operatorname{diag}(y_1, \ldots, y_m)$ and X is a matrix where each row is an observation.

In the solution, the training points with $\alpha_i > 0$ are the support vectors, all other training points will have $\alpha_i = 0$. Then we can write $\boldsymbol{w}_0$ as a linear combination of support vectors. The constant term b_0 can be computed by choosing any point, $\boldsymbol{x}_i$ with $\alpha_i > 0$ and use the relation $y_i(\boldsymbol{w}_0 \cdot \boldsymbol{x}_i + b_0) = 1$. See [21] and [18] for a more detailed derivation of the previous results.

A new observation or data point can be classified by looking at which side of the hyperplane is located. A new observation $\boldsymbol{x}_{new}$ is labeled according to

$$y_{new} = \operatorname{sign}(\boldsymbol{w}_0 \cdot \boldsymbol{x}_{new} + b_0). \tag{5.11}$$

5.1.2 Soft Margins

Usually the data points are not well separated and there is a certain overlap between the groups. In this case we cannot maximize the margins in the same way. To overcome this, we allow for some errors. This approach with soft margins was first described in [21]. The errors are illustrated in Figure 5.2, the optimal hyperplane is marked with a solid black line and the margins with dashed lines. The points in between the dashed lines are errors and the distance between point $\boldsymbol{x}_i$ and the relevant dashed line is denoted by ξ_i.

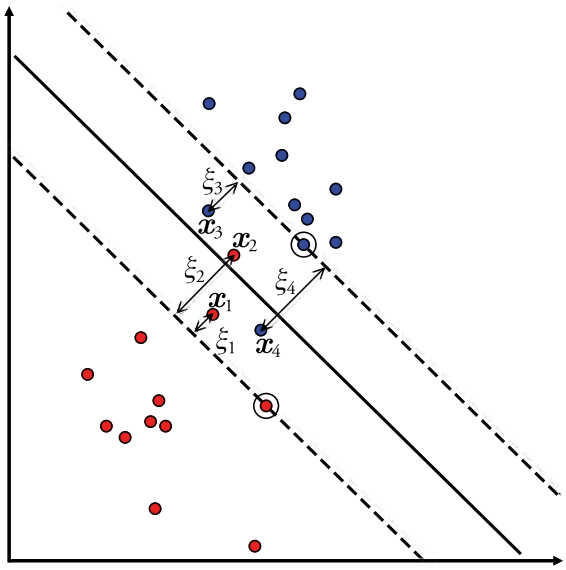


Figure 5.2: A dataset that cannot be separated by a hyperplane. The points $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_3$ and $\boldsymbol{x}_4$ are errors and the magnitude of the error is denoted by ξ_i for point $\boldsymbol{x}_i$.

We would like to find a good tradeoff between the margin size and the errors. With this modification the minimization problem becomes

$$\begin{cases} \text{minimize} & \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + \lambda\left(\sum_{i=1}^{n}\xi_i\right)^k, \\ \text{subject to} & y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \geq 1 - \xi_i \quad \forall i, \\ & \xi_i \geq 0 \quad \forall i, \end{cases} \tag{5.12}$$

where λ is a term that controls how hard to punish errors, with a small λ the minimization problem gets closer to the original optimal separating hyperplane minimization problem.

Again we construct the Lagrangian. We see that the term $\lambda \left(\sum_{i=1}^{n} \xi_i\right)^k$ disappears in the Lagrangian and the dual problem becomes

$$\begin{cases} \text{maximize} & \boldsymbol{\alpha}\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T Y X X^T Y \boldsymbol{\alpha}, \\ \text{subject to} & 0 \leq \alpha_i \leq \lambda \quad \forall i, \\ & \sum \alpha_i y_i = 0. \end{cases} \tag{5.13}$$

5.1.3 Kernels

The svm-classifier can be extended to handle data that could not be linearly separated. This is done by transforming the data by some non-linear transformation. One way to transform the data is by so called basis expansion. In this case we extend the input space by adding some new basis vectors.

One example of a quadratic basis expansion, $h(X)$, where X is one-dimensional data is

$$h(X) : \begin{bmatrix} x \end{bmatrix} \mapsto \begin{bmatrix} x & x^2 \end{bmatrix}. \tag{5.14}$$

Figure 5.3 shows some data in both the input space and the feature space. In input space, shown in Figure 5.3a the two classes are not linearly separable, but in the feature space, shown in Figure 5.3b, they can be separated by the black line.

Substituting X by $h(X)$ in (5.13) gives

$$\text{maximize} \quad \boldsymbol{\alpha}\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T Y K Y \boldsymbol{\alpha}, \tag{5.15}$$

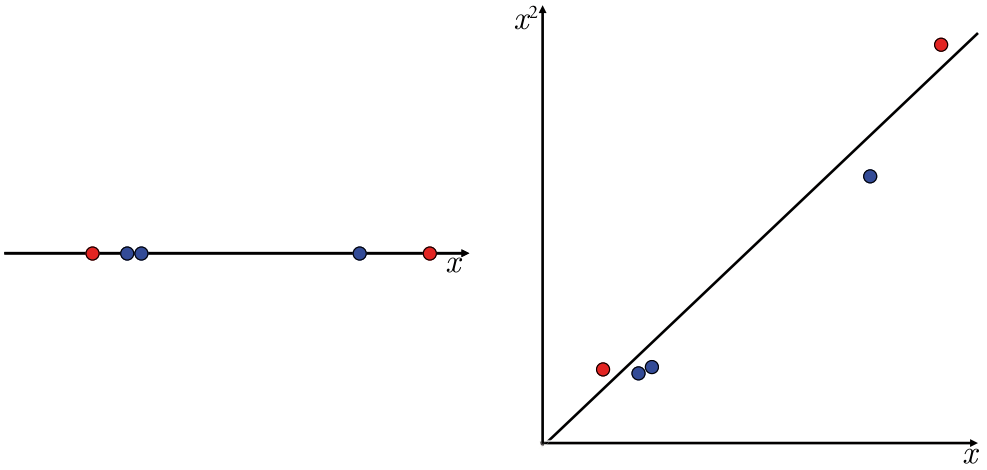
where $K = h(X)h(X)^T$ and is called a kernel. The kernel is an $m \times m$ matrix, where m is the number of observations, and does not depend on the number of basis vectors that are used.

If the data is transformed with an appropriate kernel, the data can be linearly separated in the feature space. This makes the svm-classifier very flexible and useful for many applications. The approach with kernels was suggested in [9].

Two important kernels are the polynomial and gaussian kernels. The polynomial kernel is defined by

$$K_{i,j} = (1 + x_i x_j^T)^d, \tag{5.16}$$

where $K_{i,j}$ is the element on row i and column j in the kernel matrix, x_i and x_j are the rows i and j in X. The variable d determines the degree of the pylonomial.



(a) In the input space, the two classes cannot be separated by a line.

(b) In the feature space, the two classes can be separated by the black line.

Figure 5.3: A two class problem shown in the input space in (a) and the feature space in (b).

In a similar way the Gaussian kernel is defined by

$$K_{i,j} = e^{-\|x_i - x_j\|^2/c}, \tag{5.17}$$

where c is a parameter that controls the width of the kernel.

Figure 5.4 shows the resulting hyperplane using a quadratic kernel. The plot shows the input space, where the points are the original points and the separating hyperplane has been transformed according to the kernel.

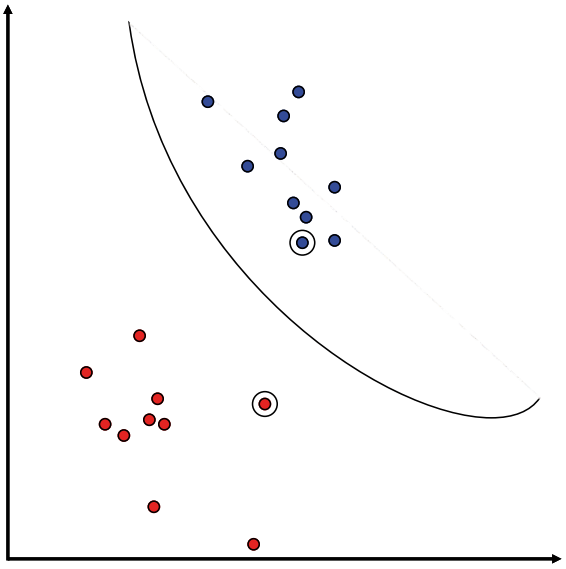


Figure 5.4: The optimal separating hyperplane in input space.

5.1.4 Multiclass SVM

There are several strategies to extend the binary svm-classifier to support multiple classes. One commonly used method is to use a one-versus-all strategy. In this case several classifiers are trained. The first one separates the first class from all the others, the second classifier separates the second class from the others and so on. By comparing the distances to the separating planes in all classifiers a new data point can be classified.

Figure 5.5 shows an example of a three class problem. The red, blue and green points represent training points of different classes and the gray point is a new data point that we would like to classify either as red, blue or green. The lines are the separating hyperplanes, the red line separates the red class from the blue and green classes. In the same way the blue line separates the blue class from the red and green classes and the green line separates the green class from the red and blue classes. A new data point, the gray one, is then classified using each of the three classifiers. We assume that the data is labeled in a way that the point will have a positive distance to the red line if it is classified as red in the classifier red versus blue and green and a negative distance if it is classified as not red. We can see in the figure that the point will have a positive distance to both the red and the green line but a negative distance to the blue line. Since the gray point is closer to the green class the distance to the green line will be larger than to the red line.

Hence we can classify a new point by looking at the distance to the separating hyperplanes for all classifiers, and choose the class that gives the largest distance.

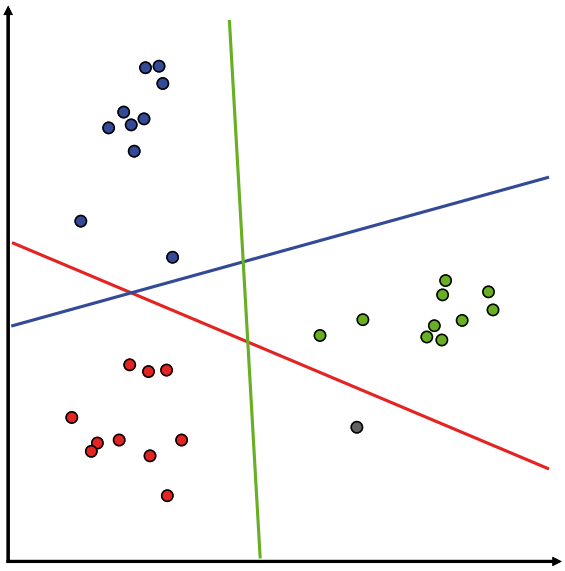


Figure 5.5: A three class problem. The red line separates the red class from the blue and green classes, the blue line separates the blue class from the red and green and the green line separates the green class from the red and blue. The gray point is a new data point that should be classified. It is located with positive distance to the red and green line but the distance to the green line is largeer than to the red line, therefore the point is classified to belong to the green class.

5.2 Random Forest

Decision trees has been used for classifications for a very long time in various applications. They are a powerful tool and easy to visualize which has made them very popular. A drawback of these trees is that they have a tendency to overfit to the data and the variance between trees is fairly big. If the tree is overfitted, it performs well on the training data but poor on the test data. For this reason many papers [19, 81, 36, 13, 1] suggest to use an ensemble of decision trees to reduce variance and to increase accuracy. This has resulted in a so called random forest. The random forest algorithm was first proposed by Leo Breiman 2001 in [14]. To reduce variance, bootstrap aggregation [13] together with variable reduction by the subspace method [36] are used to create many uncorrelated trees.

5.2.1 Decision Trees

There are two types of trees; classification trees and regression trees. Classification trees takes an input vector and classifies that into some pre-defined distinct class. Regression trees outputs a value that comes from a continuous distribution. For the rest of the chapter the classification trees are studied.

A tree consist of nodes of different types and branches between them. There is a root node, internal nodes and leaf nodes. In a classification tree the leaf nodes, also called terminal nodes, represent the class that the input vector was classified as. Internal nodes has one incoming branch and two outgoing branches where a branch leads either to a new internal node or to a leaf node. Which one of the branches to follow depends on the data and on the criterion in the node. The root node is the starting node of the tree and does not have any incoming branch. A node is a parent node to the nodes that the outgoing branches leads to and a child to the node that the incoming branch comes from.

Figure 5.6 shows a small tree for classifying an animal into one of the four species; ant, mouse, giraffe or elephant. In this tree each of the variables are categorical variables that can only take two values

Instead of using categorical variables we can also use continuous variables. Using the same example as above an animal is described by a feature vector, $\boldsymbol{x} = (x_1, x_2, x_3)$, where for example x_1 is the weight of the animal in kilograms, x_2 the number of legs and x_3 the length of the neck in meters. A new version of the tree in Figure 5.6 is shown in Figure 5.7, in this case at each non-terminal node one of the variables is compared with a threshold. Depending on the value of the

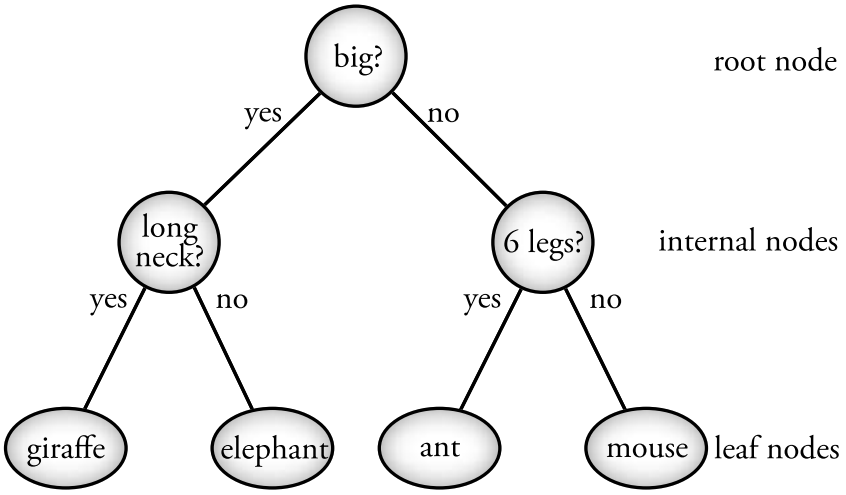


Figure 5.6: A tree to classify an animal into four different classes. In each none terminal node a question is asked and depending on the answer the left or right branches is followed.

variable and the threshold you go either to the left or the right child.

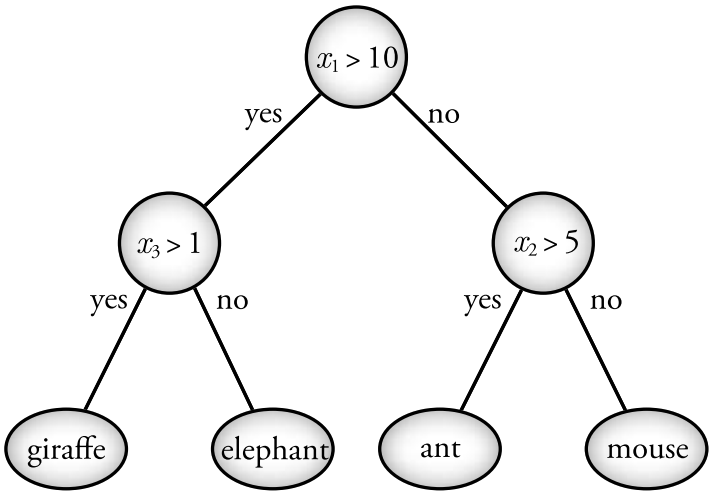


Figure 5.7: A tree to classify an animal into four different classes. In each none terminal node the value of one of the variable is checked, depending on if the value of the variable is lower or higher than some threshold the left or right branch is followed.

Growing the Tree

The classification tree is built using a training set, that consist of input vectors and a label that tells what the correct class of the vector is. There exist numerous ways to build a tree, where the main difference is how to split the tree, that is how to decide the criteria to go to the left or right in a parent node. Figure 5.7 shows an example of a univariate splitting criteria, that means that only one variable is used in a node in the splitting criteria. These are the most used types of splitting criteria and there exists a lot of different methods. There are also multivariate splitting criteria that uses several variables to decide on the split in each node. These methods are more complicated and even if they could improve the performance of the tree a lot, they are not as popular as the univariate splitting criteria. During the building of the tree, the training data is split into the different nodes until all nodes are leaf nodes. In the case where a leaf node consists of data of several classes the majority vote is used as the leaf class.

One of the most popular univariate splitting criteria is based on the Gini index of diversity. The Gini index of diversity [15, 29] is a measure of node impurity. If a node only consist of one class it has a Gini index of 0. The maximum Gini index occurs when there are equally many of each class in a node. To calculate the Gini index we first need to know the conditioned probability, $p(j|t)$, that an observation in node t is of class j. This probability is calculated by

$$p(j|t) = \frac{N_j(t)}{N(t)}, \tag{5.18}$$

where $N_j(t)$ is the number of observation of class j in node t and $N(t)$ is the total number of observations in node t. Since all observations belongs to some class it must hold that

$$\sum_j p(j|t) = 1. \tag{5.19}$$

The Gini index is then defined by

$$G(t) = \sum_{i \neq j} p(i|t)p(j|t), \tag{5.20}$$

where $G(t)$ is the Gini index for node t and i and j are two different classes.

Using that

$$\left(\sum_j p(j|t)\right)^2 = \sum_{i \neq j} p(i|t)p(j|t) + \sum_j p^2(j|t)$$

and the relation in (5.19) we can rewrite (5.20) as

$$G(t) = \left(\sum_j p(j|t)\right)^2 - \sum_j p^2(j|t) = \sum_j p(j|t)\left(1 - p(j|t)\right)$$
$$= 1 - \sum_j p^2(j|t). \quad (5.21)$$

Now, consider a parent node, t, with two child nodes, c_l and c_r, where c_l is the left child and c_r the right child. The goodness of a possible split in the parent node is evaluated by measuring the weighted node impurity for the two children. The best split will be the split that minimizes

$$S = N(c_l)G(c_l) + N(c_r)G(c_r), \quad (5.22)$$

where $N(c_l)$ is the number of observations in the left child and $G(c_l)$ the Gini index for the left child. In the same way $N(c_r)$ and $G(c_r)$ are the number of observations and the Gini index for the right child.

To find the best split in a node we need to search through all variables, for each variable we also need to find a good threshold. If we assume that we have N variables $x_1, \ldots x_N$ and M observations, then for each variable we have at most $M - 1$ possible ways to split the data. We therefore have at most $M - 1$ possible thresholds in between observations to try for each variable. Then we choose the split that gives to lowest S. See [72] for many more splitting criteria.

We also need to decide when not to split the data further. For this we need some kind of stopping criteria. There is no point splitting the node further if all observations in the node belong to the same class. We can also decide on a maximum depth of the tree, if this is reached the node becomes a terminal node. Other stopping criteria could be that there are too few observations in a node or that the best possible split is not better than some pre-defined threshold.

Pruning the Tree

It is quite difficult to know how big we should grow the tree. If we choose a weak stopping criteria resulting in a large tree it is a high risk that the tree will become overfitted to the data. In the case of overfitting, the tree is able to classify the training data, but will perform very poor on test data. If we apply a too strong stopping criteria there is a high risk of missing out on good splits. One way to come around this is to let the tree grow large and then cutting the branches that do not contribute much to the performance of the tree. This is called pruning the tree and is very important when building a single classification tree for classifying data. There exists several pruning methods, see [72] for an overview.

5.2.2 Random Forest

There are a few problems using decision trees for classification. Say that we build a few different trees using slightly different data. If we build deep trees there will be large variations among the trees, however the bias for deep trees is still low, meaning that if we average the result we are likely to correctly classify the data. If we build less deep trees the variance will be lower but the bias increases, [24].

Bagging

To reduce variance it was suggested by Leo Breiman in [13] to build an ensemble of decision trees and to use bootstrap aggregation, bagging, to slightly change the data for each tree. Giving the trees slightly different data helps to make the trees uncorrelated and this reduces the variance.

Bagging works in the following way: Having N observations, or data points, we draw N observations with replacement from the observations. At each draw some of the data will be left out and some will appear more than once. From each of this bootstrap samples we build a tree to its maximum size without pruning so that the bias will still be low.

Variable Reduction by the Subspace Method

Another way to build an ensemble of uncorrelated trees way suggested Tin Kam Ho in [36]. Instead of using all available features when building the tree a randomized subset of features were selected. By using different subspaces for each tree it was shown in [37] that the similarities between trees were lower using the

subspace method than for example using the bagging method, at least when the feature space was large. In this way the variance can be reduced even more.

Random Forest

Random forest uses a combination of bagging and the subspace method and was introduced by Leo Breiman in [14]. First, a bootstrap sample from the observation is drawn, then a subset of features is randomly chosen to build a decision tree. The number of feature to choose is usually much smaller than the total number of features. A common choice is to use $p = \sqrt{M}$, rounded downwards where p is the number of features to use and M is the maximal number of features.

Random Forest
Assume that we have N observations with M features each. An observation then consist of an M- dimensional vector.

1. Make a bootstrap sample, S, from the observations: draw N observations with replacement uniformly distributed.
2. Select p features from the M possible features, here $p << M$, and use the training data in S to build a decision tree to maximum size without pruning.
3. Repeat 1-2 until a forest of desired many trees is grown.

5.3 Deep Learning

The idea behind deep learning is to mimic the neural structure of the brain. A brain consists of a lot of neurons that are connected to each other with synapses in a giant network, a human brain has approximately 100 billions of neurons with up to 10000 synapses each. The first step towards deep learning is the development of artificial neural networks. It was first presented by Warren McCulloch and Walter Pitts in [63] in 1943 and have since then been developed further [50].

In 1957 Frank Rosenblatt introduced the percpetrone algorithm [73], that is a small single layer network. The algorithm was implemented in "Mark I perceptron", the first successful neurocomputer. It was able to recognize simple numerics with an image sensor of 20×20 pixels.

The most popular training rule for the networks is called backpropagation, which basically optimizes the weights using a gradient descent approach. It has been developed by several independent researchers but it was first presented by Paul Werbos in 1974, [86]. Even though artificial neural networks was promising in the beginning other machine learning techniques became more popular [67].

In [54], the standard backpropagation rule was used to train a convolutional neural network with the purpose to recognize numbers in zip codes. The input to this network was grayscale images with no hand-crafted features. It was shown in [55] that the use of convolutional neural networks trained with the backpropagation rule outperformed all other machine learning techniques for classifying the numbers.

Since then, several improvements of the design have been made to speed up the training time and to achieve spatial invariance. This has made the use of deep convolutional networks, convolutional networks with many layers, very popular in computer vision for object recognition and localization. The use of deep convolutional networks has been known as deep learning.

5.3.1 Artificial Neural Networks

The first step towards deep learning was the invention of artificial neural networks. The artificial neural networks have a similar structure as the biological networks but are usually much smaller. The networks consist of different layers that are connected, there is always one input layer and one output layer, but the number of layers in between, called hidden layers, varies. A small neural network can be seen in Figure 5.8. The figure shows a layered network, there are also other

types of networks but they are not relevant for the rest of the chapter. The neurons are marked with circles and connected neurons are marked with a straight line between them. Each connection also has a weight associated. The weight between neuron i and neuron j is denoted by $w_{i,j}$.

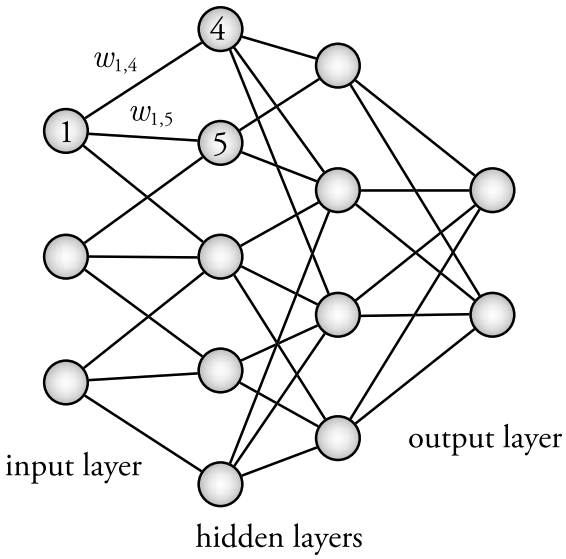


Figure 5.8: A small neural network with two hidden layers. The neurons are marked with circles and the connections are marked with straight lines.

Neurons

A layer in the network consist of different neurons. Their task is to retrieve information from the connected neurons in the previous layer, process it and send it further to the connected neurons in the next layer. The processing of the data is expressed by three functions; a propagation function, an activating function and an output function. The propagation function is the function that process the incoming information, then the activation function is called to decide the state of the neuron; active or not active. At last the output function sends the information to the next layer.

A common choice of propagation function, f_j, is a weighted sum

$$f_j = \sum_{i \in I} w_{i,j} o_i, \tag{5.23}$$

where I is the set of neurons that are connected to j, o_i is the output from neuron i and $w_{i,j}$ is the weight on the connection between neuron i and j.

The activation function then decides on the state of the neuron, the simplest function used for this is the threshold function

$$a_j(f_j) = \begin{cases} 1 & \text{if } f_j \geq \theta, \\ 0 & \text{if } f_j < \theta, \end{cases} \tag{5.24}$$

where a_j is the activating state, f_j the output from the propagation function and θ the threshold. The threshold function is simple but not differentiable so often differentiable approximations, for example the Fermi function, are used instead. The Fermi function, also called sigmoid function, expanded by a temperature parameter, is defined as

$$f(x) = \frac{1}{1 + e^{-x/T}}, \tag{5.25}$$

where T is the temperature parameter that controls how much the function is compressed on the x-axis. The smaller T, the closer the Fermi function is to the threshold function. An other choice of the activation function could be the hyperbolic tangent.

At last the output function, which is usually the identity function, forwards the information to the next layer.

Training

One important feature of the neural networks is its ability to learn, like the human brain learns by itself. This training could be supervised or unsupervised. In both cases the network is provided with some training input data. For the supervised learning it is also provided with some ground truth that is the desired output of the network. In unsupervised learning, given only the input patterns, the network tries to identify similarities and classifies the input into classes, where the data in the same class are similar. It is often more efficient to provide the network with ground truth so that the output can be compared directly to the desired output and therefore supervised learning is very practical to use.

While initializing the network the number of layers in the network and the number of neurons in each layer have to be decided in advance. The choices of processing functions in the neurons also have to be pre-defined. At first all the weights in the network are chosen by random, during the training the weights are updated in an iterative manner until the difference between the desired output and the actual output gets sufficient small.

The training is done by optimizing some function that depends on the weights. It is often useful to minimize the error, that is the difference between the desired output and the output from the network. We can define the error function as

$$Err(w) = \frac{1}{2} \sum_{p \in P} \left(\sum_{\Omega \in O} (t_{p,\Omega} - y_{p,\Omega})^2 \right), \tag{5.26}$$

where P is the set of training samples, O the set of output neurons, $t_{p,\Omega}$ the desired output of neuron Ω from training sample p. The desired output of neuron Ω is 1 if the input pattern belong to the class that the neuron represent and 0 otherwise. The variable $y_{p,\Omega}$ is the actual output of the network and depends on both the input to the network and the weights.

The most popular training rule method is called backpropagation and was presented by Paul Werbos in [86]. In backpropagation the derivatives of the error function with respect to the weight is computed in every step. The method is based on the well known gradient descent method, but with an efficient way to compute the derivatives. A derivation of the backpropagation rule is given in [87] and [50].

If we use the error function defined in (5.26) then, in each step, the weights are updated according to

$$\Delta w_{k,h} = \eta o_k \delta_h, \tag{5.27}$$

where $\Delta w_{k,h}$ is the change of the weight between neuron k and h, η the learning rate, usually a small number, and o_k the output from neuron k. The term δ_h is computed differently if the neuron is in a hidden layer, called inner neuron, or if it is a neuron in the output layer, an output neuron. It is computed by

$$\delta_h = \begin{cases} f'_{act}(net_h) \cdot (t_h - y_h) & \text{if } h \text{ is an outer neuron,} \\ f'_{act}(net_h) \cdot \sum_{l \in L} \delta_l w_{h,l} & \text{if } h \text{ is an inner neuron,} \end{cases} \tag{5.28}$$

where f_{act} is activation function for the neuron, net_h is input to neuron h, t_h the desired output from outer neuron h, y_h actual output from outer neuron h and L the set of subsequent neurons in later layers. More about artificial neural networks can be found in [50].

5.3.2 Convolutional Neural Networks

A special type of neural networks that has been used a lot in later years in image analysis are the convolutional neural networks. The input to these network is often images and features are calculated by performing convolution with different kernels.

The convolutional neural networks often consist of a number of stages. The first stages consist of layers that perform convolution, subsampling or pooling and non-linear transformations. The non-linear transformations correspond to the activation functions in the neurons. The last stages consist of fully connected layers, meaning that each neuron in the layer is connected to all neurons in the previous layer, and finally classification layers. The structure of a neural network is shown in Figure 5.9. The network in this example has an image as input and three output neurons, so there are three possible output classes.

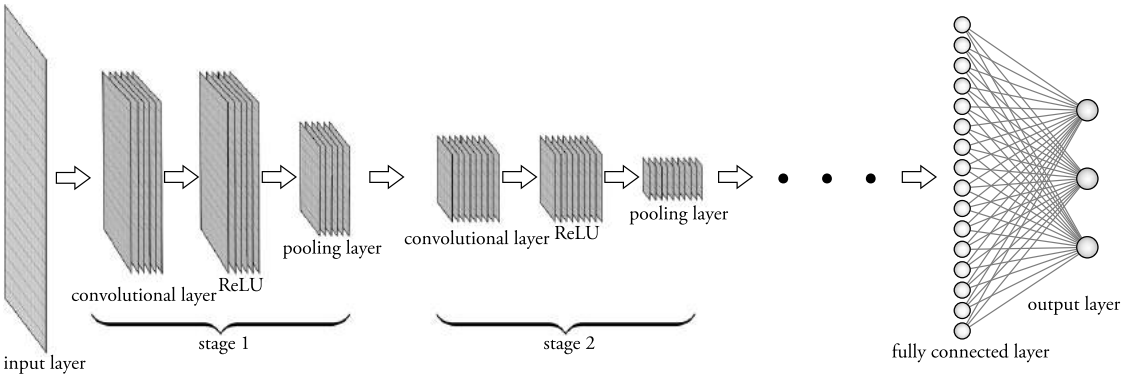


Figure 5.9: A deep convolutional network that classifies an image to one of three classes. The first stages in the network consist of a convolutional layer followed by a non-linear transformation and a pooling layer. The last stages are classification layers. The meaning of the different layers will be described in the following sections.

Convolutional Layers

The convolution of a certain layer in the network is performed with a number of convolution kernels, where all kernels are three-dimensional. The width and height of the kernels can be chosen but the depth of the kernel has to be equal to the number of channels in the current layer. A layer consist of an image, or a stack of images. Assuming that an image stack, I, is of size $m_1 \times n_1 \times d_1$, then all

convolution kernels are of size $m_2 \times n_2 \times d_1$, where m_2 and n_2 are odd numbers. Having d_2 different kernels we can combine these to a four-dimensional kernel, w, of size $m_2 \times n_2 \times d_1 \times d_2$. The result of the convolution will be a new image. The convolution for a layer in the network can be defined as

$$\begin{aligned}(I \circledast w)(i_2, j_2, k_2) &= f(i_2, j_2, k_2) \\ &= \sum_{i_1=-a}^{a} \sum_{j_1=-b}^{b} \sum_{k_1=1}^{d_1} I(i_2 - i_1, j_2 - j_1, k_1) w(i_1, j_1, k_1, k_2),\end{aligned}$$

where $(I \circledast w)(i_2, j_2, k_2)$ is the result of the convolution at position (i_2, j_2, k_2) in the new image. In the formula we set $a = (m_2 - 1)/2$ so that it correspond to half of the height of the kernel, rounded downwards. In the same way $b = (n_2 - 1)/2$ is about half of the width.

There are different ways to handle the convolution at the edges of the image, one way is to only use the valid pixels, then we write the convolution as

$$(I \circledast w)(i_2 - a, j_2 - b, k_2) = f(i_2, j_2, k_2), \tag{5.29}$$

where $1 + a \leq i_2 \leq m_1 - a$ and $1 + b \leq j_2 \leq n_1 - b$. In this case the resulting image will be smaller than the input image.

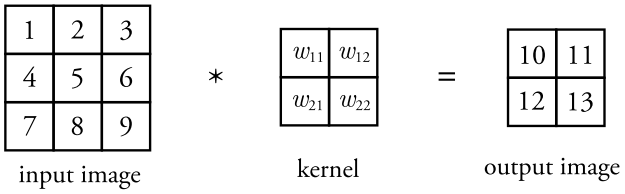
Another way is to add zeros outside the image, this is called zero-padding. The convolution in this case can be written as

$$(I \circledast w)(i_2 + a, j_2 + b, k_2) = f(i_2, j_2, k_2), \tag{5.30}$$

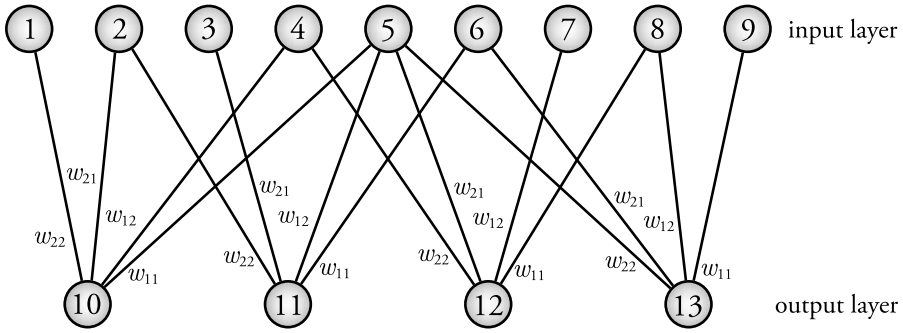
where $1 - a \leq i_2 \leq m_1 + a$ and $1 - b \leq j_2 \leq n_1 + b$. In this case the resulting image will be larger than the input image.

The convolution can be done using all images in the stack or on a smaller subset of images. The size of the convolution kernels and how many images that should be used in the convolution has to be chosen in advance but the weight are set during the training of the network.

In a neural network the convolution corresponds to a weighted sum as a propagation function were neurons correspond to pixels. The weights on the connections correspond to the values in the convolution kernel. Figure 5.10 shows the convolution of a small image with a single channel without zero-padding expressed as a neural network, the convolution in the image representation can be seen in Figure 5.10a where the numbers in the input and output images represent the numbers of the corresponding neuron in Figure 5.10b. Each neuron also has a pixel value but those are not shown in the image.



(a) Convolution of an image with a convolution kernel. The numbers in the input and output images denotes the number of the corresponding neuron in the network and not the pixel values.



(b) Corresponding network with the same numbering as in (a).

Figure 5.10: The convolution of a 3×3 image with a 2×2 convolution kernel shown as images and as two layers in a neural network.

Max-Pooling

To achieve spatial invariance it has been suggested to add subsampling layers after the convolutional layers. The subsampling layer retrieve data from neighboring neurons and combine it to one output signal. In [71] it was suggested to use max-pooling layers instead of the subsampling layers. The max-pooling layers works by taking the maximum of all neurons inside some window, the windows can be overlapping or non-overlapping. A comparison between subsampling layers and max-pooling layers was made in [77]. The authors showed that the max-pooling layers outperformed subsampling layers and that it was no gain with overlapping pooling widows.

Activation Function

The most popular choice of activation function in deep neural networks is the function $f(x) = \max(0, x)$, where x is the input to the neuron. It has been shown in [51, 68] that using this as an activation function instead of the Fermi function or the hyperbolic tangent makes the training of the network much faster. A neuron with this rectifier function is called a Rectified Linear Unit, ReLU.

Dropout

A risk when training these networks is overfitting, this means that the model fits the training data really well, but when testing on validation data that was not used to build the model, the model does not perform well. One way to reduce the risk of overfitting is to randomly omit parts of the training data or neurons in each training step. This technique is known as dropout [35, 22].

Classification

The last layer in the network is a classification layer. The classification is done by assigning the output neurons a number that represent the probability that the image used as input to the network belongs to the class that the output neuron represent.

The most popular way to compute the probabilities is with the softmax function [16], which is defined as

$$p_i = \frac{e^{q_i}}{\sum_{j=1}^{c} e^{q_j}}, \tag{5.31}$$

where p_i is the probability that the input belongs to class i, q_i is the input to output neuron i and c is the number of output neurons or classes.

Chapter 6

Gleason Grading

This chapter describes an automatic algorithm with the purpose to assist pathologists to report Gleason score on malignant prostatic adenocarcinoma specimen. A specific aim is to support intuitive interaction with the result, to let pathologists adjust and correct the output. Therefore, we have designed an algorithm that makes a spatial classification of the whole slide into the same growth patterns as pathologists do. The described algorithm is a part of a larger project envisioning a semi-automatic human-computer system that pathologists could use to increase the efficiency and accuracy when reporting the Gleason grade, which implies that both the resolution and the speed of the algorithms could be equally important as the accuracy. In this chapter, we evaluate the overall accuracy on a set of images as well as the accuracy when the images are divided into small patches, while keeping in mind that we want to keep the processing time as low as possible.

The Gleason grading system is a widely used classification system of malignant prostate adenocarcinomas based on the growth patterns of the cancer cell population [23]. In the current revision from the 2005 consensus meeting [26], visually detectable malignant growth patterns are organized into three main groups (3,4,5), which are summarized into overall scores based on patient outcome. The meeting also recommended that benign patterns in group 1 and 2 should not be reported separately. In the newly proposed revision [27], the same growth patterns should be detected but are now organized into new overall scores. This makes it possible to develop the same image analysis system for both revisions, by just changing the way that the scores are summarized and grouped.

Automatic image analysis methods for Gleason grading have already been proposed. Doyle et al. [25] described an approach using hand-crafted features based mainly on detecting individual nuclei, whereas both Gorelick et al. [30] and Jacobs et al. [41] used features derived from super-pixels. Another method was proposed in [58] where histograms of SIFT-features were used to classify the

images.

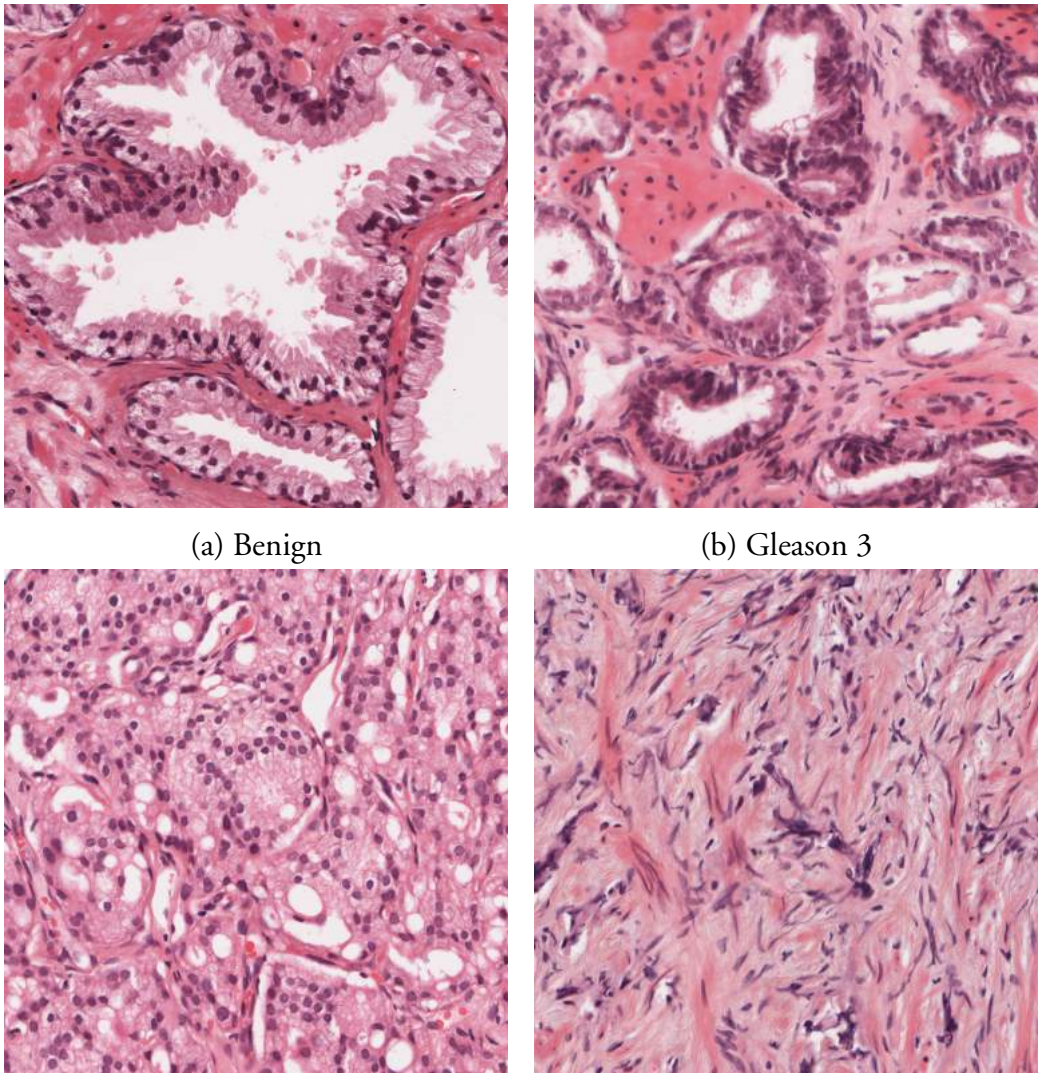
With the advent of deep learning techniques [3], it might be possible to reach high classification accuracy without using hand-crafted features, since the features can be derived during the training. One important technique is convolutional neural networks (CNN), [39, 55]. These networks consist of multiple layers with different functions. The first layer performs convolution on the different color channels of the image, following layers consist of interleaved subsampling and convolutional layers. A subsampling layer reduces the spatial resolution and a convolutional layer combines information using different kernels. With each new layer, a network of feature vectors that represent the images is produced. The length and number of these vectors can either increase or decrease with each layer depending on the design of the network.

In this chapter we used a pre-trained CNN, trained on a dataset consisting of photographic images, and applied it on pathology images, similar to the idea presented in [6]. For this purpose the classification step in the network was removed and replaced with other machine learning techniques to classify the feature vectors extracted from the network.

6.1 Material

During the development phase, self-annotated images generated by the TCGA Research Network[1], were used. The algorithm was then evaluated using cross-validation on an independent set of images that was used by Lippolis [58]. The images came from Beaumont Hospital in Dublin, Ireland and PathXL Ltd in Belfast, UK and consisted of homogeneous single class images classified by one or more pathologists. In total we had 52 images of benign glands, 52 images of Gleason grade 3, 52 images of Gleason grade 4 and 57 images of Gleason grade 5. The images were scanned in 40x magnification, which were downsampled to both 10x and 5x magnification to reduce the processing time and investigate how the scale of the images affects the classification result. Some example images can be seen in Figure 6.1. Figure 6.1a shows benign glands, Figure 6.1b Gleason score 3, Figure 6.1c Gleason 4 and Figure 6.1c Gleason 5.

[1] http://cancergenome.nih.gov/



(a) Benign (b) Gleason 3

(c) Gleason 4 (d) Gleason 5

Figure 6.1: Some example images from the dataset.

6.2 Methods

The analysis method can be divided into three parts: Feature extraction, patch classification, and whole image classification.

6.2.1 Feature Extraction

To extract the initial features, an pre-trained convolutional neural network, OverFeat, was used [78]. Overfeat is a 22-layer network in several stages. The first

stages include convolutional layers followed by a piecewise linear function, defined as $f(x) = \max(0, x)$, and, in some of the stages, max-pooling layers. Later stages include fully connected layers and classification layer but these are not interesting for our purpose. A summary of the first 5 stages in the fast version of the network is shown in Table 6.1. The table shows the size of the windows used for convolution and max-pooling. In the first two stages the convolution is performed only on valid pixels, later stages use zero-padding. The last column shows the length of the feature vectors that the different stages outputs.

Table 6.1: Summary of the first 5 stages in the OverFeat network, with the window sizes used for convolution and max-pooling. The last column shows the layer depth after each stage.

	convoultion	maxpool	depth
stage 1	11×11 window, 4×4 stride	2×2 window, 2×2 stride	96
stage 2	5×5 window, 1×1 stride	2×2 window, 2×2 stride	256
stage 3	3×3 window (full), 1×1 stride	none	512
stage 4	3×3 window (full), 1×1 stride	none	1024
stage 5	3×3 window (full), 1×1 stride	2×2 window, 2×2 stride	1024

We have extracted features from both layer 9 and layer 16 corresponding to the output from stage 3 and stage 5 respectively. In layer 9, a window of 87×87 pixels was used to compute the feature vector. A new feature vector was computed for the square, 16 pixels to the right of the first one, a schematic image of this is shown in Figure 6.2a. This way, we got several feature vectors representing the image and each feature vector was represented by 512 features. The number of feature vectors depends on the size of the input image. In layer 16, these windows were larger, 167×167 pixels, with 32 pixels between the squares, see Figure 6.2b, and the feature vectors consist of 1024 features each. The window size of layer 9 is approximately half of the size of the window in layer 16. This enables the comparison of the effect depth while maintaining the same spatial resolution if we feed the layer 9 version with a 5x image and the layer 16 version with a 10x

image, see Figures 6.2a and 6.2c.

(a) Layer 9 in 5x (b) Layer 16 in 5x (c) Layer 16 in 10x

Figure 6.2: Window sizes, image magnifications, and step sizes used in our experiments. Note that feature vectors in (a) and (c) covers the same spatial location, and that (b) and (c) use the same window size.

6.2.2 Patch classification

Two different classifiers, Random Forest [14] and Support Vector Machines [21] were used to classify the feature vectors obtained by OverFeat.

In Random Forest an ensemble of decision trees is trained from the training data. Each tree uses a number of randomly chosen features to build the decision tree. The classification result depends on the number of trees in the forest and the number of features used.

In Support Vector Machines, SVM, a separating hyperplane that separates the classes in the training set is found. A drawback of the SVM is that it is a binary classifier, so it has to be modified to support multiclass classification. Here we have used the one-versus-all strategy [38]. This is done by training four different classifiers and combining them. First, we build a classifier for benign tissue versus Gleason grade 3, 4 and 5, then another one for Gleason grade 3 versus benign tissue and Gleason grade 4 and 5 and so on. To classify a new vector we look at the distances between the separating hyperplane and the feature vector for all models. In the classifier benign versus all other classes the vectors that were classified as benign will have positive distances and the vectors that were classified as one of the other classes will have negative distances. To classify a vector we compute the distance to the separating plane for all models and choose the class that has the largest positive distance to the plane. Different kernel functions can be applied to

the data to make the SVM classifier non-linear.

6.2.3 Classification of whole images

The patch classification above classifies all individual vectors in the images, where the vectors corresponds to patches of size 87×87 or 167×167 pixels. To classify the whole image we let all patches in an image vote for the different classes and the class with the highest number of votes is chosen.

6.3 Experiments

The proposed method was evaluated for different resolutions of the images, 5x and 10x magnification. For 10x magnification, only features obtained from layer 16 were evaluated. For 5x magnification, features from both layer 9 and layer 16 were investigated. The area used to compute a feature vector in layer nine at 5x magnification roughly corresponds to the area used to compute a feature vector in layer 16 at 10x magnification.

We used 10-fold cross validation to evaluate the random forest and the support vector machines. The number of trees and the number of variables used was optimized during the cross validation of random forest. Different kernel functions were applied to the SVM classifier and the one with the lowest test error was chosen. The kernel function that performed the best differs between different experiments, but it is always either a first or second order polynomial. Figure 6.3 shows the mean training and mean test errors from the cross validation for different magnifications and layers. The plot shows the errors for the random forest classifiers and the error for the best kernel functions of the support vector machines classifier. The training error is shown in blue and the test error in red.

Confusion matrices for the best classifier for the different magnifications and layers are shown in Table 6.2 - 6.4. Table 6.2 shows the result for 10x magnification and layer 16, Table 6.3 the result for 5x magnification and layer 9 and Table 6.4 the result for 5x magnification and layer 16.

For the binary classification of benign tissue versus cancerous tissue we discovered a true positive rate of 94.5 % and a false negative rate of 5.5 % using Table 6.3.

Figure 6.4 shows the training and test errors for different magnifications and layers when whole images were classified. The plot shows the errors for the random forest classifiers and the error for the best kernel functions of the support

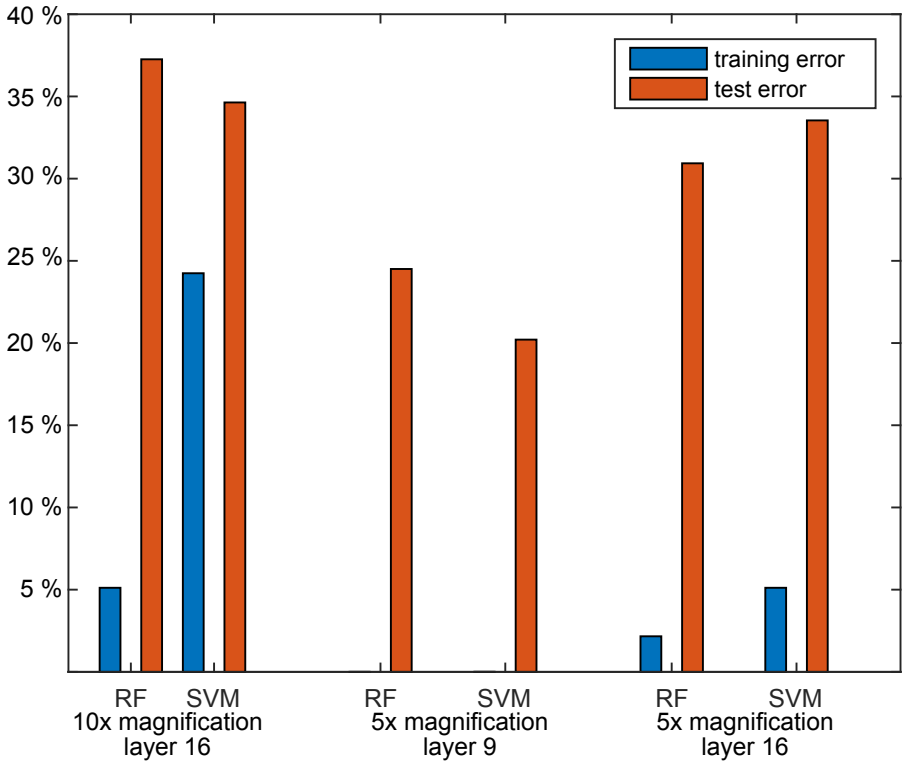


Figure 6.3: Training and test error per small patch for both random forest and support vector machines for different magnifications and layers. Some of the training errors are very close to zero which could be an indication of overfitting.

Table 6.2: Confusion matrix for 10x magnification, layer 16, classified with support vector machines. The matrix shows the classified patches. The overall error is 33.2 %.

		estimated class			
		benign	3	4	5
true class	benign	4812	843	665	511
	3	1440	2167	1635	1126
	4	474	738	4579	1361
	5	269	438	778	9097

vector machines classifier. The training error is shown in blue and the test error in red.

Confusion matrices for the best classifier for the different magnifications and layers when whole images were classified are shown in Table 6.5 - 6.7. Table 6.5

Table 6.3: Confusion matrix for 5x magnification, layer 9, classified with support vector machines. The matrix shows the classified patches. The overall error is 18.9 %.

		estimated class			
		benign	3	4	5
true class	benign	3987	632	194	78
	3	716	2620	696	353
	4	213	636	4099	306
	5	58	239	176	7772

Table 6.4: Confusion matrix for 5x magnification, layer 16, classified with random forest. The matrix shows the classified patches. The overall error is 28.1 %.

		estimated class			
		benign	3	4	5
true class	benign	774	50	45	38
	3	178	244	178	154
	4	56	82	539	278
	5	23	29	81	1493

shows the result for 10x magnification and layer 16, Table 6.6 the result for 5x magnification and layer 9 and Table 6.7 the result for 5x magnification and layer 16.

Table 6.5: Confusion matrix for 10x magnification, layer 16, classified with random forest. The matrix shows the classified images. The overall error is 20.7 %.

		estimated class			
		benign	3	4	5
true class	benign	52	0	0	0
	3	9	26	5	12
	4	0	5	37	10
	5	0	0	3	54

For the binary classification benign tissue versus cancerous tissue in whole images we discovered a true positive rate of 96.3 % and a false negative rate of

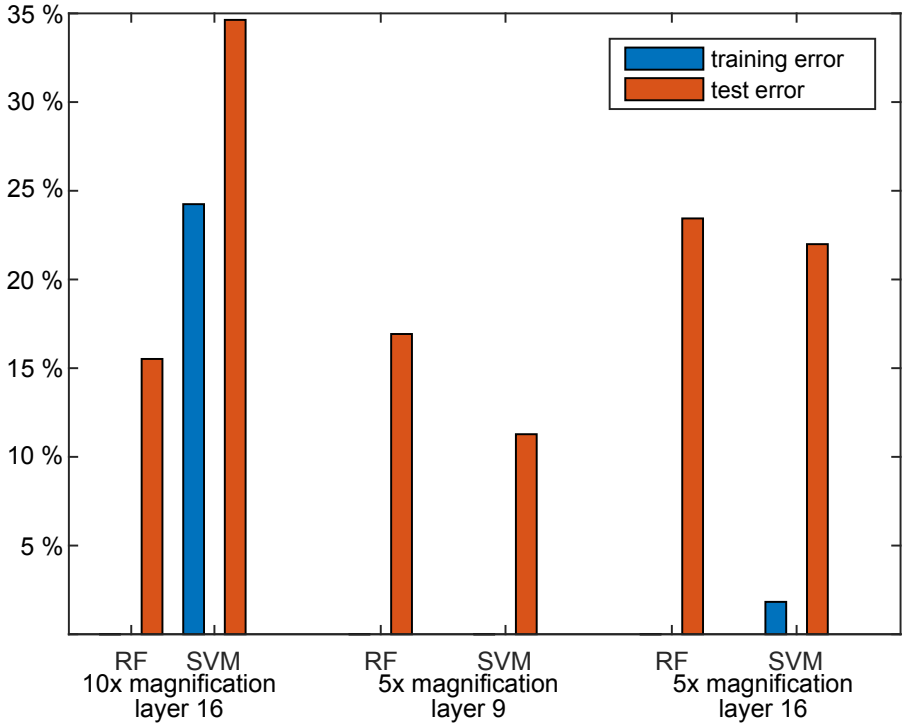


Figure 6.4: Training and test error per whole image for both random forest and support vector machines for different magnifications and layers. Also here some of the training errors are very close to zero.

Table 6.6: Confusion matrix for 5x magnification, layer 9, classified with support vector machines. The matrix shows the classified images. The overall error is 10.8 %.

		estimated class			
		benign	3	4	5
true class	benign	52	0	0	0
	3	4	40	3	5
	4	2	6	42	2
	5	0	0	1	56

3.7 % using Table 6.6.

Table 6.7: Confusion matrix for 5x magnification, layer 16, classified with support vector machines. The matrix shows the classified images. The overall error is 26.8 %.

		estimated class			
		benign	3	4	5
true outcome	benign	47	4	1	0
	3	7	24	14	7
	4	4	8	34	6
	5	0	3	3	51

6.4 Conclusions and Future Work

As a first step towards a semi-automatic tool for analyzing prostate biopsies we have presented a method to automatically classify images into benign tissue and Gleason score 3-5. The framework perform with an accuracy of 81.1 % when analyzing small patches of the image, retaining the spatial resolution of the classification. When classifying entire images the accuracy was 89.2 %. This level of accuracy is on the same level as previous work, but without using hand-crafted features or other pre-processing of the images. In the future, it would be very interesting to see if a network trained on pathologiy images could perform even better.

Part III

Traffic Surveillance

Chapter 7

Preliminaries on Tracking and Reconstruction

Given two or more two-dimensional images of the same object taken from different positions, a three-dimensional model of the object can be created. The process of getting the 3D position of a point given its location in the images is called triangulation. How the object is projected into the images depends on the rotation and position of the cameras, often those parameters has to be estimated as well.

To be able to reconstruct an object we need corresponding points in the images. Corresponding points are images of the same 3D point in the different images. There exist many methods for identifying point correspondences from two or more images. Usually a point is described by some kind of descriptor that looks in a neighborhood around the point. By doing the same for detected points in another image a correspondence can be found by comparing the descriptors. A very well know descriptor is the SIFT descriptor described in [59]. Some other interest point detectors are SURF [7], FAST [75] and BRISK [57]. If the change between the images is very small, for example if two consecutive frames from a video is used as input images, the problem of finding tentative correspondences can be solved using tracking of interest points. In this chapter such a tracking algorithm is presented together with a description of a camera model.

7.1 The Pinhole Camera

A very simple camera is the pinhole camera. In the pinhole camera, the rays from the object passes through a small hole and hits a photosensitive plate, where an image of the object is created. All rays hitting the plate go through the hole, which is therefore called the focal point or the camera center. The plate where

the image is created is called the image plane and lies on a distance f from the focal point. The equation of the image plane is $z = f$ and the distance f is called the focal length. Figure 7.1 shows a model of a pinhole camera. In the model, the image plane lies in front of the camera. Placing the image plane in front of the camera center is equivalent to placing it behind the camera center. The point, $\boldsymbol{X}$ is a point in space, a 3D point. The image point, $\boldsymbol{x}$, is where the ray from the 3D point to the focal point intersects with the image plane. $\boldsymbol{C}$ is the camera center or focal point. The principal axis goes through the camera center and is perpendicular to the image plane. The principal point, $\boldsymbol{p}$ is where the principal axis hits the image plane.

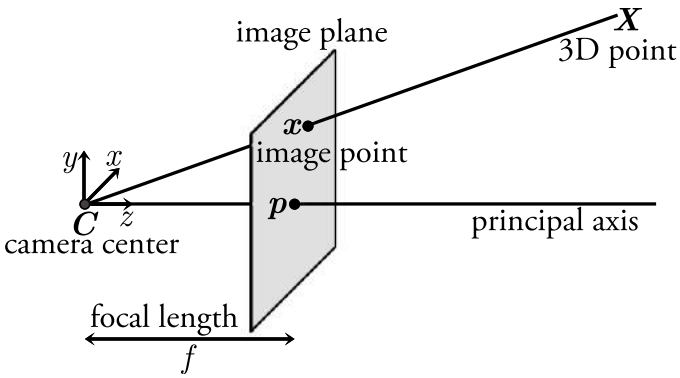


Figure 7.1: A model of the pinhole camera with the image plane in front of the camera. The distance between the camera and the camera center is called the focal length of the camera. The point $\boldsymbol{x}$ in the image plane where the 3D point $\boldsymbol{X}$ is pictured is called an image point. The principal axis of the camera is perpendicular to the image plane, goes through the camera center and hits the image plane at the principal point $\boldsymbol{p}$.

The coordinates of the image point can easily be calculated from similar triangles. The 3D point has the coordinates (X, Y, Z) and the image point has the coordinates (x, y, f), where the last coordinate can be omitted. In the case where the principal point is in the origin of the image coordinate system, the x-coordinate of the image point can be calculated with help of similar triangles as

$$\frac{x}{f} = \frac{X}{Z} \Rightarrow x = \frac{fX}{Z}. \tag{7.1}$$

In the same way, the y-coordinate can be calculated as

$$y = \frac{fY}{Z}. \tag{7.2}$$

In the usual case, when the principal point is not in the origin of the image, the image point becomes

$$(x, y) = \left(\frac{fX}{Z} + p_x, \frac{fY}{Z} + p_y \right), \tag{7.3}$$

where p_x is the x-coordinate of the principal point and p_y is the y-coordinate.

If the image point and the 3D point are expressed in homogeneous coordinates, that is $\boldsymbol{x} = \begin{pmatrix} x & y & 1 \end{pmatrix}^T$ and $\boldsymbol{X} = \begin{pmatrix} X & Y & Z & 1 \end{pmatrix}^T$, the image point can be expressed with a matrix multiplication

$$\lambda \boldsymbol{x} = \begin{pmatrix} Zx \\ Zy \\ Z \end{pmatrix} = \underbrace{\begin{pmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\boldsymbol{P}} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{7.4}$$

$\boldsymbol{P}$ is called the camera matrix and the equation above is called the camera equation. The scalar λ is the depth of the point, which is the distance along the z-axis from the camera center to the 3D point. Using vector notation the equation can be written as

$$\lambda \boldsymbol{x} = \boldsymbol{P}\boldsymbol{X}, \tag{7.5}$$

where both $\boldsymbol{x}$ and $\boldsymbol{X}$ are in homogeneous coordinates.

The camera matrix can be decomposed to

$$\boldsymbol{P} = \boldsymbol{K} \left(\begin{array}{c|c} \boldsymbol{I} & \boldsymbol{0} \end{array} \right), \tag{7.6}$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{K}$ is called the calibration matrix and is given by

$$\boldsymbol{K} = \begin{pmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{pmatrix}. \tag{7.7}$$

The calibration matrix contains the internal parameters of the camera matrix, such as the focal length and the principal point. The full calibration matrix also holds two more parameters, s and γ. The parameter s is called the skew parameter and is zero for most cameras. The aspect ratio, γ models the width and height ratio of

the pixels. For most cameras, the pixels are square and then γ is one. Including both the skew and aspect ratio in the calibration matrix it becomes

$$\boldsymbol{K} = \begin{pmatrix} f & sf & p_x \\ 0 & \gamma f & p_y \\ 0 & 0 & 1 \end{pmatrix}. \tag{7.8}$$

The camera matrix in (7.6) is in the camera coordinate system, the camera center is in the origin and the 3D point is expressed in a coordinate system with the z-axis pointing from the camera perpendicular to the image plane. If the camera center is not in the origin and the camera is rotated relative to the world coordinate system, the camera matrix becomes

$$\boldsymbol{P} = \boldsymbol{K}\boldsymbol{R}\left(\begin{array}{c|c} \boldsymbol{I} & -\boldsymbol{C} \end{array} \right), \tag{7.9}$$

where $\boldsymbol{R}$ is an orthogonal matrix expressing the rotation of the camera and $\boldsymbol{C}$ is the camera center.

The camera equation, $\lambda\boldsymbol{x} = \boldsymbol{P}\boldsymbol{X}$, can then, with $\boldsymbol{X}$ in cartesian coordinates, be written as

$$\lambda\boldsymbol{x} = \boldsymbol{K}\boldsymbol{R}\left(\begin{array}{c|c} \boldsymbol{I} & -\boldsymbol{C} \end{array} \right) \begin{pmatrix} \boldsymbol{X} \\ 1 \end{pmatrix}. \tag{7.10}$$

The image point is then calculated by

$$\lambda\boldsymbol{x} = \boldsymbol{K}\boldsymbol{R}(\boldsymbol{X} - \boldsymbol{C}), \tag{7.11}$$

where $\boldsymbol{x}$ is in homogeneous coordinates and $\boldsymbol{X}$ is in cartesian coordinates. For a more detailed derivation of the camera matrix see [34].

7.2 Kanade-Lucas-Tomasi Tracker

Tracking is a way to find corresponding points in images. In tracking we have a sequence of images, for example a movie where the difference between two consecutive frames is fairly small. By finding an interesting point in one image the corresponding point in the next image is found by looking in a small area around the position of the point in the first image. By just looking at a single pixel it is not possible to determine if it is the same pixel in the next image, it could easily be confused with adjacent pixels. Instead of tracking single points

windows of pixels, a small patch of the image, are tracked. One of the most popular tracker is the Kanade-Lucas-Tomasi tracker, known as the KLT-tracker, presented in [82]. The KLT-tracker uses the method presented in [60] to track window from one frame to the next. It also present a novel way to select features in the image that could be tracked to the next frame.

7.2.1 Tracking

A sequence of images can be written as a function $I(x, y, t)$ where x and y are space variables and t is the time. If two images are taken closely after each other, the function $I(x, y, t)$ has to satisfy

$$I(x, y, t + \tau) = I(x + \xi, y + \eta, t), \tag{7.12}$$

where τ is the time delay between the pictures, ξ and η are displacements in the x and y direction respectively. The vector $\boldsymbol{d} = (\xi, \eta)$ denotes the total displacement.

In real world this relation is almost always violated, so we have to allow some error. We set $J(\boldsymbol{x}) = I(x, y, t + \tau)$ and $I(\boldsymbol{x} - \boldsymbol{d}) = I(x + \xi, y + \eta, t)$. Then we can write J as

$$J(\boldsymbol{x}) = I(\boldsymbol{x} - \boldsymbol{d}) + n(\boldsymbol{x}), \tag{7.13}$$

where $n(\boldsymbol{x})$ denotes noise.

Our goal is to find the displacement vector, $\boldsymbol{d}$, that minimizes the residual error over the given window, $\mathcal{W}$. This error is given by

$$\epsilon = \iint_{\mathcal{W}} (I(\boldsymbol{x} - \boldsymbol{d}) - J(\boldsymbol{x}))^2 \, w \, \mathrm{d}x\mathrm{d}y, \tag{7.14}$$

where w is a weighting function. The weighting function could for example be constant or a gaussian that emphasizes the central part of the image patch.

In the KLT-tracker this problem has been solved by using the linearization method that was suggested in [60]. For small displacements we can estimate $I(\boldsymbol{x} - \boldsymbol{d})$ by its taylor series around $\boldsymbol{x}$ truncated by the linear term. This approximation is given by

$$I(\boldsymbol{x} - \boldsymbol{d}) \approx I(\boldsymbol{x}) - \boldsymbol{g}^T \boldsymbol{d}, \tag{7.15}$$

where $\boldsymbol{g} = \nabla I(\boldsymbol{x})$ is the gradient of the image.

Then we can write the error residual as

$$\epsilon = \iint\limits_{\mathcal{W}} \left(I(\boldsymbol{x}) - \boldsymbol{g}^T\boldsymbol{d} - J(\boldsymbol{x})\right)^2 w \,\mathrm{d}x\mathrm{d}y = \iint\limits_{\mathcal{W}} \left(h - \boldsymbol{g}^T\boldsymbol{d})\right)^2 w \,\mathrm{d}x\mathrm{d}y, \tag{7.16}$$

where $h = I(\boldsymbol{x}) - J(\boldsymbol{x})$ is the difference between the two images.

The minima can be found in closed form by taking the derivative with respect to $\boldsymbol{d}$ and set it to zero. Then we get

$$\iint\limits_{\mathcal{W}} \left(h - \boldsymbol{g}^T\boldsymbol{d}\right)\boldsymbol{g} w \,\mathrm{d}x\mathrm{d}y = \boldsymbol{0}. \tag{7.17}$$

Since $\left(\boldsymbol{g}^T\boldsymbol{d}\right)\boldsymbol{g} = \boldsymbol{g}\left(\boldsymbol{g}^T\boldsymbol{d}\right) = \left(\boldsymbol{g}\boldsymbol{g}^T\right)\boldsymbol{d}$ and $\boldsymbol{d}$ is assumed to be constant on the window $\mathcal{W}$ we can rewrite this as

$$\left(\iint\limits_{\mathcal{W}} \boldsymbol{g}\boldsymbol{g}^T w \,\mathrm{d}x\mathrm{d}y\right)\boldsymbol{d} = \iint\limits_{\mathcal{W}} h\boldsymbol{g} w \,\mathrm{d}x\mathrm{d}y. \tag{7.18}$$

If we set G to be the symmetric 2×2 matrix given by $G = \iint_{\mathcal{W}} \boldsymbol{g}\boldsymbol{g}^T w \,\mathrm{d}x\mathrm{d}y$ and $\boldsymbol{e}$ to be the 2×1 vector given by $\boldsymbol{e} = \iint_{\mathcal{W}} h\boldsymbol{g} w \,\mathrm{d}x\mathrm{d}y$ we can write this as the linear equation system

$$G\boldsymbol{d} = \boldsymbol{e}. \tag{7.19}$$

The displacement $\boldsymbol{d}$ is the vector that solves the linear equation system.

7.2.2 Finding Good Features

For the tracking algorithm to work well it is necessary that the window that is tracked contains sufficient information. Many other papers suggest to use points like corner points or points with high spatial frequency content. The approach suggested in [82] define a good feature to track if the displacement vector can be reliable computed. The equation system in (7.19) can be solved in a reliable way if the matrix G is well-conditioned and above the noise level. Since G is symmetric, the condition number of G is given by

$$\kappa(G) = \left|\frac{\lambda_{max}}{\lambda_{min}}\right|, \tag{7.20}$$

where $\kappa(G)$ is the condition number, λ_{max} is the larger of the two eigenvalues of G and λ_{min} the smallest. The matrix is well-conditioned if the condition number is low.

In order for G to be above the noise level, both of the eigenvalues need to be large. At the same time two eigenvalues can not differ too much for G to be well-conditioned. In practice it is enough to look at the smallest of the two eigenvalues since there is an upper limit for the eigenvalues. The upper limit exist since there is an upper limit for the pixel values. This implies that the matrix will be both well-conditioned and above noise level, and therefore a good window to track, if

$$\min(\lambda_{max}, \lambda_{min}) > \lambda, \tag{7.21}$$

where λ is a pre-defined threshold. By computing the eigenvalues for areas with approximately uniform intensities, a lower bound for λ was obtained. An upper bound was found by looking att the eigenvalues of areas at corners or rich texture. A value in the middle of the interval was chosen.

Chapter 8

Tracking and Reconstruction of Vehicles

To reduce the number of road traffic injuries it is important to know how safe certain roads and intersections are. There are different ways to evaluate this. The classic method is to count the number of accidents that occurs. Since accidents are rare it can take years to get a good assessment of safety this way. A faster approach is to predict the numbers of accidents that will happen by manually observing certain events, conflicts, during a much shorter time period [40]. This is done by letting trained personnel study video of the intersection. Naturally, this is very expensive and time-consuming and an automated method would be very beneficial.

This chapter introduce a new way to estimate the position of a vehicle in an intersection by tracking the vehicles in a movie, build models of the vehicles and then estimate the pose of the vehicles in all frames. The work in this chapter is based on the work in our article [44].

8.1 Related Work

In [53] a method for automated surveillance was proposed. To calculate the position of a vehicle, the 2D image of that vehicle was projected onto the road plane. If the camera can be placed right above the intersection, this will work rather well. In most cases though, this is not possible. The projection of the vehicle gets stretched out and the estimated position incorrect. A better way to estimate the position would be to make a three-dimensional representation of the object and use this to calculate the position of the vehicle. This is the approach considered in this chapter.

There is much work in the vision literature regarding traffic scenes. In [56] a

system for tracking pedestrians and cars was presented, which is based on object detectors. A system for making 3D shape reconstruction for traffic surveillance with multiple cameras is presented in [65]. To do this, predefined 3D models of cars are used. In [20] vehicles was tracked by tracking feature points on it. After the features exit the tracking region, they are grouped into discrete vehicles using a motion constraint. In [88] a system for automatic calibration of a camera from traffic scenes was proposed. If the height of the camera is known, both intrinsic and extrinsic parameters can be found. A method to rectify images is given in [8]. By tracking the motion of two vehicles moving in constant speed an estimation of the ground plane can be done.

The approach described in this chapter differs from most methods in at least one important aspect. Inspired by recent research in optimal methods for computer vision, the reprojection errors are minimized with respect to the L_∞-norm rather than the more common L_2-norm. This makes it possible to find the global optimum and it also makes it easier to impose extra constraints, such as the fact that vehicles are only moving in the ground plane.

8.2 Overview

We start with captured video from the intersection. For the reconstruction we need to find corresponding points between different frames. This is achieved in the following way. Every few frames, we pick an image to use as a starting image. In this image a corner detector is used to find strong corner points. These points are then tracked a few frames using a KLT tracker [82], to get corresponding image points in a later frame.

To reduce the amount of outliers in the 3D reconstruction, we need to detect which points belong to the same vehicle. For this purpose we perform a motion segmentation algorithm that will be described in detail in Section 8.4.1. This algorithm finds all vehicles that are visible in the starting frame. The vehicles are then tracked both forwards and backwards in time until the vehicles leaves the camera field of view or until the tracking fails.

This procedure yields long tracks of corresponding image points likely to belong to the same vehicle. From these tracks we use a few views to perform our 3D reconstruction, which is described thoroughly in Section 8.4.2. The scale is fixed by assuming that some reconstructed points lie in the ground plane. Having a 3D model we use this to estimate the pose of the vehicle in all views, see

Section 8.4.3. This gives us accurate information on the movement of the vehicle throughout the sequence. Section 8.5 described how the same framework can be used to estimate the camera rotation with respect to the ground plane. The idea is to do this once when the system is set up.

System Overview
1. Track points between frames.
2. Cluster points from the same vehicle.
3. Reconstruct the vehicle using a few views.
4. Compute the vehicle pose in all views.

8.3 Preliminaries

We assume that the ground is planar and that the vehicles are rigid bodies that only moves in that plane. We choose coordinate system, such that the z-axis is perpendicular to the ground. We also assume that the camera has known internal parameters. In reality we have one stationary camera and moving vehicles, but from the vehicles point of view it look like the camera is moving. Choosing this point of view, we can assume stationary vehicles and multiple cameras.

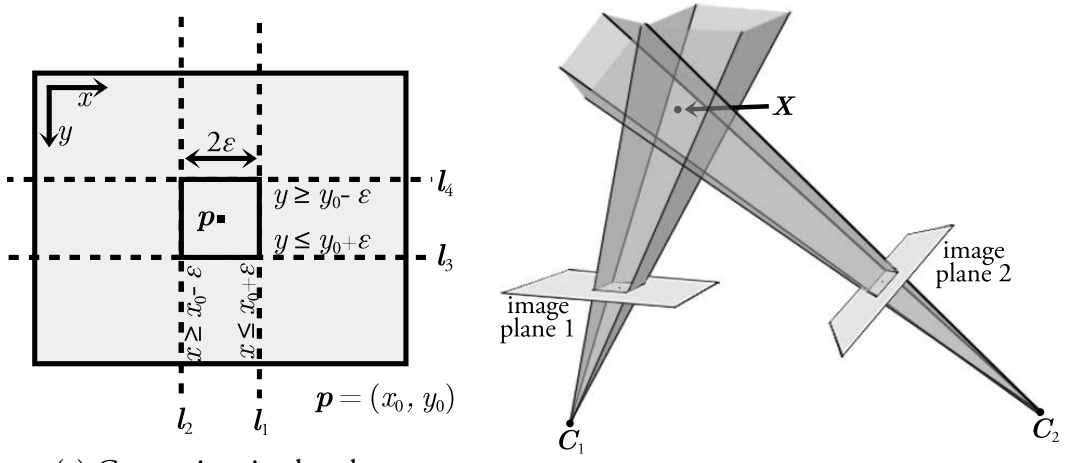
8.3.1 Structure from Motion with Known Rotations

If the rotation between the different cameras are known we can solve the structure from motion problem with multiple cameras optimally with respect to the L_∞-norm, using second-order cone programming [42]. This gives us the translation of the cameras and the position of the 3D points. An advantage of this approach is that several views are used to estimate the 3D points, unlike the standard approach that gets initial estimates from only two views.

Due to noise there is no exact solution to the reconstruction problem. Thus we allow the reprojected points to deviate somewhat from the measured image points. To get linear constraints, we allow the reprojected point within a small square around the measured image point. This differs from the approach in [42]. The simplification allows us to solve the optimization using linear programming rather than second-order cone programming.

The square has side length 2ε, where ε is the tolerated error, and is defined by four lines in the image plane, see Figure 8.1a. These lines are the intersections

between the image plane and planes going through both the camera center and the image plane. The planes through the camera center form a generalized cone in which the 3D point has to be. A bigger value on ε results in a wider cone. For every camera, we get another cone constraint on the position of the 3D point. The constraints from two views can be seen in Figure 8.1b.



(a) Constraints in the plane.

(b) Constraints in space.

Figure 8.1: Linear constraints in the image plane and in space. (a) shows the lines that define the box in which the reprojected point must lie inside. (b) shows constraints in space from two cameras, the 3D point must lie inside both cones.

If we express the 3D point in cartesian coordinates we can write the camera equation on the form

$$\lambda \boldsymbol{x} = \boldsymbol{K}\boldsymbol{R}\left(\boldsymbol{X} - \boldsymbol{t}\right), \tag{8.1}$$

where λ is the depth of the point, $\boldsymbol{x}$ the image point in homogeneous coordinates, $\boldsymbol{K}$ the camera calibration matrix, $\boldsymbol{R}$ the rotation of the camera, $\boldsymbol{X}$ the 3D point in cartesian coordinates and $\boldsymbol{t}$ is the camera center. For calibrated cameras we can exclude the calibration matrix and get

$$\lambda \boldsymbol{x} = \boldsymbol{R}\left(\boldsymbol{X} - \boldsymbol{t}\right). \tag{8.2}$$

From each one of the lines in Figure 8.1a we get constraints on the form, $\boldsymbol{l}^T\boldsymbol{x} \leq 0$, where $\boldsymbol{l}$ is the equation of the line and $\boldsymbol{x}$ is the image point in homogeneous

coordinates. Using the camera equation in (8.2) we can rewrite the constraints to

$$\boldsymbol{l}^T \frac{1}{\lambda} \boldsymbol{R} (\boldsymbol{X} - \boldsymbol{t}) \leq 0. \tag{8.3}$$

The depths has to be positive if the point is visible in the camera, therefore we can eliminate the depth from the equation. If we set $\boldsymbol{a} = \boldsymbol{l}^T \boldsymbol{R}$ we get

$$\boldsymbol{a} (\boldsymbol{X} - \boldsymbol{t}) \leq 0, \tag{8.4}$$

defining a plane through the camera center and the image plane as shown in Figure 8.1b.

For every 3D point there are four inequalities per camera, one for each plane. We collect all these inequalities into a matrix

$$\boldsymbol{A}\boldsymbol{u} \leq \boldsymbol{0}, \tag{8.5}$$

where $\boldsymbol{A}$ holds the coefficients and $\boldsymbol{u}$ holds the unknowns, consisting of the 3D points and the camera centers. Checking if there exists a solution that fulfills all constraints for a fixed error tolerance ε is a linear programming feasibility problem. To find the minimal ε we can use bisection.

A weakness of this approach is that it is sensitive to outliers. One way to get around this is to use auxiliary variables, as described in [69]. To each row of (8.5) we add a variable $s_i \geq 0$, yielding

$$\boldsymbol{a}_k^T \boldsymbol{x} \leq s_i. \tag{8.6}$$

There is one s_i for each measured image point. For example, if we have five views with ten image points in each view we get 50 auxiliary variables. Ideally we would like as many s_i's as possible to be zero, but since this is a very hard optimization problem we try to minimize the sum instead. This turns the convex feasibility problem in (8.5) into a convex optimization problem,

$$\begin{cases} \text{minimize} & \sum s_i \\ \text{subject to} & \boldsymbol{A}\boldsymbol{u} \leq \boldsymbol{P}\boldsymbol{s}, \end{cases} \tag{8.7}$$

where $\boldsymbol{P}$ is a matrix picking the relevant s_i.

8.3.2 Minimal Solver

This section presents a method to calculate the rotation and translation between two frames. The method is minimal in the sense that it requires a minimal number of correspondences. Assuming planar motion and known camera rotation with respect of the ground, we need two pairs of corresponding points.

For all cameras we have same rotation, $\boldsymbol{R}_0$, with respect to the ground and we can rewrite the camera equation (8.1)

$$\lambda \boldsymbol{x} = \boldsymbol{K}\boldsymbol{R}_0\boldsymbol{R}_z\left(\boldsymbol{X} - \boldsymbol{t}\right), \tag{8.8}$$

where $\boldsymbol{R}_z$ is the rotation around the z-axis.

Using normalized image points $\boldsymbol{u} = \boldsymbol{R}_0^T\boldsymbol{K}^{-1}\boldsymbol{x}$ we get

$$\lambda \boldsymbol{u} = \boldsymbol{R}_z\left(\boldsymbol{X} - \boldsymbol{t}\right). \tag{8.9}$$

For the first camera we set both the rotation around the z-axis and the translation to zero, getting the simpler relations,

$$\lambda_k \boldsymbol{u}_k = \boldsymbol{X}_k, \quad \text{for } k = 1, 2, \tag{8.10}$$

where $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are the two image points.

For the second camera we let γ_1 and γ_2 denote the depths and use $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ for the image points. We get

$$\gamma_k \boldsymbol{R}_z^T\boldsymbol{v}_k = \left(\boldsymbol{X}_k - \boldsymbol{t}\right), \quad \text{for } k = 1, 2. \tag{8.11}$$

Since we are dealing with planar motion the translation is zero in the z-direction, so $\boldsymbol{t} = (t_x, t_y, 0)^T$. The rotation, $\boldsymbol{R}_z$, is given by a single rotation angle, θ.

If we know all four depths, λ_1, λ_2, γ_1 and γ_2, we can calculate the 3D points from (8.10) and the rotation and translation from (8.11). Hence, one way to solve the problem is to determine the depths. First note that the depth γ_k can be expressed in λ_k using the z-component of equations (8.10) and (8.11). We get

$$X_{kz} = \lambda_1 u_{kz} = \gamma_1 v_{kz} \text{ and } \gamma_k = \frac{u_{kz}}{v_{kz}}\lambda_k. \tag{8.12}$$

To calculate the ratio between λ_1 and λ_2 we use the geometry of the scene. The scene seen from above is showed in Figure 8.2. For each camera we get one triangle with corners in the camera center and the two 3D points.

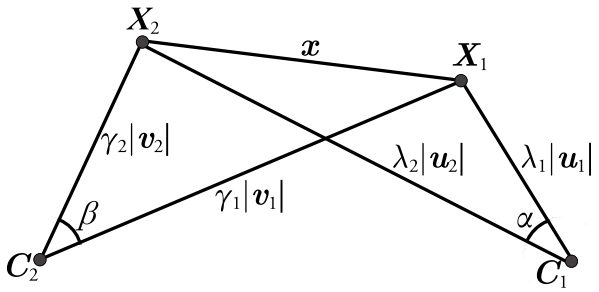


Figure 8.2: The scene seen from above.

The two triangles share the side x and the length of this side can be calculated from either of the triangles using the law of cosine. That gives us the equation

$$\lambda_1^2 |\boldsymbol{u}_1|^2 + \lambda_2^2 |\boldsymbol{u}_2|^2 - 2\lambda_1\lambda_2 |\boldsymbol{u}_1| |\boldsymbol{u}_2| \cos\alpha = \\ = \gamma_1^2 |\boldsymbol{v}_1|^2 + \gamma_2^2 |\boldsymbol{v}_2|^2 - 2\gamma_1\gamma_2 |\boldsymbol{v}_1| |\boldsymbol{v}_2| \cos\beta, \quad (8.13)$$

where α is the angle between the two vectors $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$, being the image points from the first camera. In the same way β is the angle between vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ of the second camera.

By setting λ_1 to 1 we fixate the scale. Then we have three unknown depths and three equations. Inserting (8.12) into (8.13) yields an equation of second degree that gives us up to two real solutions for the depths.

For each of these solutions we calculate the rotation and translation. This is done by solving the linear equation system below. From (8.11) we have four unused equations,

$$\begin{cases} \gamma_1 (av_{1x} + bv_{1y}) = X_{1x} - t_x, \\ \gamma_1 (-bv_{1x} + av_{1y}) = X_{1y} - t_y, \\ \gamma_2 (av_{2x} + bv_{2y}) = X_{2x} - t_x, \\ \gamma_2 (-bv_{2x} + av_{2y}) = X_{2y} - t_y, \end{cases} \quad (8.14)$$

where $a = \cos\theta$ and $b = \sin\theta$.

8.4 Tracking and Reconstruction

To do the reconstruction we have to find corresponding image points in the different frames in the video. To get that we find interest points in one frame and track those points to the following frames. Then we cluster the points so that

points in one cluster belongs to the same vehicle. When we have corresponding image points for a vehicle we use a few frames to do a 3D reconstruction. The reconstruction is then used to calculate the pose for all frames.

8.4.1 Tracking

To detect vehicles in the video, we start by finding interest points in one frame somewhere in the sequence. Then we track these points a number of frames using a KLT-tracker [82]. To separate points on stationary objects from points on moving objects we remove all points that has not moved between the first and the last frame. For the rest of the points we perform a motion segmentation algorithm to find out which points belongs to the same vehicle. For all the vehicles found in the scene the feature points are tracked until the vehicle leaves the scene. When all vehicles in the scene are found we choose another frame later in the sequence. This is repeated until we come to the end of the video.

The motion segmentation algorithm works by choosing two image points from two images, the first and the last image. From these two points the rotation and relative translation between the frames are calculated using the minimal solver from Section 8.3.2. The minimal solver requires that the rotation of the camera with respect to the ground is known. Depending on the number of real solutions for the equation of second degree in (8.13) the solver returns up to two solutions for the rotation and translation. For these solutions the rotation and translation are used to create two camera matrices per solution, one for each view. The camera matrices are calculated by

$$\boldsymbol{P}_1 = \boldsymbol{K}\boldsymbol{R}_0\left(\boldsymbol{I}|\boldsymbol{0}\right) \text{ and } \boldsymbol{P}_2 = \boldsymbol{K}\boldsymbol{R}_0\boldsymbol{R}_z\left(\boldsymbol{I}|-\boldsymbol{t}\right), \tag{8.15}$$

where $\boldsymbol{K}$ is the camera calibration matrix, $\boldsymbol{R}_0$ is the camera rotation with respect to the ground, $\boldsymbol{R}_z$ is the rotation around the z-axis between the frames and $\boldsymbol{t}$ is the translation.

The camera matrices are then used to triangulate all image points to get the positions of the 3D points in space, the triangulation is done with the method proposed in [33]. The 3D points are then reprojected with both cameras

$$\lambda\hat{\boldsymbol{x}}_i = \boldsymbol{P}_i\boldsymbol{X}, \tag{8.16}$$

where $\hat{\boldsymbol{x}}_i$ is the reprojected image points for view i and $\boldsymbol{X}$ are the 3D points. The reprojection errors, $\varepsilon_{ij} = \|\boldsymbol{x}_{ij} - \hat{\boldsymbol{x}}_{ij}\|_2$, are measured for all points in both

views. Points, j, that have a small reprojection error in both views, $\varepsilon_{1j} < \varepsilon_t$ and $\varepsilon_{2j} < \varepsilon_t$, are classified as inliers and are likely to belong to the same vehicle. The number of inliers is counted for the different rotations and translations and then two new image points are chosen. The algorithm is then repeated for all pair of image points and the number of inliers for all possible solutions are counted. The solution that gives the highest number of inliers is chosen.

The points classified as inliers are then tracked with the KLT-tracker until the vehicle leaves the scene. Points that fails to be tracked are removed. First we track the points from one frame to the next and then back again, points that do not return to the original position are removed. We also remove points which comes too close to the edge of the image. We stop the tracking when there are no points left. To get longer tracks we start by following the vehicle forward in time, from the starting frame to the following frames until the vehicle drives away. Then we go back to the starting frame and follow the vehicle backwards in time, from the starting frame to previous frames until the vehicle disappears.

The points classified as inliers in this estimation are removed, and the motion segmentation algorithm is restarted to search for more vehicles. The motion segmentation algorithm is repeated until there are too few points left.

Since we choose new starting frames at short intervals we typically get several tracks of the same vehicle. To avoid reconstructing the same vehicle more than once, we try to detect this by comparing the position of the image points in the different tracks. Tracks where the positions for some image points coincide with image points in other tracks for corresponding frames are likely to represent the same vehicle.

For a frame in the first track, the corresponding frames are found in the following tracks. The image points are compared between the different tracks, and if any image points coincide the tracks are merged together.

Tracking
The first frame in the video sequence is chosen as starting frame

1. Detect interest points.
2. Track the points a number of frames and remove points that do not move.
3. Find points belonging to the same vehicle by performing the motion segmentation algorithm.
4. Track the vehicle, until it leaves.
5. Remove points classified as inliers.
6. Repeat step 3-5 until there are too few points left.
7. Choose a new starting frame, a few frames after the previous starting frame.
8. Repeat step 1-7 until the end of the video is reached.

8.4.2 Reconstruction

After the tracking algorithm we have tracks of corresponding image points from that the vehicles enter the camera field of view until the vehicles leave. We will use a few of these views, typically 10, to make a 3D reconstruction of the vehicles.

First we have to know the rotation between the frames we have chosen. Still the vehicles only rotate around the z-axis and we can use the minimal solver from Section 8.3.2 to calculate the rotations for consecutive views. The rotations are calculated in the same way as in the motion segmentation. We calculate the rotation for two points at a time, then all points are triangulated and reprojected. This is done for all pairs of points and we choose the rotation that gives the highest number of inliers.

Knowing the rotation we can use the method from Section 8.3.1 to compute the reconstruction. To avoid translation ambiguity we fixate the position of the first camera. We also know that the vehicles do not translate in the z-direction. Then we can fixate the z-coordinate for all cameras, we also assume that the camera is placed above the vehicles and we set a upper limit of the z-coordinate for all 3D points. This limit we set somewhat lower than the height of the camera. Then we solve the optimization problem in (8.7). For the points that are inliers the value of the corresponding s_i will be very close to zero while outliers

have much higher value of the corresponding s_i. To remove outliers we remove 3D points where the corresponding s_i is higher than some small threshold for all views.

Now we have a reconstruction of the vehicle, but the scale is still unknown. For surveillance purposes it is important that the scale is consistent with respect to the other reconstructed vehicles. To achieve this we consider the point in the vehicle model that has the lowest z-coordinate. Assuming that this point is close to the ground, we choose the scale such that this point gets z-coordinate equal to zero.

8.4.3 Pose

When we have both the correspondences between frames for the vehicle and a 3D reconstruction of it, we can calculate the position of the vehicle by calculating the camera pose for all views. If the rotation of the camera is known we can calculate the pose with the method presented in Section 8.3.1, though now we just have one camera and we know the position of the 3D points. Hence we just have to calculate the position of a camera, significantly reducing the number of unknown variables.

To find the rotation of the camera we perform a branch and bound search through rotation space. The branch and bound algorithm is described in detail in Section 8.5, where it is used for a larger search space. In pose estimation, we only have to perform the rotation search in one dimension since we already know the rotation of the camera with respect to the ground and only want to find the rotation around the z-axis.

To handle outliers, we use the following scheme. We choose some of the points on the vehicle at random, typically half of the points, and calculate the pose using the chosen points. All 3D points are then projected with the estimated camera and the number of inliers is counted. This procedure is repeated a number of times with different points and the rotation that gives the highest number of inliers is chosen. Then the pose is calculated again using all points that were classified as inliers.

8.5 System Calibration

In all previous sections we assumed that the camera rotation with respect to the ground was known. To calibrate the system we need to find this rotation. This

is done by choosing a few views of some vehicle, preferably a big one, with corresponding image points, without outliers. For this vehicle a 3D reconstruction is calculated, at the same time we get the rotations and translations of the cameras.

To find the rotations between the different views we perform a branch-and-bound search, similar to that in [32]. The parameterization of the rotation is shown in Figure 8.3. The camera is first rotated around the z-axis, then around the x'-axis and at last around the z'-axis and the total rotation can be written as $\boldsymbol{R} = \boldsymbol{R}_{z'}\boldsymbol{R}_{x'}\boldsymbol{R}_{z}$,, where $\boldsymbol{R}$ is the total rotation and $\boldsymbol{R}_{z'}$, $\boldsymbol{R}_{x'}$ and $\boldsymbol{R}_{z}$ are the rotations around the z'-, x'- and z-axes respectively.

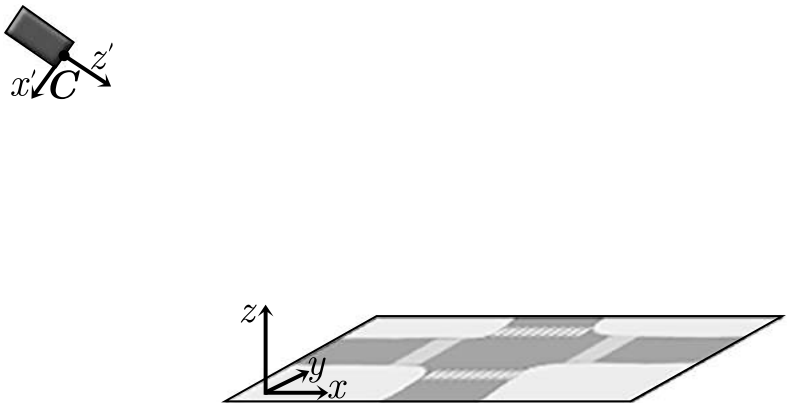


Figure 8.3: Rotation of the camera, the camera is first rotated around the z-axis, then around the x'-axis and at last around the z'-axis.

We choose coordinate system, such that the z-axis is perpendicular to the ground plane. This means that vehicles rotate only around the z-axis and thus $\boldsymbol{R}_{z'}$, $\boldsymbol{R}_{x'}$ are equal for all views. They specify the orientation of the ground plane relative to the camera. The rotation around the z-axis for the first camera can be chosen to be zero. Each of the angles can be chosen between -π and π. Thus our search space can be identified with the product space $[-\pi, \pi]^{(N+1)}$, where N is the number of cameras.

The branch and bound algorithm is initiated with a list containing one block, $[-\pi, \pi]^{(N+1)}$ as well as an initial error threshold ε. At each iteration, we pick a block from the list and try to determine if this block can contain any solution with reprojection errors $< \varepsilon$. This is determined by solving an LP feasibility problem as described in Section 8.3.1 but with error tolerance $\varepsilon + \Delta$ instead of ε. The Δ is an extra uncertainty that accounts for the size of the block. Details can be found in [43].

If this test falls out positive, then the block is divided and the new blocks are added to the list. Otherwise it is simply deleted. This continues until the remaining set of rotations is small enough.

Branch-and-bound Algorithm
Iterate until desired precision is reached.
1. Pick the first block from the list.
2. Calculate the constraints and set up the LP problem.
3. Determine if there is a solution to the LP problem.
4. If there is a solution.
   - Divide the block into smaller blocks and add them to the list.
   - Try to update the error threshold by performing the bisection algorithm.
5. Remove the current block from the list.

8.5.1 Bisection

In Section 8.3.1 we showed how to check feasibility for a fixed error tolerance ε. To find the L_∞ optimal solution we also need a method to update this tolerance. This is done with a bisection algorithm. For blocks that pass the feasibility test we try to find a solution having a smaller reprojection error. We fix the rotations to the middle of the block and perform a feasibility test with error tolerance ε. If this passes we know that we have found a better solution. To know how good, we use bisection.

We start with the interval $[0, \varepsilon]$. Let $\gamma = \varepsilon/2$ and check feasibility with error tolerance γ. If this is feasible we know that the best reprojection error is somewhere between 0 and γ and we set the upper bound to γ. If there is not a solution, we set the lower bound to γ instead. The interval is now half the length of the original interval. Again we try to find a solution in the middle of the interval and change either the upper or the lower bound. This is repeated until the interval is as short as desired. Finally we update the error threshold.

8.6 Experiments

The presented methods were evaluated on real-world data. The captured video has a resolution of 320 × 240 pixels and is around 10 minutes. The following sections describe the different parts of the evaluation.

8.6.1 Motion segmentation

To illustrate how the motion segmentation algorithm works, one frame from the video was selected. In this frame interest points were detected and tracked for a couple of frames. Figure 8.4a shows the first and last image with the image points marked with yellow dots. After removing stationary points, the points in Figure 8.4b remained. The result from the motion segmentation is shown in Figure 8.4c, the green dots are points belonging to the first vehicle and the red dots are points belonging to the second vehicle.

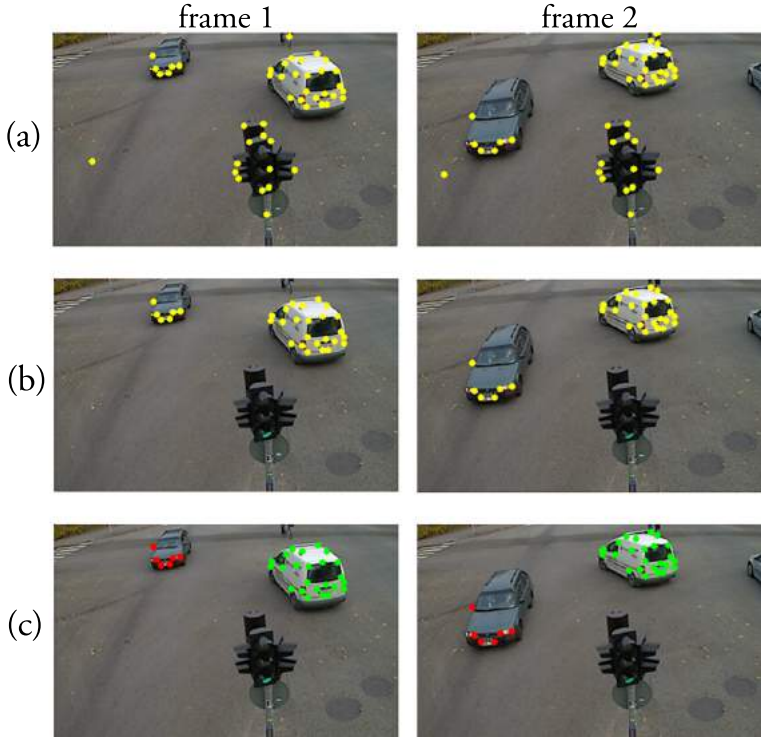


Figure 8.4: Illustration of the motion segmentation algorithm. (a) interest points in the first and last frame, (b) static points removed and (c) points after the motion segmentation, the green points belong to one of the vehicles and the red points to the other vehicle.

8.6.2 Tracking

To evaluate the tracking algorithm, we manually counted the number of vehicles in the entire video sequence to get the ground truth. We also noted in which direction the vehicles are driving. There are 16 ways to pass through the intersection. Next we calculated the number of vehicles that the tracking algorithm found and compared this number with the ground truth. The results are shown in Table 8.1. For some of the directions the tracking algorithm finds most of the vehicles. For other, the tracking algorithm fails more often. In the cases where the tracking algorithm works well, the vehicles drive closer to the camera, which gives fairly well resolution of the vehicles. Vehicles that drive far away is harder to track since they are very small. The KLT-tracker has to be able to track points on the vehicle for a number of frames for it to be detected, and when it fails, they will not be detected.

Table 8.1: Comparison between the real number of vehicles and the number the tracking algorithm finds. Each row represent one of the 16 different ways to pass through the intersection.

turn	ground truth	tracking
1	34	32
2	120	30
3	0	2
4	0	0
5	25	17
6	17	17
7	8	12
8	0	0
9	32	21
10	102	63
11	27	6
12	1	0
13	12	6
14	13	7
15	15	0
16	0	0
tot	406	213

8.6.3 Reconstruction and Pose

The reconstructions of the vehicles are made according to Section 8.4.2 and the pose for the vehicles in all frames where the vehicles are visible is calculated as in Section 8.4.3. For all frames, the L_∞-norm of the reprojection errors is calculated, that is the largest of the reprojection errors. Table 8.2 shows a summary for 20 vehicles. The reprojection errors are coordinate-wise and measured in pixels, the resolution of the images is 320×240 pixels. The number of points used to do the reconstruction varies between vehicles and the number of points used to calculate the pose varies between frames.

Figure 8.5 shows three frames for which the pose of the vehicles has been calculated. The dots representing image points on the different vehicles have different colors.

Figure 8.5: Three frames from the video. The pose for the three vehicles has been calculated for the frames where the vehicles are visible.

The 3D reconstruction of the vehicles and their relative position is shown in Figure 8.6. The points numbered with number 1 represent the frame to the left in Figure 8.5, the points numbered 2 represent the middle frame and points with number 3 represent the frame to the right.

Next, the 3D points were projected onto the ground plane to estimate the positions of the vehicles in the intersection, the result can be seen Figure 8.7. The positions can be compared with the images in Figure 8.5.

Finally the pose of the vehicles has been calculated for all frames where they are visible. Figure 8.8 shows the projection of the vehicles for all frames.

Table 8.2: Reprojection errors for 20 of the vehicles in the intersection, the table displays the largest, smallest and mean value of the L_∞ reprojection errors. The errors are measured in pixels and the resolution of the images is 320 $\times$ 240 pixels. The number of points varies between vehicles and frames.

vehicle	L_∞ error worst frame	L_∞ error best frame	L_∞ error mean
1	4.10	1.54	2.87
2	3.28	1.03	1.92
3	4.33	1.99	2.82
4	3.66	0.73	2.22
5	3.44	1.28	2.29
6	4.69	0.96	1.86
7	2.56	0.21	1.40
8	3.81	1.03	1.53
9	3.63	0.83	2.08
10	3.08	0.68	1.57
11	4.69	0.67	1.82
12	2.32	0.74	1.64
13	4.04	0.32	2.45
14	3.85	0.43	2.11
15	4.02	1.46	3.00
16	2.86	0.19	0.82
17	4.03	1.06	1.86
18	3.02	0.74	1.57
19	4.39	1.25	2.68
20	4.38	1.07	2.60

8.6.4 Calibration

To estimate the camera rotation with respect to the ground plane, three images of a bus driving through the intersection were used. Figure 8.9 shows the images used, while the result of the reconstruction can be seen in Figure 8.10.

Now it is possible to create a map of the intersection by rectifying an image using the estimated ground plane. This map was used to produce the images in Figures 8.7 and 8.8. The accuracy of this estimation should give us a rough quality measure.

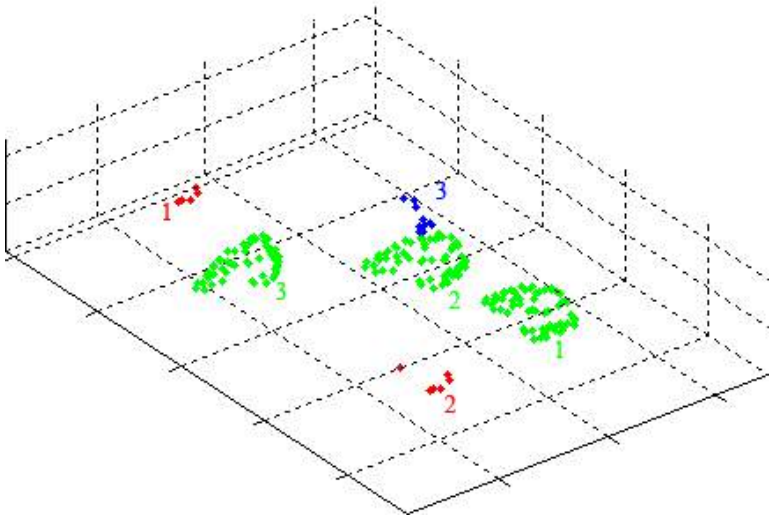


Figure 8.6: 3D reconstruction and pose for three frames in the video. Different colors represent different vehicles.

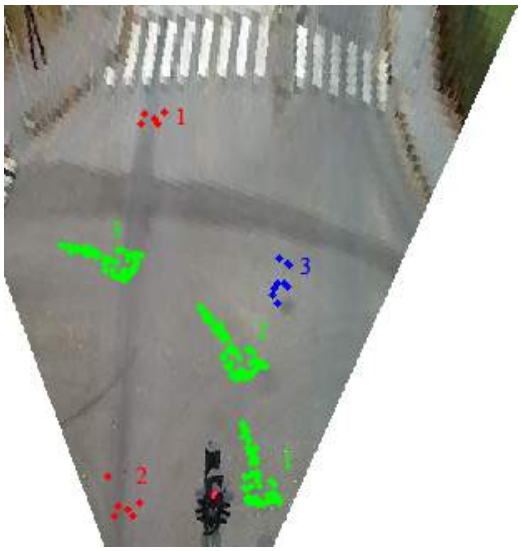


Figure 8.7: Projection of the 3D points into the ground plane to show the position of the vehicles. The positions for three different time points are shown. Like before different colors represent different vehicles.

By comparing distance in the rectified image and a real map of the intersection, we can estimate the size of the bus. The height was estimated to 2.65 m, the width to 2.15 m and the length to 12.7 m. The bus is 2.8 m high, 2.3 m wide and 12 m long, giving us an average error of 0.3 m. That is significantly

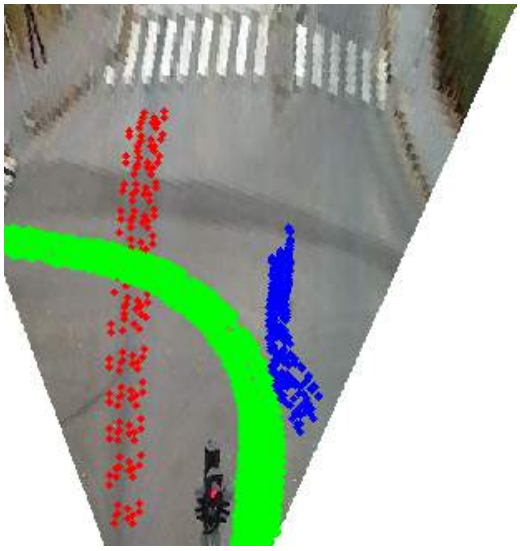


Figure 8.8: Same as Figure 8.7 but the positions for all time points are shown. Points above the crosswalk are removed during the tracking algorithm.

Figure 8.9: Three images of a bus driving through the intersection. The yellow dots mark the image points which were used for the reconstruction and the yellow lines show the boundaries of two sides of the bus.

better than the approach in [53] of simply projecting a segmented object onto the ground which in this case would give an error of several meters.

8.7 Conclusions

We presented some ideas on how to achieve accurate 3D reconstructions for traffic surveillance. The approach builds on recent research in optimal methods for computer vision. This makes it possible to fully exploit restrictions such as the ground being planar. The approach requires only one camera, which means that no synchronization between cameras is needed. Moreover, the camera position with

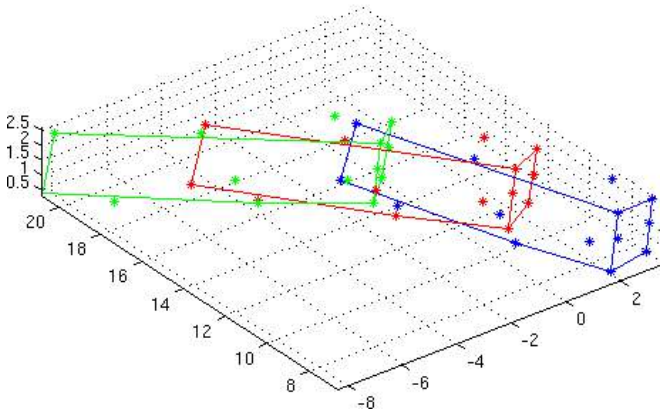


Figure 8.10: Reconstruction of the bus driving through the intersection at three time points. The tracked points used for the reconstruction are marked as well as some lines used for visualization. The blue bus corresponds to the first time point, the red one to the second and the green to the third.

respect to the ground can be estimated automatically as described in Section 8.5. Altogether the system is accurate, yet easy to set up and these are characteristics that should be attractive to many traffic scientists.

References

[1] Yali Amit and Donald Geman. Shape quantization and recognition with randomized trees. *Neural computation*, 9(7):1545–1588, 1997. 72

[2] Anon. SS-EN 12697-11:2012 bituminous mixtures – test methods for hot mix asphalt – part 11: Determination of the affinity between aggregate and bitumen, 2012. 23

[3] Itamar Arel, Derek C Rose, and Thomas P Karnowski. Deep machine learning-a new frontier in artificial intelligence research [research frontier]. *Computational Intelligence Magazine, IEEE*, 5(4):13–18, 2010. 63, 88

[4] Kalle Åström and Anders Heyden. Stochastic modelling and analysis of sub-pixel edge detection. In *International Conference on Pattern Recognition*, 1996. 29

[5] Kalle Åström and Anders Heyden. Stochastic analysis of image acquisition, interpolation and scale-space smoothing. *Advances in Applied Probability*, 31(4):855–894, 1999. 29

[6] Hossein Azizpour, Ali Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. Factors of transferability for a generic convnet representation. 2014. 88

[7] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006. 99

[8] Biswajit Bose and Eric Grimson. Ground plane rectification by tracking moving objects. In *IEEE International Workshop on Visual Surveillance and PETS*, 2004. 108

[9] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992. 68

[10] Yuri Boykov and Gareth Funka-Lea. Graph cuts and efficient nd image segmentation. *International Journal of Computer Vision*, 70(2):109–131, 2006. 40

[11] Yuri Boykov and Vladimir Kolmogorov. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222–1239, 2001. 11, 40

[12] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1124 –1137, sept. 2004. 11, 40

[13] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996. 72, 76

[14] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 63, 72, 77, 91

[15] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984. 74

[16] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990. 85

[17] Leonardo Bruno, Giuseppe Parla, and Clara Celauro. Image analysis for detecting aggregate gradation in asphalt mixture from planar images. *Construction and Building Materials*, 28(1):21–30, 2012. 49

[18] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998. 66

[19] Richard G. Casey and Chentung Robert Jih. A processor-based ocr system. *IBM Journal of Research and Development*, 27(4):386–399, 1983. 72

[20] Benjamin Coifman, Benjamin Coifman Corresponding, Philip Mclauchlan, and Jitendra Malik. A real-time computer vision system for vehicle tracking and traffic surveillance, 1998. 108

[21] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 63, 66, 67, 91

[22] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013. 85

[23] Brett Delahunt, Rose J Miller, John R Srigley, Andrew J Evans, and Hemamali Samaratunga. Gleason grading: past, present and future. *Histopathology*, 60(1):75–86, 2012. 87

[24] Thomas G Dietterich and Eun Bae Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Technical report, Department of Computer Science, Oregon State University, 1995. 76

[25] Scott Doyle, Michael D Feldman, Natalie Shih, John Tomaszewski, and Anant Madabhushi. Cascaded discrimination of normal, abnormal, and confounder classes in histopathology: Gleason grading of prostate cancer. *BMC bioinformatics*, 13(1):282, 2012. 87

[26] Jonathan I Epstein, William C Allsbrook Jr, Mahul B Amin, Lars L Egevad, ISUP Grading Committee, et al. The 2005 international society of urological pathology (isup) consensus conference on gleason grading of prostatic carcinoma. *The American journal of surgical pathology*, 29(9):1228–1242, 2005. 87

[27] Jonathan I Epstein, Michael J Zelefsky, Daniel D Sjoberg, Joel B Nelson, Lars Egevad, Cristina Magi-Galluzzi, Andrew J Vickers, Anil V Parwani, Victor E Reuter, Samson W Fine, et al. A contemporary prostate cancer grading system: a validated alternative to the gleason score. *European urology*, 2015. 87

[28] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981. 29

[29] Saul B Gelfand, CS Ravishankar, and Edward I Delp. An iterative growing and pruning algorithm for classification tree design. In *Systems, Man and Cybernetics, 1989. Conference Proceedings., IEEE International Conference on*, pages 818–823. IEEE, 1989. 74

[30] Lena Gorelick, Olga Veksler, Mena Gaed, Jairo Alejandro Gomez, Madeleine Moussa, Glenn Bauman, Aaron Fenster, and Aaron D Ward. Prostate histopathology: learning tissue component histograms for cancer detection and classification. *Medical Imaging, IEEE Transactions on*, 32(10):1804–1818, 2013. 87

[31] Robert M Haralick, Stanley R Sternberg, and Xinhua Zhuang. Image analysis using mathematical morphology. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (4):532–550, 1987. 7

[32] Richard Hartley and Fredrik Kahl. Global optimization through rotation space search. *International Journal of Computer Vision*, 82(1):64–79, 2009. 118

[33] Richard Hartley and Peter Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157, 1997. 114

[34] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2003. 102

[35] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 85

[36] Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995. 72, 76

[37] Tin Kam Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844, 1998. 76

[38] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, Mar 2002. 91

[39] Fu Jie Huang and Yann LeCun. Large-scale learning with svm and convolutional for generic object categorization. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 284–291. IEEE, 2006. 88

[40] Christer Hydén. *The development of a method for traffic safety evaluation: The Swedish Traffic Conflicts Technique*. PhD thesis, Department of Traffic Planning and Engineering, Lund University, 1987. 107

[41] Joseph G Jacobs, Eleftheria Panagiotaki, and Daniel C Alexander. Gleason grading of prostate tumours with max-margin conditional random fields. In *Machine Learning in Medical Imaging*, pages 85–92. Springer, 2014. 87

[42] Fredrik Kahl and Richard Hartley. Multiple-view geometry under the L_∞-norm. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(9):1603–1617, 2008. 109

[43] Hanna. Källén. 3d reconstruction for traffic surveillance. Master's thesis, Dept. of Mathematics, Lund University, Sweden, 2009. 118

[44] Hanna Källén, Håkan Ardö, and Olof Enqvist. Tracking and reconstruction of vehicles for accurate position estimation. In *Applications of Computer Vision (WACV), 2011 IEEE Workshop on*, pages 110–117. IEEE, 2011. 107

[45] Hanna Källén, Anders Heyden, Kalle Åström, and Per Lindh. Measurement of bitumen coverage of stones for road building, based on digital image analysis. In *Applications of Computer Vision (WACV), 2012 IEEE Workshop on*, pages 337–344. IEEE, 2012. 25, 26

[46] Hanna Källén, Anders Heyden, Kalle Åström, and Per Lindh. Measuring and evaluating bitumen coverage of stones using two different digital image analysis methods. *Measurement*, 84:56 – 67, 2016. 26

[47] Hanna Källén, Anders Heyden, and Per Lindh. Measuring bitumen coverage of stones using a turntable and specular reflections. In *8th International Conference on Computer Vision Theory and Applications (VISAPP 2013)*, pages 333–337. SciTePress, 2013. 25, 26

[48] Hanna Källén, Anders Heyden, and Per Lindh. Estimation of grain size in asphalt samples using digital image analysis. In *SPIE Optical Engineering+ Applications*, pages 921714–921714. International Society for Optics and Photonics, 2014. 49

[49] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(2):147–159, 2004. 11, 40

[50] David Kriesel. *A Brief Introduction to Neural Networks*. 2007. available at http://www.dkriesel.com. 78, 81

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 85

[52] Riccardo Lamperti, Claudio Lantieri, Cesare Sangiorgi, Gabriele Bitelli, and Andrea Simone. Semi-automatic evaluation of the degree of bitumen coverage on bitumen-coated aggregates. In *8th RILEM International Symposium on Testing and Characterization of Sustainable and Innovative Bituminous Materials*, pages 15–24. Springer, 2016. 25

[53] Aliaksei Laureshyn and Håkan Ardö. Automated video analysis as a tool for analysing road user behaviour. In *ITS World Congress*, London, UK, 2006. 107, 125

[54] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 78

[55] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 78, 88

[56] Bastian Leibe, Konrad Schindler, Nico Cornelis, and Luc Van Gool. Coupled object detection and tracking from static cameras and moving vehicles. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(10):1683–1698, 2008. 107

[57] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011. 99

[58] Giuseppe Lippolis. *Image analysis of prostate cancer tissue biomarkers*. PhD thesis, Lund University, 2015. 87, 88

[59] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999. 99

[60] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981. 103

[61] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967. 39, 63

[62] Jean-Michel Marin, Kerrie Mengersen, and Christian P Robert. Bayesian modelling and inference on mixtures of distributions. *Handbook of statistics*, 25(16):459–507, 2005. 25

[63] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. 78

[64] Filippo Merusi, Allessandro Caruso, Riccardo Roncella, and Felice Giuliani. Moisture susceptibility and stripping resistance of asphalt mixtures modified with different synthetic waxes. *Transportation Research Record: Journal of the Transportation Research Board*, 2180(-1):110–120, 2010. 24

[65] Karsten Müller, Aljoscha Smolic, Michael Drose, Patrick Voigt, and Thomas Wiegand. 3-d reconstruction of a dynamic environment with a fully calibrated background for traffic scenes. *IEEE Trans. Circuits Syst. Video Techn.*, 15(4):538–549, 2005. 108

[66] Christian Mulsow and Lars Marschke. Multidirektionale reflexionsanalyse zur bestimmung des umhüllungsgrades von bitumenumhüllten gesteinskörpern. *Strasse und Verkehr*, 98(11):27, 2011. 25

[67] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012. 78

[68] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010. 85

[69] Carl Olsson, Anders Eriksson, and Richard Hartley. Outlier removal using duality. In *CVPR 2010*, 2010. 111

[70] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, pages 1065–1076, 1962. 32

[71] Marc Aurelio Ranzato, Fu Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007. 84

[72] Lior Rokach and Oded Maimon. Top-down induction of decision trees classifiers-a survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 35(4):476–487, 2005. 75, 76

[73] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957. 78

[74] Murray Rosenblatt et al. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956. 32

[75] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1508–1515. IEEE, 2005. 99

[76] Elisabeth Rouy and Agnès Tourin. A viscosity solutions approach to shape-from-shading. *SIAM Journal on Numerical Analysis*, 29(3):867–884, 1992. 17

[77] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks–ICANN 2010*, pages 92–101. Springer, 2010. 84

[78] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR 2014)*, page 16. CBLS, 2013. 89

[79] James A Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences of the United States of America*, 93(4):pp. 1591–1595, 1996. 14

[80] James A Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 1999. 14

[81] Seymour Shlien. Multiple binary decision tree classifiers. *Pattern Recognition*, 23(7):757–763, 1990. 72

[82] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Computer Science Department, Pittsburgh, PA, April 1991. 103, 104, 108, 114

[83] Morteza Vadood, Majid Safar Johari, and Ali Reza Rahaei. Introducing a simple method to determine aggregate gradation of hot mix asphalt using image processing. *International Journal of Pavement Engineering*, 0(0):1–9, 0. 50

[84] Weixing Wang. Image analysis of particles by modified ferret method – best-fit rectangle. *Powder Technology*, 165(1):1 – 10, 2006. 53

[85] Frohmut Wellner, Sascha Kayser, Lars Marschke, Dmitrij Schlesinger, Alexander Morgenstern, and Christoph Schulze. Optimierung der affinitätsprüfung: Verbesserung der präzision der prüfung zur bestimmung des haftverhaltens zwischen groben gesteinskörnern und bitumen. *Forschung Strassenbau und Strassenverkehrstechnik*, (1066), 2011. 24

[86] Paul J Werbos. *Beyond regression: new tools for prediction and analysis in the behavioral sciences*. PhD thesis, 1974. 78, 81

[87] Paul J Werbos. Maximizing long-term gas industry profits in two minutes in lotus using neural network methods. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(2):315–333, 1989. 81

[88] Zhaoxiang Zhang, Min Li, Kaigi Huang, and Tieniu Tan. Practical camera auto-calibration based on object appearance and motion for traffic scene visual surveillance. *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008. 108

Populärvetenskaplig sammanfattning

Hur kan vi använda oss av bildanalys för att förbättra våra vägar? Kan vi genom bildanalys hjälpa patologerna att hitta avvikelser i biopsier? Går det att mäta hur farlig en korsning är genom att bevaka den med övervakningskameror? Detta är tre frågor som den här avhandlingen försöker svara på. Genom att använda oss av matematiska algoritmer kan vi analysera bilder för att lösa just de här problemen

Del 1 - Bildanalys för asfaltslaboratorier
När man bygger vägar vill man såklart att vägen ska hålla så länge som möjligt och där spelar kvaliteten på asfalten en stor roll. Asfalt består av stenar av olika storlek och av ett bindemedel som ska hålla ihop stenarna. Detta bindemedel kallas bitumen. För att inte stenarna ska lossna från varandra är det viktigt att vidhäftningen mellan bituminet och stenarna är så bra som möjligt. Detta undersöks med den så kallade rullflaskmetoden. I rullflaskmetoden lägger man stenar som man har täckt med bitumen i en flaska med vatten och låter den snurra på ett rullbord. Detta gör att en del av bituminet lossnar och efter en stund plockar man ut stenarna från flaska och försöker uppskatta hur mycket av stenen som fortfarande är täckt av bitumen, alltså täckningsgraden. Detta är väldigt svårt att göra manuellt, därför har vi försökt att använda oss av bildanalys för att göra denna uppskattning.

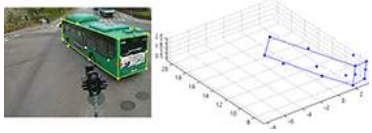
En sten delvis täckt av bitumen. Hur stor del är täckt?

Detta är ett förhållandevis lätt problem då färgen på stenen skiljer sig markant, men vad gör man om stenen är väldigt mörk och är väldigt likt bitumen? Det som ändå skiljer bitumen från sten är att bituminet reflekterar ljus på ett helt annat sätt en stenen. Dessa reflexer kan vi utnyttja för att bedöma var det finns bitumen. Genom att ta flera bilder med ljus från olika håll kan vi bedöma om det bildas reflexer, i så fall är en pixel mörk då ljuset inte reflekteras men väldigt ljus när det bildas en reflex.

En annan sak man undersöker på laboratoriet är om receptet för asfalten har följts då vägen tillverkades. Receptet säger hur mycket av varje storlek av stenarna man ska ta. För att undersöka detta kan man borra ett litet hål i vägen, du får man fram en liten asfaltspuck. Denna puck analyseras genom att man löser upp bituminet i metylenklorid, sedan siktas stenarna genom en serie spaltsiktar av olika storlek och andelen som fastnar vid varje sikt uppmäts. Tyvärr är metylenklorid väldigt giftigt och inte så miljövänligt och därför vill man helst inte använda det. Istället har vi sågat sönder puckarna och tittat på snitten för att uppskatta storleksfördelningen. Detta görs i ett par steg. Först identifieras stenarna i provet, vilket betyder att vi tar bort bakgrunden som är bitumen. Sedan anpassar vi rektanglar till alla stenar för att kunna bestämma bredden på stenarna. När vi vet bredden på alla stenar i provet kan vi enkelt beräkna storleksfördelningen.

Del 2 - Cancerdetektion
En annan fråga som tas upp i avhandlingen är om bildanalys kan hjälpa patologer att på ett effektivt sätt ställa diagnos på histopatologiska bilder. En mycket vanlig cancerform är prostatacancer, och en stor mängd prostatabiopsier behöver analyseras av en minskande mängd patologer. Det patologerna tittar på är växtmönstret på körtlarna i vävnadsprovet. Beroende på hur växtmönstret ser ut graderas provet på den så kallade Gleasonskalan. Man försöker också uppskatta hur stor del av provet som består av de olika Gleasongraderna. I den här avhandlingen har vi tittat på ett delproblem och försökt avgöra automatiskt vilken Gleasongrad ett litet urklipp av bilden har. Detta har vi gjort genom att plocka ut väldigt generella egenskaper i bilderna, som till exempel kanter i bilden. Dessa egenskaper samlas sedan ihop i en lista och genom att jämföra dessa listor med varandra kan vi avgöra vilken grad av cancer vi har. Än så länge kan vi inte göra detta lika bra som en patolog, men i framtiden är tanken att patologen skulle kunna få hjälp med att hitta områden som innehåller cancer och göra en förklassificering som sedan kan kontrolleras och godkännas av en patolog.

Del 3 - Trafikövervakning
Den sista delen av avhandlingen handlar om att följa bilar i en korsning. Genom att filma bilarna i korsningen med en statisk kamera får vi många tvådimensionella bilder av bilarna. Dessa bilder kan vi sedan använda för att göra tredimensionella rekonstruktioner av fordonen. Detta görs genom att vi plockar ut intressanta punkter i en bild, det kan vara hörn i bilden eller liknande, dessa punkter följs sedan till nästa bild. Genom att analysera hur punkterna flyttat sig mellan bilderna kan vi bestämma var i rummet punkterna befinner sig. Detta är sedan tänkt att användas för att beräkna var i korsningen fordonet befinner sig vid varje tidpunkt och om bilar kommer farligt nära varandra.



En bild på en buss och motsvarande 3d-modell