Towards Robust Illumination Invariant Tracking of Vehicles using Video Cameras

ROBERT OLOFSSON

Master’s Degree Project
Stockholm, Sweden 2005

TRITA-NA-E05088
Towards Robust Illumination Invariant Tracking of Vehicles using Video Cameras

ROBERT OLOFSSON

TRITA-NA-E05088

Master’s Thesis in Computer Science (20 credits)
at the School of Electrical Engineering,
Royal Institute of Technology year 2005
Supervisor at Nada was Henrik Christensen
Examiner was Henrik Christensen
Existing techniques are used to create a robust illumination invariant system for measuring traffic parameters on car ferries, using video cameras. Parameters measured are vehicle lengths and number of vehicles. Background estimation is performed in RGB-color space using cylindrical Gaussian mixture models at pixel-level. Foreground pixels are identified using this model and grouped into ellipses. Ellipses are associated to vehicle models and their future positions are predicted into subsequent frames. Stationary objects are identified and incorporated into the background model. Lengths are measured in a metrically rectified image. Partial illumination invariance is achieved by identifying unexplained foreground pixels (i.e. pixels not predicted to be foreground). Shadows and highlights are removed using a computational color model. A final system was implemented in C++ and shows real-time performance. Tracking in night and daylight conditions are illustrated.

Mot robust belysningsinvariant följning av fordon via videokameror

Examensarbete

Acknowledgments

First of all I would like to thank my supervisor and examiner Professor Henrik I. Christensen for giving me the opportunity to do my masters project at the Center for Autonomous Systems at the Royal Institute of Technology. It has been a very interesting project although it sometimes felt like I did not make any progress. This project was performed in collaboration with Vägverket-Färjeredet, Vaxholm, Sweden. During the course of the project I spent several days on the ferries in Vaxholm and I would like to thank Roger Nyman, Joachim Ekenberg and Karl-Anders Rönning working there for encouraging me and giving me feedback during the project. I especially appreciate the trips out to Vaxholm, which were a welcome break in my research. I would also like to thank Johan Algell, Marketing Manager at Färjeredet for showing interest in the project.

This master thesis is a part of my M. Sc. studies in Electrical Engineering at the Royal Institute of Technology. It has been hard but inspiring work during these years. My family deserves a big thank you, for everything. Last but not least I send my thanks to my dear friends Mathias, Patrik, Erik and Peter for making the years at KTH so much fun. Thank you!
In memory of my father.
## Contents

1 Introduction ................................. 1
   1.1 The problem ................................. 1
   1.2 Possible solutions ........................... 1
       1.2.1 Induction coils .......................... 2
       1.2.2 Pressure sensors .......................... 2
       1.2.3 Radar .................................. 2
       1.2.4 Laser-scanner ............................ 2
       1.2.5 Computer vision ............................ 2
       1.2.6 Tripwires ................................ 3
   1.3 Problem definition .......................... 3
   1.4 Limitations of the project .................. 3
   1.5 Overview of the pipeline .................... 4
   1.6 Outline of thesis ........................... 8

2 Survey of techniques considered for implementation 11
   2.1 Survey of general vehicle tracking approaches ........... 11
       2.1.1 3D model based tracking ................. 12
       2.1.2 Region-based tracking ...................... 12
       2.1.3 Active contour based tracking .............. 12
       2.1.4 Feature based tracking ..................... 13
       2.1.5 Motivation ................................. 13
   2.2 Background estimation ........................ 14
       2.2.1 Survey of techniques ....................... 15
   2.3 Illumination and shadows ..................... 19
       2.3.1 Survey .................................. 19
   2.4 Pixel association ............................. 21
       2.4.1 Survey .................................. 21
   2.5 Tracking and data association ................. 22
       2.5.1 Nearest neighbor .......................... 22
       2.5.2 Kalman filter(s) ......................... 23
3 Detailed design

3.1 Background estimation

3.1.1 Motivation

3.1.2 Illumination invariant per-pixel mixture model

3.1.3 Improved background mixture model

3.1.4 Discussion

3.2 Illumination and shadows

3.2.1 Motivation

3.2.2 Computational color model

3.2.3 Discussion

3.3 Post-processing of segmentation mask

3.3.1 Motivation

3.3.2 Erosion

3.3.3 Dilation

3.3.4 Discussion

3.4 Region processing

3.4.1 Motivation

3.4.2 Connected component analysis

3.4.3 Ellipse fitting

3.4.4 Ellipse parameters

3.4.5 Two-pass algorithm

3.4.6 Discussion

3.5 Tracking

3.5.1 Motivation

3.5.2 Stationary ellipses

3.5.3 Matching ellipses to vehicle models

3.5.4 Final matching

3.5.5 Updating matched vehicle models

3.5.6 Spawn new vehicle models

3.5.7 Update Kalman filters

3.5.8 Length measurements

3.5.9 Discussion

3.6 Kalman filter-based tracking

3.6.1 Trajectory coordinate system

3.6.2 State space model

3.6.3 Initialization of the filter

3.6.4 Time and measurement update

3.6.5 Discussion

3.7 Metric rectification

3.7.1 From projective to affine

3.7.2 From affine to metric
3.7.3 Metric rectification ................................. 58
3.7.4 Implementation details ............................... 59

4 Workflow and implementation details 61
  4.1 Workflow ........................................ 61
  4.2 Program flow ...................................... 62
  4.3 Equipment and data ............................... 62

5 Conclusion and evaluation 66
  5.1 System performance ................................. 68

References 71

A Point-ellipse distance 75
  A.1 Aligning semi-major axis with y-axis ............... 75
  A.2 Point-ellipse distance .............................. 76
  A.3 Newton’s method .................................. 77
  A.4 Ellipse-ellipse distance ............................ 77

B Minimization of $\beta_t$ 79

C Derivation of $t$ and $d$ 80

D Illustrations of tracking performance 82
List of Figures

1.1 Field of view, stern and bow ........................................ 4
1.2 The four video sequences. ............................................. 5
1.3 Illustration of GUI and pipeline. ..................................... 6
1.4 Frames while being processed by pipeline. ........................ 9
1.5 Tracked objects. ...................................................... 10

2.1 Illustration of the problem of matching ellipses (or connected components) between frames. ........................................ 23
2.2 Ellipse perimeter distance ............................................. 24
2.3 Illustration of the proposed tracking algorithm. .................... 26

3.1 Spherical and cylindrical mixture models. .......................... 30
3.2 Background pixel values. ............................................. 31
3.3 Computational color model. .......................................... 34
3.4 Frame with shadow. .................................................. 36
3.5 Frame with highlight. ................................................ 37
3.6 Foreground segmentation and connected component ellipse. ..... 39
3.7 Connected component ellipse. ....................................... 42
3.8 Starting area, trajectory line, edge distances. ....................... 48
3.9 Length measurement. ................................................ 50
3.10 Trajectory coordinate system. ....................................... 51
3.11 Kalman filter behavior. ............................................. 54
3.12 Measured lengths. .................................................. 56
3.13 Constraint lines. ................................................... 59
3.14 Rectified image and lengths. ....................................... 60

4.1 Illustration of the program flow in the pipeline. .................... 63

5.1 Activated AGC ..................................................... 67
5.2 Plot of execution time. .............................................. 69
5.3 Histogram of execution time. ....................................... 70
A.1 Ellipse and axis-oriented ellipse ............................... 75
D.1 Tracking in overcast sequence. ................................. 83
D.2 Tracking in overcast sequence continued. ......................... 84
D.3 Tracking in night sequence. ................................. 85
D.4 Tracking in night sequence continued. ......................... 86
Chapter 1

Introduction

1.1 The problem

The purpose of this master thesis project was to investigate if traffic parameters on car ferries could be measured with a camera based surveillance system. It was carried out in collaboration with Vägverket-Färjeredet in Vaxholm. Currently all traffic parameters are measured by hand by the crew onboard the ferry. This can be a labor intensive and uninspiring work prone to errors and a safety hazard. Usually the commander of the ferry measures the traffic parameters and this will distract his or her attention and is a safety risk. A fully automatic system would have several advantages. The traffic parameters that are to be measured is:

- the number of vehicles
- the lengths of vehicles in the following classes: less than 6 m, 6-12 m and 12-24 m.

The ferries has to be equipped with some sort of automatic traffic measurement system. Reliability of manual traffic measurement i.e visual inspection is not up to the task and does not give adequate statistics. At best the vehicles are counted, accurate classification of vehicles into length-classes is not performed. An automatic traffic measurement system would be able collect and send data in realtime to head-quarters. Additionally, the statistics could be automatically collected and presented on an hourly, daily, weekly, monthly, yearly or seasonally basis thus giving more information when allocating resources like ferries and staff.

1.2 Possible solutions

To perform the measurement automatically a number of different systems could be considered. Some of them are:

- Induction coils
- Pressure sensors
• Radar
• Laser-scanner
• Computer vision
• Tripwires

1.2.1 Induction coils

This is the same method that is used for controlling traffic signs, passing cars are sensed by the principle of induction. It is a robust method in terms of not being affected by weather conditions and it is an available solution. This method is probably not so good at distinguishing different cars and measuring their lengths which makes it less useful for this particular application.

1.2.2 Pressure sensors

This solution involves having pressure sensors on the road-surface and identifying vehicles as the pass over the pressure sensors. This method also has problems with length estimation.

1.2.3 Radar

This is a solution that Färjerederiet has tried without any success. It was based on the same principle as automatic speed signs which uses a radar to determine the speed of vehicles. Vehicles had to be loaded onto the ferry one at a time for the system to function and they were going to slow for the system to detect them, so this solution was not feasible.

1.2.4 Laser-scanner

A laser-scanner could be used for measuring the number of axis on vehicles or estimating their length. One problem with the laser-scanner method is that snow and rain will cause problems, bad weather conditions will hinder the line-of-sight of the scanner.

1.2.5 Computer vision

This is the method that was finally chosen. The idea is to mount video cameras at the bow and stern of each ferry. The video streams are then fed into a computer that calculates the number of vehicles and their lengths. The equipment and maintenance cost for this method is relatively low. The main drawback is that the system has to be robust to illumination changes and different weather conditions. Length estimation and counting vehicles is feasible.
1.2.6 Tripwires

Tripwires consists of cables placed on the road surface which registers vehicle tyres passing over them. This solution is cheap but does not meet the accuracy requirement and length estimation is impossible.

1.3 Problem definition

The main purpose of this master thesis project was to evaluate and implement a solution to the traffic measurement problem. To solve this problem a solution based on computer vision was chosen, partly because it was the most promising technology. It is also relatively cheap when it comes to infrastructure, all that is needed is a couple of cameras. Onboard the ferry there were already two surveillance cameras fitted so no additional equipment had to be installed. The cameras where placed as shown in figure 1.1. In this figure the field of view is illustrated, as can be seen in these figures two cameras are mounted on the ferry. One in the bow and one in the stern. The angle of the camera gives a good compromise between an overhead view for minimizing occlusion, and a lower view for length estimation. Note that tall vehicles entering the ferry close to the camera might occlude vehicles further away.

A solution based on computer vision suffers from problems caused by different illumination and weather conditions. For a computer vision based system to function 24 hours a day it has to handle these problems in some way. A system is no good if you have to turn it off during night and bad weather. Vägverket never gave any good figures on the required accuracy of the system but a vehicle counting performance of 90 percent is probably acceptable, particular compared to the accuracy of the current system. Accuracy of the length estimation was never discussed.

1.4 Limitations of the project

This project was to be a feasibility study of a computer vision based traffic measurement system, resulting in a prototyp system implemented in C++. Because of the limited time-frame the system was developed and tested on a limited number of test sequences. These sequences were selected to represent the most common illumination conditions. These sequences were:

**Daylight** Captured during daylight condition, with shadows.

**Dusk** Captured during twilight condition in dim illumination.

**Overcast** Captured during overcast condition, without shadows.

**Night** Captured during night condition, poor lighting.

the sequences are shown in figure 1.2 on page 5. The system was developed and tweaked to perform with satisfaction on these four sequences. Initially the idea was
Figure 1.1. Illustration of the approximate field of view of the cameras.

to make a realtime system (i.e on-line) but this would require more time spent on optimization of the actual code and this was not really interesting as long as the system was not functioning satisfactorily. So instead the system was designed as an off-line system which means that video sequences are captured when vehicles are loaded onto the ferry and the processing is done afterwards. Removing the realtime requirement made it possible to spend more time on making a robust system.

1.5 Overview of the pipeline

The final implemented system consists of two parts: a graphical user interface (gui) and a processing pipeline. The two parts of the system are implemented as two separate threads in the actual program. The gui handles all input from the user.
Figure 1.2. The four different video sequences used in this project. Every ninth frame is shown.
At the moment this is just starting and stopping of the processing pipeline. All video frame processing and tracking is done in the processing pipeline, see thread two in Figure 1.3. In this pipeline everything related to machine vision is executed. For the pipeline to work it has to know on which side the camera is located. There are two possible positions: to the right as in Figure 1.2(b) on the preceding page and to the left as in Figure 1.2(c) on the page before, the theory in this thesis is described as if a camera at the right side is used, the left side case is analogous. The pipeline is controlled by a pipelineManager which controls the flow of data between the different parts of the pipeline. Before each new loop is executed the pipelineManager requests a frame from the sequenceProcessor and sends it to the background estimator. Each step in the pipeline creates data that should be sent further down the pipeline, all this is handled by the pipelineManager. Different parts of the pipeline only sees the data it needs to see to complete its task, this makes the pipeline highly modifiable. The pipeline can be divided into five major parts:

1: **Background estimation** Background estimation.

2: **Post-processing** Erosion and dilation.

3: **Region processing** Two steps of connected component analysis and ellipse fitting.

![Figure 1.3. Illustration of GUI and pipeline.](image)
4: **Tracking** Tracking objects, measuring lengths, updating vehicle models, checking for stationary ellipses.

5: **Adjust update mask** Predict next frame and change update rate for background estimation.

In the background estimation module new frames are incorporated into the background model and at the same time a foreground segmentation mask is created, see Figure 1.4(b). This mask indicates if pixels are considered as foreground, shadowed background, highlighted background, or background.

The next step is to post-process the foreground segmentation to remove small regions and isolated pixels, this is done using an erosion step, see Figure 1.4(c). After the erosion, regions close to each other are connected by performing one pass of dilation, see Figure 1.4(d). Connecting regions with dilation is necessary since the foreground segmentation is usually very fragmented and this step helps to reduce the fragmentation.

Now the foreground mask contains mostly large regions corresponding to moving objects in the scene. Each object will usually consist of one, two or three regions in the mask after this step. The pixels in the foreground mask are now grouped into meaningful entities, in this case connected components which later is converted to (i.e represented by) ellipses, see Figure 1.4(e) and Figure 3.7 on page 42. Working with individual pixels can be a cumbersome task. Ellipses are more manageable particularly when tracking and counting objects, the outline of the ellipses has been drawn in light-grey in Figure 1.4(e) for clarity.

Ellipses are then associated to vehicle models or hypotheses, see Figure 1.5. Large ellipses in the starting area spawns new vehicle models if they are not associated to an existing model. Models that are supported by ellipses for several frames are believed to be true vehicles and they are counted and their length is estimated. Kalman filter-based trackers are used to predict the position of the vehicle in the next frame to aid the matching of ellipses to models. Prediction of vehicle positions also helps in checking for unexplained pixels from the foreground segmentation, see Figure 1.4(f). Unexplained pixels can arise due to sudden illumination changes, if numerous foreground pixels appears at positions where no vehicle is predicted to be, then this information is used to incorporate these pixels into the background model. Stationary ellipses that stay at the same place for several frames are identified and their underlying pixels are forced to be incorporated into the background.

The update rate in the background model is increased for pixels underneath stationary ellipses and for unexplained pixels. These new update rates are then used in the next loop of the processing pipeline. This makes the system adaptive to illumination changes and quickly incorporates stationary objects into the background model.
1.6 Outline of thesis

This thesis is divided into three major sections. A survey of different methods considered for implementation is described in the Survey chapter 2, strengths and weaknesses of these methods are discussed. In the Detailed design chapter 3 the chosen methods are more thoroughly described and a motivation for using each particular method is given at the end of each section. The thesis ends with a chapter discussing the work flow of the project and some things about the design of the system. Finally the system is evaluated and some conclusions drawn from the project are given.
Figure 1.4. Frames after different stages in the processing pipeline. In the foreground segmentation white is foreground, black is background, dark grey is shadowed background and light grey is highlighted background.
Figure 1.5. Frame showing information about tracking. Vehicle models whose length has been estimated are drawn with a colored ellipse.
Chapter 2

Survey of techniques considered for implementation

In this chapter the different techniques considered for implementation in this system will be described and discussed. As will be described in the first section the choice was made to use a region-based tracker. Regions can be extracted based on texture, color, motion or a combination of these properties. In this system regions are identified by comparing pixels to a background model. Pixels that do not seem to belong to the background are grouped into regions which then can be tracked and associated to vehicles. Different ways of maintaining a background model or estimate will be described. At the end some techniques for tracking are discussed.

2.1 Survey of general vehicle tracking approaches

This section describes various methods for tracking vehicles and discusses strengths and weaknesses of them. Vehicle tracking is an ongoing research area and the focus has been on tracking vehicles at freeways for measuring traffic parameters [eBMM98a, KM03, ea97, CBMM98b, DCP00, TTLS03, BHD96, CSZ01, KN96, Mag02]. Interesting parameters might be the number of vehicles traveling down a lane, the average speed, the type of vehicles (e.g. trucks, suvs and vans) or accidents. Some system employ higher level reasoning to identify accidents, stalled vehicles and driver behavior. As pointed out in [ea97] vehicle tracking falls in one of the following four categories

1. 3D Model based tracking
2. Region-based tracking
3. Active contour based tracking
4. Feature based tracking

with region-based tracking being the most common method.
2.1.1 3D model based tracking

Three-dimensional model based tracking utilizes wire-frame models of vehicles to perform tracking. In [KM03] extracted edges from the image are fitted to a general 3D-model of a vehicle. One problem with model-based trackers is that a model for all types of vehicles is not available. [KM03] solves this by using a general model that describes the overall appearance of vehicles. The advantage of this system is that it does not fail when objects are stationary so parked vehicles can be identified.

2.1.2 Region-based tracking

This is the approach chosen for this system. Moving regions in the image are identified and tracked. Foreground segmentation is used for creating moving regions and these are then tracked. Because of bad foreground segmentation and partial occlusion, higher-level reasoning has to be employed to aid the tracker when objects are badly segmented. Major problems with region-based trackers are:

- Vehicles that stop.
- Slow-moving vehicles.
- Occlusion.
- Shadows.
- Slow-moving traffic.

All these problems are addressed in this system except for the slow-moving traffic. Slow-moving traffic will not be a problem for this system. A region-based tracker that has shown excellent results is the one described by Kamijo, Ikeuchi and Sakauchi in [SKM01]. They employ block-matching between frames to estimate motion vectors and a Spatio-Temporal Markov Random Field is used to optimize the motion vector labels. This results in a very robust system which can track vehicles at a very low camera angle.

2.1.3 Active contour based tracking

Vehicles have a prominent contour and this can be exploited for tracking. Tracking contours can be seen as a parameterized version of region tracking where instead of tracking regions the contour is tracked. Usually the contour is described by B-Splines [TTL03] or active snakes. The contour model is continuously updated from frame to frame so that it adapts to new conditions. Contour based trackers are able to handle partial occlusions if the contour has been initialized to an occlusion free vehicle, this is their main advantage. Vehicles often enter the field of view at a predetermined point and thus initialization of the contour is relatively easy.
2.1.4 Feature based tracking

Instead of tracking whole objects like the previous three approaches, features can be tracked and grouped [BMCM97]. Example of features are edges and lines. Extracted edges are grouped based on their motion. Edges corresponding to rigid body motion are assigned to the same vehicle. The problem is to find salient features and to group them, but if this can be done this tracking becomes robust to partial occlusion. Depending on what features are chosen the tracking can also be made robust to illumination changes.

2.1.5 Motivation

Feature based and model based tracking is made harder with the camera located at a low angle as the case is for this system. Vehicles in the lane closest to the camera are imaged very different from vehicles in the lane farthest away from the camera. Vehicles entering the scene close to the camera are imaged from behind or from an overhead position while vehicles entering farthest away from the camera are imaged from a side view. A model based tracker would have to have a very general model of vehicles and at least one model for cars and one for trucks. This disqualifies the model-based tracker as long as the current camera position is used.

Feature based trackers are also disqualified for the same reasons. Feature based trackers will have problems with disappearing features. Consider a vehicle entering the scene at the lane closest to the camera, as the vehicle travels down the lane corner features at the hood of the vehicle will become occluded. The relative distance between features (e.g. corners) as vehicles drive down the lanes, will diminish, giving the appearance of a non-rigid object. If frames are acquired sufficiently fast these problems can be handled by adaptation between frames but a real-time system will almost surely have to drop several frames with a loss of track as a result. Another problem is the poor resolution of the images. To track features between frames there has to be enough resolution in the images so that salient features can be found. Some empirical studies where performed with feature based tracking and it was found that features were hard to match if the vehicle was not close to the camera. A simple edge map and correlation were used. Correlation was done by extracting a block of pixels from the edge map of frame \( n \) and then this block was correlated at various positions in frame \( n + 1 \).

Only two methods are left to consider. Their advantages and disadvantages are similar to each-other so there is no real winner. Contour based trackers suffer from problems with shadows and no good solution to this problem was found in the literature, another problem is that contours often merge as vehicles partially occlude each other. The region-based tracker on the other hand has a nice framework for handling shadows directly at the pixel level making it a natural choice.

2.2 Background estimation

Most computer vision applications used for tracking various objects, employ some sort of background estimation or background modeling to be able to identify moving objects. The reason for this is that the objects being tracked tend to be moving objects in-front of a static background. To be able to separate the moving object from the background we need to have some way of telling what constitutes background and what does not. Most background modeling techniques work by forming a statistical model of the background, and classify pixels as foreground if they cannot be explained by this model. Intensity based images, where each pixel is represented by a scalar value, has been the predominant format. But lately more and more techniques focusing on 2- and 3-dimensional pixel-values like HS- and RGB-color has appeared, probably due to the increase in computer power during the last years. The computing power required increases linearly with the dimension of the pixel-value so RGB-images requires roughly three times the computing power of intensity based images. The main motivation for using color is that it makes it easier to distinguish shadows and discriminating between background and foreground. If an object has the same intensity as the background but a different hue or color, a color-based background model stands a better chance of classifying the pixels correctly by utilizing the extra information in the chromaticity channel.

In [CK04] an overview of the most common techniques for background estimation and subtraction is given. The output from background subtraction algorithms is typically a foreground segmentation-mask:

\[ F_{ij} = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1N} \\ f_{21} & \ddots & \vdots \\ \vdots \\ f_{M1} & \cdots & f_{MN} \end{bmatrix} \quad f_{ij} = \begin{cases} 0 & \text{false background,} \\
1 & \text{true foreground} \end{cases} \]

where each element indicates if the pixel at position \((i,j)\) is classified as a foreground or background pixel.

Ideally, the segmentation-mask should agree with the human notion of what is considered as background. The background is almost always assumed to be visible all the time and only temporarily covered by a foreground-object (e.g. a vehicle or person). This means that the expected state of a pixel is to be background and only occasionally it will be foreground. Further, the pixel-value \(x\) of the background is thought to be relatively stable (i.e. the color or intensity if grey-scale images are used) does not change to much. This assumption will be violated for scenery containing illumination changes and must be handled in some way. Small variations in pixel-value will always be present due to noise or inter-reflections among objects in the scene and will have to be considered.

Background estimation and background maintenance systems are burdened with the following problems listed in [TKBM99]:

\[ \text{...} \]

\[ \text{...} \]
1. **Moved objects** When a object in the scene is moved to a new location it should not be considered foreground forever.

2. **Time of day** In outdoor scenes, as the day goes by, the illumination will gradually change and alter the appearance of the background.

3. **Light switch** Sudden changes in illumination and other scene parameters alter the appearance of the background i.e clouds passing by the sun and automatic-gain-control of cameras.

4. **Waving trees** Background can consist of two or more alternating pixel values, like rippling water and waving trees.

5. **Camouflage** A foreground object’s pixel values may be similar to the background making it hard to distinguish.

6. **Bootstrapping** In some environments a training period without foreground objects is not available (e.g a heavily trafficked street.)

7. **Foreground aperture** The interior pixels of large homogenously colored objects can be difficult to detect.

8. **Waking person** When an object initially moves in the background, both it and the newly revealed parts of the background appear to change (e.g. car in parking lot.)

9. **Shadows** Foreground objects tend to cast shadows which will be classified as foreground if they are not explicitly handled.

These are the major problems any background subtraction and maintenance algorithm has to address. Some of them can be handled at the pixel-level while others need to be addressed at region-level or frame-level. In this system mainly problems 2, 3, 4, 5, 7 and 9 will be present. Problem 4 appears during bad weather conditions like snow and rain. Problem 8 is ignored by assuming that once a vehicle has stopped it will not move again, an assumption that of course will be violated sometimes. Bootstrapping, problem 6, will also not be of immediate interest as the ferry-deck is empty before any vehicles start to roll-on.

### 2.2.1 Survey of techniques

[CK04] did a comparison of the following background models:

1. **Frame difference** The simplest background modeling technique [LFP98]. The difference of intensity \( I \) between two consecutive frames \( \Delta_n = |I_n - I_{n-1}| \) is formed. This model fails to identify pixels inside large uniformly-colored regions as the difference will be zero here\(^1\), problem 7. Pixels

\(^1\)Known as aperture problem
with a difference above a certain threshold is classified as foreground. This technique is only applicable on intensity based frames. Instead of using frame $I_{n-1}$ it is possible to use $I_{n-k}$ which could make the aperture problem less noticeable.

2. **Median filtering** Rests on the assumption that the background will be visible in more than half of the frames. The median value of each pixel of all the frames is used as background model. Usually used on intensity based frames but there is an extension to color based frames, see [CK04] for reference.

3. **Linear predictive filter** A one-step Wiener prediction filter is employed for each pixel using the history of pixel values. The pixel in the next frame is classified as foreground if its value differs too much from the value predicted by the filter. Actually two predictions are used for classification, one based on a Wiener filter created using the history of predicted values and one using past pixel values [TKBM99].

4. **Non-parametric model** Forms a non-parametric estimate of the probability density function $f(x_n)$, that a pixel will have the intensity value $x_n$ in frame $n$:

$$f(x_n) = \frac{1}{T} \sum_{i=1}^{N} K(x_n - x_i)$$

where $\{x_1, x_2, \ldots, x_N\}$ are the $N$ most recent pixel intensities and $K$ is a suitable kernel, often Gaussian. This makes it possible to classify a new pixel as foreground if $f(x_n) < T$. In [EHD00] an implementation can be found along with a comparison to the mixture of $K$ Gaussians model. The advantage of this technique is its ability to model multi-modal background densities, like those formed by rippling water. By using a window of the $N$ most recent pixel intensities the model ‘forgets’ earlier observations’, see [MP04].

5. **Approximated median filter** This is a recursive version of the median filter (2). *In this scheme, the running estimate of the median is incremented by one if the input pixel is larger than the estimate, and decreased by one if smaller*[CK04]

6. **Kalman filter** A Kalman-filter is used for tracking the pixel intensities. The state-space model of the system can consist of the background intensity and its temporal derivative. [RMK95] employ a Kalman-filter based background estimator for tracking people.

7. **Time-adaptive per-pixel mixture of Gaussians (TAPPMOGS)** Since their introduction TAPPMOGS and variations of this technique has become increasingly popular for background modeling and subtraction [SG99,
Mag02]. Each pixel value \( \mathbf{x} \) is assumed to be the outcome of a unknown random \textit{pixel process}. The pixel value \( \mathbf{x} \) can be one or \( n \)-dimensional, in the case of RGB-colors \( \mathbf{x} = [R, G, B]^T \) or in HS-space \( \mathbf{x} = [H, S]^T \).

The unknown \textit{pixel process} \( \{X_1, X_2, \ldots, X_t\} \) is assumed to be modeled by a mixture of \( K \) Gaussians

\[
f(X_t) = \sum_{i=1}^{K} \omega_{i,t} \cdot \eta(X_t, \mu_{i,t}, \Sigma_{i,t})
\]

where \( \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \) is the \( i^{th} \) Gaussian and \( \omega_{i,t}, \mu_{i,t} \) and \( \Sigma_{i,t} \) are the weight, mean value and covariance matrix, respectively of the \( i^{th} \) Gaussian. The covariance matrix is often approximated by

\[
\Sigma_{i,t} = \sigma_i^2 \mathbf{I}
\]

which means that the components of \( \mathbf{X} \) are assumed independent with the same variance. For a new pixel \( \mathbf{x}_t \) the distance

\[
\|\mathbf{x}_t - \mu_{i,t}\| < D \cdot \sigma_i
\]

to the mean of each Gaussian is calculated, and the Gaussian \( m \) which lies within \( D \) standard deviations of the mean is taken as a \textit{match}. The parameters of the \textit{matched} Gaussian \( m \) are then updated as follows:

\[
\omega_{m,t} = (1 - \alpha)\omega_{m,t-1} + \alpha
\]

\[
\mu_{m,t} = (1 - \rho)\mu_{m,t-1} + \rho \mathbf{x}_t
\]

\[
\sigma^2_{m,t} = (1 - \rho)\sigma^2_{m,t-1} + \rho(\mathbf{x}_t - \mu_{m,t})^T(\mathbf{x}_t - \mu_{m,t})
\]

\( \alpha \) is a user-defined learning rate \( 0 \leq \alpha \leq 1 \) and \( \rho \approx \alpha / \omega_{i,t} \). If no \textit{match} can be found the Gaussian with the lowest weight is replaced by one with mean \( x_t \), a large initial variance \( \sigma \) and a low weight \( \omega \). Parameters of unmatched Gaussians are left unchanged except for the weights which are updated as follows\(^2\):

\[
\omega_{i,t} = (1 - \alpha)\omega_{i,t-1} \quad \forall \ i \neq m
\]

this makes the weights decay exponentially. Gaussians are then sorted in decreasing order according to the value:

\[
\frac{\omega_{i,t}}{\sigma_{i,t}}
\]

This value will be large for Gaussians with large evidence \( \omega \) and low variance \( \sigma \) which is the assumed properties of the background. The background model is taken as the first \( B \) Gaussians, where

\[
B = \arg\min_b \left( \sum_{k=1}^{b} \omega_k > T \right)
\]

\(^2\)note the missing \( \alpha \) from Eq. (2.2)
$T$ measures how much of the data that the background should explain. So our background model will consist of the Gaussians $M = \{1, \ldots, B\}$. A new pixel $x_i$ is classified as background if its distance to the mean of any of the Gaussians $M$ are within $D$ standard deviations see (2.1). This is the approach given in [SG99]. [Mag02] made some modifications to this which will be explained more thoroughly later.

8. Hidden Markov Models Hidden Markov models (HMM) can be used to classify the pixel process $\{X_1, X_2, \ldots, X_t\}$ as having been generated by one of the three states {"shadow","background","foreground"}. A HMM model consists of an initial state distribution, a state transition matrix and an observation probability distribution. After learning the parameters of a HMM it can be used for calculating the probability that it generated a particular sequence of observations. In this way each pixel (usually a block of pixels) can have HMM models associated to it, and the observation of pixel-values can be classified as being generated by one of the models. Learning the parameters is usually done off-line and this is a problem for background estimation, but there are ways to circumvent this (e.g. sliding windows). Finding a good initialization of the models is difficult.

Techniques (1-4) can be called non-recursive and (5-7) recursive. The non-recursive techniques uses a fixed history of the $N$ latest pixel values $\{x_t, x_{t-1}, \ldots, x_{t-N+1}\}$ when forming the background model. The non-recursive models sometimes need to keep a large history of pixel values in order to give satisfying performance especially for slow-moving traffic. The recursive models on the other hand does not need any history of the pixel values, this information is incorporated into the model parameters, and the parameters are updated as new samples arrives. In a real-time system one of the recursive models has to be used.

For this system time-adaptive per-pixel mixture of Gaussians (tappmogs) were chosen as a background model. Mainly because it is the most general of the models and has proven to give good results [Mag02, SG99]. Another reason is that frame difference, median filtering, non-parametric models and approximated median filter all can be seen as special cases of tappmog, so if tappmogs are implemented the others are given for free. Background models based on Kalman filters can not handle the waving trees situation, see 2.2, which will occur during rain. So given the tappmogs good record in previous implementations and lack of real disadvantage to the other methods this was a natural choice. The probabilistic nature of this model has a nice appeal to it.

The background model will only classify pixels as foreground or background in its original form. Shadows will be included into the foreground segmentation as has to be removed somehow. In the next section various ways of doing this will be discussed along with the final implemented method.
2.3 Illumination and shadows

Changing illumination and shadows are two of the most elusive problems in computer vision. Nearly any computer vision system used for tracking or recognizing objects in the real world will have to address these problems. This is certainly the case with this system. Over the years numerous attempts has been made to reduce the effect of these problems, some more successfully than others. It is hard to come up with a tracking technique that is unaffected by changes in lighting.

2.3.1 Survey

One might think that using an edge base tracker [CE01] will make the system (partially) invariant to illumination changes but soon one will see that shadows produces strong edges which will appear as objects and thus not really solving the problem. In [ISS02] Javed, Shafique and Shah uses TAPMOGS in combination with gradient based subtraction at the region-level to make the system robust to illumination changes. They calculate a background (gradient based) distribution using the highest weighted Gaussians in a TAPMOGS-background model. For each pixel a gradient vector is calculated and the probability that it belongs to the background gradient based distribution is calculated. If this probability is below a threshold the pixel belongs to the foreground otherwise it belongs to the background. The idea is that the gradient will be unaffected by the illumination change. They also employed frame level processing to identify global illumination changes. Global illumination changes are thought to be present if more than 50 percent of the pixels are classified as foreground.

Illuminants There is no system\(^3\) that can completely ignore the effect of illumination changes and shadows, so these issues has to be addressed. If the illuminant changes, appearance of objects under this new illuminant changes, either by intensity or color, or both. Template based trackers can lose track of objects when the illuminant changes. Under the new illuminant the color or intensity of the object will be different from the template causing the tracker to fail.

Foreground segmentation methods like the one described in 3.1 will also fail. A new illuminant can cause the appearance of background pixels to change, either in only intensity if the new illuminant is of the same color or in color, resulting in misclassification. The improved background estimation algorithm described in Section. 3.1 is more robust to changing intensity than the original method. The problem of a new illuminant with a different color is not examined, as this did not seem to be a major problem with this system. The focus was on reducing the effect of changing intensity and shadows instead.

Shadows A shadow falling on a surface causes the intensity to decrease but the chromaticity and texture are still the same. By observing this one realizes that

\(^3\)to my knowledge
shadows merely reduces the intensity leaving the color and texture unchanged. This is something an illumination invariant system could exploit. Systems that use illumination invariant colors are taking advantage of this fact. A comparative study of different shadow detection algorithms can be found in [PMTC03, PMGT01].

In [MRG98] McKenna, Raja and Gong use the HSV color model to track color objects. They model the color distribution of objects in HSV-space which makes the tracker robust to changes in illumination. For our implemented system, partial illumination invariance is achieved by using cylindrical Gaussian mixture models, so background modeling in HSV-space were never implemented, and could not have been implemented because of the cylindrical model. Note that using cylindrical mixture models in the HSV-color space does not make sense because a color of different intensity does not lie on a straight line in this space. In the RGB-color space colors with the same intensity lies on a straight line from the origin. Still the problem of strong shadows needs to be handled in our system. The cylindrical model only accounts for small changes in intensity.

To remove shadows one would like to use a color model that assigns the same value to colors which only differ in intensity. Some basic tests with the HSV, CIE-L*\(a^*b^*\) and RGB-color models were performed. The purpose of these tests were to see how close shadowed and non-shadowed pixels, of the same color, were in each model. The comparisons were made by calculating the Euclidian distance between a shadowed and non-shadowed pixel of the same color. When doing the calculations the hue component \(H\) of HSV and lightness \(L\) of \(L^*a^*b^*\) were ignored. The result showed that \(L^*a^*b^*\) performed slightly better than RGB in terms of giving a shorter distance between two colors a human would perceive as shaded and un-shaded. HSV would probably have performed better if the right similarity measure had been used. If two HSV colors \([h_1, s_1, v_1]^T\) and \([h_2, s_2, v_2]^T\) are given the distance should be calculated using

\[
1 - \frac{1}{\sqrt{5}}[(v_1 - v_2)^2 + (s_1 \cos(h_1) - s_2 \cos(h_2))^2 + (s_1 \sin(h_1) - s_2 \sin(h_2))^2]^{\frac{1}{2}} \tag{2.5}
\]

as pointed out by Smith and Chang in [SC96]. Although both HSV and CIE \(L^*a^*b^*\) turned out to be illumination-invariant another method gave even better results. This method is described in [KB01] and is based on a computational color model described in [HHID99]. This method is was chosen for implementation because of its superior performance over the others and it is described next.

**A computational color model** This method calculates the brightness distortion \(\beta\) and color distortion \(\gamma\) between a reference pixel \(E = [R_b, G_b, B_b]^T\) and an observed pixel \(I = [R_o, G_o, B_o]^T\):

\[
\beta = \arg \min \left( I - zE \right)^2 
\]

\[
\gamma = \| I - \beta E \| \tag{2.7}
\]

Once the color- and brightness distortion between two pixels \(E\) and \(I\) are known, \(I\) can be classified as a shadowed version of \(E\) if \(T_1 < \beta < 1\) and \(\gamma < T_2\), the brightness
\( \beta \) should be lower for shadowed pixels. By choosing the thresholds \( T_1 \) and \( T_2 \) properly this method proved to be very successful at grouping similar\(^4\) colors. By calculating the chromaticity distortion (2.6) between shaded and un-shaded pixels (i.e. pixels with the same hue but different brightness) it was found that \( \gamma \) tended to be small for these pixels and large for pixels of different chromaticity.

### 2.4 Pixel association

Background segmentation will, as the name suggests, create a segmentation of the current frame. Hopefully the background model is accurate enough to segment out only the interesting objects. Interesting refers to moving objects like people or as in this case vehicles. The segmentation will be a matrix or mask \( F_{xy}(x, y) \) indicating whether the pixel at position \( (x, y) \) is considered foreground or background. In order to be useful the pixels in the segmentation mask has to be grouped somehow. Tracking the individual pixels makes no sense so higher knowledge will be introduced to group pixels into semantic objects. Some of the common ways to do this will be described next.

#### 2.4.1 Survey

**Pixel-pixel association** The obvious way of creating objects of individual pixels is to perform connected component analysis on the segmentation mask \( F_{xy} \). The result of a connected component analysis is several sets of pixels \( C_i \) where pixels in each set \( i \) are connected according to some connectivity criteria. Pixels in different sets are not connected so there is no pixel in set \( C_i \) connected to any pixel in \( C_k \forall i \neq k \). This creates a more convenient entity to work with namely the connected components \( C_i \). These individual components can then be treated as separate objects in the real-world, possibly after the smaller ones has been discarded. Components can then be used when tracking. This is the approach used in [SG99, NJHW03]. This method has to explicitly handle situations were two separate objects in the real-world merges to one connected component. A similar approach were pixels are connected to each other is described in [DCP00]. They calculate the convex hull of foreground pixels and uses the centroid of this to aid in tracking. Tracking is performed by enforcing colinearity of the centroids of the convex hulls.[DCP00]

**Pixel-object association** Instead of associating pixels to each other, the pixels can be associated to an object model. In [Mag02] foreground pixels are associated to object models, each model consists of position, size and color distribution. Object model positions are predicted forward in time using a Kalman filter and pixels are assigned to the best model. The best model match is found by comparing the distance of pixels to each model center, in some cases the color value is compared to the color distribution of the model. All the parameters of the model are continuously

\(^4\)Similar in terms of being different shades of one color.
updated by associated pixel-values to make it adaptive. This method was primarily chosen because the size of objects were relatively small and the segmentation generated unconnected pixels. In [MRG98] a Gaussian mixture model is created for an object using HSI color values. Pixel-values from each frame are compared to the Gaussian mixture model and those that match build up the object. The mixture model is updated with matching pixels to make it adaptive to illumination changes. A similar approach could be used to associate segmented foreground pixels to objects. If an accurate initial model of the object color distribution can be built.

2.5 Tracking and data association

Tracking and data association are the cornerstones of surveillance systems, without good tracking and data association between frames, information gained in one frame would not be able to propagate to the next frame. There is of little use to locate an object in frame \( n \) if there is no information about where it will be in frame \( n + k \). By tracking an object through frames information about the number of objects in the scene can be collected. There are numerous techniques for tracking objects and associating data. Data association can be seen as a kind of tracking, where the tracking is performed by associating data between frames. In this section, tracking will be thought of as the problem of finding an object in the next frame. Common techniques for tracking and data association are

1. Nearest neighbor
2. Kalman filter(s)
3. Graph based methods

2.5.1 Nearest neighbor

This is one of the simplest approaches to tracking but it can still be quite effective. Consider different objects segmented as connected components, then the tracking problem consists of locating component \( k \) from frame \( t \) in frame \( t + 1 \). The obvious solution is to assume that object motion between frames is negligible and thus assign \( k \) to the component in the next frame that is closest to its current position. This approach or variations thereof gives satisfactory results when inter-frame motion is small but fails when this is not a valid assumption.

In figure 2.1 the problem of ellipse matching is illustrated. The nearest neighborhood approach would correspond to the matching of the centers. Simply matching ellipses to the nearest neighbor failed because often ellipses where mismatched to new ellipses like the right one in figure 2.1(b) instead of the left one. The reason why it failed was that it found the nearest ellipse in the screen coordinate system instead of the ground coordinate system.

Ellipses does not have axes of the same length which makes it harder to determine the closest ellipse. A situation like the one illustrated in figure 2.2 often arise. In
this figure one can see that matching ellipses based on the distance between centers can give wrong matches in this particular implementation. In order to improve the matching capabilities a distance measure based on the distance between perimeters of ellipses is used.

2.5.2 Kalman filter(s)

Kalman filters has found widely use in numerous fields like for example navigation, surveying, vehicle tracking, aircraft tracking, missile tracking and demographic estimation. It was first introduced in 1960 by Rudolf Kalman and has since proved to be a valuable tool. Kalman filters can be used to predict the states of an unobservable state process where measurements are taken according to a measurement process. It is assumed that the system is modeled by a state process with zero mean white Gaussian noise added to it and that measurements are made according to the measurement process with zero mean white Gaussian noise added. Essentially the filter refines the state estimates by incorporating the information contained in the measurements and the properties of the noise process. Being a recursive filter means that no data has to be stored other than the matrices of the filter.

In [BER02] Black, Ellise and Rosin uses a Kalman filter to track objects in multiple views. Two filters are used to track objects in 2D image coordinates and in 3D world coordinates. During occlusion the object position is predicted into the next frame until a match is found.

In [MP01] Osama Masoud and Nikolaos P. Papanikolopoulos update the parameters of an object model by using a Kalman filter. Tracking of objects (pedestrians)
is done by using an extended Kalman filter (EKF\(^5\)).

### 2.5.3 Graph based methods

Graph based methods is a data association approach to tracking. Tracking is performed by creating correspondences between data in different frames. The problem of tracking is posed as a matching problem where entities in one frame should be matched to entities in the next. Consider the result of a connected component analysis on a foreground segmentation, this will render several components which should be matched between frames. When matching components or blobs between frames some issues has to be addressed

**Splitting** On blob can split into two blobs, caused by improper segmentation.

**Merging** Two blobs can merge into one blob, caused by objects being close to each other.

\(^5\)An extension of the Kalman filter to handle non-linear systems. Non-linearity is achieved by linearizing the system and measurement model in each update step.
**Addition** A new *blob* can appear, caused by objects entering the scene.

**Removal** A *blob* might disappear, caused by objects leaving the scene.

These problems make the matching problem harder. In [SA02] Park and Aggarwal forms a graph where blobs in the previous frame are matched to blobs in the current frame. To facilitate the matching an dissimilarity measure is introduced. For each blob a *feature* vector $m$ is constructed consisting of

- Blob size $\in \mathbb{R}^1$
- Blob color $\in \mathbb{R}^3$
- Blob centroid, median position $\in \mathbb{R}^2$

Dissimilarity between blob $i$ and $j$ is then defined as

$$\Delta_{ij} = (m_i - m_j)^T (\Sigma_i + \Sigma_j)^{-1} (m_i - m_j)$$

where $i$ indicates a blob in frame $n$ and $j$ a blob in frame $n+1$, $\Sigma_x$ is the covariance for all feature vectors in a frame $x$. Then they search for the matches that minimizes the overall dissimilarity. Special care is taken to avoid the problems described before. The results in [SA02] showed this approach to be robust to occlusion.

This approach were also implemented for this particular system but suffered from some problems. The first problem was to find a good description of the color of each *blob*, simply averaging all the colors in each *blob* tend to give a grayish mean color for all vehicles. So the color dissimilarity would be zero between all *blobs* and thus not improve the matching. Calculating the dissimilarity of the sizes of *blobs* was also prone to errors because in this particular system, the size varies between frames. As one can see in figure 2.1(b) the most similar *blob* in terms of size would be the incorrect right *blob*. So if the dissimilarity measure between size and color are removed what remains is simply a nearest neighbor matching.

### 2.5.4 Proposed tracking solution

None of the discussed tracking approaches gave adequate results on their own so tracking was performed using a combination of the methods described. First of all one can note that vehicles drive in lanes when boarding the ferry, this means that the ellipses will tend to go in a straight line between frames. To exploit this one can narrow the search area so that matching takes place in a area where they most probably will be. This is done by first predicting the position of the blob in the next frame using a Kalman-filter, using this predicted position a search-cone is created like illustrated in figure 2.3. This particular ellipse will now only be matched to other ellipses appearing inside the search-cone in the next frame, thus preventing it from matching ellipses in different lanes, at least when the ellipses are close to the camera.

If the predicted position of a vehicle falls inside an ellipse, this ellipse is assumed to be the one representing this vehicle in the next frame. If the predicted position
Figure 2.3. Illustration of the proposed tracking algorithm.

does not fall inside an ellipse, then the minimum distance between the perimeters of the ellipse associated to the vehicle and ellipses in the new frame is calculated to find a match.
Chapter 3

Detailed design

This chapter describes the design and theory behind the final system. The theory will be explained in the same order as the various methods are executed in the processing pipeline. First the background estimation is described, then the post-processing and ellipse forming steps, at the end the tracking is explained and the theory behind the Kalman filter and the chapter ends with a description of the metric rectification used for length estimations.

3.1 Background estimation

Background estimation is necessary for keeping a model of what is background and what is not. Having a model of the background makes it possible to classify pixels as being background or foreground.

3.1.1 Motivation

When deciding which background model to use one need to keep in mind the problems described in Section 2.2.1. Because the system is supposed to operate in widely different weather conditions like snow, rain, day and night for 365 days a year, this limits the choice of model. Among the models described in Section 2.2.1 only the Non-parametric model and TAPPMOGS can handle multi-modal pixel distributions. Multi-modal background distributions will appear in a scene containing the waving-tree problem, where the waving motion of the tree or something similar only can be explained by a multi-modal distribution. In our case this will probably be needed for snowfall and rain. [CK04] came to the conclusion that TAPPMOGS gave the best result. TAPPMOGS and their variations have proved to be successful in numerous tracking applications. [SG99, Mag02, Har02] and [Mag02] improved the model by making it illumination invariant, which is just property needed for this system. In a TAPPMOG system one has the opportunity to handle shadows directly at pixel level. The decision were to use TAPPMOGs as described by [Mag02] for the background estimation. Mixture models has the advantage of not having to store past pixel values, they are implicitly stored when learning the parameters of the model.
Memory is probably not the first resource to get exhausted on a modern computer though. Another advantage with mixture models is its ability to save an old model of the background. If it is adopted to a new background the old one will still exist if sufficiently many components are used, the old background will be described by components with lower weights.

The next section will describe the theory behind this model.

### 3.1.2 Illumination invariant per-pixel mixture model

Let us start off this discussion by first describing the original mixture-model described by [SG99] and then later incorporate the improvements made by D. Magee.

The idea is to model the background as a multi-modal distribution and the foreground will be those pixels that are not explained by this background-model. Stauffer and Grimson make some assumptions about the background namely that:

1. Background has more data supporting it, since it is repeated.
2. Most of the foreground objects does not have the same color as the background.
3. Stationary pixel-values should be incorporated into the background model.

Denote the history of pixel-values by

\[ \{X_1, X_2, \ldots, X_t\} = \{I_i(x_0, y_0) : 1 \leq i \leq t\} \quad (3.1) \]

where \( I_i \) is the frame at time \( i \) and \( X_t \in \mathbb{R}^n \) is a certain pixel-value at time \( t \) at position \( (x_0, y_0) \). If we are using pixel-values with 3 scalars per pixel then \( X_t \in \mathbb{R}^3 \), as the rgb-color space will be used in this discussion, as well as in the rest of this thesis, this will be the case henceforth, if not otherwise stated.

**Gaussian mixture models** The pixels can be modeled as a mixture of \( K \) Gaussians, with the probability of observing the pixel-value \( X_t \) written as

\[
P(X_t) = \sum_{i=1}^{K} \omega_{i,t} \cdot \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \quad (3.2)
\]

where \( \omega_{i,t}, \mu_{i,t} \) and \( \Sigma_{i,t} \) are the weight, mean and covariance of the \( i^{th} \) Gaussian at time \( t \). Further, \( \eta \) is the Gaussian probability density function

\[
\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} \sqrt{|\Sigma|}} e^{-\frac{1}{2} (X_t - \mu)^T \Sigma^{-1} (X_t - \mu)} \quad (3.3)
\]

the covariance matrix is taken to be

\[
\Sigma_{i,t} = \sigma_i^2 I \quad (3.4)
\]

which means that the red, green and blue color values are independent and with equal variance. Forcing the variance to be the same for all the color-channels of
the pixel can create problems as pointed out by [PS02] Care should be taken with non-linear color spaces like hue, saturation and value and especially with spaces combining unlike quantities such as intensity and range, where each dimension is likely to have a peculiar distribution. By introducing the parameter set \( \theta_i = \{ \mu_i, \sigma_i \} \) and \( \Theta = \{ \omega_1, \ldots, \omega_K, \theta_1, \ldots, \theta_K \} \) and dropping the subscript \( t \) on the parameters, equations (3.3) and (3.4) can be re-written as:

\[
P(X_t|\Theta) = \sum_{i=1}^{K} \omega_{i,t} \cdot \eta(X_t|\theta_i) \tag{3.5}
\]

\[
\eta(X_t|\theta_i) = \frac{1}{(2\pi)^{n/2} \sqrt{\Sigma}} e^{-\frac{1}{2} (X_t - \mu)^T \Sigma^{-1} (X_t - \mu)} \tag{3.6}
\]

**Expectation maximization** Each mixture model has to be learned from observed pixel values. The standard technique for learning the parameters of Gaussian mixture models is expectation maximization (EM) [Bi95]. Estimating the parameters \( \Theta \) with the EM-algorithm requires that all data \( \{X_1, X_2, \ldots, X_t\} \) is known when doing the estimation and that the process is stationary. The requirement that all data should be known is violated for an on-line system where frames are acquired one at a time. The stationarity requirement is also violated because the background tend to change, especially in outdoor scenes. So an online or recursive EM algorithm is needed which can estimate the parameters as new pixel-values arrive. Stauffer and Grimson implements an on-line K-means approximation to the EM-algorithm which works as follows.

Once a new pixel-value \( X_t \) arrives, the Mahalanobis distance \( d \) to the existing \( K \) Gaussians is calculated

\[
d_i = \| X_t - \mu_{i,t} \|
\]

and the Gaussian with the shortest distance such that \( d_i < D \cdot \sigma_i \) (i.e. within \( D \) standard deviations of the mean) is selected as a match. The matching Gaussian \( m \) at time \( t \) gets its parameters updated in the following manner:

\[
\omega_{m,t} = (1 - \alpha) \omega_{m,t-1} + \alpha 
\]

\[
\mu_{m,t} = (1 - \rho) \mu_{m,t-1} + \rho X_t 
\]

\[
\sigma_{m,t}^2 = (1 - \rho) \sigma_{m,t-1}^2 + \rho (X_t - \mu_{m,t})^T (X_t - \mu_{m,t}) 
\]

where \( X_t \) is the newly acquired pixel-value and

\[
\rho = \alpha \eta(X_t|\theta_i) \quad \theta_i = \{ \mu_{i,t}, \sigma_{i,t} \}
\]

Unmatched Gaussians only get their weights updated according to:

\[
\omega_{i,t} = (1 - \alpha) \omega_{i,t-1} \quad \forall i \neq m 
\]

If no match is found the Gaussian with the lowest weight \( \omega \) is replaced by one with mean \( \mu_{i,t} = X_t \) and a low intital-variance \( \sigma_{i,t}^2 \) and weight \( \omega_{i,t} \).

\footnote{In the implementation \( D \) is set to 3}
Figure 3.1. Figure (a) illustrates the spherical Gaussian mixture model proposed by Stanfill and Grimson. Figure (b) illustrates the cylindrical Gaussian mixture model proposed by Magee. The 1-dimensional Gaussian curve is also illustrated.

3.1.3 Improved background mixture model

The previous approach were modified by Magee [Mag02] to make it robust to illumination changes. Magee suggested to use a mixture of axes oriented cylindrical distributions instead of the original spherical distributions. Observing that colors of the same hue but different intensity lies along a straight line through the origin in the RGB-color space, shows that using cylinders makes perfect sense, see Figure 3.2.

The model described in [Mag02] is the one used in this system although some changes has been made, which will be explained later. For each mixture component define a direction \( \nabla_i \) from the origin to the mixture mean \( \mu_i \):

\[
\nabla_i = \frac{\mu_i}{\| \mu_i \|} \tag{3.13}
\]

Each mixture component consists of an estimated 1-D mean value \( X_{Pr_i} \in \mathbb{R}^1 \) and variance \( \sigma_i^2 \) along this direction:

\[
X_{Pr_i} = X_{RGB} \cdot \nabla_i \tag{3.14}
\]

\[
\sigma_i^2 = (X_{Pr_i} - \| \mu_i \|)^2 \tag{3.15}
\]

These, together with \( \mu_i \) and \( \omega_i \) are the parameters of the mixture components. The parameters are updated as in (3.8, 3.9 and 3.10) using \( X_{Pr_i} \) as \( X_t \). When checking for the closest match in the update step, any Gaussian with perpendicular deviation

\[
X_{Per_i} = \sqrt{(\| X_{RGB,I} \|^2 - X_{Pr_i}^2)}
\]

above a noise threshold \( T_1 \) is not considered as a match causing the model to be cylindrical. If \( X_{Per_i} \) is below the threshold \( T_1 \) the Mahalanobis distance is calculated
as:

\[ d_i = \frac{(X_{Pr_i} - \| \mu_i \|)^2}{\sigma_i} \]  \hspace{1cm} (3.16)

If this distance \( d_i \) is below a threshold \( T_2 \) the Gaussian is considered as a *match* and the parameters are updated. If no *match* can be found the lowest ranked Gaussian is replaced by a new one, as in the original approach by Stauffer and Grimson. All weights for unmatched mixture components are updated according to (3.12).

**Foreground/Background classification** To classify pixels, the Gaussians are sorted by decreasing weight \( w_i \) and the first \( B \) are chosen as background:

\[ B = \arg\min_b \left( \sum_{k=1}^b \omega_k > T_b \right) \]  \hspace{1cm} (3.17)

where \( T_b \) is a threshold for how much of the data should be *explained* by the background model. A possible improvement could be to include the variance when ranking the weights, such that low variance mixture components are favored over high variance ones. A low \( T_b \) means that fewer Gaussians will be used in the background model. When a new pixel-value \( X' \) arrives it is classified by calculating the distance
(3.16) to each of the first $B$ Gaussians\textsuperscript{2}, and if the distance (to any of them) is below $M_{th}$ the pixel is marked as background otherwise it is marked as foreground. Classifying a whole frame in this manner gives a foreground segmentation mask

$$ F(x, y) $$

which is just a matrix where each element $f_{xy}$ indicates whether that pixel is foreground or background, see Figure 1.4(b).

Variable update rate: If the background model is updated every frame the variance of matching Gaussians can shrink too much if successive pixel-values tend to be close. To prevent the variance from shrinking too fast and keeping the model from being dominated by one single Gaussian a variable update rate is employed by Magee [Mag02]. The idea is to lower the update rate for pixels that match the background and increase it for foreground pixels (pixels that do not fit the background model) in the following way:

$$ R_{xy} = \begin{cases} 
R_{input} & \text{for } N_{bgd_{xy}} < N_{max} \\
\frac{N_{bgd_{xy}}}{R_{input}} & \text{otherwise}
\end{cases} $$

where

$ R_{xy} = $ update frame rate (fps) at pixel $(x, y)$.

$ R_{input} = $ input frame rate (fps).

$ N_{bgd_{xy}} = $ number of consecutive frames classified as background at pixel $(x, y)$.

$ N_{max} = $ maximum threshold on frame rate division.

### 3.1.4 Discussion

One problem with this kind of adaptive Gaussian mixture model\textsuperscript{3} is that it can incorporate slow-moving homogeneously colored objects into the background. For these kind of objects the pixel-values inside the object will be the same for several successive frames causing this color to be adopted into the background. Variable update rates as described in the previous paragraph can help to partly mitigate this effect.

Another problem is that the foreground segmentation usually exhibits noise generated foreground pixels caused by pixel-values falling in the tails of the distributions. Both of these problems can partly be solved at higher levels. Noise generated foreground pixels can be dealt with by post-processing the frames as described in Section 3.3.

Slow-moving objects can be prevented from affecting the background model by lowering the update-rate of pixels known to represent a foreground object, see section 3.6 about prediction. Output-data from the background segmentation algorithm will be a matrix $F_{xy}(x, y)$ indicating whether pixel $(x, y)$ is foreground or background.

\textsuperscript{2}The first $B$ sorted Gaussians

\textsuperscript{3}both cylindrical and spherical.
Mixture models are on the edge of what can be done by using information from single pixels. To further improve the segmentation one has to consider pixels at region level and frame level. Markov random fields is one way to impose spatial constraints on the foreground segmentation. In this system some post-processing is done to slightly improve the segmentation. To improve the foreground segmentation regions and frames can be analyzed to guide the segmentation.

3.2 Illumination and shadows

Illumination changes and cast shadows are two of the hardest problems to solve for a real-time tracking system like this. As one can see in Figure 1.4(a), the size of shadows can be quite large compared to the size of vehicles and render the length estimate useless. Shadows from one vehicle can be cast onto another vehicle causing them to be linked in the connected component analysis of the foreground segmentation. Sudden illumination changes, such as those caused by clouds passing in-front of the sun, gives problem with the foreground segmentation. Although some degree of illumination change is accounted for by the particular background model used in this system, large variations can not be handled. To improve the systems ability to detect shadows and highlights or rather shadowed background and shadowed highlight a computational color model is introduced. The computational color model has proved its ability to cluster different shades of the same color into single clusters.

3.2.1 Motivation

Some ways of eliminating the effect of shadows and illumination changes were discussed in Section 2.3. In this system a modified version of the approach given in [KB01] was implemented. This computational color model fits nicely into the framework of the proposed background model. The idea of this color model is to compare two colors by ignoring their intensity and just look at the chromaticity difference. Two colors of different intensity value but with a small difference in chromaticity value will be regarded as the same. As previously pointed out, shadowed background pixels has the same chromaticity as the background pixel but having a lower brightness or intensity. When testing some of the color models, as described in the survey section, this model showed the most promising results on real-world data.

3.2.2 Computational color model

In Figure 3.3 an illustration of the model can be found. The expected chromaticity line $\|E\|$ is the line going through the origin of the RGB color space to the expected color value $E$. Given a background pixel with expected RGB color value $E_b$, the
chromaticity difference $\gamma_t$ and brightness difference $\beta_t$ to an observed pixel value $I_t$ can be calculated as follows:

$$\beta_t = \arg\min_z (I_t - zE_t) \cdot (I_t - zE_t)$$  \hspace{1cm} (3.20)

$$\gamma_t = \|I_t - \beta_tE_t\|$$  \hspace{1cm} (3.21)

where $\beta_t$ measures how much the observed pixel value differs in brightness and $\gamma_t$ measures the difference in chromaticity (i.e. the orthogonal distance to the expected chromaticity line). See Appendix B for calculations of $\beta_t$. In this system the pixels

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{color_model.png}
\caption{Illustration of the computational color model.}
\end{figure}

are classified as background, foreground, shadowed background or highlighted foreground. This classification is performed just after the pixel has been determined to not belong to the background model (i.e. it is a foreground pixel). Denote the pixel value at time $t$ by $I_t$ and suppose that it was classified as foreground by the background model. The system now checks whether this pixel is part of a shaded background and misclassified as foreground. If it is a shaded background pixel, the classification needs to be corrected. To correct the classification the chromaticity and brightness distortions are calculated using (3.21) and (3.20) but instead of using the mean of the most probable background Gaussian as $E_t$ another approach is suggested.

In this system the average of the means of the highest ranked background Gaussians in a neighborhood around pixel $I_t$ is used. Write the pixel $I_t$ located at

34
position \((x, y)\) as \(I_t(x, y)\) and the mean of the highest ranked background Gaussian at position \((x, y)\) as \(\mu_t(x, y)\). The expected color \(E_t\) is then calculated as the average of the means

\[
M = \{\mu_t(x, y), \mu_t(x-1, y), \mu_t(x+1, y), \mu_t(x, y-1), \mu_t(x, y+1), \mu_t(x-1, y-1), \mu_t(x+1, y+1)\}
\]

in a neighborhood, like this

\[
E_t = \frac{1}{7} \sum_{i=1}^{7} M_i \quad E_t \in \mathbb{R}^3
\]  

(3.22)

The chromaticity and brightness difference between \(E_t\) and \(I_t\) are then calculated using (3.21 and 3.20). Denote the states \(S\) a pixel can be in by

\[
S = \{"\text{foreground"}, "\text{shadowed background"},
        "\text{highlighted background"}, "\text{background"}\}
\]  

(3.23)

and the states a foreground pixel can be in by

\[
S' = S \setminus \{"\text{background"}\}
\]

A possibly misclassified foreground pixel \(I_t\) can then be re-classified into one of the states \(S'\) using the classifier \(C\)

\[
C(I_t) = \begin{cases} 
"\text{shadowed background"} & \text{if } (\beta_t < T_{\beta 1}) \land (\gamma_t < T_{\gamma 1}), \\
"\text{highlighted background"} & \text{if } (\beta_t > T_{\beta 2}) \land (\beta_t < T_{\beta 3}) \land (\gamma_t < T_{\gamma 2}), \\
"\text{foreground"} & \text{otherwise.}
\end{cases}
\]

(3.24)

Where \(T_{\beta 1}, T_{\beta 2}, T_{\beta 3}, T_{\gamma 1}\) and \(T_{\gamma 3}\) are user-defined thresholds\(^4\). Values for the thresholds were found by trial-and-error. The classifier basically compares the brightness of the new pixel to the background pixel and if this brightness is lower and the chromaticity difference is not too large then it is classified as shadowed foreground. Highlighted background is classified in a similar way but the brightness should be larger in this case. During the foreground segmentation the classifier \(C\) is applied to all foreground pixels to reduce the number of misclassified pixels, an example of the resulting segmentation can be found in Figure 3.4(b). This step proved to significantly reduce the number of false positives (FPs) (i.e foreground pixels that should

\(^4\)typical values are \(T_{\beta 1} = 0.5, T_{\beta 2} = 1.25, T_{\beta 3} = 5, T_{\gamma 1} = 26, T_{\gamma 3} = 50\)
have been background pixels.) Classifying all foreground pixels $F_{xy}$ (see 3.18 on page 32) from the background segmentation step then gives the true segmentation

$$\hat{F}_{xy} = C(F_{xy}(x, y)) \text{ where } \hat{F}_{xy}(x, y) \in S$$ (3.25)

where the states $S$ are re-mapped according to

- "shadowed background" $\sim 0$
- "highlighted background" $\sim 0$
- "foreground" $\sim 1$
- "background" $\sim 0$ (3.26)

Sometimes when images of the background segmentation are displayed in this thesis report the states $S$ has been re-mapped to values according to

- "shadowed background" $\sim 127$
- "highlighted background" $\sim 200$
- "foreground" $\sim 255$
- "background" $\sim 0$ (3.27)

these are the grey-values used in images pertaining to foreground segmentation in this thesis. Usually only the foreground and background pixels will be shown.

### 3.2.3 Discussion

The classifier (3.24) gave very good results if the threshold were fine-tuned. Moving shadows were almost completely removed in daylight scenes. Highlights caused by headlights were also partly removed in night sequences, see Figure 3.5(b). In
this image the thresholds for the classifier are the same as those in the image of Figure 3.4(b) on the facing page. The night scene is not really useful for tracking because it is heavily compressed and has bad lighting, although it can be interesting for comparison.

One problem with the classifier is that it can not be blindly turned on and used for all scenes. In those scenes where strong shadows were not present, like the overcast sequence in Figure 1.2(b) on page 5, the classifier caused the frames to be under-segmented, giving large holes in objects with colors similar to the background. This was remedied by manually activating the classifier for those scenes which contained salient shadows. In a final system this would have to be done automatically. Thresholds can not be set too tight because the system has to be able to handle different scenes autonomously, adaptive thresholds should be investigated.

![Image](image_url)

(a) Video frame with highlight.  
(b) Corresponding foreground segmentation. Highlighted background is correctly classified (light-grey).

Figure 3.5. Video frame and the resulting foreground segmentation.

### 3.3 Post-processing of segmentation mask

After the foreground segmentation and re-classification of pixels described in Section 3.2 on page 33 the next step is to post-process the foreground segmentation. In this step, noise generated pixels are removed and close regions are connected to simplify the connected component analysis.

#### 3.3.1 Motivation

Typically the segmentation mask given from the background estimation module will be highly fragmented and contain noise. Noise generated foreground pixels are especially numerous when compressed video sequences are used. The noise consists
of single pixels or small groups of pixels scattered over the foreground mask. To diminish their effect a post-processing step is implemented. Basically it consists of a series of morphological operations designed to reduce noise while keeping the overall structure of objects. Morphological operations are a very powerful tools considering their simplicity. In this system the objects being tracked are relatively large especially when they enter the scene. If this were not the case morphological operations might not have been useful for noise removal. Eroded and dilated masks are shown in Figure 1.4(c) and 1.4(d) on page 9.

3.3.2 Erosion

Erosion is performed on the foreground mask \( \hat{F}_{xy} \) where the states have been remapped according to (3.26), two states only exists, 255 for foreground and 0 for background. In the erosion step the following structure element is used

\[
\begin{bmatrix}
  0 & 1 & 0 \\
  1 & 1 & 1 \\
  0 & 1 & 0 
\end{bmatrix}
\]

(3.28)

with anchor point at the center, indicated with bold typeface. This will remove isolated pixels (i.e noise) while keeping the structure of larger segments. With a larger structuring element large regions would be removed. This structuring element is designed to only remove very tiny regions like isolated pixels.

3.3.3 Dilation

Dilation is performed to connect segments that are just a couple of pixels apart, thus creating larger more manageable segments for connected component analysis. The structuring element used for dilation is

\[
\begin{bmatrix}
  1 & 0 & 0 & 0 & 1 \\
  1 & 1 & 0 & 1 & 1 \\
  0 & 0 & 1 & 0 & 0 \\
  1 & 1 & 0 & 1 & 1 \\
  1 & 0 & 0 & 0 & 1 
\end{bmatrix}
\]

(3.29)

with anchor point at the center.

3.3.4 Discussion

Erosion and dilation improves the segmentation and removes small uninteresting structures. The foreground segmentation mask after erosion and dilation will be called

\[
F_{xy}^p(x, y) \quad f_{xy}^p \in \{0, 1\}
\]

(3.30)

Erosion and dilation are very powerful tools despite their simplicity.
3.4 Region processing

Region processing is the step where pixels are grouped together to create objects or parts of objects. Post-processing results in the foreground segmentation $F_{xy}^{n}$ containing only true foreground pixels, shadows and highlights have been removed according to (3.25). In Section 2.4 a survey of different techniques for grouping individual pixels into meaningful entities were described. Figure 3.6(a) shows the foreground segmentation in one frame of the overcast sequence, tracking objects directly from this segmentation only, will be hard. To simplify the tracking, ellipses are created as shown in Figure 3.6(b), this reduces the segmentation to only six objects. Keeping track of these ellipses between frames is definitely easier, the ellipses has well defined centers and spatial extension.

(a) Foreground segmentation (before post-processing).  
(b) Connected component ellipses.

Figure 3.6. Foreground segmentation before post-processing and connected component ellipses.

This section describes how the ellipses are created from connected components in the post-processed foreground segmentation.

3.4.1 Motivation

In this system a connected component analysis is performed and ellipses are fit to each component. Henceforth connected components and components will be used interchangeably. The global shape of each component is described by an ellipse and these ellipses will be the smallest entity used, instead of connected components or pixels. Ellipses are more manageable entities to work with, since they can be grouped together to form objects or a single ellipse can correspond to a real-world object. Each ellipse captures most of the properties of a connected component.
3.4.2 Connected component analysis

First a formal definition of connected components is given and then a short discussion about the details of the implementation in this system follows.

**Neighbors** These definitions are from [GW02]. A pixel $p$ at position $(x, y)$ has four neighbors in the horizontal a vertical direction like this

$$N_4(p) = \{p(x-1,y), p(x+1,y), p(x,y-1), p(x,y+1)\}$$

called 4-neighbors of $p$. The four diagonal neighbors of $p$

$$N_D(p) = \{p(\pm 1x, y+1), p(x+1, y-1), p(x-1, y+1), p(x-1, y-1)\}$$

together with the 4-neighbors $N_4(p)$ are called 8-neighbors and are denoted by $N_8(p)$.

**Adjacency** Let $V = 1$ be a set of gray-level values in the image then we can define two types of adjacency

1. 4-adjacency. Two pixels $p$ and $q$ with values from $V$ are 4-adjacent if $q$ is in the set $N_4(p)$.

2. 8-adjacency. Two pixels $p$ and $q$ with values from $V$ are 8-adjacent if $q$ is in the set $N_8(p)$.

In this thesis 8-adjacency will be used.

**Path** A path from pixel $p$ with position $(x, y)$ to pixel $q$ with position $(s, t)$ is a sequence of distinct pixels with positions

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$$

where $(x_0, y_0) = (x, y)$, $(x_n, y_n) = (s, t)$, and pixels $(x_i, y_i)$ and $(x_{i-1}, x_{i-1})$ are adjacent for $1 \leq i \leq n$ [GW02].

**Connected** Let $S$ represent a subset of pixels in a mask or image. Two pixels $p$ and $q$ are said to be connected in $S$ if there exists a path between them consisting entirely of pixels in $S$. For any pixel $p$ in $S$, the set of pixels that are connected to it in $S$ is called a connected component of $S$ [GW02].
Notation  Denote all connected components of a mask by the matrices

\[ C_i(k, l), \quad i = 0, \ldots, N \]  
\[ k = 1, 2 \]  
\[ l = 1, \ldots, L_i \]

where \( N \) is the number of components and \( L_i \) is the number of points in component \( i \). Each column of \( C_i \) represents the \( x \) and \( y \) coordinates of one pixel in component \( i \):

\[ C_i = \begin{bmatrix} x_0 & x_1 & \ldots & x_{L_i} \\ y_0 & y_1 & \ldots & y_{L_i} \end{bmatrix} \]

When the foreground segmentation and post-processing is done the connected components are extracted from the mask \( F_{xy} \). Only components consisting of foreground pixels are considered, the background is discarded. The components are called \( C_i \) using the notation in the previous paragraph. All components are converted to ellipses because ellipses are easier to work with. The conversion is done by fitting an ellipse to the points (i.e. pixels) building up each component. Actually the final ellipses are generated by a two-step process of component analysis and ellipse fitting, this will be evident later.

**3.4.3 Ellipse fitting**

Given a set of components \( C_i \), one ellipse can be fitted to each component. The first component consists of the pixels at positions

\[ C_1 = \begin{bmatrix} x_0 & x_1 & \ldots & x_{L_1} \\ y_0 & y_1 & \ldots & y_{L_1} \end{bmatrix} \]

where each column is a pair of coordinates \((x, y)\) to one pixel. Call the first column \( x_0 = [x_0, y_0]^T \), the second \( x_1 = [x_1, y_1]^T \) and so forth. Then the coordinates of all points inside component \( n \) are given by

\[ x_0, x_1, \ldots, x_{L_n} \]

If (3.34) are samples from a stochastic variable \( X \). The mean and covariance of \( X \) are

\[ \mu_x = E\{X\} \]  
\[ \Sigma_x = E\{(X - \mu)(X - \mu)^T\} \]

which can be estimated by the sample mean and sample covariance as follows

\[ \mu_x = \frac{1}{L_1} \sum_{i=1}^{L_1} x_i \]  
\[ \Sigma_x = \frac{1}{L_1} \sum_{i=1}^{L_1} x_i x_i^T - \mu_x \mu_x^T \]

41
The dimension of $\Sigma_x$ will be $2 \times 2$. An ellipse containing the points in (3.34) can now easily be found from the covariance matrix and the mean.

### 3.4.4 Ellipse parameters

Denote the normalized eigenvectors of $\Sigma_x$ by $e_1$ and $e_2$ and the corresponding eigenvalues by $\sigma_1$ and $\sigma_2$. Eigenvalues are extracted using SVD eigendecomposition. Two eigenvector-eigenvalue pairs can be formed, $\{\sigma_1, e_1\}$ and $\{\sigma_2, e_2\}$ where $\sigma_1 > \sigma_2$. The first pair $\{\sigma_1, e_1\}$ will represent the semi-major axis of the ellipse and the second pair $\{\sigma_1, e_1\}$ the semi-minor axis. The first pair with the largest eigenvalue corresponds to the direction of maximum variation of the data, see Figure 3.7. The second pair will be orthogonal to the first. Two vectors can be formed, $s_1 = d\sqrt{\sigma_1}e_1$ and $s_2 = d\sqrt{\sigma_2}e_1$, where $d$ is a scale factor which determines how much of the data the ellipse should inscribe\(^5\). Now the ellipse is defined by the center $\mu_x$, the semi-major axis direction $s_1 = d\sqrt{\sigma_1}e_1$ and the semi-minor axis direction $s_2 = d\sqrt{\sigma_2}e_1$, see Figure 3.7.

![Figure 3.7](image)

**Figure 3.7.** Connected component and ellipse inscribing points. The ellipse has been drawn with $d = 1.96$, ellipse axes are orthogonal to each other.

To simplify notation write the parameter set $\xi_i$ of the ellipse inscribing the points in component $i$ as

$$\xi_i = \{\mu_x^i, s_1^i, s_2^i, d\}$$

(3.39)

\(^5\)typically 1.96
3.4.5 Two-pass algorithm

Once the foreground segmentation is done ellipses are created to capture the properties of the segmentation. This is done in a two-step process described next. Connected components having small area are automatically discarded when creating the ellipses.

**Pass I** First the connected components $C_i$ of the foreground mask $F^{xy}_{xy}$ are extracted and one ellipse $\xi_i$ for each $C_i$ is created as describe in Section 3.4.3. The problem now is that this can generate several ellipses close to each other, ellipses which all represents the same object, compare with the small ellipse in Figure 3.6(b) on page 39. So instead of tracking these clusters of ellipses, which could be quite cumbersome, they are *forced* to group, this is the reason for step II.

**Pass II** Forcing the ellipses into a group is done by first creating a temporary foreground mask with the same size as $F^{xy}_{xy}$. Then the interior points of the ellipses $\xi_i$, $i = 1, \ldots, N$ with new scale-factor $d \approx 2.5$ are marked by 1s in this temporary mask. This will have the effect of enlarging the ellipses, forcing ellipses close to each other to overlap and thus they will be considered as one component. The next step is to do a second connected component analysis on the temporary mask and then convert these components to ellipses as before. Call the ellipses generated in the second pass

$$\xi^*_i \quad i = 1, \ldots, M$$

(3.40)

After this step, ellipses close to each other hopefully has merged to one large ellipse. One has to chose the threshold $d$ carefully so that ellipses from different vehicles do not merge.

3.4.6 Discussion

After this second step most of the vehicles are built up of just one single ellipse. Those vehicles which generate a highly fragmented foreground segmentation can be built up of more than one ellipse, usually this happens when the vehicle just enters the field of view. But after a couple of frames as the vehicle travels further away from the camera this reduces to one or two ellipses at most. One evident problem with the second pass is that two ellipses from two different vehicles could be forced to group if they are sufficiently close. This did not seem to be a major problem in those video sequences the system was tested on. But nevertheless it is a potential problem. The output from this step is the ellipses $\xi^*_i$.

3.5 Tracking

This section describes how the vehicles are tracked between frames. New models are spawned as soon as a new (large) ellipse appears at the location where vehicles enter
the deck, see starting area in Figure 1.5 on page 10. Ellipses are associated to vehicle models and the models are predicted forward in time by Kalman filters. Models that have ellipses associated to them over several frames are considered as true vehicles. Only true vehicles are counted and their length are estimated. Stationary ellipses are also identified and the background model is forced to incorporate them into the background. These steps will now be described in the same order as they are executed inside the system, this means that some steps will be referred to before they are explained. But first a little about the notation used in this section.

When referring to the ellipse of a vehicle model this means the largest ellipse in the model, sometimes it will be explicitly pointed out but not always. Each model can have one or more ellipses associated to it. The starting area is illustrated in Figure 3.8 on page 48 and the vehicle trajectory is also shown it this figure, all vehicles have a trajectory line that originates from a point \( P_0 \).

### 3.5.1 Motivation

Tracking is performed by matching ellipses to vehicle models, this makes it possible to associate fragmented objects to a vehicle (i.e. objects that are divided into one or more ellipses). Using the approximated distance between ellipse perimeters as distance measure gives a more accurate result for large ellipses than the distance between centers would have given. Using a Kalman filter to predict the vehicle position in future frames not only makes the matching more accurate but it also makes it possible to detect unexplained pixels. Unexplained pixels can help to indicate global illumination changes.

### 3.5.2 Stationary ellipses

The first step is to remove stationary ellipses. Stationary ellipses are caused by foreground regions that are stationary for several frames, usually caused by vehicles that have stopped. A stopped vehicle will keep getting segmented as foreground until it is incorporated into the background model, which can take several frames, and in the meantime they will cause problems to the foreground segmentation and connected component analysis. So to remove them the background model is forced to incorporate their pixels into the background. This is done by increasing the learning rate \( \alpha \) in the update equations for the cylindrical mixture model (compare to 3.8-3.11 on page 29) at stationary ellipses.\(^6\) This causes stationary objects to quickly be incorporated into the background.

The identification step consists of first noting the center of all ellipses in frame \( n \) (i.e. \( \mu_i \) from \( \xi_i \) \( \forall i = 1, \ldots, M \)). If ellipses from \( k \) successive frames all have centers within a circle of radius \( r \) from the center of any ellipse in the previous frame then the ellipse in frame \( k \) is called stationary.\(^7\) All pixels inside this ellipse will have their learning rate increased in the background model. These pixels will keep having

\(^6\) typically \( \alpha = 0.8 \)

\(^7\) typically \( r = 0.8 \)
the higher learning rate until they are classified as background after which they will retain their old learning rate. This effectively reduces the problem with stationary vehicles being foreground over too many frames, with this step they will only be foreground for roughly $k$ frames. Next it is time to match the ellipses to vehicle models.

### 3.5.3 Matching ellipses to vehicle models

The ellipses $\xi^*_i$ in frame $n$ given by the pixel association step will now be matched to existing vehicle models $\nu_k$ from frame $n - 1$, where $k$ is the number of the model. How the models are spawned will be explained in Section 3.5.6 but for the sake of the discussion lets assume that there are some vehicle models already spawned. Each vehicle model contains the following information at frame $n$

- A list of those ellipses in frame $n - 1$ that were assigned to it.
- A position which is the center of the largest ellipse in the model.
- A vehicle trajectory angle.
- An Age variable, indicating for how many frames this model has been alive.
- A unique ID.
- A Kalman filter which is added once the vehicle has left the starting area (SA).

The vehicle trajectory angle is the estimated travel direction of the vehicle, see Figure 3.8 on page 48. For an ellipse to be inside the starting area the connected component corresponding to this ellipse has to have at least one pixel inside this area.

**Match matrix** Ellipses in the image has to be matched to vehicle models, this is facilitated by a distance matrix consisting of the distances between predicted vehicle model positions and ellipses. To match ellipses and vehicle models a distance matrix is constructed. The elements of this matrix are the match-distances from each vehicle to each ellipse. Denote the match-distance between vehicle $\nu_r$ and ellipse $\xi^*_c$ by

$$d(\nu_r, \xi^*_c)$$

The actual distance measure used will be explained later. The distance matrix of size $N \times M$ can then be written as

$$D_{rc} = \begin{bmatrix}
  d(\nu_1, \xi^*_1) & d(\nu_1, \xi^*_c) & \ldots & d(\nu_1, \xi^*_M) \\
  d(\nu_r, \xi^*_1) & \ddots & \cdots & \vdots \\
  \vdots & \ddots & \ddots & \vdots \\
  d(\nu_N, \xi^*_1) & \ldots & d(\nu_N, \xi^*_c) & d(\nu_N, \xi^*_M)
\end{bmatrix}$$

(3.42)
**Match-distance**  The match-distance is calculated by first checking if the center of the ellipse is an advanced position, advanced means that the position is inside the funnel or cone as shown in Figure 3.8. See the appendix for exact calculations. This is based on the assumption that vehicles always move down the deck, an assumption that does not seem to be violated. This is especially useful when cars are trailing each other and prevents vehicles to be matched with ellipses behind them. If the center is not inside the funnel then the distance is set to $\infty$, otherwise the distance to the ellipse is calculated as follows:

**If** the vehicle is still in the starting area **then** calculate the distance between the largest ellipse $a$ in the vehicle model and the new ellipse $b$.

**else** calculate the distance between the predicted largest ellipse $a$ in the vehicle model and the new ellipse $b$.

*Predicted largest ellipse* means that the position of the ellipse has been predicted one step ahead using the Kalman filter, all other parameters of the ellipse are the same. The reason for not using the predicted ellipse in the first case is that vehicles that are partly inside the starting area does not have a Kalman filter associated to them and hence the position can not be predicted. The distance that is calculated is an approximation to the minimum distance between the perimeters of the two ellipses. So the distance is calculated as follows

**If** center of $b$ is inside $a$ **then** $d(\nu_r, \xi^*_c) = 0$

**else** $d(\nu_r, \xi^*_c) = "\text{Approximate min distance between perimeters of } a \text{ and } b"$

For details on how to calculate the approximate minimum distance between two ellipses see the appendix.

### 3.5.4 Final matching

Now the distance matrix (3.42) can be populated and the best matching ellipses can be found. An exhaustive search is performed over the matrix to find the first vehicle-ellipse pair $(r, c)$ (i.e. an element $d_{rc}$ in 3.42) where

$$ (r, c) = \arg \min_{r, c} d_{rc} < T_1 $$

(3.43)

and that satisfies the following condition

$$ T_2 < \Pi(r, c) < T_3 $$

(3.44)

where $\Pi(r, c)$ is the area-ratio between the largest ellipse in vehicle model $r$ and the new ellipse $c$, using the area ratio to guide the search prevents totally wrong matches to happen where the area difference is too large. The threshold $T_1$ is an upper-bound on the distance and $T_2$ and $T_3$ are lower- and higher-bounds for the
area ratio. Once a match \((r,c)\) is found the row \(r\) and column \(c\) of (3.42) are replaced with \(\infty\) to prevent these from being matched again. The list of ellipses for the matched vehicle model \(\nu_r\) is emptied and \(\xi^*_c\) is inserted. The search is then performed again and this is repeated until all vehicle models have been considered. Vehicle models that have not been matched at the end are removed. Call the set of vehicle models that matched an ellipse for \(\nu^*_i\).

Finally all \(\nu^*_i\) are checked for close un-matched ellipses, allowing the model to incorporate nearby ellipses. The approximate minimum distance between the perimeter of the ellipse in \(\nu^*_i\) and the un-matched ellipses from the previous step is calculated and the following rules are used:

- **If** \(\nu^*_i\) is in the starting area **and** distance \(< T_4\) **then** add this ellipse to \(\nu^*_i\)
- **else if** \(\nu^*_i\) is not in the starting area **and** distance \(< T_4/2\) **then** add this ellipse to \(\nu^*_i\)
- **otherwise** do nothing.

The reason for having different thresholds\(^9\) inside the starting area and outside is that inside the starting area ellipses can be further away from each other and still belong to the same real-world object. Thus \(T_4\) can be a little less restrictive here.

### 3.5.5 Updating matched vehicle models

Once all vehicles model are matched to an ellipse or removed, the new vehicle models \(\nu^*_i\) are updated

- The estimate of the vehicle trajectory angle is improved.
- The ID’s are propagated.

and the length is estimated if the vehicle is at the right position.

**Update of vehicle trajectory angle** The trajectory angle is indicated in Figure 3.8 and it is estimated as long as the vehicle model is inside the starting area. This is done by using a recursive average to update the (old) trajectory angle \(\theta_i\) with the new value \(\theta'\), calculated from the center of the largest ellipse in \(\nu^*_i\).

\[
\theta_i = \frac{\theta_i(n-1) + \theta'}{n}
\]  

(3.45)

where \(n\) is the age of the vehicle (i.e. the number of frames it has been supported by ellipses.) This angle is later used when initializing a Kalman filter for the model. Kalman filters are initialized and the length is measured as soon as the vehicle leaves the starting area or if any of the distances \(\min(\Delta x_1, \Delta x_2)\) or \(\Delta y\) is greater than \(T_6\), see Figure 3.8.

\(^8\) Chosen ad-hoc as \(T_1 = 26\) pixels, \(T_2 = 0.4\) and \(T_3 = 2.4\).

\(^9\) \(T_4 = 30\) pixels

\(^{10}\) \(T_6 = 20\) pixels
Figure 3.8. Illustration of the starting area, trajectory line and the distances $\Delta x_1, \Delta x_2$ and $\Delta y$. $\Delta x_1$ is the distance from the left edge of the image to the leftmost pixel in the ellipse, $\Delta x_2$ is the distance from the right edge of the image to the rightmost pixel in the ellipse and $\Delta y$ is the distance from the bottom edge of the image to the bottommost pixel in the ellipse. The angle $\theta_c$ determines the width of the cone, note the displacement of the cones center.

Initializing of the Kalman filter and length estimation Each vehicle model has a Kalman filter attached to it, which is used to predict the position of the vehicle in subsequent frames. They are attached at a certain point in time as describe before. For further details about the Kalman filters, see Section 3.6. The length is estimated at the same time as the filters are attached, see Section 3.5.8.

Propagation of ID All vehicles get a unique ID when they are created so that they can be counted. This ID is propagated for vehicle models.

3.5.6 Spawn new vehicle models

When all this have been done there can still be un-matched ellipses in $\xi^c$ and they are used to spawn new vehicle models. For an un-matched ellipse to spawn a new
vehicle model it has to be inside the starting area and have an area greater than \( A \) pixels.\(^{11}\) If this is the case than a new model is initialized with this ellipse and an initial trajectory angle is calculated using the ellipse center. Next the distance between this vehicle and any un-matched ellipses are calculated and those that are within \( T_s \)\(^{12}\) pixels are added to the model. This continues until no more vehicles can be spawned or there are no more un-matched ellipses. Only ellipses inside the starting area spawns new vehicle models, this makes sure that ellipses appearing in the middle of the image do not create new models. Illumination changes can cause ellipses to appear at new places in the image, and this effectively prevents them from being considered as new vehicles.

### 3.5.7 Update Kalman filters

Kalman filters for vehicle models are updated with the position of the new ellipse just associated to it. For further details see the section about updating the Kalman filter in Section 3.6.4 on page 53.

### 3.5.8 Length measurements

The length of the vehicle is approximated by the orthogonal projection of the ellipse axes endpoints onto the vehicle trajectory line as shown in Figure 3.9. Those two endpoints giving the longest projected distance are taken to be the endpoints of a line \( l = P_{\text{max}} \). Calculating the Euclidian distance of \( l \) in image coordinates will not give the accurate length of the vehicle so it has to be projected onto the ground plane as described in Section 3.7 on page 55, but before it is projected onto the ground plane it has to be moved so that its center is at the lowest axis endpoint. Moving the line to the axis endpoint having the lowest y-coordinate helps to prevent the length of high vehicles to be projected onto the extended parts of the ground plane, see Figure 3.9. The Euclidian distance of the endpoints in the ground plane can then be calculated and used as an estimate of the vehicle length, see (3.55) on page 55.

### 3.5.9 Discussion

Tracking can be improved by using more information when matching ellipses to vehicle models. For example, color information could be used to improve matching. Using the existing cameras this was not an option due to the lack of color information for vehicles in the shaded area, compare with images from the daylight sequence. The vehicles also seemed to lose color information when traveling further down the lanes, away from the camera. Lack of color information was a problem under some lighting conditions especially during sunny days as in the daylight sequence, the overcast sequence did not suffer from this problem.

\(^{11}\)\( A = 240 \)

\(^{12}\)\( T_s = 30 \)
The choice was made to perform tracking in image coordinates rather than in ground-plane coordinates because the low camera angle made it hard to project the position into the ground-plane. Vehicles traveling in the lane farthest away from the camera are seen from the side, while vehicles close to the camera are seen from behind or from above, this made it hard to project the position onto the ground plane in a consistent way.

**Figure 3.9.** Illustration of the projection of ellipse’s axis onto the trajectory line. The *extended* parts of the ground plane are also illustrated.
3.6 Kalman filter-based tracking

In this system Kalman filters are used to do one-step-ahead predictions of the position of vehicles in future frames. These predicted positions are used for matching ellipses to vehicle models and for creating update-masks for the background model. The update-masks are used for changing the learning rate at pixels where the vehicle is predicted to be, it also lowers the learning rate at unexplained pixels (i.e. pixels that are not predicted to be foreground but still are deemed to be foreground.)

To facilitate these predictions Kalman filters are used, one for each vehicle model. This section will describe the state space model of the Kalman filters and the update equations used. First an overview is given of the coordinate system used for the state space model. All tracking is performed in the image-plane not the ground-plane. The reason for not using the ground-plane for tracking is that vehicle appearances can be heavily distorted in the ground-plane, especially those belonging to vehicles at the lane farthest away from the camera. see Figure 3.14 on page 60.

3.6.1 Trajectory coordinate system

To simplify the state space model for the Kalman filters a new coordinate system is designed. This coordinate system makes it easier to formulate the state space model for tracking vehicles going in a straight line. Instead of using the \((x, y)\) coordinates in the image plane a new coordinate system is introduced, where the position is described by a point on a trajectory line and an orthogonal distance to the line. First designate the origin of the trajectory line by \(P_0 = (x_0, y_0)\),

![Figure 3.10. Trajectory coordinate system.](image)

\[ P_0 = (x_0, y_0) \]
see Figure 3.10, the angle of the trajectory line is called \( \theta_n \), the corresponding direction vector is \( n = [\cos \theta_n, \sin \theta_n]^T \), and \( r_d \) is the point where the line between \( P_m \) and the trajectory line intersects the trajectory line at right angles. The point \( P_m = (x_m, y_m) \) is a point in the coordinate system of the image, the same point in the new coordinate system is described by the distances

\[
\begin{align*}
t = \begin{cases} 
-\|r_d - P_0\| & \text{If } \theta_n - 90^\circ \leq \theta_r \leq \theta_n + 90^\circ, \\
\|r_d - P_0\| & \text{else.}
\end{cases}
\end{align*}
\]

and

\[
\begin{align*}
d = \begin{cases} 
\|P_m - r_d\| & \text{If } 180^\circ - \theta_n \leq \theta_r \leq \theta_n, \\
-\|P_m - r_d\| & \text{else.}
\end{cases}
\end{align*}
\]

This means that \( r_d \) is the projection of \( P_m \) onto the line passing through \( P_0 \) and \( (P_0 + n) \), and \( d \) is the orthogonal distance from this line to \( P_m \). For exact calculations of \( t \) and \( d \) see the appendix.

### 3.6.2 State space model

In this new coordinate system, \( d \) will symbolize the perpendicular deviation of the vehicle position from the trajectory and \( t \) will be the distance along this trajectory. This means that the velocity of the vehicle only affects \( t \) and not \( d \), although \( d \) will get smaller as the vehicle moves away from the camera. The states of the Kalman filter can then be described by \( t \), \( d \) and the velocity \( v_t \) along the trajectory. This gives the state vector

\[
x(n) = [t, d, v_t]^T
\]

(3.48)

Assume that the vehicle position is governed by the following process

\[
\begin{align*}
x(n + 1) &= F \cdot x(n) + G \cdot w(n) \\
y(n) &= H \cdot x(n) + v(n)
\end{align*}
\]

(3.49) (3.50)

where (3.49) is the process equation and (3.50) is the measurement equation. Estimates of a value \( x \) will be written as \( \hat{x} \). Future values are denoted by \( n + 1 \) and current values \( n \). The matrices are

\[
F = \begin{bmatrix} 1 & 0 & T \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\
G = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \\
H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}
\]

where \( T = -5 \) represents the initial velocity towards \( P_0 \). This means that the states \( x(n) \) evolves according to

\[
\begin{align*}
t(n + 1) &= t(n) + T \cdot v_t(n) \\
d(n + 1) &= d(n) + w_1(n) \\
v_t(n + 1) &= v_t(n) + w_2(n)
\end{align*}
\]
where an acceleration change has been modeled as zero mean white noise \( w_1(n) \) affecting the velocity. The orthogonal direction is also affected by zero mean white noise \( w_2(n) \).

Equation (3.50) says that the two states \( t \) and \( d \) are measured and that these measurements also are affected by zero mean white noise. The covariance of the process noise \( R_1 \) and measurement noise \( R_2 \) is set to

\[
R_1 = E\{w(n)w(n)^T\} = \begin{bmatrix} 0.3 & 0 \\ 0 & 1 \end{bmatrix}
\]

(3.51)  

\[
R_2 = E\{v(n)v(n)^T\} = \begin{bmatrix} 10 & 0 \\ 0 & 22 \end{bmatrix}
\]

(3.52)

these values were chosen because they gave reasonable tracking results when the filter was tested in MATLAB. These values basically means that the measurements are not trusted and that the model (3.49) is believed to be accurate. The reason for believing this is that the vehicle trajectory is assumed to be estimated fairly correctly and that the vehicle position will not deviate too much from this line. The state \( d \) has a higher degree of uncertainty associated with it compared to \( t \) in the state process noise covariance. The covariance of the measurement noise says that the state \( d \) is measured with a higher degree of uncertainty than the state \( t \). Tracking was found to be quite robust without tuning these values.

### 3.6.3 Initialization of the filter

When initializing the Kalman filter, one can use different techniques depending on how the update equations are written, in this case the error covariance \( Q(n) \) and an initial state vector \( x(n) \) are needed. The initial error covariance is always given by

\[
Q_0 = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

(3.53)

and \( x(n) \) will be calculated from the position and the estimated trajectory angle of the vehicle model.

### 3.6.4 Time and measurement update

Each time a new measurement \( y(n) \) arrives the update equations are calculated. First the time update:

\[
\dot{x}(n + 1|n) = F\hat{x}(n|n)
\]

\[
y(n + 1|n) = H\hat{x}(n + 1|n)
\]

\[
P(n + 1) = FQ(n)F^T + GR_1G^T
\]
and then the measurement update:

\[
L(n+1) = \mathcal{P}(n+1)^T \mathcal{H}(n+1)^T [\mathcal{H}(n+1)^T \mathcal{H} + R_2]^{-1}
\]

\[
\hat{x}(n+1|n+1) = \hat{x}(n+1|n) + L(n+1)(y(n+1) - \mathcal{H}\hat{x}(n+1|n))
\]

\[
Q(n+1) = \mathcal{P}(n+1) - \mathcal{P}(n+1)^T [\mathcal{H}(n+1)^T \mathcal{H} + R_2]^{-1} \mathcal{H} \mathcal{P}(n+1)
\]  \hspace{1cm} (3.54)

These equations are recursive which means that the output at time \( n \) will be used as input at time \( n + 1 \) the only new value used is the measurement \( y(n) = [t, d]^T \). Note that the measurement is not an \((x, y)\) position but a \( (t, d) \) position.

### 3.6.5 Discussion

In this filter, the measurements are not trusted and the process model is assumed to explain the evolution of the states accurately. To get an idea of how the filter behaves see Figure 3.11(a), this figure shows the observed positions \( \circ \) and the predicted positions \( \times \). In Figure 3.11(b) the measurements are stopped after a while and the future positions \( \cdot \) are predicted using the process equation (3.49). If an acceleration term had been used in the process equation the predictions would not have been equally spaced along the vehicle trajectory as they are now. An acceleration was not modeled because the predictions would not be more than one frame ahead so an acceleration term was not motivated. The predictions would still be accurate enough but as one can see in Figure 3.11(b) predictions for several frames ahead are equally spaced along a straight line. So if predictions for more than one frame are needed a model of the (de-)acceleration would be necessary.

![Figure 3.11](image1.png)

(a) Measurements \( \circ \) and filter predictions \( \times \).  \hspace{1cm} (b) Measurements \( \circ \), filter predictions \( \times \) and predictions after the measurements has stopped \( \bullet \).

**Figure 3.11.** Figure (a) shows the predicted position and measurements, in figure (b) the measurements stops and the filter predicts the rest of the positions. The trajectory line is indicated by an arrow.
3.7 Metric rectification

Besides counting the number of vehicles, the system is supposed to measure the length of vehicles. If the camera would have been placed above the deck facing downwards, the length measurements could have been made directly in image coordinates and then re-scaled to give the length in meters. Unfortunately this is not the case. The camera gives a perspective distorted image of the ground-plane (i.e. the deck of the ferry) and this distortion has to be removed before any length measurements can be made. The process of removing this perspective distortion is referred to as metric rectification. Metric rectification means that the ground-plane is transformed into a rectified plane from which the following measurements can be made:

- angles
- length ratios
- and in our case lengths.

In [LZ98], Liebowitz and Zisserman describes a method to metrically rectify an image-plane from:

- A known angle in the image-plane.
- Two equal unknown angles in the image-plane.
- A known length ratio in the image-plane.

two of these constraints will together be enough information to determine the projective transformation matrix $H$, which maps an image point $x$ to the point $x'$ in the ground-plane.

$$x' = Hx$$  \hfill (3.55)

All vectors in this section will be homogeneous vectors, for an excellent book on geometry in computer vision see [HZ00]. The projective transformation $H$ can be decomposed into a concatenation of three matrices [LZ98, HZ00], the similarity transformation $H_S$, the affine transformation $H_A$ and a projective transformation $H_P$

$$H = H_S H_A H_P$$

Details about how to calculate $H_P$ and $H_A$ are given in [LZ98] and are restated here for clarity.
Figure 3.12. Measured lengths.

Lengths measured from the image, true lengths are shown and the deviation from the true values are written in parenthesis.

3.7.1 From projective to affine

First,

$$H_P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

needs to be determined. In this matrix the homogeneous coordinates of the vanishing line $l_\infty = [l_1, l_2, l_3]^T$ of the plane has to be found. Parallel lines on the world plane intersect at points on the vanishing line $l_\infty$ in the image see Figure. So two sets of parallel lines in the image give two points on the vanishing lines, which thus is determined.

If two points $x$ and $x'$ are known, then the line through these can be written as

$$l = x \times x'$$

Given four points in the image-plane, which are points on two parallel lines in the ground-plane, then these two lines can be written as

$$l_1 = x_1 \times x_1' \quad \quad l_2 = x_2 \times x_2'$$
and their intersection is

\[ x = l_1 \times l_2 \]

Calculating the intersections \( x \) and \( x' \) of two sets of parallel lines give the vanishing line as

\[ l_\infty = x \times x' \]

Plunging the values of \( l_\infty \) into (3.56) gives the projective transformation matrix.

### 3.7.2 From affine to metric

The affine transformation matrix

\[
H_A = \begin{bmatrix}
\frac{1}{\gamma} & -\frac{\alpha}{\gamma} & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

is determined by \( \alpha \) and \( \beta \). The geometric interpretation of these parameters is that they specify the image of the circular points, see [LZ98]. This means that the affine transformation can be found by determining the intersections of two circles, the point of intersection will be \( (\alpha, \beta) \).

The parameters of the circles can be determined by using two of the following three constraints. Note that all points in this section are assumed to have been transformed from projective to affine space using the matrix \( H_p \).

From [LZ98]:

1: **known angle** Given an angle \( \theta \) on the ground plane, between the lines imaged as \( l_a \) and \( l_b \). Then \( \alpha \) and \( \beta \) lie on the circle with center

\[
(c_\alpha, c_\beta) = \left( \frac{(a + b)}{2}, \frac{(a - b)}{2} \cot \theta \right)
\]

and radius

\[
r = \left| \frac{(a - b)}{2 \sin \theta} \right|
\]

where

\[
a = -\frac{l_{a2}}{l_{a1}} \\
b = -\frac{l_{b2}}{l_{b1}}
\]

are the directions of the lines.
2: **Equal (unknown) angles** Suppose the angle in the ground plane between two lines imaged with direction \(a_1, b_1\) is the same as that between two lines imaged with direction \(a_2, b_2\). Then \(\alpha\) and \(\beta\) lie on the circle with center at

\[
(c_\alpha, c_\beta) = \left( \frac{a_1b_2 - b_1a_2}{a_1 - b_1 - a_2 + b_2}, 0 \right)
\]

and squared radius

\[
r^2 = \left( \frac{a_1b_2 - b_1a_2}{a_1 - b_1 - a_2 + b_2} \right)^2 + \frac{(a_1 - b_1)(a_1b_1 - a_2b_2)}{a_1 - b_1 - a_2 + b_2} - a_1b_1
\]

(3.59)

(3.60)

3: **Known length ratio** Suppose the length ratio of two non-parallel line segments is \(s\) on the ground-plane. The first line consists of the imaged end-points \((x_{12}, y_{12}), (x_{11}, y_{11})\) and the second \((x_{21}, y_{21}), (x_{22}, y_{22})\). Introduce the distances \(\Delta x_n = x_{n1} - x_{n2}\) and \(\Delta y_n = y_{n1} - y_{n2}\). Then \(\alpha\) and \(\beta\) lie on the circle with center

\[
(c_\alpha, c_\beta) = \left( \frac{\Delta x_1\Delta y_1 - s^2\Delta x_2\Delta y_2}{\Delta y_1^2 - s^2\Delta y_2^2}, 0 \right)
\]

and radius

\[
r = \left| \frac{s(\Delta x_2\Delta y_1 - \Delta x_1\Delta y_2)}{\Delta y_1^2 - s^2\Delta y_2^2} \right|
\]

By using two of the previous constraints and searching for the intersection \((\alpha, \beta)\) of these circles gives \(\alpha\) and \(\beta\). Plunging these values into (3.57) gives the affine transformation matrix.

### 3.7.3 Metric rectification

Given the concatenated transformation \(H\), the image-plane can be transformed into the ground-plane using (3.55) on page 55. To be able to do Euclidian measurements in the image the scaling \(s\) in the similarity transform

\[
H_S = \begin{bmatrix}
sR & t \\
0^T & 1
\end{bmatrix}
\]

has to be determined, where \(R\) is a rotation matrix and \(t\) a translation vector. Note that the rotation and translation will not affect the length measurement so they need not be determined. The scale is determined by measuring a known distance in the rectified image and then re-scale calculated distances so they match the real distance (i.e. length). An advantage of this metric rectification is that orthogonal lines are very common in man-made structures and they are quit easy to find in an automatic way (e.g. using a hough-transform) and thus the rectification process can be made almost automatic.

58
3.7.4 Implementation details

The implementation is based on code written by Peter Kovesi at the University of Western Australia [Kov]. Figures 3.13 and 3.14 were produced by MATLAB-code written by Kovesi. Figure 3.14 shows the rectified image and in this image one can see that the parts of the extended ground plane, illustrated in Figure 3.8 on page 48, are heavily distorted. This is the reason why length measurements are only accurate inside the ground plane.

![Figure 3.13. Example of constraint-lines used when calculating R. The vanishing line is determined by the lines P1P2, P3P4, P5P6 and P7P8. The constraints used are: known angle between S1S2 and S2S3, known length ration between S1S2 and S2S3, and known length ratio between R1R2 and R2R3.](image)

59
Figure 3.14. Rectified image, the difference ± of the estimated lengths are shown and with the true lengths in parenthesis.
Chapter 4

Workflow and implementation details

In this chapter the workflow of the project will be roughly outlined and then details about the implementation will be explained, particularly the pipeline. Specifications of the different hardware and software that was used during the project is also given.

4.1 Workflow

During the development of this system most of the algorithms were first implemented in MATLAB and then ported to C++, this required some extra work but the algorithm development is easier in MATLAB. Those algorithms that were to complex too execute in an interpreted language like MATLAB were written in C++ and then interfaced with MATLAB by the external MATLAB-interface/API. This made it possible to take advantage of the speed of C++ programs while enjoying the superior algorithm design capabilities of MATLAB.

Background estimation was first implemented in MATLAB but it turned out to be too slow and was re-written in C++ and then compiled to a mex-file which was interfaced with the rest of the system which ran in MATLAB. After most of the algorithms had been partly studied and implemented in MATLAB everything was ported to C++ to make it possible to build a prototype. All I/O operations like loading video sequences had been handled by MATLABs built-in functions and these had to be replaced by similar functions in C++. Two alternative solutions were possible, one was to use an existing library to read AVI-files from disk and the second was to implement a filter graph in DirectShow/Direct X. Since the prototype was to function in real-time and capture frames from a frame grabber the choice was made to put extra time on implementing a DirectShow filter graph.

DirectShow filter graphs can easily be modified to receive input from a file or from a frame grabber thus making the transition from an off-line to an on-line system easier. At the end, the system was never modified to capture frames on-line, though the possibility still exists. The filter graph is used to pre-fetch a number of frames
from disk into memory and then the pipeline requests one frame at a time. When the memory is empty the graph pre-fetches another couple of frames. This means that the system could be modified to work on-line with little effort, the pipeline would not see the difference as long as it get a frame each time it requests one. In an on-line system the filter graph would grab the frames from the capture card and send them to the pipeline on request instead of from a file.

The OpenCV/ipl libraries were used to some extent, mainly for handling image-data using the iplImage and cvImage classes, this made it easy to plot debugging information. The only other processing performed by OpenCV/ipl was the erosion and dilation operations and some logical operations on images.

4.2 Program flow

The gui and the processing pipeline are executed in two different working threads, and they are synchronized using simple events. The final system was implemented completely in C++ and the data was given as an AVI-file which was used as input. For an illustration of the complete pipeline in the final system see figure 4.1. This figure illustrates the program flow in the processing pipeline.

The class cPipelineManager controls the flow of information between the various steps in the pipeline. When execution begins the cPipelineManager requests a frame from the class cSequenceProcessor. It does not matter if this class grabs a frame directly from a video camera or from a file, the pipeline manager wont notice, which makes it easy to change the system to an online one if needed. Currently frames are grabbed from a file. This frame is then sent further down the processing pipeline and the pipeline manager handles passing of information between the various steps, effectively hiding the data from the different classes. This makes the pipeline highly modifiable. In the final steps one can see that the information extracted from the frames is simply a number of ellipses along with the current vehicle models. By examining the number of unique vehicle models one easily get the number of vehicles that has been present along with their estimated lengths.

If the pipeline is supposed to be run on-line, frames will have to be dropped. As the pipeline is designed now it can not be stopped if one frame takes to long to process, this means that if some frames take to long to process quite a lot of frames might be dropped.

4.3 Equipment and data

Video sequences were acquired by connecting a digital camcorder to the surveillance cameras onboard the ferry and then transferred to a computer. All sequences were stored as AVI-movies which is the video format used inside the system. Because of the large amount of data the lossy AVI-compression had to be used. All videos had a resolution of 265 × 200 pixels and the frame rate was roughly 30 frames per second with 24-bit RGB colors (i.e 8 bits per channel). Figure 1.2 shows the four different
video sequences that were used. Sequences in Figure 1.2(a), 1.2(b) and 1.2(d) were all more heavily compressed than the one in Figure 1.2(c).

As one can see in the table below the daylight sequence is not as heavily compressed as the other ones and has a higher data rate. The daylight sequence also exhibits less noise and the chromaticity channel did not seem to be as heavily compressed as for the other video sequences.
Specifications of video sequences

<table>
<thead>
<tr>
<th></th>
<th>Dusk</th>
<th>Overcast</th>
<th>Daylight</th>
<th>Night</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression</td>
<td>high</td>
<td>high</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>Original resolution</td>
<td>256 × 200</td>
<td>256 × 200</td>
<td>256 × 200</td>
<td>320 × 240¹</td>
</tr>
<tr>
<td>Frames</td>
<td>1839</td>
<td>4313</td>
<td>2254</td>
<td>1676</td>
</tr>
<tr>
<td>Color depth</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Frames per second</td>
<td>30.0</td>
<td>30.0</td>
<td>25</td>
<td>30.0</td>
</tr>
<tr>
<td>Megabytes per second</td>
<td>0.234</td>
<td>0.235</td>
<td>3.66</td>
<td>0.437</td>
</tr>
<tr>
<td>Total size</td>
<td>14.3 MB</td>
<td>33.7 MB</td>
<td>330 MB</td>
<td>24.4 MB</td>
</tr>
</tbody>
</table>

**Video camera**  On-board the ferries a surveillance camera of type JVC-TK-C701EG were used for video acquisition. Specifications of this camera follows:

**Specifications of TK-C701EG**

<table>
<thead>
<tr>
<th></th>
<th>1/3&quot; Interline-Transfer CCD (with complementary color filter)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pickup device</td>
<td>1/3&quot; Interline-Transfer CCD (with complementary color filter)</td>
</tr>
<tr>
<td>Number of effective pixels</td>
<td>290,000 (500 H × 582 V)</td>
</tr>
<tr>
<td>Pickup area</td>
<td>4.8 mm (H) × 3.6 mm (V)</td>
</tr>
<tr>
<td>Video S/N</td>
<td>50 dB (AGC OFF)</td>
</tr>
<tr>
<td>Minimum required illumination</td>
<td>0.75 lux (F1.2 AGC ON) – Video level 25% 1.5 lux (F1.2AGC ON) – Video level 50%</td>
</tr>
<tr>
<td>Iris control</td>
<td>Video iris/DC iris</td>
</tr>
<tr>
<td>White balance</td>
<td>ATW/Manual</td>
</tr>
</tbody>
</table>

**Computer equipment**  During the development an Intel based computer was used with the following specifications:

**Specifications of computer hardware**

<table>
<thead>
<tr>
<th></th>
<th>Intel pentium 4 hyper-threading 2.81 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel pentium 4 hyper-threading 2.81 GHz</td>
</tr>
<tr>
<td>Motherboard</td>
<td>Gigabyte 8IPE100</td>
</tr>
<tr>
<td>Memory</td>
<td>Kingston 2 × 512 MB DIMM 400 MHz, Single channel</td>
</tr>
</tbody>
</table>

and the software used was:

**Specifications of computer software**

<table>
<thead>
<tr>
<th></th>
<th>Windows XP Professionals SP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Windows XP Professionals SP2</td>
</tr>
<tr>
<td>C++ compiler</td>
<td>Microsoft visual studio .NET 2003</td>
</tr>
<tr>
<td>Other software</td>
<td>Matlab student version release 13</td>
</tr>
<tr>
<td></td>
<td>WinEdt</td>
</tr>
<tr>
<td></td>
<td>RAD Video Tool Bink Version 1.6c Smacker Version 4.1f</td>
</tr>
<tr>
<td></td>
<td>Intel Image Processing Library 2.5</td>
</tr>
<tr>
<td></td>
<td>OpenCV</td>
</tr>
</tbody>
</table>
All development in C++ was made in Visual Studio .NET 2003. The result of this project was a complete program written in C++ that implemented the processing pipeline described in the theory section. Everything from reading video frames off disk to counting vehicles and measuring lengths was implemented.
Chapter 5

Conclusion and evaluation

The final system is robust to modest illumination changes, some lighting variations exists in the sequences used, though not as severe as those caused by moving clouds, and the system handled them quite well. During overcast days the system has no problem to track vehicles. Under these circumstances the only source of problem is slow-moving objects like trucks and busses, which tend to get incorporated into the background model, but fine-tuning the update rate helps to solve this problem. Sometimes vehicles get under-segmented (i.e holes in the segmentation) this was mainly a problem for grey or dark colored vehicles. This is caused by the low intensity of these pixels. For low intensity RGB-pixels the difference between different hues of the same color tends to be hard to determine.

During sunny days with the sun high in the sky, shadows and the automatic gain control\(^1\) of the camera caused most of the problems. Shadows are always a problem but they are handled quite well by this system if thresholds are fine-tuned. Removing the shadows does however have an impact on the foreground segmentation of vehicles. In this system pixels that seem to belong to shadowed parts of the background are removed. Unfortunately this means that vehicles with dark colors or with a color similar to the background will be under-segmented. This was the problems with vehicles driving in the shaded are in the lane farthest away from the camera as in Figure 1.2(c) on page 5. In this lane vehicles were in constant shadow and looked more or less like shadows. The reason that vehicles still were recognizable by the background model was that the top of the vehicles were reflecting sunlight thus making these pixels brighter.

The automatic gain control (AGC) caused problems during bright sunlight. When the sun was reflected of vehicles directly into the camera the intensity suddenly dropped and then raised again for 1-2 seconds, giving large areas of new foreground pixels from the background model, see frames in Figure 5.1. As one can see in frames from the daylight sequence vehicles in the shaded lane has less color information, only bright colored vehicles stick out from the background. Vehicles in this lane often failed to spawn new vehicle models because they generated foreground components.

\(^1\)I think this was the cause of the problems.
of too small size. It would be interesting to see if some kind of filter could be placed in front of the camera to alleviate some of these problems (polarization filter?). Another possibility is to investigate another type of camera. Maybe the spawning of new models can be changed such that it is more sensitive in the shaded lane? In night sequences brake lights from vehicles causes problem, major parts of the

![Images of camera frames showing changes in intensity.](image)

**Figure 5.1.** Figure (a) shows the frame when the AGC has increased the intensity and figure (b) is the normal intensity the second after.

scene is colored red by brake lights from vehicles. This problem can be reduced by better lighting. Better lighting will also reduce the effect of headlights lighting up the ferry-deck.

One of the cameras used on the ferry did not produce saturated colors, colors were *washed out*. I think this was the result of the particular location of this camera, at its current location it points directly into the sun sometimes.

**High traffic** Sequences of fully loaded ferries have not really been tested. One sequence exists in which the lanes are almost filled up with vehicles. Testing the system on this sequence showed that the removal of stationary ellipses greatly improved tracking. By tuning the time it takes for ellipses to be considered as stationary, the tracking performance increased because new vehicles did not merge with old ones in the foreground masks. If stationary ellipse is not removed stopped vehicles interferes with the connected component analysis.

**Possible improvements** This system could be improved in some ways.

- First of all a more thoroughly study of night scenes has to be done. The night scene it was tested on now produced good results but it is not sure how general these results are, especially during high traffic.
- If the initial background model is learnt when the sun is obscured by clouds the results of tracking are unknown. This could be investigated.

- Improve the handling of changing light conditions, use gradient/edge information?

- Improve the system by setting the thresholds automatically so they are optimal for the prevailing lighting. This is done manually in this system.

- Measure the lengths in a smarter way. Maybe there is some better way than to project the ellipse axes onto the trajectory line.

- Is it possible to position the cameras in a more optimal way?

- Does a filter in front of the camera have an impact on tracking performance during sunny days?

- Is it possible to use artificial light at night to reduce the effect of headlights and brake-lights? Artificial light could create a problem with shadows.

- Use dual CPUs and multi-threading to speed up the processing pipeline.

- Investigate if the single instruction multiple data (SIMD) capabilities of new processor can be used to speed-up frame processing.

5.1 System performance

Some rough performance measures will be given for the system. There were to little time to do extensive testing of performance on different video sequences but these values give an idea of what performance values that could be expected. All sequences in Figure 1.2 on page 5 were tested. These are the results:

<table>
<thead>
<tr>
<th>Type of light</th>
<th>Night</th>
<th>Dusk</th>
<th>Daylight</th>
<th>Overcast</th>
</tr>
</thead>
<tbody>
<tr>
<td>True number of vehicles</td>
<td>8</td>
<td>13</td>
<td>11</td>
<td>27</td>
</tr>
<tr>
<td>Counted vehicles</td>
<td>8</td>
<td>14</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Measured vehicles</td>
<td>8</td>
<td>13</td>
<td>12</td>
<td>28</td>
</tr>
<tr>
<td>Missed vehicles</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

True number of vehicles is the actual number of vehicles in the sequence, counted vehicles is the final count given by the system, measured vehicles are the number of vehicle lengths measured and missed vehicles are the number of vehicles that did not spawn a vehicle model. In daylight, the system misses vehicles because one of the lanes are shadowed, giving poor color information for dark vehicles. One should note that some thresholds\(^2\) in the algorithms were adjusted manually before each

\(^2\)threshold for classifying a pixel as highlighted background or shadowed background.
sequence was tested, to get optimal performance, this is something that has to be done automatically in a final system.

The processing time of the pipeline was also evaluated which might be interesting as the system was developed to some extent with real-time requirements in mind. When doing the calculations every fourth frame was dropped, this did not have a negative impact on tracking, tracking is actually improved by dropping frames because of the aperture problem. The execution time of the pipeline for each frame was measured, loading of frames were not included in the timing. In Figure 5.2 the execution time for each frame is shown.

![Graphs of processing time for different conditions](image)

**Figure 5.2.** Execution times for each frame, x-axis shows the frame number and the y-axis the time in seconds.

A histogram of the execution times can be found in Figure 5.3. In the overcast sequence one bus and one truck enters the ferry, this is the reason for the large execution times for some of the frames in this sequence. Large vehicles give an increase in execution time because of more foreground pixels. For an idea of how
Figure 5.3. Histogram of the execution times, x-axis shows the execution time interval and y-axis shows the number of frames. Mean and median values are shown in the caption of each figure.

The system behaves in action see Section D, where a couple of screen-shots are located. There were not enough time to gather length data to validate the length estimations for all camera angles but the overcast sequence should have somewhat accurate lengths. Estimated lengths has been written beside the projected lengths in the Figures found in Appendix D.
References


Appendix A

Point-ellipse distance

A.1 Aligning semi-major axis with y-axis

The way the equations are stated at the paragraph below, one needs to align the semi-major axis with the y-axis. This is done as follows: The point (in the image)

\[ p' = p - \mu \]

then the point is rotated, which is equivalent to rotating the ellipse so it gets its semi-major axis aligned with the y-axis. The rotation matrix is given by the angle \( \theta \)
between the semi-major axis of the ellipse centered at position \((0, 0)\) and the x-axis. Because the semi-major axis should be rotated back to the x-axis, the negative angle \(-\theta\) is used. Thus

\[
T = \begin{bmatrix}
\cos(\theta) & -\sin(-\theta) \\
\sin(-\theta) & \cos(-\theta)
\end{bmatrix}
\]

multiplying with \(T\) gives the rotated point \(p_r\)

\[p_r = p'T\]

now the point just needs to be scaled so that the position is relative to an ellipse with the equation (A.1), the scaling is

\[p_s = 1/d \cdot p_r\]

\(d\) is the scaling of the semi-major and semi-minor axis, see (3.39) on page 42. The closest point \(p_c\) can then be calculated as explained in paragraph below, using the length \(\|s_1\|\) as \(a\) and \(\|s_2\|\) as \(b\), compare with (3.39) on page 42.

Once the closest point is found all transformations are reversed to get the point \(p_c\) back in image coordinates again, like this

\[p_{ci} = T^{-1} \cdot d \cdot p_c + \mu\]

### A.2 Point-ellipse distance

This is essentially the description given in [DE]. It is sufficient to solve this problem when the ellipse is centered at the origin and is axis-aligned with the major axis on the x-axis. Other ellipses can be rotated and translated to such an ellipse and the distance can be measured in that system. Let \((u, v)\) be the point in question. Let

the ellipse be

\[
\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1 \tag{A.1}
\]

with \(a < b\). The closest point \((x, y)\) on the ellipse to \(u, v\) must occur so that \((u - x, v - y)\) is normal to the ellipse. An outward pointing ellipse normal is

\[
\frac{1}{2} \nabla \left( \left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 - 1 \right) = \left(\frac{x}{a^2}, \frac{y}{b^2}\right) \tag{A.2}
\]

The orthogonality condition is

\[
(u - x, v - y) = t \left(\frac{x}{a^2}, \frac{y}{b^2}\right) \tag{A.3}
\]

for some value \(t\). Equations (A.1) and (A.3) are used to determine the value \(t\). Attention can be restricted to the case \(u \geq 0\) and \(v \geq 0\). Points in other quadrants
may be handled by symmetry. For example, the distance from $(-u, v)$ to the ellipse is the same as the distance from $(u, v)$ to the ellipse. Also by symmetry, if $(u, v)$ is in the first quadrant, the closest ellipse point $(x, y)$ is in the first quadrant.

Consider when $u > 0$ and $v > 0$. Equation (A.3) implies that $u - v = tx/a^2$ and $v - y = ty/b^2$ for some $t$. Solving yields

$$x = \frac{a^2 u}{t + a^2} \text{ and } y = \frac{b^2 v}{t + b^2}$$

(A.4)

From the symmetry of the ellipse the closest point $(x, y)$ should be in the first quadrant, so we need $x > 0$ and $y > 0$. These constraints force $t > -a^2$ and $t > -b^2$. Since $a > b$, the total constraint on $t$ is $t > -b^2$.

Replacing (A.3) in the ellipse equation (A.1) yields

$$F(t) = \left( \frac{au}{t + a^2} \right)^2 + \left( \frac{bv}{t + b^2} \right)^2 - 1 = 0$$

(A.5)

which is a rational function of $t$. The roots of $F(t)$ provide the candidate points of being closest to $(u, v)$.

The convexity of $F$ on its domain, namely that $F''(t) > 0$, makes Newton’s method an ideal numerical method for locating the root. Given an initial guess $t_0$, the Newton iterates are

$$t_{n+1} = t_n - \frac{F(t_n)}{F'(t_n)}, \quad n \geq 0$$

It is important to choose an initial guess for which the method converges. Newton’s method has an intuitive geometric appeal to it. The next value $t_{n+1}$ is computed by determining where the tangent line to the graph at $(t_n, F(t_n))$ intersects the $t$-axis. The intersection point is $(t_{n+1}, 0)$. If we choose an initial guess $t_0 < \bar{t}$, the tangent line to $(t_0, F(t_0))$ intersects the $t$-axis at $(t_1, 0)$ where $t_0 < t_1 < \bar{t}$. An initial guess $t_0 > \bar{t}$ leads to a new iterate $t_1 < \bar{t}$, but potentially $t_1 < -b^2$ which puts it outside the domain of interest, namely $(-b^2, \infty)$. To avoid this potential problem, it is better to choose an initial guess to the left of the root. Any $t_0$ for which $F(t_0) > 0$ will work. In particular, one can choose $t_0 = bv - b^2$, in which case $F(t_0) = [au/(bv - b^2 + a^2)]^2 > 0$.

### A.3 Newton’s method

When using Newton’s method to find the root of (A.5) the zero-tolerance of the root is set to 0.1 and the maximum number of iterations to 9.

### A.4 Ellipse-ellipse distance

The distance between two ellipses $a$ and $b$ is approximated by the distance between the closest points on each ellipse. The idea is to first calculate the closest point $x_a$
on ellipse $a$ to the center of ellipse $b$ and then calculate the closest point $x_b$ on ellipse $b$ to the center of ellipse $a$. The final distance is then approximated as the Euclidean distance between these two points

$$\|x_a - x_b\|$$
Appendix B

Minimization of $\beta_t$

The quantity

$$\beta_t = \arg \min_z (I_t - zE_t) \cdot (I_t - zE_t)$$

should be minimized with respect to $z$. First re-write the equation as

$$J(z) = (I_{t1} - zE_{t1})^2 + (I_{t2} - zE_{t2})^2 + (I_{t3} - zE_{t3})^2$$

take the derivative of $J(z)$

$$\frac{\partial J(z)}{\partial z} = -E_{t1} \cdot 2(I_{t1} - zE_{t1}) - E_{t2} \cdot 2(I_{t2} - zE_{t2}) - E_{t3} \cdot 2(I_{t3} - zE_{t3})$$

set $\partial J(z)/\partial z$ to zero

$$0 = -E_{t1} \cdot 2(I_{t1} - zE_{t1}) - E_{t2} \cdot 2(I_{t2} - zE_{t2}) - E_{t3} \cdot 2(I_{t3} - zE_{t3})$$

and solve for $z$ gives

$$z = \frac{2E_{t1}I_{t1} + 2E_{t2}I_{t2} + 2E_{t3}I_{t3}}{2E_{t1}^2 + 2E_{t2}^2 + 2E_{t3}^2} \quad \text{(B.1)}$$

$$z = \frac{E_{t1}I_{t1} + E_{t2}I_{t2} + E_{t3}I_{t3}}{E_{t1}^2 + E_{t2}^2 + E_{t3}^2} \quad \text{(B.2)}$$

which can be written as

$$z = \frac{E_t \cdot I_t}{E_t \cdot E_t}$$
Appendix C

Derivation of $t$ and $d$

If $P_0 = (x_0, y_0)$ is the origin of the trajectory line and $\mathbf{n} = (\cos \theta_n, \sin \theta_n)$ is the direction, see Figure 3.10 on page 51, then the trajectory line can be written as

$$L_0(t) = P_0 + t \cdot \mathbf{n}$$

The measured point $P_m$ lies on the line

$$L_m(t) = P_0 + t(P_m - P_0)$$

and the vector from $P_0$ to $P_m$ is

$$\mathbf{r}_m = P_m - P_0$$

The orthogonal projection of $\mathbf{r}_m$ on $L_0(t)$ is given by

$$\text{Proj}_n \mathbf{r}_m = \frac{\mathbf{r}_m \cdot \mathbf{n}}{\|\mathbf{n}\|^2} \cdot \mathbf{n}$$

this gives us the length $t$ of the projection as

$$t = \|\text{Proj}_n \mathbf{r}_m\| = \frac{|\mathbf{r}_m \cdot \mathbf{n}|}{\|\mathbf{n}\|}$$

and the orthogonal distance $d$ as

$$d = \|\mathbf{r}_m - \frac{\mathbf{r}_m \cdot \mathbf{n}}{\|\mathbf{n}\|^2} \mathbf{n}\|$$

Now the sign of $d$ and $t$ has to be determined. This is done by first re-writing $\mathbf{n} = [n_1, n_2, 0]^T$ and $\mathbf{r}_m = [r_{m1}, r_{m2}, 0]^T$ as vectors having dimensionality 3 and
then taking the cross product \( z = r_m \times n \) between them, the sign of the third component of \( z \) will determine if \( r_m \) is above or below \( n \)

\[
d = \begin{cases} 
  +d & \text{if } n_1 r_{m2} - r_{m1} n_2 < 0, \\
  -d & \text{else.}
\end{cases} \tag{C.1}
\]

The sign of \( t \) is determined in a analogous manner using \( n_d = [\cos(\theta_n - 90^\circ), \sin(\theta_n - 90^\circ), 0]^T \) and \( r_m = [r_{m1}, r_{m2}, 0]^T \)

\[
t = \begin{cases} 
  +t & \text{if } n_d1 r_{m2} - r_{m1} n_d2 > 0, \\
  -t & \text{else.}
\end{cases} \tag{C.2}
\]
Appendix D

Illustrations of tracking performance

Screen-shots from the system in action. For information about the symbols in the screen-shots see Figure 1.5 on page 10. See page(s) after this one for images.
Figure D.1. Tracking in overcast sequence.
Figure D.2. Tracking in overcast sequence continued.
Figure D.3. Tracking in night sequence.
Figure D.4. Tracking in night sequence continued.