Strukton
Rail

University of Twente
Enschede - The Netherlands

Railroad crossing video monitoring

The development and comparison of algorithms

Masters Thesis
in Applied Mathematics
by

R.J.N. de Jong

June, 2009

Chairman:
Dr. Ir. G. Meinsma

Supervisors:
Prof. Dr. A.A. Stoorvogel
Dr. Ir. F. van der Heijden
Ing. R. Westenberg
Ir. E.J. van Beek

University of Twente
Faculty of Electrical Engineering, Mathematics and Computer Science
Department of Applied Mathematics
Mathematical Systems and Control Theory

Summary

At a railroad crossing several events take place. Usually these events are moving objects, barriers going down if a train is approaching and going up if the train has passed. This is escorted by flashing lights and the sound of a warning bell. Strukton Rail Consult possesses an application (VOM, Visual Railroad Crossing Monitoring) which is able to observe these events. The goal of observing these events is to detect dangerous situations. For instance when a car is stationary at the railroad crossing or (slalom) driving through the barriers while these are closed. Another dangerous situation is when the barriers do not reach to 0°.

Research is done on the algorithms used by the application. The algorithms needed a more efficient approach, since in future the algorithms are to be implemented in a warning bell at several railroad crossings. So the application only transmits a signal if a dangerous situation occurs. Besides that, some algorithms had to be extended. In the current situation it is not possible to differentiate between several types of traffic, for instance a car or a cyclist.

Motion is usually detected by the relation (coherence) between a current frame and its background. If something has changed this suggets motion. So the first step is to keep track of a background. This can be done by the previous frame or by taking some more frames into account, for instance by averaging over previous frames or adapting the background by a stepsize or by exponential adaptation. The exponential adaptation appears to be the best method.

The (adapting) background is used to detect moving objects. The calculations are limited to measuring loops. In these measuring loops the coherence between the current frame and its background is determined. Several methods for the coherence are discussed, namely detection based on threshold, sum over the measuring loop, averaging and correlation. The correlation of the current frame and its background appears to be the best technique to detect moving objects.

If the objects are moving it is desirable to differentiate between types of traffic. Several properties of a moving object are discussed and used to type the object. For instance, a car is longer than a cyclist and has a higher speed. Besides that a car is wider and thus the indicator for motion is larger and rises faster. With a combination of these properties an algorithm is created which is able to classify the moving objects for the available sample video's.

The indicator for motion is also used to check for dangerous situations. If the barriers are closed, usually no motion is detected on the railroad (unless when a train is passing) and the coherence is great since nothing is going on. If motion is detected at this point this might be because a object crosses the railroad while it is not allowed. The development of the indicator is used to detect if objects are stationary at the railroad crossing. If an object is moving, the indicator fluctuates. While a stationary object generates a more constant value. Besides that the value can be used to detect solid objects at the railroad crossing such as the andreas cross.

To observe the path of the barriers, various methods have been tested to detect the barrier in a frame. The position of the barrier yields a set of data points. By the set of these points the best line through these points can be determined. To do this the least squares, total least squares and L1-approximation are examined. By the slope of the best line, the angle of the barrier is known.

Besides the visual aspects also the audio of a warning bell is studied. Before the barriers close the warning bells turn on. It is desirable to check the status of the warning bells and how many warning bells are turned on. To do this, the fast Fourier transformation is applied to see in which frequency spectrum the sound of a warning bell lies. Realtime checking is done using the fast Fourier transformation or a bandpassfilter. It appears that the technique by the Fourier transformation is the fastest method to detect the status of the bells.

Preface

At the end of my project I look back on a challenging time at Strukton Rail Consult. Thanks to this project I got the oppurtunity to work on a real situation of applied mathematics. During my traineeship from September 1st till December 12th I got the occassion to become used to MATLAB® and VOM. From half December I continued the research for my final project.

I would like to thank all my colleagues for the pleasent time here at Strukton Rail Consult, espacially Ir. Elbert van Beek and Ing. Roel Westenberg for their guidance and their feedback during the project. My supervisors from the University of Twente I would like to thank for their confidence in my research. On a final note I would like to say to my first supervisor Dr. Ir. Gjerrit Meinsma:
Tige tank foar de begelieding yn de ôfrinde tyd en foar de goede ried dy't Jo my joun hawwe.

Roel de Jong
June, 2009

List of Symbols

Symbol	Description	Page
m	Height of the video frame (pixels)	19
M	Height of the measuring loop (pixels)	29
n	Width of the video frame (pixels)	19
N	Width of the measuring loop (pixels)	29
c	Color channel usually $c \in \{1,2,3\}$ or $c \in \{R,G,B\}$	19
NoF	Number of frames in the videofile	19
k	Discrete time to denote frame number $k \in [1,2,\ldots,NoF]$	19
X_k	Frame k short notation for $X(1:m,1:n,1:3,k)$	19
B_k	Background for frame k	19
f_{video}	Video sampling frequency (video samples per second)	20
z	Step size for adapting background	21
β	Speed with which background is adapted	24
t_{set}	Settling time	25
k_{set}	Discrete settling time	25
R_k	Indicator for the amount of motion	29
I	Difference between the background and the current frame	29
θ	Threshold value	29
γ	Correlation coefficient	35
F	Filterorder for averaging over time	44
$\hat{\beta}$	Speed with which R_k is adapted to its previous value	44
L_{mot}	Minimum value of R_k for motion detection	49
D_k	Absolute difference between R_k and R_{k-1}	51
d	Difference limit for the stagnation check	52
W	Maximum value of the stagnation counter to give warning	52
j	Vehicle number	60
V_j	Speed of vehicle j	60
t_j	Time that $R_k \geq L_{\text{mot}}$ for vehicle j	61
L_{speed}	Fixed limit to tell vehicle type based on vehicle speed	60
L_{diff}	Fixed limit to tell vehicle type based on entering speed	60
L_{car}	Fixed limit to tell vehicle type based on amount of fill	61
L_{length}	Fixed limit to tell vehicle type based on the length	61
α	Angle of the barrier	71
l	Number of data points for barrier detection	73
δ	Length of the barrier	81
s_h	Sound sample h	90
S_p	Fourier transformation of s for frequency p	89
f_{audio}	Sound sampling frequency (sound samples per second)	92
L_{bell}	Fixed limit to tell if the bell is 'on' or 'off'	92

List of Figures

List of Tables

Contents

Chapter 1

Introduction

To have an idea of the current work done for VOM (Visual Railroad Crossing Monitoring), Section 1.1 describes the current situation for VOM. Then in Section 1.2 the assignment is explained. During the research some problems were encoutered which are described in Section 1.2, also an indication is given on how these problems can be approached.

1.1 Current situation

Strukton Rail Consult (division Systems and Software Engineering), located in Hengelo, has developed a railroad crossing safety system. This system contains of a computer application which is able to analyse video images of a railroad crossing. The application reads a video file and presents this as video to a user on a screen. The user is then able to see if any anomalies (unwanted situations) occur at the railroad crossing.

The application can also read a live video-stream, MJPEG (Moving JPEG), instead of a recorded video-file. This stream is acquired from a camera which is located at a railroad crossing in Koekange (Drenthe, The Netherlands). The camera sends images to a central server at Hengelo, where they are processed by the application.

The purpose of the application is to detect unwanted situations and give an alarm if such a situation occurs, for instance when a non moving (stationary for more than a couple of seconds) object is blocking the railroad crossing.

1.2 Assignment

The final goal of the system is to implement the application at a system attached to the bell of a railroad crossing, see Figure 1.1. The concept of the implemented situation is given in Figure 1.2. Then the application only transmits a signal to the server if a dangerous situation at the railroad crossing occurs. At present this is not possible because the algorithms, which are used by the application, are not efficient enough to be applied at the location.

The main assignment for this project is to research the currently used algorithms [9]. The goal is to make the algorithms currently in use more efficient, meaning faster, and also extend their functionality. The extension of the algorithms is needed because the current algorithms are, for instance, not able to make a distinction between cyclists and cars. This is one of the desires for the final application. Besides that the current algorithms can not detect non moving objects blocking the railroad crossing.

The development and comparison of the algorithms is done with MATLAB®. Information about image processing in MATLAB is found in [3, 4, 6]. For image and video processing in general see [2].

Figure 1.1: Warning bell of railroad crossing with camera (the black circle above the logo)

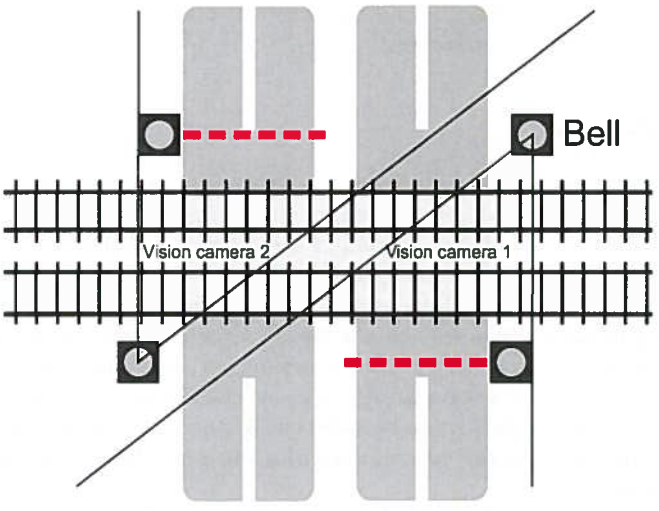


Figure 1.2: Outline of the implementation; the dotted lines denote the barriers of a railroad crossing and the solid lines denote the cameraview

Problem formulation As explained in Section 1.2 there were two main problems taken into account in this project. The first one and at the same time the most important goal is to speed up the algorithms or to make them work more efficient. This is done by adapting them or totally rewriting the algorithms. The second goal is about implementing new functionalities at the algorithms. The work needed for this project can be divided using the MoSCoW method (**M**ust have, **S**hould have, **C**ould have, **W**ould like to have but not right now).

Must have: detection of trains and traffic, path of the barrier, functionality of the lamps, intensity of the traffic

Should have: stationary check (check if vehicles are non-moving on the railroad crossing), slalom driving (driving around the barriers while they are closed), driving through the red light, differentiate types of traffic (for instance cyclists, cars, etcetera)

Could have: checking the solid objects (e.g. warning cross), sound of the bell

Would like to have: discerning types of trains.

Structure Figure 1.3 gives an overview of the application and shows the flow of the data. The camera receives an image, with this image a background can be created. The coherence between these two images creates an indicator for motion. This indicator is used to detect dangerous situations, solid objects, several types of traffic and the barrier. Since the camera has an installed microphone the audio can be analysed.

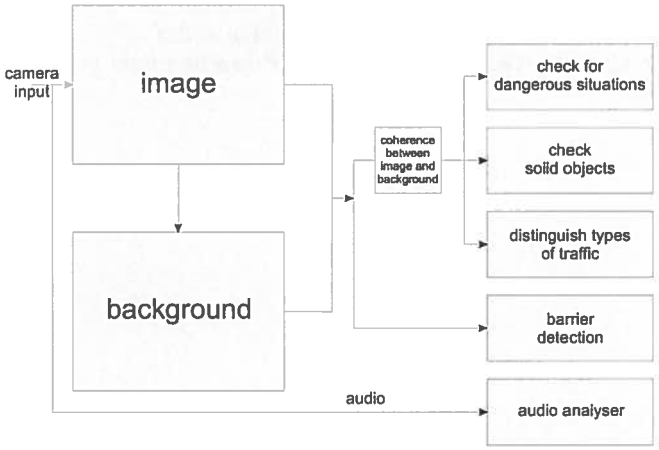


Figure 1.3: Overview of the application (total)

Motion is usally detected by comparing two images, the background and the current frame. The easiest way is to set the background as the previous frame but better is to adapt the background to the current situation more slowely. Chapter 2 describes the technique for the background adaptation. Several algorithms are discussed in this chapter for the adaptation of the background.

If the background is adapted to the current situation, the coherence between the background and the current frame can be determined. The coherence of these images is determined in Chapter 3. This yields an indicator for motion. One of the problems which is discussed, deals with the luminious intensity. If the sun starts shining this may lead to false positive detections. Say traffic is detected while the railroad crossing is empty. This chapter describes several techniques to evade these false positive detections.

An extension for the indicator of motion is the stagnation of vehicles at the railroad crossing. A stationary vehicle leads to dangerous situations. Other dangerous situations are for instance when vehicles slalom around the closed barriers. The check for these dangerous situation is discussed in Chapter 4. Problem to this check is how to distinguish a vehicle which is stationary and a long vehicle like a lorry which is driving by. Related to stagnation of vehicles is the presence of the

andreas cross and fenches. These objects are also stationary and may be detected by the same technique.

If the vehicles are moving, it is desirable to know which type of vehicle is driving by. So the indicator for motion is used to differentiate between several types of traffic, which is described in Chapter 5. The current application is not able to discern the several types of traffic [9]. So this algorithm has to be written. Several methods are tested which describe the properties of the traffic. For instance a cyclist has a lower speed in comparison to a car. A problem which occured is the detection of two in sequence or side by side driving vehicles. These vehicles might be detected as one vehicle.

The image and background can also be used to detect the angle of the barrier. By vandalism the barrier may describe another path, which must be detected. Chapter 6 describes some techniques to detect the position of the barrier (and how to determine the angle of the barrier). Problems which occur during the detection are the vehicles which drive by on the background or moving trees. These objects cause noise that disturbs the measurements. Several solutions for this problems are also discussed in Chapter 6.

Besides the video stream also the audio stream of the camera is researched. This is done to investigate the sound of the bell, see Chapter 7. The techniques used for this research are the Fourier transformation and bandpassfiltering. In the final concept two cameras are attached at the railroad crossing. This enables detection of multiple sound bells. The second camera can also be used to detect the other side of the railroad crossing.

Every chapter first describes some algorithms and later on in the chapter the discussed algorithms are tested and compared. The final chapters describe the conclusion of this research and recommandations for future work. The develloped algorithms are tested to several sample video's. These sample video's are denoted by the text "sample video" and a number. Appendix B contains a legend for these video's. The same holds for "sample audio".

References are placed between brackets, [0], and can be found in the bibliography.

Chapter 2

Background adaption

Detecting moving objects is done by comparing two images. If a moving object is on some position at frame X_{k-1} and in the next frame X_k the object is at another position, then movement can be detected by comparing the pixels of two frames. More generally, to dectect motion the current frame, X_k, is compared to a background, B_k, which is some function of the frames in the past. Figure 2.1 shows the overview of the application.

2.0 Defining data

A frame is denoted as X_k and contains $m \times n$ pixels. Here m denotes the height of the frame and n denotes the width of the frame. The variable k denotes the number of the frame, $k \in [1, 2, \ldots, NoF]$, where NoF stands for the **N**umber **o**f **F**rames. Every pixel in X_k at position (i, j), $(i = 1 : m, j = 1 : n)$, usually consist of three color channels, $c \in \{R, G, B\}$. This is numerically denoted as $c \in \{1, 2, 3\}$. Every color channel the pixel $X(i, j, c, k)$ is an element of $\{0 \ldots 255\}$. For the background adaptation these values can be non-integer and they are rounded to integers for display. If a pixel of X_k is $\{0, 0, 0\}$ for $\{R, G, B\}$, the pixel is black. Otherwise if $\{R, G, B\}$ are $\{255, 255, 255\}$ the pixel is white. A background, B_k, is defined with the same dimensions and structure as X_k.

Remark 2.0.1. For images with only one colorchannel there holds that, if $c = 0$ the pixel is black and if $c = 255$ the pixel is white.

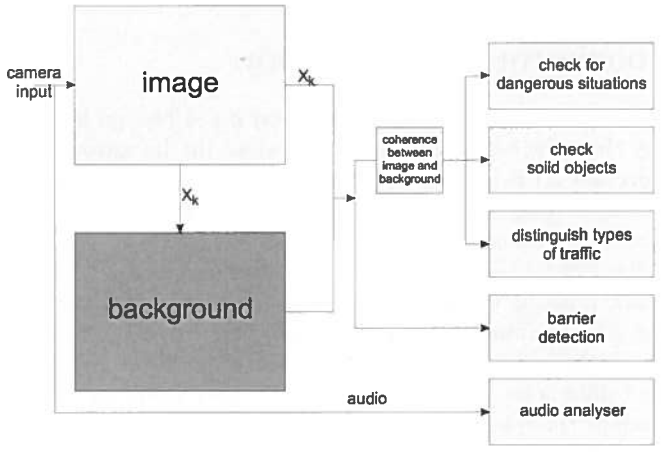


Figure 2.1: Overview of the application (background)

Given the background, the current image can be compared by for instance taking the absolute difference of these two,

$$|X_k - B_k| = \sum_c |X(i,j,c,k) - B(i,j,c,k)|. \tag{2.1}$$

In almost every video the images are updated through a sampling frequency. This is denoted by f_{video} and is usually 24 Hz.

2.1 Fast background adaptation

The most easy way to see motion is by taking the previous frame, X_{k-1}, as the background,

$$B_k := X_{k-1},$$

to which the absolute difference of X_k and B_k is taken,

$$|X_k - X_{k-1}|.$$

Here $|\ldots|$ means elementwise absolute value.

Figure 2.2 displays two frames from sample video 1, X_{209} and X_{210}. Since the current frame, X_k, does not differ much from the background, B_k $(= X_{k-1})$, the difference is mostly zero which results in black pixels.

Due to the sampling frequency the difference between X_k and X_{k-1} is very small if an even colored car is driving by. Which means that only the contours of the car remain visible, as can be seen in Figure 2.2c.

(a) X_{209} (b) X_{210} (c) $|X_{210} - X_{209}|$

Figure 2.2: Simple motion detection

2.2 Slow background adaptation

Intuitively it is clear that the best result is achieved if the background is an empty road. When a car is driving by then the car is high lighted against the background entirely. To see this the same video as in Section 2.1 is loaded with the same frame (X_{210}). The only difference is that the background is not frame X_{209} but frame X_1 (see Figure 2.3a), the result of the absolute difference is shown in Figure 2.3c. Something remarkable is the truck at the upper right corner of the figure. In reality this truck is not there any more. Because of the fact that the background does not change the truck remains visible in $|X_k - B_k|$ for all k. The same happens if the sun starts shining. Then the current frame, X_k, becomes brighter while the background, B_k, remains the same. So the absolute difference, $|X_k - B_k|$ becomes larger. If the absolute difference is a measure for motion, a larger difference, as a result of the sun, may result in a false positive detection. So it is desirable to adapt the background to new situations, but not as fast as $B_k = X_{k-1}$. There are a couple of possibilities to do this, namely;

(a) X_1 (b) X_{210} (c) $|X_{210} - X_1|$

Figure 2.3: Extended motion detection

- Average over frames (see Subsection 2.2.1)
- Step size by pixel (see Subsection 2.2.2)
- Exponential adaptation (see Subsection 2.2.3)

2.2.1 Average over frames

One of the possibilities is to set the background as the average over the last h frames.

$$\bar{B}_k = \frac{1}{h} \sum_{l=k-h+1}^{k} X_l. \tag{2.2}$$

A disadvantage of this technique is that it is calculation intensive (that means consumes a lot of time). The number of operations for this formula is $h \times (3mn)$ (the dimension of the matrix is $3mn$). The bigger h will be, the more operations that are needed and the longer the application takes.

2.2.2 Step size by pixel

Another approach for slow background adaptation is by introducing a step size (z). For every pixel (i, j) in the current frame, X_k, take the difference between the previous background, B_{k-1}, and the current image. The new background is the old background plus the maximum between the negative step size and the minimum of the difference and the step size,

$$\hat{B}_k = \hat{B}_{k-1} + \max(-z, \min((X_k - \hat{B}_{k-1}), z)). \tag{2.3}$$

This technique is shown in Algorithm 2.1 and is used in the current application [9].

The working of Algorithm 2.1 is explained in the following example.

Example 2.2.1. Choose a background which is even white (every pixel has a value of [255, 255, 255]). Now a black area (every pixel in this area has a value of [0, 0, 0]) is entering the background. The difference in this area is everywhere [255, 255, 255]. If z is set to 10, the background becomes [245, 245, 245]. This process continues until the background is fully adapted. If the black area has gone before the background is adapted, the process continues in the other direction.

This algorithm requires $4 \times (3mn)$ operations.

2.2.3 Exponential adaptation

This technique is based on the exponential function. The new background consists of the previous background plus a factor β times a correction,

$$\check{B}_k = \check{B}_{k-1} + \beta\,(X_k - \check{B}_{k-1}) \qquad \check{B}_0 := X_1, \tag{2.4}$$

Algorithm 2.1: Determine new background with step size [Subsection 2.2.2]

Input: X_k and $\hat{B}_{k-1}$
Output: $\hat{B}_k$
for $i = 1 \ldots m$ **do**
  for $j = 1 \ldots n$ **do**
    if $X(i,j,c,k) - \hat{B}(i,j,c,k-1) \geq z$ **then**
      step1 $= z$
    else
      step1 $= X(i,j,c,k) - \hat{B}(i,j,c,k-1)$
    end
    if *step1* $\geq -z$ **then**
      step2 $=$ step1
    else
      step2 $= -z$
    end
    $\hat{B}(i,j,c,k) = \hat{B}(i,j,c,k-1)$+step2
  end
end

Algorithm 2.2: Determine new background with exponential factor [Subsection 2.2.3]

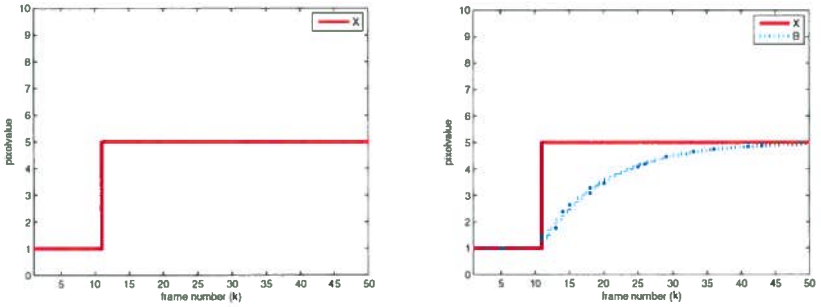
Input: $X(i,j,c,k)$ and $\check{B}(i,j,c,k-1)$
Output: $\check{B}(i,j,c,k)$
for *i=1..m* **do**
  for *j=1..n* **do**
    $\check{B}(i,j,c,k) = \check{B}(i,j,c,k-1) + \beta\left(X(i,j,c,k) - \check{B}(i,j,c,k-1)\right)$
  end
end

with $\beta \in [0, 1]$. The number of operations required by this technique described in Algorithm 2.2 is $3 \times (mn3)$.

To see what happens, take a look at a pixel (i, j). Suppose that the pixel value (total amount of red, green and blue),

$$\text{pixel value} = \sum_{c=1}^{3} X(i, j, c, k), \tag{2.5}$$

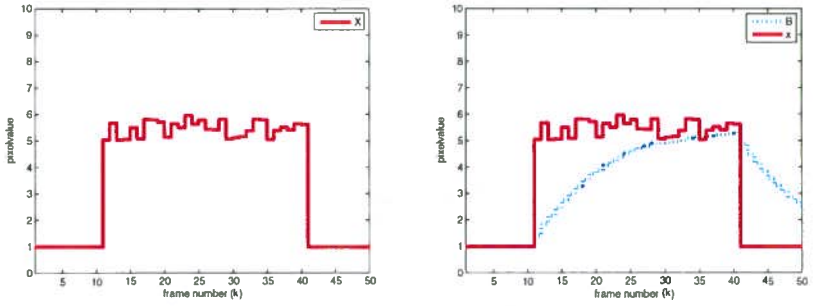
is one and at frame X_{10} the pixel value rises up to five then the background, $\check{B}_k$, follows it with a exponential factor. Figure 2.4 shows this in a graph. The red solid line (shown in Figure 2.4a and 2.4b) is the pixel value of the image on position (i, j) and the blue dotted line (shown in Figure 2.4b) is the pixel value of the background. As one can see the background adjusts to the pixel value exponentially fast.



(a) Pixel value (b) Pixel value and background

Figure 2.4: Simulation of the Algorithm 2.2, β=0.2

For this example the pixel value looks a bit like the step function. In reality this value is not that straight as in Figure 2.4, it fluctuates around a value and decreases after some time. A more realistic reproduction of the pixel value and the adapting background are shown in Figure 2.5.



(a) Pixel value (b) Pixel value and background

Figure 2.5: More realistic simulation of Algorithm 2.2, β=0.2

The greater the factor β, the faster the background adjusts. The β chosen in Figures 2.4 and 2.5 of 0.2 is not a good value. Because after some k the difference between the pixel and the background is almost zero. Zero difference results in that no motion is detected while there still might be objects moving at the railroad crossing. Choosing the right value β is not trivial, as is discussed in Section 2.3.

2.2.4 Comparing the algorithms

As the values for m and n are usually 320 and 240, it is clear that the technique from Subsection 2.2.1 requires the most time. Furthermore Algorithm 2.2 is somewhat faster than Algorithm 2.1. To test this, a function is written, in MATLAB, which compares the calculation time of Algorithm 2.1 and 2.2, using a sample video. The results are shown in Table 2.1. The values at the bottom of the table are average values of ten measurements. The results show that Algorithm 2.2 is approximately two times faster compared to Algorithm 2.1 for this particular sample video. The sample video contains (only) 295 frames. So the same comparison is performed on a lager video ($NoF = 2184$) and the results were similar. Algorithm 2.2 is still approximately two times faster compared to Algorithm 2.1, see Table 2.2.

Table 2.1: Operation times of Algorithms 2.1 and 2.2 in seconds ($NoF = 295$)

	Time in seconds		
index	Algorithm 2.1 (p.22)	Algorithm 2.2 (p.22)	Proportion
1	1.8191	0.8758	2.0770
2	1.7711	0.8762	2.0212
3	1.8061	0.8714	2.0728
4	1.8242	0.8689	2.0995
5	1.8244	0.8713	2.0939
6	1.8289	0.8702	2.1017
7	1.8148	0.8701	2.0857
8	1.7385	0.8692	2.0001
9	1.8146	0.8705	2.0845
10	1.8011	0.8705	2.0691
average	1.8043	0.8714	2.0705

The reason that Algorithm 2.2 is almost two times faster in comparison with Algorithm 2.1 can be explained by the fact that it requires fewer operations. But this proportion is $\frac{4}{3}$. The reason for the factor 2 is explained by the structure of the algorithms. Algorithms 2.1 takes for every number four operations (two **for** loops and two **if** statements). Otherwise, Algorithm 2.2 needs for every number two operations (two **for** loops). The fraction between these two explains the factor 2 in the proportion of Table 2.1 and 2.2.

2.3 Choice of β

As discussed in Subsection 2.2.3 a smaller β results in slower background adjustments. Thus the road in B_k is empty for a longer time and therefore highlights a car better in $|X_k - B_k|$. But the

Table 2.2: Operation times of Algorithms 2.1 and 2.2 in seconds ($NoF = 2184$)

	Time in seconds		
index	Algorithm 2.1 (p.22)	Algorithm 2.2 (p.22)	Proportion
1	12.3952	6.4391	1.9250
2	12.3971	6.7176	1.8455
3	12.4198	6.5711	1.8901
4	12.3782	6.4383	1.9226
5	12.3680	6.4937	1.9046
6	12.3698	6.4356	1.9221
7	12.3741	6.4423	1.9208
8	12.4430	6.4209	1.9379
9	12.4960	6.4212	1.9461
10	12.6167	6.3944	1.9731
average	12.4258	6.4774	1.9183

value of β must not be too small, because then a rise of luminious intensity is seen as motion. To see the effect of different values of β, a measuring loop is placed onto sample video 1 as shown in Figure 2.6. With this loop (rectangle) it is possible to measure the mean pixel value for the pixels in this loop. This technique is applied for various values of β and the results are shown in Figure A.1 in Appendix A. The results lead to the conclusion that a smaller β is better. This results in better peaks, which indicates the difference between motion and no motion.

Figure 2.6: X_{25} of sample video 1 with measuring loop (rectangle)

A smaller β denotes that the background takes longer to adapt to the current image. But when β is 0, then the background does not change and remains the first frame. This may result that rotations of the sun are denoted as motion. So it is better to define the β dependent on the settling time.

Definition 2.3.1. The settling time (t_{set}) is defined as the elapsed time for which the original difference between B_k and X_k is less than 1% of $(B_0 - X_0)$. The discrete settling time, k_{set}, equals $k_{\text{set}} = t_{\text{set}} \times f_{\text{video}}$.

The t_{set} and β are correlated according to the following theorem.

Theorem 2.3.2. *Given the video sampling frequency, f_{video}, and the settling time, t_{set}, β in Algorithm 2.2 approximately equals*

$$\beta \approx \frac{\ln(100)}{f_{video} t_{set}}. \tag{2.6}$$

□

Proof. Suppose that the pixel value at position (i, j) of X_1 starts at zero (and so the pixel value of $\check{B}_0 = 0$). When $k = p$ define $X_k(i, j) := X_p(i, j) = 1$. Then (2.4) becomes:

$$\begin{array}{rl} \check{B}_k = \check{B}_{k-1} + \beta\left(X_k - \check{B}_{k-1}\right), & \check{B}_0 = 0 \\ \check{B}_p = \check{B}_{p-1} + \beta\left(X_p - \check{B}_{p-1}\right), & \check{B}_0 = 0 \\ \check{B}_p = (1-\beta)\,\check{B}_{p-1} + \beta, & \check{B}_0 = 0 \\ \check{B}_p - (1-\beta)\check{B}_{p-1} = \beta, & \check{B}_0 = 0 \end{array} \tag{2.7}$$

This can be solved by the technique of difference equations. The homogeneous solution yields:

$$\begin{array}{rcll} \check{B}_p - (1-\beta)\check{B}_{p-1} & = & 0 & \\ \lambda^p - (1-\beta)\lambda^{p-1} & = & 0 & \\ 1 - (1-\beta)\frac{1}{\lambda} & = & 0 & \\ \lambda & = & (1-\beta) & \\ \check{B}_p^{\text{hom}} & = & h(1-\beta)^p, & h = \text{constant} \end{array} \tag{2.8}$$

The particular solution can be taken equal to:

$$\check{B}_p^{\text{part}} = 1 \tag{2.9}$$

combining (2.8) and (2.9) gives a solution for $\check{B}_p$ which is equal to $\check{B}_k$:

$$\check{B}_k = 1 + h(1-\beta)^k, \qquad \check{B}_0 = 0$$

Introducing the initial condition, $\check{B}_0 = 0$, gives a solution for h:

$$\check{B}_0 = 0 \rightarrow \check{B}_k = 1 + h(1-\beta)^0 \rightarrow h = -1$$

Combining this results in

$$\check{B}_k = 1 - (1-\beta)^k. \tag{2.10}$$

The background converges exponentially to X_k, so it takes ∞ frames for B_k and X_k to be exactly the same. X_k was set to 1, so $B_{k_{\text{set}}}$ has to be at least 0.99. Then the original difference is approached to 1%.

$$\begin{aligned} 0.01 &= (1-\beta)^{k_{\text{set}}} \\ \ln\left(\frac{1}{100}\right) &= k_{\text{set}} \ln(1-\beta) \\ -\ln(100) &\approx -k_{\text{set}}\beta \\ k_{\text{set}}\beta &\approx \ln 100 \\ \beta &\approx \frac{\ln 100}{k_{\text{set}}} \\ &\approx \frac{\ln 100}{f_{\text{video}} t_{\text{set}}} \end{aligned}$$

■

This is okay as long as $f_{\text{video}} \cdot t_{\text{set}} > 20$, f_{video} is usually 24 Hz so (2.6) can be used. With Theorem 2.3.2 it is possible to define a good β.

Example 2.3.3. Take t_{set} at thirty seconds then β will be

$$\beta = \frac{\ln 100}{30 \cdot 24} \approx 0.0064$$

Henceforth, all calculations are done with

$$\beta := 0.001$$

corresponding to a settling time of

$$0.001 = \frac{\ln 100}{t_{\text{set}} \cdot 24} \rightarrow t_{\text{set}} := \frac{\ln 100}{24 \cdot 0.001} \approx 192 \text{ seconds}$$

Conclusion

To detect motion a background is needed such that it can be compared with the current image. The background can be derieved in different ways. One way is by taking the background as the previous image, $B_k := X_{k-1}$. A disadvantage of this method is that only the edges of a moving vehicle remain visible. This leads to few detection points and attendantly a larger probability for incorrect detection (vehicles are present while they are not detected).

A better way is setting the background as the first image, $B_k := X_1$. This highlights the moving object much better which results in more detection points with a higher proballity of good measurements. A disadvantage of this method is, that it is very sensitive to changes in luminous intensity. This leads to detection while there are no moving objects at the railroad crossing.

So a better method is to adapt the background to the current situation. One way to make this happen is by defining the background as the average over a couple of images. On this way the background adapt itself to the new situation. A disadvantage of this method is the required number of operations, as shown in Subsection 2.2.1.

A faster way to adapt the background is using the stepsize method as described in Subsection 2.2.2. This method is also used in the current application [9]. An advantage of this method is that the rate with which the background is adapted can be adjusted. This allows finetuning and thus better detection.

By the research done it was concluded that using the exponential adaptation, see Subsection 2.2.3, is even almost two times faster in comparison with the stepsize method. Both techniques are described in Algorithms 2.1 and 2.2. From now on the background, B_k, is calculated according to the exponential adaptation (Algorithm 2.2) i.e.,

$$B_k := \check{B}_k.$$

The rate of adaption for the background, β, can be calculated using the settling time, t_{set}. The relation between these two variables is described in Theorem 2.3.2.

Chapter 3

Motion detection

As described in Chapter 2 (and shown in Figure 3.1) motion can be detected by comparing the current frame, X_k, and a background, B_k. In Chapter 2 an algorithm was found to adapt the background by using the current image. This chapter discusses several methods for comparing X_k and B_k, see the overview of the application in Figure 3.1. All methods have in common that a parameter, R_k (as indicator), is adapted each time a new frame is acquired. A larger R_k means a higher probability for detection of motion. The calculation of R_k is restricted to measuring loops. The size of the measuring loops is usually denoted as $M \times N$.

3.1 Algorithms for detection of movement

3.1.1 Threshold by pixel

This section explains the method currently in use by the application [9]. The currently used technique is based on using a threshold, θ. For every pixel check if the pixel value (2.5) is more than θ, that is,

$$I(i,j,c,k) := |X(i,j,c,k) - B(i,j,c,k)| \tag{3.1}$$

$$I_\theta(i,j,k) := \begin{cases} 0 & \text{if } \sum_{c=1}^{3} I(i,j,c,k) \leq \theta \\ 255 & \text{if } \sum_{c=1}^{3} I(i,j,c,k) > \theta. \end{cases} \tag{3.2}$$

So if the pixel value of the absolute difference between the image and the background at position (i,j) is above theta the pixel is set to white otherwise it is white. Figure 3.2a shows the original

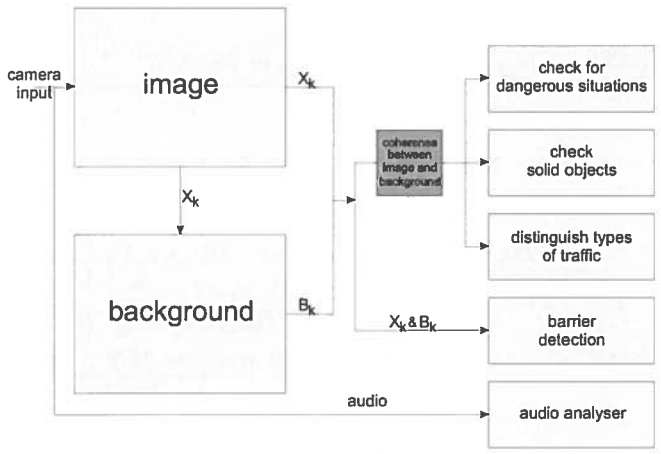


Figure 3.1: Overview of the application (compare background and image)

(a) $\sum_{c=1}^{3} I(i,j,c,210) := |X_{210} - B_{210}|$ (b) $I_{100}(i,j,210) \quad \forall i,j$ (c) $I_{250}(i,j,210) \quad \forall i,j$

Figure 3.2: Sample video 1 with and without threshold filtering

difference, $\sum_{c=1}^{3} I(i,j,c,k)$, without threshold filtering. Figure 3.2b and 3.2c show the difference, $I_\theta(i,j,k)$, with filtering respectively for $\theta = 100$ and $\theta = 250$.

By summing I_θ over all pixels (i,j) in a measuring loop, like the one in Figure 2.6, and dividing by the maximum value of the measuring loop $(255MN)$ it returns a value between zero and one which is an indicator for motion,

$$R_k^{\mathrm{TH}} := \frac{\sum_{i,j} I_\theta(i,j,k)}{255MN} =: \bar{I}_\theta(i,j,k), \quad (i,j) \text{ in measuring loop of size } MN \tag{3.3}$$

$$0 \leq R_k^{\mathrm{TH}} \leq 1 \tag{3.4}$$

This can be summarized in Algorithm 3.1. The number of operations required for this algorithm is shown in Table 3.1.

Algorithm 3.1: Determing R_k^{TH} [Subsection 3.1.1]

Input: X_k and B_k
Output: R_k^{TH}
for *i=1..m* **do**
  for *j=1..n* **do**
    $I(i,j,c,k) = |X(i,j,c,k) - B(i,j,c,k)|$
    if $\sum_{c=1}^{3} I(i,j,c,k) \leq \theta$ **then**
      $I_\theta(i,j,k) = 0$
    else
      $I_\theta(i,j,k) = 1$
    end
  end
end
$R_k^{\mathrm{TH}} = \frac{\sum_{i,j} I_\theta(i,j,k)}{MN}$ (i,j) in measuring loop of size MN

Table 3.1: Number of operations for Algorithm 3.1 (p. 30)

$B_k = B_{k-1} + \beta(X_k - B_{k-1})$	$3(3mn)$
$I(i,j,c,k) = \lvert X(i,j,c,k) - B(i,j,c,k)\rvert$	$3mn$
$I_\theta(i,j,k) = \begin{cases} 0 & \text{if } \sum_{c=1}^{3} I(i,j,c,k) \leq \theta \\ 255 & \text{if } \sum_{c=1}^{3} I(i,j,c,k) > \theta \end{cases}$	$2(3mn)$
$R_k^{TH} = \frac{\sum_{i,j} I_\theta(i,j,k)}{255MN}$ (i,j) in measuring loop of size MN	$(MN)+1$
Total	$6(3mn)+MN+1$

Algorithm 3.1 can be accelerated by determing the absolute value only on the measuring loop instead of the whole image. Besides that, B_k only needs to be calculated for the measuring loop,

which requires $3(3MN)$ operations. Then the total number of operations for computing I_θ is less. Table 3.2 shows the adapted number of operations.

Table 3.2: Adapted number of operations for Algorithm 3.1

$B_k(i,j) = B_{k-1}(i,j) + \beta(X_k(i,j) - B_{k-1}(i,j))$	(i,j) in measuring loop	$3(3MN)$
$\breve{I}(i,j,c,k) = \lvert X(i,j,c,k) - B(i,j,c,k)\rvert$	(i,j) in measuring loop	$3MN$
$\breve{I}_\theta(i,j,k) = \begin{cases} 0 & \text{if } \sum_{c=1}^{3} \breve{I}(i,j,c,k) \le \theta \\ 1 & \text{if } \sum_{c=1}^{3} \breve{I}(i,j,c,k) > \theta \end{cases}$		$2(3MN)$
$\breve{R}_k^{\mathrm{TH}} = \frac{\sum_{i,j} I_\theta(i,j,k)}{MN}$	(i,j) in measuring loop	$(MN)+1$
	Total	$6(3MN)+MN+1$

Algorithm 3.1 has been applied on two sample video's, where different values of θ are tested. The results for sample video 1 are in Figure A.2, (Appendix A), and for sample video 2 in Figure A.3 (Appendix A). The results in Figure A.3 contain two lines. This is because measurements has been done over two measuring loops, see Figure 3.3a. By using two measuring loops it is possible to measure the speed of moving objects. The number of frames between these two measuring loops is used to calculate the time which takes the objects to go from the first untill the second loop, since f_{video} is known. With this time and a known distance between the measuring loops, it is possible to calculate the speed of the objects.

(a) Loops for traffic detection (b) Loops for train detection

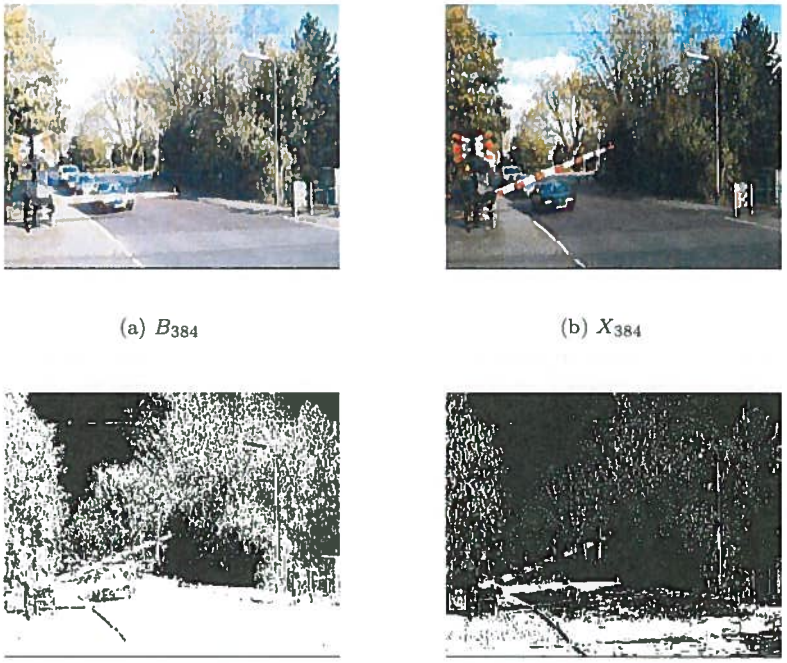
Figure 3.3: Measuring loops

The detection of objects may also be applied to detect trains. One of the differences is the position of the measuring loops. These loops are placed on a position where no traffic is detected. The best result is obtained when there is a continuous stream of motion (window - no window - window etc.). An example for the position of the measuring loops is shown in Figure 3.3b.

A big disadvantage of Algorithm 3.1 occurs when the luminious intensity changes, which happens a couple of times a day for instance when the sun breaks through the clouds. Since B_k adapts slowly to X_k, I rises above θ resulting in false positive detections. Figure 3.4b shows X_{384} and its background, B_{384}, (see Figure 3.4a) from sample video 3. As can be seen in Figure 3.4c no conclusion about motion can be drawn from this picture. There are several ways to solve this.

Adapting θ One way to compensate for luminious intensity alterations is by rising θ like in Figure 3.4d. This picture is more clear then the one in Figure 3.4c. A disadvantage of rising the threshold is that it leads to fewer detection points. See for instance Figure 3.2. So rising θ is not advisable.

Adapting β Another way is by adapting β. A larger value of β results that B_k adapts faster to X_k. So a rising difference as a result of the changed luminious intensity is corrected faster. A disadvantage of rising β is that a faster background impairs detection because the difference between the background and the image is adjusted faster.



(a) B_{384} (b) X_{384}

(c) $|X_{384} - B_{384}| > 100$ (d) $|X_{384} - B_{384}| > 250$

Figure 3.4: Rising luminious intensity at sample video 3

Adapting the speed with which the background adapts to the image is used in the current application [9]. If the luminious intensity rises the background speed is adjusted for a short time. The advantage is that the background does not approach the image much, so the difference remains enough to detect motion. This solution has a disadvantage that the luminious intensity has to be stored in a separate measuring loop. Besides that it is difficult to find a relation with which β or θ is adapted in comparison with the changing luminous intensity.

3.1.2 Sum

Another way to compensate luminious intensity changes and detect motion is by summing X_k over c and dividing it by the maximum value of this sum ($3 \times 255 \times M \times N$). The result is a mean over the intensity,

$$R_k^{\text{SUM}} := \frac{\sum_{i,j} \sum_{c=1}^{3} |X_k(i,j,c,k) - B_k(i,j,c,k)|}{3 \cdot 255 \cdot MN}, (i,j) \text{ in measuringloop}, \tag{3.5}$$

$$0 \leq R_k^{\text{SUM}} \leq 1. \tag{3.6}$$

Since R_k^{SUM} does not depend on θ this is an advantage in accordance to R_k^{TH}. This technique is described in Algorithm 3.2.

This algorithm can be accelerated by applying the absolute difference and the calculation for the background only on the measuring loop, just like has been done in Subsection 3.1.1. The number of operations required for this algorithm is shown in Table 3.3.

Algorithm 3.2: Determing R_k^{SUM} [Subsection 3.1.2]

Input: X_k and B_k
Output: R_k^{SUM}
for *i=1..m* **do**
  for *j=1..n* **do**
    $I(i,j,c,k) = |X(i,j,c,k) - B(i,j,c,k)|$
  end
end
$R_k^{\text{SUM}} = \frac{\sum_{i,j}\sum_{c=1}^{3} I(i,j,c,k)}{3 \cdot 255 \cdot M \cdot N}$ (i,j) in measuring loop

Table 3.3: Number of operations for Algorithm 3.2 (p. 33)

$B_k = B_{k-1} + \beta(X_k - B_{k-1})$	(i,j) in measuring loop	$3(3MN)$
$I(i,j,c,k) = \lvert X(i,j,c,k) - B(i,j,c,k)\rvert$	(i,j) in measuring loop	$3MN$
$R_k^{\text{SUM}} = \frac{\sum_{i,j}\sum_{c=1}^{3} I(i,j,c,k)}{3 \cdot 255 \cdot MN}$	(i,j) in measuring loop	$3(MN)+3$
	Total	$5(3MN)+3$

3.1.3 Averaging

Actually, what is done in Section 3.1.2 is taking the average of the image. The following train of thought may clarify this. If the luminous intensity of the picture changes, this happens on the entire measuring loop. When a vehicle drives by only a part of the measuring loop rises. A simple version of this is given in Figure 3.5.

(a) sunlight

(b) object

Figure 3.5: Increase of the luminous intensity

Now do not compare B_k with X_k but first divide both by their mean,

$$\begin{aligned} \bar{X}_k &:= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{c=1}^{3} X(i,j,c,k)}{3mn} \\ \bar{B}_k &:= \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{c=1}^{3} B(i,j,c,k)}{3mn} \\ R_k^{\text{AVG}} &:= \sum_{i,j}\sum_{c=1}^{3}\left|\frac{X_k(i,j,c)}{\bar{X}_k} - \frac{B_k(i,j,c)}{\bar{B}_k}\right| \quad (i,j) \text{ in measuring loop.} \end{aligned} \tag{3.7}$$

If the luminous intensity changes, this typically happens over the entire image. Then the average, $\bar{X}_k$, also changes such that the fraction of the image and its average, $\frac{X_k}{\bar{X}_k}$, compensates for this increase. If an object is passing the railroad crossing the luminous intensity does not change over the entire image but only changes on a small part. This means that the average, $\bar{X}_k$, does not change that much as X_k itself and so the fraction over the image and the average, $\frac{X_k}{\bar{X}_k}$, is changed but not that much as when the luminious intensity changed. Therefore it is possible to distinguish between a rise of the luminous intensity and a passing object. The described technique for determing R_k^{AVG} is shown in Algorithm 3.3. Table 3.4 shows the number of operation required for Algorithm 3.3.

Algorithm 3.3: Determing R_k^{AVG} [Subsection 3.1.3]

Input: X_k and B_k
Output: R_k^{AVG}
$\bar{X}_k \leftarrow$ take average of X_k
$\bar{B}_k \leftarrow$ take average of B_k
foreach i, j *in measuring loop* **do**
  $I(i,j,c,k) = |\frac{X(i,j,c,k)}{\bar{X}_k} - \frac{B(i,j,c,k)}{\bar{B}_k}|$
end
$R_k^{\text{AVG}} = \frac{\sum_{i,j}\sum_{c=1}^{3} I(i,j,k)}{3 \cdot 255 \cdot M \cdot N}$ (i,j) in measuring loop

The bottleneck of Algorithm 3.3 lies in the computation of the mean for the image and the background. It is clear that this can be accelerated by determing the mean over the measuring loops instead of over the whole image. Table 3.5 shows the adapted number of operations. This table also has taken into account that the calculation of the background only needs to be done on the measuring loops, as described in Subsection 3.1.1.

Table 3.4: Number of operations for Algorithm 3.3 (p. 34)

$B_k = B_{k-1} + \beta(X_k - B_{k-1})$	$3(3mn)$
$\bar{X}_k = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{c=1}^{3} X(i,j,c,k)}{3 \cdot m \cdot n}$	$2(3mn) + mn + 2$
$\bar{B}_k = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{c=1}^{3} B(i,j,c,k)}{3 \cdot m \cdot n}$	$2(3mn) + mn + 2$
$I(i,j,c,k) = \left\lvert \frac{X(i,j,c,k)}{\bar{X}_k} - \frac{B(i,j,c,k)}{\bar{B}_k} \right\rvert$	$3(3mn)$
$R_k^{\text{AVG}} = \sum_{i,j}\sum_{c=1}^{3} I(i,j,c,k)$ (i,j) in measuring loop	$(3MN) - 1$
Total	$10(3mn) + 2mn + (3MN) + 3$

Table 3.5: Number of operations for averaging over measuring space

$B_k = B_{k-1} + \beta(X_k - B_{k-1})$	(i,j) in measuring loop	$3(3MN)$
$\breve{X}_k = \frac{\sum_{i,j}\sum_{c=1}^{3} X(i,j,c,k)}{3MN}$	(i,j) in measuring loop	$2(3MN) + MN + 2$
$\breve{B}_k = \frac{\sum_{i,j}\sum_{c=1}^{3} X(i,j,c,k)}{3MN}$	(i,j) in measuring loop	$2(3MN) + MN + 2$
$\breve{I}(i,j,c,k) = \left\lvert \frac{X(i,j,c,k)}{\bar{X}_k} - \frac{B(i,j,c,k)}{\breve{B}_k} \right\rvert$	(i,j) in measuring loop	$3(3MN)$
$\breve{R}_k^{\text{AVG}} = \sum_{i,j\in\mathbb{R}}\sum_{c=1}^{3} \breve{I}(i,j,c,k)$		$(3MN) - 1$
Total		$9(3MN) + 2MN + 3$

A disadvantage for averaging over the measuring loop occurs when in only a small part of the measuring loop the luminious intensity changes. For instance when the shadow of a tree is covering a small part of the measuring loop. Then X_k changes while the average of X_k changes less, resulting in an alteration of R_k^{AVG}. This leads to a false positive detection.

3.1.4 Correlation between image and background

Instead of looking at the absolute difference between the image and its background, $|X_k - B_k|$, it is also possible to determine the correlation of two images, X_k and B_k. In the signal processing the correlation coëfficient, γ, at time k is defined as [8].

$$\gamma_k := \frac{\sum_{i,j}\left(\sum_{c=1}^{3} B(i,j,c,k)\sum_{c=1}^{3} X(i,j,c,k)\right)}{\sqrt{\sum_{i,j}\sum_{c=1}^{3} B^2(i,j,c,k)\sum_{i,j}\sum_{c=1}^{3} X^2(i,j,c,k)}} \quad (i,j) \text{ in measuring loop.} \tag{3.8}$$

If the Cauchy-Schwarz inequality [16],

$$\left(\sum_{i=1}^{n} x_i y_i\right)^2 \leq \left(\sum_{i=1}^{n} x_i^2\right)\left(\sum_{i=1}^{n} y_i^2\right) \Rightarrow \left|\sum_{i=1}^{n} x_i y_i\right| \leq \sqrt{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}, \tag{3.9}$$

is applied to (3.8) this yields that $0 \leq \gamma_k \leq 1$ and in fact $\gamma_k = 1 \Leftrightarrow X_k = \lambda B_k$ (where λ is a positive constant). The more the images are correlated, the more γ_k is around one. So if the luminous intensity changes evenly in the measuring loop, it is more or less a multiple of the background such that γ_k is unaltered. When a vehicle is driving by, the background and the image are correlated less such that γ_k decreases towards zero. Now define,

$$R_k^{\text{CORR}} := 1 - \gamma_k. \tag{3.10}$$

Algorithm 3.4 summarizes this and Table 3.6 shows the number of operations for this algorithm.

The algorithms discussed in Section 3.1 are now tested for three different video's namely:

- Sample video 4 (see Section 3.2)
- Sample video 3 (see Section 3.3)
- Sample video 5 (see Section 3.4)

For each video the calculation times and results are compared.

Algorithm 3.4: Determing R_k^{CORR} [Subsection 3.1.4]

Input: X_k and B_k
Output: R_k^{CORR}
$\gamma_k \leftarrow$ determine the correlation coëfficient of X and B
as described in (3.8)
$R_k^{\text{CORR}} = 1 - \gamma_k$

Table 3.6: Number of operations for Algorithm 3.4 (p. 36)

$B_k = B_{k-1} + \beta(X_k - B_{k-1})$	$3(3MN)$
$\gamma_k = \frac{\sum_{i,j}\left(\sum_{c=1}^{3} B(i,j,c,k) \sum_{c=1}^{3} X(i,j,c,k)\right)}{\sqrt{\sum_{i,j}\sum_{c=1}^{3} B^2(i,j,c,k) \sum_{i,j}\sum_{c=1}^{3} X^2(i,j,c,k)}}$ (i,j) in measuring loop	$10MN$
$R_k^{\text{CORR}} = 1 - \gamma_k$	1
Total	$3(3MN) + 10MN + 1$

3.2 Comparing the calculation times and results for sample video 4

The first video which is tested is sample video 4. Figure 3.6 shows the real situation obtained by visual scoring and gives an explanation about what the peaks represent. In Figure 3.7 the results of the algorithms are shown.

As discussed in Subsection 3.1.3 the bottleneck of Algorithm 3.3 lies in calculating the mean of the image and the background. This can be skirted by computing the mean only for the measuring loops. The result of this technique is shown in Figure 3.8a. A disadvantage of this technique is that the peaks all have similar magnitude. This is explained by that when a cyclist is driving by the influence for $\bar{X}_k$ is less than the influence when a car is driving by. Conversely the influence of a cyclist for $\hat{X}_k$ is in accordance to the influence of a car similar. This results in similar peak heights for $\hat{X}_k$ and so the value of R_k^{AVG} can not be used to distinquish the several types of traffic. A solution for this is raising the signal to the square. Then the difference in the peak heights becomes larger.

Since R_k^{AVG} (neither R_k^{AVG} squared) is between zero and one, it needs to be scaled. There is no guarantee that every video has the same scaling factor.

Another disadvantage are the small peaks preceding to the peak of the train. This may be explained by a small rise of luminous intensity in the measuring loops. This can be compensated by splitting the signal into three color threads. So instead of averaging over the mean of the three colors, divide every channel by its own mean,

$$\begin{aligned}
\bar{X}_{k,c} &= \frac{\sum_{i,j} X(i,j,c,k)}{MN} \qquad c = 1,2,3 \quad (i,j) \text{ in measuring loop} \\
\bar{B}_{k,c} &= \frac{\sum_{i,j} B(i,j,c,k)}{MN} \qquad c = 1,2,3 \quad (i,j) \text{ in measuring loop} \\
R_k^{\text{AVG}} &:= \sum_{i,j} \left| \sum_{c=1}^{3} \frac{X(i,j,c,k)}{\bar{X}_{k,c}} - \frac{B(i,j,c,k)}{\bar{B}_{k,c}} \right| \quad (i,j) \text{ in measuring loop.}
\end{aligned} \tag{3.11}$$

The result of (3.11) is shown in Figure 3.8b.

As expected the calculation time of this technique is somewhat longer since some extra operations have to be done. But an advantage of this technique is that the peaks preliminary to the peak of the train are smaller. This can be explained by the following train of thought. If only one channel changes in luminious intensity while the other two remain the same, this alteration is

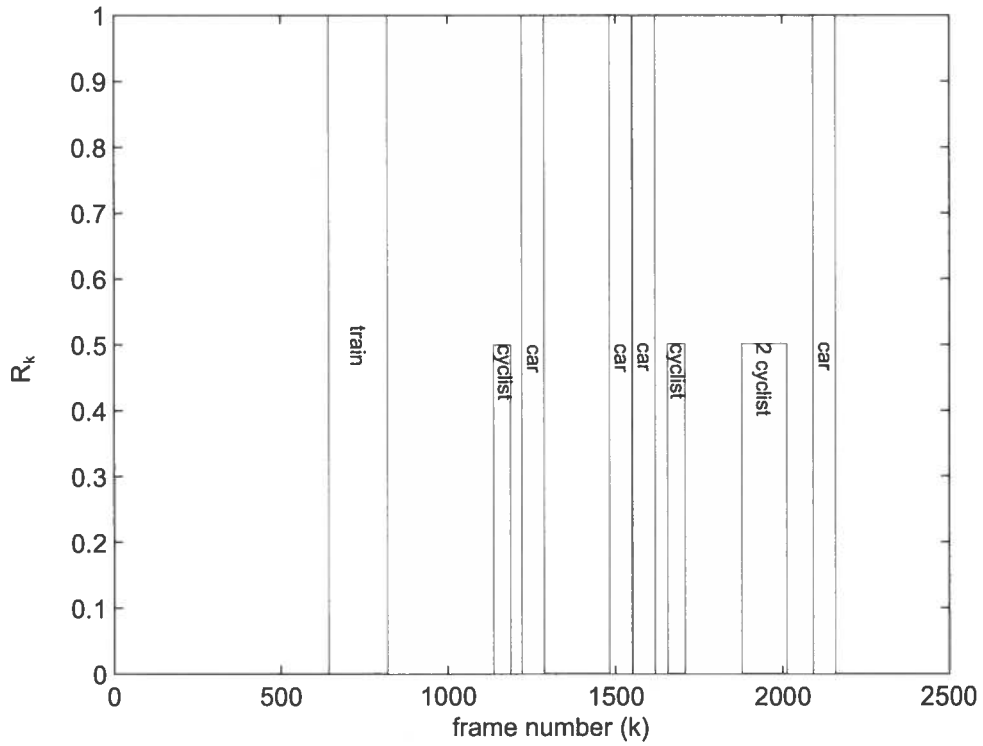
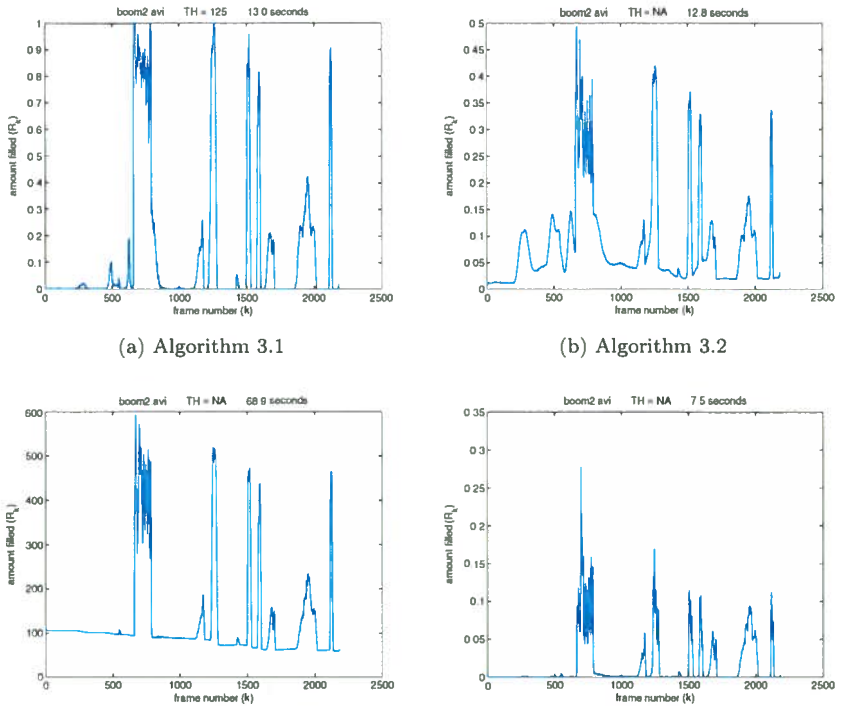


Figure 3.6: Result for sample video 4 obtained by visual scoring



(a) Algorithm 3.1 (b) Algorithm 3.2

(c) Algorithm 3.3 (d) Algorithm 3.4

Figure 3.7: Results (R_k) of the algorithms applied to sample video 4

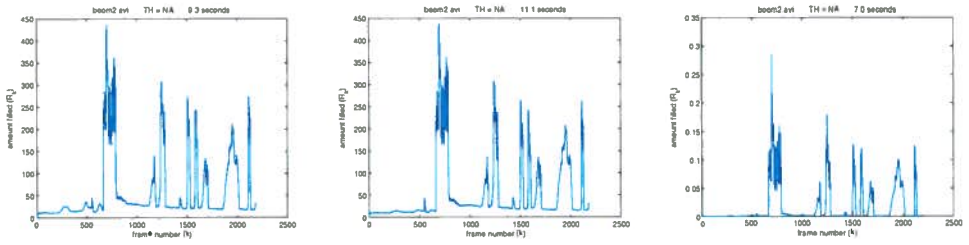
corrected for only one third,

$$\begin{aligned}
&\left|\frac{B_{k,1}+B_{k,2}+B_{k,3}}{[B_{k,1},B_{k,2},B_{k,3}]}-\frac{X_{k,1}+X_{k,2}+X_{k,3}}{[X_{k,1},X_{k,2},X_{k,3}]}\right| = \\
&\left|\frac{B_{k,1}}{[B_{k,1},B_{k,2},B_{k,3}]}-\frac{X_{k,1}}{[X_{k,1},X_{k,2},X_{k,3}]}+\dots\right. \\
&\quad\frac{B_{k,2}}{[B_{k,1},B_{k,2},B_{k,3}]}-\frac{X_{k,2}}{[X_{k,1},X_{k,2},X_{k,3}]}+\dots \\
&\quad\left.\frac{B_{k,3}}{[B_{k,1},B_{k,2},B_{k,3}]}-\frac{X_{k,3}}{[X_{k,1},X_{k,2},X_{k,3}]}\right| = \\
&\left|\frac{B_{k,1}}{\frac{1}{3}\left(\bar{B}_{k,1}+\bar{B}_{k,2}+\bar{B}_{k,3}\right)}-\frac{X_{k,1}}{\frac{1}{3}\left(\bar{X}_{k,1}+\bar{X}_{k,2}+\bar{X}_{k,3}\right)}+\dots\right. \\
&\quad\frac{B_{k,2}}{\frac{1}{3}\left(\bar{B}_{k,1}+\bar{B}_{k,2}+\bar{B}_{k,3}\right)}-\frac{X_{k,2}}{\frac{1}{3}\left(\bar{X}_{k,1}+\bar{X}_{k,2}+\bar{X}_{k,3}\right)}+\dots \\
&\quad\left.\frac{B_{k,3}}{\frac{1}{3}\left(\bar{B}_{k,1}+\bar{B}_{k,2}+\bar{B}_{k,3}\right)}-\frac{X_{k,3}}{\frac{1}{3}\left(\bar{X}_{k,1}+\bar{X}_{k,2}+\bar{X}_{k,3}\right)}\right|.
\end{aligned} \tag{3.12}$$

Whereas the raise is corrected entirely by averaging over each color channel,

$$\left|\frac{B_{k,1}}{\bar{B}_{k,1}}-\frac{X_{k,1}}{\bar{X}_{k,1}}+\frac{B_{k,2}}{\bar{B}_{k,2}}-\frac{X_{k,2}}{\bar{X}_{k,2}}+\frac{B_{k,3}}{\bar{B}_{k,3}}-\frac{X_{k,3}}{\bar{X}_{k,3}}\right|. \tag{3.13}$$

The technique of averaging over each color channel can also be applied on the correlation algorithm. Instead of first summing over all colors for the image and the background calculate the correlation coëfficient for every color channel. The result for R_k^{CORR} split into colors is shown in Figure 3.8c.



(a) Average over loops (b) Average split into colors (c) Correlation split into colors

Figure 3.8: Results (R_k) for the extended versions of the algorithms applied to sample video 4

3.3 Comparing the calculation times and results for sample video 3

The algorithms discussed in Section 3.1 are now applied to sample video 3. The result are shown in Figure 3.10. For a reference there is a general graph, based on visual scoring, shown in Figure 3.9 with some explanation for the several peaks.

After the passing of the train, the intensity rises. As can be seen Algorithm 3.1 does not give any workable result since the value of R_k^{TH} remains high. Even Algorithm 3.2 is not able to solve this problem correctly. Algorithm 3.3 and 3.4 on the other hand have a much better result and are independent of the luminous intensity. Results of the extended versions of Algorithm 3.3, averaging over measuring loops and averaging split into color channels, and the extended version of Algorithm 3.4, correlation split into colors, are shown in Figure 3.11. The results shown in Figure 3.10 and 3.11 confirm the results of Section 3.2.

3.4 Comparing the calculation times and results for sample video 5

This section treats another video namely sample video 5 the choice for this video is because several vehicles are driving by at the railroad crossing. Figure 3.12 gives a general result and explains what the peaks represent. Figure 3.13 shows the results of the algorithms.

For this video also the extended algorithms are applied. The results of the extended algorithms are shown in Figure 3.14. These results confirm what is shown in Sections 3.2 and 3.3.

As can be seen, the correlation split into colors is the fastest algorithm. A remarkable phenomenon is that the algorithm where the correlation is split into colors is faster than the algorithm where it is not split into colors. This may be a result of the efficiency of MATLAB.

Train detection The algorithms discussed in Section 3.1 can also be applied for detection of a train. The results for the three videos discussed in Sections 3.2 up to 3.4 are shown in Appendix A.3. These results conclude that R_k^{TH}, R_k^{SUM}, R_k^{AVG} and R_k^{CORR} fluctuate when a train is passing. An explanation is that the train is not even colored at the position of the loops, see Figure 3.3b. At this position a continious motion takes place (window - no window - window - no window - etc) which is needed to detect the train. This causes that the value of R_k fluctuates. A solution for this problem can be found by averaging the signal over time.

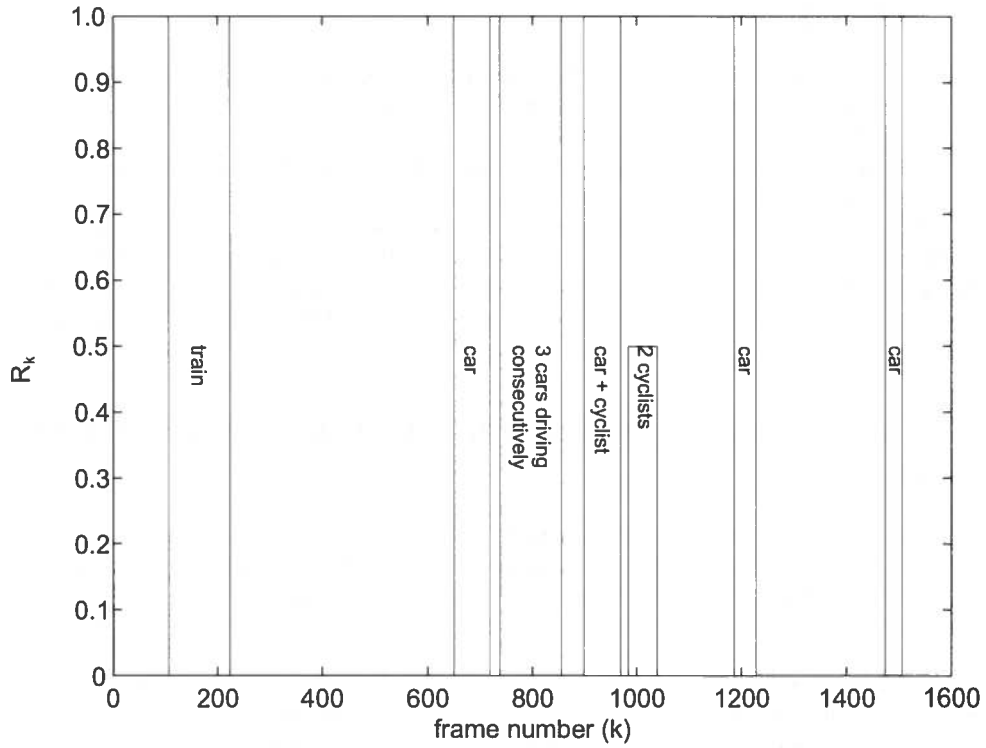
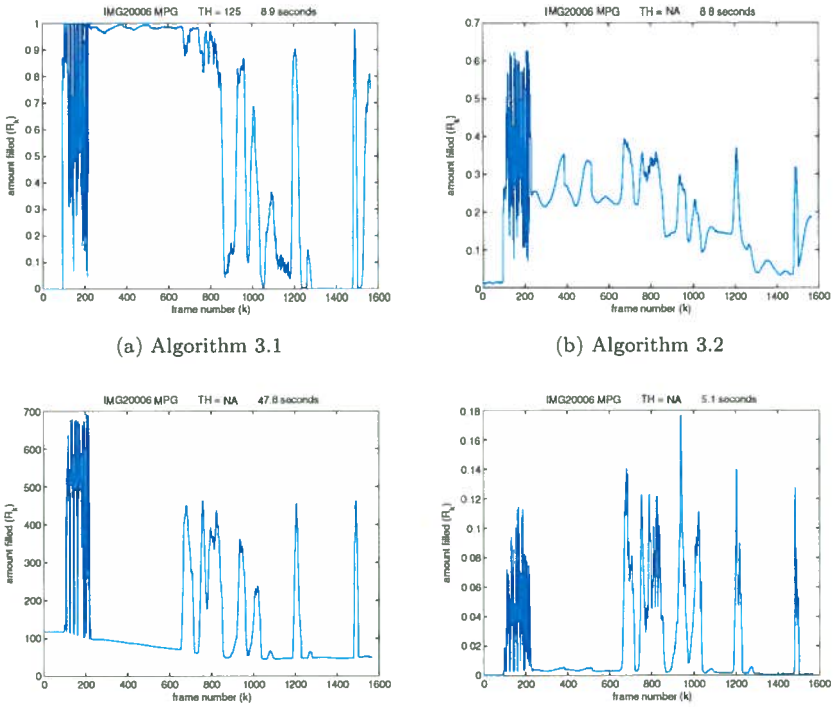


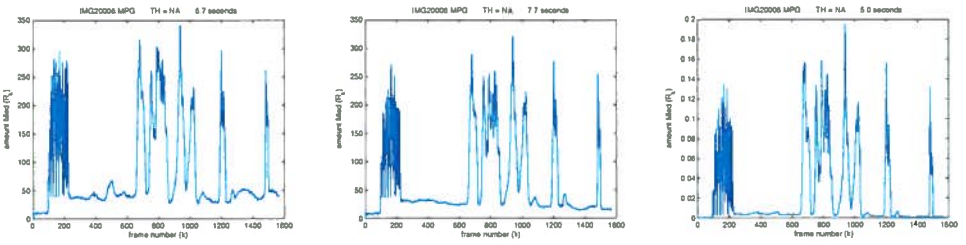
Figure 3.9: Result for sample video 3 obtained by visual scoring



(a) Algorithm 3.1 (b) Algorithm 3.2

(c) Algorithm 3.3 (d) Algorithm 3.4

Figure 3.10: Results (R_k) of the algorithms applied to sample video 3



(a) Average over loops (b) Average split into colors (c) Correlation split into colors

Figure 3.11: Results (R_k) for the extended algorithms applied to sample video 3

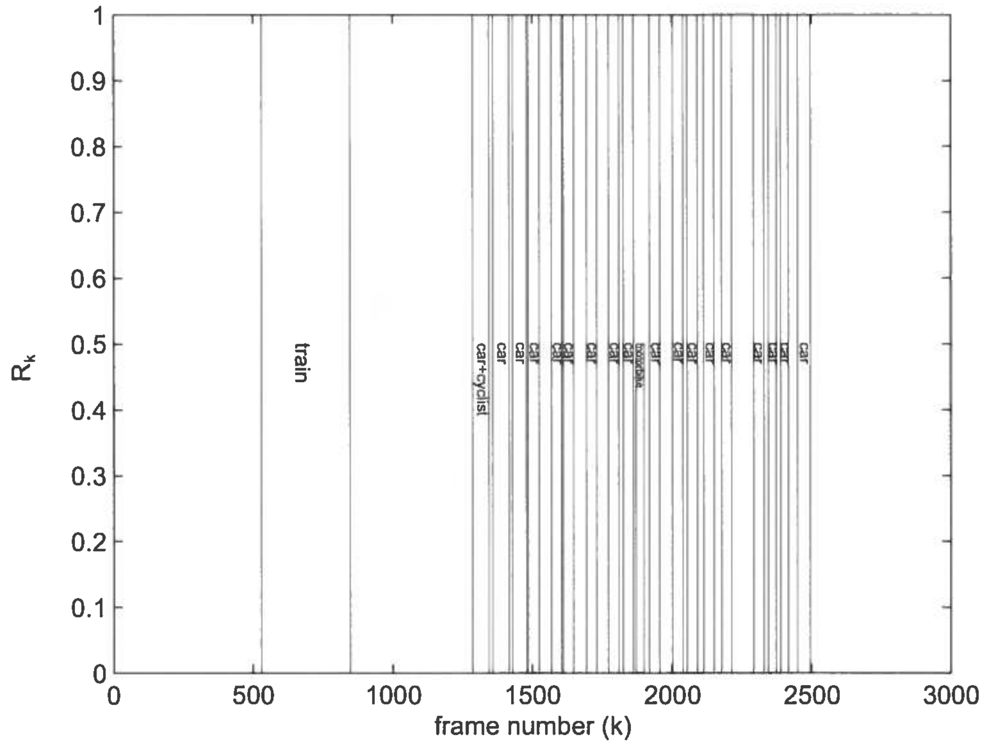
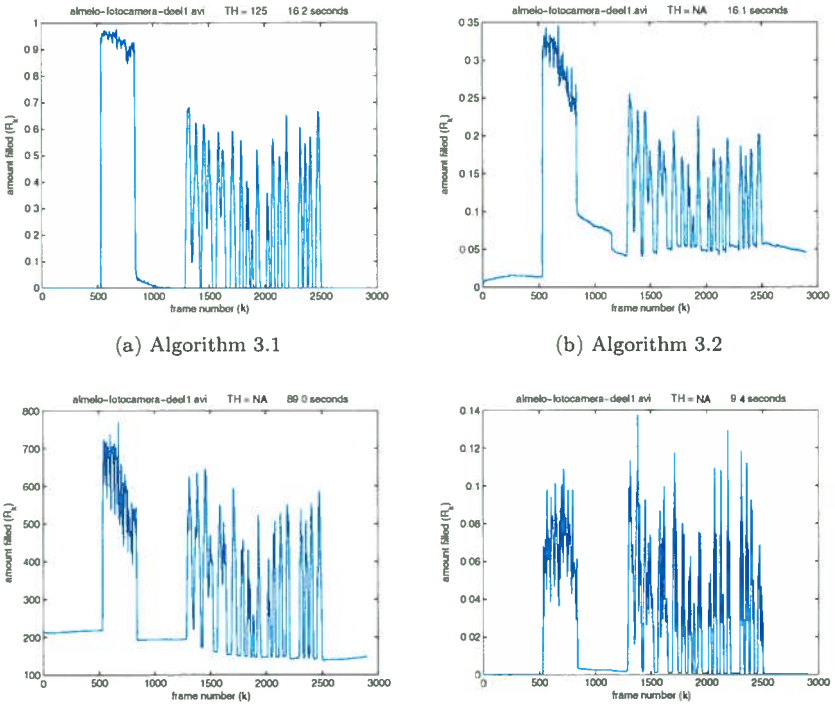


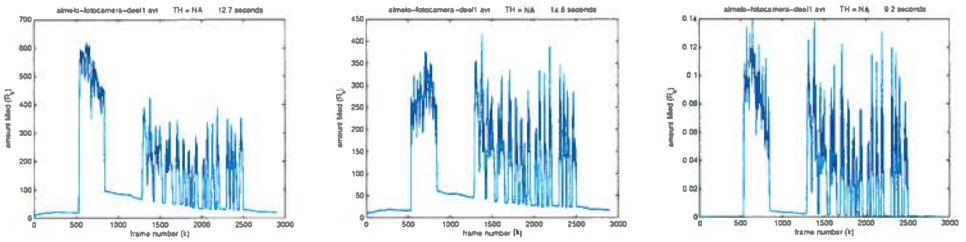
Figure 3.12: Result for sample video 5 obtained by visual scoring



(a) Algorithm 3.1 (b) Algorithm 3.2

(c) Algorithm 3.3 (d) Algorithm 3.4

Figure 3.13: Results (R_k) of the algorithms applied to sample video 5



(a) Average over loops (b) Average split into colors (c) Correlation split into colors

Figure 3.14: Results (R_k) for the extended algorithms applied to sample video 5

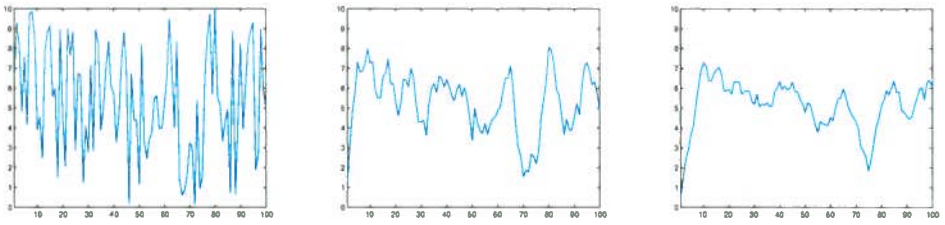
3.5 Averaging over time

Averaging over time can be seen as taking the average over the last F measurements, setting F as the filter order.

$$\bar{R}_k = \frac{1}{F}\left(R_k + R_{k-1} + \ldots + R_{k-F+1}\right) \tag{3.14}$$

Large peaks are filtered and so the signal is more flat.

Example 3.5.1. As an example a vector has been generated with 100 uniformly distributed pseudorandom numbers between zero and ten, Figure 3.15a shows this data. (The original data can be seen as filtering with order $F = 1$). Figure 3.15b and 3.15c show the filtered data for respectively $F = 5$ and $F = 10$.



(a) $F = 1$ (b) $F = 5$ (c) $F = 10$

Figure 3.15: 100 random numbers and different filter orders

The filterorder may not be too large since then the peaks are flatten out. On the other hand a too small filterorder is also useless because then the signal fluctuates too much. To determine a good filterorder a couple of filterorders are tested and the results are shown in Appendix A.4. From this results it may be concluded, by visual scoring, that for the train a good filterorder might be around forty or fifty and for the traffic around ten or fifteen. From the results there is chosen to set $F_{\text{train}} = 48$ and $F_{\text{traffic}} = 12$. The filterorder can be linked to f_{video}. Then F divided by f_{video} is the time about which is filtered. So with $f_{\text{video}} = 24$ Hz the signal for the train detection is filtered over two seconds while the singal for traffic is filtered over 0.5 seconds.

In the example there are 100 data points this is doable to store for filtering. But in reality the number of data points depends on the number of frames and the number of measuring loops. When streaming a video it is not known how many frames this will be, so saving every data point may cost lots of memory which slows down the application. A workaround for this is done by only saving the last F data points. Every time when a new data point is measured it is set at the end of a vector and the rest is shifted to the left,

$$R^k = \begin{bmatrix} R_{k-F+1} & R_{k-F-2} & \ldots & R_{k-1} & R_k \end{bmatrix}. \tag{3.15}$$

At the beginning a zero-vector of size $1 \times F$ is created to store the measurements of R. The vector R is filtered according to (3.14) such that it will become $\bar{R}$. More information about averaging over time can be found in [7].

A disadvantage of this technique is that the vector R needs to be stored in memory, the larger F is, the more memory it requires. The technique of the exponential forgetting theorem, described in Subsection 2.2.3, can also be used to filter the values of R_k

$$\breve{R}_k = \breve{R}_{k-1} + \hat{\beta}\left(R_k - \breve{R}_{k-1}\right). \tag{3.16}$$

With

$$0 \leq \hat{\beta} \leq 1,$$

which can be different from the value of β. The larger $\hat{\beta}$ is, the faster the adaptation to R_k. Again Theorem 2.3.2 is used to determine a value for $\hat{\beta}$.

Example 3.5.2. If the filter settling time for traffic is set to one second then $\hat{\beta}_{\text{traffic}} = \frac{\ln 100}{24} \approx 0.1919$ and for the train to 0.5 seconds corresponding with $\hat{\beta}_{\text{train}} = \frac{\ln 100}{24 \cdot 0.5} \approx 0.3838$.

An advantage of this method is the number of required operations, since R_k is a (real) number (3.16) requires only three operations while (3.15) requires $F+2$ operations. So the second method is the fastest method tested and is from now on used to filter the values of R_k.

3.6 Averaging over position

Another way to filter background noise is by averaging over position. This means that the image is shifted and summed over the new values and then averaged.

$$\tilde{X}(i,j,c,k) = \frac{X(i,j,c,k) + X(i+1,j,c,k) + X(i,j+1,c,k) + X(i+1,j+1,c,k)}{4} \tag{3.17}$$

Example 3.6.1.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \rightarrow \left(\tfrac{1}{4}\right) \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} + \left(\tfrac{1}{4}\right) \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$
$$+ \left(\tfrac{1}{4}\right) \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 5 & 6 & 7 \\ 0 & 9 & 10 & 11 \\ 0 & 13 & 14 & 15 \end{bmatrix} + \left(\tfrac{1}{4}\right) \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 5 & 6 & 7 \\ 0 & 9 & 10 & 11 \end{bmatrix} = \begin{bmatrix} 0.25 & 0.75 & 1.25 & 1.75 \\ 1.50 & 3.50 & 4.50 & 5.50 \\ 3.50 & 7.50 & 8.50 & 9.50 \\ 5.50 & 11.50 & 12.50 & 13.50 \end{bmatrix}$$

This technique can be extended by shifting over all directions.

$$\begin{aligned} \tilde{\tilde{X}}(i,j,c,k) \;=\; & \frac{1}{9}\big(X(i,j,c,k) + X(i+1,j,c,k) + X(i+1,j+1,c,k) + \ldots \\ & X(i,j+1,c,k) + X(i-1,j+1,c,k) + X(i-1,j,c,k) + \ldots \\ & X(i-1,j-1,c,k) + X(i,j-1,c,k) + X(i+1,j-1,c,k)\big) \end{aligned}$$

If the image is shifted over position the algorithms are redefined, the sum over all colors is redefined by:

$$\sum_{c=1}^{3} \tilde{X}(i,j,c,k)$$

And (3.2) becomes:

$$\begin{aligned} \tilde{I}(i,j,c,k) &:= |\tilde{X}(i,j,c,k) - \tilde{B}(i,j,c,k)| \\ \tilde{I}_\theta(i,j,k) &:= \begin{cases} 0 & \text{if } \sum_{c=1}^{3} \tilde{I}(i,j,c,k) \le \theta \\ 255 & \text{if } \sum_{c=1}^{3} \tilde{I}(i,j,c,k) > \theta \end{cases} \end{aligned} \tag{3.18}$$

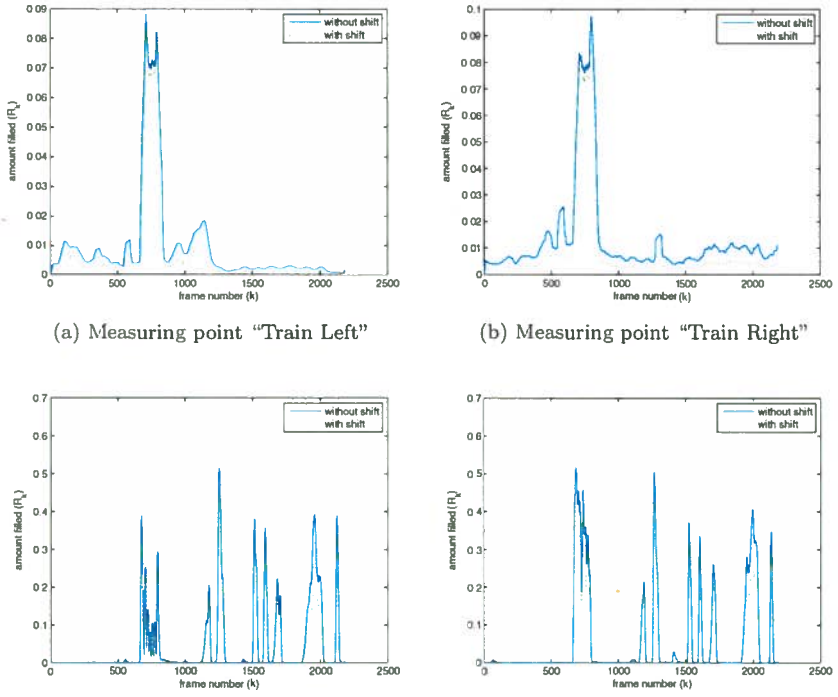
In this way R_k^{TH} can be redefined by:

$$\tilde{R}_k^{TH} := \frac{\sum_{i,j} \tilde{I}(i,j,k)}{MN} \quad (i,j) \text{ in measuring loop} \quad =: \tilde{\tilde{I}}_\theta(i,j,k) \tag{3.19}$$

By filtering this over time:

$$\breve{\tilde{R}}_k^{\text{TH}} = \breve{\tilde{R}}_{k-1}^{\text{TH}} + \hat{\beta}\left(\tilde{R}_k^{\text{TH}} - \breve{\tilde{R}}_{k-1}^{\text{TH}}\right) \tag{3.20}$$

To test if this algorithm is useful, for every measuring loop the amount of fill with shifting $\left(\breve{\tilde{R}}\right)$ and without shifting $\left(\breve{R}\right)$ is measured for sample video 4. These amounts have been plotted in Figure 3.16. By the look of it there is no big difference between the values of $\breve{\tilde{R}}$ and $\breve{R}$. But the number of operations for this technique is for every multiplication $3MN$ and for every summation another $3MN$. So the total number of operations are $7(3MN)$. But as seen from Figure 3.16 the result does not have that much effect that the noise is left out totally so this technique is not applied for detecting objects. Anyways this method might still be useful to compensate for moving trees, when the barrier needs to be detected.



(a) Measuring point "Train Left"

(b) Measuring point "Train Right"

(c) Measuring point "Traffic loop 1"

(d) Measuring point "Traffic loop 2"

Figure 3.16: Difference between shifting $\left(\tilde{\bar{R}}\right)$ and no shifting $\left(\bar{R}\right)$

3.7 Colorspace

3.7.1 Red, Green and Blue

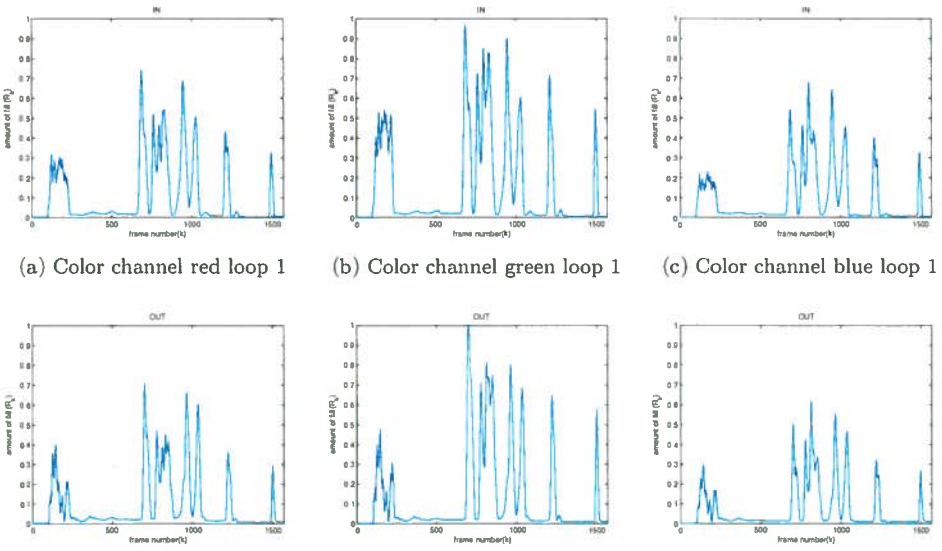
As was concluded at the end of Section 3.4, Algorithm 3.4 split into colors is the fastest. So it is better to look at one color channel instead of looking at all three color channels. This speeds up Algorithm 3.4 even more. Figure 3.17 shows the results for respectively the red, green and blue channel. Figures a, b and c are for measuring loop 1, figures d, e and f ones are for measuring loop 2. These result show that for sample video 3, all three colorchannels provide a continious result. Unfortunately there is no difference in peak heights.

3.7.2 Hue, Saturation and Value

Another colorspace which is used frequently is Hue, Saturation and Value [6]. Figure 3.18 shows the results for each channel of this colorspace. One disadvantage can be seen from the results that the values of R_k^{CORR} are sometimes not defined. Another disadvantage is the fact that the images must be converted to this color space before it can be used for calculation.

3.7.3 YCbCr

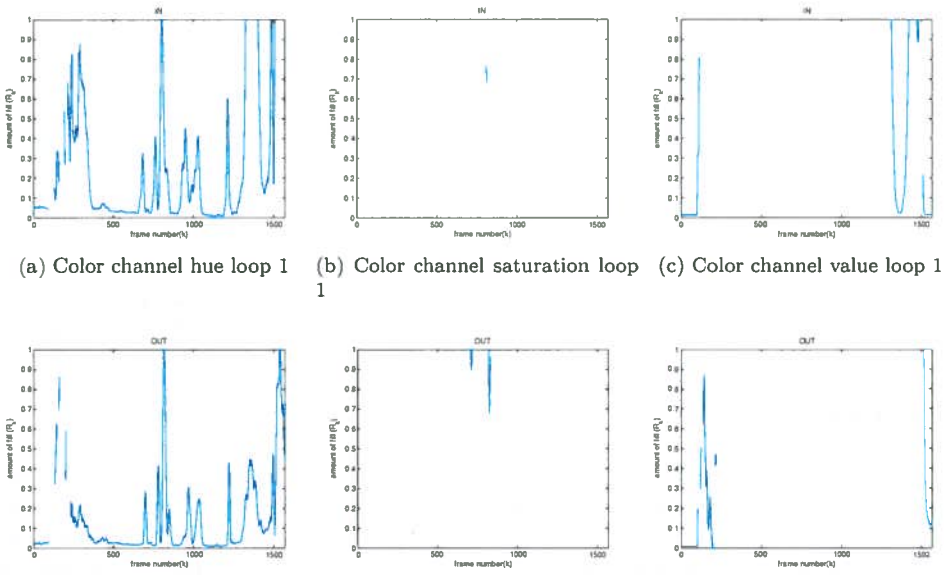
A third colorspace is YCbCr [5]. A big advantage of this colorspace is the fact that JPEG-images are normally stored in this format. At the final application a MPJEG (Moving **JPEG**) stream delivers the video images. Figure 3.19 shows the results for this color space. The only usable channel is the Y-channel (this is a channel for the luminious intensity of the image).



(a) Color channel red loop 1 (b) Color channel green loop 1 (c) Color channel blue loop 1

(d) Color channel red loop 2 (e) Color channel green loop 2 (f) Color channel blue loop 2

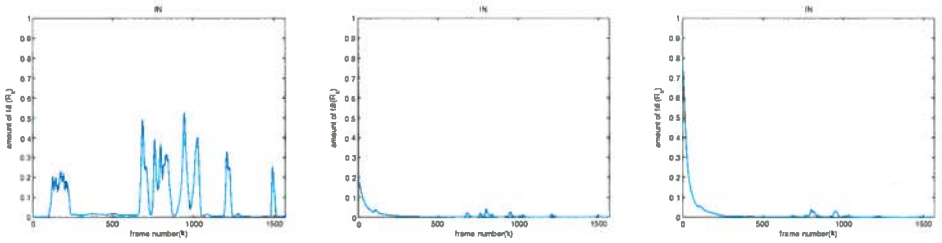
Figure 3.17: Results for R_k^{CORR} for the three channels (RGB)



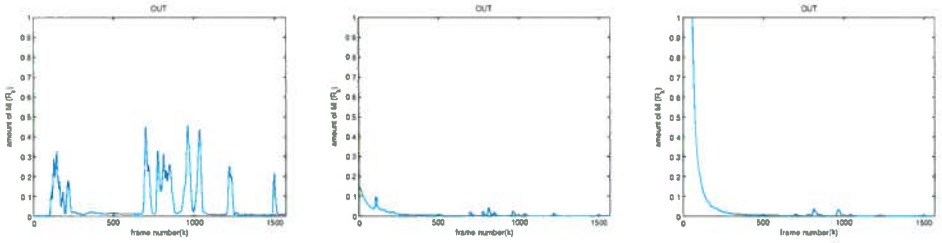
(a) Color channel hue loop 1 (b) Color channel saturation loop 1 (c) Color channel value loop 1

(d) Color channel hue loop 2 (e) Color channel saturation loop 2 (f) Color channel value loop 2

Figure 3.18: Results for R_k^{CORR} for the three channels (HSV)

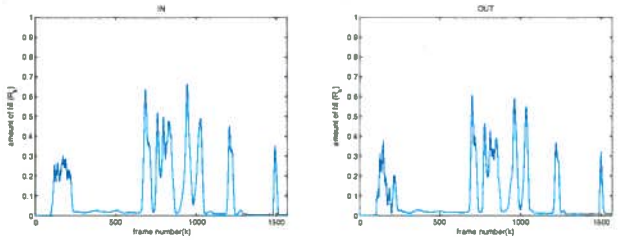


(a) Color channel Y IN (b) Color channel Cb IN (c) Color channel Cr IN



(d) Color channel Y OUT (e) Color channel Cb OUT (f) Color channel Cr OUT

Figure 3.19: Results for R_k^{CORR} for the three channels (YCbCr)



(a) Color channel Grayscale IN (b) Color channel Grayscale OUT

Figure 3.20: Results for R_k^{CORR} (Grayscale)

3.7.4 Grayscale

Instead of looking at the colored image, it might be even better to apply the algorithms on the grayscale images. A big advantage of this technique is that only one color channel remains. Besides that the influence of changes in the luminious intensity is less, since the grayscale is usually a mean of the channels Red, Green and Blue. The results for grayscale images are shown in Figure 3.20.

Conclusion

This chapter described several algorithms for detecting traffic. When the algorithms are applied to the tested video's, some advantages and disadvantages arise. By Algorithm 3.1 (see Subsection 3.1.1) it is easy to see the vehicles because the clustered white points demonstrate their position. Problem with this algorithm is how to choose θ correctly. A large θ means less detection but also less noise. Otherwise a small θ leads to more accurate measurements. Unfortunately this gets along with more noise.

Besides that, a big disadvantage of this algorithm is the dependence on the luminous intensity. When the luminous intensity changes the absolute difference of X_k and B_k rises. This may lead to false positive detection. This can be get around by a reference area which adapts β or θ, which is used in the current application [9]. Another possiblity is by skipping θ.

Algorithm 3.2 (see Subsection 3.1.2) does not use θ so the position detection of the object is more difficult. But still this algorithm is not independent on the luminious intensity. When the luminous intensity changes, it still leads to false positive detection. This is because the absolute difference is divided by a constant factor. If the absolute difference rises, as a consequence of an alteration in the luminious intensity, the numerator of (3.6) changes while the denominator does not change, resulting that (3.6) also changes.

The problem of a changing luminious intensity is skirted by the use of Algorithm 3.3 (see Subsection 3.1.3). Since the denominators of (3.7) contains the averages of X_k and B_k there is a correction for a change in the luminious intensity. If the luminious intensity in the image changes, also the average of X_k changes and so the influence to R_k^{AVG} is less. A disadvantage of this algorithm is the number of operations required to calculate R_k^{AVG}. This is for instance more than the operations required for R_k^{TH}.

Instead of the absolute difference of B_k and X_k, Algorithm 3.4 (see Subsection 3.1.4) uses the correlation between B_k and X_k. If the luminous intensity changes over the whole measuring loop, then X_k is still more or less a linear multiple of B_k. Then the correlation factor remains around one. Instead of the case when a car is driving by, then the luminious intensity does not change over the whole measuring loop resulting that X_k and B_k are less correlated. So the correlation factor fluctuates. This confirms that Algorithm 3.4 is independent from alternating luminious intensity. Besides that this algorithm is normalized between zero and one. A disadvantage for this algorithm is when the intensity changes only partial on the measuring loop. This may lead to false positive detection.

The results in Sections 3.2 up to 3.4 and the number of operations required by the algorithms show that Algorithm 3.4, which is based on the correlation, is the best algorithm to use. This algorithm is independent from intensity changes and even requires the least operations of the tested algorithms. This confirms that β can be chosen small since it does not have influence on the luminious intensity.

This algorithm is also robust, since only detection takes place if a object is passing. When nothing passes the railroad crossing nothing is detected.

To supress the noise the image can be averaged over its position. By this technique the influence of noise is filtered. A big disadvantage of this technique is the number of operations required for this technique, as shown in Section 3.6. A better method to filter noise is to flatten R_k this is done using the exponential adaptation as described in Section 3.5.

In further research it may be useful to only use the Y-channel of YCbCr-colorspace or grayscale images. Besides that it is currently unknown how R_k^{CORR} will react on wheater changes like rain or snow. Unfortunately from the available video's it was not possible to test these situations. For these weather situations it might be interesting to research the possibilities of a infrared camera.

Now define L_{mot} as the limit for R_k^{CORR} to detect a vehicle. So if

$$R_k^{\text{CORR}} \geq L_{\text{mot}}$$

a vehicle is detected.

Chapter 4

Detection of dangerous situations

In the previous chapter several indicators, R_k, are discussed for detecting motion. The method with correlation, R_k^{CORR}, is the most likely method for detecting motion. So from now on R_k is defined as:

$$R_k := R_k^{\text{CORR}}$$

This chapter treats an expansion on motion detection to detect stationary objects abd other dangerous situations on the railroad crossing, see Figure 4.1. To test this, measuring loops are placed onto the video and the value of R_k is further investigated.

4.1 Algorithm

Chapter 3 already defined that a vehicle is present if

$$R_k \geq L_{\text{mot}}.$$

As was seen in some of the figures where R_k was shown, R_k rises when a vehicle drives into the measuring loop and it goes down if the vehicle drives out of the measuring loop. When a long vehicle (for instance a lorry) is driving by, R_k fluctuates for some time. But when a vehicle is stationary, R_k does not fluctuate in the ideal case.

So by keeping up the latest values of R_k and comparing these values, theoretically it might be possible to detect stagnation of objects on the railroad crossing. The choice for comparing the values is done by checking if the absolute difference, D_k, of R_k and R_{k-1},

$$D_k = |R_k - R_{k-1}|, \tag{4.1}$$

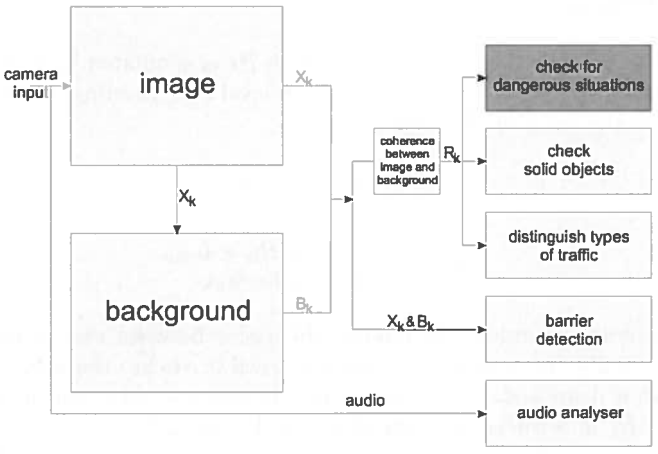


Figure 4.1: Overview of the application (stagnation)

is less then or equal to a specified value d,

$$D_k \leq d.$$

When this is the case, start a counter which keeps track for how long R_k has not changed. When the difference of R_k and R_{k-1} is outside the range (greater than d) the counter resets,

$$\begin{cases} \text{counter} = 0 & \text{if } D_k \leq d \\ \text{counter} = \text{counter} + 1 & \text{otherwise} \end{cases}.$$

If the counter is above a certain level, W, the application gives a warning status. The technique described is shown in Algorithm 4.1.

The check for stationary objects only puts into effect if R_k has crossed the border of motion, L_{mot},

$$R_k \geq L_{\text{mot}}.$$

This is because otherwise a foul seem to be detected when there is no traffic. Since the value of R_k remails also almost flat when nothing is going on.

Algorithm 4.1: Detection of stationary objects [Section 4.1

Input: R_k, R_{k-1}
Output: foul boolean
if $R_k > L_{mot}$ **then**
  $D_k \leftarrow$ Calculate the difference between R_k and R_{k-1}
  if $D_k \leq d$ **then**
    counter = counter + 1
  else
    counter = 0
  end
  if *counter* $\geq W$ **then**
    foul boolean = 1
  else
    foul boolean = 0
  end
end

4.2 Simulation

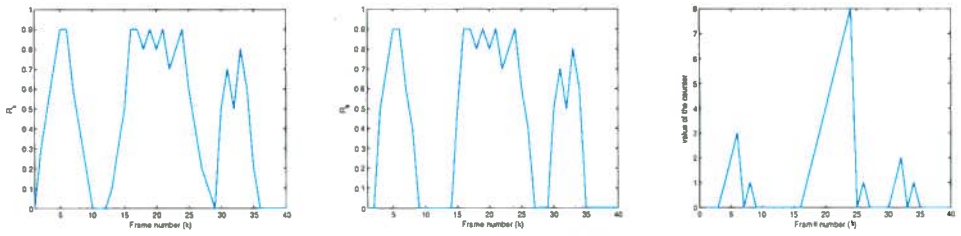
To test Algorithm 4.1 a function is written in which R_k is simulated by fantasy data. For the test of the algorithm these numbers are first filtered on level 0.4, meaning that numbers below 0.4 are set to zero. This is the same as defining

$$L_{\text{mot}} := 0.4$$

$$\grave{R}_k := \begin{cases} R_k & \text{if } R_k \geq L_{\text{mot}} \\ 0 & \text{otherwise} \end{cases}.$$

The next step is to determine the relative difference between two successive values. If D_k is below 0.25 (so $d = 0.25$) the counter is increased. Until it reaches the value four (so $W = 4$), then a warning message is displayed. When the counter is above seven a danger message is shown. The original data, R_k, $\grave{R}_k$ and the counter are shown in Figure 4.2.

The values of L_{mot} and d are chosen for this simulation. These values do not correspond for each video. More information about choosing the value for L_{mot} is found in Chapter 5.



(a) Original simulation R_k (b) Filtered simulation $\grave{R}_k$ (c) Counter simulation

Figure 4.2: Results of the simulation (Section 4.2)

4.3 Reality

From Section 4.2 it can be concluded that Algorithm 4.1 is working well for this simulation. Now it needs to be tested on a real video. Since there was no video available in which a car was stationary on the railroad crossing it was made from another video. In sample video 1 a car is driving by at frame 230. A new video is created where frame 230 is strechted for four seconds (96 frames if $f_{\text{video}} = 24$ Hz). The rest of the frames are kept in place. Now in this new video it looks like at X_{230} a car is stationary on the railroad crossing. The result of R_k is shown in Figure 4.3.

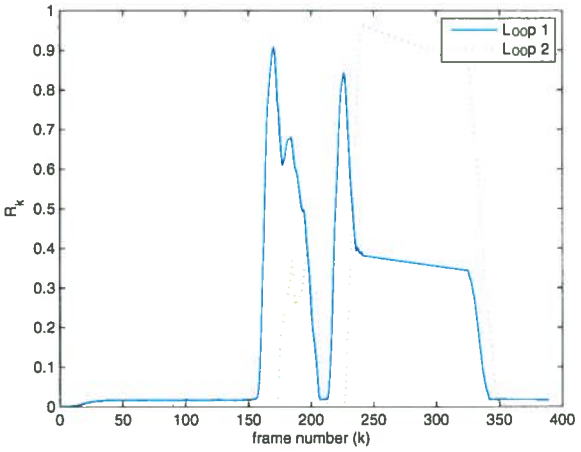


Figure 4.3: R_k for the video with stagnation

From Figure 4.3 it can be concluded that it is easy to see where a vehicle is stationary.

4.4 Position of the loop

As shown in Figure 3.3a to detect the moving objects two measuring loops have been placed on the railroad crossing. The first loop is usually placed in front of the railroad. Stationary objects at this position is not a dangerous situation. For instance this can occur when a vehicle is stationary because the barriers are closed. A more dangerous situation is created when a object is stationary at the second measuring loop. Usually this one is placed at the end of the railroad or on the railroad itself, see for instance Figure 3.3a. So it is better to apply Algorithm 4.1 only on the second measuring loop, this even takes less operations.

Another approach is to introduce an extra measuringloop, positioned over the whole railroad. For instance like is done in Figure 4.4. Name this loop the 'loop 12' since it lies between loop 1 and loop 2. R_k is denoted as $R_{12,k}$.

Definition 4.4.1. $R_{12,k}$ is defined as R_k for the 'stationarity loop'.

An advantage of this method is the possibility to detect other fouls like driving through the red light. This is the case when for instance the barriers are already closing or are still opening. Another dangerous situation is crossing the railroad while the barriers are closed. For instance by slalom driving around the barrier or crawling underneath the barriers.

Figure 4.4: Measuring loop on railroad

By the position of the extra measuring loop these fouls can be detected. If the barriers are open then this measuring loop can be used to detect stationary objects. In this scenario, $R_{12,k}$, must be fluctuating when it is above L_{mot}. In the other scenario when the ligths are are on, in this case a train is approaching and the barriers will go down, this loop needs to be empty and so

$$\begin{cases} \text{Status: 'Safe'} & \text{if } R_{\text{stat},k} \leq L_{\text{mot}} \\ \text{Status: 'Danger'} & \text{otherwise} \end{cases}.$$

Except when a train is driving by, in this case $R_{12,k}$ is larger than L_{mot}. This situation is detected by the measuring loops for the train, see also Figure 3.3b. Since $R_{12,k}$ is based on the correlation method as described in Subsection 3.1.4 there is no easy way to determine the direction of the train using loop 12. Even better is to disable the loop when a train is detected since it has no additive value.

Another advantage of this extra loop is the sampling frequency. For this loop it is not needed to perform this check at a frequency f_{video}. Figure 4.5 shows $R_{12,k}$ for various frequencies applied to sample video 14.

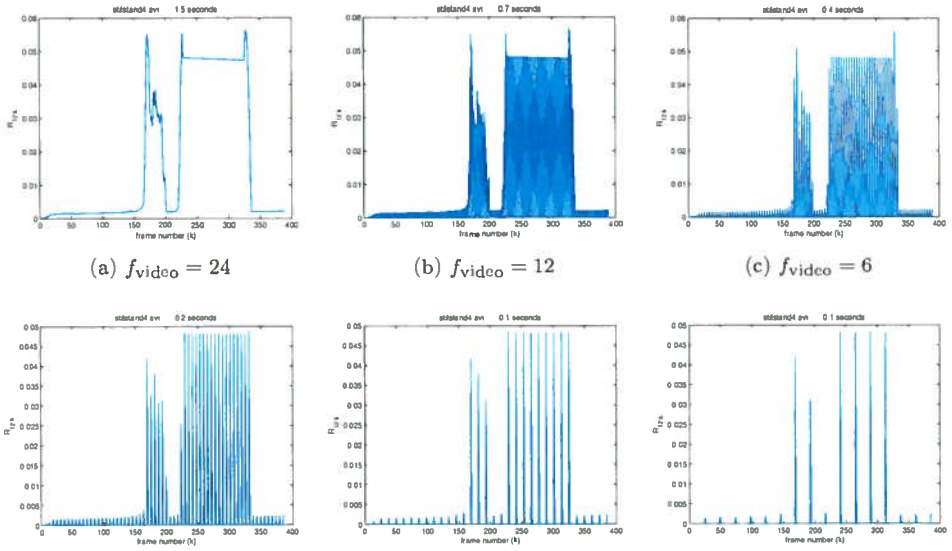
As can be seen the fastest result is with $f_{\text{video}} = 1$. This is clear since this takes the least number of frames. For the results in Figure 4.5,

$$\beta := \frac{1}{10000},$$

corresponding to $t_{\text{set}} \approx 1919$ seconds. Filtering the signal is not useful in this case because then $R_{12,k}$ is affected by the previous values resulting in a less straight line. So

$$\hat{\beta}_{12} := 1,$$

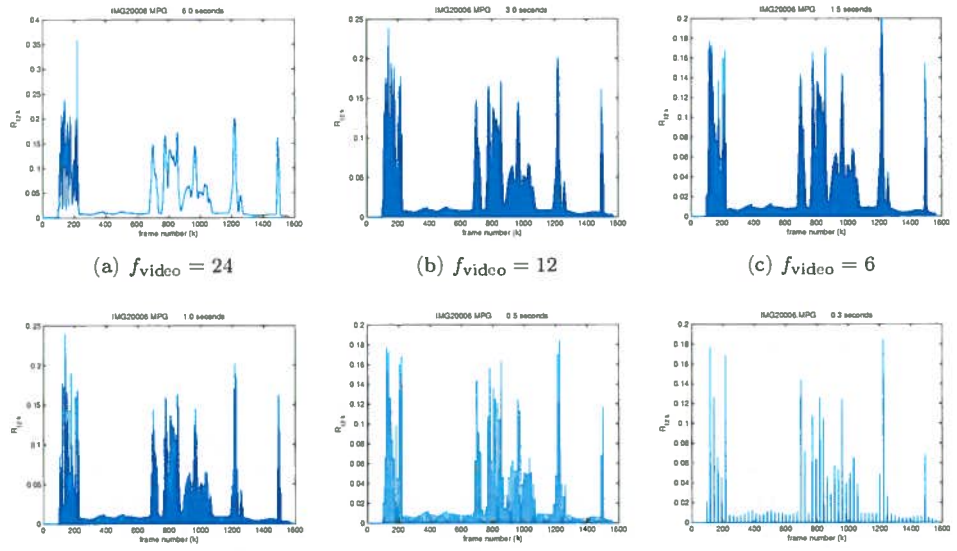
or said differently: filtering is totally skipped. In this video a car was stationary, but when vehicles are driving on, this also needs to be concluded from $R_{12,k}$. Besides that a changing luminious intensity should again be skirted. To test if this technique satisfies these conditions, it is applied to sample video 3. The results for this video are shown in Figure 4.6. These figures show that the technique is working. The value of $R_{12,k}$ is fluctuating when a vehicle is driving on and the luminious intensity does not influence the value of $R_{12,k}$.



(a) $f_{\text{video}} = 24$ (b) $f_{\text{video}} = 12$ (c) $f_{\text{video}} = 6$

(d) $f_{\text{video}} = 4$ (e) $f_{\text{video}} = 2$ (f) $f_{\text{video}} = 1$

Figure 4.5: Various values of f_{video} applied to sample video 14



(a) $f_{\text{video}} = 24$ (b) $f_{\text{video}} = 12$ (c) $f_{\text{video}} = 6$

(d) $f_{\text{video}} = 4$ (e) $f_{\text{video}} = 2$ (f) $f_{\text{video}} = 1$

Figure 4.6: Various values of f_{video} applied to sample video 3

Unfortunately there were no videos available where vehicles are driving around the closed barriers or where people cross the railroad while the barriers are closed. Also there was no video available where a vehicle drives through the red light.

The last problem can be solved by defining at which frame the ligths turn off. In the final scenario this is done by the application. For this test situation, define a k for which the lights turn on and a k for which they turn off for the last time.

Definition 4.4.2. Light_{on}: the value of k for which the lights turn on for the first time.
$\text{Light}_{\text{off}}$: the value of k for which the lights turn off for the last time.

Between these two frames there must be no motion in 'loop 12'. So

$$R_{12,k} \leq L_{\text{mot}}, \quad k = [\text{Light}_{\text{on}} \ldots \text{Light}_{\text{off}}].$$

Example 4.4.3. Now for instance take a video where a train is driving by. By visual scoring it is known at which k the lights turn on and off and at which k the first vehicle is driving on the railroad after the barrier has opened. By setting $\text{Light}_{\text{off}}$ later than the frame where the first vehicle is driving through 'loop 12' a foul is simulated to see if it is detected. The detection of fouls can be added to Algorithm 4.1 resulting in Algorithm 4.2.

Algorithm 4.2: Foul detection [Section 4.4]

```
Input: R_{12,k}, R_{12,k-1}
Output: foul boolean
if R_{stat,k} > L_{mot} then
    if Lights are on then
        foul boolean = 1
    else
        D_k ← Calculate the absolute difference between R_{12,k} and R_{12,k-1}
        if D_k ≤ d then
            counter = counter + 1
        else
            counter = 0
        end
        if counter ≥ W then
            foul boolean = 1
        else
            foul boolean = 0
        end
    end
end
```

4.5 Solid objects

The technique used for the detection of staionary vehicles can also be used to check the presence of solid objects like the andreas cross and fences, see Figure 4.7 and Figure 4.8. The rate at which these areas needs to be checked can be very low, say once in a quarter of an hour. Usually these areas do not detect motion, or at least the difference between R_k and R_{k-1} is small.

If the difference between these two values is not small this denotes that motion is at this place. This can be caused by several possibilities. A first scenario is when for instance pedestrians come by, they are seen as a change in the image in comparison with the background and so they are denoted as motion. A second scenario is when something is broken for instance by vandalism. This scenario also yields to a difference and is denoted as motion.

In the first scenario there is nothing to worry about, since after a little while this motion is not denoted any more. At the other scenario it is not desirable if the motion is not denoted any more.

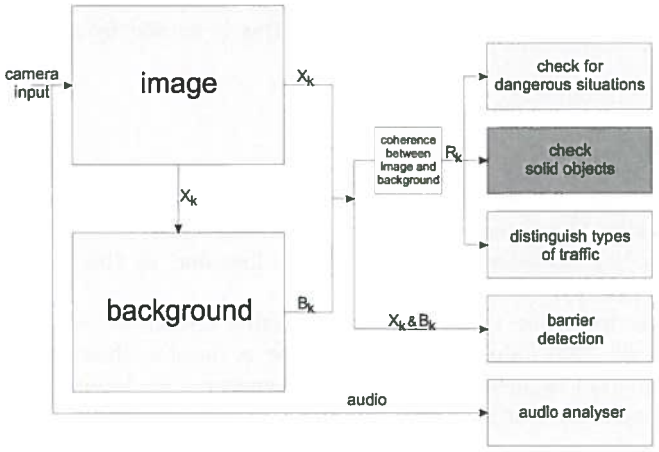


Figure 4.7: Overview of the application (solid objects)

Figure 4.8: Railroad crossing with andreas cross and fence

This situation is solved by increasing the rate at which the images are checked when motion is detected for instance up to once per five minutes. If R_k again differs from R_{k-1}, this is because there was a moving object. If it does not differ this is caused by a solid change, like a broken fence.

Conclusion

Stationary objects may be detected by researching the development of R_k. When a vehicle is driving by this value fluctuates in contrast to the case when a vehicle is stationary. Then the value of R_k does not fluctuate, it is more a flat line and so this makes it possible to detect stagnation at the railroad crossing.

To detect other fouls, like crossing while the lights are on, an extra measuringloop is placed on the railroad itself. This loop checks for detection of motion while the lights are on. This check is skipped when a train is passing, since in this scenario it is obvious that motion is detected in this loop. The frequency sampling for this loop is set to one per second. This is done to save operations.

The check of solid objects can also be done by the technique of stagnation. For future work it is advisable to define seperate areas for these objects self. In this way it is easy to detect at which position these objects are broken.

Chapter 5

Distinguish different types of traffic

Chapter 3 discussed algorithms for detecting objects. The results showed that Algorithm 3.4 based on the correlation between the image, X_k, and the background, B_k, was the best algorithm for detecting objects. As seen in the general results, there are several types of objects passing the railroad crossing. So the next step is to distinguish these different types of traffic, see Figure 5.1. This chapter discusses some methods which are researched to discern these types of objects. There might be several better methods but these are not discussed in this research.

5.1 Algorithms for distinguishing

To distinguish the different types of traffic, several properties of the vehicles are discussed. The next sections research the following attributes of the traffic.

- velocity between loops (see Subsection 5.1.1)
- 'velocity' when entering loop (see Subsection 5.1.2)
- value of R_k (see Subsection 5.1.3)
- length of vehicle (see Subsection 5.1.4)

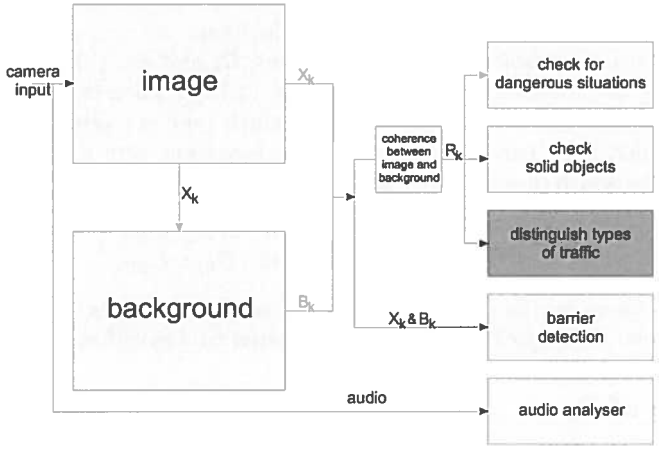


Figure 5.1: Overview of the application (types of traffic)

5.1.1 Velocity between loops

When two measuring loops are placed onto X_k like in Figure 3.3a it is possible to measure the speed (S_j) of the vehicles if the distance between the measuring loops is known. Here j denotes the vehicle number. This speed can tell something about the vehicle type. If the speed is above a fixed limit L_{speed} the vehicle is typed car, while it is typed cyclist if the speed is below L_{speed},

$$\text{type}_j = \begin{cases} \text{car} & \text{if} \quad S_j \geq L_{\text{speed}} \\ \text{cyclist} & \text{if} \quad S_j < L_{\text{speed}} \end{cases}.$$

Since now there are two measuring loops there will also be two indicators of R_k, one for each loop. The indicator where traffic enters the railroad crossing, is denoted by R_{1k}. The second indicator, where the traffic is leaving the railroad crossing, is denoted by R_{2k}.

Algorithm 5.1: Types of traffic (velocity between measuring loops) [Subsection 5.1.1]

Input: R_{1k}, R_{2k}
Output: type of vehicle
if $R_{1k} \geq L_{mot}$ **then**
| frame$_1$:= k
end
if $R_{2k} \geq L_{mot}$ **then**
| frame$_2$:= k
end
Time $\leftarrow$ (frame$_2$ - frame$_1$) divided by f_{video}
$S_j \leftarrow$ Distance divided by time
if $S_j \geq s$ **then**
| type$_j$:= 'car'
else
| type$_j$:= 'cyclist'
end

A disadvantage of Algorithm 5.1 arises if it is getting busy at the railroad crossing. This may cause slow driving cars to be seen as cyclists. So this algorithm is not robust.

5.1.2 "Velocity" when entering loop

Instead of determing the type by the speed between the two measuring loops, it is also possible to take a look at the speed with which the vehicle enters the measuring loop. Since a car is wider, it fills the measuring loop faster and so this may be an indicator for the type. For stagnation detection, see Chapter 4, the difference is determined according to (4.1). For a car this value is larger than it is for a cyclist. By checking if this value is above a fixed limit L_{diff} it is possible to give a type to the vehicle which has just entered the loop.

It is even better not to look at the last two values, R_k and R_{k-1}, but at some more values. For instance by taking the absolute difference of R^k (see (3.15)). Then there is a better approximation of the difference and it can be established better which type the vehicle is. Since a car is usually wider than a cyclist, for a car R_k rises faster in comparison with a cyclist. Even if vehicles are accelarating by the width of a car, the value of D_k will be larger;

$$\text{type}_j = \begin{cases} \text{car} & \text{if} \quad D_k \geq L_{\text{diff}} \\ \text{cyclist} & \text{if} \quad D_k < L_{\text{diff}} \end{cases}.$$

The same can be done by the other measuring loop. If both types are the same there is a clear type for the vehicle. So Algorithm 5.1 can be adjusted to Algorithm 5.2.

5.1.3 Value of R_k

As can be seen in the figures in Section 3.2 up to Section 3.4 the different types of vehicles have different peak values for R_k. This is a good indicator for characterizing the traffic. A car is wider

Algorithm 5.2: Types of traffic (loop entering velocity) [Subsection 5.1.2]

Input: R_{1k}, R_{2k}
Output: type of vehicle
if $R_{1k} \geq L_{mot}$ **then**
  $D_{1k} \leftarrow$ absolute difference of $[\ R_{1,k-F+1} \quad R_{1,k-F-2} \quad \dots \quad R_{1,k-1} \quad R_{1,k}\ ]$
  if $D_{1k} \geq L_{diff}$ **then**
    type$_1$:= 'car'
  else
    type$_1$:= 'cyclist'
  end
end
if $R_{2k} \geq L_{mot}$ **then**
  $D_{2k} \leftarrow$ absolute difference of $[\ R_{1,k-F+1} \quad R_{1,k-F-2} \quad \dots \quad R_{1,k-1} \quad R_{1,k}\ ]$
  if $D_{2k} \geq L_{diff}$ **then**
    type$_2$:= 'car'
  else
    type$_2$:= 'cyclist'
  end
end
if *type*$_1 \equiv$ *type*$_2$ **then**
  type$_j$:= type$_1$
else
  type$_j$:= 'unknown'
end

than a cyclist, so a car covers a larger area of the measuring loop resulting in a higer value for R_k. Two side by side driving cyclists which cross the railroad crossing, cover not that much of the loop as when a car is passing. The peak of R_k for these cyclists is still lower than the peak of R_k when a car is passing. If R_k is larger than L_{car} the vehicle is typed as a car and if R_k is less than L_{car} the vehicle is typed as a cyclist,

$$\text{type}_j = \begin{cases} \text{car} & \text{if} \quad R_k \geq L_{\text{car}} \\ \text{cyclist} & \text{if} \quad R_k < L_{\text{car}} \end{cases}.$$

This technique is described in Algorithm 5.3.

5.1.4 Length of the vehicle

Another property which may characterise the type of the vehicle is its length. The length of a cyclist is usally less than the length of a car. Since the speed of the vehicle is known, S_j, and it is possible to see for how long the vehicle is in the loop, t_j, the length of the vehicle is known.

$$\text{length of vehicle } j = S_j t_j$$

When the length is above a fixed limit, L_{length}, the vehicle can be typed as a car. Otherwise it is denoted as a cyclist,

$$\text{type}_j = \begin{cases} \text{car} & \text{if length of vehicle } j \geq L_{\text{length}} \\ \text{cyclist} & \text{if length of vehicle } j < L_{\text{length}} \end{cases}.$$

This technique is described in Algorithm 5.4.

Algorithm 5.4 also uses the calculation of S_j from Algorithm 5.1, so it is useful to combine these techniques. It is even better to combine all four properties. The more properties to be checked the better the result is.

Algorithm 5.3: Types of traffic (value of R_k) [Subsection 5.1.3]

Input: R_{1k}, R_{2k}
Output: type of vehicle
if $R_{1k} \geq L_{mot}$ **then**
  if $R_{1k} < L_{car}$ **then**
    type$_1$:= 'cyclist'
  else
    type$_1$:= 'car'
  end
end
if $R_{2k} \geq L_{mot}$ **then**
  if $R_{2k} < L_{car}$ **then**
    type$_2$:= 'cyclist'
  else
    type$_2$:= 'car'
  end
end
if *type*$_1 \equiv$ *type*$_2$ **then**
  type:= type$_1$
else
  type:= 'unknown'
end

5.1.5 Combining the algorithms

Algorithms 5.1 up to 5.4 can be combined to one general algorithm. So the properties for the velocity between the loops (S_j), the 'velocity' when entering the loop, the value of R_k and the length of the vehicle all define a 'type$_1$' and a 'type$_2$'. Comparing all these properties every time a vehicle enters the loop is useful but takes a lot of operations.

Even better is to clasify the algorithms first. This is done by using a scatter plot, see [13]. For every available video where traffic is driving by, visual scoring is done. The results are shown in the first collumn of Table 5.3. For every crossing vehicle its length$_1$, length$_2$, D_{k1}, D_{k2}, R_{k1} and R_{k2} are measured. The speed, S_j, is left out, since this is already processed in length$_{1j}$ and length$_{2j}$. This data is plotted in 3D-scatter plots, see Figure 5.2. The different colors of the points in these figures stand for different types of traffic. Table 5.1 gives an explanation of these colors.

Remark 5.1.1. By the results of $R_{i,k}$ $(i = 1, 2)$ for alle videos (Appendix A.5) and visual scoring it was possible to choose a value for L_{mot},

$$L_{\text{mot}} := 0.2.$$

This value works fine for almost every tested video except for sample video 6. The luminious intensity of this video was lower and so R_{1k} is multiplied by a factor two, otherwise $R_{i,k}$ $(i = 1, 2)$ does not reach L_{mot}.

It might happen that a car and a cyclist are driving side by side or just consecutively such that the cyclist is denoted first. What happens then is that it is typed as a car from the value of $R_{1,k}$ but by $D_{1,j}$, the entering velocity part, it is typed as a cyclist. Since the car is driving faster this car reaches the second measuring loop before the cyclist does. So the second measuring loop denotes in both cases 'car'. Since the check for the speed also announces 'car', the output of Algorithm 5.5 is in this case 'car'. The consequence is that the cyclist is not noticed. So an extra type (instead of only 'car' and 'cyclist') is defined called 'car + cyclist'. This occurs when the type is 'car' according to the value of $R_{i,k}$ for $i = 1, 2$ and the type is 'cyclist' by the value of $D_{i,k}$ for $i = 1, 2$.

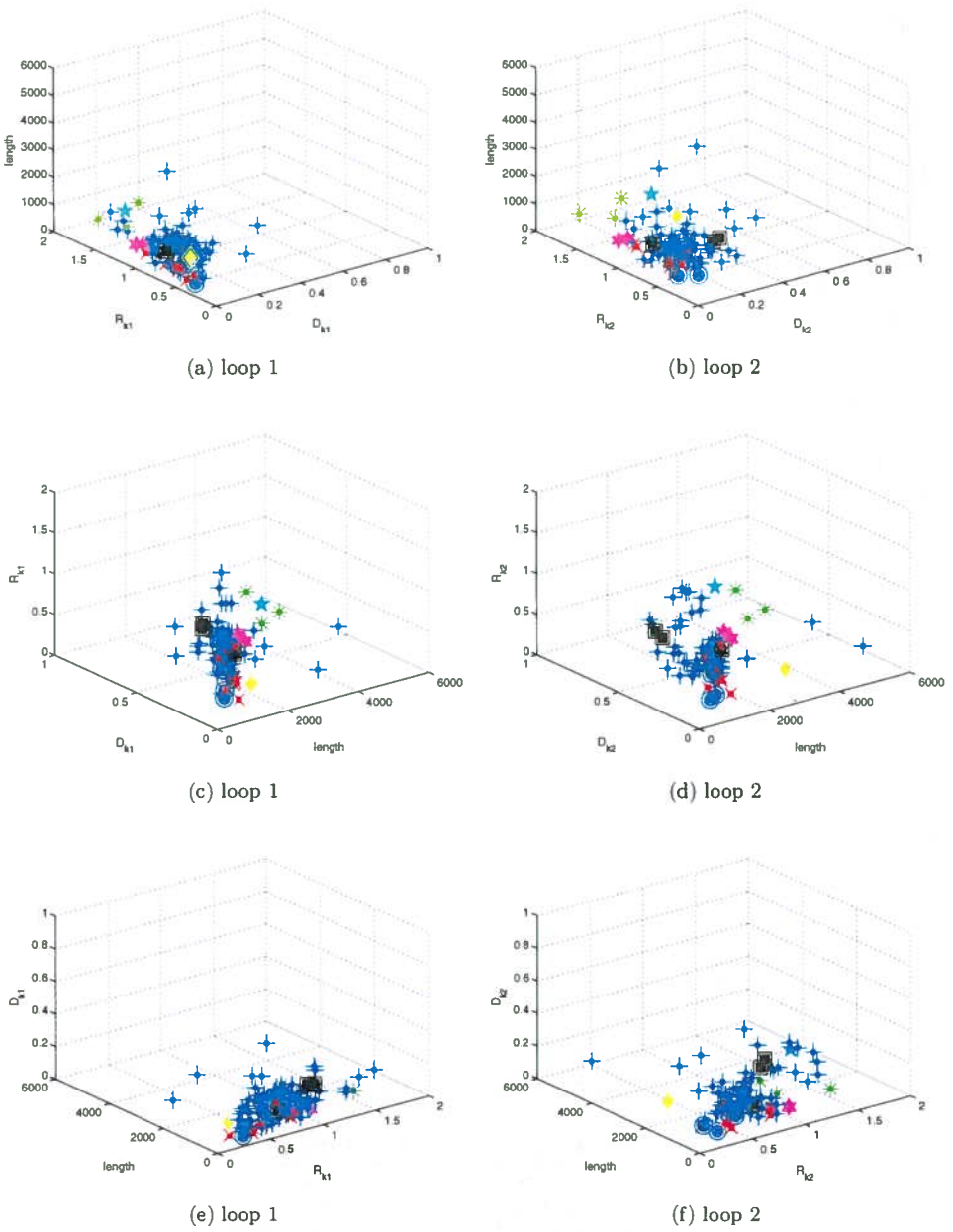
The plots in Figure 5.2 do not give a clear overview if the vehicles are scattered. For this reason also 2D-scatter plots have been made. The results are shown in Figure 5.3. The colors are the same as explained in Table 5.1.

Algorithm 5.4: Types of traffic (length of the vehicle) [Subsection 5.1.4]

Input: R_{1k}, R_{2k}
Output: type of vehicle
if $R_{1k} \geq L_{mot}$ **then**
| timeOn$_1$:= k
end
if $R_{1k} < L_{mot}$ **then**
| timeOff$_1$:= k
end
$t_{1j} \leftarrow$ timeOn$_1$ - timeOff$_1$
if $R_{2k} \geq L_{mot}$ **then**
| timeOn$_2$:= k
end
if $R_{2k} < L_{mot}$ **then**
| timeOff$_2$:= k
end
$t_{2j} \leftarrow$ timeOn$_2$ - timeOff$_2$
Time $\leftarrow$ (timeOn$_2$ - timeOn$_1$) divided by f_{video}
$S_j \leftarrow$ Distance divided by Time
length$_{1j} \leftarrow S_j t_{1j}$
length$_{2j} \leftarrow S_j t_{2j}$
if $length_{1j} \geq L_{length}$ **then**
| type$_1$:='car'
else
| type$_1$:='cyclist'
end
if $length_{2j} \geq L_{length}$ **then**
| type$_2$:='car'
else
| type$_2$:='cyclist'
end
if $type_1 \equiv type_2$ **then**
| type$_j$:= type$_1$
else
| type$_j$:= 'unknown'
end

Table 5.1: Legend for the scatter plots

vehicle type	symbol	color
car	+	blue
cyclist	×	red
car + cyclist	*	green
moped	○	blue
lorry	□	black
two cars	pentagram	cyan
two cyclists	hexagram	magenta
two walkers	◇	yellow

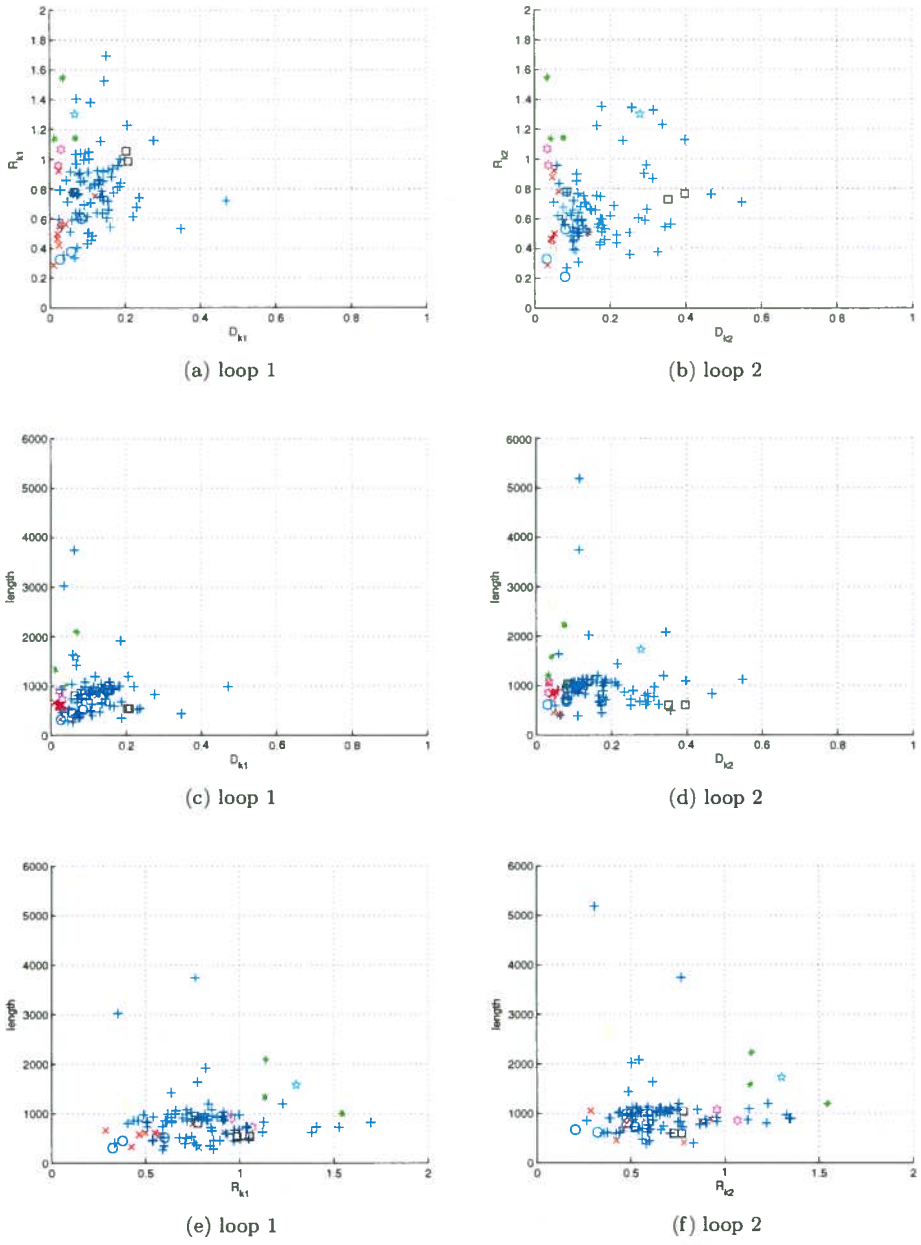


(a) loop 1 (b) loop 2

(c) loop 1 (d) loop 2

(e) loop 1 (f) loop 2

Figure 5.2: 3D scatter plots



(a) loop 1 (b) loop 2

(c) loop 1 (d) loop 2

(e) loop 1 (f) loop 2

Figure 5.3: 2D scatter plots

The results in Figure 5.3 show that the cars and cyclists are clustered the best when $D_{k,i}$ is plotted against $R_{k,i}$. So $D_{k,i}$ and $R_{k,i}$ for $i = 1, 2$ might be the best methods to classify the traffic. The length$_{i,j}$ $(i = 1, 2)$ can be used to detect two consecutively or side by side driving vehicles. The combination of Algorithms 5.2 up to 5.3 leads to four different checks of the type;

- type$_{1,D}$:= type of the vehicle on loop 1 based on D_{1k}
- type$_{2,D}$:= type of the vehicle on loop 2 based on D_{2k}
- type$_{1,R}$:= type of the vehicle on loop 1 based on R_{1k}
- type$_{2,R}$:= type of the vehicle on loop 2 based on R_{2k}

The combinations of these types leads to several possibilities. For instance when the vehicle is typed 'car + cyclist' at the first measuring loop and typed 'car' at the second measuring loop. The chance is great that this was a car and a cyclist. Some other possibilities are explounded in Table 5.2. Now the algorithms can be combined into one general algorithm shown in Algorithm 5.5.

Table 5.2: Several possibilities of type combinations

type$_{1,D}$	type$_{1,R}$	type$_1$	type$_{2,D}$	type$_{2,R}$	type$_2$	result
cyclist	cyclist	cyclist	cyclist	cyclist	cylist	cyclist
car	car	car	car	car	car	car
cyclist	cyclist	cyclist	car	car	car	depends on speed
car	car	car	cyclist	cyclist	cyclist	depends on speed
cyclist	car	car + cyclist	cyclist	car	car + cyclist	car + cyclist
cyclist	car	car + cyclist	car	car	car	car + cyclist
car	car	car	cyclist	car	car + cyclist	car + cyclist
cyclist	cyclist	cyclist	cyclist	car	car + cyclist	car + cyclist
cyclist	car	car + cyclist	cyclist	cyclist	cyclist	two cyclists

5.2 Placement of measuring loops

The current application has a small measuring loop [9]. For this new situation there should be placed a loop over the whole road, just as in Figure 3.3a. In this way every vehicle drives through the measuring loop. Since the camera is not standing straight ahead on the railroad crossing, it is better to place a loop such that it is a bit obliquely. For easy programming there has been taken five measuringloops placed such that it forms a skew measuring loop. Figure 5.4 shows this idea. Since there are now five measuring loops at each side and two measuring positions there are also ten different values of $R_{i,k}$, $i \in 1, 2, \ldots, 10$. All ten are treated as described before.

5.3 Test results for the algorithm

For testing Algorithm 5.5 (combined with Table 5.2) the values of $R_{i,k}, i \in 1, 2, \ldots, 10$ are calculated based on correlation, Algorithm 3.4 (Subsection 3.1.4). The types of the vehicles are determined with Algorithm 5.5. The script contains some extra commands which are not described in Algorithm 5.5 these are used to display and store the measured data correctly.

Before Algorithm 5.5 can be tested, first some variables as $L_{\text{car}}, L_{\text{diff}}, L_{\text{speed}}, L_{\text{length}}, \beta, \hat{\beta}$ must be defined. These variables can be defined using the scatter plots in Figure 5.3. The scatter plots can be used to see how many red $(\times)$, green $(*)$ and magenta (hexagram) points lie above L_{diff} and how many blue $(+)$, black $(\square)$ and cyan (pentagram) points lie below L_{diff}. By minimizing the sum over these two values the optimal L_{diff} is found. This same technique can be done for the value of L_{car}. As said before a value for L_{mot} was chosen to be 0.2 based on visual scoring. Also

Algorithm 5.5: Types of traffic (combination of algorithms) [Subsection 5.1.5]

Input: R_{1k}, R_{2k}
Output: type of vehicle
if $R_{1k} \geq L_{mot}$ **then**
  $D_{1k} \leftarrow$ absolute difference of $[\ R_{1,k-F+1} \quad R_{1,k-F-2} \quad \ldots \quad R_{1,k-1} \quad R_{1,k}\ ]$
  $\text{frame}_1 :=$ k
  if $D_{1k} \geq L_{diff}$ **then**
    $\text{type}_{1,D} :=$ 'car'
  else
    $\text{type}_{1,D} :=$ 'cyclist'
  end
  if $R_{1k} \geq L_{car}$ **then**
    $\text{type}_{1,R} :=$ 'car'
  else
    $\text{type}_{1,R} :=$ 'cyclist'
  end
end
if $R_{2k} \geq L_{mot}$ **then**
  $D_{2k} \leftarrow$ absolute difference of $[\ R_{2,k-F+1} \quad R_{2,k-F-2} \quad \ldots \quad R_{2,k-1} \quad R_{2,k}\ ]$
  $\text{frame}_2 =$ k
  if $D_{2k} \geq L_{diff}$ **then**
    $\text{type}_{2,D} :=$ 'car'
  else
    $\text{type}_{2,D} :=$ 'cyclist'
  end
  if $R_{2k} \geq L_{car}$ **then**
    $\text{type}_{2,R} :=$ 'car'
  else
    $\text{type}_{2,R} :=$ 'cyclist'
  end
end
$S_j \leftarrow$ distance times f_{video} divided by (frame_2 - frame_1)
switch $type_{1,D}$, $type_{1,R}$, $type_{2,D}$, $type_{2,R}$ **do**
  case $type_{1,D} \equiv type_{2,D}$ *and* $type_{1,R} \equiv type_{2,R}$
    if $type_{1,D} \equiv type_{2,D}$ **then**
      result:= $\text{type}_{1,D}$
    else
      if $S_j \geq L_{speed}$ **then**
        result:= 'car'
      else
        result:= 'cyclist'
      end
    end
  case $type_{1,D} \neq type_{1,R}$ *and* $type_{2,D} \neq type_{2,R}$
    if $S_j \geq L_{speed}$ **then**
      result:= 'car'
    else
      result:= 'cyclist'
    end
  case $type_{1,D} \neq type_{1,R}$ *xor* $type_{2,D} \neq type_{2,R}$
    if $type_{1,D} \equiv type_{2,R}$ **then**
      result:= $\text{type}_{1,D}$
    else
      result:= $\text{type}_{2,R}$
    end
  end
end

Figure 5.4: X_1 of sample video 2 with extended measuring loops

the value for L_{speed} is chosen based on experiences. This yields the following set of variables,

$$\begin{aligned} \beta &:= 0.0001 \\ \hat{\beta} &:= \tfrac{\ln 100}{24} \\ L_{\text{mot}} &:= 0.2 \\ L_{\text{diff}} &:= 0.05 \\ L_{\text{speed}} &:= 30 \\ L_{\text{car}} &:= 0.6, \end{aligned}$$

is used to test Algorithm 5.5 (combined with Table 5.2). Table 5.3 displays the results.

Remark 5.3.1. For sample video's 3, 4, 5 and 7 the measurements did not start at $k = 1$ but at another value. This is because first a train is crossing and to prevent that the train is detected as a car by the algorithm the videos started somewhat later. In the column reality there is a term *'(cyclist)'* this is because a cyclist is crossing loop 1. But does not cross loop 2, see Figure 5.5.

Figure 5.5: X_{1360} from sample video 13

Some other remarkable facts are for instance that the stationary car in sample video 14 is detected as two cars. This is because the check for stagnation was not implemented in the script for traffic detection. Something else are the two pedestrians in sample video 8. These two are walking on the road for a short time such that they are detected as vehicles, normally these two are not noticed. An extension for the traffic detection is by introducing an extra measuringloop placed above the other measuringloops. If the value of R_k for the lower and upper loop are both above their limit then a high vehicle is driving by. In this way it is possible to detect 'high traffic' like lorries.

Video	Reality	Algorithm
Sample video 1	lorry - car	car - car
Sample video 2	car - car + cyclist - car	car - car + cyclist - car
Sample video 3 (start 250)	car - car - two cars driving consecutively - cyclist + car - two cyclists - car - car	car - two cars - car + cyclist - car + cyclist - car - car
Sample video 4 (start 800)	cyclist - car - car - car - cyclist - two cyclists - car	cyclist - car - car - car - cyclist - two cyclists - car
Sample video 5 (start 900)	cyclist + car - car - car - car - car - car - car - car - car - moped - car - car - car - car - car - car - car - car - car	two cars - car - car - car - car - car - car - car - car - car - car - car - car - car - car - car - car - car - car - car
Sample video 6 (loop 1 ×2)	moped - car - car - car - lorry - car - car - car	car - cyclist - car + cyclist - cyclist - car - car - car - car
Sample video 7 (start 100)	car - car - car + trailer - car - car - car - car - car - car - car - car	car - car - car - car - car - car - car - car - car - car - car
Sample video 8	cyclist - two pedestrians	car + cylist - car
Sample video 9	moped - car	car - car
Sample video 10 (start 750)	car - car	car - car
Sample video 11	car - car - cyclist - car - car - cyclist - cyclist	car - car - cyclist - car - car - cyclist - cyclist
Sample video 12	car - cyclist - car - car - cyclist - car	car - cyclist - car - car + cyclist - car - car
Sample video 13	cyclist - car - car - car - car - car - car - car - car - car - (cyclist) - car - car - car - cyclist - car - car - car - car - car - car - car - car - car - car - car - car - two cyclist + lorry - car - car	cyclist - car - car - car - car - car - car - car - car - car - car - car - car - car - car - car + cyclist - car - car - car - car - car - car - car - car - car - car - car - car - two cyclists - car - car
Sample video 14	lorry - stationary car	car - two cars

Table 5.3: Results of the algorithm for type detection of the vehicles

Conclusion

From Table 5.3 the conclusion can be drawn that Algorithm 5.5 (combined with Table 5.2) is in 88% of the cases correct. Some cars are typed as cyclists and some cyclists are typed as cars. This has to do with the luminious intensity of the video's, since the intensity is not the same everywhere. This yields that the values of R_k are not the same in every video. A hundred percent result (with this algorithm) is therefore almost impossible since in every situation there are different situations for the environment. These results are the optimal result for the tested video's with for every video the same settings.

In future work it is advisable to implement a calibration for the video's. Adaptation of the variables to the luminious intensity yields even a better result, but problem is how to find a relation between the luminious intensity and the variables for the distinguishment of the objects. As shown in Table 5.3 for sample video 6 R_{1k} is for instance already multiplied by a factor two. This can be skirted by dividing the limits by two for this video. But then all other video's give a worse result.

As discussed in the conclusion of Chapter 3 it is unknown how R_k^{CORR} will respond to other weather situations. Besides that there might be other methods to clasify the traffic which are not investigated in this research.

Chapter 6

Barrier detection

Previous chapters discussed the detection and distinction of the several types of traffic. As discussed in the introduction another important check of the application is the detection of the barrier. See also the overview in Figure 6.1. This detection is used to see if the barrier is making the correct movement. There are several techniques which can be used for detecting the barrier movement, in terms of the angle α. This chapter treats some of these techniques, namely:

- Detection based on masks (see Section 6.1)
- Detection based on least square fitting (see Section 6.2)
- Detection based on total least square fitting (see Section 6.3)

6.1 Detection based on masks

The current application uses the technique of masks to check the barrier [9]. For each barrier angle a mask is generated beforehand. The barrier is then easily detected by checking which mask matches best. Figure 6.2 shows this schematically. Here ten different masks are taken, in reality ninety masks are used for a better accuracy.

One way to speed this up is by checking a certain area around the last detected angle. (If the angle at X_i was α then the angle at X_{i+1} lies around α.) This accelerates the application, since fewer masks needs to be checked. A disadvantage of checking fewer masks is that large errors (for instance by a technical disturbance the angle is lowered a lot in one frame) are not detected. This may cause that the application is in alarm status while nothing special is going on.

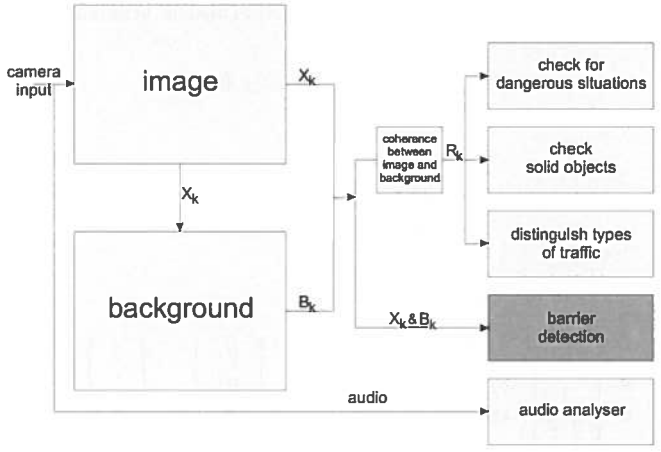


Figure 6.1: Overview of the application (barrier)

Figure 6.2: Sample video 10, X_{200} with masks

The technique of the detection by masks is summarized in Algorithm 6.1. (Algorithm 6.1 can be accelarated by checking the masks around the last α instead of all possible masks between 0° and 90°.)

Algorithm 6.1: Barrier detection based on masks [see Section 6.1]

Input: Masks, Coordinates x and y
Output: α
for $i \leftarrow 0$ **to** 90 **do**
  foreach *pair* (x, y) **do**
    if $(x,y) \in Mask_i$ **then**
      $\text{counter}_i = \text{counter}_i + 1$
    end
  end
end
$\alpha = \{i \,|\text{counter}_i \text{ is maximal}\}$

6.2 Detection based on the least squares method

Another way to detect the barrier is by using the least square method [17]. If there is a set consisting of x and y coordinates the least square method is able to find a straight line through these points,

$$y_{\text{best}}(x) = a + bx$$

such that

$$\sum_{i=1}^{l} (y_{\text{best}}(x_i) - y_i)^2$$

is minimized. This can be rewritten in

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_l \end{bmatrix} = \begin{bmatrix} a \\ a \\ a \\ \vdots \\ a \end{bmatrix} + b \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_l \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \epsilon_l \end{bmatrix} \Rightarrow \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_l \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} a + b \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_l \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \epsilon_l \end{bmatrix}$$

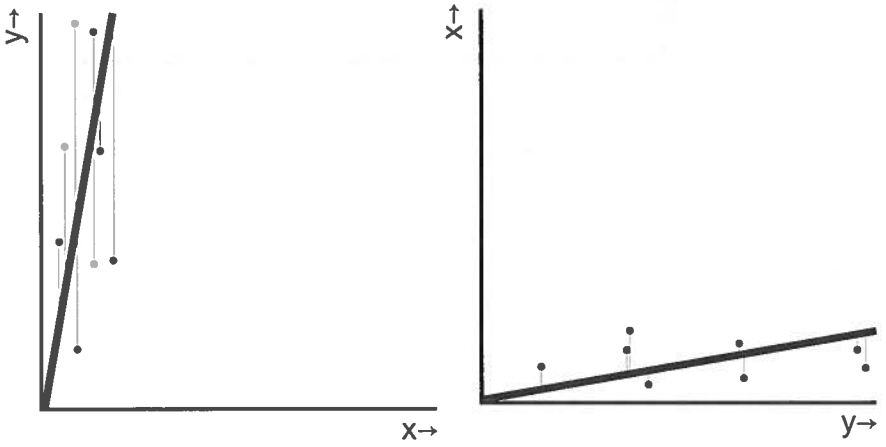
and in matrix formula, $Ax = b$,

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_l \end{bmatrix}}_{b} \approx \underbrace{\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ & \vdots \\ 1 & x_l \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{x} \tag{6.1}$$

such that it can be solved easily, where l is the number of found coordinates for the barrier. If a and b are known it is easy to determine the angle α by

$$\alpha = \arctan(b)\frac{180}{\pi}. \tag{6.2}$$

A problem for this method is the rotation. When α approaches 90° the distance between the coordinates and the ideal line is very large since the least square method is looking at the vertical offset. Figure 6.3a shows schematicly what is happening when α approaches 90°. One way to solve this problem is by flipping the axis, as has been done in Figure 6.3b. In this way the errors are less, such that the coefficients are more accurate. Unfortunately there is no good value for k at which the axis should be flipped.



(a) Normal schedule (b) Flipped schedule

Figure 6.3: Schematic representation of $\alpha \approx 90°$

Algorithm 6.2 shows an algorithm to calculate α based on the least square method.

Algorithm 6.2: Barrier detection based on least squares [see Section 6.2]

Input: Coordinates x and y
Output: α
a,$b \leftarrow$ least squares solution to (6.1)
$\alpha = \arctan(b)\frac{180}{\pi}$

6.3 Detection based on the total least squares method

The total least squares method is based on the perpendicular offset of the data in relation to the best line [14, 18]. Figure 6.4 shows this schematically.

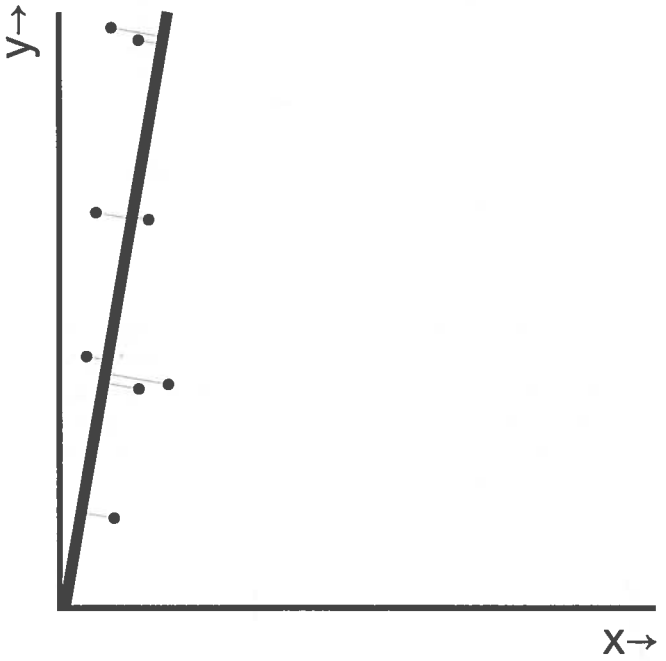


Figure 6.4: Schematic representation of perpendicular offset

The slope based on the total least square method can be calculated using the **S**ingular **V**alue **D**ecomposition (SVD). SVD is a matrix-factorisation. More information about SVD can be found in [14]. According to Theorem 6.3.1 the best approximation with respect to the perpendicular offset is given by a column vector times a row vector.

Theorem 6.3.1. *Given a matrix $A \in \mathbb{R}^{n\times m}$. A rank one matrix K minimizes the Frobenius norm ([14]) of $A - K$,*

$$\|A - K\|^2_{FROB} \tag{6.3}$$

iff

$$K = U_1 \sigma_1 V_1^*$$

with U, σ and V from the SVD of A. Where U_1 is the first column of U, σ_1 is the first singular value of A and V_1 is the first column of V. □

This is also known as the Eckart-Young-Mirsky matrix approximation theorem [11, 14]. First a lemma is defined which is needed to proof Theorem 6.3.1.

Lemma 6.3.2. *Let w be a column with norm 1 and σ_1 be the first singular value of a matrix A then there holds;*

$$\sup_{\|w_1=1\|} w_1^* (A^* A) w_1 = \sigma_1^2.$$

Proof of Lemma 6.3.2. Take USV as a SVD of A. Since a unitary matrix preserves the length of a norm;

$$\|V^* w_1\| = \|w_1\| = 1.$$

Implementation of the SVD yields;

$$w_1^* (A^* A) w_1 = w_1^* (VSSV^*) w_1 = g^* S^2 g,$$

where $g = V^* w_1$ and

$$\|g\| = 1 \iff \|w_1\| = 1.$$

So the maximum for $g^* S^2 g$ is σ_1^2.

■

Proof of Theorem 6.3.1. Let $K = USV^*$ be the SVD of K. Split matrix V into $\begin{bmatrix} w_1 & W \end{bmatrix}$, where w_1 denotes a column.

The squared Frobenius norm of $A - K$ is given by;

$$\|A - K\|_{\text{FROB}}^2 .$$

Since the norm is unitarily invariant this yields

$$\|A - K\|_{\text{FROB}}^2 = \|(A - K)V\|_{\text{FROB}}^2 \geq \|(A - K)W\|_{\text{FROB}}^2 . \tag{6.4}$$

Since K has rank one, there holds $KW = 0$. So (6.4) turns into

$$\|AW\|_{\text{FROB}}^2 = \text{tr}\,(AWW^*A^*) \tag{6.5}$$

Due to the fact that V is a unitary matrix there holds,

$$VV^* = \begin{bmatrix} w_1 & W \end{bmatrix} \begin{bmatrix} w_1^* \\ W^* \end{bmatrix} = w_1 w_1^* + WW^* = I. \tag{6.6}$$

Implementation of (6.6) into (6.5) yields,

$$\|AW\|_{\text{FROB}}^2 = \text{tr}\,(A\,[I - w_1 w_1^*]\,A^*) = \text{tr}\,(AA^*) - w_1^*\,(A^*A)\,w_1$$

It follows that $\|AW\|_{\text{FROB}}^2$ is minimal if and only if

$$w_1^*\,(A^*A)\,w_1$$

is maiximal. Since w_1 has norm one by Lemma 6.3.2,

$$w_1^*\,(A^*A)\,w_1 \leq \sigma_1^2,$$

with $sigma_1$ the first singular value of A. So

$$\|A - K\|_{\text{FROB}}^2 \geq \text{tr}\,(AA^*) - \sigma_1^2 \qquad \forall K$$

And with $w_1 = V_1$ (the first column of V from the SVD of A) otherwise;

$$K = U_1 \sigma_1 V_1^* \quad \text{From the SVD of } A,$$

equality is achieved. Which completes the proof. ■

Before the matrix A is defined first the centered x and y need to be defined.

Definition 6.3.3. Given vector x, the centered $\tilde{x}$ is given by

$$\tilde{x} := x - \bar{x}.$$

Where $\bar{x}$ is the mean of x.

Now define the matrix A by

$$A := [\tilde{x} \quad \tilde{y}].$$

Then the best approximation for A with rank 1 is defined by

$$K := U_1 \sigma_1 V_1^*,$$

with $U \in \mathbb{R}^{l\times 2}$, $\Sigma \in \mathbb{R}^{2\times 2}$, $V \in \mathbb{R}^{2\times 2}$. Since K has rank 1, the second column is a multiple of the first column. The proportion between these two values is the slope of the approximated line and is given by

$$b = \frac{V_{2,1}}{V_{1,1}}.$$

This will give an error if $V_{1,1} = 0$. Since V is an unitairy matrix there holds

$$V^*V = I.$$

Where V^* is the conjugate tranpose of V and I is the identity or unit matrix. So

$$V_{1,1}^2 + V_{2,1}^2 = 1,$$

this means that $V_{1,1}$ and $V_{2,1}$ lie on an unit circle, see Figure 6.5. As can be seen in Figure 6.5,

$$\alpha = \arctan\left(\left|\frac{V_{2,1}}{V_{1,1}}\right|\right)\frac{180}{\pi} = \arcsin|V_{2,1}|\frac{180}{\pi} = \arccos|V_{1,1}|\frac{180}{\pi}.$$

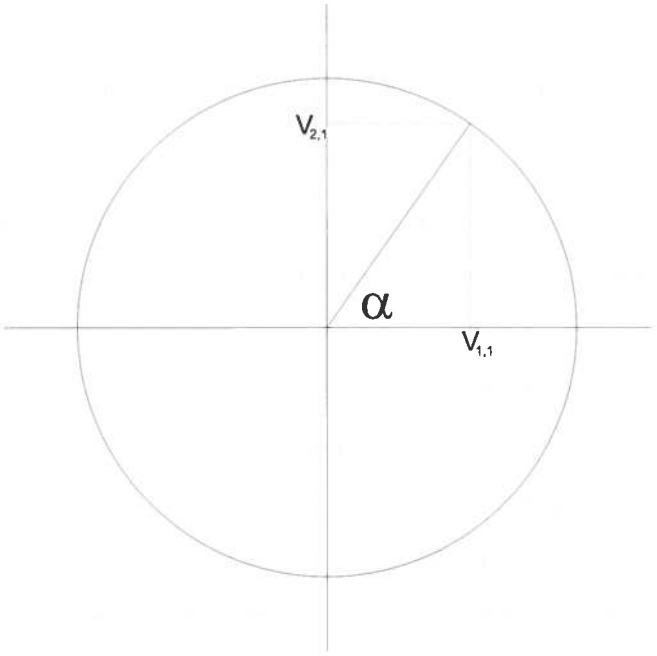


Figure 6.5: Unit circle for $V_{2,1}$ and $V_{1,1}$

The detection of α by the total least square method can be written in Algorithm 6.3.

Algorithm 6.3: Barrier detection based on total least squares [see Section 6.3]

Input: Co-ordinate vectors x and y
Output: α
$\tilde{x} = x - \bar{x}$
$\tilde{y} = y - \bar{y}$
$[U, S, V] = \text{SVD}\begin{pmatrix}\tilde{x} & \tilde{y}\end{pmatrix}$
$\alpha = \arcsin|V_{2,1}|\frac{180}{\pi}$

A disadvantage of this method is the implementation into C#. The SVD is not a standard function in C#. Since the matrix V is a matrix of eigenvectors the calculation of eigenvalues and

eigenvectors may be used. First define a matrix B by

$$B := A^*A = \begin{bmatrix} \|\tilde{x}\|^2 & \langle \tilde{x}, \tilde{y} \rangle \\ \langle \tilde{x}, \tilde{y} \rangle & \|\tilde{y}\|^2 \end{bmatrix}$$

Determine the largest eigenvalue, λ_1, of B using the characteristic polynominal or the eigenvalue solvers [1, 14, 19]. The corresponing eigenvector is found by

$$\begin{bmatrix} B - \lambda_1 \end{bmatrix} \begin{bmatrix} V_{1,1} \\ V_{2,1} \end{bmatrix} = 0$$

This eigenvector needs to be normalized for a unitary basis.

$$\begin{bmatrix} V_{1,1} \\ V_{2,1} \end{bmatrix} = \frac{1}{\left\| \begin{bmatrix} V_{1,1} \\ V_{2,1} \end{bmatrix} \right\|} \begin{bmatrix} V_{1,1} \\ V_{2,1} \end{bmatrix} \tag{6.7}$$

The technique is described in Algorithm 6.4.

Algorithm 6.4: Barrier detection based on total least squares with use of eigenvalues [see Section 6.3]

Input: Co-ordinate vectors x and y
Output: α
$\tilde{x} = x - \bar{x}$
$\tilde{y} = y - \bar{y}$
$A = \begin{bmatrix} \tilde{x} & \tilde{y} \end{bmatrix}$
$B := A^*A$
$\lambda_1 \leftarrow$ largest eigenvalue of B
$\begin{bmatrix} V_{1,1} \\ V_{2,1} \end{bmatrix} \leftarrow$ eigen vector corresponding to λ_1
Normalize the eigen vector using (6.7)
$\alpha = \arcsin |V_{2,1}| \frac{180}{\pi}$

6.4 Detection based on L_1-approximation

Instead of minimizing the squared distance between the approximated line and the coordinates, the L_1-approximation minimizes absolute distance between the approximated line and the coordinates,

$$\min \sum_{i=1}^{l} |y_i - (a + bx_i)|. \tag{6.8}$$

Since the absolute distance is minimized heavy peaks have less influence to the best line approximation. This type is an exmaple of Linear Programming (LP) problem. In general a LP-problem is defined as

$$\begin{aligned} \text{minimize} \quad & f^T x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0. \end{aligned} \tag{6.9}$$

More information about LP-problems can be found in [15].

To rewrite (6.8) into the form of (6.9), express y_i as

$$y_i = a + bx_i + u_i - v_i, \quad u_i \geq 0, v_i \geq 0 \quad i \in 1, \ldots, l.$$

The new objective is to minimize the sum of u and v. Written as LP-problem:

$$\begin{aligned}
\text{minimize} \quad & \begin{bmatrix} 0 & 0 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ u_1 \\ \vdots \\ u_l \\ v_1 \\ \vdots \\ v_l \end{bmatrix} \\
\text{subject to} \quad & \begin{bmatrix} 1 & x_1 & 1 & 0 & \dots & 0 & -1 & 0 & \dots & 0 \\ 1 & x_2 & 0 & 1 & \dots & 0 & 0 & -1 & \dots & 0 \\ \vdots & \vdots & & & \ddots & & & & \ddots & \\ 1 & x_l & 0 & 0 & \dots & 1 & 0 & 0 & \dots & -1 \end{bmatrix} \begin{bmatrix} a \\ b \\ u_1 \\ \vdots \\ u_l \\ v_1 \\ \vdots \\ v_l \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_l \end{bmatrix} \\
& u \geq 0 \quad v \geq 0
\end{aligned} \tag{6.10}$$

One way to solve this LP-problem is by using the Simplex method. More information about the Simplex method can be found in [10].

Algorithm 6.5 shows the calculation of α based on the L_1 approximation.

Algorithm 6.5: Barrier detection based on linear programming [see Section 6.4]

Input: Co-ordinate vectors x and y
Output: α
a,b $\leftarrow$ solution of the LP-problem (6.10)
$\alpha = \arctan(b) \frac{180}{\pi}$

6.5 Finding the coordinates

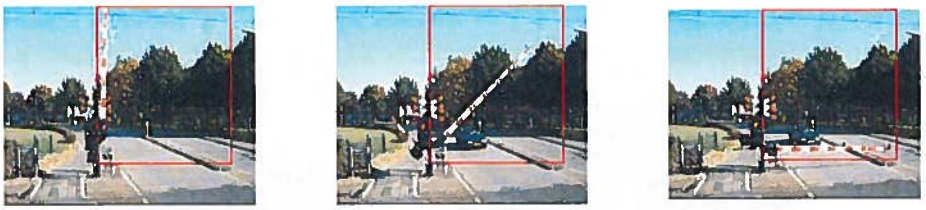
Sections 6.1 up to 6.4 describe algorithms for calculating the angle α. As described these algorithms need coordinates (x_i, y_i) for the calculation. Chapter 2 already specified an algorithm for background adaptation. By comparing the current background with the current image it may be possible to high light the barrier such that it can be detected. To detect the coordinates of the barrier this section treats some methods.

6.5.1 Threshold

The easiest way to find the coordinates of the barriers is by using the threshold algorithm described in Subsection 3.1.1. Make every pixel white when its pixelvalue is above the threshold and black if the pixelvalue is below the threshold,

$$\begin{aligned}
I(i,j,c,k) &:= \left| X(i,j,c,k) - \check{B}(i,j,c,k) \right| \\
I_\theta(i,j,k) &:= \begin{cases} 0 & \text{if } \sum_{c=1}^{3} I(i,j,c,k) \leq \theta \\ 255 & \text{if } \sum_{c=1}^{3} I(i,j,c,k) > \theta \end{cases} .
\end{aligned} \tag{6.11}$$

The positions of these white pixels are the coordinates needed. It is not useful to check the whole image, but only check the loop where the barrier is moving. See also Figure 6.6, which displays the measuring loop for various frames.



(a) X_1 (b) X_{200} (c) X_{400}

Figure 6.6: Various frames for sample video 10

6.5.2 Background

To determine this background Algorithm 2.2 (exponential adaptation) on page 22 is used. For the detection of the barrier a faster background adaption, β, is desired. Since with a larger β the background adapts faster such that the influence of the previous frames is less. Here β is set to $\frac{\ln 100}{1000} \approx 0.0046$, which corresponds to $t_{\text{set,barrier}} \approx 41.67$ seconds.

Algorithm 3.1, based on threshold, uses the absolute difference between the image and the background,

$$|X_k - B_k|.$$

A disadvantage of this technique occurs when the barrier is starting to move. Since the absolute difference is taken the original barrier remains visible in this difference. This may lead to a wrong detection of the angle. Figure 6.7c shows the absolute difference for $k = 200$. A solution to this problem is to take the 'normal' difference between the image and the background,

$$X_k - B_k.$$

By the absolute difference the barrier remains visible. Since negative values become positive. The normal difference sets negative values to zero resulting in more black pixels. Figure 6.7d shows the 'normal' difference for $k = 200$. Then (6.11) needs to be redefined as

$$\begin{aligned} \hat{I}(i,j,c,k) &:= X(i,j,c,k) - \check{B}(i,j,c,k) \\ \hat{I}_\theta(i,j,k) &:= \begin{cases} 0 & \text{if } \sum_{c=1}^{3} \hat{I}(i,j,c,k) \leq \theta \\ 255 & \text{if } \sum_{c=1}^{3} \hat{I}(i,j,c,k) > \theta. \end{cases} \end{aligned} \tag{6.12}$$

6.5.3 Colorchannel red

Since the images are RGB-format, it may be useful to apply the threshold algorithm only on the red-channel. This lowers the number of operations by $2MN$, since the summation for the pixelvalue can be skipped. Besides that it erases some noise and high lights the barrier better since this is colored red-white-red-etc. Now redefine (6.12) by

$$\begin{aligned} \hat{\hat{I}}(i,j,1,k) &:= X(i,j,1,k) - \check{B}(i,j,1,k) \\ \hat{\hat{I}}_\theta(i,j,k) &:= \begin{cases} 0 & \text{if } \hat{\hat{I}}(i,j,1,k) \leq \theta \\ 255 & \text{if } \hat{\hat{I}}(i,j,1,k) > \theta. \end{cases} \end{aligned} \tag{6.13}$$

Apply (6.13) to X_{200} of sample video 10 which results are shown in Figure 6.8. As can be seen the least noise is in Figure 6.8c. As a matter of fact the chosen θ in Figure 6.8a and 6.8b has to be multiplied by a factor three, since there are three color-channels. This adaption is shown in Figure 6.8d and 6.8e. As can be seen there is no great difference between Figure 6.8c and 6.8e. But to create Figure 6.8c fewer operations are required.

Unfortunately there is still some noise in Figure 6.8 which disturbs the measuring of the right angle.

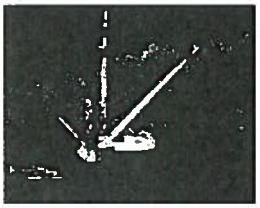
(a) B_{200}

(b) X_{200}

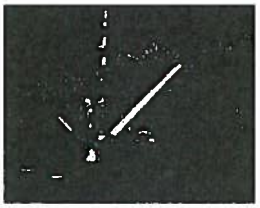
(c) $|X_{200} - B_{200}|$

(d) $X_{200} - B_{200}$

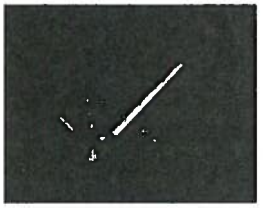
Figure 6.7: Normal and absolute difference applied to sample video 10



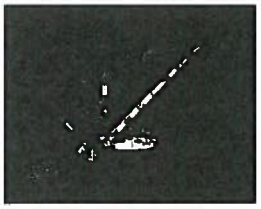
(a) $I_{100}(i, j, 200)$



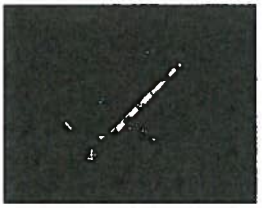
(b) $\tilde{I}_{100}(i, j, 200)$



(c) $\grave{\tilde{I}}_{100}(i, j, 200)$



(d) $I_{300}(i, j, 200)$



(e) $\tilde{I}_{300}(i, j, 200)$

Figure 6.8: Threshold applied to X_{200} from sample video 10

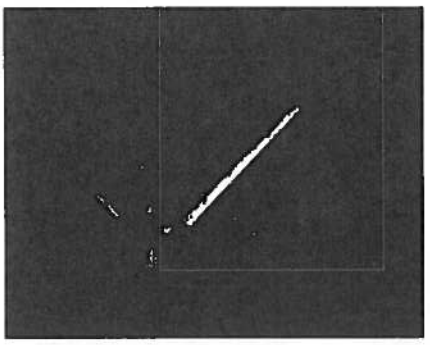


Figure 6.9: $\grave{\tilde{I}}_{100}(i, j, 200)$ of sample video 10 averaged over position

6.5.4 Averaging over position

Section 3.6 already discussed a technique for averaging the image over position to supress noise. As shown in that section in practice it was not useful for the detection of objects. Since the results were not that good in accordance to the number of operations it requires. But it may be useful to use it at the barrier detection. Figure 6.8c already showed that the barrier is a straight line, but still some noise is around it. If the measuring area is averaged over its position according to (3.17) only the clustered points remain exactly white. By skipping all other values, only the barrier remains visible. Figure 6.9 shows the result of this technique. The red line shows the measuring area.

6.5.5 Averaging over time

As was treated in Section 3.5, it is useful to average the signal for traffic detection over time to smooth it. This technique can also be used to smooth α. Then the course of the angle contains less noise.

6.5.6 Adapting measuring area

A problem with the techniques described in Subsections 6.5.1 up to 6.5.4 may occur when the barrier is up, at 90°. If a (red) vehicle is driving through the measuring loop of the barrier the vehicle may be noticed as a movement of the barrier which causes wrong measurements. To prevent this the measuring loop can be adapted to the position of the barrier. If the angle, α, and the length, δ, of the barrier, are known, the height and width of the measuring loop are easily calculated according to;

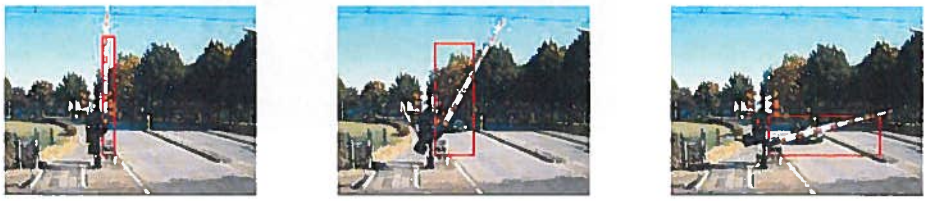
$$\begin{aligned} \text{height} &:= \delta\cos(\alpha) \\ \text{width} &:= \delta\sin(\alpha). \end{aligned} \tag{6.14}$$

Figure 6.10 shows three frames with the adapted measuring loop.

But this yields the same problem as discussed in Section 6.1. If the barrier falls down a lot (for instance by a technical disurbance) this is not noticed since the barrier is mostly outside the area. A small part is still visible but this might be not enough to detect the barrier correctly.

6.5.7 Contour filter

For future work it is advisable to research the possibilities for a contour filter. Since the barrier is usually a straight line, this can be detected easily by the contour filter. The barrier is generally the only straight line visible. MATLAB contains a function to create a contour plot of a image. When this function is applied to $X_{200} - B_{200}$ from sample video 10, the result is shown in Figure 6.11.



(a) X_1 (b) X_{150} (c) X_{300}

Figure 6.10: Adapting measuring loop for sample video 10

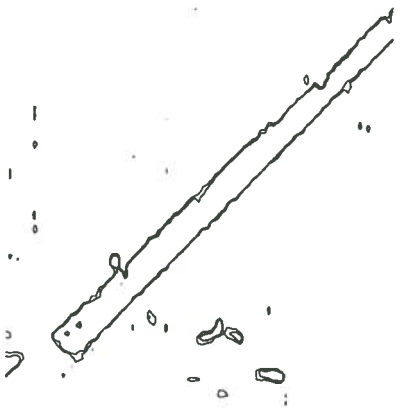


Figure 6.11: Contourplot of $X_{200} - B_{200}$

6.6 Comparing and testing the algorithms

Section 6.5 discussed some techniques for finding the coordinates of a barrier. With the found coordinates it is possible to calculate α according to the techniques described in Sections 6.1 up to 6.4. This section compares and tests these techniques.

The first method is based on the detection of masks, see Section 6.1. An advantage of this technique is the small influence from the noise at the background of the measuring loop. Since only a small area around the last α is checked, noise that is not in this area is skipped. Besides that there is only one α which corresponds best to the coordinates. This is shown in Figure 6.12. All six lines cover the mask, the two middle lines cover the mask best. But there is exactly one mask which covers the barrier best.

Figure 6.12: Detection based on lines vs detection based on lines

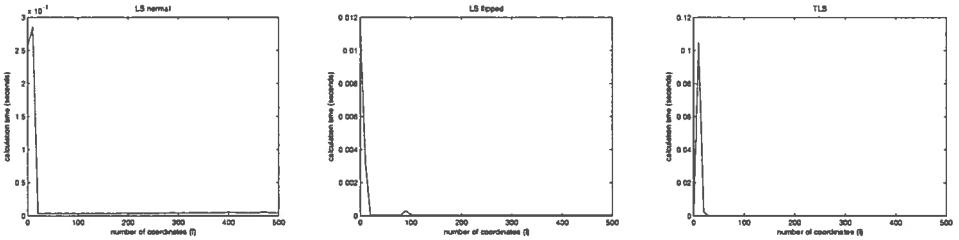
A big disadvantage of this technique is the calculation of which mask fits best. For every mask a check has to be done for how many coordinates lie in that specified mask. So for every possible mask around the last α all the coordinates have to be checked if they lie in the specified mask. The number of operation required to check all masks with a dipersion of h is then $O(hl)$, where l is the number of coordinates.

Example 6.6.1. Suppose the dispersion of the masks around the last α is $+5°$ and $-5°$. Then for every frame check if the pair (x, y) is within the mask which belongs to a specified angle. This has to be done for eleven different masks. The mask which covers the most pairs (x, y) is the new α.

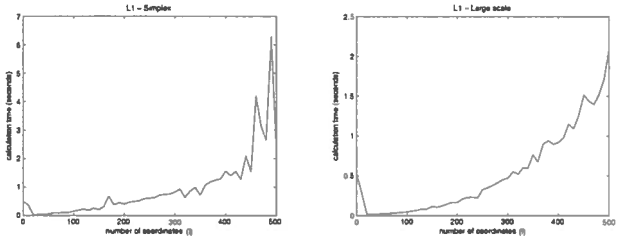
A faster way may be the technique of the least squares method. This method gathers all coordinates in a matrix at each frame to determine α. The total calculation takes about $O(l)$ operations [14]. Figure 6.13a and Figure 6.13b show the calculation time plotted against l. Zooming in shows that the calculation time is linear in l. As described in Section 6.2, there is a problem with the rotation factor. Switching the axis is useful but then arises the problem at which point to switch.

A solution to this problem was found in the total least squares method, since this method minimizes the perpendicular offset to the barrier it is independent for rotation of the barrier. The number of operations needed for his method is $O(l)$ [14]. To see the development of the calculation time it is plotted against l and shown in Figure 6.13c

The third method which is discussed was the L_1-approximation. An advantage of this method is the fact that is less sensible for noise. If coordinates lie far away from the line, the L_1-approximation is able to ignore these points. The big disadvantage of this method is the number of required operations, this is approximately $O(l^4)$ when the solution is found using the simplex algorithm. The calculation time is plotted against l and shown in Figure 6.13d. Instead of the simplex algorithm MATLAB also is able to a 'Large-scale' algorithm. This algorithm is based on LIPSOL (Linear-programming Interior Point SOLvers, see [20]) and can also be used to solve the L_1-approximation. This also takes about $O(l^4)$ operations, see Figure 6.13e. The order of operations is the same but the large scale algorithm takes fewer operations when there is a large amount of data points. The more data points, the faster the large scale algorithm is in proportion to the calculation time of the simplex algorithm.



(a) Least squares (normal) (b) Least squares (flipped) (c) Total least squares



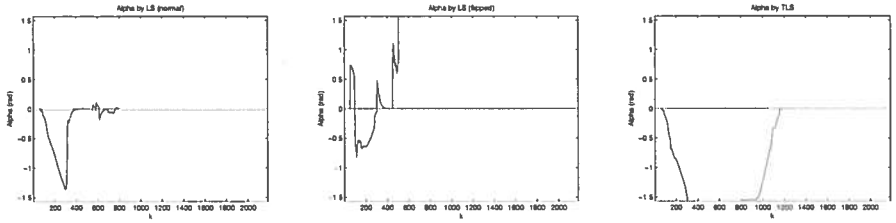
(d) L_1 (Simplex) (e) L_1 (Large scale)

Figure 6.13: Calculation time of the algorithms

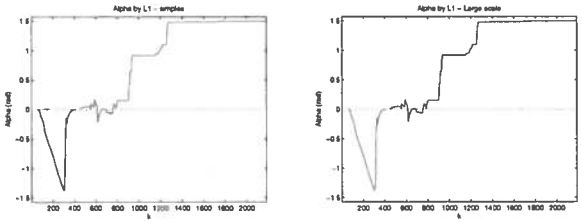
All results in Figure 6.13 show a peak at the first values of l. This can be explained by the efficiency of MATLAB. Table 6.1 shows the operation times of the algorithms applied to sample video 4. The results for the angle are shown in Figure 6.14.

Table 6.1: Operation times for tested algorithm

	Time in seconds				
index	Least Squares (normal)	Least Squares (flipped)	Total Least Squares	L_1 (Simplex)	L_1 (Large scale)
1	20.2050	21.3739	22.7826	214.0752	132.1119
2	21.0833	21.1156	22.9899	212.9121	126.2807
3	20.6102	21.6637	21.3123	212.5387	127.0201
4	21.2032	21.2573	20.8572	211.7042	125.9847
5	20.4128	21.1687	20.4009	210.9406	125.7218
6	20.3740	21.3658	20.5204	217.3878	126.3036
7	20.3223	21.1822	21.0053	213.5475	128.4090
8	20.2895	21.5292	21.1978	216.9851	126.8530
9	20.0501	21.5477	20.7715	213.7597	127.1518
10	20.0494	21.3119	208402	213.0759	123.4988
average	20.4600	21.3516	21.2678	213.6925	126.9335



(a) Least squares (normal) (b) Least squares (flipped) (c) Total least squares



(d) L_1 (Simplex) (e) L_1 (Large scale)

Figure 6.14: Results for the several algorithms applied to sample video 4

The reason that the L_1-approximation require more operations in comparison with the least square methods can be explained by that the solution technique is based on iterations. The larger l is, the more iterations have to be done to perform a good solution. This can be skirted by setting a maximum number of iterations, but this influences the error of the solution. More operations usually means a smaller error and so a more accurate result.

Table 6.1 and Figure 6.14 conclude that the best algorithm is the total least square method. The next question is if this algorithm also gives a good resultfor other video's. Figure 6.15 shows the results.

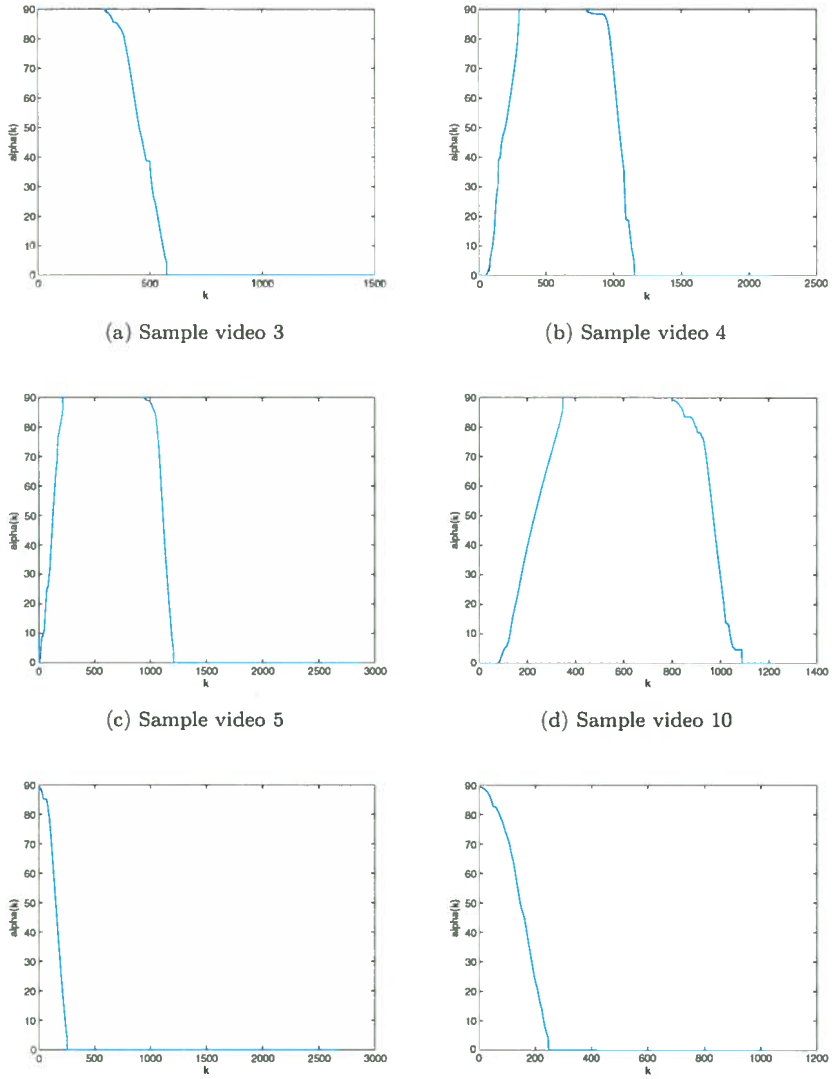
These results show a good development of the angle. Visual scoring is done to compare the results with the real situation. For the tested video's the value at which the barrier reached 0° or 90° corresponds with the frame at which the barrier is horizontal or vertical. Figure 6.15d showed that at $k = 200$ the angle is approximately 40°. Visual scoring shows that the real angle is approximately 44°, see Figure 6.16. This figure also displays the barrier at 0° and 90°.

Conclusion

From Section 6.6 the conclusion can be drawn that the total least square method, Algorithm 6.3 in Section 6.3, is the best method tested for the detection of the angle from the barrier. This is not the quickest method, which is the least square method (Algorithm 6.2 in Section 6.2). Unfortunately this method does not work quite well since this method is dependent from the angle of the barrier, as shown in Section 6.2. The third method tested is based on linear programming, Algorithm 6.5, described in Section 6.4. This method takes to long in comparison with the (total) least square methods. So the best method is the total least square method.

The results in Figure 6.15 are acquired with the method of total least squares. A conclusion from this results can be drawn that this method is working good and is independent for rotation of the barrier. For the tested video's all the barriers shows a good path from its original position to their new position. Not all sample video's contain two movements (UP-DOWN and DOWN-UP), but also the video's which contain one movement (DOWN-UP or UP-DOWN) show a good path.

As described in Section 6.6 a disadvantage for the detection of lines instead of detection based on masks may be that only one mask may cover the barrier while several lines may cover it. So at frame k it might be that the angle α is not exactly correct. This is in principle no problem since the most interesting part is the development of α.



(a) Sample video 3

(b) Sample video 4

(c) Sample video 5

(d) Sample video 10

(e) Sample video 13

(f) Sample video 15

Figure 6.15: Results for the barrier-detection

(a) X_{75}

(b) X_{200}

(c) X_{350}

Figure 6.16: Angle of the barrier from sample video 10

So instead of using the theory of masks it is better to use the technique based on the total least square method. This method works faster in comparison with the currently used method based on masks [9]. Besides that it also deploys a good result.

To find the coordinates the several methods described in Section 6.5 can be used. From these methods the averaging over position may require the most operations. But this is very useful since the coordinates on the barrier are clustered around the line. Noise points are filtered out on this way. To find the coordinates it might also be possible to use the contour filter.

Since the camera is not recording on the front side of the railroad crossing but from a angle, the path of the barrier is a bit distorted. If the angle on which the camera stands is known, it might be possible to create a better calculation for α.

Chapter 7

Analysis of the audio

While previous chapters discussed detection based on the images taken from the camera, this chapter analyses the audio track of the camera. This audio is needed to check the status of the bell and the number of bells ringing. To analyze the audio it is splitted from the sample video files. Figure 7.1 shows an overview of the application for the analysis of the audio.

7.1 Frequencies

Figure 7.2 shows an original data file for the audio taken from sample video 4. As can be seen there is no clear timestamp at which the bells of the railroad crossing turn on and turn off. This is because the data also contains sounds from the enviroment. By analyzing the frequencies of the bell it is possible to filter the noise such that the audio of the bell remains. The Fourier transformation can be used to detect the frequencies of the bells and also to filter the noise.

7.2 Fourier transformation

To determine which frequencies the bell uses, the **D**iscrete **F**ourier **T**ransformation (DFT) which maps N numbers, $s_0, \ldots, s_{N-1}$, onto N numbers $S_0, \ldots, S_{N-1}$ defined as,

$$S_p = \sum_{h=0}^{N-1} s_h e^{-\frac{2\pi i}{N} ph}, \qquad p = 0, \ldots, N-1, \tag{7.1}$$

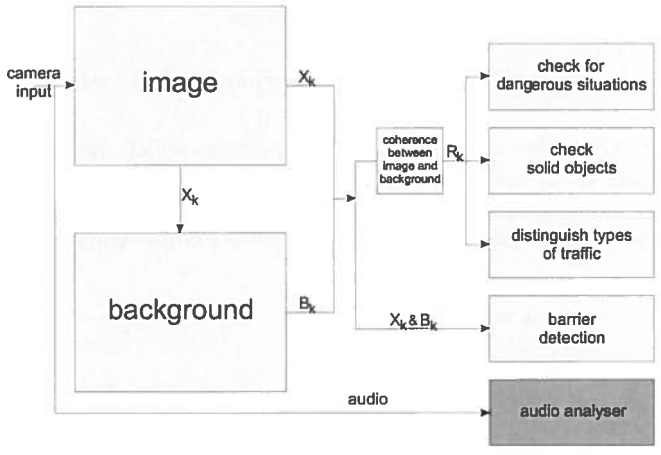


Figure 7.1: Overview of the application (audio analyser)

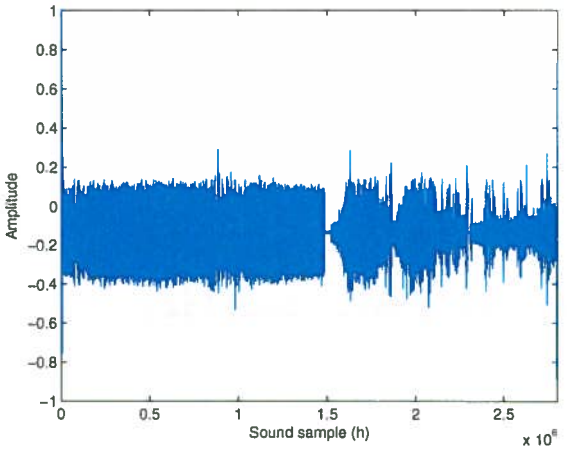


Figure 7.2: Original audio file (sample audio 1) taken from sample video 4

is used. Here s_h is the original data. For the calculation of the DFT, the **F**ast **F**ourier **T**ransformation (FFT) is used. This is an efficient algorithm for the DFT. The most common FFT algorithm is the Cooley-Tukey algorithm [12]. Figure 7.3 shows the absolute DFT of sample audio 1, where as the original audio file is shown in Figure 7.2. For elemination of the zero frequency, the original data is first centered around zero.

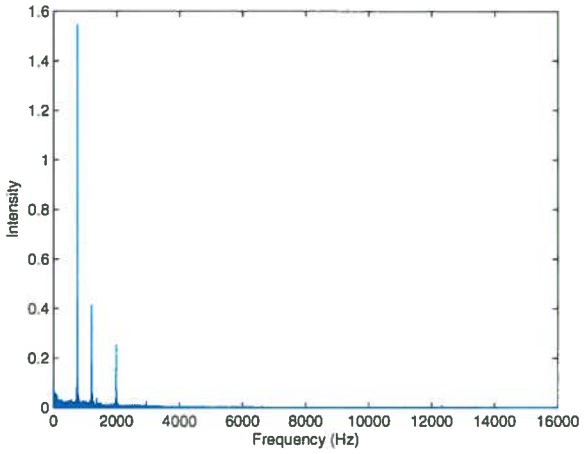


Figure 7.3: Frequency spectrum for audio sample 1

As can be seen there are three important frequencies which dominate the audio of the bell. Visual scoring shows that these frequencies lie at approximately 750 Hz, 1200 Hz and 2000 Hz, see Figure 7.4. By eliminating these frequencies (setting all other values to zero), only the audio of the bell remains. To this (new) data, $\breve{S}_p$, the inverse Fourier transformation,

$$s_h = \frac{1}{N} \sum_{p=0}^{N-1} S_p e^{\frac{2\pi i}{N} ph}, \qquad h = 0, \ldots, N-1, \tag{7.2}$$

is applied, where S_p is the Fourier transformed data from (7.1). Figure 7.5 shows $\breve{s}_h$, the inverse Fourier transformation for $\breve{S}_p$.

This figure shows the moments when the bell is 'on' and when the bell is 'off'. If the bell is 'on', there is a higher amplitude in comparison with the moments where the bell is 'off'. The

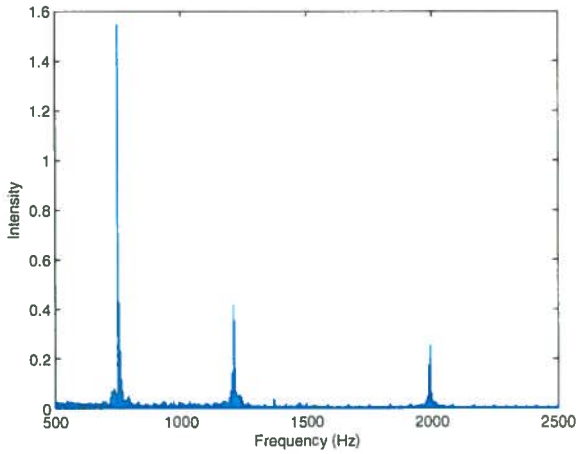


Figure 7.4: Frequency spectrum for audio sample 1 (zoomed in)

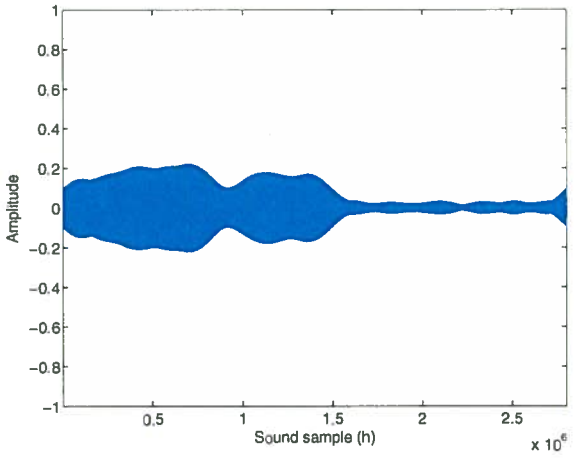


Figure 7.5: $\check{s}_h$ for audio sample 1

reason that the amplitude of the signal is lower for some time can be explained by the passing train. The audio recording device causes that the audio of the passing train drowns out the audio of the bell. So the intensity of the bell frequencies seams less, which is not the case.

To create Figures 7.3 and 7.5 the Fourier transformation is applied to all data points. In reality the data is received from an audiostream and so it is impossible to apply the Fourier transformation to all datapoints. So there is a need for real time checking of the audio. But single data points only contain one frequency (zero). Better is to apply the Fourier transformation to a set of data points.

Since the FFT algorithm requires $O(N \log N)$ operations the set must not be too great, but a small set leads to more frequent Fourier transformations. A compromise may be for instance by taking N equal to the sampling frequency for the audio f_{audio}, which is usually 32000 Hz. Every period a new set of data points is gathered and the Fourier transformation is applied to this set.

The amplitude of $\check{s}_h$ depends on the value of $\check{S}_p$. So instead of taking the inverse transformation for $\check{S}_p$, take a look at $\check{S}_p$ itself. If the bell is 'off', the frequencies have a much lower value compared to when the bell is 'on'. This is faster since there is no inverse transformation needed and by only checking a specified area around the frequencies makes it possible to see if the bell is 'on'.

The frequency areas are defined as

$$\begin{aligned} f_{\text{audio},1} &:= 700 : 800 \\ f_{\text{audio},2} &:= 1150 : 1250 \\ f_{\text{audio},3} &:= 1950 : 2050 \end{aligned} \tag{7.3}$$

Define the status of the bell as 'on' when the maximum for two out of the three frequency areas are above the limit, L_{bell}, and 'off' when one or no maxima are above the limit. So by applying the FFT to every set of 32000 samples and determing the maxima of the three areas it is feasible to see if the bell is 'on' or 'off', see Algorithm 7.1. Even better is to apply the FFT to a set of 32768 ($= 2^{10}$) samples, this is more efficient for the FFT.

Algorithm 7.1: Status of the bell [see Section 7.2]

Input: Original audio data (s_h)
Output: Status for the bell
$S_p \leftarrow$ `fft` (s_h)
if $\max S_p \geq L_{bell}$ *for* $p \in f_{audio,1}$ *and* $\max S_p \geq L_{bell}$ *for* $p \in f_{audio,2}$ **then**
| Status = 'on'
else if $\max S_p \geq L_{bell}$ *for* $p \in f_{audio,1}$ *and* $\max S_p \geq L_{bell}$ *for* $p \in f_{audio,3}$ **then**
| Status = 'on'
else if $\max S_p \geq L_{bell}$ *for* $p \in f_{audio,1}$ *and* $\max S_p \geq L_{bell}$ *for* $p \in f_{audio,2}$ **then**
| Status = 'on'
else
| Status = 'off'
end

The original audio files, the Fourier transformation for this data and the status of several audio files are shown in Appendix A.6.

When a railroad crossing contains four alarm bells, all four are 'on' when the barriers are going down. If the barriers are down, only two bells remain on. By the amplitude of the frequencies it should be able to detect how many bells are 'on'. Unfortunately as can be seen in Figure 7.5 the amplitude almost remains the same in the development of the time. The only decrease is when the train is passing. Figure 7.6 shows the amplitude of the three frequency areas.

This figure shows that the amplitude of the three frequencies almost remains the same. The explanation for this may be found in the position of the camera. If the camera is assembled near the bell which is always 'on', the other bells are not noticed. A better approach is to assemble four different audio recording devices where every device measures its own position on the railroad crossing. In this manner it is easier to detect which bells are 'on'.

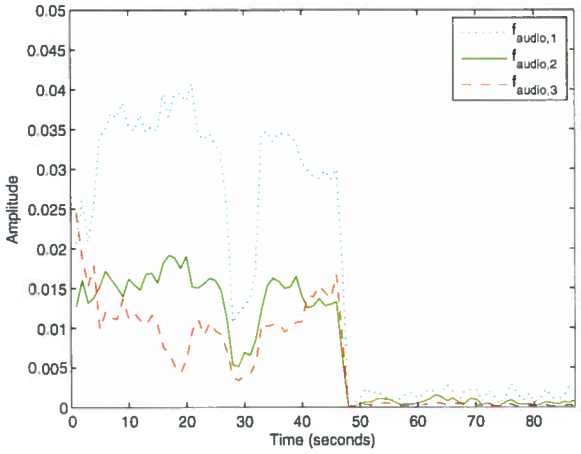


Figure 7.6: Development of the three frequencies for audio sample 1

As described previously a disadvantage for this technique is when a train is passing. This makes that the audio of the bell is drown out by the audio of the passing train. This causes that the amplitude is lower and so the satus of the bell is denoted as 'off'. This disadvantage can be solved by the video part of the application. If a train is passing and the status of the bell is denoted 'off' it can be set to 'on'.

7.3 Bandpass filter

Since the frequency areas are known another approach is to build a bandpassfilter [7]. By passing the audio data through such a filter, only the frequencies remain which lie in the specified band. A sixth order butterworth bandpass filter is created for the frequency areas as defined in (7.3). The magnitude response of these filters is shown in Figure 7.7.

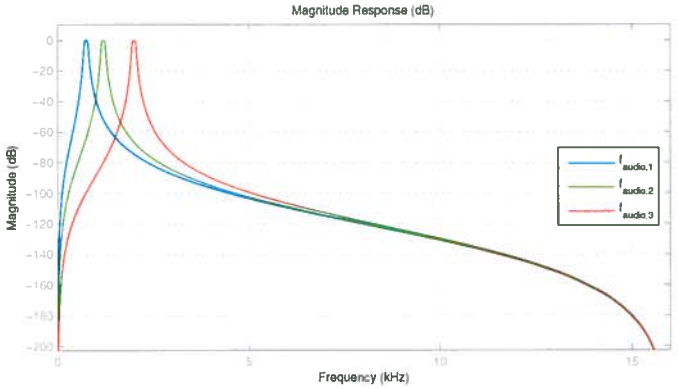


Figure 7.7: Magnitude response

The corresponding (discrete) transfer function ([7]) to this filters are:

$$H_1(z) = \frac{9.279 \cdot 10^{-7} - 2.784 \cdot 10^{-6}z^{-2} + 2.784 \cdot 10^{-6}z^{-4} + 9.279 \cdot 10^{-7}z^{-6}}{1 - 5.896z^{-1} + 14.55z^{-2} - 19.23z^{-3} + 14.36z^{-4} - 5.744z^{-5} + 0.9615z^{-6}} \quad (7.4)$$

$$H_2(z) = \frac{9.279 \cdot 10^{-7} - 2.784 \cdot 10^{-6}z^{-2} + 2.784 \cdot 10^{-6}z^{-4} + 9.279 \cdot 10^{-7}z^{-6}}{1 - 5.796z^{-1} + 14.16z^{-2} - 18.65z^{-3} + 13.98z^{-4} - 5.647z^{-5} + 0.9615z^{-6}} \quad (7.5)$$

$$H_3(z) = \frac{9.279 \cdot 10^{-7} - 2.784 \cdot 10^{-6}z^{-2} + 2.784 \cdot 10^{-6}z^{-4} + 9.279 \cdot 10^{-7}z^{-6}}{1 - 5.507z^{-1} + 13.07z^{-2} - 17.06z^{-3} + 12.9z^{-4} - 5.365z^{-5} + 0.9615z^{-6}} \quad (7.6)$$

As can be seen, the numerator in (7.4), (7.5) and (7.6) are all the same. This is because the coefficients of the numerator depend on the input data, which is the same for all three transfer functions. To see if these functions are stable, the position of the poles (roots of the denominator in the transferfunction) is needed. This is shown in Table 7.1.

	$f_{\text{audio},1}$	$f_{\text{audio},2}$	$f_{\text{audio},3}$
poles	$0.9828 + 0.1542i$	$0.9655 + 0.2404i$	$0.9160 + 0.3885i$
	$0.9828 - 0.1542i$	$0.9655 - 0.2404i$	$0.9160 - 0.3885i$
	$0.9858 + 0.1375i$	$0.9697 + 0.2240i$	$0.9227 + 0.3730i$
	$0.9858 - 0.1375i$	$0.9697 - 0.2240i$	$0.9227 - 0.3730i$
	$0.9796 + 0.1447i$	$0.9630 + 0.2308i$	$0.9149 + 0.3787i$
	$0.9796 - 0.1447i$	$0.9630 - 0.2308i$	$0.9149 - 0.3787i$

Table 7.1: Poles of the transferfunctions

For the transferfunction to be stable, these poles have to lie within the unit circle. Figure 7.8 shows that this is the case.

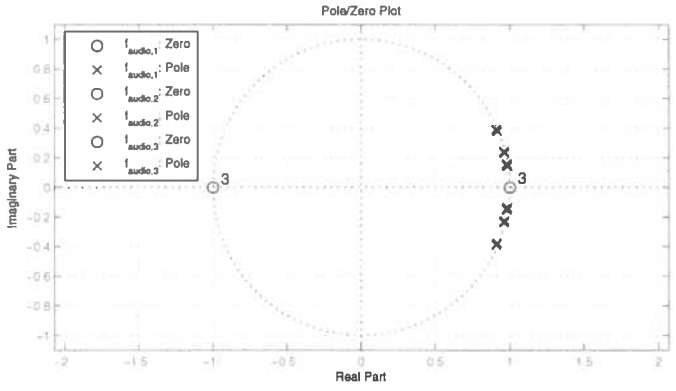


Figure 7.8: Pole zero plot for the transferfunctions

These transfer functions are applied to the original data as shown in Figure 7.2. Which yields the output as shown in Figure 7.9.

A disadvantage of this technique is the number of required operations to apply the bandpass-filter. The fast Fourier algorithm is almost two times faster in comparison to the bandpassfilter. Table 7.2 gives an overview of mean computation times. This is because the bandpass filter has to be applied three times, for each frequency area. Conclusion from this table is that the Fourier transformation (see Section 7.2) is the fastest algorithm tested to analyse the audio.

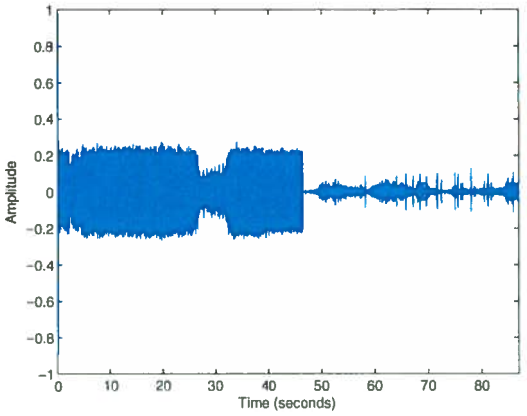


Figure 7.9: Sum of the three bandpass filters applied to audio sample 1

Table 7.2: Comparison of the operation times in seconds

	Time in seconds		
Index	Fourier transformation (see Section 7.2)	Bandpassfilter (see Section 7.3)	Proportion
1	0.9343	2.5787	2.7602
2	0.9338	2.5753	2.7580
3	0.9399	2.5801	2.7451
4	0.9327	2.5902	2.7771
5	0.9343	2.5729	2.7537
6	0.9346	2.5890	2.7702
7	0.9328	2.5845	2.7705
8	0.9350	2.5787	2.7580
9	0.9325	2.5748	2.7610
10	0.9330	2.5818	2.7671
average	0.9343	2.5806	2.7621

Conclusion

The status of the bell can be detected by looking in the frequency domain. Three frequency areas define the audio of the bell. If two out of the three frequencies are above the limit, the bell is defined as 'on', otherwise it is defined as 'off'. To transform the signal into the frequency domain, the FFT-algorithm is used. In the current situation it is not yet possible to detect how many bells are ringing, to do this it is better to have four measuring points one for each bell.

Another approach which is investigated is the bandpassfilter. This method requires more operations in comparison to the Fourier transformation and is therefore skipped.

Future research may be done by eliminating the audio of the train and researching this.

Chapter 8

Conclusion

During the research several properties of the application has been tested. The currently used algorithms are compared to new developed algorithms. As discussed in Chapter 2 a background is needed to observe motion. This background needs to be adapted to new situations. Otherwise a change in luminious intensity causes false positive detection. Several methods for the adaptation have been tested and Algorithm 2.2 on page 22 (exponential adaptation) is the most efficient algorithm tested.

This algorithm is approximately two times faster than Algorithm 2.1 which is used in the current application [9]. The rate at which the background is adapted (β) is chosen according to the settling time, see Threorem 2.3.2 on page 25.

The (adapted) background is used for the coherence between the current frame and its background. With this coherence it is possible to detect motion. Several methods have been discussed for this coherence. The biggest problem with this methods is the changing luminious intensity. If detection is based on threshold (θ) with the absolute difference (Algorithm 3.1 on page 30) a brigther image yields a larger difference, since the background remains the same. This larger difference results in detection of motion. One way to solve this problem is to adapt θ or β. A larger θ results in less accuracy for detection, the same occurs with a larger β. This problem can be solved by temporarily adapting θ or β. A disadvantage of this technique is to find relation between the intensity and θ or β.

A better approach is to skip θ, by summing over colorchannel c and dividing this by the number of colorchannels the image is converted to grayscale, see Algorithm 3.2 on page 33. The intensity of the grayscale is a indicator for motion, R_k. Unfortunately this method is not independent to the luminous intensity. If the luminous intensity changes then the average over the colorchannels also changes. To compensate changing luminous intensity it is better to divide the pixelvalue by the average over the measuringloop (Algorithm 3.3 on page 34). If now the luminous intensity changes, the average pixelvalue changes such that the fraction of the pixelvalue with its average approximately remains the same. When a object is passing, the average changes less in comparison with the pixelvalue itself. Resulting in that the fraction rises and motion is detected. A disadvantage of this method is the required operations, this is particularly caused by the average.

The best approach tested appears to be detection by correlation, see Algorithm 3.4 on page 36. This algorithm determines the correlation between the current frame and its corresponding background. This yields a correlation factor (γ) with a value between zero and one. Besides that it is normalized between zero and one and independent on the luminous intensity, it is the fastest algorithm tested for detection. An even better result is acquired when the correlation is applied on each colorchannel seperate in stead of the sum of the three colorchannels.

The indicator for motion is used to detect dangerous situations on the railroad crossing. If the barriers are closed usually no motion is detected and so R_k lies around zero. If motion is detected while the barriers are closed this might be caused by a crossing object. Otherwise if the barriers are open, R_k fluctuates when a object is passing by. If this is not the case, so R_k approximately remains constant, this suggests that a object is stationary. The same technique can be used to detect vandalism on solid objects like the andreas cross and fenches.

As discussed usually the objects are moving on. In this scenario it is desirable to differentiate

between the types of traffic. This is done by researching the several properties of the objects. Every type of object has some properties. Since a car is usually wider than a cyclist, it covers more area of the measuring loop. So the correlation between the image and its background is less, resulting that R_k is larger. Besides that R_k rises faster because of the width of a car in comparison to a cyclist. Also the length and speed of a car are larger in comparison to a cyclist. These properties can be combined into Algorithm 5.5 on page 67. This algorithm yields in approximately 88% a corresponding result.

Besides the traffic another important check is the path of the barrier. This can be done by placing masks over the image to detect the angle. A disadvantage of this method is the number of required operations. A faster approach is based on line detection. Several methods for line detection have been tested namely the least square method, total least square method (perpendicular offset) and L_1-approximation. These methods showed that the total least squares was the best method tested in both ways. In develepmont as well as in operation time Algorithm 6.3 (page 76) showed the best result. The L_1-approximation, described in Algorithm 6.5 on page 78 (with solving techniques simplex and large scale), took much longer and did not produce a good result in comparison with the total least square method. The least square method (Algorithm 6.2 on page 73) was a little bit faster but is dependent on the rotation of the barrier.

Not only the video components of the railroad crossing were discussed. Also the audio of the warning bell is researched. By the Fourier transformation three frequencies are eliminated for the bell. With these frequencies a realtime check is made based on the Fourier transformations and bandpassfiltering. These methods have been compared for the operation times. The Fourier transformation is the fastest method to see the status of the bell.

As a total conclusion the new developed algorithms are faster than the currently used algorithms.

8.1 Future work

At the new developed algorithms a RGB colorspace is used. To prevent noise it might be an idea to take a look at grayscale images or YCbCr colorspace. In the YCbCr colorspace, the Y-channel denotes the luminious intensity of an image. Another idea is check the possibilities of an infrared camera. This camera might be a solution for detection of motion. Since it might have less influence on weather situation (like rain or snow). Which is not taken into account in this research.

For the tests of the algorithms, described in the thesis, several variables are defined for instance L_{mot} and L_{car}. These variables are chosen at the same value for each video. Better is to calibrate these values to the luminious intensity, since not every video has the same luminious intensity. So for instance L_{car} does not need to be the same every time. By calibrating these variables a better accuracy is obtained, resulting in better results. A third approach to improve the detection of traffic can be done by positioning the loops better. For instance placing one loop slantwise instead of five measuringloops gradually. Another approach is to adapt the luminious intensity of a image such that bright images become darker and the other way around. If in a future version objects approaching from two sides needs to be detected, then the luminious intensity or color of an object can also be used to distinguish them.

In this situation it might be even better to create a situation with two cameras. In this way two checks for the objects can be done. Besides that it might be possible to make a 3D-situation of the railroad crossing. Such that a user is even more able to see what is going on at the railroad crossing and it makes it easier to see vandalism on the solid objects.

For the detection of the barrier it might be useful to apply a contourfilter. Since the barrier consists of a straight line, this is easily detected by its contours. Increasing the number of detection points leads to more operations to calculate the angle but also leads to more accurate measurements. Since the images are not taken in front of the barrier it might be useful to take the angle of the camera into account. This also leads to a more accurate angle of the barriers.

With the algorithm for the audio it is not able to detect the number of bells which are 'on'. To do this the number of measuringpoints might be extended. With more measuring points it is easier to detect the audio of the bell at the other corners of the railroad crossing. In this research only the audio of the camera is used.

Appendices

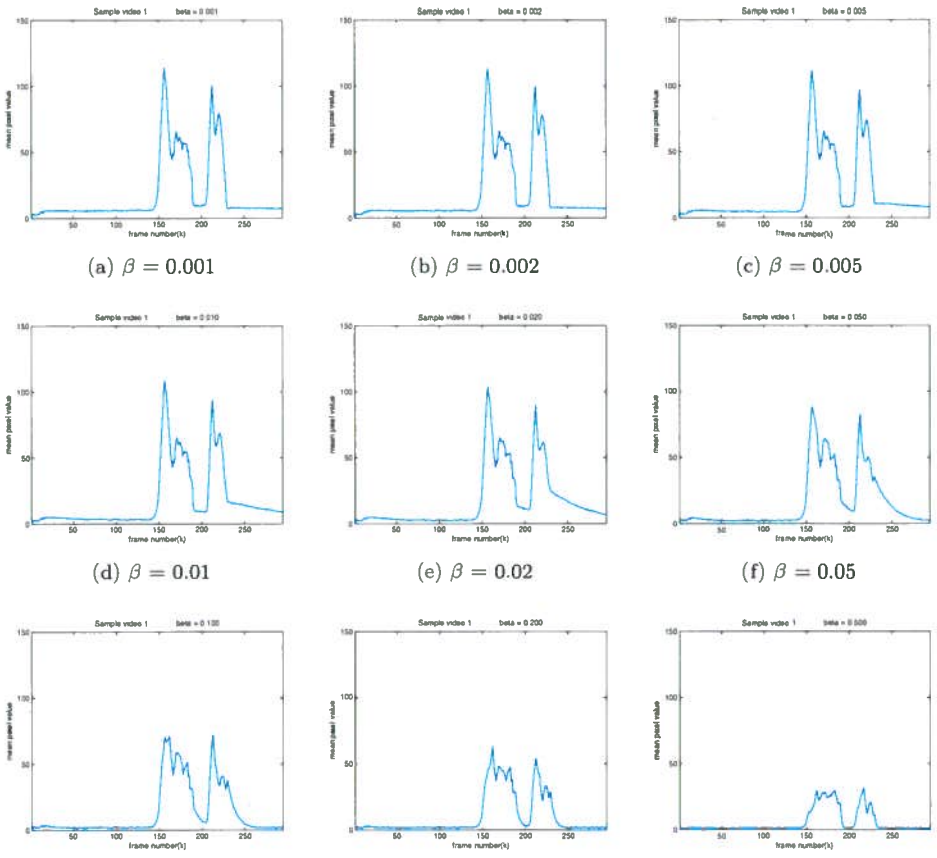
Appendix A

Figures

This chapter contains various figures.

A.1 Various values of β

As described in Section 2.3 the sample video 1 is tested with several values of β.



(a) $\beta = 0.001$ (b) $\beta = 0.002$ (c) $\beta = 0.005$

(d) $\beta = 0.01$ (e) $\beta = 0.02$ (f) $\beta = 0.05$

(g) $\beta = 0.1$ (h) $\beta = 0.2$ (i) $\beta = 0.5$

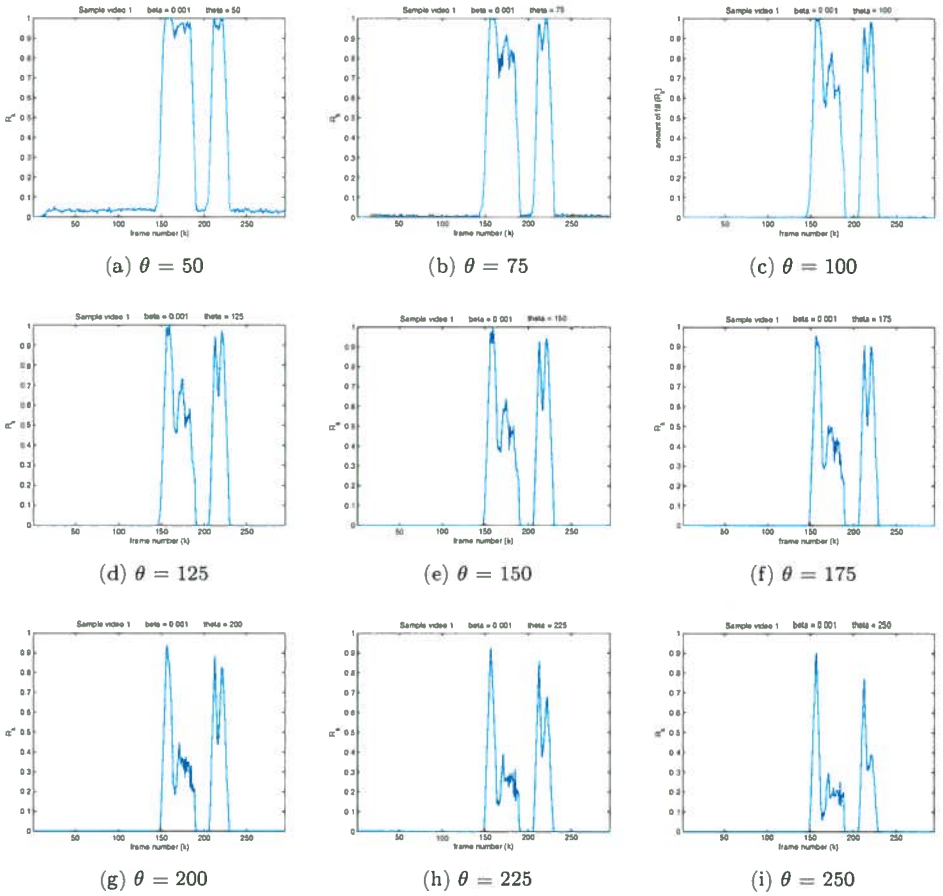
Figure A.1: A plot of mean pixel value with various values of β for the sample video 1

A.2 Various values of θ

This section shows sample video's 1 and 2 tested with various values of θ. In both video's there holds:

$$\beta = 0.001$$

A.2.1 Sample video 1



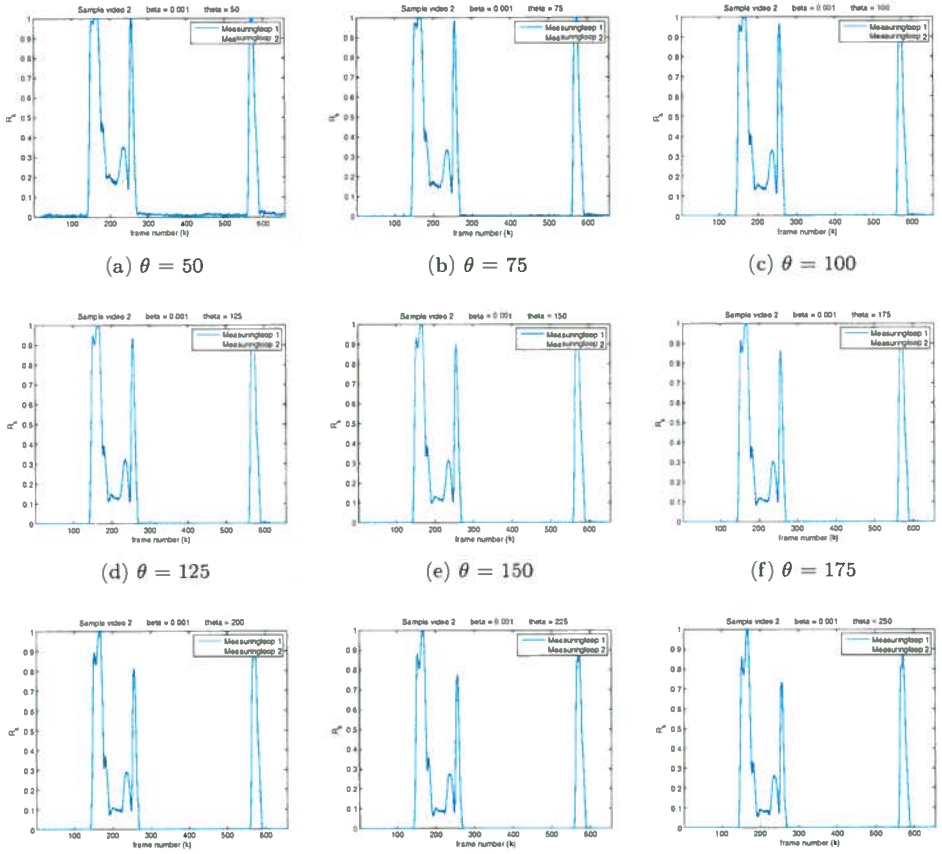
(a) $\theta = 50$ (b) $\theta = 75$ (c) $\theta = 100$

(d) $\theta = 125$ (e) $\theta = 150$ (f) $\theta = 175$

(g) $\theta = 200$ (h) $\theta = 225$ (i) $\theta = 250$

Figure A.2: A plot of R_k for various values of θ applied to sample video 1

A.2.2 Sample video 2



(a) $\theta = 50$ (b) $\theta = 75$ (c) $\theta = 100$

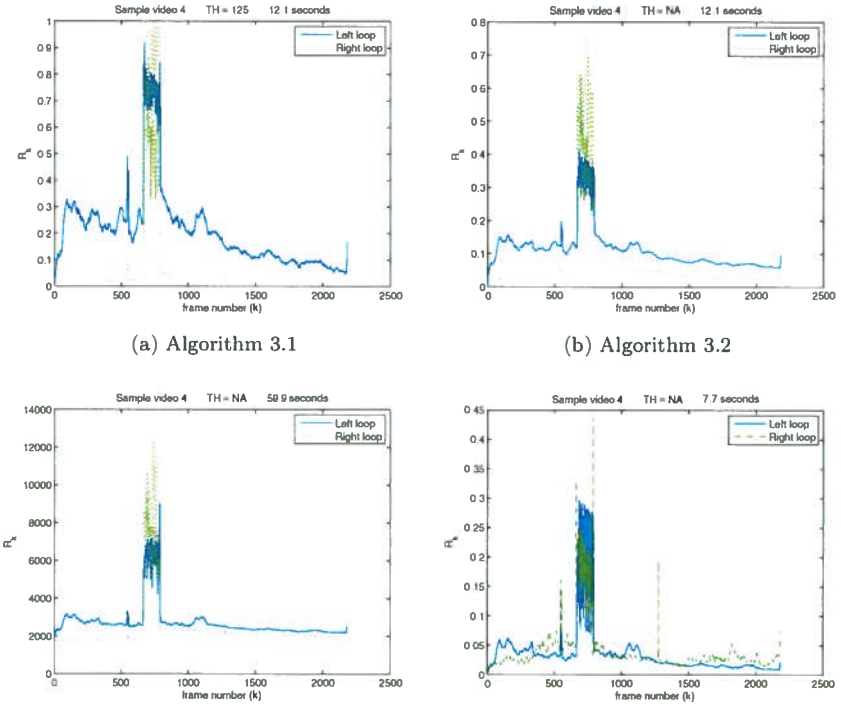
(d) $\theta = 125$ (e) $\theta = 150$ (f) $\theta = 175$

(g) $\theta = 200$ (h) $\theta = 225$ (i) $\theta = 250$

Figure A.3: A plot of R_k for various values of θ applied to sample video 2

A.3 Results for the train detection

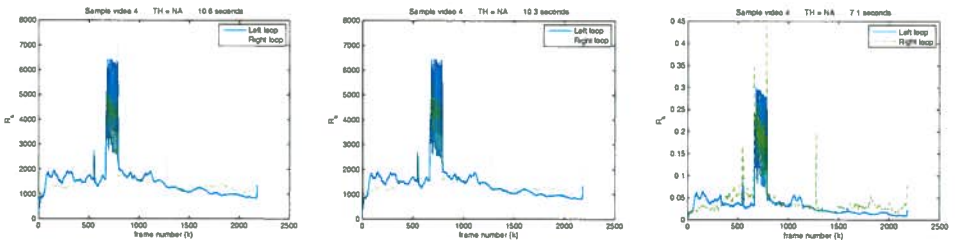
A.3.1 Sample video 4



(a) Algorithm 3.1 (b) Algorithm 3.2

(c) Algorithm 3.3 (d) Algorithm 3.4

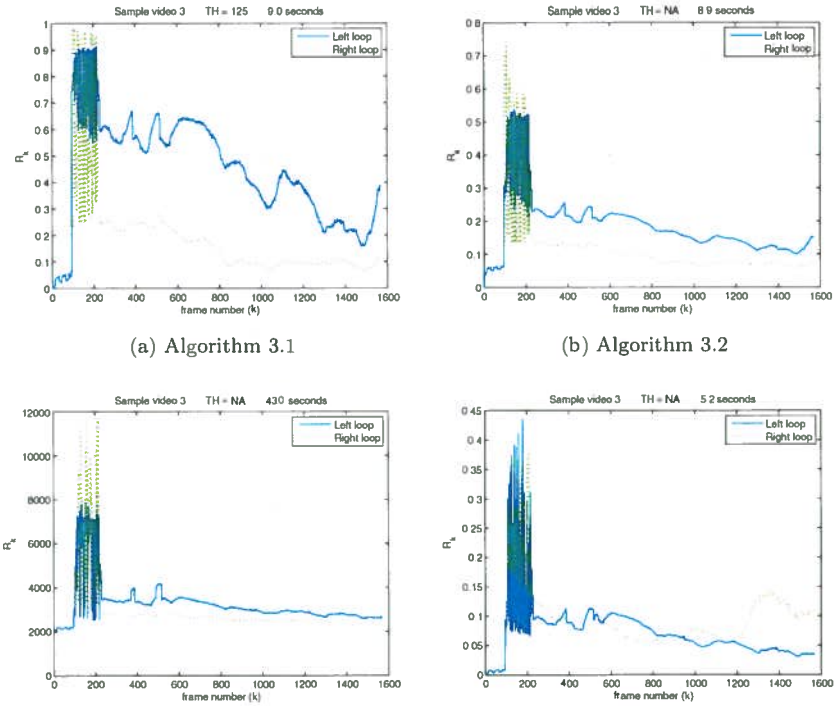
Figure A.4: Results (R_k) of the algorithms applied to sample video 4 for the train



(a) Average over loops (b) Average split into colors (c) Correlation split into colors

Figure A.5: Results (R_k) for the extended algorithms applied to sample video 4 for the train

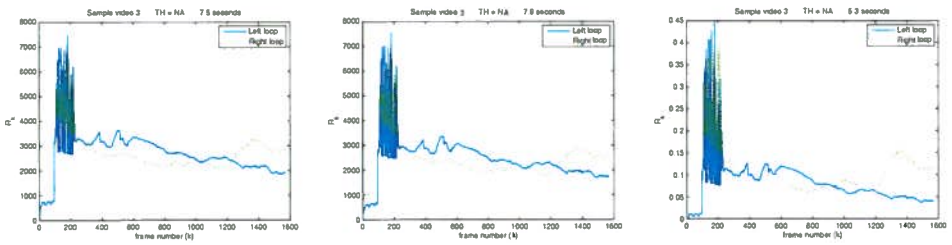
A.3.2 Sample video 3



(a) Algorithm 3.1 (b) Algorithm 3.2

(c) Algorithm 3.3 (d) Algorithm 3.4

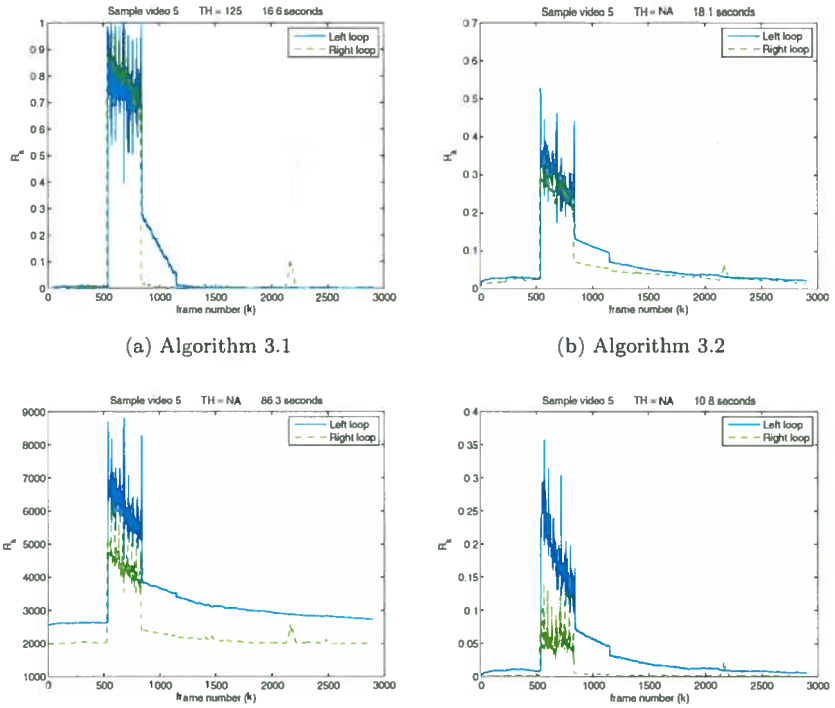
Figure A.6: Results (R_k) of the algorithms applied to sample video 3 for the train



(a) Average over loops (b) Average split into colors (c) Correlation split into colors

Figure A.7: Results (R_k) for the extended algorithms applied on sample video 3 for the train

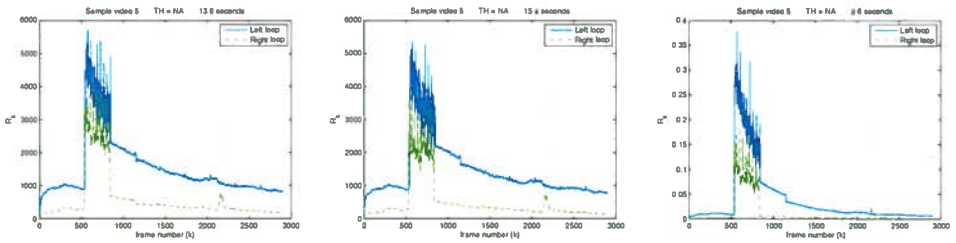
A.3.3 Sample video 5



(a) Algorithm 3.1 (b) Algorithm 3.2

(c) Algorithm 3.3 (d) Algorithm 3.4

Figure A.8: Results (R_k) of the algorithms applied to sample video 5 for the train

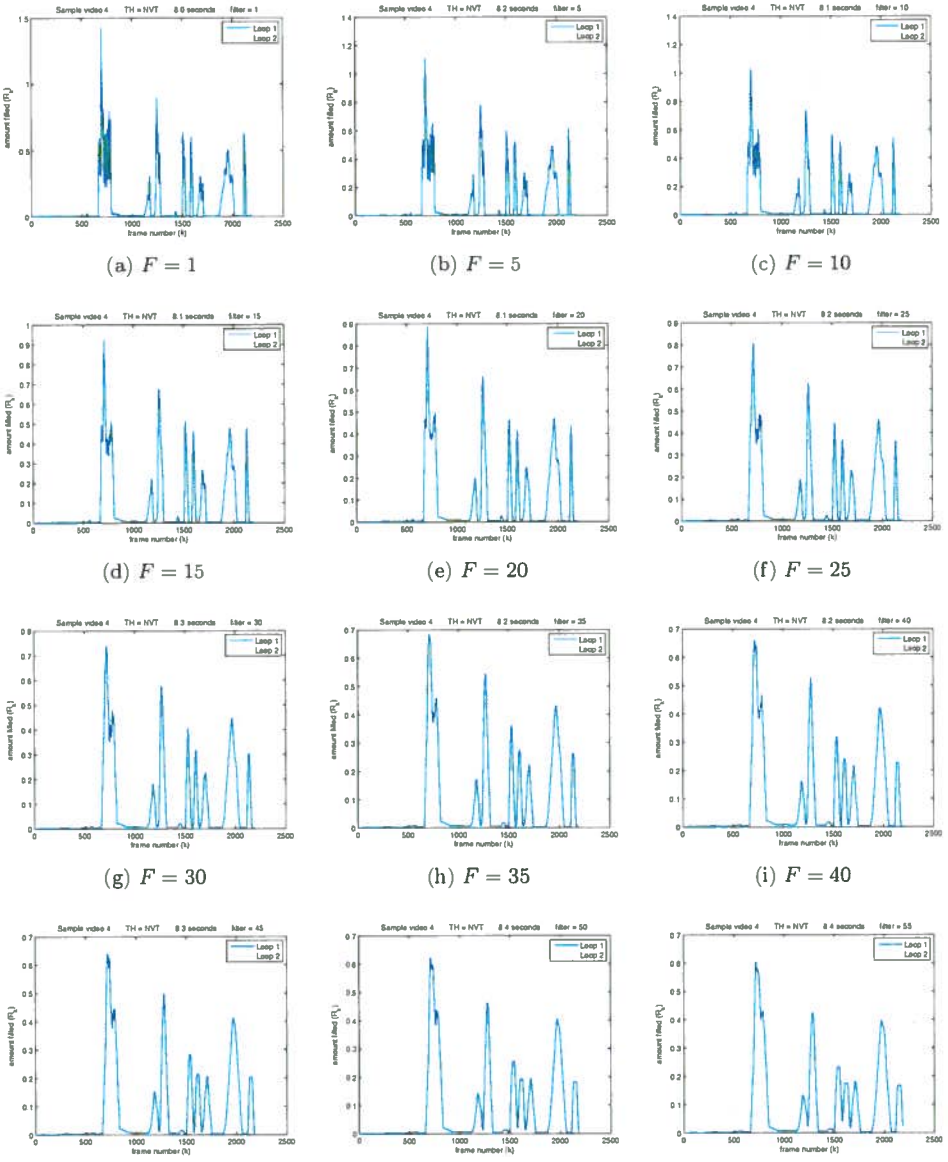


(a) Average over loops (b) Average split into colors (c) Correlation split into colors

Figure A.9: Results (R_k) for the extended algorithms applied to sample video 5 for the train

A.4 Various values of F

A.4.1 Traffic



(a) $F = 1$ (b) $F = 5$ (c) $F = 10$

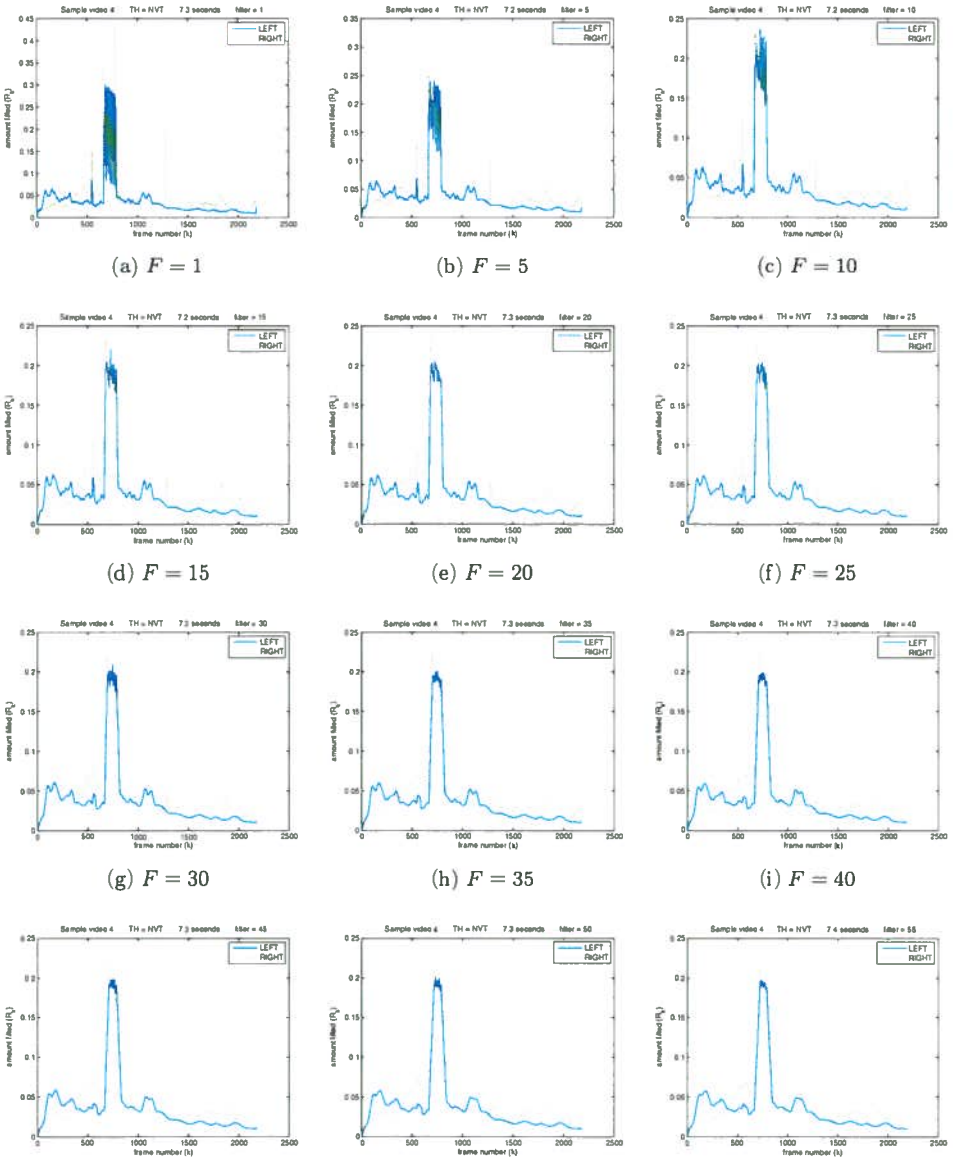
(d) $F = 15$ (e) $F = 20$ (f) $F = 25$

(g) $F = 30$ (h) $F = 35$ (i) $F = 40$

(j) $F = 45$ (k) $F = 50$ (l) $F = 55$

Figure A.10: Results of $\bar{R}_k^{\mathrm{CORR}}$ for various F applied to the traffic

A.4.2 Train



(a) $F = 1$ (b) $F = 5$ (c) $F = 10$

(d) $F = 15$ (e) $F = 20$ (f) $F = 25$

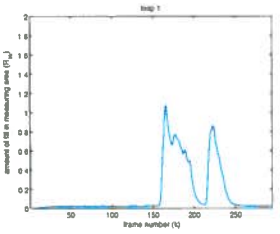
(g) $F = 30$ (h) $F = 35$ (i) $F = 40$

(j) $F = 45$ (k) $F = 50$ (l) $F = 55$

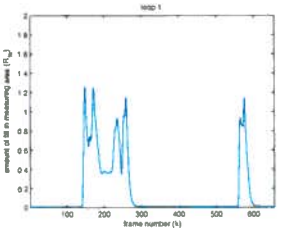
Figure A.11: Results of $\bar{R}_k^{\text{CORR}}$ for various F applied to the train

A.5 Results for R_k

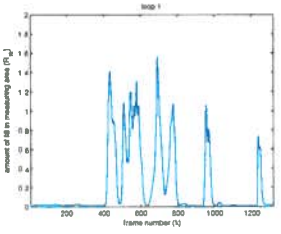
This section shows the results for R_k^{CORR} on loop 1 and loop 2 for all video's.



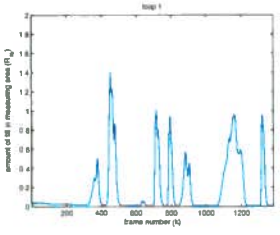
(a) Sample video 1



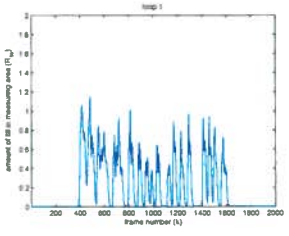
(b) Sample video 2



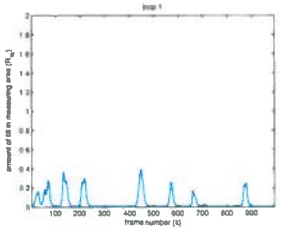
(c) Sample video 3



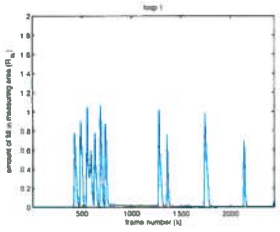
(d) Sample video 4



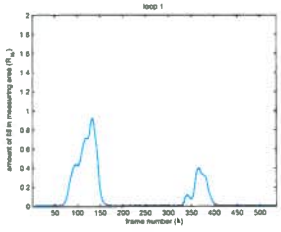
(e) Sample video 5



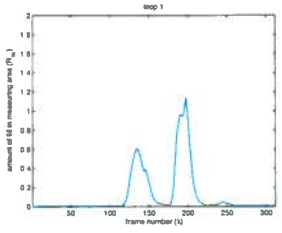
(f) Sample video 6



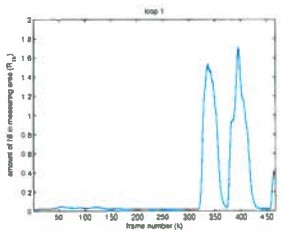
(g) Sample video 7



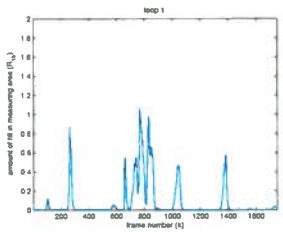
(h) Sample video 8



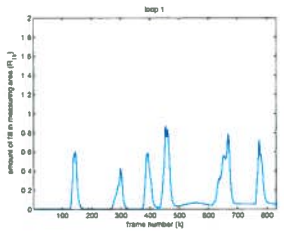
(i) Sample video 9



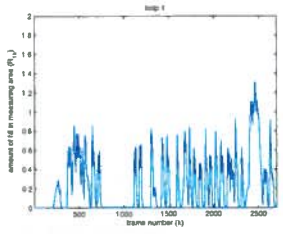
(j) Sample video 10



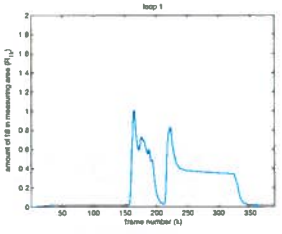
(k) Sample video 11



(l) Sample video 12

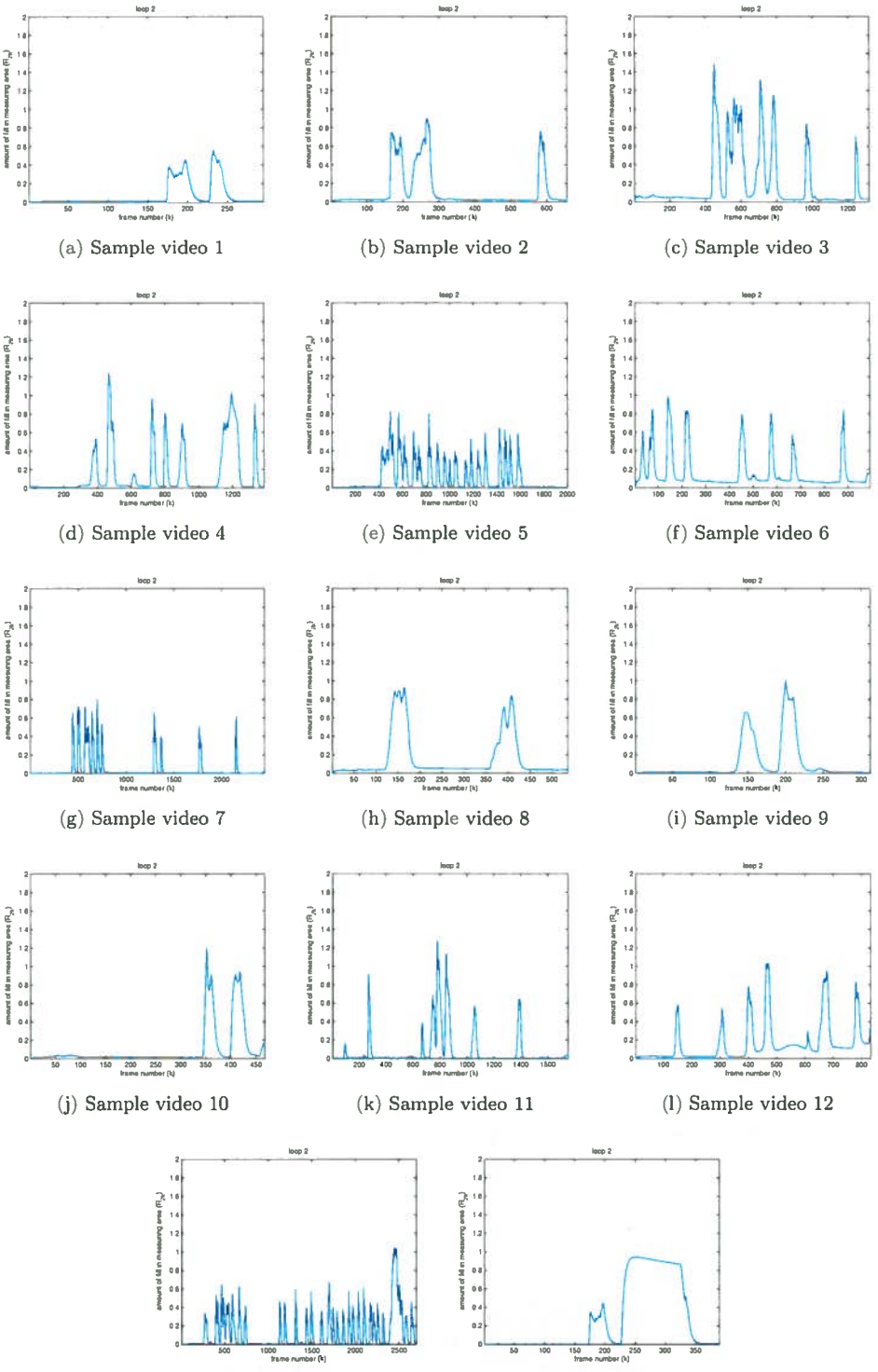


(m) Sample video 13



(n) Sample video 14

Figure A.12: Result for R_k^{CORR} applied on loop 1



(a) Sample video 1 (b) Sample video 2 (c) Sample video 3

(d) Sample video 4 (e) Sample video 5 (f) Sample video 6

(g) Sample video 7 (h) Sample video 8 (i) Sample video 9

(j) Sample video 10 (k) Sample video 11 (l) Sample video 12

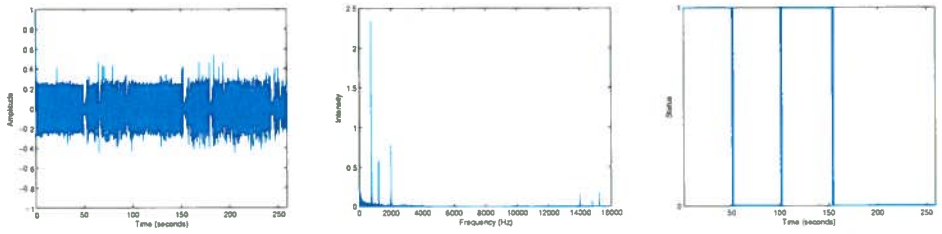
(m) Sample video 13 (n) Sample video 14

Figure A.13: Result for R_k^{CORR} applied on loop 2

A.6 Results for the audio

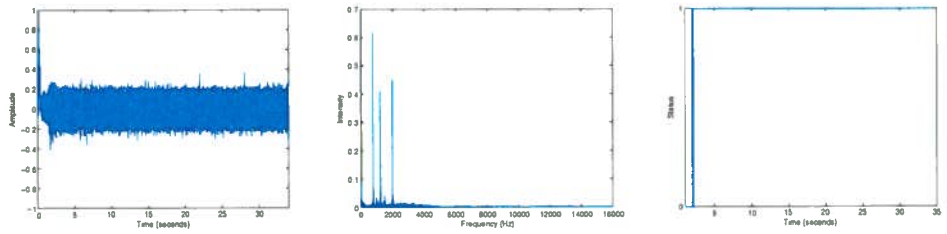
This section shows the original audio file, its Fourier transformation and the status of the bell for several sound files. For the graphs of the status there holds:

$$L_{\text{bell}} = 0.003$$



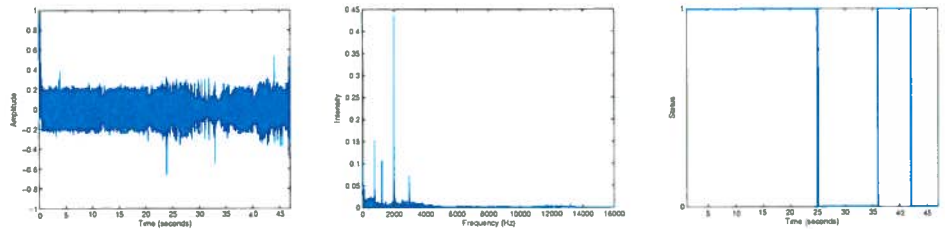
(a) original (b) Fourier (c) status

Figure A.14: Results sample audio 2



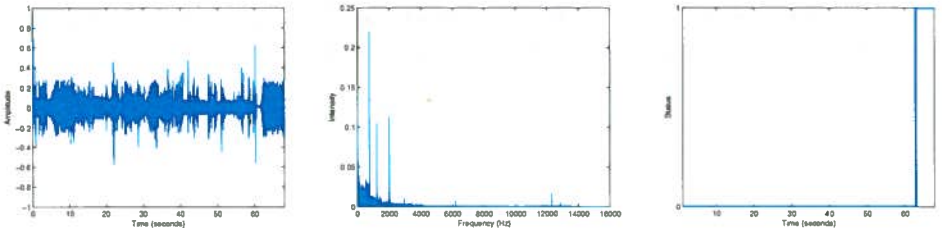
(a) original (b) Fourier (c) status

Figure A.15: Results sample audio 3



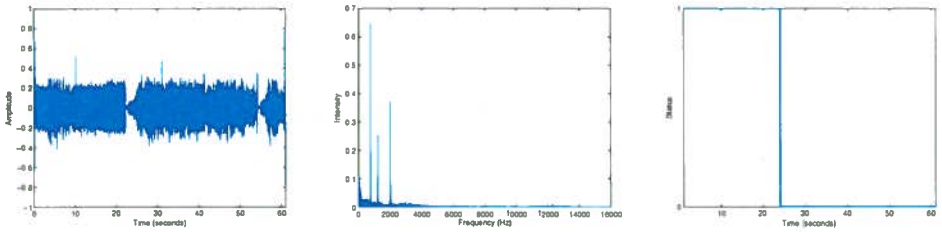
(a) original (b) Fourier (c) status

Figure A.16: Results sample audio 4



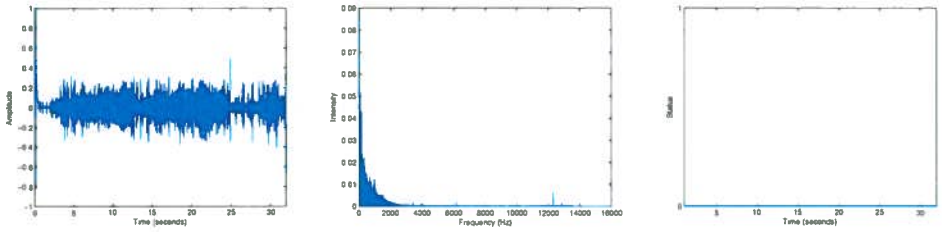
(a) original (b) Fourier (c) status

Figure A.17: Results sample audio 5



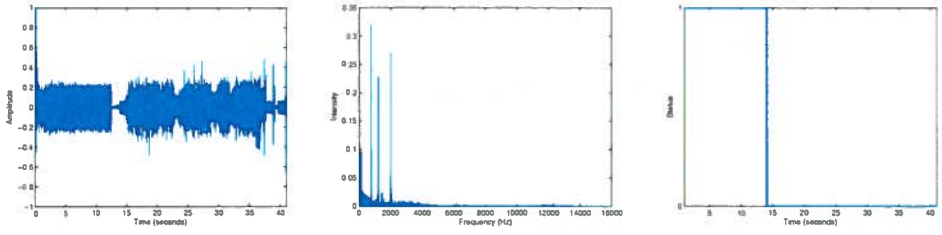
(a) original (b) Fourier (c) status

Figure A.18: Results sample audio 6



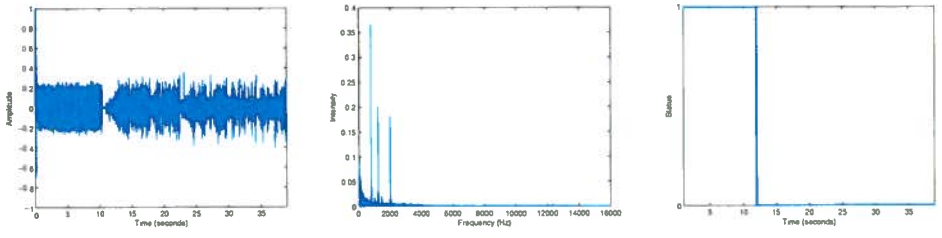
(a) original (b) Fourier (c) status

Figure A.19: Results sample audio 7



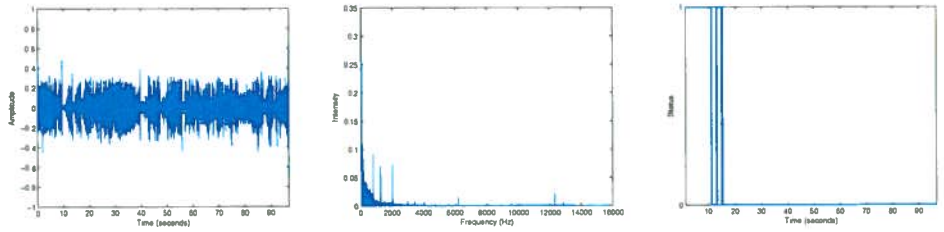
(a) original (b) Fourier (c) status

Figure A.20: Results sample audio 8



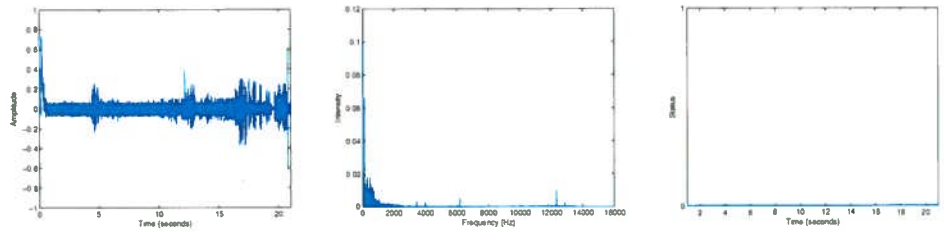
(a) original (b) Fourier (c) status

Figure A.21: Results sample audio 9



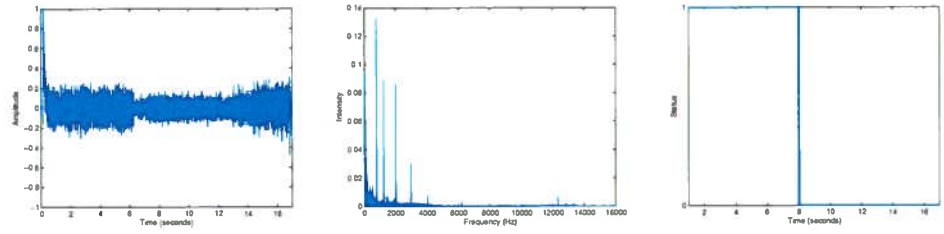
(a) original (b) Fourier (c) status

Figure A.22: Results sample audio 10



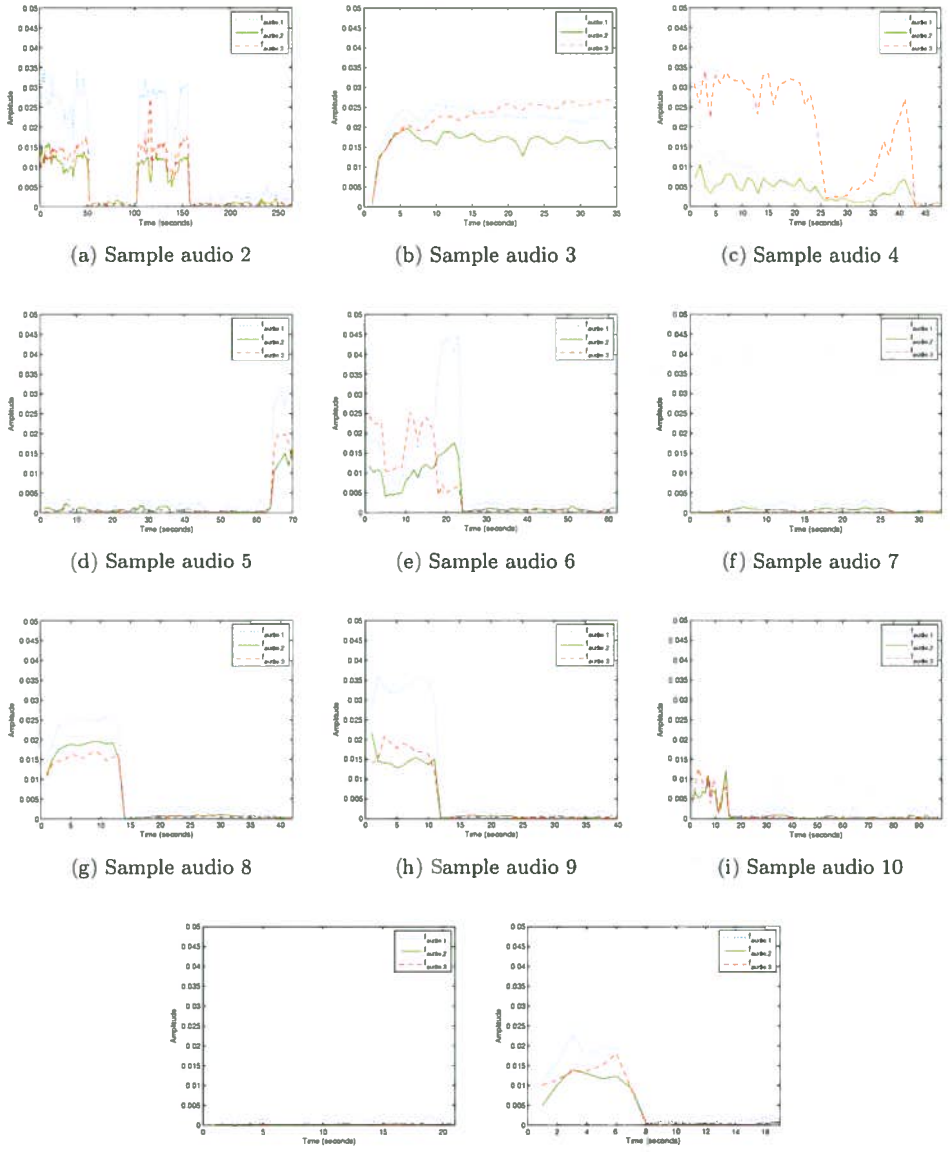
(a) original (b) Fourier (c) status

Figure A.23: Results sample audio 11



(a) original (b) Fourier (c) status

Figure A.24: Results sample audio 12



(a) Sample audio 2 (b) Sample audio 3 (c) Sample audio 4

(d) Sample audio 5 (e) Sample audio 6 (f) Sample audio 7

(g) Sample audio 8 (h) Sample audio 9 (i) Sample audio 10

(j) Sample audio 11 (k) Sample audio 12

Figure A.25: Developement of the frequencies

Appendix B

Legend for the samples

B.1 Video

Table B.1: Explanation of sample video numbers

Sample video #	Filename
1	"IMG20012.MPG"
2	"IMG21009.MPG"
3	"IMG20006.MPG"
4	"Boom2.avi"
5	"Almelo-fotocamera-deel1.avi"
6	"IMG20007.MPG"
7	"IMG20011.MPG"
8	"IMG20012_2.MPG"
9	"IMG20002_2.MPG"
10	"IMG20004.MPG"
11	"IMG20005.MPG"
12	"IMG20008.MPG"
13	"Almelo-fotocamera-deel2.avi"
14	"Stilstand4.avi"
15	"IMG20010_2.MPG"

B.2 Audio

Table B.2: Explanation of sample audio numbers

Sample audio #	Filename
1	"Boom2.wav"
2	"Almelo-fotocamera.wav"
3	"Boom1.wav"
4	"IMG20004.wav"
5	"IMG20005.wav"
6	"IMG20006.wav"
7	"IMG20008.wav"
8	"IMG20009.wav"
9	"IMG20010_2.wav"
10	"IMG20011.wav"
11	"IMG20012_2.wav"
12	"Nacht.wav"

Bibliography

[1] M. A. Botchev. Scientific computing. PDF document, Mei 2008. Lecture notes.

[2] A. Bovik. *Handbook of Image and Video processing.* Academic Press, 2000.

[3] Gerard Blanchet; Maurice Charbit. *Digital Signal and Image Processing using* MATLAB. ISTE Ltd, 2006.

[4] Rafael C. Gonzalez; Richard Eugene Woods; Steven L. Eddins. *Digital Image Processing Using* MATLAB. Pearson Prentice Hall, 2004.

[5] E. Hamilton. Jpeg file interchange format, September 1992.

[6] Alasdair McAndrew. *An Introduction to Digital Image Processing with* MATLAB. Victoria University of Technology - School of Computer Science and Mathematics, 2004.

[7] G. Meinsma. Advanced techniques for signal analysis. PDF, December 2008. Lecture notes.

[8] J. L. Rodgers; W. A. Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):5966, February 1988.

[9] M. Schoemaker. Visuele overweg monitoring : Stageverslag over de ontwikkeling van een overwegmonitoringssysteem met behulp van cameras. Internship report, University of Twente, Strukton Rail Consult, 2006.

[10] Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein. *Introduction to algorithms.* MIT Press, 2001.

[11] G. H. Golub; A. Hoffman; G. W. Stewart. Linear algebra and its applications. *A generalization of the Eckart-Young-Mirsky matrix approximation theorem*, 88-89:317–328, 1987.

[12] J. W. Cooley; J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, April 1965.

[13] F. van der Heijden; R.P.W. Duin; D. de Ridder; D.M.J. Tax. *Classification, Parameter Estimation and State Estimation : An Engineering Approach using* MATLAB. John Wiley & Sons, Ltd, 2004.

[14] G. H. Golub; C. F. van Loan. *Matrix computations.* Johns Hopkins, 3 edition, 1996.

[15] V. V. Vazirani. *Approximation Algorithms.* Springer, 2004.

[16] Eric W. Weisstein. Cauchy's inequality. June, 2009, `http://mathworld.wolfram.com/CauchysInequality.html`. From MathWorld–A Wolfram Web Resource.

[17] Eric W. Weisstein. Least squares fitting. June, 2009, `http://mathworld.wolfram.com/LeastSquaresFitting.html`. From MathWorld–A Wolfram Web Resource.

[18] Eric W. Weisstein. Least squares fitting–perpendicular offsets. June, 2009, `http://mathworld.wolfram.com/LeastSquaresFittingPerpendicularOffsets.html`. From MathWorld–A Wolfram Web Resource.

[19] James Hardy Wilkinson. *The algebraic eigenvalue problem.* Oxford University Press, 1988.

[20] Yin Zhang. Users guide to lipsol. *Optimization Methods and Software*, 11-12:385–396, 1999.