Motion Detection and Temporal Filtering of Noisy Image Sequences

Helena Olén, Marcus Wennermark

Master's thesis
2012:E32

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Motion Detection and Temporal Filtering of Noisy Image Sequences

Helena Olén π07
Marcus Wennermark F07

September 14, 2012

Abstract

When capturing digital video there is always some amount of noise in the resulting signal. This is more pronounced in low-light conditions. In this thesis we have evaluated five algorithms for motion detection and noise reduction. All algorithms produce a filtered image and a motion mask. One of the algorithms is based on block-matching, one on fuzzy logic and one on low-rank matrix completion. The final two are much simpler. The first of these estimates the standard deviation of the noise and thresholds the grey-level differences based on that. The last one is a novel approach, relying on spatial smoothing to create motion masks. The generated motion masks are then used to determine what parts of the image that can be filtered temporally. Temporal filtering is done using weighted averaging.

MATLAB is used for implementation and evaluation of all algorithms. Some are then implemented in OpenCL for testing in real-time using actual cameras.

The methods have been analysed regarding the quality of the filtered output as well as the accuracy of the generated motion masks. This has been done using a synthetic image sequence, where a noise-free reference exists. Real-world captured sequences have been judged subjectively.

The results indicate that the two simplest methods are the most efficient and the best of the evaluated algorithms for light and moderate noise. For images corrupted by heavier noise, the method based on low-rank matrix completion seems the most promising.

Acknowledgements

We would like to thank Axis Communications AB for enabling this thesis. In particular we want to thank our supervisors Fredrik Olofsson and Andreas Nilsson at Axis for all the help and support during this project. Further, we would like to thank our supervisor at LTH, Kalle Åström. Finally, a thank you to everyone else who have provided us with input and support. All the feedback has been greatly appreciated and invaluable in the evolution of this project.

Contents

Chapter 1

Introduction

1.1 Problem description

The goal of this thesis is to do a survey of the literature, investigate algorithms for noise robust scene change detection and temporal filtering, and to evaluate a few such methods on real data.

After the initial literature review, a few algorithms will be selected for implementation and further investigation. These algorithms will be evaluated mainly based on their change detection and/or their denoising performance. Also the feasibility of implementing them in hardware will be taken into consideration.

Some datasets with different characteristics will be chosen for the evaluation. In order to evaluate the effectiveness of the algorithms, some quality metrics are needed.

Ideally an algorithm should be able to accurately separate motion from noise and perform filtering using the correct pixel information from all available frames. Unfortunately, it is no simple task to achieve this. There are many problems with detecting true pixel change in the presence of heavy noise or in weak contrast areas. With strong noise, pixel values will fluctuate heavily and are easily confused with motion. This problem is the motivation of the thesis.

1.2 Overview

The layout of this thesis is as follows. Firstly, the background of the problem is explained in chapter 2. Then, a few metrics regarding the output quality of the algorithms are described in chapter 3. In chapter 4 we explain the underlying theory that is needed to understand the implemented algorithms. The algorithms themselves, and the changes or improvements we have made are de-

scribed in chapter 5. Details about the OpenCL implementations can be found in chapter 6. In chapter 7 our results are presented, followed by discussion and conclusions in chapter 8.

Chapter 2

Background

New generations of noise filtering often utilize temporal image data in order to suppress random noise without severe loss of spatial resolution. The most basic form of temporal filtering is a simple average on a per pixel basis of several consecutive images. This technique is very beneficial in comparison to spatial filtering for static scenes, i.e. scenes without motion. However, in case of a sudden change in scene content, the naive temporal average method fails and other methods must be used.

The difficulty of differentiating motion from noise is what causes problems in the temporal filtering step. If an object moves in a noisy image, and temporal averaging is performed on incorrectly classified pixels, a ghosting effect of the object will be seen in the subsequent images. An example of this can be seen in Figure 2.1.

Figure 2.1: An example of where the temporal averaging technique fails due to motion in the image.

Thus, there is a need for a robust method for detecting sudden scene changes. The challenge for such methods is to detect true scene changes in situations when the signal-to-noise ratio, *SNR*, is low. Typically this happens when the scene illumination is poor.

2.1 Related work

The problem of motion detection in images is a widely explored area, although most of the existing methods are either not suitable for images containing a high amount of noise, they require too much memory or have excessively high computational complexity.

A short description of the reviewed literature will follow. The curious reader is referred to the original articles for a more thorough explanation of each algorithm.

Binary motion detection

Several algorithms have been proposed to avoid these artifacts. The most common are the binary motion detectors, which classify each pixel as a motion pixel or a non-motion pixel. A pixel is averaged temporally if it is a non-motion pixel, otherwise the value of the pixel in the filtered frame takes the unprocessed value. An example of this approach is the method used by Bosco et al. [2]. This method uses a small working window in two consecutive frames. The algorithm works directly on the Bayer pattern data and is based on something called Duncan filtering, which requires a noise estimate to work. The motion detection is a two step method that first tries to detect the possibility of a ghost tail. The second step in the motion detection is to threshold the sum of absolute differences for the two working windows. Thereafter temporal averaging is executed on the non-motion pixels. The benefit with this approach is the speed, but the accuracy suffers in low-light conditions.

Another binary motion detector, based on the difference in grey level between the same pixel in two following frames, is proposed by Wu et al. [23]. The threshold for motion classification varies linearly with the standard deviation of the noise, which is assumed to be Gaussian noise.

A completely different motion detection approach is proposed by Stauffer and Grimson [18] and also later improved by Kaewtrakulpong and Bowden [11], where each pixel is modelled using a mixture of Gaussians. Here, objects that are not moving are integrated into the background, and motion is detected when a pixel differ too much from the background model corresponding to that pixel. These methods do not detect small movements, and the authors are not clear about the application of the algorithms on noisy images. Also, the background takes a long time to update. That is, it takes a long time for new stationary objects to be integrated in the background model, which is disadvantageous when many objects are leaving or entering the scene. This is may be the case in surveillance applications.

Non-binary motion detection

When a high level of noise is present in the image, many binary detectors introduce motion blurring and artifacts due to misclassification caused by motion ambiguity. To find ways to detect errors, or decrease uncertainties in the temporal filtering, algorithms that do not use binary motion detection are considered. Instead of determining whether the pixel is a motion pixel or not, some degree of motion probability or motion confidence is determined. The degree of motion, is typically a real number in the range $[0, 1]$, where 0 represents no motion and 1 represents motion for sure. This number then decides, directly or indirectly, the weights that are used to average the previous filtered frame and the current frame. Bennett and McMillan [1] use temporal filtering when no motion is detected, and a mixture between spatial and temporal filtering based on local motion characteristics when motion is present. The motion is detected using a normalized Gaussian weighted dissimilarity formula.

Another approach, suggested by Malm et al. [14], is based on edge detection and that filtering is performed in the direction in which the edges propagate in time. The method models the principle of nocturnal vision.

The algorithm proposed by Kim et al. [12], uses two consecutive frames to do hierarchical block matching and sum of absolute differences to determine the correlation between the two matched blocks. The degree of motion is then calculated from the correlation. The temporal filtering is thereafter performed with different weights for the current pixel in the new frame and the matched pixel in the previous frame. Filtering in the motion direction makes it a motion-compensated method.

Portz et al. [15] use optical flow to perform temporal filtering. If the flow is defined as reliable, then the image is filtered in the flow direction. Otherwise mostly spatial filtering is used. For areas with uniform colour and texture, the flow is determined as unreliable, which is a major drawback for our purpose.

Another approach is to use fuzzy logic to detect motion. This is described by Zlokolica et al. [25]. Temporal filtering is performed with adaptive weights using the new unfiltered image and the previous filtered one, depending on the motion confidence for the pixel. More details regarding fuzzy logic will be explained in chapter 5.

Yet another approach is to use a modified version of Non-local means as proposed by Xu et al. [24]. They separate temporal and spatial filtering and combine them with different spatial and temporal weights that vary depending on how much motion that is detected. The motion detection is based on computing the average similarities of the spatial and temporal neighbourhoods. One of the most important parts is this weighting of the spatial and temporal filtering. Only using the temporal part of this algorithm would not produce a meaningful result. This means that it falls too far outside the scope of this project and is therefore discarded.

Level sets is another way to solve the motion detection problem, which is discussed in Woo et al. [22] and ho Lee et al. [6]. These methods have shown very good results in differentiating the moving objects from the static background. However, these approaches are not suitable for our problem, since they detect the whole object. Detecting the whole object might be a disadvantage if the object is big and has areas with uniform colours. Even if the object has moved a bit, the temporal averaging could still be performed when the areas are similar in order to reduce the noise.

Veit et al. [19] proposed an algorithm based on motion detection in regions, which is very successful in finding moving objects. Unfortunately they only seem to detect edges of the moving objects, which is not adequate for our motion detection application. There are also issues with low-contrast regions. This method falls more into the higher-level image analysis category.

Several statistical methods using Markov random fields have been presented to solve the motion detection problem. In Crivelli et al. [4] the proposed algorithm simultaneously detects motion and updates the background image. Motion dynamics are modelled with *mixed-state discrete-continuous Markov random fields*, and to create the joint decision-estimation solution no training samples are needed. The motion is estimated by minimizing an energy function. The method suggested by Chen and keung Tang [3] uses a non-parametric noise model to allow the noise to change over time. Optical flow is used as a basic motion detector and to model and utilize the motion uncertainty, a probabilistic motion field is used. Due to limitation of time, these methods have not been implemented, even though they are certainly interesting.

An algorithm that differs from most temporal filtering algorithms is described in Ji et al. [10]. A patch matching is done, both spatially and temporally, by a fast block-matching algorithm. Thereafter a noise estimation is performed for all pixels. The matched patches are concatenated into a matrix, and the pixels that are estimated to be noise pixels are removed. The noise removal problem is hereby converted to a low rank matrix completion problem, which is solved by a minimization algorithm. This method is interesting both for the denoising performance of the sample images in the paper as well as for being such a different method.

The methods we have chosen to implement can be seen in the table below. Furthermore. a novel algorithm is presented in addition to the algorithms found in the literature.

Method	*Author*	*Section*
Adaptive Fuzzy Filtering	Wu et al. [23]	5.1
Hierarchical Block Matching Algorithm	Kim et al. [12]	5.2
Fuzzy Logic Algorithm	Zlokolica et al. [25]	5.3
Low-Rank Matrix Completion Algorithm	Ji et al. [10]	5.4
Blur Filtering Algorithm	H. Olén and M. Wennermark	5.5

Chapter 3

Quality Metrics and Data Sets

3.1 Quality metrics

When looking at different filters, it is important to be able to somehow quantify and measure the effectiveness of the filters. This is especially important when trying to determine optimal parameters for each filter and each sequence. There are of course many ways to do this. The number of options available is also different depending on whether a clean reference sequence is available or not. If no reference sequence is available then one usually resorts to measuring subjectively, i.e. one simply looks at the images and decides if the result is good or not. Typically there is no reference when using real image sequences which is why synthetic sequences are also needed. A synthetic sequence is a noise-free sequence where noise is added. With a synthetic sequence it is possible to know exactly which pixels should be considered moving or not since the sequence is manually corrupted.

The problem of finding good quality metrics is very difficult, and a research field in its own right. There exists many error measures. Two very common are *mean square error* (MSE) and *peak signal to noise ratio* (PSNR). These are popular because of their simplicity of implementation. The main objection to using these measures is their inability to factor in information about the visually perceived error, or the structure in the image. Another common quality metric is *structural similarity index* (SSIM), which also takes structural similarities into consideration [20].

The quality of the motion mask must also be measured. This can be done in many ways if there exists a reference mask. The reference mask represents the true motion pixels and can be compared to the mask generated by the algorithms.

3.1.1 Error

The error of a measured, or corrupted, pixel is defined as the difference from a, clean, reference pixel. I.e.

$$error(x, y) = I(x, y)_{ref} - I(x, y)_{noisy}.$$

3.1.2 SNR

Signal to Noise Ratio is defined as the ratio of the power of the signal to the power of the noise. An alternative definition is

$$\text{SNR} = \frac{\mu_{signal}}{\sigma_{noise}},$$

where μ_{signal} is the mean of the signal and σ_{noise} is the standard deviation of the noise [17].

3.1.3 MSE

Mean Square Error is the mean of the squared error of all pixels:

$$\text{MSE} = \frac{\sum_{\forall(x,y)}(\text{error}(x, y))^2}{N_{pixels}}.$$

This is a popular error metric because it is trivial to implement and requires no parameters.

3.1.4 PSNR

Peak Signal to Noise Ratio is another error metric that is easy to implement. The definition is

$$\text{PSNR} = \frac{R^2}{\text{MSE}},$$

where R is the maximum signal value, often 1 or 255 when dealing with images.

3.1.5 SSIM

One measure that attempts to improve on the shortcomings of metrics such as PSNR or MSE is *Structural SIMilarity index* [20], which is the objective measure of choice in this thesis. Although not a true perceptual error metric it does at least take into account changes in structural information. The closer to 1 the SSIM value gets, the more similar the two images are. An index value of 1 can really only be achieved when comparing identical images. An ideal measure would also consider things like human visual response functions. SSIM is also relatively easy to implement and several versions exist online. The one used in this project is a MATLAB implementation [21] by Wang et al.

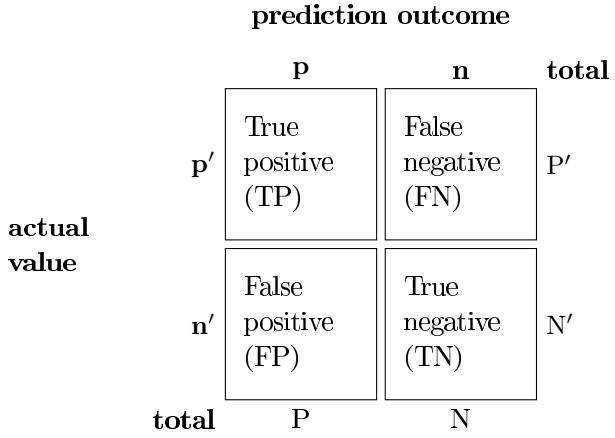


Figure 3.1: A confusion matrix.

3.1.6 Motion masks

For the synthetic sequences, a reference motion mask can be produced by subtracting two subsequent noiseless images. This motion mask can be compared to the motion mask that the different methods generate. By measuring which pixels that are classified correctly in the motion masks from the methods, a *confusion matrix* can be obtained, see Figure 3.1. The motion mask contains the numbers 0 and 1, where 0 is defined as a non-motion pixel and 1 is defined as a motion pixel. When comparing the method motion mask with the reference motion mask, four different detection statistics can be obtained. These are *true positives (TP)*, *false positives (FP)*, *true negatives (TN)* and *false negatives (FN)*, where true positives represent the number of actual motion pixels classified as motion, false positives represent the number of non-motion pixels classified as motion pixels, true negatives represent the number of correctly classified non-motion pixels and false negatives corresponds to the number of actual motion pixels classified as non-motion pixels.

From this confusion matrix, many statistics can be extracted. Only the ones that are used in this thesis will be covered.

The true positive rate, TPR, also called the sensitivity can be obtained by

$$TPR = \frac{TP}{P'} = \frac{TP}{TP + FN}. \tag{3.1}$$

This is equivalent to the hit rate. Another measurement is the false postive rate, FPR, which can be seen as the fall-out. FPR is calculated by

$$FPR = \frac{FP}{N'} = \frac{FP}{FP + TN}. \tag{3.2}$$

The accuracy, ACC, of the classification is obtained by

$$ACC = \frac{TP + TN}{P' + N'}. \tag{3.3}$$

Depending on the distribution of the number of motion and non-motion pixels, the accuracy is more or less important. If the distribution is similar for the two classes, then the accuracy is a good measurement, while if the distribution is very different, the information about the accuracy should have less influence on the conclusions.

3.1.7 Comparison between methods

Several ways of evaluating the methods are described above, but a way of comparing them to each other is just as important. One can always compare them subjectively by simply looking at the filtered images to find artifacts and see how much the noise is reduced. In this thesis we have decided to use the SSIM values and the statistics from the motion masks and plot them with the average SSIM value for a sequence on the x-axis and a value from the average motion statistics on the y-axis. The closer to 1 on both axis, the better the method is. The formula for calculating the value from the motion mask statistics is chosen to be

$$y = \frac{1}{2}\overline{TPR} + \frac{1}{4}(1 - \overline{FPR}) + \frac{1}{4}\overline{ACC}. \tag{3.4}$$

The motion mask is supposed to be good if the $\overline{TPR}$ and the $\overline{ACC}$ are high and the $\overline{FPR}$ is low, i.e. the *true negative rate*, $\overline{TNR} = (1 - \overline{FPR})$ is high. Classifying a motion-pixel as a non-motion pixel is worse than classifying a non-motion pixel as a motion pixel, since that will lead to ghosting. For this reason, the weight for the TPR is set to be higher. This plot will show if there is any correlation between the average SSIM value and the value generated from the average motion mask statistics.

3.2 Data sets

To evaluate the algorithms, some data sets are needed. Both image sequences captured by a real camera, and synthetic images are needed in order to test the methods properly. The reason for this is that the noise characteristics are different in the images captured by the actual camera compared to the images with artificially added noise. The real-world sequences contain noise that is correlated with the intensity of the signal. Pixels with higher intensity, i.e. bright pixels, are corrupted by noise with higher standard deviation compared to the darker pixels. In the synthetic sequences, where the original image is noise-free and Gaussian noise with different standard deviation is added, the standard deviation of the noise is constant in the whole image and independent of the intensity.

Also, the quality metrics from Section 3.1 can only be performed on synthetic data sets where a noise-free reference image and a motion mask are available.

It is entirely possible to add signal dependent noise to the synthetic sequences as well. The reason for not doing so is that it would add another level of complexity when studying the synthetic sequences.

In this thesis, mainly three different sequences are used to evaluate the performance of the algorithms. The first one is a sequence of a bus captured by a surveillance camera in a low-light condition. The second and third ones are taken from the animated movie Big Buck Bunny, [5]. Two sequences from the movie are used: frames 2195-2394 and frames 2495-2704. They contain static backgrounds with varying amount of texture as well as both small and large movements.

In the Appendix, a few more sequences are presented but the performance of the filtering did not differ much from the sequences mentioned above and is therefore not included in the results or discussion.

Chapter 4

Theory

4.1 Image representation

Images can be stored in different ways. In this section, the colour models used in this thesis are presented.

4.1.1 RGB

The way to describe colour image data that is most familiar to people is probably the RGB model. In the RGB colour model, each pixel is described by one red, one green and one blue component. These components are added together to produce different colours. There are several ways to represent these components. Two examples are the arithmetic, where each colour is represented by a real number in the range $[0, 1]$ and the digital 8-bit per channel, where each colour is assigned an integer number in the range $[0, 255]$. In these representations $(0, 0, 0)$ means black while $(1, 1, 1)$ or $(255, 255, 255)$ corresponds to white.

4.1.2 YCbCr

YCbCr is another colour image model, where Y represents the luminance, and Cb and Cr are the blue and red difference chrominance components respectively. The luminance is essentially a grey-scale image of the original colour image. YCbCr is a way of encoding RGB, meaning that the two representations are equivalent and it is easy to convert the image from YCbCr to RGB and back.

4.1.3 Grey-scale

A grey-scale image is an image containing different shades of grey. An image in RGB can easily be converted to grey-scale by e.g. computing a weighted average of the three components. Doing so yields a single number in the range $[0, 1]$ or $[0, 255]$, using the same ranges as above. The process of converting to grey-scale is not invertible, meaning that there is no way to get back the colour information from the grey-scale image.

4.1.4 Bayer pattern filter

Image sensors only respond to the intensity of the incoming light and as such they can really only capture grey-scale images. In order to get colour information, a colour filter array is placed in front of the sensor. This array is called a Bayer pattern filter. The Bayer filter array arranges RGB colour filters such that the filter pattern is 50% green 25% blue and 25% red. To obtain a full-colour image, the captured patterned grey-scale image is interpolated. This process of converting Bayer pattern data to RGB data is called demosaicing.

4.2 Noise sources

According to Janesick [9], sensor data is typically corrupted by four different types of noise; signal shot noise, Fano noise, fixed pattern noise and read noise. A sensor can be characterised using a method called photon transfer, see [9]. Whenever using cameras in this thesis, mainly photon shot noise, read noise and fixed pattern noise are considered.

4.2.1 Signal shot noise

Shot noise relates to how incoming photons interact with the sensor. The spatial distribution of the photons across the sensor is not uniform and this gives rise to what is called signal shot noise. This noise is signal dependent, $\sigma_{shot} \propto \sqrt{Signal}$.

4.2.2 Fixed pattern noise

The pixels in the sensor gather photons with slightly varying efficiency. These small differences generate a pattern that is the same in all frames, for a certain signal level. Because this noise is not random it is called "fixed pattern". It is possible to remove this type of noise through a process called flat-fielding. Fixed pattern noise standard deviation is proportional to the signal level, $\sigma_{FPN} \propto Signal$.

4.2.3 Read noise

Read noise encompasses all remaining noise sources that are independent of the signal level. There are a number of sources that goes into this, but it will be treated as a single source in this thesis.

4.3 Block-matching

Block-matching is a method that locates a block from one image in another image or other areas in the same image. The match can be measured in several ways. One of the most common ways to measure the degree of similarity between two blocks is to use SAD. The smaller the SAD-value, the more alike the two blocks are. Another common similarity measure is to use mean absolute difference, *MAD*, which uses the average of all absolute differences between two blocks.

There are different types of block-matching. The most accurate, but also the most computationally expensive one, is the full search, also called exhaustive search. Other block-matching techniques use different subsampling patterns and interpolation methods to reduce the computational complexity.

Exhaustive search

The exhaustive search compares the block to be matched with all possible blocks in the search area to find the optimal match in the other image. This match can then be used to compute, for example, a motion vector. For matching a block of size $k \times k$ in a $m \times n$-image $(m-k)(n-k)k^2$ pixels need to be visited.

Subsampled block matching

To reduce the computational load of the block matching, many algorithms have been proposed. One of them is the algorithm by Liu and Zaccarin [13]. This algorithm reduces the computational complexity of a full search by a factor of 8 by limiting the number of search locations and the number of pixels evaluated.

First, only every second block in the original image is matched to the blocks in the other image. For the blocks that are evaluated, a $4:1$ subsampling pattern is used. This means that only $\frac{1}{4}$ of the pixels are used when evaluating the pixel differences. Having four different patterns, see Figure 4.1 and alternating between them in a specific way, four matches are produced for each block, which leads to four motion vectors. These four motion vectors are then used to evaluate the full block, e.g. without subsampling, at the four resulting locations. The best match among these four is then selected as the final matching block. For a more complete explanation, consult [13].

a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d

Figure 4.1: The $4:1$ subsampling patterns for an 8×8 block.

The blocks that were ignored in the previous step, now need to be matched. This is done by looking at the motion vectors for the four neighbouring blocks, and among them choose the motion vector that gives the best match for the new block. This relies on the assumption that neighbouring blocks are likely to have similar behaviour, that is, they are likely to move in the same direction with the same speed. When motion vectors for all blocks in the original image are found, the block matching is finished.

4.4 Fuzzy logic

Fuzzy logic is an extension of ordinary binary logic. Instead of hard values of *true* and *false* it uses degrees of truth. This degree spans the entire continuous interval of $[false, true]$ which is often represented as a real number in $[0, 1]$. Just like binary logic, fuzzy logic also defines operations like *AND* and *OR*. These operations, however, are a little different when used in a fuzzy sense and there are several ways to define them. One property that should always be maintained is consistency with the binary versions of the operations. That is, when performing for example $1\ AND\ 0$ the result should be 0 just like in the binary case. The definitions of these two operations, for the purposes of this thesis, are:

$$a\ AND\ b \equiv a \cdot b, \tag{4.1}$$

$$a\ OR\ b \equiv a + b - a \cdot b. \tag{4.2}$$

where the variables a and b are fuzzy values defined by what is known as a membership function. Such a function takes as input some measurement and maps a truth-value to it. One example would be a glass that has a certain volume, V, of water in it. To get the degree of truth that this glass is full, the value is passed into a membership function μ_{full}. This function then outputs to what degree the glass is full. In this case the membership function could be very simple, e.g. $\mu_{full}(V) = V/V_{max}$. The term membership refers to what degree the glass can be said to belong to the group, or set, of full glasses [7].

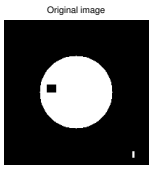
Using these membership functions, a system of *if-then* rules can be set up to determine what action to take. In the case of the somewhat full glass, one could say that

$$IF\ \mu_{full}(x)\ \ AND\ \mu_{thirsty}(y)\ \ THEN\ \ DRINK.$$

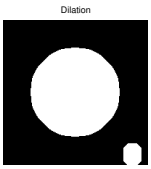
The output variable *DRINK* would then be a fuzzy value quantitatively stating the degree of certainty that you can drink.

4.5 Erosion and dilation

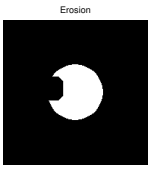
Binary images can be processed using morphological operations called *dilation* and *erosion*. In a black and white image, a cluster of white pixels is said to be an object. Dilation makes these objects bigger, and fill holes in the objects by adding pixels to the object boundaries. Erosion, on the other hand, shrinks the objects by removing pixels on the object boundaries. To add or remove pixels, a structuring element is needed. This structuring element can assume different shapes and sizes. How many pixels of each object that is added or removed depends on the shape, as well as the size of the structuring element. Erosion and dilation are often combined to get the desired results. When combining them, the order in which they are performed is important. First doing erosion and then dilation is not equivalent to first dilating and then eroding. Erosion followed by dilation results in opening and dilation followed by erosion results in closing of the objects. An example of this is shown in Figure 4.2.



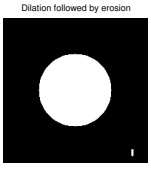
(a) Original image



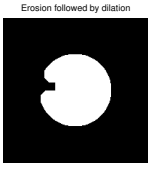
(b) Dilated image



(c) Eroded image



(d) Dilation followed by erosion



(e) Erosion followed by dilation

Figure 4.2: An image processed using dilation and erosion with a disk of radius $r = 12$ as structuring element.

Chapter 5

Methods

From the methods that we have looked at we have selected four from the literature study and one new method for further study and comparison. These are prototyped using MATLAB and some are then selected for implementation in OpenCL to achieve the goal of real-time performance.

5.1 Adaptive fuzzy filter algorithm

A fairly simple and fast algorithm is presented by Wu et al. [23]. Firstly, a noise estimation for the image is done. Thereafter, an average of the luminance values for the 3×3 block centred at the current pixel is calculated both in the current and the previous frame. The absolute difference between the previous and current average luminances is calculated. This absolute difference is then thresholded. The threshold is based on the standard deviation of the noise. If the absolute difference is larger, the processed center pixel obtains the current value. If it is lower, the new center pixel is an average of the previous filtered pixel and the new unfiltered pixel.

Noise estimation

The standard deviation of the noise in each image is a measure of the noise intensity in the image, and is therefore calculated before the actual motion detection step is performed.

Here, differential operator based method [23], is used for the estimation. The procedure starts by calculating the difference of two Laplacian directional oper-

ators $dL = -2(L_2 - \frac{1}{2}L_1)$, where

$$L_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

and

$$L_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

resulting in the matrix

$$dL = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}. \tag{5.1}$$

Applying the operator dL (5.1) reduces the impact of the underlying image according to [16]. The noise information is obtained for each pixel by

$$n(x,y) = \sum_{j=-1}^{1} \sum_{i=-1}^{1} dL(1+i, 1+j)v(x+i, y+j), \tag{5.2}$$

where $v(x,y)$ is the grey value of pixel $I(x,y)$. Then, the standard deviation of the noise can be calculated as

$$\sigma_n = \left(\frac{\sum_{y=1}^{H-2} \sum_{x=1}^{W-2} n(x,y)^2}{36(W-2)(H-2)} \right)^{1/2}, \tag{5.3}$$

where H and W is the height and width of the image respectively.

Motion detection

Once the noise is estimated, the pixels need to be classified. Assuming that the noise is zero-mean additive Gaussian noise, the average value of the closest neighbouring grey values

$$\bar{v}(x,y) = \frac{\sum_{j=-1}^{1} \sum_{i=-1}^{1} v(x+i, y+j)}{9} \tag{5.4}$$

is the value resisting the most noise according to Wu et al. [23]. It is therefore used to classify motion.

For the grey pixel $v(x,y,t)$ of image $I(x,y,t)$ at time t and pixel (x,y) the motion $\alpha(x,y,t)$ is determined by

$$\alpha(x,y,t) = \begin{cases} 0, & \text{if } \mid \bar{v}(x,y,t) - \bar{v}(x,y,t-1) \mid \leq k\sigma_n \\ 1, & \text{if } \mid \bar{v}(x,y,t) - \bar{v}(x,y,t-1) \mid > k\sigma_n \end{cases}, \tag{5.5}$$

where σ_n is the estimated standard deviation of the noise, and $\bar{v}(x,y,t-1)$ is the average grey value of the previous filtered frame at pixel (x,y). The reference implementation always uses $k = 3$. This is a parameter that can be varied to tune the algorithm.

Temporal filtering

Here, a simple averaging of the pixels from the previous filtered frame and the current noisy frame is calculated when the pixel is determined to be a non-motion pixel. To avoid blurring and artifacts, the pixel value obtains the same value as the same pixel in the noisy frame when it is classified as a motion pixel. Using simple averaging, e.g. taking half of each image means that there will always be some noise left. By giving less weight to the unprocessed image, the denoising performance can be improved. This requires an accurate motion detection or there will be ghosting again.

5.1.1 Algorithm summary

Algorithm 5.1.1: ADAPTIVE FUZZY FILTERING ALGORITHM

1. Estimate the standard deviation of the noise.
2. Calculate the average grey value for the neighboring pixels in the current frame and previous filtered frame.
3. Classify motion pixels.
4. Filter the image temporally.

5.2 Hierarchical block-matching algorithm

This algorithm is based on the method proposed by Kim et al. [12]. First, block matching is used to detect motion and compute motion vectors. Thereafter, the image is filtered with the previous filtered frame in the motion vector direction for each pixel. If no match is found, the new pixel obtains the same value as the noisy frame. Both the block-matching and the filtering are performed on the luminance channel only. In order to get a good result and to avoid error propagation, colour correction and a post processing step are done before the new filtered image is finished.

Motion detection

The motion detection is done hierarchically using block-matching. Hierarchically means that the size of the blocks to be matched starts relatively big, for example 9×9, and if no sufficient matching is found, the block size is reduced. Starting with a frame of 9×9 pixels around the current pixel in the new unfiltered image, the matching is performed in an 11×11 frame in the previous filtered image, centred around the same pixel. If both the difference between

the maximum and the minimum SAD-value, and the minimum SAD-value are smaller than a predefined threshold, a matching block in the previous frame is considered to have been found. If not, the block size is reduced and the same procedure is repeated until the block size reaches 3×3. The sizes of the search windows, as well as the block sizes, can be varied to obtain different results.

When a match is found, the correlation ρ between the two blocks, i.e. the correlation between the current pixel and the motion compensated pixel, is set to $\rho = 1$. A correlation of $\rho = 1$ means that the block from the new unfiltered frame is considered to be exactly the same as the matched block in the old filtered frame. Using this fact, the motion vector for the pixel can be found by subtracting the position of the matched pixel in the previous frame from the position of the current pixel in the new frame.

In the case where no "perfect" match is found, and the block size already is the minimum block size, the correlation, ρ, is calculated by (5.6).

$$\rho = 1 - \frac{SAD_{min}}{\beta \cdot h \cdot w}, \tag{5.6}$$

where h is the block height, w is the width of the block and β is a user-defined parameter that can be used to tweak the method. Usually $\beta = 1$.

The whole motion estimation process can be seen in a flow chart in Figure 5.1.

The order in which the blocks are evaluated is an important aspect of this algorithm. It assumes that it is more likely that a pixel has moved a shorter distance than a longer and traverses the blocks in a way that reflects this. The search pattern is an outwards spiral with the current pixel as the center. In the case that the minimum SAD-value is found at more than one place in the spiral, the algorithm favours the position closest to the center.

Filtering

The filtering weight, α, that is used when interpolating between the old filtered frame $I_f(\bar{x}, \bar{y}, t-1)$ and the new frame $I_n(x, y, t)$ is determined by subtracting the sensitivity from the correlation, see (5.7). The indices for the block in the previous frame $(\bar{x}, \bar{y})$ is the motion compensated, or matched, pixel position.

$$\alpha = \rho - s, \tag{5.7}$$

where $\alpha = 0$, if $\rho - s < 0$. s is a user-defined sensitivity parameter.

The new filtered pixel is calculated by (5.8).

$$I_f(x, y, t) = \alpha I_f(\bar{x}, \bar{y}, t-1) + (1-\alpha) I_n(x, y, t), \tag{5.8}$$

meaning that for $\alpha = 0$, the filtered image attains the value from the noisy frame. Hence, $\alpha = 0$ indicates motion, while $\alpha = \alpha_{max} = 1 - s$ is considered as non-motion.

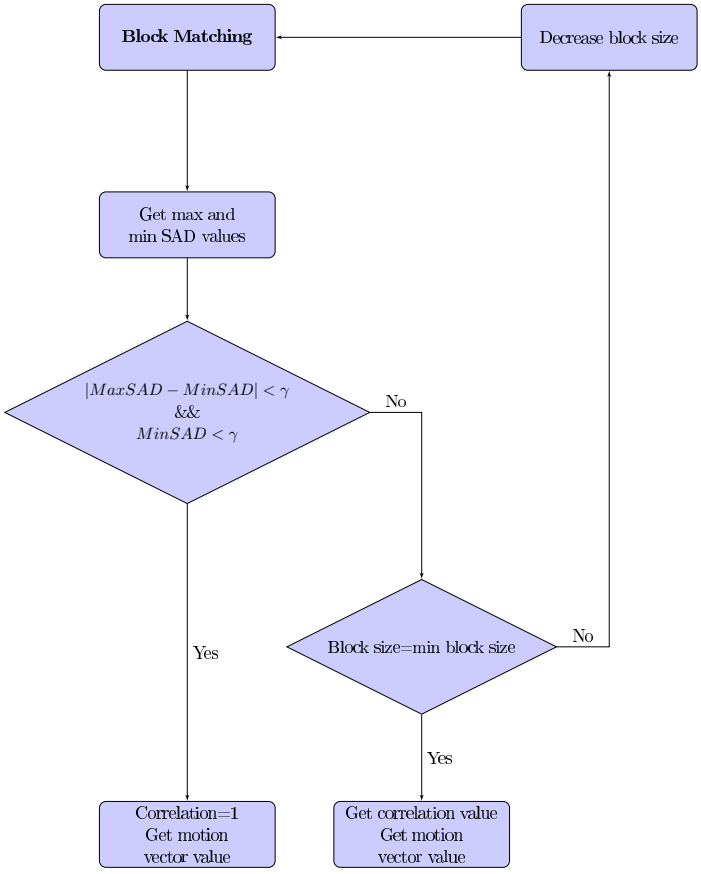


Figure 5.1: The motion estimation process for the hierarchical block-matching algorithm.

Colour correction

The images in this algorithm are assumed to be in the YCbCr model, but all steps above are only performed in the luminance channel Y. According to Kim et al. [12], the human eye is more sensitive to errors in the chrominance channels, Cb and Cr. Therefore, colour correction is necessary for these values in order to obtain a satisfactory result. If the Cb value and the Cr value for a pixel (x, y) have the same sign, equation (5.8) is applied to both chrominance channels, otherwise the previous values are used without any processing.

Post processing

In order to prevent error propagation some post processing is needed. If the differences between the three channels in the filtered frame and in the noisy frame, after the temporal filtering and the colour correction, are bigger than a threshold, the pixel is considered to be in error. The error pixel is then replaced by the value of the original unfiltered pixel. The threshold for the luminance difference, ΔY, is the same γ as above, and the threshold for the two chrominance channels $c \cdot \gamma$, where c is a user-defined constant $0 \leq c \leq 1$. The threshold is lower for the chrominances, due to the higher sensitivity to errors in these channels.

Modifications

In addition to the algorithm, we have also implemented some extensions and changes to optimize the result for our purpose.

To make the method closer to a binary method, a mapping of α is made. In the original method α attains a value in the range $[0, 1-s]$, where s is the sensitivity parameter, and if it is in the middle of the higher and lower limit it is as likely to be a motion pixel as a non-motion pixel. Since a pixel actually is motion or not motion, we try a mapping of α to make more definite decisions on the pixel classification. Two different S-curve mappings are made, see (5.9) and (5.10).

$$\alpha_{new} = 3\alpha_{old}^2 - 2\alpha_{old}^3 \tag{5.9}$$

$$\alpha_{new} = 6\alpha_{old}^5 - 15\alpha_{old}^4 + 10\alpha_{old}^3 \tag{5.10}$$

The shape of these functions can be found in Figure 5.2.

To make the algorithm work properly, a new γ is calculated for each block size. The different block sizes are specified in a vector to make it more user adaptive. The optimal γ parameter varies with the standard deviation of the noise.

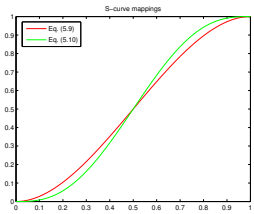


Figure 5.2: The two mappings of α in equations (5.9) and (5.10).

5.2.1 Algorithm summary

Algorithm 5.2.1: HIERARCHICAL BLOCK MATCHING ALGORITHM

1. Perform block matching on the images.
2. Calculate motion vectors and correlation between the matched blocks.
3. Filter in the motion vector direction, using the filtering weight.
4. Do colour correction.
5. Do post processing step.
6. Obtain output image.

5.3 Fuzzy logic algorithm

This implementation is based on the propositions made by Zlokolica et al. [25] and it takes the fuzzy logic approach to determine at what degree a pixel should be considered being in motion. The original algorithm includes a spatial filtering method as well. This part is completely independent from the temporal filtering and hence ignored by us.

The algorithm uses two frames from the image sequence, the previous processed frame, $I_f(t-1)$ and the new unprocessed frame, $I_n(t)$. In addition to the two images, it also needs the previous per-pixel noise variance estimates and filtering weights.

Computing the motion confidence

The first step is to compare the current noisy frame to the previous one using absolute differences. For each pixel a neighbourhood of 3×3 pixels, centred around the current pixel, is considered. The absolute difference matrix for this neighbourhood is computed and the results are used in a fuzzy rule that

determines the motion confidence $\theta(x, y, t)$, i.e the degree to which the pixel can be said to be a motion pixel. The absolute difference matrix for such a 2D neighbourhood, centred at position (x, y) at time t is defined as

$$\Delta(x, y, t) = |I_n(x + i, y + j, t) - I_f(x + i, y + j, t - 1)| \quad (i, j) \in \{-1, 0, 1\}. \tag{5.11}$$

To obtain the motion confidence, the whole neighbourhood is analysed. The degree of membership, $\mu_{large}(\Delta(x, y, t))$, is used in the fuzzy rule. All possible combinations of three neighbours out of the existing eight being large are evaluated, resulting in the motion confidence

$$\begin{aligned}\theta(x, y, t) = \mu_{large}(\Delta_{center})\ AND \\ \Big(\big(\mu_{large}(\Delta_1^1)\ AND\ \mu_{large}(\Delta_2^1)\ AND\ \mu_{large}(\Delta_3^1)\big)\ OR \\ \big(\mu_{large}(\Delta_1^2)\ AND\ \mu_{large}(\Delta_2^2)\ AND\ \mu_{large}(\Delta_3^2)\big)\ OR \\ \vdots \\ \big(\mu_{large}(\Delta_1^n)\ AND\ \mu_{large}(\Delta_2^n)\ AND\ \mu_{large}(\Delta_3^n)\big)\Big) \quad ,\end{aligned} \tag{5.12}$$

where Δ_d^c means the absolute difference for some neighbour $d \in \{1, 2, 3\}$ for some combination of neighbours $c \in 1 \ldots n$. In short, this means that to get a non-zero motion confidence, the current pixel and at least three neighbours must have non-zero μ_{large}. If there are more than three neighbours that do, all n possible combinations of those are evaluated. The maximum number of combinations to evaluate is then $n_{max} = \binom{8}{3} = 56$.

Calculation of parameters

The motion confidence $\theta(x, y, t)$ is used to compute the weights $\alpha(x, y, t)$ and a parameter, $\beta(x, y, t)$ that is used for the temporal adaptation of the σ-estimates. Specifically, the method needs to adapt to changes in noise and illumination. The membership function μ_{large} is defined as a piecewise linear function

$$\mu_{large}(x, a, b) = \begin{cases} 0 & 0 \leq x < a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ 1 & b < x \leq 1 \end{cases}, \tag{5.13}$$

The parameters a and b, that control the slope, are updated at each frame depending on the noise estimate from the previous frame. The update formulas are

$$a(x, y, t) = k_a \sigma(x, y, t - 1), \tag{5.14}$$

$$b(x,y,t) = k_b\sigma(x,y,t-1) + v(x,y,t), \tag{5.15}$$
$$\text{where } v(x,y,t) = k_{b\delta}\frac{\sigma(x,y,t-1)}{1+\delta(x,y,t)} - k_{f\delta}\frac{\delta(x,y,t)}{1+\sigma(x,y,t-1)},$$
$$\text{and } \delta(x,y,t) = \frac{1}{9}\sum\Delta(x,y,t).$$

$v(x,y,t)$ is a correction factor and $\delta(x,y,t)$ is a rough estimate of the variance of the 3×3-neighbourhood. The constants $k_a, k_b, k_{b\delta}, k_{f\delta}$ are found empirically in [25].

Once the motion confidence is determined, the weighting parameters can be computed. There is one parameter, β, that is used for updating σ and one parameter for the actual inter-frame weighting denoted α. The variance update formula is rather straightforward:

$$\sigma(x,y,t) = (1-\beta(x,y,t))\delta(x,y,t) + \beta(x,y,t)\sigma(x,y,t-1), \tag{5.16}$$

where $\beta(x,y,t) = min(1,\ 1.5\sqrt{\theta(x,y,t)})$ and $\delta(x,y,t)$ is the same as in equation 5.15. The variance update formula needs starting values. This starting value is simply set uniformly across the whole first frame, i.e. $\sigma(x,y,0) = \sigma_0 \quad \forall(x,y) \in I$.

The updating of the $\alpha(x,y,t)$-parameters is almost just as straightforward:

$$\alpha(x,y,t) = 0.5\alpha(x,y,t-1)^2 + (1-0.5\alpha(x,y,t-1))\alpha'(x,y,t), \tag{5.17}$$

where $\alpha'(x,y,t) = min(1,\ 1.15\sqrt{\theta(x,y,t)})$ is a rough initial estimate solely based on the motion confidence in the current frame. The initial value for $\alpha(x,y,t)$ is constant for all (x,y) at $t=0$. Different values will only affect the time it takes for the algorithm to settle in. In the long run it makes no difference.

Temporal filtering

Now, finally the new processed pixel value can be computed by interpolation between the previous filtered and the new current noisy pixel using the newly obtained $\alpha(x,y,t)$:

$$I_p(x,y,t) = \alpha(x,y,t)I_n(x,y,t) + (1-\alpha(x,y,t))I_p(x,y,t-1), \tag{5.18}$$

5.3.1 Modifications

Weight parameter, α

A variation of the proposed algorithm has also been tested. The modification affects the behaviour of the weighting parameter, α. This parameter can be tweaked to include more or less information from the previous frames. Instead

of using $\alpha(x, y, t-1)$ directly in (5.17), the mapping in (5.9) is used. Doing this shifts the α to depend more on the current motion estimate.

The function μ_{large} does not have to be a linear function. Experiments have also been done using (5.9).

Motion mask

A modification to how the motion mask is generated is also used. The modification uses scaling to suppress noise in the motion mask. It works by first downscaling and then upscaling the motion mask that the original Fuzzy logic algorithm produces. The scale factor used is 4. Also, the erosion/dilation technique has been used. The modifications to the motion mask does not affect the algorithm itself and the resulting modified mask can only used for binary filtering.

5.3.2 Algorithm summary

Algorithm 5.3.1: FUZZY LOGIC ALGORITHM

1. Update membership function.
2. Compute motion confidence for blocks.
3. Compute σ and α, save for next iteration.
4. Interpolate using new α.

5.4 Low-rank matrix completion algorithm

All algorithms described above are based on the assumption that the noise is Gaussian. In reality, that is not the case. Therefore there is a need for a noise robust algorithm which can handle other types of noise, e.g. the noise types described in section 4.2.

This algorithm, as described by Ji et al. [10], transforms the denoising problem to a low-rank matrix completion problem by keeping the pixels that are considered reliable, removing the pixels corrupted by noise, and thereafter concatenate columns of matched blocks to a matrix. This matrix is assumed to be a low-rank matrix since the matched blocks are supposed to be similar. The low-rank matrix problem is solved using a minimization algorithm. The motion detection is based on a fast block matching algorithm, which matches blocks both spatially and temporally. Temporally, the $a-1$ previous frames are used, and in each

frame b similar blocks are found. Thus $a \cdot b$ matching blocks are used for each block that is denoised.

Preprocessing

The preprocessing step accomplishes two things. First, the pixels corrupted by impulse noise, also called *salt-and-pepper* noise, are identified and removed. This is done by applying an adaptive median filter [8], on the images. In addition to identifying the impulse-corrupted pixels, this denoising is used as a preprocessing step for the block matching to obtain better matches. The actual denoising however, uses the unprocessed frames.

Patch matching

Let I_k be the k:th image in the sequence $I_{1\ldots a}$, where I_a is the image to be denoised. Each image can be described as $I_k = g_k + n_k$, where g_k is the clean image and n_k is the noise in image k.

To reduce the high complexity of the block matching, the algorithm proposed in [13] is used. As seen in section 4.3, this reduces the computational time complexity by a factor of 8.

The block matching is used to find b blocks in each of the a images used to denoise I_a. These 8×8 blocks are then stacked as columns in a matrix $P = (\mathbf{p}_1, \ldots \mathbf{p}_m)$, where $m = a \cdot b$ is the number of patches. Thus P is a $n^2 \times m$ matrix, where n is the block size. In our case $n = 8$.

Denoising a patch matrix

The unreliable pixels have to be found in order to create an incomplete matrix. These consist of two subsets, the pixels found by the adaptive median filter and the pixels that are too far from the mean value of each row in the patch-matched matrix P. The unreliable pixels are then replaced by 0, to form the incomplete matrix $P\,|_\Omega$. Indices of all reliable pixels form the set Ω.

Let Q denote the clean patch matrix from the image, i.e. $P = Q + N$, where N is the noise matrix for the patches. The goal is to recover Q from the incomplete observation $P\,|_\Omega$. To do this, a minimization problem is considered. Before the minimization problem is defined, a few notations will be introduced.

The Frobenious norm of a matrix X is defined as

$$\| X \|_F := \left(\sum_{i,j} | x_{i,j} |^2 \right)^{1/2}. \tag{5.19}$$

Furthermore, the nuclear norm of a matrix X is defined by

$$\| X \|_* := \sum_i \big(\sigma_i(X)\big), \tag{5.20}$$

where σ_i is the i:th largest singular value.

Let $X = U\Sigma V^T$ be the singular value decomposition, SVD, of X. The soft shrinkage operator $\mathcal{D}_\tau(X)$ is defined as

$$\mathcal{D}_\tau(X) = U\Sigma_\tau V^T, \tag{5.21}$$

where $\Sigma_\tau = diag\big(\max(\sigma_i - \tau, 0)\big)$, for a constant τ. Also, let $X \mid_\Omega$ denote the vector, concatenated from the matrix X, including the elements in the set Ω only.

The minimization problem, i.e. to recover Q from its incomplete observation $P \mid_\Omega$ using the fact that the rank of Q should be small is not itself a convex problem. Therefore the nuclear norm (5.20), which leads to a convex minimization problem, is used to approximate the rank of the matrix. Thus, we need to solve the following problem:

$$\begin{aligned} &\min_Q \| Q \|_* \qquad (5.22)\\ \text{s.t.} \quad &\| Q \mid_\Omega - P \mid_\Omega \|_F^2 \leq \#(\Omega)\hat{\sigma}^2, \end{aligned}$$

where $\#(\Omega)$ is the number of elements in the set Ω and $\hat{\sigma}$ is the estimation of the standard deviation of the unfiltered observations in Ω.

Instead of solving (5.22), an unconstrained Lagrangian version of the problem is derived.

$$\min_Q \frac{1}{2} \| Q \mid_\Omega - P \mid_\Omega \|_F^2 + \mu \| Q \|_*, \tag{5.23}$$

where μ is a constant s.t.

$$\mu = (\sqrt{n^2} + \sqrt{m})\sqrt{p}\hat{\sigma}, \tag{5.24}$$

where $p = \frac{\#(\Omega)}{mn^2}$, i.e. the ratio between the number of pixels in the set Ω and the total number of pixels in the patch matrix.

This problem is then solved by fixed point iteration.

Algorithm 5.4.1: FIXED POINT ITERATION

---

comment: Solve Minimization Problem (5.23)

let $Q^0 := 0$

while $\| Q^k - Q^{k-1} \|_F \leq \epsilon$

do $\begin{cases} R^k = Q^k - \tau\mathcal{P}_\Omega(Q^k - P) \\ Q^{k+1} = \mathcal{D}_{\tau\mu}(R^k) \\ k = k + 1 \end{cases}$

return $(Q := Q^k)$,

---

where μ and $\tau \in [1, 2]$ are predefined parameters, $\mathcal{D}$ is the shrinkage operator defined in (5.21) and $\mathcal{P}_\Omega$ is the projection operator of Ω defined as

$$\mathcal{P}_\Omega(Q)(i,j) = \begin{cases} Q(i,j) & \text{if } (i,j) \in \Omega \\ 0 & \text{otherwise} \end{cases} \tag{5.25}$$

Denoising the image

Now, a set of denoised patches for the image is available. Since an overlap of the blocks is used when performing the matching, the final pixel will get an average value of the corresponding pixel in all overlapping denoised blocks.

After this step, the denoised image is obtained.

The motion mask

It is not a straightforward task to extract a meaningful motion mask from this algorithm. The reason for that is the fact that filtering is done using multiple matching patches. This means that there is not a single unique motion vector to use. The way it is done now is that we look at all motion vectors from the current to the previous frame only. These motion vectors are then summed and if the length of the sum is less than a certain threshold the block is considered to not be in motion. The very nature of this algorithm makes it hard to compare the motion mask it produces to that of the other algorithms.

5.4.1 Algorithm summary

Algorithm 5.4.2: Low Rank Matrix Completion Algorithm

1. Filter the images using Adaptive Median Filter.
2. Match the images using a block matching algorithm.
3. Concatenate all columns for the blocks, and create a matrix from the matching blocks.
4. Identify unreliable pixels.
5. Solve the minimization problem to calculate the low rank matrix.
6. Average the values of each pixel in the denoised patches.

5.5 Blur Filtering

This is an algorithm that has developed during the course of this thesis. The algorithm is based on spatial smoothing of the input images and consists of two steps.

Similar to most other methods, the blur filtering method works with the previous filtered frame and the current unfiltered frame. First, the current frame is smoothed using a weighted average of the surrounding pixels. This spatially smoothed image is then used with the previous filtered image to create a difference frame that is then thresholded. This motion mask is then processed using a down-/upscaling technique.

Blurring

The spatial blurring amounts to convolving the image with a kernel containing the weights. There are many ways to choose the kernel, but to get the smoothing effect typically a symmetric kernel, with a relatively high weight in the center, is used. One example of such a kernel is

$$K = \begin{bmatrix} 1 & 3 & 1 \\ 3 & 9 & 3 \\ 1 & 3 & 1 \end{bmatrix}.$$

The same kernel is applied to both the current and the previous, processed frame.

The motion mask

The first step in creating the motion mask is simply differencing the blurred noisy frame and the blurred previous filtered frame. The absolute values are then thresholded. This however is only an intermediate mask that is potentially still very noisy. Finally, scaling of the mask is used to fill in holes, remove noise and smooth the edges. To scale the image, bilinear and bicubic scaling is used for down- and up scaling respectively.

Denoising

The actual denoising is performed by simple binary filtering using the newly obtained mask.

5.5.1 Algorithm summary

Algorithm 5.5.1: Blur Filtering Algorithm

1.Blur images.
2.Create difference frame.
3.Threshold to get motion mask.
4.Down-/upscale mask.
5.Perform binary filtering.

Chapter 6

Camera Implementation

Implementations on actual cameras are done using a simulation framework based on OpenCL. This framework runs on a computer but uses the video stream from a camera. OpenCL is a programming language based on the C99-standard. The purpose of the language is to provide a simple and effective way to run highly parallel workloads using highly parallel hardware such as graphics processors, *GPU:s*. The syntax is very similar to C, with some parallelism-specific additions.

6.1 Camera calibration

When detecting scene-change in real time on actual cameras, performance can often be improved by calibrating the camera. In this case that means analysing the noise at different levels of illumination. The output is the standard deviation of the noise as a function of the signal level, which is linear with respect to the intensity of the incoming light. The calibration is mainly used to adjust thresholds in the detection algorithms.

In order to calculate the standard deviation of the noise, a completely static and evenly illuminated scene is required. This is accomplished in a lab by using the inside of a sphere that is lit by a precisely controlled lamp. The first step is to acquire image sequences at several different levels of illumination, ranging from black to the point when the sensor is saturated, i.e. when everything is white. From each sequence, a series of difference images is created. The only thing that remains in the difference images is the noise, since the scene itself does not change. Now, computing the standard deviation of the noise simply amounts to computing the standard deviation of the difference images and dividing by $\sqrt{2}$, [9]. Having many noise estimates means that they can be averaged to get a more accurate result. The factor $\frac{1}{\sqrt{2}}$ is derived as

$$\begin{aligned}\sigma_{diff}^2 &= E\big(((S+F+X)-(S+F+Y))^2\big)- \\ &\quad E\big((S+F+X)-(S+F+Y)\big)^2 = \\ &= E((X-Y)^2) - E(X-Y)^2 = \cdots = \sigma_X^2 + \sigma_Y^2 = \\ &= 2\sigma_{noise}^2 \\ \Leftrightarrow \sigma_{noise} &= \frac{\sigma_{diff}}{\sqrt{2}},\end{aligned} \tag{6.1}$$

where σ_{diff}^2 is the standard deviation of the difference frame. The signal S and the fixed pattern noise F are both constant, which means that they will cancel out in the difference image. The random noise X and Y from the two frames used to construct the difference frame is all that remains. The characteristics of the noise, X and Y, does not change between frames which is why $\sigma_X = \sigma_Y = \sigma_{noise}$.

All algorithms work on demosaiced images and not directly on Bayer data. Because of this the calibration has been performed on the demosaiced images. This means that the result is not strictly the correlation between the signal and the standard deviation. Instead it is the dependence of the standard deviation of the noise in the grey levels of the demosaiced input data. Using the data points, a polynomial can be fitted to approximate the noise level which then can be used to adjust thresholds in the algorithms.

6.2 Implementations

Out of the five evaluated methods, three were chosen for OpenCL implementation: *Fuzzy Logic*, *Adaptive Fuzzy Filtering* and *Hierarchical block matching*. They were chosen because we felt that they are the most promising, and also because they are not overly complex. The generally high complexity of the low-rank method, and in particular the need for SVD, make it unreasonable to implement in OpenCL within the timespan of this thesis.

Chapter 7

Results

7.1 Result overview

The methods have been tested on several image sequences, to find where the methods perform well and where they give less satisfactory results. Both existing image sequences from surveillance cameras and artificial, computer animated, sequences with noise added have been used to evaluate the methods. For the sequences where noise has been added, a comparison between the original noise-free frame and the filtered frame can be made.

A motion mask is created for all methods. These method-generated masks are compared to the reference mask when one exists, i.e. when using the synthetic sequences. The number of true positives, true negatives, false positives and false negatives are measured to determine how similar the motion mask for the method is to the real motion mask. These statistics are then used to compute the *TPR*, *FPR* and *accuracy*.

Moreover, the performance of the different methods is measured using structural similarities, SSIM.

The statistics alone are not enough to evaluate the methods, since human visual system aspects are not included. Therefore, we have also included a subjective measurement in order to differ and compare the methods. The subjective measurement is only a qualitative measurement.

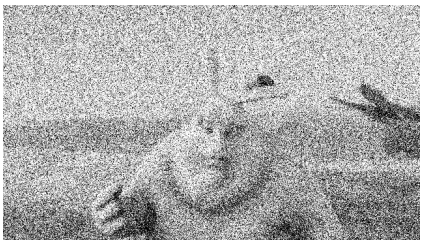
The parameters for each method is optimised for each sequence, thus generating the best possible results in all cases. This ought to give the most fair comparison.

Mainly, the three datasets described in Section 3.2 have been used to test the methods. One is a sequence of a bus passing on a road. This is captured in low-light conditions, which makes the sequence quite noisy. A few of the challenges in this sequence is to preserve the arrow pattern on the bus, where the contrast

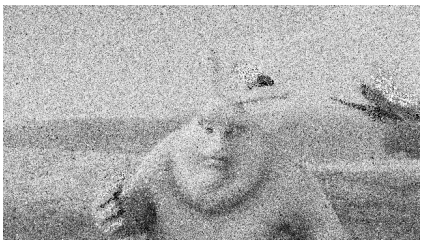
is low, keep the text on the bus clear and readable and avoid ghosting artifacts where motion has occurred.

The other sequences are taken from the computer animated movie *Big Buck Bunny* [5]. The sequences are chosen so that they include static backgrounds, many details and contain both small and big movements. Noise, here a zero-mean normal distribution, is added to the sequences. Several different standard deviations are used to test the methods on various levels of noise. All images, when presented here, are 8 bits per channel meaning that the range of the colours is $[0, 255]$.

For binary filtering, different weights can be used for the old filtered frame and the new noisy frame, for pixels which are classified as non-motion pixels. When a higher weight is set to to the old filtered image, the noise reduction becomes much higher for static scenes after some start-up time. However, there are some issues with this approach. One issue is that if a motion pixel is classified incorrectly as a non-motion pixel, the error will propagate and will be seen much longer. Another problem is that the areas where motion is detected become extremely noisy compared to the filtered static areas. This results in that motion areas look like artifacts or ghosting due to the very different appearance between the two areas. In an image processing pipeline, where the image is processed and noise is removed, several denoising steps are performed. To solve the problem described above, spatial filtering in these areas is performed. Spatial filtering is not included in this thesis, and therefore the resulting filtered images can look strange. In Figure 7.1, two images filtered with the same method but using different weights between the previous filtered frame and the new noisy frame are shown. Here, one image is filtered using equal weights, e.g. 0.5, for both images. The other image use the weights 0.8 and 0.2 for the old filtered frame and the new noisy frame respectively. Filtering only occurs where no motion is detected.



(a) Original noisy image



(b) Filtering $I_f(x,y,t) = 0.5 \cdot I_f(x,y,t-1) + 0.5 \cdot I_n(x,y,t)$ for non-motion pixels

(c) Filtering $I_f(x,y,t) = 0.8 \cdot I_f(x,y,t-1) + 0.2 \cdot I_n(x,y,t)$ for non-motion pixels

Figure 7.1: One image from the Big Buck Bunny sequence filtered using different weights in the binary filtering. The added noise has standard deviation $\sigma = 100$.

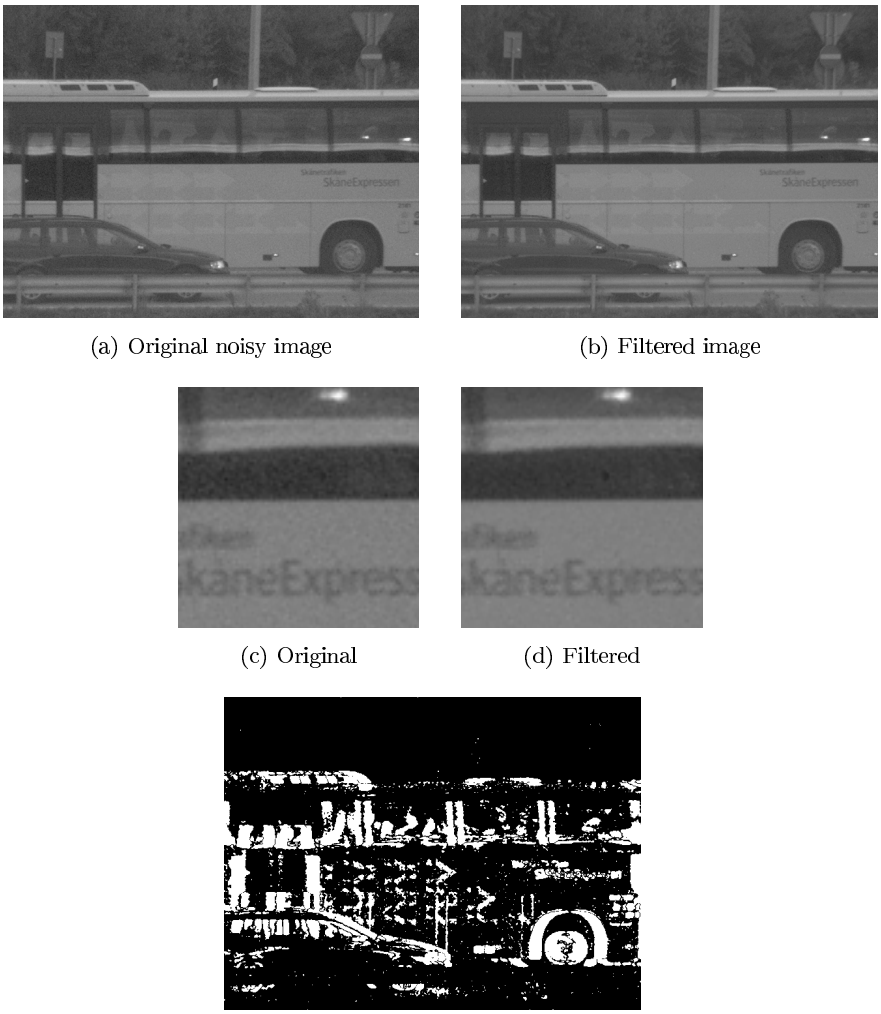
7.2 Adaptive fuzzy filter algorithm

The Adaptive fuzzy filter algorithm appears to be one of the most efficient and fastest algorithms of the ones that have been implemented, it also performs well for all image sequences.

The bus sequence

The motion mask generated here is very good. There are some pixels, classified as motion that probably should have been classified as non-motion, but it is hard, even manually, to determine if the trees in the background are moving or not.

The algorithm finds both the low contrast arrows and the small details, such as the text on the bus, and very little artifacts are introduced, see Figure 7.2.



(a) Original noisy image

(b) Filtered image

(c) Original

(d) Filtered

(e) Motion mask

Figure 7.2: Frame 9 from the bus sequence and corresponding filtered image and motion mask using the adaptive fuzzy filter algorithm.

The Big Buck Bunny sequences

The adaptive fuzzy filter algorithm detects the bunny in the sequences fairly well. The higher the standard deviation of the noise is, the less movement the method detects. For a sequence where noise with standard deviation $\sigma = 20$ is added, see an example of an image in Figure 7.3, the method detects most of the major movements, but when the standard deviation of the noise increases to $\sigma = 40$, much less is detected unless the threshold for the motion detection is lowered. With a lower threshold, the motion mask looks noisier. An example image where standard deviation $\sigma = 40$ has been used can be seen in Figure

7.5. The statistics for the sequence using the original adaptive fuzzy filter with $\sigma = 20$, $\sigma = 40$ and $\sigma = 100$ can be seen in Figure 7.4, 7.6 and 7.8.

(a) Original image

(b) Original noisy image

(c) Filtered image

(d) Original

(e) Original noisy

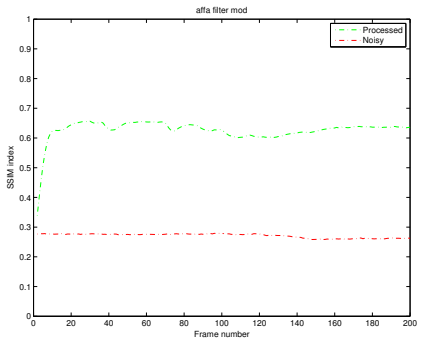
(f) Filtered

(g) Reference motion mask

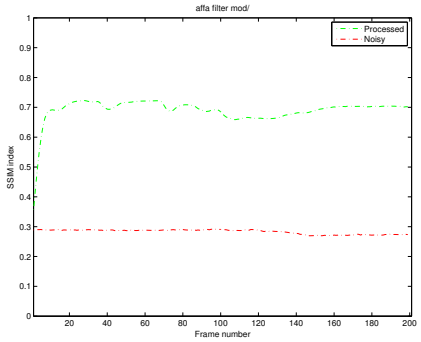
(h) Motion mask

Figure 7.3: Frame 2323 from the Big Buck Bunny sequence filtered using adaptive fuzzy filter algorithm. The added noise has standard deviation $\sigma = 20$.

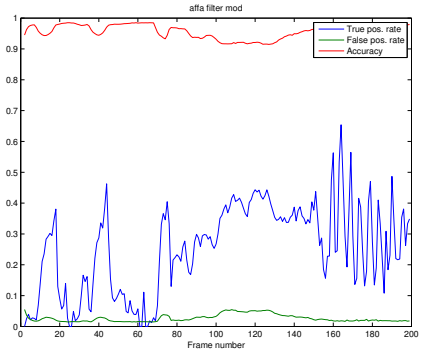
When the SNR is low, the adaptive fuzzy filter algorithm runs into some issues. The number of false positives tends to follow the true positives a bit too closely. It either detects more motion, but also classifies slightly more noise as motion, or when trying to avoid classifying the noise as motion, it does not detect enough



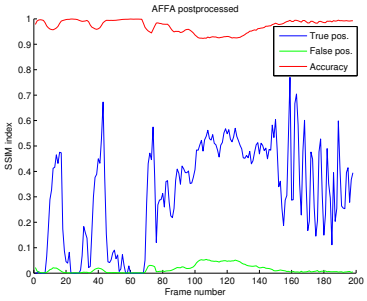
(a) SSIM for the original adaptive fuzzy filter method.



(b) SSIM for the post processed adaptive fuzzy filter method.



(c) Detection statistics for the original adaptive fuzzy filter method.



(d) Detection statistics for the post processed adaptive fuzzy filter method.

Figure 7.4: Statistics from one of the sequences from the Big Buck Bunny movie using the original and modified adaptive fuzzy filter algorithm. The added noise has standard deviation $\sigma = 20$.

motion. The difference is quite small, but might matter in some cases. For an example of an image with $SNR \approx 1.5$ see Figure 7.7. When the image is as noisy as in that case, where the standard deviation is $\sigma = 100$, the filtering can sometimes look very strange. This is because the difference between the filtered parts of the image and the non-filtered parts, i.e. the parts where motion is detected, is very big. Therefore it can look like the filtering has created ghosting effects and artifacts in the image in areas that are simply unfiltered. This happens when there are unfiltered "islands" surrounded by filtered areas. For an example, take a close look at the butterfly in Figure 7.9. The wings of the butterfly are moving, and one can clearly see where the wings have moved from the previous image in the sequence since no filtering is performed there.

Since some of the motion masks look noisy, the erosion/dilation method is applied. First, dilation with a disk of radius $r = 2$ is performed, thereafter an erosion of radius $r = 3$ is applied, and at last one more dilation is used. This

(a) Original image

(b) Original noisy image

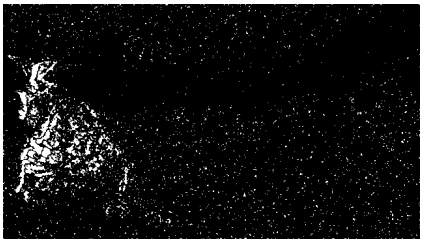
(c) Filtered image

(d) Original

(e) Original noisy

(f) Filtered

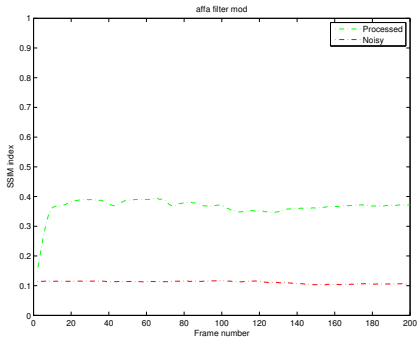
(g) Reference motion mask



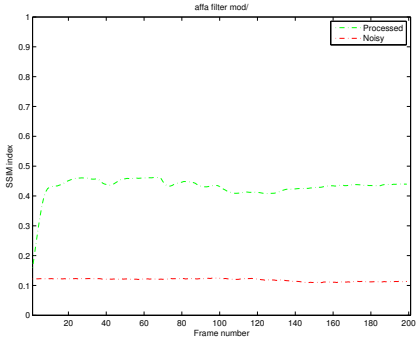
(h) Motion mask

Figure 7.5: Frame 2323 from the Big Buck Bunny sequence filtered using adaptive fuzzy filter algorithm. The added noise has standard deviation $\sigma = 40$.

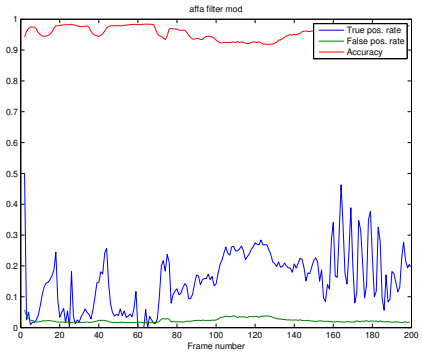
time the structuring element of the dilation is a disk with radius $r = 1$. Some results can be seen in Figure 7.10.



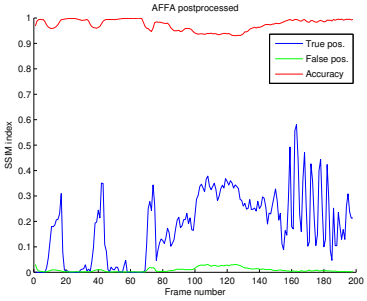
(a) SSIM for the original adaptive fuzzy filter method.



(b) SSIM for the post processed adaptive fuzzy filter method.



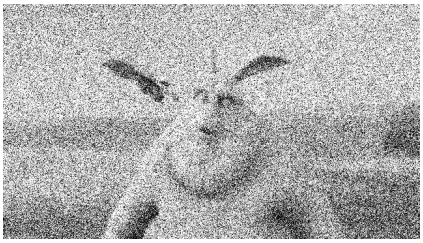
(c) Detection statistics for the original adaptive fuzzy filter method.



(d) Detection statistics for the post processed adaptive fuzzy filter method.

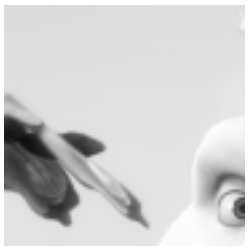
Figure 7.6: Statistics from one of the sequences from the Big Buck Bunny movie using the original and modified adaptive fuzzy filter algorithm. The added noise has standard deviation $\sigma = 40$.

(a) Original image

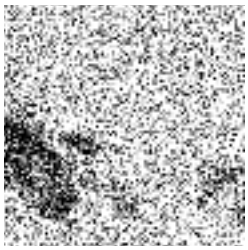


(b) Original noisy image

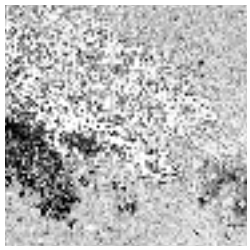
(c) Filtered image



(d) Original

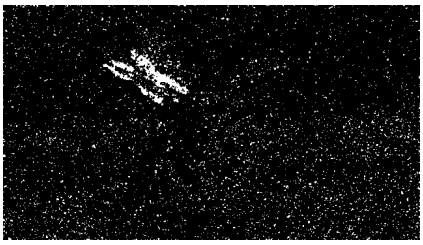


(e) Original noisy



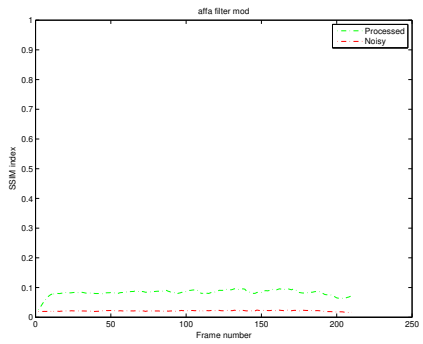
(f) Filtered

(g) Reference motion mask

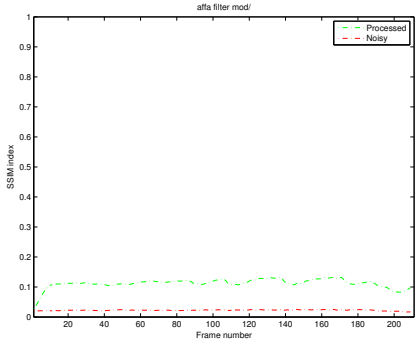


(h) Motion mask

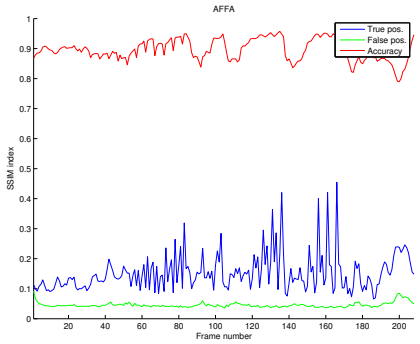
Figure 7.7: Frame 2597 from the Big Buck Bunny sequence filtered using adaptive fuzzy filter algorithm. The added noise has standard deviation $\sigma = 100$.



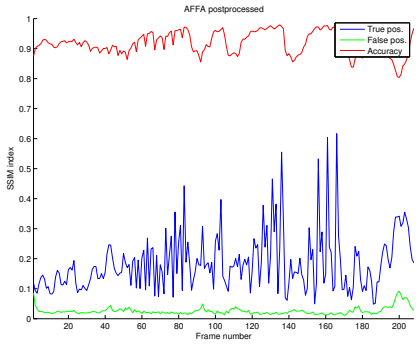
(a) SSIM for the original adaptive fuzzy filter method.



(b) SSIM for the post processed adaptive fuzzy filter method.



(c) Detection statistics for the original adaptive fuzzy filter method.



(d) Detection statistics for the post processed adaptive fuzzy filter method.

Figure 7.8: Statistics from one of the sequences from the Big Buck Bunny movie using the original and modified adaptive fuzzy filter algorithm. The added noise has standard deviation $\sigma = 100$.

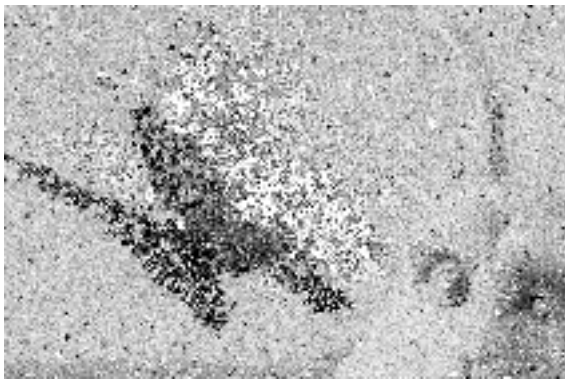
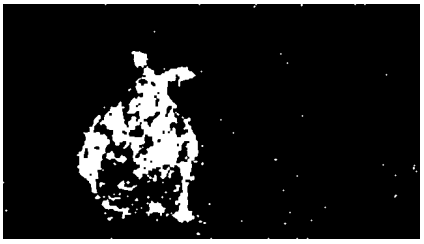


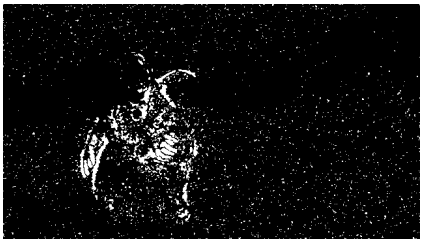
Figure 7.9: An image filtered with the adaptive fuzzy filtering algorithm. The butterfly has moved and that part is therefore unfiltered. Since the image is very noisy, the unfiltered parts around the butterfly look like artifacts. The added noise has standard deviation $\sigma = 100$.

(a) Reference motion mask

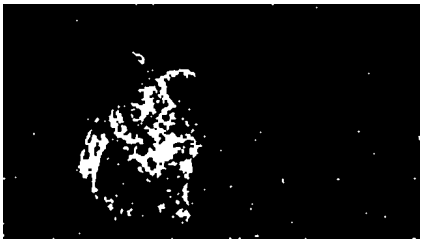
(b) Motion mask $\sigma = 20$



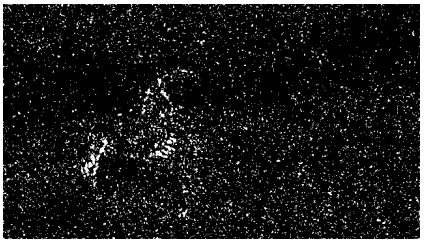
(c) Modified motion mask $\sigma = 20$



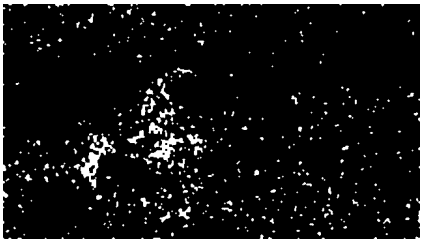
(d) Motion mask $\sigma = 40$



(e) Modified motion mask $\sigma = 40$



(f) Motion mask $\sigma = 100$



(g) Modified motion mask $\sigma = 100$

Figure 7.10: Motion masks for the adaptive fuzzy filtering algorithm and modified masks using dilation and erosion for noise with different standard deviation σ.

7.3 Hierarchical block-matching algorithm

The hierarchical block-matching algorithm performs, according to our subjective observations, better using our suggested mapping of α from eq. (5.9). In all tests this remapping of the filtering coefficient α has been used. The difference when using the mapping in eq. (5.10) is very small. The block sizes are varied to see how much a bigger search area can affect the result. This is done because many movements are bigger than 5 pixels.

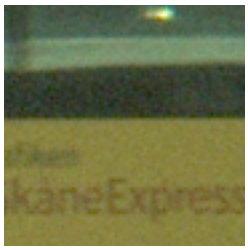
The bus sequence

Using a search window of 11×11 and block sizes 9×9 and 7×7 and another search window of 25×25 with block sizes 13×13, 9×9 and 7×7, the filtered images can be seen in Figure 7.11 together with the motion masks and the original noisy image.

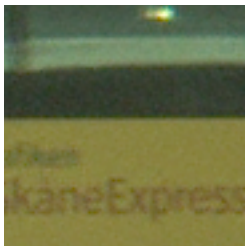
(a) Original noisy image

(b) Filtered, search window 11×11

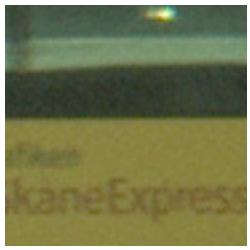
(c) Filtered, search window 25×25



(d) Original

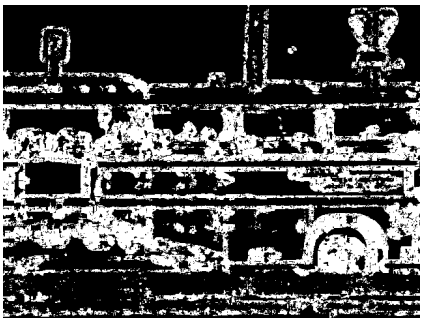


(e) Filtered, 11×11



(f) Filtered, 25×25

(g) Motion mask, search window 11×11



(h) Motion mask, search window 25×25

Figure 7.11: An image from the bus sequence and corresponding filtered images and motion masks using hierarchical block-matching algorithm with search windows 11×11 and 25×25 and block sizes 9×9, 7×7 and 13×13, 9×9, 7×7 respectively.

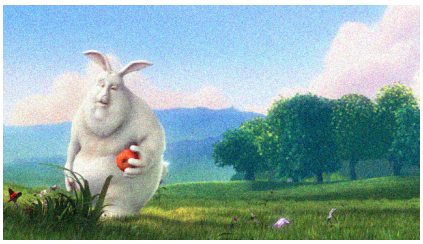
Big Buck Bunny sequences

One of the images from one of the sequences from the Big Buck Bunny movie can be seen in Figure 7.12. This figure shows the original image, the image with added noise with standard deviation $\sigma = 20$, the filtered image for two different search window sizes, and also the motion mask. The whole sequence consists of 200 images, and the statistics are shown in Figure 7.13.

When the noise level is higher, the algorithm has more problems with differentiating motion from noise. This can be seen in Figure 7.14. Also the statistics from the sequence can be viewed in Figure 7.15.

The behaviour of the algorithms when the SNR is low have to be investigated. Here, by adding noise of standard deviation $\sigma = 100$ for one of the sequences from the Big Buck Bunny movie we obtain a signal-to-noise ratio of approximately $SNR \approx 1.5$. For the hierarchical block-matching algorithm with the same search windows and block-sizes as above, the result is shown in Figure 7.16. The statistics are shown in 7.17.

(a) Original image



(b) Original noisy image

(c) Filtered image, search window 11×11

(d) Filtered image, search window 25×25

(e) Original

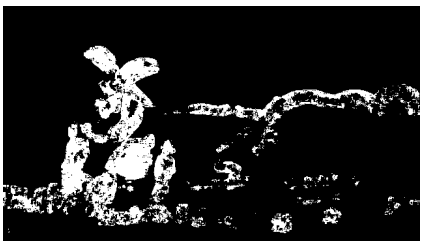
(f) Original noisy

(g) Filtered, 11×11

(h) Filtered, 25×25

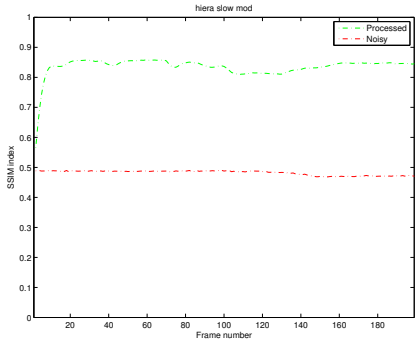
(i) Reference motion mask

(j) Motion mask, search window 11×11

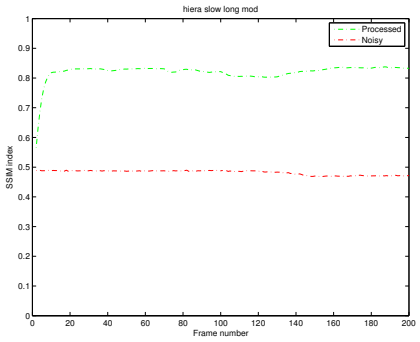


(k) Motion mask, search window 25×25

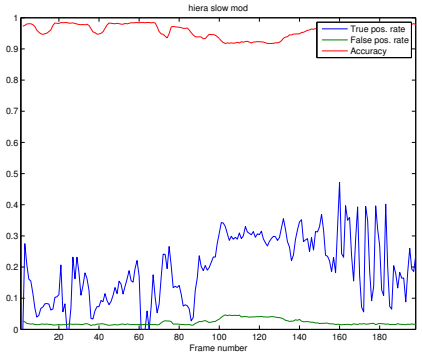
Figure 7.12: Frame 2309 from the Big Buck Bunny sequence filtered using the hierarchical block-matching method for search windows 11×11 and 25×25 with block sizes $9 \to 7$ and $13 \to 9 \to 7$ respectively. The noise added has a standard deviation of $\sigma = 20$.



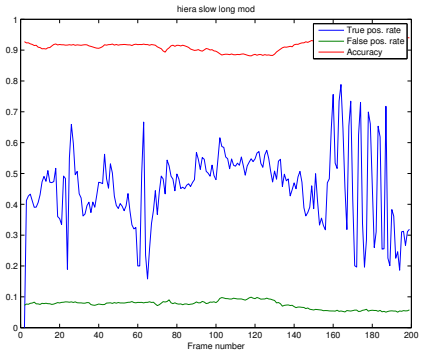
(a) SSIM for the original image and the filtered image for search window 11×11



(b) SSIM for the original image and the filtered image for search window 25×25



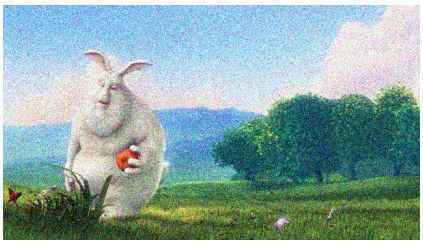
(c) Detection statistics for search window 11×11



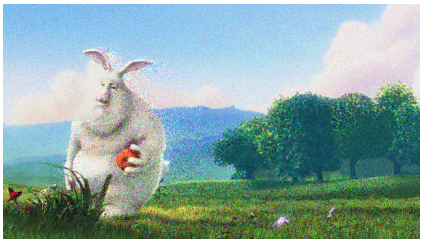
(d) Detection statistics for search window 25×25

Figure 7.13: Statistics from one of the sequences from the Big Buck Bunny movie, where the added noise has standard deviation $\sigma = 20$. The sequence is run through the hierarchical block-matching algorithm with search windows 11×11 and 25×25 and with block sizes $9 \to 7$ and $13 \to 9 \to 7$ respectively.

(a) Original image



(b) Original noisy image



(c) Filtered image, search window11 × 11

(d) Filtered image,search window 25 × 25

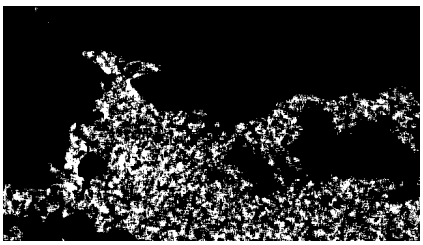
(e) Original

(f) Original noisy

(g) Filtered, 11 × 11

(h) Filtered, 25 × 25

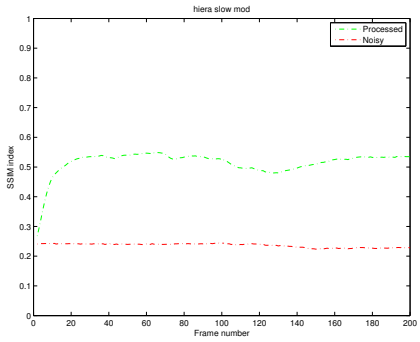
(i) Reference motion mask



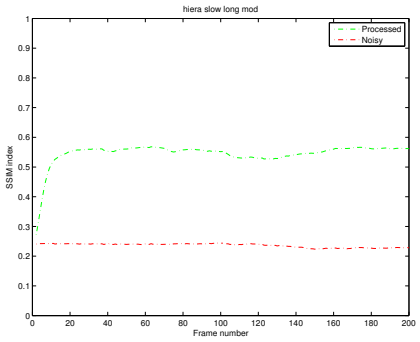
(j) Motion mask, search window 11 × 11

(k) Motion mask,search window25 × 25

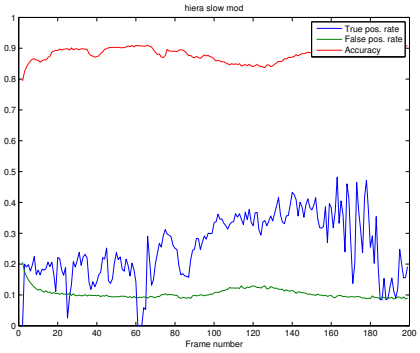
Figure 7.14: Frame 2309 from the Big Buck Bunny sequence filtered using the hierarchical block-matching method for search windows 11×11 and 25×25 with block sizes $9 \rightarrow 7$ and $13 \rightarrow 9 \rightarrow 7$ respectively. The noise added has a standard deviation of $\sigma = 40$.



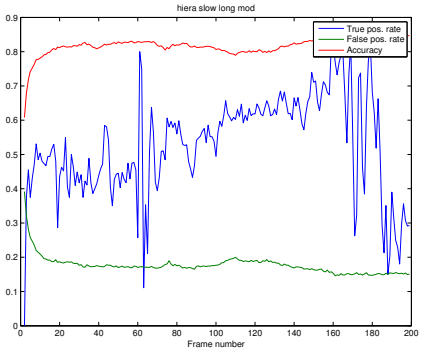
(a) SSIM for the original image and the filtered image for search window 11×11



(b) SSIM for the original image and the filtered image for search window 25×25



(c) Detection statistics for search window 11×11



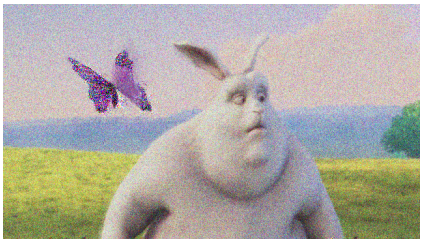
(d) Detection statistics for search window 25×25

Figure 7.15: Statistics from one of the sequences from the Big Buck Bunny movie, where the added noise has standard deviation $\sigma = 40$. The sequence is run through the hierarchical block-matching algorithm with search windows 11×11 and 25×25 and with block sizes $9 \to 7$ and $13 \to 9 \to 7$ respectively.

(a) Original image

(b) Original noisy image

(c) Filtered image, search window11 × 11



(d) Filtered image,search window 25 × 25

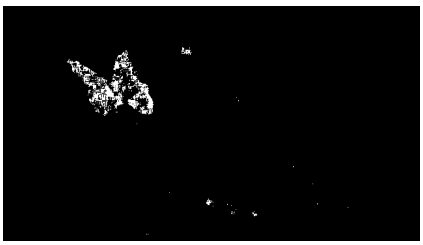
(e) Original

(f) Original noisy

(g) Filtered, 11 × 11

(h) Filtered, 25 × 25

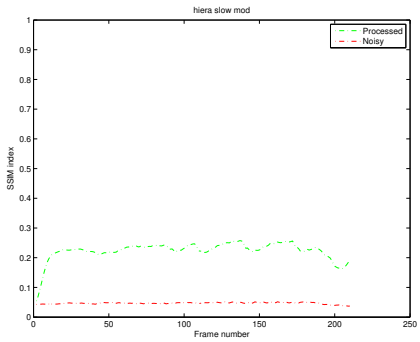
(i) Reference motion mask



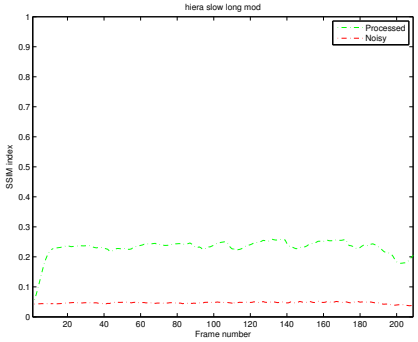
(j) Motion mask, search window 11 × 11

(k) Motion mask,search window 25 × 25

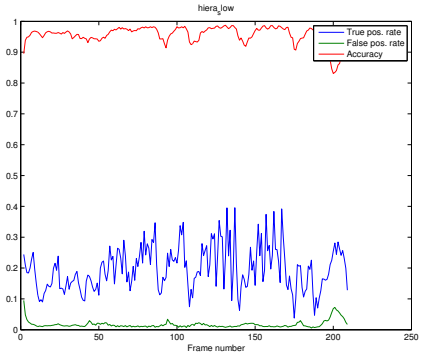
Figure 7.16: Frame 2553 from the Big Buck Bunny sequence filtered using the hierarchical block-matching method for search windows 11 × 11 and 25 × 25 with block sizes 9 → 7 and 13 → 9 → 7 respectively. The noise added has a standard deviation of $\sigma = 100$.



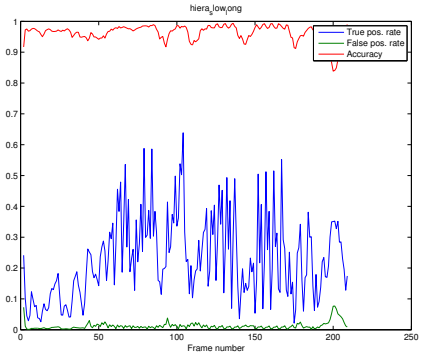
(a) SSIM for the original image and the filtered image for search window 11×11



(b) SSIM for the original image and the filtered image for search window 25×25



(c) Detection statistics for search window 11×11



(d) Detection statistics for search window 25×25

Figure 7.17: Statistics from one of the sequences from the Big Buck Bunny movie, where the added noise has standard deviation $\sigma = 100$. The sequence is run through the hierarchical block-matching algorithm with search windows 11×11 and 25×25 and with block sizes $9 \to 7$ and $13 \to 9 \to 7$ respectively.

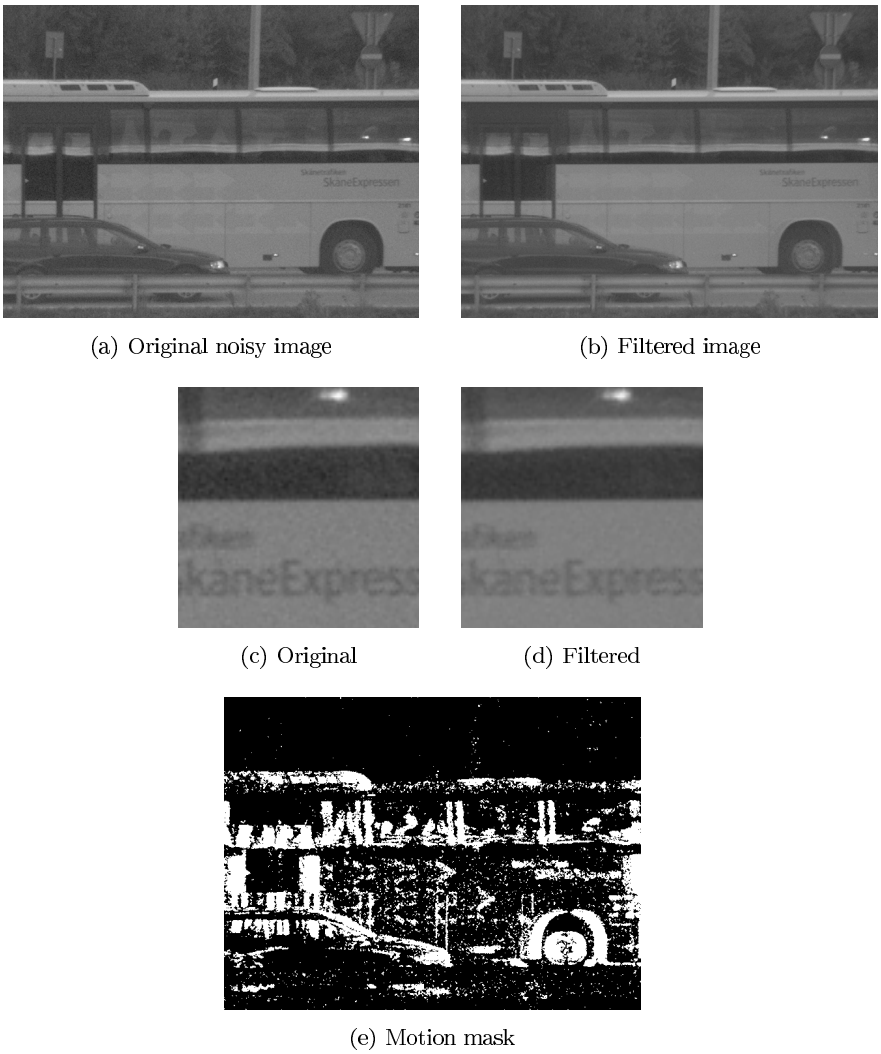
7.4 Fuzzy logic algorithm

The fuzzy logic algorithm tends to produce very noisy motion masks. A way of reducing the noise is to use dilation and erosion. The combination used here is this: first dilate with radius $r = 2$, then erode with $r = 3$ and finally dilate with $r = 1$. The structuring element is a disk. This approach is used since the first dilation clusters unconnected pixels that are close enough. Then the erosion of $r = 3$ removes single pixels, which often are noise pixels. The new eroded/dilated motion mask can now be used for binary filtering. The filtering in the original fuzzy logic algorithm is not binary and does not depend on the motion mask, rather the motion mask is a by-product of the algorithm. The results can be seen for the different sequences under the respective sections.

Another way of reducing the noise in the motion mask is to use a down-/upscaling technique.

Bus sequence

The fuzzy logic algorithm seems to detect the arrows with low contrast partly, but not enough to avoid blurring. This can be seen in Figure 7.18.



(a) Original noisy image

(b) Filtered image

(c) Original

(d) Filtered

(e) Motion mask

Figure 7.18: An image from the bus sequence and corresponding filtered image and motion mask using fuzzy logic filtering algorithm.

The Big Buck Bunny sequences

In the Big Buck Bunny sequences, the fuzzy logic algorithm, in addition to the real motion pixels, detects much noise. The erosion/dilation technique is beneficial when the noise level is not extremely high, but at a mean SNR of approximately 1.5, the noise is quite "dense", meaning that the erosion/dilation will cluster the noise as in Figure 7.23. For higher SNR, as in Figures 7.19 and 7.21, the erosion/dilation improves the motion mask. Also, the filtered image resulting from binary filtering using the new motion mask, has less ghosting effects compared to the filtered image from the original fuzzy logic filtering

algorithm.

The SSIM values and detection statistics from the motion masks for the Big Buck Bunny sequence with $\sigma = 20$, $\sigma = 40$ and $\sigma = 100$ can be seen in Figure 7.20, 7.22 and 7.24 respectively.

In Figure 7.25 the difference in filtering when using the faster-decaying α-modification is seen. When suppressing the influence of the older weight, there is less ghosting but also more remaining noise. The masks from the two methods are also shown along with the motion mask after down-/upscaling.

(a) Original image

(b) Original noisy image

(c) Filtered, orignal algo

(d) Filtered, binary

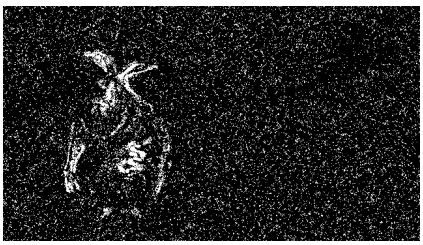
(e) Original

(f) Original noisy

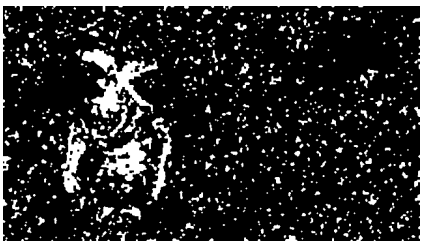
(g) Filtered, original algo

(h) Filtered, binary

(i) Reference motion mask

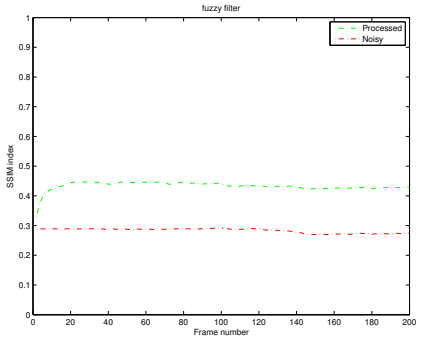


(j) Motion mask, orignal

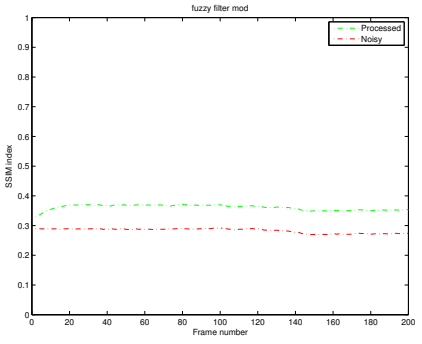


(k) Motion mask, binary after erosion/dilation

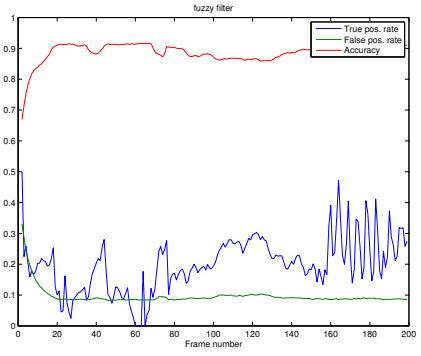
Figure 7.19: Frame 2309 from the Big Buck Bunny sequence filtered using the fuzzy logic algorithm. Both the original filtering and the binary filtering from the mask created using erosion and dilation are included. The noise added has a standard deviation of $\sigma = 20$.



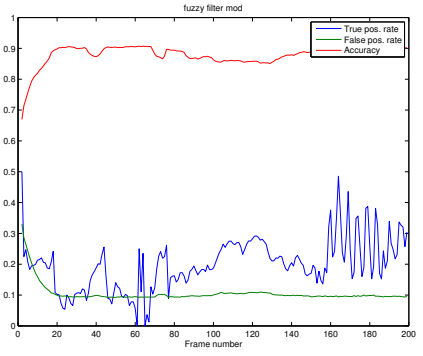
(a) SSIM for the filtered image using original fuzzy logic algorithm and noisy image.



(b) SSIM for the filtered image using modified fuzzy logic algorithm and noisy image.



(c) Detection statistics for original fuzzy logic algorithm.



(d) Detection statistics for modified fuzzy logic algorithm.

Figure 7.20: Statistics from one of the sequences from the Big Buck Bunny movie, where the added noise has standard deviation $\sigma = 20$. The statistics are generated from the original fuzzy logic algorithm and the modified fuzzy logic algorithm.

(a) Original image

(b) Original noisy image

(c) Filtered, orignal algo

(d) Filtered, binary

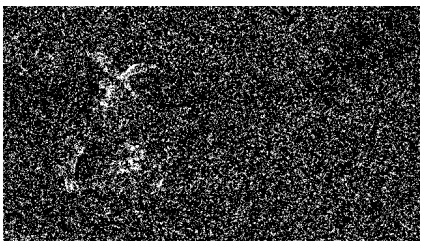
(e) Original

(f) Original noisy

(g) Filtered, original algo

(h) Filtered, binary

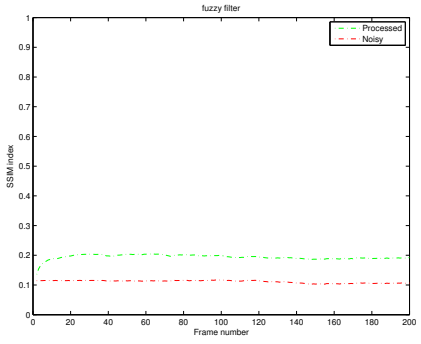
(i) Reference motion mask



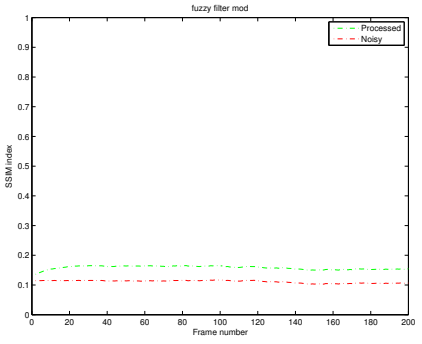
(j) Motion mask, orignal

(k) Motion mask, binary after erosion/dilation

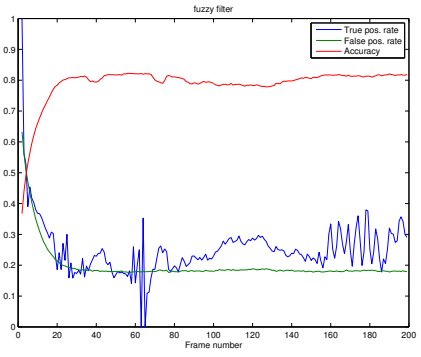
Figure 7.21: Frame 2309 from the Big Buck Bunny sequence filtered using the fuzzy logic algorithm. Both the original filtering and the binary filtering from the mask created using erosion and dilation are included. The noise added has a standard deviation of $\sigma = 40$.



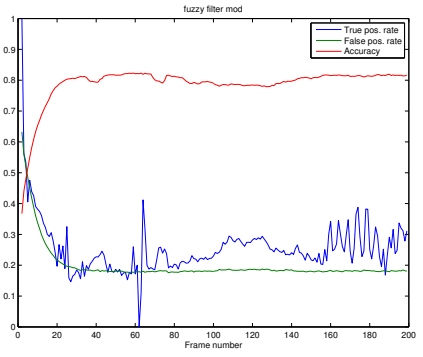
(a) SSIM for the filtered image using original fuzzy logic algorithm and noisy image.



(b) SSIM for the filtered image using modified fuzzy logic algorithm and noisy image.



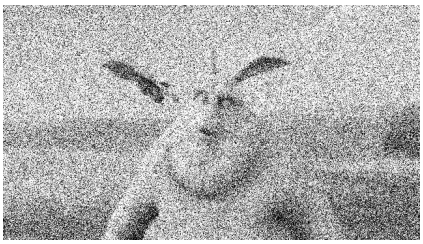
(c) Detection statistics for original fuzzy logic algorithm.



(d) Detection statistics for modified fuzzy logic algorithm.

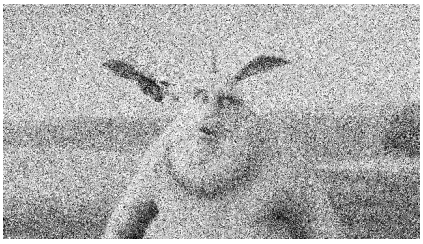
Figure 7.22: Statistics from one of the sequences from the Big Buck Bunny movie, where the added noise has standard deviation $\sigma = 40$. The statistics are generated from the original fuzzy logic algorithm and the modified fuzzy logic algorithm.

(a) Original image

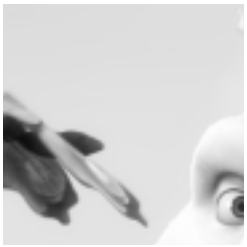


(b) Original noisy image

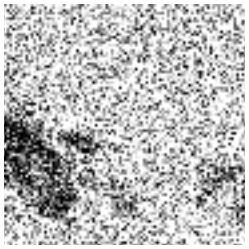
(c) Filtered, orignal algo



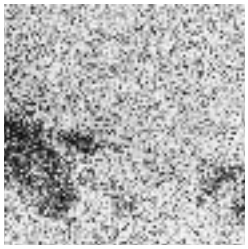
(d) Filtered, binary



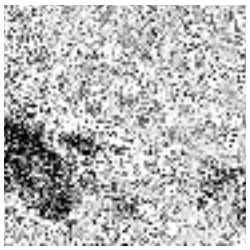
(e) Original



(f) Original noisy

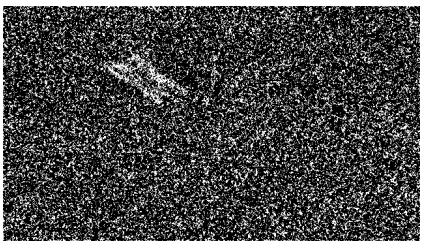


(g) Filtered, original algo



(h) Filtered, binary

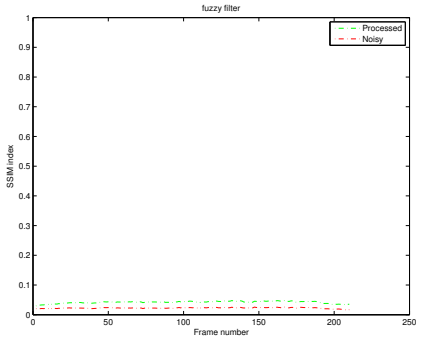
(i) Reference motion mask



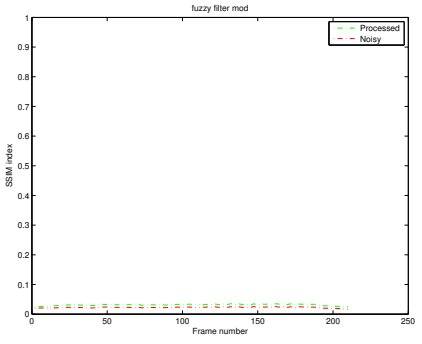
(j) Motion mask, orignal

(k) Motion mask, binary after erosion/dilation

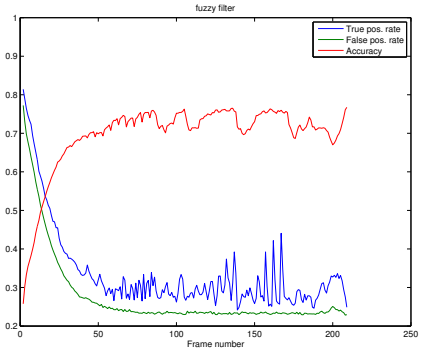
Figure 7.23: Frame 2597 from the Big Buck Bunny sequence filtered using the fuzzy logic algorithm. Both the results from the original filtering and the binary filtering from the mask created using erosion/dilation are shown. The added noise has a standard deviation of $\sigma = 100$.



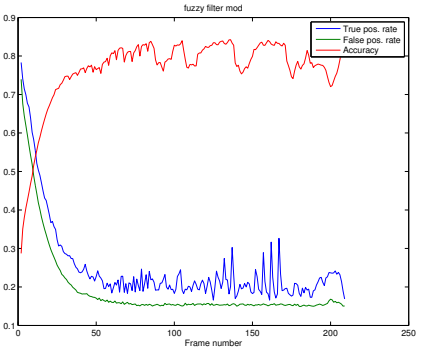
(a) SSIM for the filtered image using original fuzzy logic algorithm and noisy image.



(b) SSIM for the filtered image using modified fuzzy logic algorithm and noisy image.



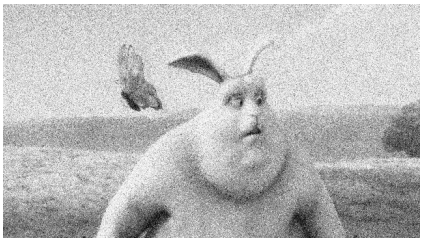
(c) Detection statistics for original fuzzy logic algorithm.



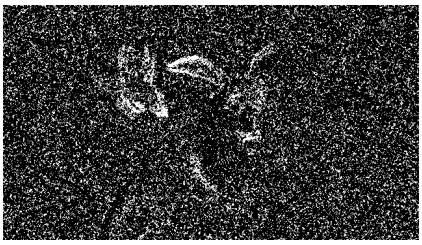
(d) Detection statistics for modified fuzzy logic algorithm.

Figure 7.24: Statistics from one of the sequences from the Big Buck Bunny movie, where the added noise has standard deviation $\sigma = 100$. The statistics are generated from the original fuzzy logic algorithm and the modified fuzzy logic algorithm.

(a) Filtered, original algo

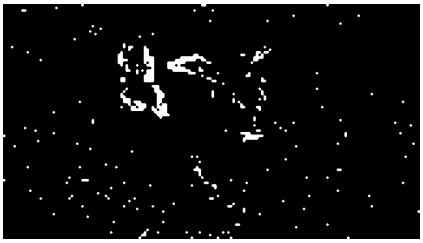


(b) Filtered, α-modification



(c) Mask, original algo

(d) Mask, α-modification



(e) Mask, scaling-modification (also α-mod.)

Figure 7.25: One image from the Big Buck Bunny sequence filtered using the fuzzy logic algorithm. The masks from the original and modified methods are shown as well as the mask created using down-/upscaling. The noise added has a standard deviation of $\sigma = 40$.

7.5 Low-rank matrix completion algorithm

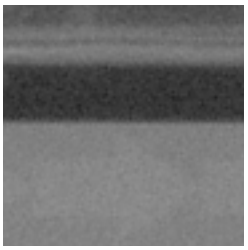
This algorithm is very time-consuming but it seems to work fairly well in a variety of cases. No motion masks are provided here, simply because of the way the algorithm works. There is no way to determine the motion status of a single-pixel, only blocks can be classified as motion or non-motion. This is not the biggest problem with the motion mask however. Since the algorithm always finds several, more or less accurate, matching blocks there is no clear way to extract a motion mask at all.

Bus sequence

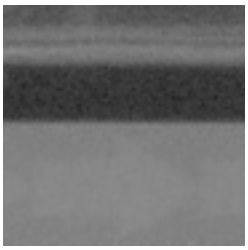
The results from the bus sequence can be found in Figure 7.26. In some areas, where motion has occurred, for example the arrows on the bus and the car passing the bus, almost all noise is suppressed and no artifacts can be seen. In the rest of the image, however, the noise is still present and not reduced by any noticeable amount. This strange phenomenon can be seen clearly in Figure 7.26(d).

(a) Original noisy image

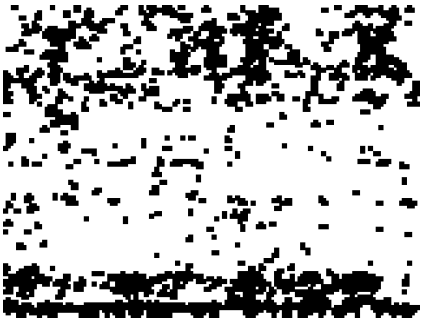
(b) Filtered image



(c) Original



(d) Filtered



(e) Motion mask

Figure 7.26: An image from the bus sequence and corresponding filtered image and motion mask using low-rank matrix completion with 35 images and 2 matches in each image.

Big Buck Bunny sequences

In Figure 7.27 and 7.28 the results from the low-rank matrix completion algorithm can be seen.

As explained earlier, the motion mask for this algorithm is complicated to create. Since the method is not a per-pixel method, the motion mask will look very different from the reference motion mask, regardless of which sequence that is used. Therefore, we have chosen not to provide any detection statistics for the motion mask. Because of the high complexity of the low-rank matrix completion

(a) Original image

(b) Original noisy image

(c) Filtered image

(d) Original

(e) Original noisy

(f) Filtered

(g) Original

(h) Original noisy

(i) Filtered

Figure 7.27: Frame 2305 from the Big Buck Bunny sequence filtered using the low-rank matrix completion method. The added noise has standard deviation $\sigma = 40$.

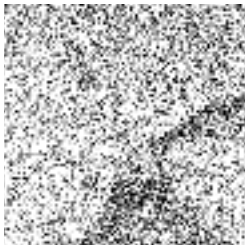
algorithm no SSIM-plots for the complete sequences are provided. The SSIM value for frame 2305 when processed is ≈ 0.092 while the noisy image has an SSIM value of 0.032.

(a) Original image

(b) Original noisy image

(c) Filtered image

(d) Original

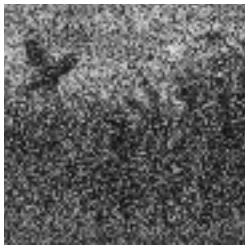


(e) Original noisy

(f) Filtered

(g) Original

(h) Original noisy



(i) Filtered

Figure 7.28: Frame 2305 from the Big Buck Bunny sequence filtered using the low-rank matrix completion method. The added noise has standard deviation $\sigma = 100$.

7.6 Blur Filtering

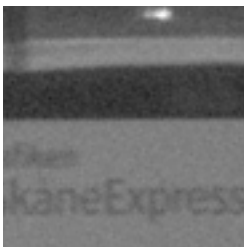
This is a very fast algorithm that seems to work well in most test cases.

The bus sequence

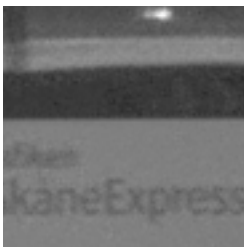
The blur filtering algorithm detects the difficult parts, e.g. the arrow pattern, in the bus sequence as seen in Figure 7.29.

(a) Original noisy image

(b) Filtered image



(c) Original



(d) Filtered

(e) Motion mask

Figure 7.29: Frame 9 from the bus sequence and corresponding filtered image and motion mask using the blur filter algorithm.

The Big Buck Bunny sequences

The results from the big buck bunny sequence can be seen in Figure 7.30, Figure 7.32 and Figure 7.34. The statistics for the sequence using the blur filter with $\sigma = 20$, $\sigma = 40$ and $\sigma = 100$ can be seen in Figure 7.31, 7.33 and 7.35

respectively.

(a) Original image

(b) Original noisy image

(c) Filtered image

(d) Original

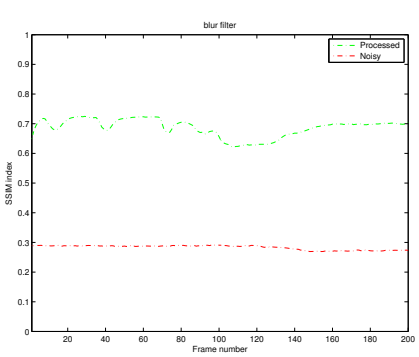
(e) Original noisy

(f) Filtered

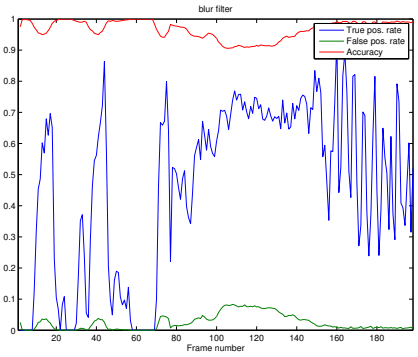
(g) Reference motion mask

(h) Motion mask

Figure 7.30: Frame 2323 from the Big Buck Bunny sequence filtered using the blur filter algorithm. The added noise has standard deviation $\sigma = 20$.



(a) SSIM.



(b) Detection statistics .

Figure 7.31: Statistics from one of the sequences from the Big Buck Bunny movie using blur filter algorithm. The added noise has standard deviation $\sigma = 20$.

(a) Original image

(b) Original noisy image

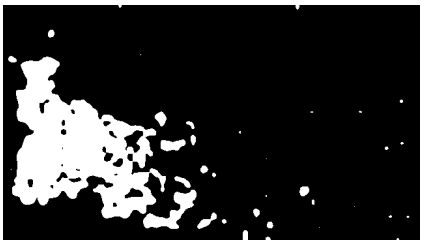
(c) Filtered image

(d) Original

(e) Original noisy

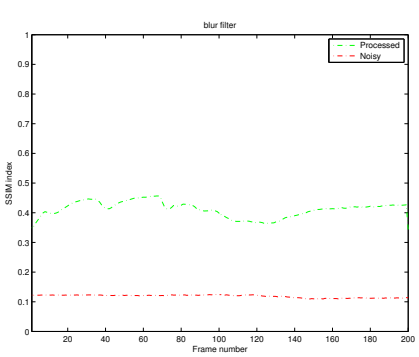
(f) Filtered

(g) Reference motion mask

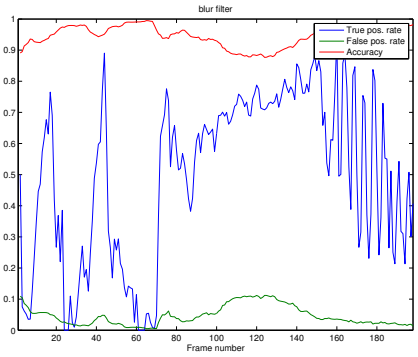


(h) Motion mask

Figure 7.32: Frame 2323 from the Big Buck Bunny sequence filtered using the blur filter algorithm. The added noise has standard deviation $\sigma = 40$.



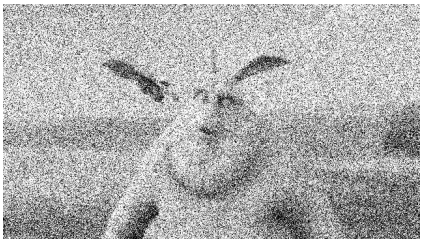
(a) SSIM.



(b) Detection statistics.

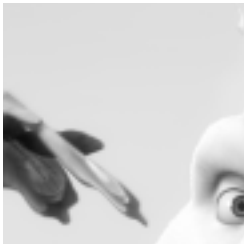
Figure 7.33: Statistics from one of the sequences from the Big Buck Bunny movie using the blur filter algorithm. The added noise has standard deviation $\sigma = 40$.

(a) Original image

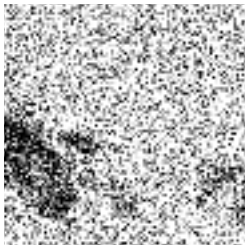


(b) Original noisy image

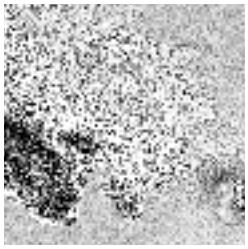
(c) Filtered image



(d) Original

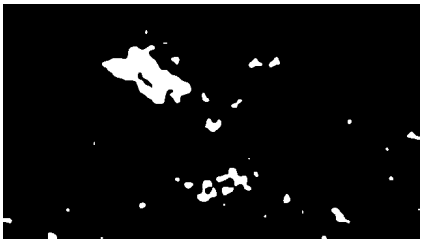


(e) Original noisy



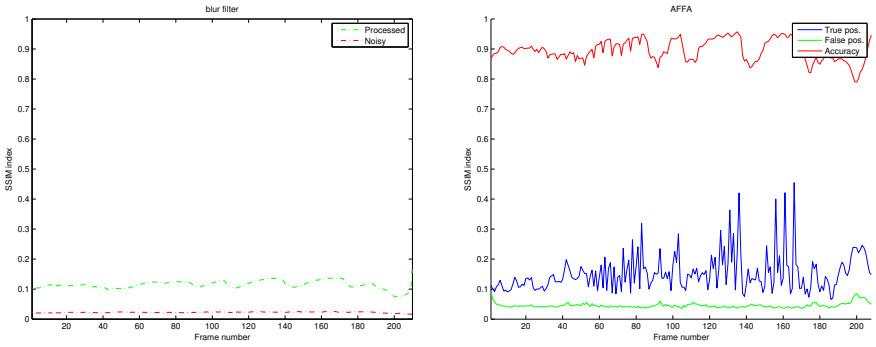
(f) Filtered

(g) Reference motion mask



(h) Motion mask

Figure 7.34: Frame 2597 from the Big Buck Bunny sequence filtered using blur filter algorithm. The added noise has standard deviation $\sigma = 100$.



(a) SSIM. (b) Detection statistics.

Figure 7.35: Statistics from one of the sequences from the Big Buck Bunny movie using the blur filter algorithm. The added noise has standard deviation $\sigma = 100$.

Chapter 8

Discussion and Conclusions

8.1 Conclusion overview

After having tested all methods in varying conditions, it is time to judge the algorithms. We will look at the results and compare the methods among themselves. The goal here is to evaluate them both subjectively and objectively by using both filtered results and plots of objective measures.

The artificial noise in the computer generated sequences differ from the noise in the real-world sequences captured by actual cameras. This is because there is a correlation between the noise and the captured intensity. The noise scales as $\sigma \propto \sqrt{Signal}$ [9]. As a consequence, dark parts of the images are corrupted by noise with lower standard deviation and lower SNR since $SNR \approx \frac{Signal}{\sqrt{Signal}} = \sqrt{Signal}$. The artificial noise has a constant distribution, independent of the signal. The methods, when used on real camera-captured sequences, can behave differently and probably better if the camera is calibrated, because calibrating means that we know something about how the noise behaves with respect to the signal.

Calibration is not possible to do for the sequence with the bus. The reason for this is that the camera used to capture it is not available to us and hence impossible to calibrate. The calibration data cannot be used in the MATLAB versions of our algorithms without rewriting much of the code. For that reason we only use the data for the methods implemented in OpenCL since those work directly with the camera.

There is one effect that is common to all temporal filtering algorithms that have been tested here. It stems from the fact that none of them use any spatial filtering whatsoever. It occurs when motion is detected in a small region surrounded by stationary pixels. What happens is that since the motion area is not filtered, while the non-motion area is, the noise in the unfiltered area can become extremely prominent. This could be mistaken for an artifact, but is actually the

way it should be.

8.2 Adaptive fuzzy filter algorithm

This method mostly produces good motion masks, and with the dilation and erosion post processing they get even better. Sometimes, too little motion is detected, but no obvious artifacts can be seen in the image sequences we have tested.

To work optimally, some tuning of the parameters is needed even though the threshold is based on the estimated standard deviation of the noise in each image. In the original paper [23], the threshold is set to $thresh = 3 \cdot \sigma_{est}$, but instead we have used a variable x such that $thresh = x \cdot \sigma_{est}$. Optimal results in each sequence was achieved with $1 \leq x \leq 4$. A method for estimating x for each image would be better, perhaps by using calibration data. The advantage with no calibration is that it works for all cameras. The higher the noise standard deviation estimate, the lower x was needed.

This method is one of the best when it comes to the real camera sequences. The variable x did not vary as much for the real camera sequences as for the synthetic sequences. One reason for this might be that the standard deviation of the noise was much higher for the synthetic sequences. None of the available real-camera sequences were as noisy as the synthetic ones.

The adaptive fuzzy filter algorithm is also one of the fastest of our methods. In OpenCL, it is limited by the frame rate of the camera and the refresh rate of the screen.

8.3 Hierarchical block-matching algorithm

The hierarchical block matching algorithm worked fairly well for most sequences. The main concern is the speed of the algorithm. It is extremely memory bandwidth hungry and performance degrades quickly when using larger search-windows. The advantage of bigger search windows, however, is that larger motion can be detected and compensated for. In some areas, too much motion is detected. This can be seen in Figure 7.11 and Figure 7.12.

A drawback of the hierarchical block-matching algorithm sometimes is revealed in smooth areas with uniform colour. In such areas, the block-matching algorithm can sometimes find structures in the noise and match those blocks. This both leads to emphasizing of the noise as well as incorrect motion detection. The problem could be reduced or avoided by again thresholding the difference between the block at vector $(0, 0)$ and the matched block and favouring the $(0, 0)$-block if the difference is small.

The speed of the algorithm does not achieve real-time performance in OpenCL, especially not when the search windows were bigger than 11×11. Even though only two frames are required, the block-matching algorithm is expensive when the search window sizes are increased.

8.4 Fuzzy logic algorithm

The Fuzzy logic approach suffers from artifacts such as ghosting and excessive blurring. The blurring is most severe in low-contrast areas. The arrows in the bus-sequence is one such problem-area. This effect can be slightly improved with a little tweaking but not avoided completely. The worst problem, perceptually, is the ghosting that appears. The ghosting appears due to the fact that the decay of the α-parameter is sometimes too slow. The slowness is a problem when misclassification happens, i.e. when a motion pixel is not detected as such in the previous step. This has proven to be a very difficult problem to deal with. We have experimented with using different update formulas but are unable to find one that completely removes the problem. We end up with either too much or too little filtering.

By modifying the previous alpha-parameter we reduce the influence of old motion data. This means that the filtering weight is more dependent on the motion estimate from the current frame. This reduces, but does not completely eliminate, ghosting but also results in generally lower noise reduction.

When the noise level is not extremely high, the motion masks can be post-processed using dilation and erosion or scaling. This removes noise in the motion mask. The binary filtered images from this modified mask look much better compared to the filtered images using the original fuzzy logic filtering.

When implemented in OpenCL, this method performs very well. It is only slightly slower than the adaptive fuzzy filter algorithm.

8.5 Low-rank matrix completion algorithm

This algorithm retains its denoising abilities for higher noise-levels than the others. It performs better in the sequences with $\sigma = 100$, for example, than the rest of the methods. Quite a lot of detail is preserved, or restored, even in that rather extreme case. The drawback is that the potential for denoising in static areas is lower than when using binary filtering with low weight for the new noisy image.

The low-rank matrix completion algorithm has a strange effect on the bus sequence. It reduces the noise very well in some areas while almost no difference in noise can be seen in other areas. Unfortunately, we do not find an explanation for that behaviour, and it does not occur in any other sequence.

This method is different from the rest because it is not just a motion detector or temporal filter. This algorithm also performs spatial filtering which is a benefit because it removes the false artifacts mentioned in section 8.1. This means that no additional spatial filtering is necessary.

The overall impression of this rather "expensive" method is that the method definitely has its strong points, but it is does not always perform better than cheaper, simpler, methods.

8.6 Blur Filtering

This method works well for most noise-levels and produces usable motion masks for a whole range of noise levels. The method appears tunable enough that reasonable results can be obtained even for the highest levels of noise.

The method can be tweaked with using things like different blurring kernels, different scaling factors or methods and thresholds. It is a very light-weight algorithm which makes it interesting from a hardware implementation perspective.

Currently, the algorithm has some problems correctly detecting the butterfly when the noise is very heavy.

8.7 Comparison between the methods

The methods can be compared in more than one way, as discussed in chapter 3. One of the aspects that can be compared, is of course the SSIM statistics, but in many cases they are not enough. Therefore, one has to look at the images and compare them subjectively to include all human aspects.

To easily compare the results from the algorithms, the same image from the bus sequence is filtered using the different methods and the corresponding motion masks are displayed in Figure 8.1 and 8.2. The filtered images of the bus sequence, for example, cannot be measured using SSIM since no reference image or reference motion mask exist. Therefore, only subjective measurements can be used. The adaptive fuzzy filtering algorithm seem to generate the best output when considering the whole image. Although the arrows that actually were filtered in the low-rank matrix completion algorithm were even better, the noise in the rest of the image remains more or less unchanged.

Note that the comparison with the hierarchical block-matching is slightly unfair. This is because the conversion to grey-scale from RGB suppresses the standard deviation of the noise by a factor of $\sqrt{3}$.

It is easier to compare the performance of the algorithms using synthetic images since the SSIM-plots are available. The SSIM-plots have to be considered

(a) Noisy image

(b) Hierarchical block-matching, 11×11

(c) Hierarchical block-matching, 25×25

(d) Adaptive fuzzy filter

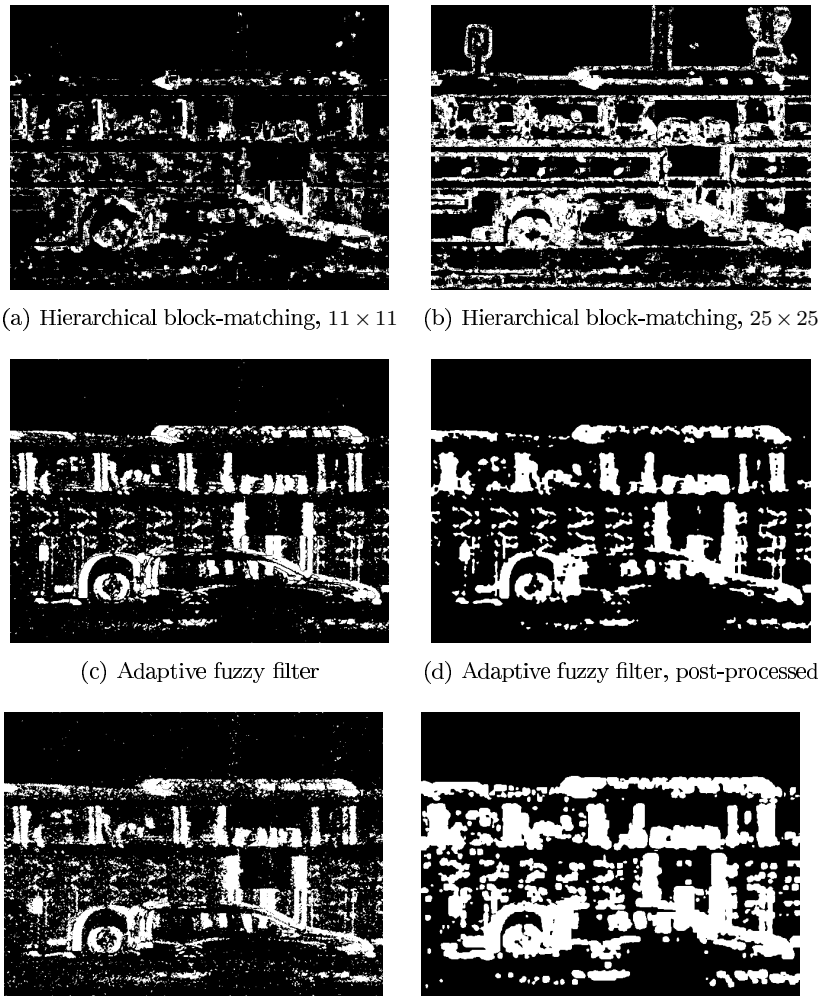
(e) Adaptive fuzzy filter, post-processed

(f) Modified fuzzy logic algorithm

(g) Low-rank matrix completion algorithm

(h) Blur filter algorithm

Figure 8.1: One image from the bus sequence filtered using a few our methods.



(a) Hierarchical block-matching, 11×11 (b) Hierarchical block-matching, 25×25

(c) Adaptive fuzzy filter (d) Adaptive fuzzy filter, post-processed

(e) Modified fuzzy logic algorithm (f) Blur filter algorithm

Figure 8.2: Motion masks from the bus sequence, corresponding to the filtered images in Figure 8.1.

together with the subjective measurements, to come to a conclusion of which method that performs the best. Also, the method that is best for one sequence may not be best for another. Noise level also plays a role since some methods break down faster than others. One image from the Big Buck Bunny sequence where the noise added has a standard deviation of $\sigma = 40$ is filtered using all methods. The filtered images and corresponding motion masks can be viewed in Figure 8.3 and 8.4. The SSIM-plots for the whole sequence with this particular static scene can be seen in Figure 8.5.

The SSIM values for the filtered output of all methods and for the noisy images are plotted in Figure 8.5(a). For comparison, the SSIM values using binary

(a) Noisy image

(b) Original image

(c) Hierarchical block-matching search window 11×11

(d) Hierarchical block-matching, search window 25×25

(e) Adaptive fuzzy filter

(f) Post processed adaptive fuzzy filter

(g) Modified fuzzy logic

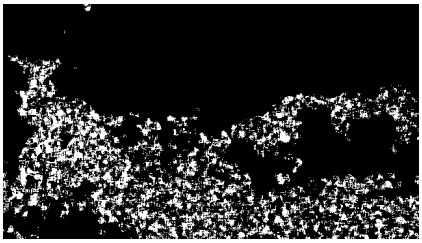
(h) Blur filter

Figure 8.3: Frame 2323 from the Big Buck Bunny sequence filtered using a few of our methods. The added noise has standard deviation $\sigma = 40$.

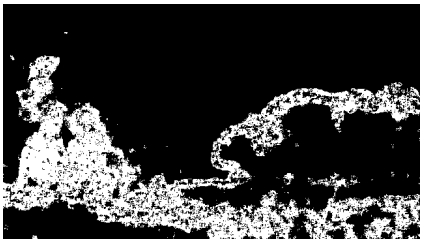
filtering with the reference motion mask are also included.

In Figure 8.5, it is shown that the hierarchical block-matching algorithm has potential when the size of the search window increases. The problem, however, is that the algorithm becomes extremely computationally complex when the window size gets bigger. The post-processed adatptive fuzzy filtering algorithm

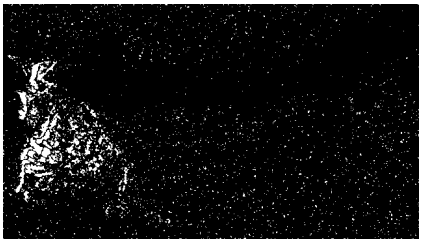
(a) Reference motion mask



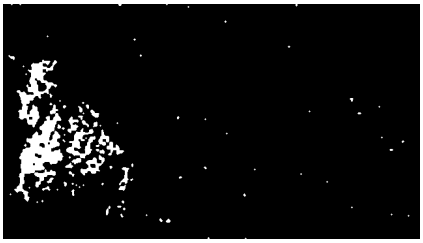
(b) Hierarchical block-matching, 11×11



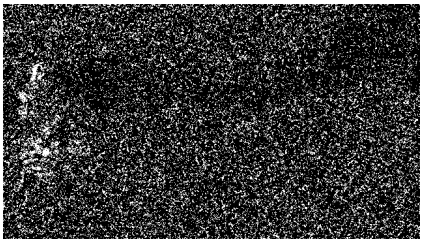
(c) Hierarchical block-matching, 25×25



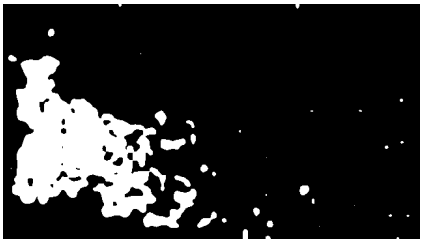
(d) Adaptive fuzzy filter



(e) post-processed adaptive fuzzy filter



(f) Modified fuzzy logic



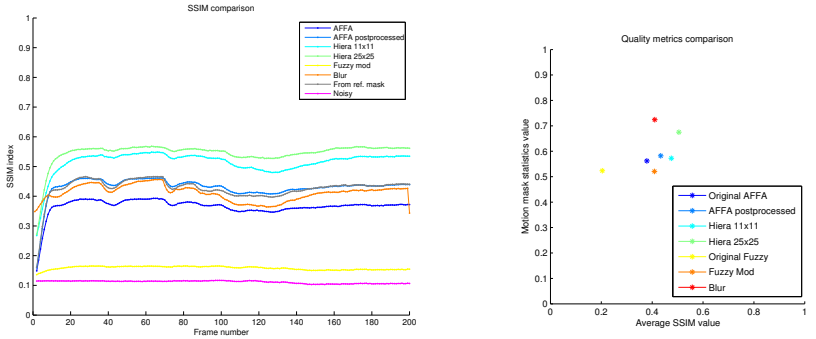
(g) Blur filter

Figure 8.4: Motion masks for the image in Figure 8.3. The added noise has standard deviation $\sigma = 40$.

performs better than the original algorithm, which can be seen both on the images and in the statistics plots. One reason that the SSIM values are better for this method, compared to the others, might be that it is a motion compensated method and not that many "false" artifacts are introduced since the filtering is performed in the motion direction.

There seems to be a correlation between the SSIM values and the motion mask

statistics. A better SSIM value yields a better motion mask statistic value in most cases, as seen in Figure 8.5.



(a) SSIM

(b) Detection statistics value vs. SSIM

Figure 8.5: The SSIM plots and the detecion statistics value vs. the average SSIM value. The added noise has standard deviation $\sigma = 40$.

Generally it seems that the blur filtering algorithm and the adaptive fuzzy filtering algorithm work best for moderate noise, i.e. up to about $\sigma = 55$. For noise with $\sigma \geq 100$ the method that works best is the low-rank matrix completion algorithm. Figure 8.6 shows how one method behaves when the other performs well and vice versa. Here, the blur filtering algorithm is used.

(a) $\sigma = 100$, Blur filtering algorithm.

(b) $\sigma = 40$, Blur filtering algorithm.

(c) $\sigma = 100$, Low-rank matrix completion algorithm.

(d) $\sigma = 40$, Low-rank matrix completion algorithm.

Figure 8.6: Comparison of frame 2305 from the Big Buck Bunny sequence filtered using the two best algorithms.

For more figures where the same image for all methods is filtered, see Appendix.

8.8 Conclusion

For noise levels not extremely high, the adaptive fuzzy filter algorithm seems to perform the best along with the blur algorithm. Although in some cases the hierarchical block matching algorithm with large search windows performs better. The drawback is that when the window size increases, the algorithm gets extremely computationally heavy. Increasing the windows to more than the 25×25 windows used here is simply not feasible. We have therefore not been able to experiment that much with bigger search windows for long sequences.

When the noise level increases, to around $\sigma = 100$, the low-rank matrix completion algorithm seems to outperform the other methods. However, it is sometimes hard to say how well the methods behave since the "false" artifacts can be extremely hard to distinguish from real artifacts in some cases. When looking at Figure 8.5 it may seem that the hierarchical block-matching is the best method, but we trust our own subjective judgement more. Therefore, we think that the adaptive fuzzy filtering method is better.

8.9 Future work

If the camera is calibrated, that data could perhaps be used to automatically determine the parameter x for the adaptive fuzzy filter algorithm. This could possibly be achieved by combining calibration with the standard deviation estimate of the method.

The parameters have been tuned manually to give the best result for each image sequence. Especially the hierarchical block-matching algorithm has many parameters that can be tweaked. A future work would be to optimize these parameters automatically. Here, calibration of the camera is probably not enough. The parameters are simply too many.

Another important aspect is that all algorithms, except the hierarchical block-matching algorithm need to be extended to work with colour images. This can probably be done similarly to how the hierarchical algorithm performs the colour correction. Since the results of the algorithms can be seen clearly, even with grey-scale images, this has been ignored due to limitations of time.

The low-rank algorithm could perhaps also be improved by progressively using the denoised images in the sequence. By doing so, the denoising of static areas should hopefully approach the same potential denoising performance as binary filtering with high weight to the previous filtered image.

Due to time constraints, the blur filtering algorithm has not received as much attention as we would have liked. It shows real potential but needs further investigation. Things like choosing the blur kernel and threshold adjustment could perhaps be automated, which would be a very nice improvement.

Bibliography

[1] Eric P. Bennett and Leonard McMillan. Video enhancement using per-pixel virtual exposures. *ACM Trans. Graph.*, 24(3):845–852, July 2005. ISSN 0730-0301. doi: 10.1145/1073204.1073272. URL `http://doi.acm.org/10.1145/1073204.1073272`.

[2] A. Bosco, M. Mancuso, S. Battiato, and G. Spampinato. Temporal noise reduction of bayer matrixed video data. In *Multimedia and Expo, 2002. ICME '02. Proceedings. 2002 IEEE International Conference on*, volume 1, pages 681 – 684 vol.1, 2002. doi: 10.1109/ICME.2002.1035873.

[3] Jia Chen and Chi keung Tang. Spatio-temporal markov random field for video denoising. In *In CVPR*, 2007.

[4] Tomas Crivelli, Patrick Bouthemy, Bruno Cernuschi-Frias, and Jian-Feng Yao. Simultaneous motion detection and background reconstruction with a conditional mixed-state markov random field. *International Journal of Computer Vision*, 94(3):295–316, 2011. doi: 10.1007/s11263-011-0429-z. URL `http://hal.archives-ouvertes.fr/hal-00628797`.

[5] Blender Foundation. Big buck bunny, June 2012. URL `http://www.bigbuckbunny.org`.

[6] Suk ho Lee, Nam seok Choi, and Moon Gi Kang. Motion detection with level set-based segmentation. volume 7538, page 75380O. SPIE, 2010. doi: 10.1117/12.839159. URL `http://link.aip.org/link/?PSI/7538/75380O/1`.

[7] M. Hürlimann and P.D.A. Schenker-Wicki. *Dealing with Real-World Complexity: Limits, Enhancements and New Approaches for Policy Makers*. Gabler Edition Wissenschaft. Gabler Verlag, 2009. ISBN 9783834914934.

[8] H. Hwang and R.A. Haddad. Adaptive median filters: new algorithms and results. *Image Processing, IEEE Transactions on*, 4(4):499 –502, apr 1995. ISSN 1057-7149. doi: 10.1109/83.370679.

[9] J.R. Janesick. *Photon Transfer: Dn [Lambda]*. Press Monographs. SPIE, 2007. ISBN 9780819467225.

[10] Hui Ji, Chaoqiang Liu, Zuowei Shen, and Yuhong Xu. Robust video denois-

ing using low rank matrix completion. In *CVPR*, pages 1791–1798. IEEE, 2010. URL http://dblp.uni-trier.de/db/conf/cvpr/cvpr2010.html#JiLSX10.

[11] P. Kaewtrakulpong and R. Bowden. An improved adaptive background mixture model for realtime tracking with shadow detection, September 2001. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.3705.

[12] Yoon-Jong Kim, Jung-Jae Oh, Byung-Tae Choi, Seung-Jong Choi, and Eung-Tae Kim. Robust noise reduction using a hierarchical motion compensation in noisy image sequences. *Computers in Education, International Conference on*, 0:1–2, 2009. doi: http://doi.ieeecomputersociety.org/10.1109/ICCE.2009.5012248.

[13] B. Liu and A. Zaccarin. New fast algorithms for the estimation of block motion vectors. *Circuits and Systems for Video Technology, IEEE Transactions on*, 3(2):148 –157, apr 1993. ISSN 1051-8215. doi: 10.1109/76.212720.

[14] Henrik Malm, Magnus Oskarsson, Petrik Clarberg, Jon Hasselgren, Calle Lejdfors, and Eric Warrant. Adaptive enhancement and noise reduction in very low light-level video. In *International Conference on Computer Vision, Rio de Janeiro, Brazil*, 2007.

[15] Travis Portz, Li Zhang, and Hongrui Jiang. High-quality video denoising for motion-based exposure control. In *ICCV Workshops*, pages 9–16. IEEE, 2011. ISBN 978-1-4673-0062-9. URL http://dblp.uni-trier.de/db/conf/iccvw/iccvw2011.html#PortzZJ11.

[16] K. Rank, M. Lendl, and R. Unbehauen. Estimation of image noise variance. *Vision, Image and Signal Processing, IEE Proceedings* -, 146(2):80 –84, aug 1999. ISSN 1350-245X. doi: 10.1049/ip-vis:19990238.

[17] Daniel J. Schroeder. Chapter 17 - detectors, signal-to-noise, and detection limits. In *Astronomical Optics (Second Edition)*, pages 425 – 443. Academic Press, San Diego, second edition edition, 2000. ISBN 978-0-12-629810-9. doi: 10.1016/B978-012629810-9/50018-9. URL http://www.sciencedirect.com/science/article/pii/B9780126298109500189.

[18] C. Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. volume 2, pages 246–252, Los Alamitos, CA, USA, August 1999. IEEE Computer Society. doi: 10.1109/CVPR.1999.784637. URL http://dx.doi.org/10.1109/CVPR.1999.784637.

[19] Thomas Veit, Frédéric Cao, and Patrick Bouthemy. An a contrario decision framework for region-based motion detection. *Int. J. Comput. Vision*, 68 (2):163–178, June 2006. ISSN 0920-5691. doi: 10.1007/s11263-006-6661-2. URL http://dx.doi.org/10.1007/s11263-006-6661-2.

[20] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality

assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600 –612, april 2004. ISSN 1057-7149. doi: 10.1109/TIP.2003.819861.

[21] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. The ssim index for image quality assessment, June 2012. URL `https://ece.uwaterloo.ca/~z70wang/research/ssim/`.

[22] Hyenkyun Woo, Yoon Mo Jung, Jeong-Gyoo Kim, and Jin Keun Seo. Environmentally robust motion detection for video surveillance. *Trans. Img. Proc.*, 19(11):2838–2848, November 2010. ISSN 1057-7149. doi: 10.1109/TIP.2010.2050644. URL `http://dx.doi.org/10.1109/TIP.2010.2050644`.

[23] Jing Wu, Xin Du, Yun fang Zhu, and Gu Wei-kang. Adaptive fuzzy filter algorithm for real-time video denoising. In *International Conference on Signal Processing Proceedings*, 2008. doi: 10.1109/ICOSP.2008.4697367.

[24] Qing Xu, Hailin Jiang, Riccardo Scopigno, and Mateu Sbert. A new approach for very dark video denoising and enhancement. In *Image Processing, IEEE International Conference*, pages 1185–1188, 2010. doi: 10.1109/ICIP.2010.5651838.

[25] Vladimir Zlokolica, Aleksandra Pizurica, Wilfried Philips, Stefan Schulte, and Etienne Kerre. Fuzzy logic recursive motion detection and denoising of video sequences. *Journal of Electronic Imaging*, 15(2):023008, 2006. doi: 10.1117/1.2201548. URL `http://link.aip.org/link/?JEI/15/023008/1`.

Appendix A

Additional Images

(a) Noisy image

(b) Original image

(c) Hierarchical block-matching, 11×11

(d) Hierarchical block-matching, 25×25

(e) Adaptive fuzzy filter algorithm

(f) post-processed adaptive fuzzy filter algorithm

(g) Modified fuzzy logic algorithm

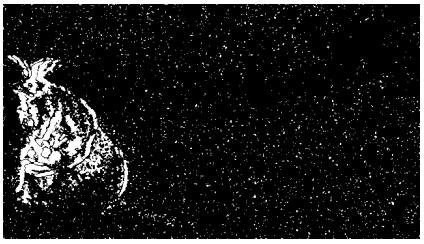
(h) Blur filter algorithm

Figure A.1: Frame 2323 from the Big Buck Bunny sequence filtered using our methods. The added noise has standard deviation $\sigma = 20$.

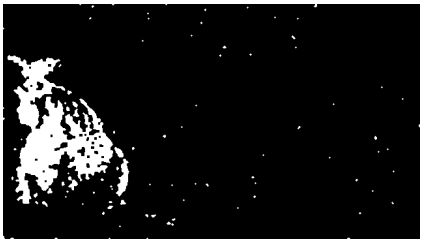
(a) Reference motion mask

(b) Hierarchical block-matching, 11×11

(c) Hierarchical block-matching, 25×25



(d) Adaptive fuzzy filter algorithm

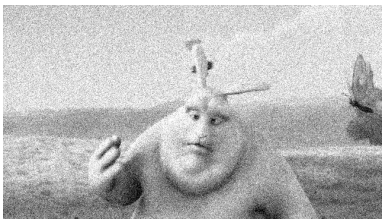


(e) post-processed adaptive fuzzy filter algorithm

(f) Modified fuzzy logic algorithm

(g) Blur filter algorithm

Figure A.2: Motion masks for the corresponding filtered images in Figure A.1. The added noise has standard deviation $\sigma = 20$.



(a) Noisy image

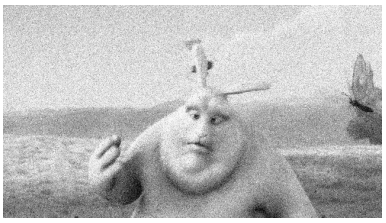
(b) Original image

(c) Hierarchical block-matching, 11×11

(d) Hierarchical block-matching, 25×25

(e) Adaptive fuzzy filter algorithm

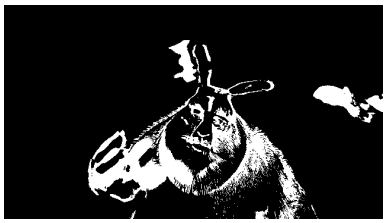
(f) Post-processed adaptive fuzzy filter algorithm



(g) Modified fuzzy logic algorithm

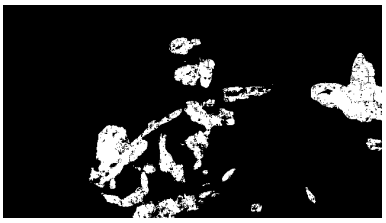
(h) Blur filter algorithm

Figure A.3: Frame 2543 from the Big Buck Bunny sequence filtered using a few of our methods. The added noise has standard deviation $\sigma = 30$.



(a) Reference motion mask

(b) Hierarchical block-matching, 11×11

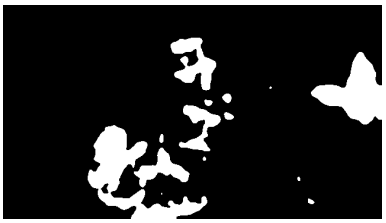


(c) Hierarchical block-matching, 25×25

(d) Adaptive fuzzy filter algorithm

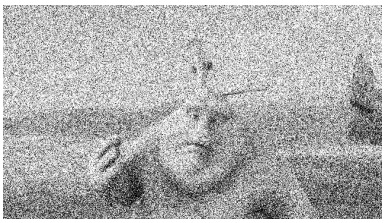
(e) Post-processed adaptive fuzzy filter algorithm

(f) Modified fuzzy logic algorithm



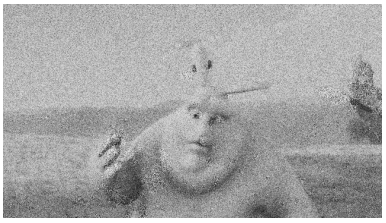
(g) Blur filter algorithm

Figure A.4: Motion masks for the corresponding filtered images in Figure A.3. The added noise has standard deviation $\sigma = 30$.

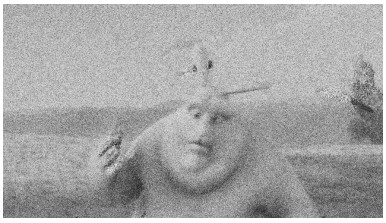


(a) Noisy image

(b) Original image



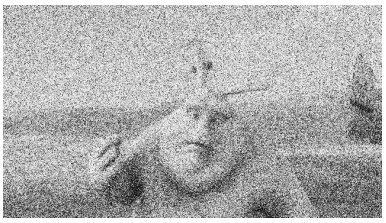
(c) Hierarchical block-matching, 11×11



(d) Hierarchical block-matching, 25×25

(e) Adaptive fuzzy filter algorithm

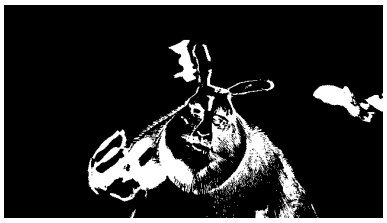
(f) Post-processed adaptive fuzzy filter algorithm



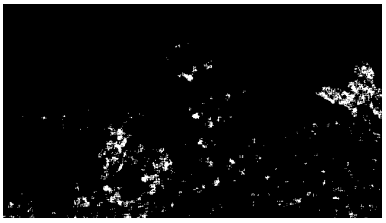
(g) Modified fuzzy logic algorithm

(h) Blur filter algorithm

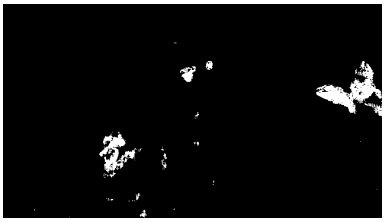
Figure A.5: Frame 2543 from the Big Buck Bunny sequence filtered using our methods. The added noise has standard deviation $\sigma = 100$.



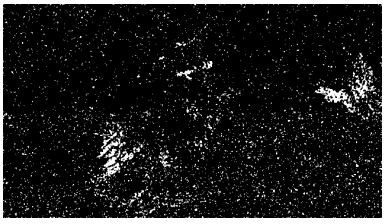
(a) Reference motion mask



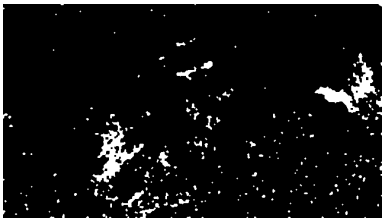
(b) Hierarchical block-matching, 11×11



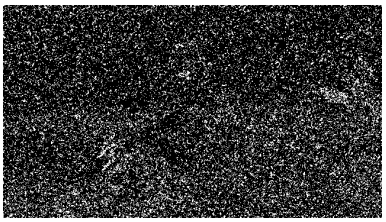
(c) Hierarchical block-matching, 25×25



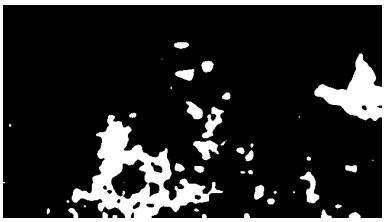
(d) Adaptive fuzzy filter



(e) Post-processed adaptive fuzzy filter



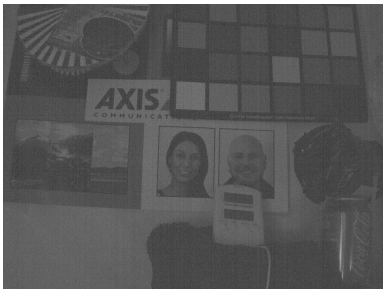
(f) Modified fuzzy logic algorithm



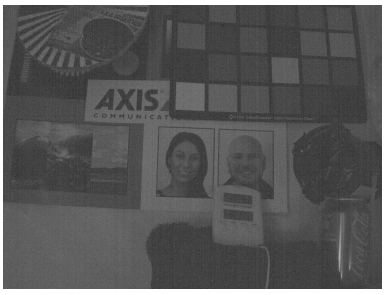
(g) Blur filter algorithm

Figure A.6: Motion masks for the corresponding filtered images in Figure A.5. The added noise has standard deviation $\sigma = 100$.

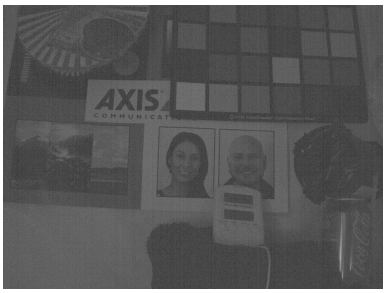
(a) Noisy image



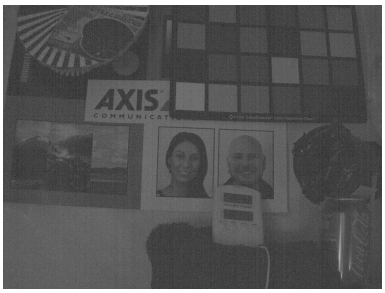
(b) Hierarchical block-matching algortihm



(c) Adaptive fuzzy filter algorithm



(d) Fuzzy logic algorithm



(e) Blur filter algorithm

Figure A.7: One image from the polaris sequence filtered using a few of our methods.

(a) Hierarchical block-matching

(b) Adaptive fuzzy filter

(c) Fuzzy logic algorithm

(d) Blur filter algorithm

Figure A.8: Motion mask fo the filtered images in Figure A.7.

Master's Theses in Mathematical Sciences 2012:E32
ISSN 1404-6342

LUTFMA-3232-2012

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

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