Analysis of Optical Flow Algorithms for Denoising

Markus Larsson
Louise Söderström

Master's thesis
2015:E15

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

ANALYSIS OF OPTICAL FLOW ALGORITHMS FOR DENOISING

Markus Larsson
Louise Söderström
June 4, 2015

Master's thesis work carried out at Axis.
Axis supervisor: Fredrik Olofsson
Axis supervisor: Gunnar Dahlgren
LTH supervisor: Anders Heyden

Abstract

When a video sequence is recorded in low-light conditions, the image often become noisy. Standard methods for noise reduction have difficulties with motion. But the interesting parts in a video is often the ones that are moving, for instance a burglar captured in a surveillance video.

One approach for denoising video sequences is to use temporal filtering controlled by optical flow, which describes how pixels move between two image frames. Today, there exists few studies comparing how different optical flow algorithms perform on noisy video sequences. Four different algorithms have been analyzed in the thesis. Moreover, a comparison on how well they can be used to improve the result of a temporal noise filter has been done. The conclusion of the comparison is that optical flow is useful for noise reduction. Algorithms based on patch matching and edge consistency perform better than algorithms based on color consistency.

A recommendation for future work is to combine the best parts of each algorithm to develop a new optical flow algorithm, specialized on noisy image sequences. Furthermore, develop and implement a sophisticated optical flow based noise filter in camera hardware.

Keywords: Optical flow, noise reduction, video sequences, video surveillance, algorithms

Acknowledgment

We would like to thank our supervisors at Axis, Fredrik Olofsson and Gunnar Dahlgren, for much feedback and help during the thesis. We would also like to thank Jesper Lindström and all members of the Core Technologies Imaging Systems group for our time at Axis. Furthermore, a thanks to our supervisor Anders Heyden at Centre for Mathematical Sciences at LTH.

Contents

Chapter 1

Introduction

A common problem in video recording is insufficient incident light to the camera. To mitigate the low light level, the camera may increase its gain or select a longer exposure time. Both methods have drawbacks. Increasing the gain amplifies the entire image content and therefore also increases the amount of noise, while a longer exposure time causes blurring of motion in the captured video sequence. In this thesis, the focus is how to handle noisy image sequences and not motion blur. Noise filtering needs to understand how the scene has changed to filter it optimally. This can be performed with help of optical flow, which is a way of describing the motion in the scene as seen by the observer (a person's eye or the camera). If optical flow is used to guide a noise filter, the optical flow must perform well on noisy images. Several comparisons of optical flow algorithms have earlier been done [1–3], but these have not tested the performance on noisy image sequences, nor whether the algorithms can steer a noise filter, which is the focus of this thesis.

1.1 Related work

Both optical flow and image denoising have been studied for decades. As it is well beyond the scope of this report to give a thorough review of the entire literature on these subjects, the focus is on the most closely related work.

Research in optical flow goes back to 1981, when the classical paper "Determining optical flow" by Horn and Schunck [4] was published. The same year Kanade and Lucas [5] presented their local approach. Since then, a massive amount of articles and papers have been written about the subject. Among this variety of approaches and algorithms, four have been investigated and evaluated in this thesis. They have been chosen according to performance and execution time. The names of the algorithms are SimpleFlow [6], Fast Edge-Preserving PatchMatch (EPPM) [7], Classic+NL-Fast [8] and Large

Displacement Optical Flow (LDOF) [9].

In order to evaluate and compare optical flow algorithms, several benchmarks, all with their own dataset, have been published. One classical such benchmark is Barron [10], while more recent benchmarks include Middlebury [1], MPI-Sintel [2] and KITTI [3]. The purpose of the datasets, belonging to these benchmarks is to test a variety of things, including shadows, low contrast, great textureless areas, motion blur, reflections and varied light conditions, but none of them focuses on noisy images.

Noise reduction based on optical flow has been discussed for at least the past ten years. While Buades, Coll and Morel argued that denoising image sequences does not require motion estimation [11, 12], more recent papers, such as Dudek, Quintana and Cuenca from 2009 [13] and Liu and Freeman from 2010 [14], say the opposite and prove that image denoising can be improved with the help of optical flow. These papers have though their focus on the noise filter and does not discuss which optical flow algorithm that is most suitable to use for noisy image sequences. Furthermore, Liu and Freeman [14] argued for the advantage of using motion estimation in denoising, especially for structured noise.

1.2 Aim of the thesis

The goal of this thesis is to investigate which optical flow algorithms or types of algorithms that are suitable for noisy video sequences in order to denoise. A few optical flow algorithms will be implemented and executed on noisy video sequences. In order to analyze the result and compare the algorithms, a ranking system will be developed. A simple noise filter based on optical flow will also be implemented.

1.2.1 Research questions

- How do existing optical flow algorithms perform on noisy video sequences?
- What kind of optical flow method generates the best result for noise filtering?
- How can noise reduction take advantage of optical flow?

1.3 Limitations

One of the main limitations in this thesis is the number of optical flow algorithms considered. Only four of the numerous existing optical flow algorithms

have been tested and compared.

While the thesis is performed in cooperation with Axis Communications AB, the focus is noise filtering of network video with possibility to be extended to real time. Therefore, several algorithms that may produce a better result on noisy video sequences, but are considered too slow, are omitted.

1.4 Error sources

A major error source is that it is hard to know exactly how the considered optical flow algorithms which have been implemented during the thesis (SimpleFlow and EPPM) are originally implemented. In the case of EPPM, the only source is a paper, where some parts of the algorithm are explained thoroughly, while other parts are just mentioned passingly [7]. Even if SimpleFlow [6] has available, uncompiled code, it is not used directly, but translated into OpenCL, so some of its benefits may get lost. All parameter choices are not mentioned in the papers or code. All this together leads to the implementations in this report possibly varying quite a lot from the original implementations, both when it comes to performance and execution time.

The selection of algorithms to be implemented is, except the execution time limitation mentioned in Section 1.3, also based on availability of papers and code and on an own classification (see Section 3.1 for details). Furthermore, only optical flow algorithms present on the MPI-Sintel benchmark [2] are considered, so there is a risk that there exists other algorithms than SimpleFlow, EPPM, Classic+NL-Fast and LDOF, which are better suitable for noise removal.

1.5 Structure of the report

Chapter 2 begins with some terminology needed to understand the rest of the report. Thereafter, theory about relevant parts in image processing is presented. This part contains information about, for instance, different filtering and interpolation methods. Then, some color theory is presented. Background theory about noise and noise measurements, as well as optical flow and a presentation of three existing benchmarks for measuring optical flow are also included in this chapter.

Chapter 3 describes the method used during the thesis. The sections included are choosing algorithms, implementation and parameter chooses, choosing test images and analysis of different noise levels.

Chapter 4 starts with a presentation of the four optical flow algorithms

considered: SimpleFlow, EPPM, Classic+NL-fast and LDOF. In the case of SimpleFlow and EPPM, implementation details and parameter choices are also presented because these two algorithms have been implemented during the thesis while the others have only been evaluated. The chapter continues with a description of the noise filters that have been developed during the thesis.

Chapter 5 introduces the dataset used in the thesis. First comes a description of each sequence from the dataset and some image frames from them. Then, example images of the different noise levels and image histograms showing the intensities of one frame typical for each sequence are presented.

Chapter 6 presents the results of the evaluation of the noise filtering. The filter is tested on four image sequences with five different noise levels each. All experiments are performed both with and without pre-processing. Besides the result measured by the structured similarity (SSIM) error measure, the amount of occlusions found in each sequence by each algorithm is presented.

Chapter 7 consists of an analysis of the results and occlusion detection, presented in Chapter 6. Whether the algorithms gains of the pre-processing step and how well they handle illumination changes are also included in the analysis.

Chapter 8 contains a general discussion about the thesis and especially about the conclusions made in the analysis. A final conclusion, which answers the research questions (see Section 1.2.1), is also included in this chapter. Furthermore, ideas of future work in the field are presented.

Chapter 2

Theory

2.1 Terminology

Two adjectives often heard in image analysis are *spatial* and *temporal*. A spatial feature is related to the position, size or area of an object and a temporal feature is related to time. For instance, a spatial image filter uses nearby pixels in the same image frame, while a temporal image filter uses the same pixel in several subsequent frames.

Warp means to transform an image so that shapes in it become distorted [15].

A *patch* is a small area around a certain pixel [15]. In this report, the word patch is used for a smaller, typically quadratic, segment of an image. The *patch radius* is then the shortest distance from the center pixel to an edge pixel of the patch, i.e. a patch having patch radius r consists of $(2r + 1) \times (2r + 1)$ pixels. For an example of this, see Figure 2.1.

Motion boundaries are boundaries between adjacent regions with different velocities in an image. The detection of motion boundaries can be used to improve optical flow estimation and to extract information about the surface boundaries in the image. The study of occlusions (see Section 2.5.1) together with motion boundaries can tell the relative depth of adjacent objects, i.e. which of the objects that are closest to the camera [16].

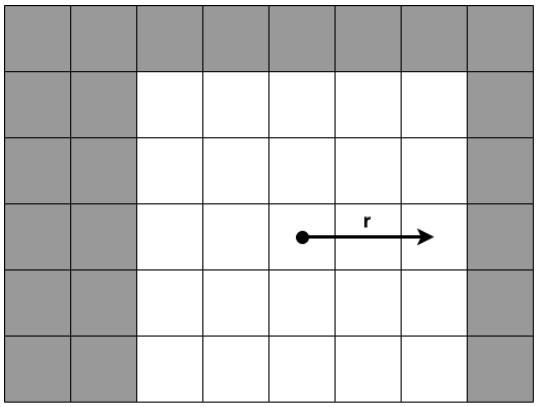


Figure 2.1: An example of a patch with radius $r = 2$, which thus consists of $(2r + 1) \times (2r + 1) = 25$ pixels.

2.2 Image processing theory

2.2.1 Filtering methods

In a linear filter, the value of the output pixel is a weighted sum of the values of the input pixels. This can be written as

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l) = \sum_{k,l} f(k,l)h(i-k, j-l) \ , \tag{2.1}$$

where f is the input image and h is called *kernel* or *mask*. One common kernel is the Gaussian kernel, which in two dimensions is written as

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \ . \tag{2.2}$$

Equation (2.1) can also be written using convolution notation as

$$g = f * h \ . \tag{2.3}$$

While linear filters are convenient to work with, a better result is often obtained by non-linear filters, which use a non-linear combination of nearby pixels [15].

2.2.1.1 Bilateral filter

An example of a non-linear filter is the *bilateral filter*. The bilateral filter is an adaptive filter, which tries to preserve edges by comparing the value of the pixels. Nearby pixels with similar values will influence the averaging

more than other nearby pixels. The output of a bilateral filtering at a point p in image I is

$$BF[I]_p = \frac{1}{W_p} \sum_q G_{\sigma_s}(||p-q||) G_{\sigma_r}(|I_p - I_q|) I_q \ , \tag{2.4}$$

where W_p is a normalization factor and G is the 2D Gaussian kernel (equation (2.2)). The parameters σ_s and σ_r decide how much the spatial distance and difference in value, respectively, will affect the result. A large σ_s smooths larger features while a large σ_r means that the filter approximates a Gaussian convolution [17]. A bilateral filter's ability to preserve edges in comparison with a Gaussian filter is shown in Figure 2.2.

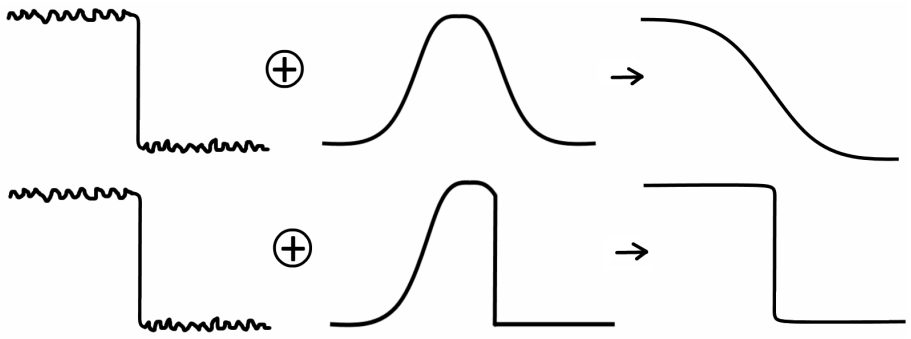


Figure 2.2: The principal difference between a Gaussian filter (upper part of the image) and a bilateral filter (lower part of the image).

2.2.1.2 Median filter

Another non-linear filtering method, often used for noise removal, is *median filtering*. For each pixel, the median filter picks the median of the pixel's neighborhood and assign that value to the pixel. Two drawbacks with this kind of filter is that it performs poorly on regular Gaussian noise and that it is not very fast because the pixels need to be sorted in order to find the median. It is, however, good for removing impulse noise, where one pixel value is very far from its neighbors. There are also weighted median filters where the closest pixels are used multiple times in the median calculation [15]. Another type of weighted median filter uses bilateral weights between the current pixel and the pixels inside the filter for giving different pixels different impact [18].

2.2.2 Interpolation methods

Assume that the function $f(k,l)$ describes some property (for instance the amount of red or the optical flow) at each pixel (k,l) on a coarse grid. The goal of interpolation is to create a function $g(i,j)$ on a finer grid which is equivalent to f on the coarser grid. The function g can be found by using the convolution formula

$$g(i,j) = \sum_{k,l} f(k,l)h(i - r \cdot k, j - r \cdot l) \ , \tag{2.5}$$

where the integer r is the upsampling rate. The interpolation kernel h depends on the interpolation method used [15].

2.2.2.1 Bilinear interpolation

The interpolation kernel of the bilinear interpolation is

$$h(x,y) = \begin{cases} \frac{4}{16} & \text{if } x = 0, y = 0 \\ \frac{2}{16} & \text{if } |x| = 1, y = 0 \quad \text{or } x = 0, |y| = 1 \\ \frac{1}{16} & \text{if } |x| = 1, |y| = 1 \\ 0 & \text{otherwise} \end{cases} . \tag{2.6}$$

This corresponds to the convolution kernel $h = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$.

Bilinear interpolation is often used when speed is crucial, but it gives worse visual quality than more complicated interpolation methods [15].

2.2.2.2 Bicubic interpolation

The interpolation kernel of the bicubic interpolation is

$$h(x) = \begin{cases} 1 - (a+3)x^2 + (a+2)|x|^3 & \text{if } |x| < 1 \\ a(|x| - 1)(|x| - 2)^2 & \text{if } 1 \leq |x| < 2 \\ 0 & \text{otherwise} \end{cases} . \tag{2.7}$$

The convolution with this kernel is performed in both dimensions. This interpolation is a piecewise-cubic spline. Spline means that the function is piecewise polynomial and C^1. The parameter a in Equation (2.7) is the specification of the derivative at $x = 1$, which can be chosen, but often is set to -1 [15]. Figure 2.3 shows the bicubic interpolation kernel for $a = -1$.

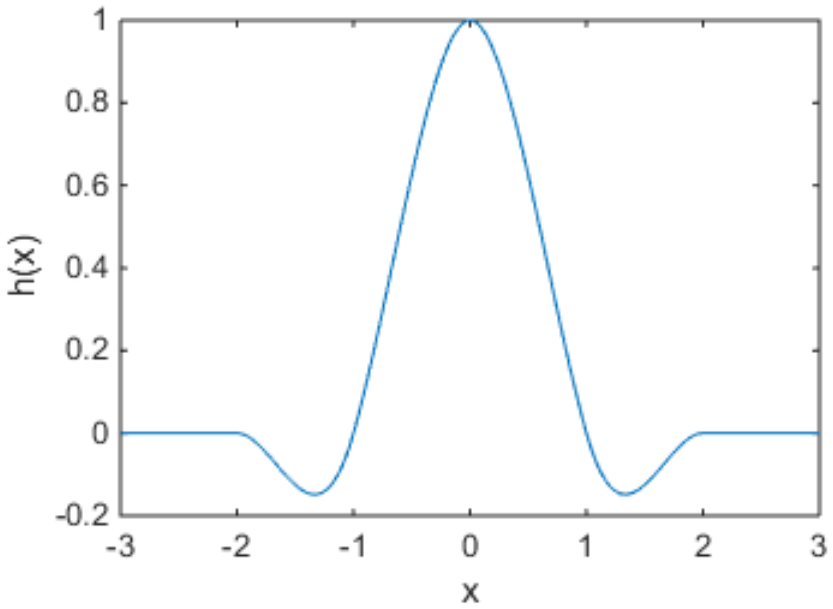


Figure 2.3: The bicubic interpolation kernel with $a = -1$.

2.2.2.3 Joint-bilateral upsampling

Bilateral upsampling is based on the edge preserving bilateral filter (explained in Section 2.2.1.1). The bilateral filter consist of one spatial filter and one range filter. Joint (or cross) bilateral filters differ from normal bilateral filters in the way that they are using another image in the range filter. The image that should be upsampled is put in the spatial filter (often truncated Gaussian). For instance, when upsampling optical flow, the high resolution image is used in the range filter and the flow field is used in the spatial filter. In this way the upsampling process gives a less smooth result [19].

2.2.3 SIFT and HOG

SIFT is an algorithm that detects and describes feature points (descriptors) in images. The goal of this is to match areas that corresponds to the same 3D object in different images, which can be useful for, among other things, segmentation and optical flow. The features are found by calculating gradients in a 16×16 window. The algorithm weights the gradients such that gradients far from the center point get less weight. The weight function is a kind of Gaussian fall-off function. Gradient histogram orientations is created on each 4×4 quadrant of the 16×16 window. Every histogram has eight bins each. It gives $4 \times 4 \times 8$ bins which form the descriptor. SIFT normalizes the vector to handle affine change of illumination. After that, all values are

cut down so they are less than 0.2. The reason of that is to throw away photometric variations. As a last step the vector is normalized again [15].

HOG is a kind of simplified version of SIFT. HOG finds descriptors on a dense grid (unlike SIFT) which is uniformly distributed. HOG only works in one scale and a fixed orientation. Thus, it can only handle very small changes in scaling and rotation [15].

2.2.4 Nearest neighbor search

Nearest neighbor search is about finding the closest matches for high-dimensional vectors. Formally, the nearest neighbor search problem can be defined as follows:

Definition 2.1 *Given a set of points $P = \{p_1, ..., p_n\}$ in a vector space X, and a new point $q \in X$, find the closest point in P to q.*

The vector space X is usually the Euclidean vector space. For high-dimensional spaces, it is common that there does not exist any known algorithm faster than simple linear search, which is too slow for many applications. Therefore, approximate algorithms which are much more efficient, but not always returns the optimal result, are often used. Applications that can take advantage of nearest neighbor search include finding matches for local image features between two frames (i.e. find which pixel/patch in one frame that best matches a pixel/patch in another frame) as well as machine learning, data compression and document retrieval [20].

2.3 Color theory

Brightness is the amount of visible light from a point on an object, traveling in a specific direction [21]. The perceived relative brightness is called *luminance* [15]. Independent of its luminance, a point of an object also has a *chromaticity*, which consists of two independent parameters: hue and saturation. The hue decides which color the point has and the saturation decides how intense the color is, a low saturation means a more grayish color.

2.3.1 RGB color system

In a human eye, there are three kinds of color cone receptors, sensitive to different wavelengths of visible light and a fourth kind, rods, which are sensitive to luminance and not color in low light conditions. The peaks of the color cones, which can be seen in Figure 2.4, correspond to a blueish, greenish and reddish color, respectively, but as can be seen in the figure, there are large overlaps between the sensitive areas. This overlap in the human seeing makes it impossible to generate all colors that can be perceived by the brain by using only three additive components. Nevertheless, many color system is based on precisely three components [22].

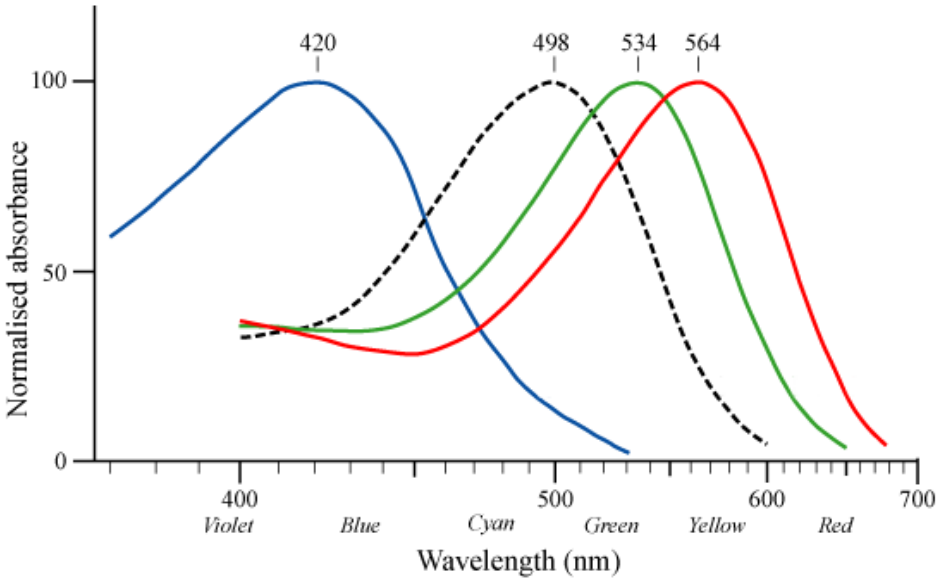


Figure 2.4: The sensitivity curves for the three color cone receptors (and the rods in black) in the human eye. Retrieved from `http://upload.wikimedia.org/wikipedia/commons/c/c2/Cone-response.png`.

Based on the ability of the human eye, the RGB color system consists of

the three components or channels, which are called R, G and B and corresponding to red, green and blue color. The RGB system also takes into account another property of the human eye, namely that it is more sensitive to luminance variation when the luminance is low compared with the same variation when the luminance is high. Thus, the R, G and B components are not linear in intensity, but instead scaled to be perceived as linear of the human eye. There exist various RGB color spaces with different scalings [22].

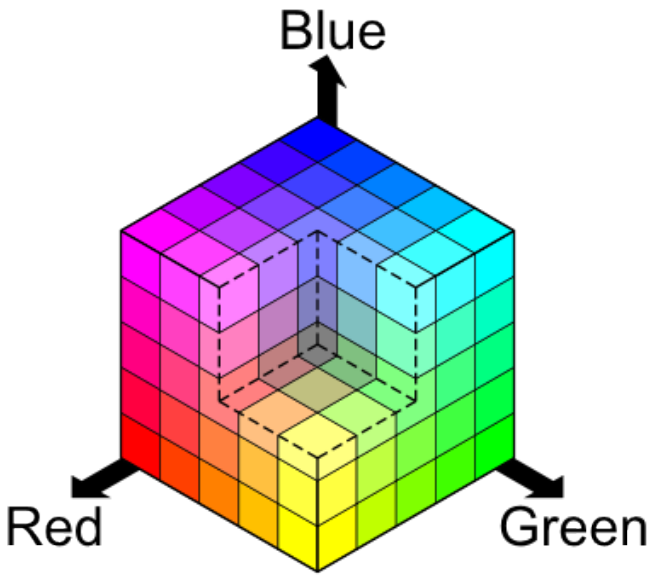


Figure 2.5: The RGB color cube. Retrieved from `http://commons.wikimedia.org/wiki/File:RGBCube_b.svg`.

The scale for the three components start at 0 and goes up to some common maximal value, typically 255 or in this thesis often 4095. Sometimes, the color components are normalized. Then, the maximal value is 1. The RGB color system is additive and a higher value of one component means more of that color. If a pixel's R, G and B components have the same value, it corresponds to grayscale. If all components are zero the pixel is black and if all components have the maximal value the pixel is white. The concept can be illustrated as in Figure 2.5.

2.3.2 CIELab color system

Except the often used RGB system, there are also several other color systems around. One of them, which is used in some of the algorithms considered in

this thesis, is the CIELab color system from 1976 [23]. The CIELab system is based on the fact that humans perceive difference in luminance and color in a roughly logarithmic way. CIELab has thus three non-linear components: L, a and b. The component L corresponds to the luminance or lightness, while a and b describes chromaticity [15]. A high a corresponds to more red color and a low a corresponds to a more green color, while a high b corresponds to a more yellow color and a low b to a more blue color, as can be seen in Figure 2.6.

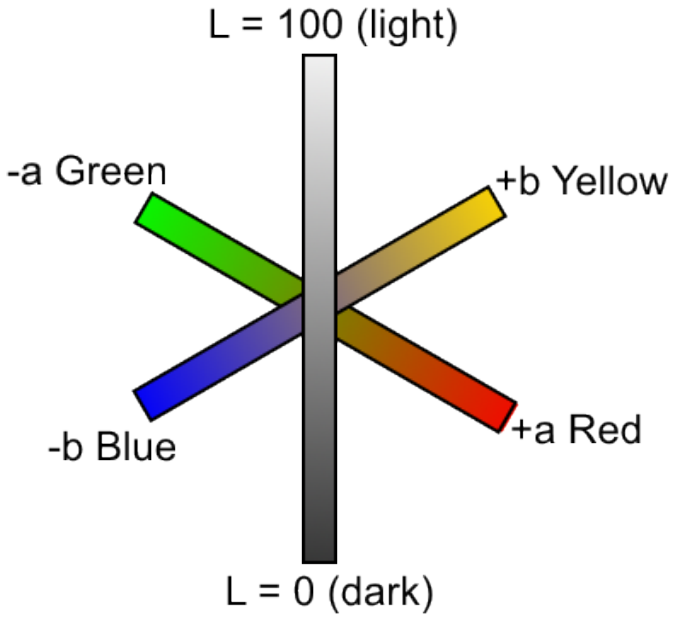


Figure 2.6: A graphical explanation of the CIELab color system.

2.3.2.1 RGB to CIELab conversion

Consider a certain pixel from an image frame and normalize its RGB values to be in the interval $[0, 1]$ and call the result R, G and B, respectively. Define the function f as

$$f(t) = \begin{cases} t^{1/3} & \text{if } t > 0.008856 \\ 7.787 \cdot t + 16/116 & \text{otherwise} \end{cases} . \tag{2.8}$$

Let X, Y, Z be

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} . \tag{2.9}$$

Finally, let X_n, Y_n, Z_n be the X, Y and Z corresponding to the reference whingite, in the implementation in this thesis the X, Y and Z corresponding to $R = G = B = 1$ are used.

Then, the CIELab values of the current pixel can be calculated as [23]

$$\begin{aligned} L &= \begin{cases} 116 \cdot (Y/Y_n)^{1/3} - 16 & \text{if } (Y/Y_n) > 0.008856 \\ 903.3 \cdot Y/Y_n & \text{otherwise} \end{cases} \\ a &= 500 \cdot (f(X/X_n) - f(Y/Y_n)) \\ b &= 200 \cdot (f(Y/Y_n) - f(Z/Z_n)) \end{aligned} \quad . \tag{2.10}$$

2.3.2.2 CIELab to RGB conversion

Assume that the pixel currently considered has the CIELab color values L, a and b. Again, let X_n, Y_n, Z_n correspond to the reference white and let

$$P = \frac{L + 16}{116} \ .$$

Then

$$\begin{aligned} X &= X_n \cdot (P + a/500)^3 \\ Y &= Y_n \cdot P^3 \\ Z &= Z_n \cdot (P - b/200)^3 \end{aligned}$$

and

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.240479 & -1.537150 & -0.498535 \\ -0.969256 & 1.875992 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \ . \tag{2.11}$$

With the formulas in Equation (2.11), some of the produced normalized RGB values may be outside the range $[0, 1]$. Negative values are truncated to 0 and values greater than one are truncated to 1 [23].

2.4 Noise

After passing through the optics (lenses) of a camera, the incoming light reaches an image sensor. During the exposure time, light photons are converted to electrons, accumulated in the sensor pixels [15]. The full well capacity (in the rest of the report called only full well) is the maximum amount of electrons that can be accumulated in each sensor pixel. Provided that the exposure is adjusted so that the brightest spots in the image just fill their pixels, a lower full well leads to a lower dynamic range and a lower signal-to-noise ratio [24]. During the sensing process different kinds of noise are added, for instance fixed pattern noise, quantization noise and shot noise [15]. The temporal system noise occurring in absence of an input signal is called read noise [25]. Before the analog-to-digital conversion of the electrons, the signal is amplified with a factor, called gain. In theory, a higher gain leads to a better performance in low light conditions, but it is not only the signal, but also some of the sensor noise sources that become amplified [15].

The noise model used in this thesis includes photon shot noise and read noise. The signal is normalized to be in the interval $[0, 1]$, where 0 corresponds to black and 1 corresponds to white. In order to get the unit electrons, the signal is multiplied with the full well, giving

$$\text{signal\_electrons} = \text{signal} \cdot \text{full well} \ . \tag{2.12}$$

The photon shot noise is called PSN and is modeled as a Gaussian function (see Equation (2.2)) with mean 0 and standard deviation

$$\sigma_{\text{psn}} = \sqrt{\text{signal\_electrons}} \ .$$

The read noise, called RN is also modeled as a Gaussian function with mean 0, but it has the standard deviation σ_{read}. The σ_{read} parameter has been set to 5 in all experiments in this thesis. The noisy signal is then given by the sum of the signal and the noise, i.e.

$$\text{noisy\_signal\_electrons} = \text{signal\_electrons} + \text{PSN} + \text{RN} \ . \tag{2.13}$$

The final image should have intensities in the interval $[0, 4095]$. To accomplish this, the final signal is calculated as

$$\text{noisy\_signal} = 240 + \frac{\text{noisy\_signal\_electrons} \cdot (4095 - 240)}{\text{full well}} \ , \tag{2.14}$$

where 240 is called black level. Due to the noise the signal can still be outside the wanted interval. In that case, pixel values below 0 are set to 0 and pixel values above 4095 are set to 4095 [26].

2.4.1 Measurements

There exists several different metrics and methods to measure the amount of noise in an image. Three of them are MSE (mean square error), PSNR (peak signal-to-noise ratio) and SSIM (structural similarity).

2.4.1.1 MSE

The mean square error is defined as

$$\text{MSE} = \frac{1}{n}\sum_{x}\left[I(x) - \hat{I}(x)\right]^2 \quad , \tag{2.15}$$

where n is the number of pixels, $I(x)$ is the original image and $\hat{I}(x)$ is the noisy image [15].

2.4.1.2 PSNR

Peak signal-to-noise ratio is defined as

$$\text{PSNR} = 10\log_{10}\frac{I_{\text{max}}^2}{\text{MSE}} \quad , \tag{2.16}$$

where I_{max} is the maximal color value (4095 for 12 bit images) [15].

2.4.1.3 SSIM

Structural similarity is a metric based on perceptual similarity. The metric uses the luminance, contrast and structure of the image. A 11×11 Gaussian kernel (see Section 2.2) with standard deviation 1.5 and normalized to unit sum is used as a weighting function in SSIM. For an image I of N pixels, let $w = \{w_i | i = 1, 2...N\}$ be the weighting function for each pixel. The luminance for the image is then estimated by

$$\mu_I = \sum_{i=1}^{N} w_i I_i \tag{2.17}$$

and the contrast is estimated by the standard deviation

$$\sigma_I = \left(\sum_{i=1}^{N} w_i (I_i - \mu_I)^2\right)^{1/2} \quad . \tag{2.18}$$

The luminance comparison, $l(I, J)$, between two images I and J is defined as the comparison between μ_I and μ_J

$$l(I, J) = \frac{2\mu_I\mu_J + c_1}{\mu_I^2 + \mu_J^2 + c_1} \quad , \tag{2.19}$$

where c_1 is a constant to avoid singularity, when $\mu_I^2 + \mu_J^2$ is close to zero. The contrast comparison, $c(I, J)$, between two images I and J is in a similar way defined as the comparison between σ_I and σ_J

$$c(I, J) = \frac{2\sigma_I\sigma_J + c_2}{\sigma_I^2 + \sigma_J^2 + c_2} \ , \tag{2.20}$$

where c_2 is a constant to avoid singularity, when $\sigma_I^2 + \sigma_J^2$ is close to zero. The structural comparison, $s(I, J)$, between two images I and J is defined as the correlation between $(I - \mu_I)/\sigma_I$ and $(J - \mu_J)/\sigma_J$, which is equal to the correlation coefficient between I and J

$$s(I, J) = \frac{\sigma_{IJ} + c_3}{\sigma_I\sigma_J + c_3} \tag{2.21}$$

with

$$\sigma_{IJ} = \sum_{i=1}^{N} w_i(I_i - \mu_I)(J_i - \mu_J) \ , \tag{2.22}$$

where c_3 is a constant to avoid singularity, when $\sigma_I\sigma_J$ is close to zero. Now a similarity function, $S(I, J)$ is created as

$$S(I, J) = f(l(I, J), c(I, J), s(I, J)) \ . \tag{2.23}$$

For SSIM the function f is defined as

$$f(l(I, J), c(I, J), s(I, J)) = l(I, J)^\alpha \cdot c(I, J)^\beta \cdot s(I, J)^\gamma \tag{2.24}$$

where $\alpha > 0$, $\beta > 0$ and $\gamma > 0$ are parameters to modify the relative importance. To simplify the formula, α, β and γ is set to one and $c_3 = c_2/2$. It gives

$$\text{SSIM}(I, J) = \frac{(2\mu_I\mu_J + c_1)(2\sigma_{IJ} + c_2)}{(\mu_I^2 + \mu_J^2 + c_1)(\sigma_I^2 + \sigma_J^2 + c_2)} \ . \tag{2.25}$$

SSIM has the following properties

1. Symmetry: $\text{SSIM}(I, J) = \text{SSIM}(J, I)$
2. Bounded: $\text{SSIM}(I, J) \leq 1$
3. Unique maximum: $\text{SSIM}(I, J) = 1 \Leftrightarrow I = J$

The constants, c_1 and c_2, are normally set to $c_1 = (K_1L)^2$ respective $c_2 = (K_2L)^2$ where L is the maximum color value (4095 for 12 bit images). K_1 and K_2 should be chosen to small values. In this thesis K_1 has been set to 0.01 and K_2 has been set to 0.03. All parameters are set according to the authors of SSIM [27].

2.5 Optical flow

Optical flow can either be defined as the apparent motion of brightness patterns in an image or as a motion field, which is the 2D projection of the 3D motion of objects and surfaces in the image. Most algorithms for determining optical flow estimates a 2D motion vector for each pixel in a frame from a video sequence containing of at least two frames [1], see Figure 2.7.

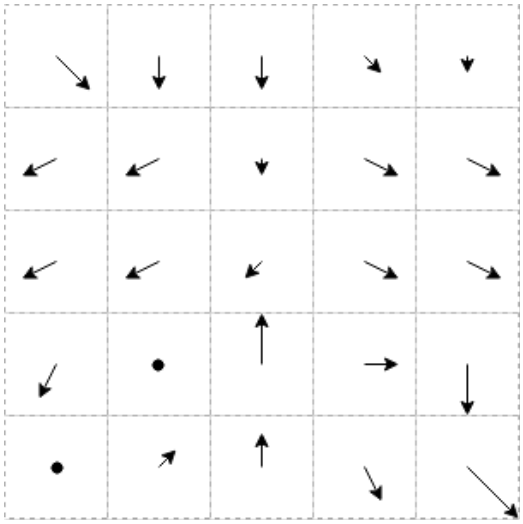


Figure 2.7: An example of an optical flow vector field, where each arrow represents how far and in which direction the current pixel have moved between two image frames.

A way to estimate the optical flow, used by many algorithms, is to model it as a global energy function (also called objective function [8]), which should be minimized. The energy equation is typically on the form

$$E_{Global} = E_{Data} + \lambda E_{Prior} \ , \tag{2.26}$$

where E_{Data} is called the data term and E_{Prior} the prior term [1] or the spatial regularity term [8]. Algorithms for calculating optical flow differs both in the exact formulation of the energy function and which optimization algorithm that is used to minimize it [8].

The data term in the energy function is often based on the *brightness constancy assumption* (BCA), which implies that the corresponding pixel values are assumed to be constant between frames [15]. The term contains

a robust penalty function for measuring the error of the chosen feature between the pixels. Some examples of penalty functions are the L1-norm, the L2-norm, the Lorentzian function $\rho(x) = \log(1 + \frac{x^2}{2\sigma^2})$, the quadratic HS penalty function $\rho(x) = x^2$ and the Charbonnier function $\rho(x) = \sqrt{x^2 + \epsilon^2}$ [8].

The prior term is based on how the optical flow varies across the image [8]. The *total variational* (TV) is most often used for the term. The TV for a function $f(\mathbf{x})$ over an interval $[a, b]$ is defined as

$$\mathrm{TV}(f(\mathbf{x})) = \sup \sum_{i=1}^{n} |f(x_i) - f(x_{i-1})| \ , \tag{2.27}$$

where $x_0 = a < x_1 < x_2 \cdots < x_n = b$ are any partition of points between a and b. The TV for one dimensional absolutely continuous functions is

$$\mathrm{TV}(f(\mathbf{x})) = \int_a^b |f'(x)| dx \tag{2.28}$$

and for two dimensions (x over $[a, b]$ and y over $[c, d]$)

$$\mathrm{TV}(f(\mathbf{x}, \mathbf{y})) = \int_a^b \int_c^d |\nabla f(x, y)| dx dy \ . \tag{2.29}$$

Like the data term, other norms could also be used for the penalty function [28, 29].

A classical formulation of the energy function is

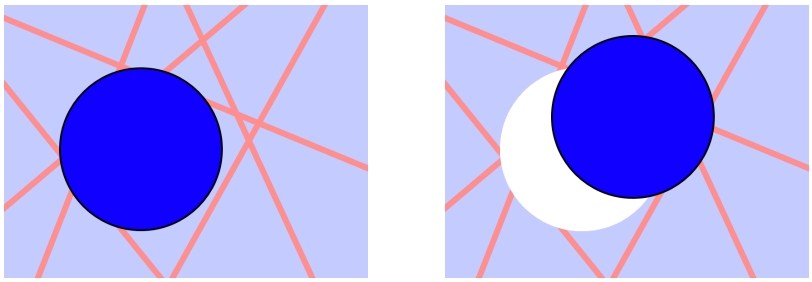
$$\begin{aligned} E(f_x, f_y) = &\sum_{i,j} \rho_D(I_1(i,j) - I_2(i + f_x(i,j), j + f_y(i,j))) \\ +\lambda\Big(&\rho_S(f_x(i,j) - f_x(i+1,j)) + \rho_S(f_x(i,j) - f_x(i,j+1)) \ . \\ &+\rho_S(f_y(i,j) - f_y(i+1,j)) + \rho_S(f_y(i,j) - f_y(i,j+1)) \Big) \end{aligned} \tag{2.30}$$

Here, I_1 and I_2 are the two image frames, f_x and f_y are the x- and y-component of the optical flow field to be estimated, ρ_D and ρ_S are the penalty functions and λ is a so called regularization parameter [30].

To deal both with large motions and thin structures moving fast, a coarse-to-fine estimation is often used in optical flow algorithms. The image is then downsampled to a coarser scale, on which the flow first is estimated. The flow field is then upsampled and refined until the original scale is reached [8].

2.5.1 Occlusions

A key concept in optical flow is the detection and handling of *occlusions*. An object or area, which is not visible in both image frames, between which the optical flow should be estimated, is considered occluded, see Figure 2.8. Occlusions can occur when objects move into or out of the image, when an object is moving in front of another object or when the camera is moving. Occlusion filling means to estimate flow in occluded areas, which are hard since the motion cannot be measured based on the two input frames. Occlusion filling is related to image inpainting, which can be solved by diffusion-based methods or exemplar-based methods [31]. Diffusion is most often used to solve occlusion filling. Partial derivative equations are used to propagate flow from non-occluded to occluded regions. The technique of exemplar-based methods is to copy pixels from observed images to the unknown part [31].



(a) Frame t (b) Frame $t+1$

Figure 2.8: The white area in image (b) symbolizes the occluded area for frame $t+1$.

2.5.2 Classical approaches

Optical flow algorithms can be divided into local and global methods. The local methods look at each pixel individually and estimate how it has moved locally to calculate optical flow. Global methods look at the full image and calculate the flow for all pixels at the same time [32].

The most classical global optical flow algorithm is Horn-Schunk [4] from 1980. It relies on the brightness constancy assumption. The algorithm assumes that neighboring pixels on an object move in the same way. In that way the algorithm has a smoothness constraint [4]. The global energy equation for Horn-Schunk can be written as

$$\begin{aligned} E &= \int \left(\frac{\partial I}{\partial x} f_x(p,t) + \frac{\partial I}{\partial y} f_y(p,t) + \frac{\partial I}{\partial t}\right)^2 \\ &+ \alpha^2(||\nabla f_x||^2 + ||\nabla f_y||^2) dp_x dp_y \end{aligned} , \tag{2.31}$$

where I is the intensity of the image, $f_x(p,t)$ and $f_y(p,t)$ is the flow in x- and y-direction from a pixel p at time t and the regularization parameter α decides the level of smoothness of the optical flow [32]. The function is solved by using the Euler-Lagrange equations, giving

$$\begin{cases} \frac{\partial I}{\partial x}\left(\frac{\partial I}{\partial x}f_x + \frac{\partial I}{\partial y}f_y + \frac{\partial I}{\partial t}\right) - \alpha^2 \Delta f_x = 0 \\ \\ \frac{\partial I}{\partial y}\left(\frac{\partial I}{\partial x}f_x + \frac{\partial I}{\partial y}f_y + \frac{\partial I}{\partial t}\right) - \alpha^2 \Delta f_y = 0 \end{cases} . \tag{2.32}$$

These equations are then solved by an iterative algorithm. A problem with Horn-Schunk is that discontinuities occur at the motion boundaries [32].

Kanade-Lucas is a local optical flow algorithm, which assumes that the flow is locally constant. Let $I(p,t)$ be the intensity and f_x and f_y be the flow in x- and y-direction for a certain pixel p at time t. For stability reasons, a small neighborhood window $\{p_1, p_2, ..., p_m\}$ is used. The system of equations to be solved to find the flow for pixel p then becomes

$$\begin{cases} \frac{\partial I(p_1,t)}{\partial x}f_x + \frac{\partial I(p_1,t)}{\partial y}f_y = -\frac{\partial I(p_1,t)}{\partial t} \\ \\ \frac{\partial I(p_2,t)}{\partial x}f_x + \frac{\partial I(p_2,t)}{\partial y}f_y = -\frac{\partial I(p_2,t)}{\partial t} \\ \vdots \\ \frac{\partial I(p_m,t)}{\partial x}f_x + \frac{\partial I(p_m,t)}{\partial y}f_y = -\frac{\partial I(p_m,t)}{\partial t} \end{cases} . \tag{2.33}$$

In order to solve this, often over-determined, equation system, the least square method is used. The solution then becomes

$$\begin{pmatrix} f_x \\ f_y \end{pmatrix} = -\begin{pmatrix} \sum_i \left(\frac{\partial I(p_i,t)}{\partial x}\right)^2 & \sum_i \frac{\partial I(p_i,t)}{\partial x}\frac{\partial I(p_i,t)}{\partial y} \\ \\ \sum_i \frac{\partial I(p_i,t)}{\partial x}\frac{\partial I(p_i,t)}{\partial y} & \sum_i \left(\frac{\partial I(p_i,t)}{\partial y}\right)^2 \end{pmatrix}^{-1} . \tag{2.34}$$
$$\cdot \begin{pmatrix} \sum_i \frac{\partial I(p_i,t)}{\partial x}\frac{\partial I(p_i,t)}{\partial t} \\ \\ \sum_i \frac{\partial I(p_i,t)}{\partial y}\frac{\partial I(p_i,t)}{\partial t} \end{pmatrix}$$

In comparison with Horn-Schunk, Kanade-Lucas produces a less smooth optical flow. On the other hand, it has not the same problem with discontinuities

as Horn-Schunk has [32]. Other formulations of the energy function includes formulations based on image segmentation and oriented smoothness [8].

2.6 Existing benchmark methods

Some of the newer benchmarks for ranking optical flow algorithms around are the Middlebury benchmark from 2011 [1], the MPI-Sintel benchmark from 2012 [2] and the KITTI benchmark, also this from 2012 [3]. Each benchmark has its own dataset for evaluation, split into a training set and a test set. For a comparison between these three datasets, see Table 2.1. All four optical flow algorithms implemented in this thesis (SimpleFlow, EPPM, Classic+NL-fast and LDOF) are listed in the MPI-Sintel benchmark [33], all except Classic+NL-fast in the Middlebury benchmark [34] and all except SimpleFlow are listed in the KITTI benchmark [35].

2.6.1 Middlebury

The benchmark Middlebury [1] has a dataset containing natural scenes. The dataset consists of 12 image sequences with available ground truth used for training (for instance parameter choices and debugging) and 12 image sequences for testing. One of the sequences called Yosemite comes from an earlier benchmark, created by Barron [10] in 1994. This sequence is included to make comparison with older algorithms possible. The Middlebury benchmark tests four types of data (examples of the types are shown in Figure 2.9):

- Scenes with non-rigid motion. These sequences are taken with a camera in the real world and contains different challenges, for instance shadows, low contrast and large areas without texture. All of them have small motions, typically about 10 pixels per frame and lack of motion blur. Hidden fluorescent texture, which can be captured by UV-light after the motions, is used for determining a dense ground truth flow. This method for finding ground truth will, however, encounter problems in the occluded areas.

- Realistic synthetic scenes with larger independent motions and complex occlusions. These scenes are generated by computer graphics, so it is easier to generate ground truth, especially for occluded pixels.

- High frame rate video sequences from the real world. The ground truth is an image captured between the two current frames. The algorithms are tested by creating an interpolation image using the estimated optical flow and the two frames and compare this image to the ground truth image. These sequences provide motion blur, as well as more complex motion and textures.

- Static scenes where the whole scene is moving horizontally [36].

Figure 2.9: Four of the data sequences from Middlebury benchmark: "Army" with non rigid motion (upper left), static scene "Teddy" (upper right), synthetic scene "Grove" (lower left) and high frame rate video "Backyard" (lower right).

The Middlebury benchmark measures four kinds of errors. The first is average angular error, which is the same error metric that Barron [10] uses. In this metric, the 3D angle between the estimated flow vectors and the ground truth vectors are measured, meaning small motions become more penalized than larger ones. The endpoint error is simply the Euclidean distance between the estimated flow vector and the ground truth vector. The third error metric is interpolation error, which is the root mean square between all pixels in the estimated image and the corresponding pixels in the ground truth image. The last is a normalized version of the interpolation error. These errors are calculated in three different types of areas: in all parts of the image (including occlusions) where ground truth flow could be reliably determined, in the parts around the motion discontinuities and in areas without texture. All of the errors are reported in several statistical ways as average, standard deviation and percentage of pixels with an error higher than a certain threshold, leading to many possible ways of ranking algorithms. The algorithms are required to generate a dense optical flow field in order to be ranked in the benchmark.

2.6.2 MPI-Sintel

The MPI-Sintel [2] benchmark has a 3D dataset, based on the animated short film Sintel, which is an entertainment movie in 24 fps (frames per second) and not created specifically for research. The dataset contains 35 sequences from Sintel, divided in 23 sequences for the training set and 12 for the test set. Most of these video sequences consist of 50 frames. Sintel is somewhat specialized on large motions with both small and large objects moving fast with a maximum flow of over 100 pixels per frame. Some other challenges are specular reflections, motion blur and defocus (objects which are not in focus) . Since Sintel is animated, not all physical laws are followed, but things like luminance, spatial power spectra and flow have been compared to real world images with good result. The movie is considered close enough to reality to tell how well the optical flow algorithms will perform in the real world. An advantage with an animated dataset is that it is easy to measure the exact motion between two frames and thus create a perfect ground truth, something that no current image sensor is able to do in an accurate way.

In the MPI-Sintel dataset, it is possible to choose between three render settings for each sequence in order to test different types of conditions (for an example image see Figure 2.10):

- Albedo: Contains no illumination, thus the brightness constancy assumption is fulfilled, except in occluded regions.
- Clean: Introduces illumination changes and reflections.
- Final: The rendering from the real film. Contains motion blur, atmospheric effects and much more.

The algorithms tested by the people behind MPI-Sintel (Classical+NL [8], Classical++ [8], Classic+NL-fast [8], HS [4], LDOF [9] and H-L1 [37]) generally perform best on Clean, while they have problems with Albedo, which contains large homogeneous regions not suitable for optical flow algorithms. Most challenging for most of the algorithms is the Final render setting [2].

The MPI-Sintel benchmark measures different kinds of errors: average endpoint error (EPE), error for pixels that are moving in a certain speed or having a certain distance to occlusions and error in matched and unmatched regions. Unmatched regions, consisting of nearly 8.5% of all pixels, are the parts of the image which are shown in only one of the two frames.

(a) Albedo

(b) Clean

(c) Final

Figure 2.10: The three render settings available in MPI-Sintel on the same image from the training sequence bandage_1.

Some advantages for the MPI-Sintel dataset compared to the Middlebury dataset [1] is that MPI-Sintel has longer sequences and larger training sets, meaning many more ground truth fields and a possibility of testing algorithms based on machine learning. MPI-Sintel also offers higher resolution, larger motions and more complexity. Algorithms which performs well on the Middlebury benchmark with an average endpoint error of 0.5 pixels can do as bad as an error of about 10 pixels on the MPI-Sintel benchmark [2].

2.6.3 KITTI

KITTI [3] is a benchmark used not only for optical flow, but also for stereo matching, measurement of distances in images and 3D object detection. The images for the datasets have been captured from an autonomous driving platform, equipped with four high resolution video cameras (out of which two are grayscale cameras and two are color cameras), a laser scanner, an advanced localization system and a powerful computer. Currently, only the images from the grayscale cameras are used because they are of better quality than the color ones. The car has driven in several areas: both on highways, in a mid-sized town and in the countryside. A typical image captured by the driving platform can be seen in Figure 2.11. In total, about 3 TB of material has been filmed. In order to get a diverse optical flow dataset representing many different kinds of motions, k-means clustering has been used on the material to pick 400 image pairs. Eleven of them has been omitted due to bad illumination condition, leaving 194 image pairs for the training set and 195 for the test set. The challenges in the final dataset include specular reflections and transparent surfaces, large variety of materials, large displacements and varied light conditions.

Figure 2.11: A typical image from the KITTI dataset.

The flows produced by the optical flow algorithms are compared to a ground truth, which has been determined by using the instruments of the driving platform and five frames before and five frames after the frame of interest. Ambiguous image regions, such as windows and fences have been removed during the calculations. No interpolation of the ground truth has been performed, leading to the average ground truth density for KITTI being only about 50% [3]. The optical flow algorithms are evaluated using the average disparity and end-point error as well as the number of incorrect pixels. Two types of evaluations are performed: one with all pixels that have ground truth and one where occluded regions are omitted. KITTI uses an error threshold of between two and five pixels, where three pixels are used for the main ranking [35].

In comparison with the Middlebury dataset [1], the KITTI dataset has less risk of overfitting because the dataset contains a lot more image pairs: 389 instead of 24. The images from KITTI also have a higher resolution (0.5 Megapixels compared to 0.2 Megapixels). On the KITTI benchmark, methods using the classical variational approach usually perform best, but they have problems with large displacements, occurring when the car travels at high speed. Methods that perform very well on the Middlebury benchmark, typically performs below average on the KITTI benchmark. This can be explained by the fact that, while the Middlebury dataset contains quite small motions performed in a controlled environment, the KITTI dataset only has real video sequences where the camera is moving, potentially leading to a high optical flow in the entire image [3].

	Middlebury	MPI-Sintel	KITTI
Number of sequences in training set	12 [1]	23 [2]	194 [3]
Number of sequences in test set	12 [1]	12 [2]	195 [3]
Number of frames in each sequence	mostly 8, but 2 for three of the sequences [36]	mostly 50 [2]	2 [3]
Resolution	Between 316×252 and 640×480 [36]	1024×436 [2]	about 1240×376 [3]

Table 2.1: Comparison between the three datasets for the benchmarks Middlebury, MPI-Sintel and KITTI.

Chapter 3

Method

3.1 Choosing algorithms

In order to choose which algorithms to focus on, the MPI-Sintel benchmark [2] was considered. All algorithms ranked in the benchmark, for which a paper or a similar source of information was present, were briefly investigated. Then, the algorithms were classified after which approach they use to estimate the optical flow. Some of them have a specific approach, very different from all other algorithms. These were not considered, instead algorithms representing frequently used approaches (patch matching, approximate, energy equation minimization, gradients and descriptors) were chosen. The aim of this classification was to be able to answer the research question about what kind of method that generates the best result for noise filtering (see Section 1.2.1).

During the selection, algorithms with too long execution time were omitted due to the real time limitation, mentioned in Section 1.3. Admittedly, the execution time for a given software implementation, for instance in MATLAB does not necessarily says very much about the execution time in a future hardware implementation in a camera. Nevertheless, the execution time reported on the benchmarks are the best available source about how fast different optical flow algorithms are and have therefore been used to do a rough discernment of the algorithms.

Algorithms with available code were preferred over other algorithms because they would be easier to implement. In the end, four algorithms were selected: SimpleFlow [6], EPPM (Fast Edge-Preserving PatchMatch) [7], Classic+NL-Fast [8] and LDOF (Large Displcement Optical Flow) [9].

SimpleFlow is based on color consistency and patch matching. EPPM is an approximation algorithm which also is based on patch matching, but with a more advanced matching procedure than SimpleFlow. Classic+NL-

Fast is a global method, which is based on the energy equation and related to Horn and Schunck [4]. It uses color consistency and smoothing and median filtering of the optical flow. LDOF is also based on the energy equation, color consistency and smoothness of the flow, but it also contains gradient consistency and descriptors.

First, SimpleFlow and EPPM, which both are among the fastest algorithms on the benchmarks, were implemented. After this the decision to use the available code for LDOF and Classic+NL-Fast was made.

3.2 Implementation and parameter choices

EPPM and SimpleFlow have been implemented in a framework developed by Axis, called Algsim. Each part of an algorithm implemented in Algsim consists of a block with input and output ports and a number of parameters. Several instances of each block can be created and these instances can be connected by their input and output ports via either code or a graphical flow chart. The interaction part of the framework, with the connections, input parameters, etc. is coded in Python. Almost all other code is written in a cross-platform framework called OpenCL (Open Computing Language), which can run calculations in parallel on the GPU (Graphics Processing Unit) [38]. The programming language used are formally called OpenCL C and is a version of C, which is extended to handle parallelization, but in some other ways restricted [38].

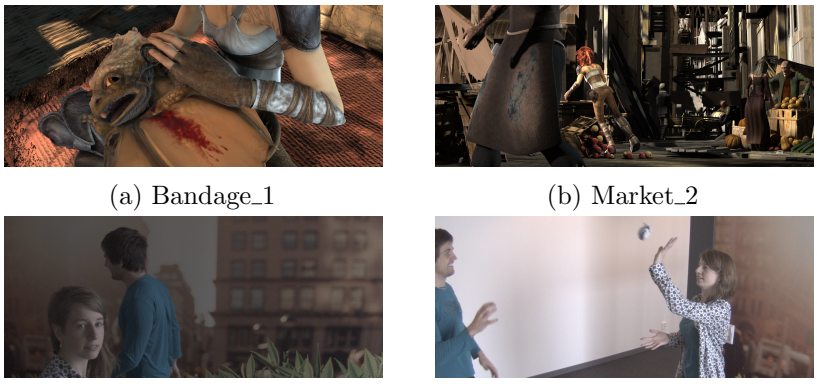
Both SimpleFlow and EPPM had available code, but the code for EPPM was already compiled, so the source code could not be viewed and the implementation is, therefore, based only on the paper about EPPM [7] and other papers and not on any existing code. The first step in implementing EPPM was to read the EPPM paper [7] and some of the papers it refers to it. The point of this was to get a general understanding of the algorithm. After this, EPPM was implemented part by part in Algsim.

SimpleFlow was implemented in parallel with EPPM. The present code is divided into two parts. One part for pre-processing (MATLAB code) and one part for optical flow caluclation (C++ code). The first step was to translate the MATLAB code which uses several built-in functions, into parallelized similar OpenCL C code. The next step was to translate the C++ code into OpenCL C code. Some of the C++ methods were parallelized by using shared resources, which cannot be done in an effective way in OpenCL. Therefore, a lot of the C++ code must be rewritten to take advantage of the parallelism available in OpenCL.

Most parameters of EPPM were taken from theory, while most parameters from SimpleFlow were from the code that the implementation is build on. The implementations of EPPM and SimpleFlow were concluded with a parameter analysis, where many of the parameters were varied independently of each other. Between each modification the result of the noise filtering (see Section 3.4) on a part of the image sequence bandage_1 from MPI-Sintel [2] was measured. If the modification led to an improvement of the result, it was kept, otherwise it was discarded. The specific parameter choices are presented and motivated in Section 4.1 for SimpleFlow and Section 4.2 for EPPM. The parameters of LDOF and Classic+NL-Fast has not been analyzed at all, but all parameter choices made in the MATLAB implementations have remained unmodified.

3.3 Choosing test images

During the implementation of SimpleFlow and EPPM, several image sequences from the MPI-Sintel dataset were used for testing the algorithms. The focus was on the render setting ”clean”, which most optical flow algorithms handle best (see Section 2.6.2). For the analysis, two image sequences from this dataset were picked: ”bandage_1” and ”market_2” (see Figure 3.1), both from the training part of the dataset. These sequences were picked because they had least motions in the background of all image sequences from MPI-Sintel. Bandage_1 has minimal background motion, while market_2 has some rotation in the background.



(a) Bandage_1 (b) Market_2

(c) Real_1 (d) Real_2

Figure 3.1: The four image sequences used in the thesis.

In addition to the image sequences from MPI-Sintel, two more sequences were used for the evaluation. These image sequences, called "real_1" and "real_2" (see Figure 3.1), were captured at Axis Communication. The goal of recording own image sequences was to get more realistic material with people moving in front of a static background, which is a typical surveillance situation. In total, over 6000 image frames where captured in 25 fps (frames per second). The material was then divided into shorter sequences based on the events and motions appearing in the video. These shorter sequences were then searched through to find sequences with interesting motions and reasonable length. Two sequences were chosen. They had the advantage that they could be looped because they ended in almost exactly the same way as they started.

3.4 Analysis of different noise levels

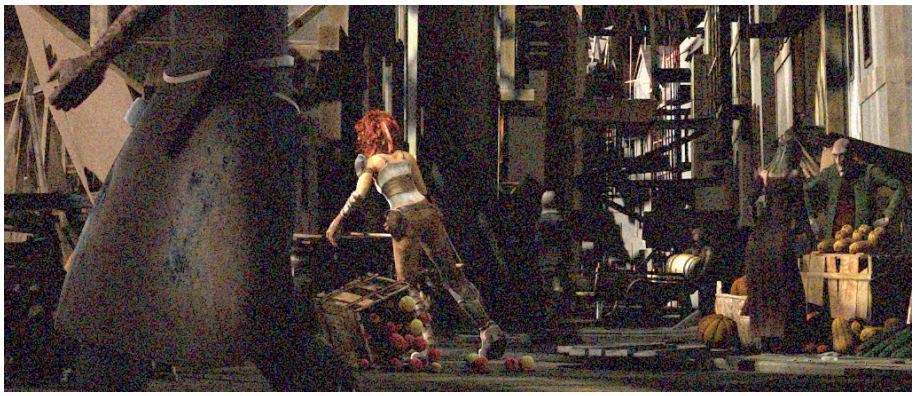
In the sensor simulator in Algsim, the full well and read noise standard deviation σ_{read}, described in Section 2.4 could be set to generate noisy image sequences. During the thesis, a decision was made: to test the algorithms on several noise levels to see where each algorithm failed to produce optical flow useful for noise reduction. On each noise level the algorithms were tested both with and without the pre-processing described in Section 3.4.1.

The noise levels were initially set to full well 200, 400, 600, 800 and 1000 electrons, respectively. The parameter σ_{read} was set to 5 electrons for all noise levels [26]. While all algorithms performed relatively good even on the noisiest image sequence (full well 200 electrons) from the MPI-Sintel dataset, noisier sequences were needed. After experiments the new noise levels full well 50 electrons and full well 100 electrons were added. Due to the amount of time for evaluating all optical flow algorithms on all noise levels, both with and without pre-processing, seven noise levels were regarded as too many and two of the initial noise levels were removed: full well 1000 electrons because it did not correspond to very much noise and full well 600 electrons because it was quite similar to adjacent levels. The final choice of noise levels were chosen logarithmic scale with full well 50, 100, 200, 400 and 800 electrons.

3.4.1 Pre-processing

In this thesis, the optical flow algorithms have been analysed for pre-processed respective non pre-processed noisy image sequences. The idea is to investigate if the algorithms performs better if the noise is smoothed out. The pre-processing uses a bilateral filter (see Section 2.2.1.1) with a kernel size of 11×11 and $\sigma_s = 8$. Depending on the noise level, different values for σ_r is used. For images with full well 50, 100, 200 and 400 electrons $\sigma_r = 70$ is

used, and for images with full well 800 electrons $\sigma_r = 40$ is used. Figure 3.2 shows an example of a pre-processed image.



(a) Noisy image

(b) Pre-processed image

Figure 3.2: The pre-processed image for full well 200 electrons on frame 15 from the image sequence market_2.

3.4.2 Noise filter

Two noise filters are implemented for the analysis of denoising using optical flow. One of the filters is called temporal filter and does not use information about optical flow. The temporal filter is, therefore, used as a reference for a noise reduction without optical flow. The second filter is called optical flow noise filter and is similar to the temporal filter, but with the difference that it uses optical flow estimations (see Section 4.5 for more details about the filters).

3.4.3 Test framework

The test framework to perform noise filtering with optical flow algorithms consists of seven steps. Figure 3.3 shows how the steps are connected. The seven steps are

1. Raw image: Generate raw images from the sequences of the dataset described in Chapter 5.

2. Add noise: Add noise on the image sequences. The different noise levels are described in Section 3.4.

3. Gamma correction: The human eye can see a much wider range of brightness and colors than can be shown on a screen with limited range. This step tries to reproduce this visibility and increase the contrast of the image [39]. One reason for using this step is that it was an early attempt to spread out the noise over all intensity levels in the images. Furthermore, the optical flow algorithms are developed to be used for the kinds of sequences that the benchmarks offer, which are gamma corrected.

4. Pre-processing: Perform pre-processing on the image sequences. The pre-processing is described in Section 3.4.1.

5. Optical flow calculation: A specific algorithm is used to estimate the optical flow between two images. The different optical flow algorithms are described in Chapter 4.

6. Noise filter: The optical flow filter described in Section 4.5.2 is used for noise reduction. The filter consists of an occlusion algorithm which both finds occlusions (finder step) and tries to remove incorrect flow (detect, remover and filler steps). In order to compare the optical flow noise filter with a non optical flow filter, a temporal filter (see Section 4.5.1) is used in this step to get an alternative noise reduction.

7. Noise measurements: SSIM and MSE (see Section 2.4.1) are used to measure the amount of noise for the filtered images from step 6.

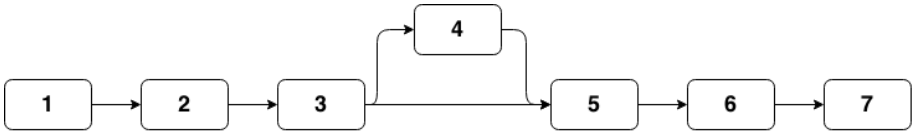


Figure 3.3: The main structure of the test framwork. 1: Raw image, 2: Add noise, 3: Gamma correction, 4: Pre-processing (optional), 5: Optical flow calculation, 6: Noise filter, 7: Noise measurements.

Chapter 4

Algorithms and implementation

This thesis focuses on four different optical flow algorithms; Classical NL-Fast, SimpleFlow, EPPM and LDOF. Two of them, SimpleFlow and EPPM, are implemented in a GPU-based framework (see Section 3.2). The remaining two algorithms have source code in MATLAB and were decided not to be implemented in the framework. The details of the different algorithms are explained in respective section below.

4.1 SimpleFlow

SimpleFlow is a non-iterative optical flow algorithm with sublinear time complexity with respect to the number of pixels. In contradiction to most of the state-of-art algorithms this algorithm does not use global optimization, but instead only local methods. SimpleFlow is developed to be a fast algorithm which can manage high resolution images in a reasonable amount of time. It uses a multiscale approach whereat each pixel is processed independent of the others at each scale. In that way the algorithm can use parallelization to reduce the calculation time.

First a single scale version of the algorithm is going to be discussed and thereafter a multiscale version. Figure 4.1 and 4.2 present the main structure of the multiscale version of the algorithm.

4.1.1 Single scale approach

The single scale approach is based on two assumptions; color consistency and locally smooth flow. The assumptions lead to a minimization problem for each pixel. The assumptions are described in the next two sections.

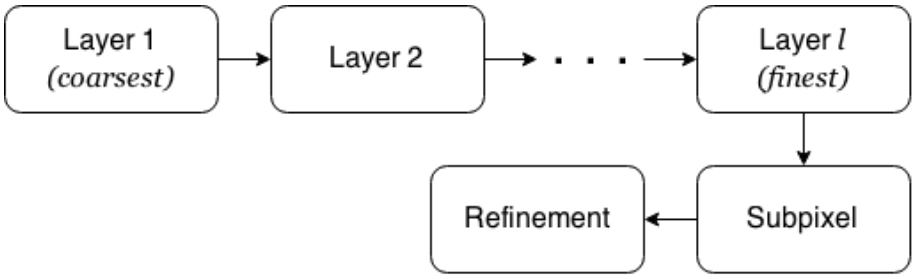


Figure 4.1: The flow chart for the multiscale approach for SimpleFlow with l layers. For each layer an initial optical flow is estimated by upsampling the optical flow from the previous layer. For the first layer, the initial optical flow is set to zero for all pixels. A subpixel estimate and refinement are performed on the optical flow at the end.

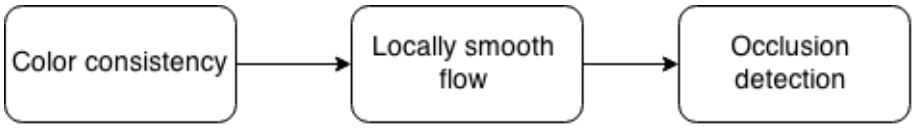


Figure 4.2: The flow chart for a layer of the multiscale approach for SimpleFlow. The algorithm first performs color consistency calculations followed by calculations for a locally smooth optical flow and at the end occlusion detection.

4.1.1.1 Color consistency

The model for the color consistency assumption is given by

$$d(x, y, u, v) = ||F_t(x, y) - F_{t+1}(x + u, y + v)||^2 \ , \tag{4.1}$$

where $F_t(x, y)$ is the RGB color of the pixel (x, y) in frame t and d is the square difference between two RGB vectors. The likelihood probability $p \propto e^{-d}$ is used in the algorithm for the color consistency calculations.

4.1.1.2 Locally smooth flow

The optical flow is assumed to be locally smooth. If the optical flow in a specific pixel is (u_0, v_0), then the assumption says that it is also a good approximation of the optical flow in its neighborhood N_0. In the implementation N_0 is the 9×9 area around the pixel. The assumption is expressed by

$$(u_0, v_0) = \arg\max_{(u,v)\in\Omega} \prod_{(x,y)\in N_0} p(x, y, u, v) \ , \tag{4.2}$$

where p is from the color consistency and $\Omega = \{(u, v)|u, v \leq a\}$. The parameter a is set to 2 in the implementation. To calculate Equation (4.2) is

the same thing as solving the negative-log-likelihood equation

$$(u_0, v_0) = \underset{(u,v)\in\Omega}{\arg\min} \sum_{(x,y)\in N_0} d(x, y, u, v) \ . \tag{4.3}$$

4.1.1.3 Weights and minimization

Equation (4.3) has the problem that it does not take edges into account. This can be solved by adding weights to the equation. Let

$$w_d = e^{-\frac{||(x_0, y_0) - (x, y)||^2}{2\sigma_d^2}} \tag{4.4}$$

be a weight for the distance difference and

$$w_c = e^{-\frac{||F_t(x_0, y_0) - F_t(x, y)||^2}{2\sigma_c^2}} \tag{4.5}$$

be a weight for the color difference. This results in the equation

$$E(x_0, y_0, u, v) = \sum_{(x,y)\in N_0} w_d w_c d(x, y, u, v) \ . \tag{4.6}$$

The parameters σ_d and σ_c is set to 4.1 respective 25.5 in the implementation. Equation (4.6) is the equation SimpleFlow minimizes to find the best approximation for the optical flow in pixel (x_0, y_0).

4.1.1.4 Occlusion detection

SimpleFlow calculates both the backward and the forward optical flow between two frames. Occlusions can then be detected by comparing the backward and forward optical flow. An occlusion threshold is used to determine if a given pixel is occluded or not.

4.1.1.5 Subpixel estimate and regularization

After Equation (4.6) has been minimized, (u_0, v_0) is achieved. Both u_0 and v_0 are integers. A 3×3 parabola with center at (u_0, v_0) is used to get a subpixel estimate of (u_0, v_0). One more regularization is made by SimpleFlow after that. It uses a bilateral filter with w_d and w_c from above, but also

$$w_r(x, y) = \underset{(u,v)\in\Omega}{\text{mean}}\, d(x, y, u, v) - \underset{(u,v)\in\Omega}{\min} d(x, y, u, v) \ . \tag{4.7}$$

The weight $w_r(x, y)$ indicates how reliable the optical flow is at (x, y). In this last regularization step the occluded pixels are ignored.

4.1.2 Multiscale approach

The input frames are first downsampled l times each with a factor of 2. The coarsest scale uses the single scale approach described above in Section 4.1.1. The flow of the remaining $l-1$ layers are computed by using the optical flow from the previous layer.

4.1.2.1 Initial flow estimate

By upsampling the optical flow from the previous layer an estimate of the optical flow in the current layer is obtained. After that the single scale approach (see Section 4.1.1) is used to approximate the optical flow, but with one change. The change is that the neighborhood N_0 is now centered at $(x_0 + \bar{u}, y_0 + \bar{v})$ where $(\bar{u}, \bar{v})$ is the initial estimate.

4.1.3 Adaptive multiscale approach

The source code of SimpleFlow uses an adaptive multiscale approach to decrease the computational time. The idea is to find regions where the optical flow changes slowly and then interpolate in these regions instead of the more time-consuming minimization of the likelihood equation. Due to implementation problems of a thread based adaptive multiscale algorithm, this part is not implemented in this thesis GPU version of the algorithm.

4.2 EPPM

EPPM (Fast Edge-Preserving PatchMatch [7]) is a local optical flow algorithm based on PatchMatch [40] and specialized on large motions and speed, i.e. short execution time of the algorithm. It is among the fastest algorithms on the benchmarks sites with 0.25 seconds on Sintel and KITTI and 0.20 seconds on Middlebury. EPPM uses an approximate nearest neighbor field (NNF). The patch size is relatively large in order to increase the spatial smoothness, so that patches close to each other in one frame maps to positions close to each other in the next frame. This has some drawbacks. Motion boundaries are not that well-preserved and the speed of the algorithm becomes slower with a growing patch size. The first problem is addressed by creating an edge-preserving version of PatchMatch and the second one by using a fast randomized approximation algorithm when calculating the matching cost. Another problem with NNFs is that they do not handle occlusions, but for this aim an occlusion detection and filling step (see Section 4.2.5) is included in the algorithm. The main structure of the algorithm is presented in Figure 4.3. The different steps are explained and implementation issues and parameter choices are presented in the subsections below.

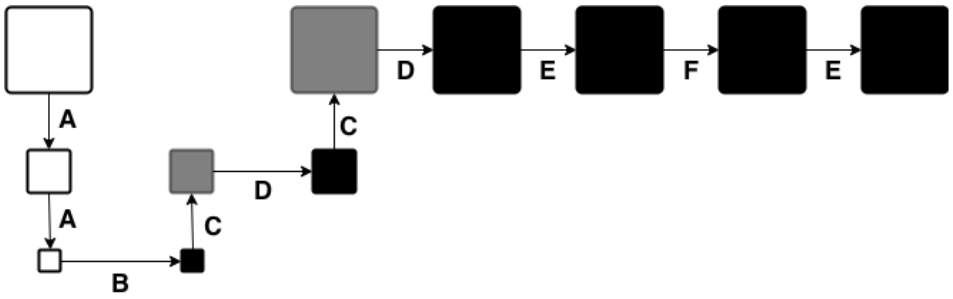


Figure 4.3: The main structure of the EPPM algorithm. White blocks are before any flow is calculated, gray blocks are when a sparse flow is present for the current resolution and black blocks when a dense flow is calculated. The steps are: A - Downsampling, B - PatchMatch, C - Joint bilateral upsampling, D - Local PatchMatch, E - Occlusion handling, F - Weighted median filtering. The algorithm is preferably concluded with some kind of refinement step, which is not present in this figure.

4.2.1 Downsampling

In order to speed the algorithm up, the images are first downsampled twice with a factor 2 in each dimension, leading to one sixteenth of the original resolution. While the authours of the EPPM paper [7] does not specify which downsampling method they use, the implementation in this thesis uses a standard downsampling method. Pixel (i, j) in the downsampled image will have the mean value of the four corresponding pixels in the high resolution image as pixel value. Mathematically, this can be written as

$$I_d(i,j) = \big(I_u(2\cdot i, 2\cdot j) + I_u(2\cdot i+1, 2\cdot j) + I_u(2\cdot i, 2\cdot j+1) + I_u(2\cdot i+1, 2\cdot j+1)\big)/4 \quad , \tag{4.8}$$

where I_d is the downsampled image and I_u is the high resolution original image. This calculation is performed for each color channel independently.

4.2.2 PatchMatch

The matching step of the algorithm is based on PatchMatch [40], which is a randomized matching algorithm attempting to find an approximate nearest neighbor match fast. First, pixels are matched randomly. An alternative to this would be to instead use prior information, as for instance the flow from the previous frames, as initial flow. Then, an iterative process follows in order to improve the matching. In each iteration (typically 4 - 5 iterations are used) the image is scanned either from the top left to the bottom right corner or in the opposite direction depending on which iteration the algorithm is in. Each pixel in the image first undergoes a propagation and then a random search. In the propagation step, the translations of neighboring pixels are

considered. Which neighbors that are used depend on the scan order. The translation, which leads to a minimal matching cost when a whole patch centered around the pixel is moved, is selected.

In the random search step, different translations of the form $u_i = v_0 + w\alpha^i R_i$ are investigated and the one with the lowest matching cost is chosen. Here, v_0 is the current translation, w is the maximum image dimension, α is a parameter set to 0.5, R_i is uniformly distributed random variable in the interval $[-1, 1] \times [-1, 1]$ and i is an integer starting at 0 and increased by one until $w\alpha^i \leq 1$ [40].

4.2.2.1 Self-similarity propagation

In the original PatchMatch [40], all pixels from the current patch are used in the calculation of the matching cost. While it can be shown that it is mainly pixels similar to the center pixel of a patch that contributes to the matching cost, an approximate matching cost can be determined by using only the n most similar pixels. Even if using fewer pixels speeds the calculation up, picking these n pixels out of a patch is too slow when the patch size is large. Therefore, EPPM offers a randomized algorithm for picking n pixels out of a patch, which are not necessarily the top n most similar pixels to the center one, but similar enough for giving an approximation of the matching cost.

For each pixel in both image frames a so-called *self-similarity vector* $S(x, y)$ is created, where x and y are the coordinates of the pixel. First, the vectors are filled with the positions of n random pixels from the patch which has the current pixel as center. Moreover, the similarity to the pixels, measured in CIELab color (see Section 2.3.2) is also stored in the vectors. Then, they are sorted after similarity. To improve the vectors, the frames are scanned twice, one time where each $S(x, y)$ is merged with $S(x-1, y)$ and $S(x, y-1)$ and one time when it is merged with $S_i(x+1, y)$ and $S_i(x, y+1)$ [7]. Based on suggestions from the EPPM paper [7], the patch radius is set to 17 and $n = 50$ is used.

4.2.2.2 Census transform

One feature that can be used for the data term in Equation (2.26) is the *census transform*. The idea behind the census transform is to transform the image and then find correspondences using correlation. The transform is looking at a neighborhood of a given pixel. It creates a bit string with the positions corresponding to pixels with smaller values than the given pixel set to 1 and the rest set to 0. The census transforms for different pixels in different frames are then compared. The difference between two pixels is calculated as the Hamming distance, i.e. the number of bits differing in the

two bit strings [41]. An example of this is shown in Figure 4.4.

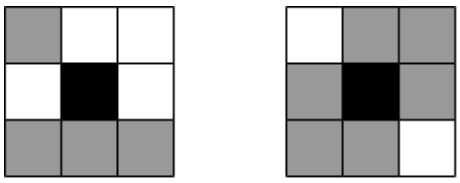


Figure 4.4: An example of a 3×3 census transform: gray blocks correspond to pixels with smaller values than the black center pixel and white blocks correspond to pixels with greater or equal pixel values. The left pixel will have the bit string "10000111" and the right pixel will have the bit string "01111110", leading to the Hamming distance 6.

4.2.2.3 Matching cost

The matching cost of matching a pixel a in the fist frame to a pixel b in the second frame, suggested in the EPPM paper [7], is

$$d(a,b) = \frac{1}{W} \sum_{\Delta} w(a,b,\Delta) C(a,b,\Delta) \ . \tag{4.9}$$

In Equation (4.9), Δ are the distances in x- and y-direction from pixel a to the neighbors in its self-similarity vector. The cost function consists of two terms, the bilateral weighting function $w(a,b,\Delta)$ and the robust cost function $C(a,b,\Delta)$ and a normalization factor $\frac{1}{W}$, where W is the sum of the weights. The bilateral weighting function is

$$\begin{aligned} w(a,b,\Delta) = \exp\left(-\frac{||\Delta||^2}{\sigma_s^2}\right) \exp\left(-\frac{||I^A(a+\Delta) - I^A(a)||^2}{\sigma_r^2}\right) \\ \cdot \exp\left(-\frac{||I^B(b+\Delta) - I^B(b)||^2}{\sigma_r^2}\right) \end{aligned} \ . \tag{4.10}$$

In the implementation, the weighting function was omitted due to the implementation of it giving a more random flow, leading to the reduced cost function

$$d(a,b) = \sum_{\Delta} C(a,b,\Delta) \ . \tag{4.11}$$

The robust cost function for a specific Δ is

$$\begin{aligned} C(a,b,\Delta) = \rho_i(C_{\text{AD}}(a+\Delta, b+\Delta), \lambda_{\text{AD}}) \\ + \rho_i(C_{\text{census}}(a+\Delta, b+\Delta), \lambda_{\text{census}}) \end{aligned} \ , \tag{4.12}$$

where C_{AD} is the mean value of the color differences measured in the CIELab color system and C_{census} is the difference in census transform (see Section 4.2.2.2) of the luminance [42]. The function ρ is a robust penalty function, used for rejecting outliers and balance between the terms of the cost function. In the implementation, the penalty function

$$\rho(x, \sigma) = 1 - \exp\left(-\frac{x^2}{\sigma^2}\right) \tag{4.13}$$

is used. The parameters are set to $\lambda_{AD} = 0.1$ and $\lambda_{census} = 0.5$ based on empirical tests and to make them balance the two terms in Equation (4.12).

4.2.2.4 GPU implementation issues

The propagation step in PatchMatch as well as the merging of the self-similarity vectors are in theory typical serial operations, but they are very slow when implemented in a serial way. For a fast parallel GPU implementation the authors of both EPPM and PatchMatch suggest using the Jump flooding algorithm [43] for these steps. This algorithm handles all pixels in parallel and runs several iterations where the distance to the neighbors used is halved in each iteration, see Figure 4.5. The initial distance is set to 8 for PatchMatch as suggested by the authors of PatchMatch [40] and to 16 for the self-similarity propagation, while vectors from further away will not improve the result when the patch radius is set to 17.

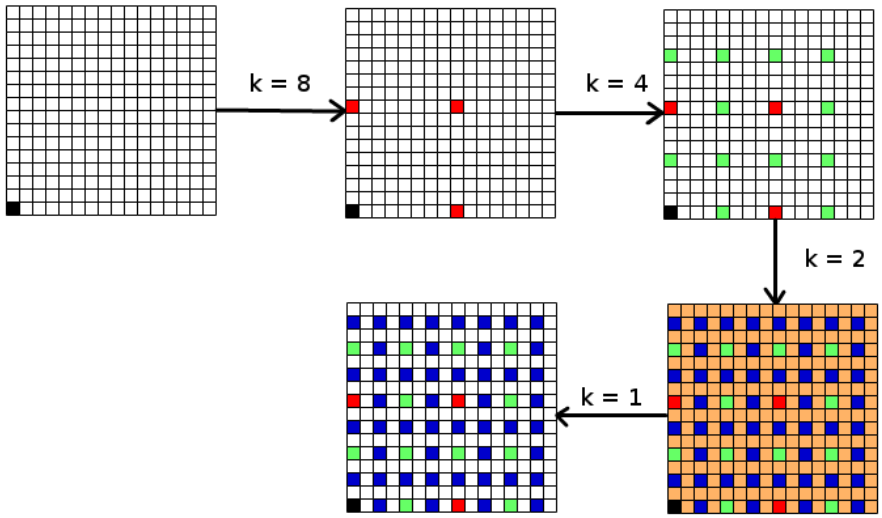


Figure 4.5: Jump flooding algorithm with initial distance $k = 8$.

4.2.3 Joint bilateral upsampling

The optical flow is upsampled twice with a factor 2 to get an optical flow on a higher resolution. The method used is joint bilateral upsampling (see Section 2.2.2.3) with the parameters $\sigma_s = 8.5$, $\sigma_r = 0.1$. The parameters are set empirically.

4.2.4 Local PatchMatch

After each of the two upsampling steps a 3×3 local patch match is performed in order to refine the optical flow [7]. The matching costs for the closest nine pixels to the current match are determined and the optical flow is modified to make the pixel with the lowest cost the new match. The self similarity vectors from Section 4.2.2.1 are upsampled and then again used when calculating the matching costs.

4.2.5 Occlusion handling

In order to find occlusions, the forward and backward optical flows are compared. The main idea is that if an unoccluded pixel in the first frame has a certain forward flow, the corresponding pixel in the second frame will have the opposite optical flow as backward flow. Assume that pixel a in the first frame is mapped to pixel b in the second frame according to its forward flow and that pixel b is mapped to pixel c in the first frame according to its backward flow. If the euclidean distance between a and c are greater than a threshold, which by experiment is set to 1, pixel a is considered occluded. Occluded pixels are filled with new flows by an algorithm, which searches through the nearby pixels and copies the flow from the unoccluded pixel with the lowest bilateral weight. If no such pixel is found, the flow is set to 0. When calculating bilateral weights, the parameters $\sigma_s = 8.5$ and $\sigma_r = 0.1$ are used just like before.

4.2.6 Weighted median filtering

In order to make the optical flow smoother and remove outliers, a weighted median filtering is performed on the flow field between the two iterations of occlusion detection and filling [7]. An edge preserving filter framework [44] with the parameters $\eta = 0.15$ and $N = 40$ is used. The former parameter is set empirically, while the number of intensity planes is set to 40 as a trade-off between accuracy and execution time. The weighted median filter used in the framework is a 7×7 filter which chooses the flow value β that minimizes $\sum_i w_i \cdot |x_i - \beta|$ where x_i is flow number i and w_i is the bilateral weight between the current pixel and pixel i [18]. The parameters σ_s and σ_r for the bilateral weights are the same as earlier and the size of the filter is set empirically as a trade-off between performance and execution speed.

4.2.7 Final refinement

The original EPPM is ended by a subpixel refinement step followed by a bilateral filtering with small parameters for smoothing out small outliers created in the refinement step [7]. The subpixel refinement is tricky to implement and heavy to run in OpenCL. Thus, the decision to skip the final refinement steps of EPPM in the implementation and instead use a modification of the smoothing from SimpleFlow was made.

4.3 Classic+NL-Fast

Classic+NL-Fast is one of several so-called Classic algorithms developed by Sun, Roth and Black [8]. It is implemented in MATLAB. The algorithm has a classical baseline approach based directly on the flow formulation from Hort-Schunk [4]. The baseline algorithm has then been systematically varied using modern implementation and optimizing techniques until a satisfying result is reached. During this process it was shown that only a few of the variations gave a statistically significant improvement, leading to the final algorithm becoming relatively simple, but still producing a good result [8].

4.3.1 Pre-processing

The image sequence is pre-processed using Rudin-Osher-Fatemi (ROF) structure texture decomposition. ROF is originally a noise removal algorithm [45], but can also be used to find textures [8]. The algorithm is based on nonlinear partial differential equations. The function that should be minimized uses the total variation norm. It can have an arbitrary number of constraints, both linear and nonlinear ones [45].

4.3.2 Multiscale approach

Classic+NL-Fast uses the standard multi-resolution technique, i.e. it works on several scales. The image is downsampled with a factor 2 recursively, step-by-step, until it has a width or height of about 20 - 30 pixels. The flow is first determined on the coarsest scale and then refined step-wise towards the finest scale. On each scale, 3 iterations of warping is performed. Each warping step consists of linearizing the data term in the objective function by calculating temporal and spatial derivatives of the image. The upsampling of the flow field is performed using bicubic interpolation (see Section 2.2.2.2) [8].

4.3.3 Median filtering

In order to remove outliers, a 5×5 median filtering is performed once after each warping step. This is the step that gives the most significant improvement of the baseline algorithm [8]. Given the current flow, motion

boundaries are detected with a so-called Sobel edge detector, which is based on the Sobel operator. The flow field is convoluted with two 3×3 convolution kernels, one for estimating the gradient in x-direction and one for estimating the gradient in y-direction. The x- and y-directional gradients are used to find horizontal and vertical edges, respectively [46]. The motion boundaries are then extended to form flow boundary regions using a 5×5 mask. In these regions, the median is computed in a 15×15 neighborhood with weights from Equation (4.17). In all other regions, the normal 5×5 median filter with equal weights are used [8].

4.3.4 Objective function

Even if the median filter improves the flow field, it can lead to that the classical objective function (Equation (2.30)), which should be minimized, gets a higher instead of a lower energy. To fix this inconsistency, a new robust, non-local term and a coupling term are added to the objective function, which now becomes

$$\begin{aligned} E_A(\mathbf{f_x}, \mathbf{f_y}, \hat{\mathbf{f}}_\mathbf{x}, \hat{\mathbf{f}}_\mathbf{y}) = \sum_{i,j} & \rho_D(I_1(i,j) - I_2(i + f_x(i,j), j + f_y(i,j))) \\ & + \lambda(\rho_S(f_x(i,j) - f_x(i+1,j)) + \rho_S(f_x(i,j) - f_x(i,j+1)) \\ & + \rho_S(f_y(i,j) - f_y(i+1,j)) + \rho_S(f_y(i,j) - f_y(i,j+1))) \\ & + \lambda_2(||\mathbf{f_x} - \hat{\mathbf{f}}_\mathbf{x}||^2 + ||\mathbf{f_y} - \hat{\mathbf{f}}_\mathbf{y}||^2) \\ + \sum_{i,j} \sum_{(i',j') \in N_{i,j}} & w_{i,j,i',j'}(|\hat{f}_x(i,j) - \hat{f}_x(i',j')| + |\hat{f}_y(i,j) - \hat{f}_y(i',j')|) \end{aligned} \quad . \tag{4.14}$$

In Equation (4.14), $N_{i,j}$ contains the pixels from a, possibly large, neighborhood to pixel (i, j), while $\hat{\mathbf{f}}_\mathbf{x}$ and $\hat{\mathbf{f}}_\mathbf{y}$ are an extra assistance flow field. The two new terms represent the weighted median filter. While minimizing the objective function, two iterations of a graduated non-convexity (GNC) scheme is performed [8]. GNC is a deterministic approximation method that approximate a function by another function which has a limited second derivative. By changing the objective function according to GNC the problem becomes convex and a solution can easier be found [47].

4.3.4.1 Penalty function

One of the robust penalty functions tested by the authors of Classic+NL-Fast is the Charbonnier function

$$\rho(x) = \sqrt{x^2 + \epsilon^2} \quad . \tag{4.15}$$

Classic+NL-Fast uses a generalized version of this penalty function for both the data term and the spatial term in Equation (4.14), namely

$$\rho_D(x) = \rho_S(x) = \rho(x) = (x^2 + \epsilon^2)^a \quad . \tag{4.16}$$

The parameter a in Equation (4.16) is set to 0.45 and the penalty function is thus referred to as **GC-**0.45. The parameter ϵ in the same equation is set to $\epsilon = 0.001$. The Charbonnier function is convex, but the GNC scheme described above is still useful due to the nonlinear data term [8].

4.3.4.2 Weights

The scalar weights λ and λ_2 are both set to 3 in the MATLAB implementation of Classic+NL-Fast. To prevent over-smoothing of thin structures and near corners, the weight $w_{i,j,i',j'}$ in front of the local term depends on the likelihood that pixel (i, j) and (i', j') belongs to the same surface in the image. This weight is approximated using the color-value distance, the spatial distance and the occlusion state of the two pixels and is written as

$$w_{i,j,i',j'} = \exp\left(-\frac{|i-i'|^2+|j-j'^2|}{2\sigma_1^2} - \frac{|I(i,j)-I(i'-j')|^2}{2\sigma_2^2}\right)\frac{o(i',j')}{o(i,j)} \ , \tag{4.17}$$

where $I(i, j)$ is the color vector in the CIELab color system (see Section 2.3.2). The parameters σ_1 and σ_2 in Equation (4.17) are set to 7 [8]. Define the divergence of the optical flow field in a pixel (i,j) at time t as

$$\mathrm{div}(i, j, t) = \frac{\partial}{\partial x} f_x(i, j, t) + \frac{\partial}{\partial y} f_y(i, j, t) \tag{4.18}$$

and let

$$d(i, j, t) = \begin{cases} \mathrm{div}(i, j, t) & \mathrm{div}(i, j, t) < 0 \\ 0 & \text{otherwise} \end{cases} \ . \tag{4.19}$$

Let further the pixel projection difference be

$$e(i, j, t) = I(i, j, t) - I(i + f_x(i, j, t), j + f_y(i, j, t), t + 1) \ . \tag{4.20}$$

The occlusion state at a specific time point t is then calculated as

$$o(i, j, t) = N(d(i, j, t); \sigma_d) \cdot N(e(i, j, t); \sigma_e) \ , \tag{4.21}$$

where N is a Gaussian function with mean 0 and standard deviation $\sigma_d = 0.3$ and $\sigma_e = 20$, respectively. The value of Equation (4.21) is close to zero for occluded pixels and close to one for unoccluded pixels [48].

4.4 LDOF

The Large Displacement Optical Flow (LDOF) [9] algorithm is based on descriptor matching and a variational model. The algorithm is implemented in MATLAB.

4.4.1 Variational Model

The variational model for LDOF is based on five terms. Let $I_1, I_2 : (\Omega \subset \mathbb{R}^2) \rightarrow \mathbb{R}^3$ be the functions for the RGB values in the first and second frame. Let Ω be the image domain, $\mathbf{x} = (x, y)$ a point in Ω and $\mathbf{w} = (u, v)$ its optical flow. The first term is based on a color consistency assumption and written as

$$E_{\text{color}}(\mathbf{w}) = \int_{\Omega} \Psi(|I_2(\mathbf{x} + \mathbf{w}(\mathbf{x})) - I_1(\mathbf{x})|^2) d\mathbf{x} \ , \tag{4.22}$$

where $\Psi(x) = \sqrt{x + \epsilon^2}$ with $\epsilon = 0.001$. The next term, which is based on a gradient consistency assumption, is

$$E_{\text{grad}}(\mathbf{w}) = \int_{\Omega} \Psi(|\nabla I_2(\mathbf{x} + \mathbf{w}(\mathbf{x})) - \nabla I_1(\mathbf{x})|^2) d\mathbf{x} \ . \tag{4.23}$$

The color consistency assumption does not work well for illumination changes. The gradient consistency assumption works better for illumination effects and is therefore a supplement for the color consistency. The third term is based on smoothness regularization and is written as

$$E_{\text{smooth}}(\mathbf{w}) = \int_{\Omega} \Psi(|\nabla u(\mathbf{x})|^2 + |\nabla v(\mathbf{x})|^2) d\mathbf{x} \ . \tag{4.24}$$

This term penalizes variations of the flow, which leads to a smooth flow. The fourth term, which is based on descriptor matching, is written as

$$E_{\text{match}}(\mathbf{w}, \mathbf{w}_1) = \int \delta(\mathbf{x}) \rho(\mathbf{x}) \Psi(|\mathbf{w}(\mathbf{x}) - \mathbf{w}_1(\mathbf{x})|^2) d\mathbf{x} \tag{4.25}$$

where

$$\delta(\mathbf{x}) = \begin{cases} 1 & \text{if there exists at least one descriptor} \\ & \text{at the point } \mathbf{x} \text{ in the first frame} \\ 0 & \text{otherwise} \end{cases} .$$

The formula contains a factor $\rho(\mathbf{x})$ which is the matching score described in Section 4.4.2.1. The fifth and last term is based on descriptors and written as

$$E_{\text{desc}}(\mathbf{w}_1) = \int \delta(\mathbf{x}) |\mathbf{f}_2(\mathbf{x} + \mathbf{w}_1(\mathbf{x})) - \mathbf{f}_1(\mathbf{x})|^2 d\mathbf{x} \ , \tag{4.26}$$

where $\mathbf{f}_1$ and $\mathbf{f}_2$ are the feature vectors from the first and second frame, respectively.

Combining the five terms above gives the variational model for LDOF

$$E(\mathbf{w}, \mathbf{w}_1) = E_{\text{color}} + \gamma E_{\text{grad}} + \alpha E_{\text{smooth}} + \beta E_{\text{match}} + E_{\text{desc}} \ , \tag{4.27}$$

where γ, α and β are parameters set to 30, 300 and 5 as default.

4.4.2 Solving the variational model

The variational model for LDOF (see Equation (4.27)) needs an initial guess to be solved. The initial guess is based on two methods, descriptor matching and a continuation method. The continuation method does not care about image details, whereas the descriptor matching does not care about regularity.

4.4.2.1 Descriptor matching

The descriptor matching that minimizes E_{desc} in Equation (4.26) is independent of the rest of the terms in Equation (4.27). In the LDOF paper [9], the authors present three alternative ways to represent and find descriptors. The source code for LDOF used for the analysis in this thesis is based on histograms of oriented gradients (HOG) descriptors. Therefore, only the method to find the HOG descriptors is going to be discussed in this report.

HOG descriptors (see Section 2.2.3) are computed in both the first and second frame. The gradient histograms are calculated in a 7×7 neighborhood and includes 15 different orientations. The default HOG descriptors neglect the signs of the gradients. LDOF does not neglect the signs.

The matching score from Equation (4.25) is defined as

$$\rho(\mathbf{x}_i) = \frac{d_2 - d_1}{d_1} \ , \tag{4.28}$$

where d_1 is the distance for the best match and d_2 is the distance for the second best match. The distance is calculated by taking the sum of the square difference between descriptor vectors.

In Equation (4.26), $\delta(\mathbf{x})$ is zero for some $\mathbf{x}$. Actually $\delta(\mathbf{x})$ is a grid generated by removing every fourth pixel in x-direction and y-direction. The only reason to make such a grid is performance. The grid does not disregard the small textures since the histograms are based on a 7×7 area.

In order to eliminate some of the false matches in regions without texture, the LDOF algorithm computes the smallest eigenvalue of $\nabla I \nabla I^T$ for all descriptor points. $\nabla I \nabla I^T$ is a measure of the structure in the image. Descriptors at points with an eigenvalue smaller than 1/8 of the average of the whole image is then removed, by setting $\delta(\mathbf{x})$ to zero. At the end $\delta(\mathbf{x})$ is set to zero for inconsistent backward and forward matches. Forward matching is performed from the first frame to the second and backward matching is the other way around.

4.4.2.2 Continuation method

Equation (4.27) can be solved by a continuation method after E_{desc} is detached. The method splits the original problem into several subproblems. The subproblems are achieved by downsampling the input image to different resolution levels. For level k the downsampling factor r is

$$r = 0.98^{k_{max}} - k \tag{4.29}$$

with

$$k_{max} = \left\lceil \frac{\log(40) - \log(h)}{\log(0.98)} \right\rceil \ ,$$

where h is the height of the input image. The global optimum for a subproblem is achieved by considering the Euler-Lagrange equations for Equation (4.27), which is

$$\begin{aligned}
&\Psi'(I_z^2)I_zI_x + \gamma\Psi'(I_{xy}^2 + I_{yz}^2)(I_{xx}I_{xz} + I_{xy}Iyz) \\
&+ \beta\phi\Psi'\big((u-u_1)^2 + (v-v_1)^2\big)(u-u_1) \\
&- \alpha\text{div}\big(\Psi'(|\nabla u|^2 + |\nabla v|^2)\nabla u\big) = 0 \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad , \\
&\Psi'(I_z^2)I_zI_y + \gamma\Psi'(I_{xy}^2 + I_{yz}^2)(I_{xy}I_{xz} + I_{yy}I_{yz}) \\
&+ \beta\phi\Psi'\big((u-u_1)^2 + (v-v_1)^2\big)(v-v_1) \\
&- \alpha\text{div}\big(\Psi'(|\nabla u|^2 + |\nabla v|^2)\nabla v\big) = 0
\end{aligned} \tag{4.30}$$

where

$$\begin{aligned}
I_x &= \partial_x I_2(\mathbf{x}+\mathbf{w}) & I_{xy} &= \partial_{xy} I_2(\mathbf{x}+\mathbf{w}) \\
I_y &= \partial_y I_2(\mathbf{x}+\mathbf{w}) & I_{yy} &= \partial_{yy} I_2(\mathbf{x}+\mathbf{w}) \\
I_z &= I_2(\mathbf{x}+\mathbf{w}) - I_1(\mathbf{x}) & I_{xz} &= \partial_x I_z \\
I_{xx} &= \partial_{xx} I_2(\mathbf{x}+\mathbf{w}) & I_{yz} &= \partial_y I_z \ .
\end{aligned}$$

LDOF uses fixed point iterations to solve Equation (4.30) [9]. The estimated flow $\mathbf{w}$ for each level is calculated as

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \mathbf{dw}^k \ , \tag{4.31}$$

where the initial flow for the coarsest level is $\mathbf{w}^0 = \mathbf{0}$. The term $\mathbf{dw}^k$ is the solution of the fix point iterations for level k.

Every subproblem can be optimized globally, but that does not imply that the original problem achieves a global optimum. On the other hand, the method avoids most of the local minima for the original problem [9].

The algorithm is running a last iteration with $\beta = 0$, which is motivated by

the fact that the descriptor correspondences have high impact on the solution at the coarse scale. For higher resolutions, the number of pixels increases and, therefore, the impact of the descriptor correspondences decreases. For the finest scale the continuous limit of the ratio becomes zero [9].

4.5 Noise filter

Two different types of noise filters are developed in the project. The first one, which is called ”temporal noise filter”, uses no information about optical flow. The other one is called ”optical flow noise filter”. The two filters are structured in a similar way, but there are some differences. The filters have one output and two input signals. The output signal for the pixel (x, y) is called $\text{out}_{x,y}$. One of the input signals is $\text{last}_{x,y}$ which is the output signal for pixel (x, y) for the previous frame. The second input signal is $\text{new}_{x,y}$, which is the pixel value at (x, y) for the latest noisy image in the sequence. All the parameters for the filters are determined after advises from the supervisors of the thesis [26] and experiments.

4.5.1 Temporal noise filter

The temporal filter uses the pixel values of the previous filtered frame in combination with the latest noisy image in the sequence. One part of the output signal for a given pixel is from the filtered image and the other is from the noisy image. The size of the parts for a given pixel is determined by the number of frames since the pixel was not stationary. A pixel for a given frame is called stationary if the patch around the pixel is similar to the corresponding patch for the latest filtered image. The pixels that are stationary are called fixed and the remaining pixels are called unfixed.

The filter starts by finding stationary pixels. After the stationary pixels are found, different actions are performed on the fixed and unfixed pixels respectively.

4.5.1.1 Find stationarity

Stationary pixels are found by comparing the pixels at the same location in two frames. The comparison is accomplished by consider a patch of size 9×9 around each pixel. For each patch, the mean value is calculated for the R, G and B component (see Section 2.3.1) respectively. The mean value for each component is then compared between the two patches. If the difference between any of the three components is above $\sigma = 200$, the pixel is marked as unfixed. After the mean value check, the filter performs a remover step. This is done by calculating the number of fixed pixels in a 5×5 area around each unfixed pixel. If the number of fixed pixels is above the threshold 1, the

pixel is also marked as fixed. At the end, a filler step is performed. The filter looks at a 7×7 area around each fixed pixel and if the number of unfixed pixels is above the threshold 1, the pixel is marked as unfixed.

4.5.1.2 Unfixed pixels

First, the number of frames since stationarity is set to one for the pixel. The value for $\text{out}_{x,y}$ is set to the mean value of the 5×5 area around $\text{new}_{x,y}$.

$$\text{out}_{x,y} = \frac{1}{25} \sum_{i=-2}^{2} \sum_{j=-2}^{2} \text{new}_{x+i,y+j} \ , \tag{4.32}$$

4.5.1.3 Fixed pixels

First, the number of frames since stationarity is checked for the pixel. Then, Equation (4.33) is used to calculate the value for $\text{out}_{x,y}$. At the end, the frames since stationarity value is increased by one for the pixel.

$$\text{out}_{x,y} = (1 - \alpha)\text{last}_{x,y} + \alpha\text{new}_{x,y} \ , \tag{4.33}$$

where $\alpha = \dfrac{1}{1 + \text{frames since stationarity}}$.

4.5.2 Optical flow noise filter

The optical flow noise filter is working in nearly the same way as the temporal filter described in Section 4.5.1. Though, there are two differences. The first is that the optical flow filter is using occlusions instead of stationary points. The second is that the optical flow filter is using the pixel value in last filtered image, whereat the optical flow is pointing.

4.5.2.1 Occlusion finder and detection

The optical flow is calculated for both the backward and forward flow, which means that the flow is calculated between frame_{t-1} and frame_t, but also between frame_t and frame_{t-1}. A pixel is marked as occluded if the difference between the backward and forward flow is above the threshold 0.5. This step is called *finder*. Both SimpleFlow and EPPM use the same technique to find occlusions in their algorithms [6, 7]. Next a comparison is accomplished by consider a patch of size 9×9 around $\text{new}_{x,y}$ and $\text{last}_{x+u,y+v}$ respectively, where u, v is the optical flow in point (x, y). For each patch, the mean value is calculated for the R, G and B component (see Section 2.3.1) respectively. The mean value for each component is then compared between the two patches. If the difference between any of the three components is above σ, the pixel is marked as occluded. The parameter σ is set to 200 for sequences

with full well 800 and 400 electrons, and 350 for sequences with full well 100 and 50 electrons. This step is called *detection*. At the end, occlusion removing and occlusion filling performed in the same way as described above for the stationary removing and filling for the temporal filter. The last three steps (detection, removing and filling) try to eliminate incorrect flow.

4.5.2.2 Occluded pixels

The value for $\text{out}_{x,y}$ is calculated by taking the mean value of the 5×5 area around $\text{new}_{x,y}$, according to Equation (4.34). Furthermore, the frames since occlusion value is set to zero.

$$\text{out}_{x,y} = \frac{1}{25} \sum_{i=-2}^{2} \sum_{j=-2}^{2} \text{new}_{x+i,y+j} \ , \tag{4.34}$$

4.5.2.3 Unoccluded pixels

Unlike the previous filter, the new pixel value is calculated by using the optical flow for $\text{new}_{x,y}$. Let (u, v) be the flow value for $\text{new}_{x,y}$. Equation (4.33) is then modified by replacing $\text{last}_{x,y}$ with $\text{last}_{x+u,y+v}$, which gives

$$\text{out}x, y = (1 - \alpha)\text{last}x + u, y + v + \alpha\text{new}x, y \ . \tag{4.35}$$

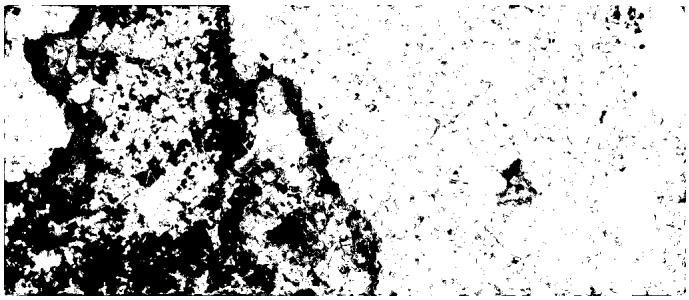
Since (u, v) are floats and not integers a subpixel estimation is performed to get $\text{last}_{x+u,y+v}$. Equation (4.36) is used to get an estimate.

$$\begin{array}{rcl} \text{last}x + u, y + v & = & a \cdot b \cdot \text{last}x + \overline{u}, y + \overline{v} \\ & + & (1 - a) \cdot b \cdot \text{last}x + \overline{u} + c, y + \overline{v} \\ & + & a \cdot (1 - b) \cdot \text{last}x + \overline{u}, y + \overline{v} + d \\ & + & (1 - a) \cdot (1 - b) \cdot \text{last}x + \overline{u} + c, y + \overline{v} + d \end{array} \ , \tag{4.36}$$

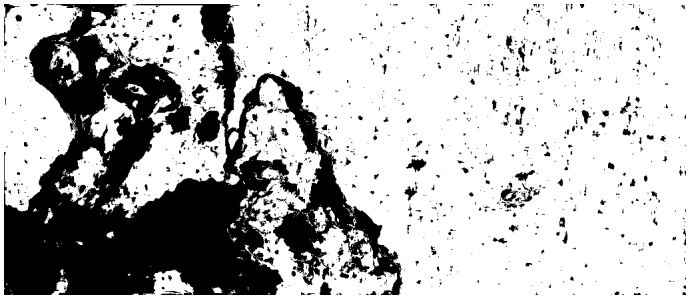
where

$$\begin{array}{rcl} a & = & 1 - |\overline{u} - u| \\ b & = & 1 - |\overline{v} - v| \\ c & = & \begin{cases} 1 & \text{if } \overline{u} - u > 0 \\ 0 & \text{otherwise} \end{cases} \\ d & = & \begin{cases} 1 & \text{if } \overline{v} - v > 0 \\ 0 & \text{otherwise} \end{cases} \end{array} \ .$$

At the end, the frames since occlusion value is increased by one.



(a) Finder



(b) Detection

(c) Remover

(d) Filler

Figure 4.6: The four occlusion steps when LDOF is used on frame 15 from the sequence market_2 with full well 100 electrons.

Chapter 5

Dataset

The dataset used for testing the optical flow algorithms consist of four image sequences, all with the resolution 1024×436 pixels. Two of them, called bandage_1 and market_2, comes from the MPI-Sintel dataset. The two other image sequences, called real_1 and real_2, have been recorded at Axis Communication during the thesis. The image sequences are presented below.

5.1 Bandage_1

The sequence bandage_1 consists of 50 image frames. The sequence is animated and shows a girl caring for a wounded baby dragon. The background is more or less stationary, while a relatively large part in the middle of the foreground is moving. The motions are small. One of the frames from bandage_1 can be seen in Figure 5.1 and every fifth frame can be seen in Figure 5.2.

Figure 5.1: Frame 20 from the image sequence bandage_1.

Figure 5.2: The image sequence bandage_1.

5.2 Market_2

The sequence market_2 shows people moving in a marketplace. One of the frames from market_2 can be seen in Figure 5.3. Just like in the case of bandage_1, the sequence is animated. Most motions are relatively large. The background has some rotation. The sequence consists of 50 image frames. Every fifth of them is shown in Figure 5.4.

Figure 5.3: Frame 15 from the image sequence market_2.

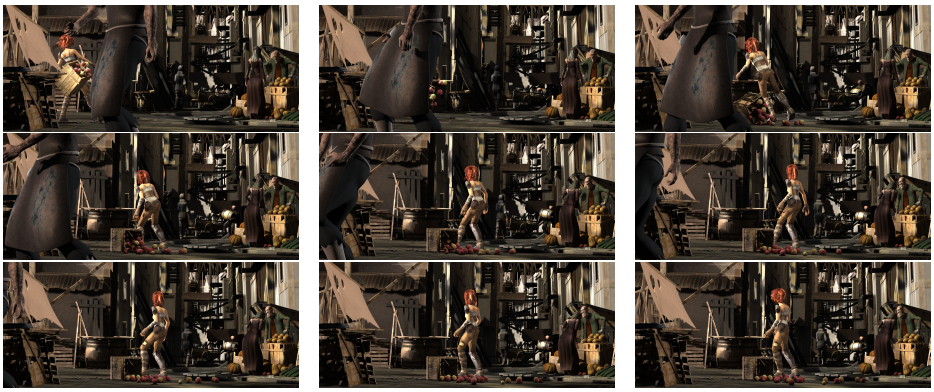


Figure 5.4: The image sequence market_2.

5.3 Real_1

The sequence real_1, is recorded at Axis. It shows two people walking in opposite directions, with their side against the camera and crossing each other's paths in front of a static wall with a picture on. One of the frames can be seen in Figure 5.5. The sequence contains textured as well as flat areas, both in the moving and stationary parts of the motive. This is the darkest of the four sequences. Motion does not appear in very big parts of each frame, but during the 104 frames of the image sequence, the people are walking through the entire image. Figure 5.6, which consists of frame 20, 30, 40, 50, 60, 70, 80, 90 and 100, gives an idea about the type of motions in this sequence.

Figure 5.5: Frame 60 of on of the image sequences captured at Axis, called real_1.

Figure 5.6: The image sequence real_1.

5.4 Real_2

The other sequence recorded at Axis, real_2, shows two people throwing a plastic bag between each other. This sequence consists of 58 image frames, out of which frame 39 is shown in Figure 5.7. The background is static and mainly consisting of a white wall. The motion is centered to small parts of the image and the objects moving are mostly thin or small and moving fast, for instance hands and the plastic bag. The sequence also contains shadows and a mixture of flat and textured areas. Real_2 is much lighter than real_1 and the contrast between the moving parts and the background is higher (see Figure 5.11 and 5.12). A short version of the sequence is shown in Figure 5.8, which shows frame 9, 14, 19, 24, 29, 34, 39, 44 and 49.

Figure 5.7: Frame 39 of one of the image sequences captured at Axis, called real_2.

Figure 5.8: The image sequence real_2.

5.5 Image histograms

The intensity varies for the four different sequences. In Figure 5.9–5.12 the gray scale histograms for a frame in each sequence are represented. The intensity for each image reaches from 0 to 4095. The darkest sequence is market_2, which has a mean pixel value of 980, closely followed by real_1, with 990. The mean pixel value for bandage_1 and real_2 is 1330 and 2550, respectively.

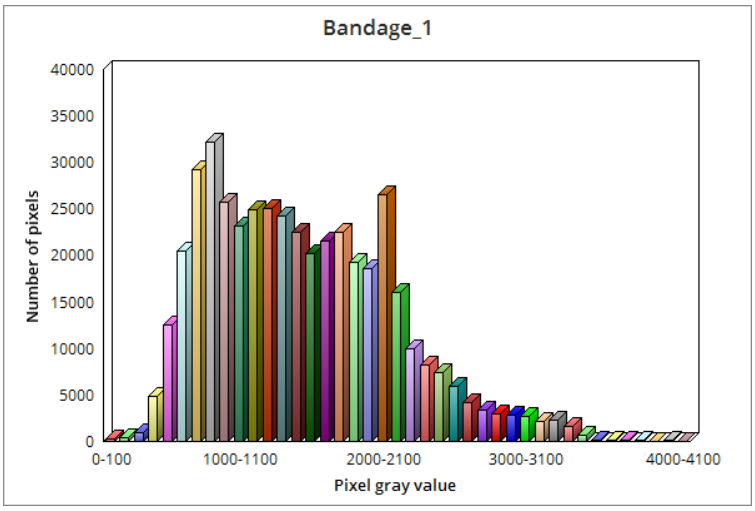


Figure 5.9: Histogram for frame 15 for the sequence bandage_1. The mean value is 1330.

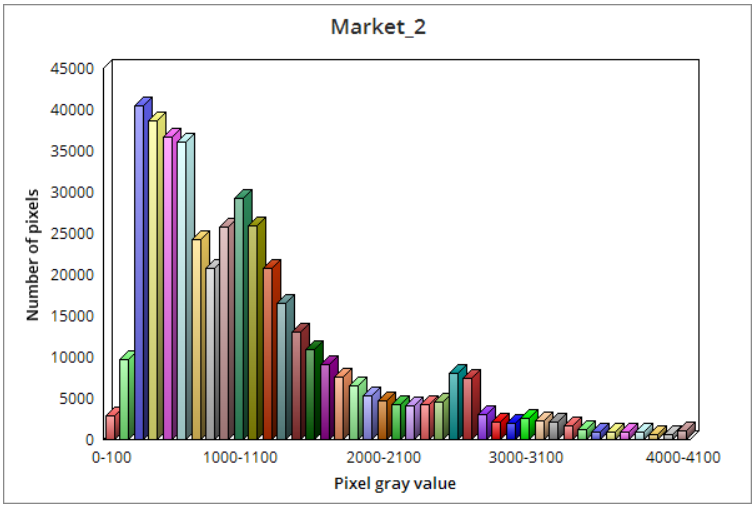


Figure 5.10: Histogram for frame 20 for the sequence market_2. The mean value is 980.

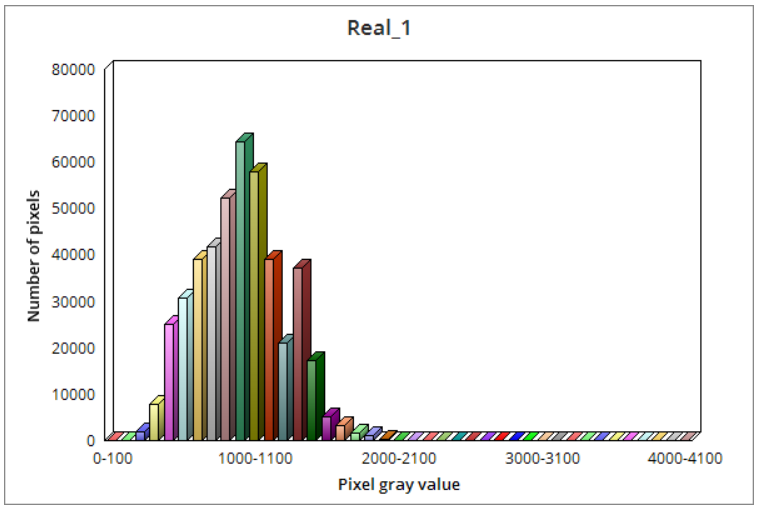


Figure 5.11: Histogram for frame 60 for the sequence real_1. The mean value is 990.

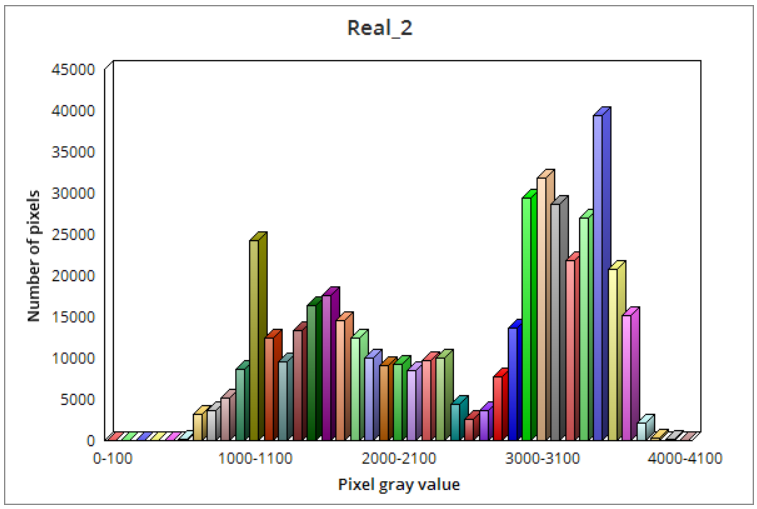


Figure 5.12: Histogram for frame 39 for the sequence real_2. The mean value is 2510.

5.6 Noise levels

As have been presented in Section 3.4, five different noise levels have been used in the thesis, corresponding to full well 50, 100, 200, 400 and 800 electrons. The σ_{read} parameter described in Section 2.4 has been set to 5 electrons for all levels. The amount of noise produced by these parameter choices varies a bit among the different sequences. This is because of the fact that the amount of noise in an image depends on its intensity. Dark areas contains less noise than lighter areas, but the signal-to-noise-ratio is lower and thus is the mean error in intensity larger. Figure 5.13 shows the mean error in intensity between a noisy grayscale image (full well = 100 electrons) and the same image without noise for the pixels at different light levels.

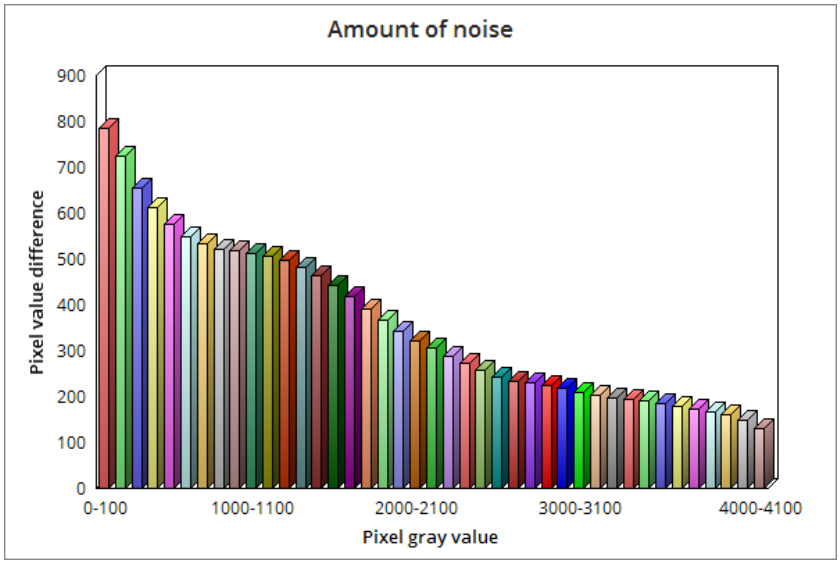


Figure 5.13: Amount of noise in different light levels of a grayscale image.

The amount of noise in the image sequences bandage_1 and real_1 can be seen in Figure 5.14 and Figure 5.15, respectively. According to Section 5.5, real_1 has a lower mean pixel value than bandage_1 and contains more dark areas. This means that the signal-to-noise-ration is lower for real_1, which is confirmed by the fact that the noise is more visible in Figure 5.15, compared with the same noise levels in Figure 5.14.

(a) Full well 800 electrons

(b) Full well 400 electrons

(c) Full well 200 electrons

(d) Full well 100 electrons

(e) Full well 50 electrons

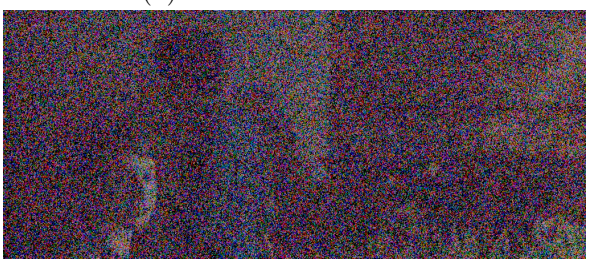
Figure 5.14: The different noise levels on frame 20 from the image sequence bandage_1.

(a) Full well 800 electrons

(b) Full well 400 electrons

(c) Full well 200 electrons

(d) Full well 100 electrons



(e) Full well 50 electrons

Figure 5.15: The different noise levels on frame 60 from the image sequence real_1.

Chapter 6

Result

The results presented in this chapter comes from the evaluation of the noise filter on four different image sequences: bandage_1 and market_2 from the MPI-Sintel dataset [2] and real_1 and real_2, which have been recorded at Axis during the thesis. For each sequence, three different tables with SSIM (see Section 2.4.1.3) values for 5 different noise levels (full well 800, 400, 200, 100 and 50 electrons) are presented. All values are the mean SSIM over the whole image sequence compared with the same image sequence without noise.

The first table contains SSIM and MSE (see Section 2.4.1.1) of the unfiltered image.

The second table for each image sequence consist of the SSIM obtained when filtering with the optical flow produced by each of the four optical flow algorithms considered. Here, the best algorithm for each noise level is marked with gold background, the second best with silver background and the third best with bronze background. The third table is almost as the second one, but the image sequence has been pre-processed (see Section 3.4.1) before the optical flow was determined.

Both the second and the third table also contains SSIM for *Temporal* and *Ground truth*. Temporal is the sequence after filtering with the temporal filter without optical flow. Ground Truth is the sequence produced by noise filtering using the noise filter from Section 4.5.2, fed with the ground truth optical flow and ground truth occlusions. This represents the performance of the noise filter when using a perfect optical flow, i.e. a best-case result for the noise filter. For the sequences real_1 and real_2, no ground truth optical flow exists.

In addition to the SSIM tables, tables with the amount of occlusions found in each image sequence are presented. The quantity of occlusions in one image frame is measured as the percentage of the pixels in that image frame, which

are considered occluded when the current optical flow algorithm is used. The values presented are the mean over all image frames in the sequence. The occlusions come from the steps *finder* and *filler*, explained in Section 4.5.2.1.

6.1 Bandage_1

	Noise level				
	1	**2**	**3**	**4**	**5**
MSE	**38 000**	**65 300**	**129 400**	**264 400**	**523 500**
SSIM	**0.798**	**0.649**	**0.467**	**0.294**	**0.164**

Table 6.1: MSE and SSIM for the noisy images of bandage_1.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	0.851	0.820	0.747	0.641	0.458
EPPM	0.885	0.837	0.731	0.604	0.415
Classic+NL-Fast	0.883	0.820	0.711	0.555	0.385
LDOF	0.900	0.858	0.772	0.640	0.434
Temporal	0.833	0.796	0.721	0.591	0.402
Ground truth	0.893	0.865	0.815	0.734	0.624

Table 6.2: SSIM for the filtered images of bandage_1 when using optical flow based filters, a temporal filter and a filter based on the ground truth optical flow.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	0.830	0.772	0.680	0.557	0.437
EPPM	0.879	0.824	0.718	0.578	0.410
Classic+NL-Fast	0.890	0.827	0.721	0.567	0.393
LDOF	0.892	0.854	0.780	0.673	0.495
Temporal	0.833	0.796	0.721	0.591	0.402
Ground truth	0.893	0.865	0.815	0.734	0.624

Table 6.3: SSIM for the filtered images of bandage_1 when using optical flow based filters, a temporal filter and a filter based on the ground truth optical flow. The optical flow is calculated by first pre-processing the image sequence.

6.1.1 Occlusions

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	11%	12%	16%	26%	45%
EPPM	28%	28%	25%	26%	34%
Classic+NL-Fast	16%	27%	42%	57%	68%
LDOF	6%	7%	8%	11%	21%
Ground truth	4%				

Table 6.4: Occluded pixels after the finder step for bandage_1.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	12%	12%	7%	8%	17%
EPPM	36%	36%	27%	26%	35%
Classic+NL-Fast	17%	27%	39%	53%	67%
LDOF	8%	9%	8%	10%	19%
Ground truth	4%				

Table 6.5: Occluded pixels after the filler step for bandage_1.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	16%	18%	23%	26%	26%
EPPM	32%	36%	37%	39%	42%
Classic+NL-Fast	9%	17%	33%	55%	71%
LDOF	8%	10%	9%	13%	10%
Ground truth	4%				

Table 6.6: Occluded pixels after the finder step for bandage_1 pre-processed.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	19%	22%	21%	19%	17%
EPPM	41%	45%	44%	45%	49%
Classic+NL-Fast	11%	19%	33%	56%	75%
LDOF	9%	12%	9%	9%	15%
Ground truth	4%				

Table 6.7: Occluded pixels after the filler step for bandage_1 pre-processed.

6.2 Market_2

	Noise level				
	1	**2**	**3**	**4**	**5**
MSE	**60 500**	**101 000**	**183 800**	**341 700**	**633 100**
SSIM	**0.685**	**0.560**	**0.426**	**0.303**	**0.201**

Table 6.8: MSE and SSIM for the noisy images of market_2.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	0.797	0.754	0.666	0.578	0.435
EPPM	0.799	0.739	0.642	0.540	0.385
Classic+NL-Fast	0.797	0.725	0.615	0.489	0.362
LDOF	0.821	0.777	0.693	0.585	0.420
Temporal	0.692	0.649	0.582	0.483	0.351
Ground truth	0.834	0.795	0.732	0.643	0.533

Table 6.9: SSIM for the filtered images of market_2 when using optical flow based filters, a temporal filter and a filter based on the ground truth optical flow.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	0.781	0.718	0.613	0.508	0.416
EPPM	0.788	0.727	0.619	0.495	0.368
Classic+NL-Fast	0.808	0.739	0.625	0.498	0.368
LDOF	0.803	0.747	0.678	0.606	0.475
Temporal	0.692	0.649	0.582	0.483	0.351
Ground truth	0.834	0.795	0.732	0.643	0.533

Table 6.10: SSIM for the filtered images of market_2 when using optical flow based filters, a temporal filter and a filter based on the ground truth optical flow. The optical flow is calculated by first pre-processing the image sequence.

6.2.1 Occlusions

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	13%	14%	18%	28%	45%
EPPM	18%	21%	21%	22%	34%
Classic+NL-Fast	28%	40%	51%	60%	67%
LDOF	9%	10%	10%	13%	22%
Ground truth	4%				

Table 6.11: Occluded pixels after the finder step for market_2.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	14%	14%	8%	8%	15%
EPPM	25%	27%	20%	19%	34%
Classic+NL-Fast	28%	39%	46%	55%	89%
LDOF	11%	12%	11%	13%	20%
Ground truth	4%				

Table 6.12: Occluded pixels after the filler step for market_2.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	24%	26%	28%	29%	26%
EPPM	28%	32%	33%	38%	40%
Classic+NL-Fast	17%	26%	43%	60%	71%
LDOF	10%	11%	11%	11%	12%
Ground truth	4%				

Table 6.13: Occluded pixels after the finder step for market_2 pre-processed.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	26%	30%	25%	20%	17%
EPPM	35%	41%	39%	44%	47%
Classic+NL-Fast	19%	29%	44%	62%	75%
LDOF	12%	14%	11%	12%	16%
Ground truth	4%				

Table 6.14: Occluded pixels after the filler step for market_2 pre-processed.

6.3 Real_1

	Noise level				
	1	**2**	**3**	**4**	**5**
MSE	**29 600**	**79 500**	**183 000**	**343 400**	**606 800**
SSIM	**0.625**	**0.379**	**0.181**	**0.088**	**0.051**

Table 6.15: MSE and SSIM for the noisy images of real_1.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	0.902	0.775	0.764	0.445	0.209
EPPM	0.872	0.710	0.507	0.319	0.201
Classic+NL-Fast	0.849	0.690	0.471	0.308	0.210
LDOF	0.915	0.783	0.690	0.338	0.200
Temporal	0.903	0.812	0.597	0.321	0.195

Table 6.16: SSIM for the filtered images of real_1 when using optical flow based filters and a temporal filter.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	0.864	0.697	0.513	0.349	0.216
EPPM	0.855	0.705	0.522	0.319	0.204
Classic+NL-Fast	0.858	0.700	0.486	0.318	0.214
LDOF	0.904	0.759	0.701	0.453	0.210
Temporal	0.903	0.812	0.597	0.321	0.210

Table 6.17: SSIM for the filtered images of real_1 when using optical flow based filters and a temporal filter. The optical flow is calculated by first pre-processing the image sequence.

6.3.1 Occlusions

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	18%	20%	27%	42%	57%
EPPM	29%	28%	37%	38%	39%
Classic+NL-Fast	70%	76%	78%	80%	82%
LDOF	10%	13%	18%	35%	56%

Table 6.18: Occluded pixels after the finder step for real_1.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	17%	21%	7%	22%	57%
EPPM	29%	41%	41%	52%	67%
Classic+NL-Fast	70%	80%	77%	84%	91%
LDOF	12%	22%	18%	40%	73%

Table 6.19: Occluded pixels after the filler step for real_1.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	38%	48%	50%	35%	33%
EPPM	46%	45%	41%	32%	36%
Classic+NL-Fast	58%	76%	85%	86%	86%
LDOF	15%	20%	16%	19%	41%

Table 6.20: Occluded pixels after the finder step for real_1 pre-processed.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	40%	57%	39%	38%	51%
EPPM	55%	59%	47%	47%	65%
Classic+NL-Fast	60%	81%	88%	91%	99%
LDOF	16%	28%	19%	30%	67%

Table 6.21: Occluded pixels after the filler step for real_1 pre-processed.

6.4 Real_2

	Noise level				
	1	**2**	**3**	**4**	**5**
MSE	**6 450**	**17 570**	**50 550**	**126 060**	**284 730**
SSIM	**0.881**	**0.771**	**0.605**	**0.414**	**0.242**

Table 6.22: RMSE and SSIM for the noisy images of real_2.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	0.949	0.927	0.884	0.794	0.628
EPPM	0.948	0.914	0.846	0.751	0.592
Classic+NL-Fast	0.949	0.909	0.821	0.687	0.516
LDOF	0.958	0.940	0.890	0.747	0.549
Temporal	0.929	0.913	0.864	0.747	0.570

Table 6.23: SSIM for the filtered images of real_2 when using optical flow based filters and a temporal filter.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	0.946	0.911	0.825	0.692	0.515
EPPM	0.947	0.911	0.832	0.729	0.584
Classic+NL-Fast	0.954	0.917	0.826	0.693	0.523
LDOF	0.954	0.929	0.877	0.771	0.574
Temporal	0.929	0.913	0.864	0.747	0.570

Table 6.24: SSIM for the filtered images of real_2 when using optical flow based filters and a temporal filter. The optical flow is calculated by first pre-processing the image sequence.

6.4.1 Occlusions

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	19%	21%	23%	26%	35%
EPPM	32%	31%	26%	24%	14%
Classic+NL-Fast	46%	59%	70%	76%	77%
LDOF	9%	10%	13%	22%	38%

Table 6.25: Occluded pixels after the finder step for real_2.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	19%	19%	15%	14%	28%
EPPM	41%	37%	30%	29%	31%
Classic+NL-Fast	48%	63%	71%	78%	83%
LDOF	10%	12%	13%	23%	45%

Table 6.26: Occluded pixels after the filler step for real_2.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	29%	35%	44%	50%	50%
EPPM	48%	44%	38%	32%	27%
Classic+NL-Fast	24%	41%	60%	75%	81%
LDOF	19%	24%	26%	22%	30%

Table 6.27: Occluded pixels after the finder step for real_2 pre-processed.

	Noise level				
	1	**2**	**3**	**4**	**5**
SimpleFlow	32%	39%	47%	52%	58%
EPPM	53%	53%	45%	41%	42%
Classic+NL-Fast	26%	44%	63%	79%	87%
LDOF	20%	25%	26%	26%	42%

Table 6.28: Occluded pixels after the filler step for real_2 pre-processed.

Chapter 7

Analysis

7.1 Analysis of the result

Figure 7.1 - 7.4 shows the performance of the noise filter on each image sequence. The SSIM from Chapter 6 is presented for the noisy image sequence, the sequence temporal filtered without optical flow and the result after the filter has been fed with each of the four optical flow algorithms. The result after pre-processing is not included in the diagrams, neither in the analysis in this section. In the case of the sequences bandage_1 and market_2 are also the sequence filtered with the noise filter with the ground truth optical flow presented.

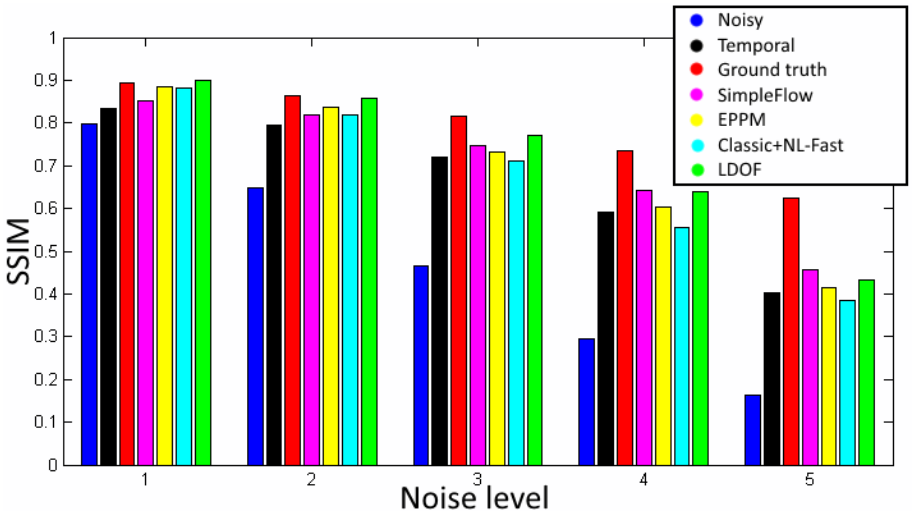


Figure 7.1: The result of the noise filtering of the image sequence bandage_1.

As expected, the unfiltered image sequence (called *Noisy* in Figure 7.1 - 7.4) always has the lowest SSIM, meaning that the noise filtering has served its purpose in all experiments. The noise filter fed with ground truth optical flow

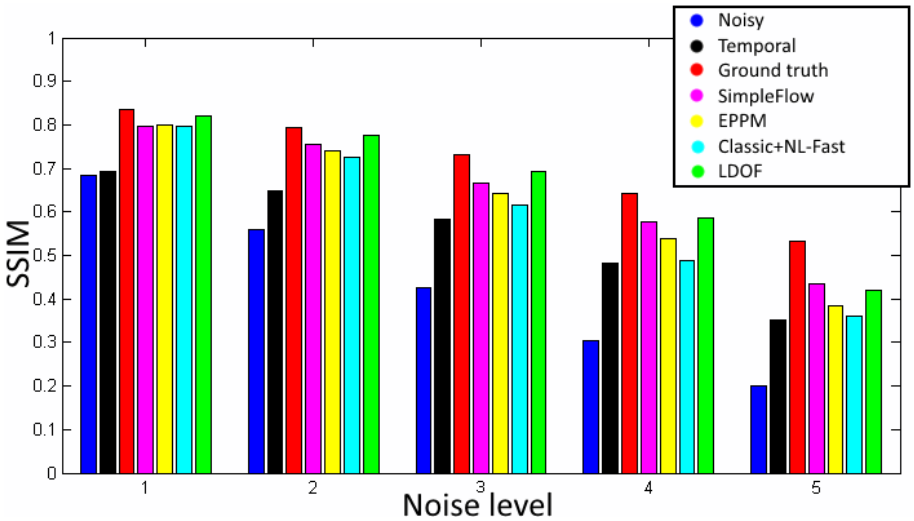


Figure 7.2: The result of the noise filtering of the image sequence market_2.

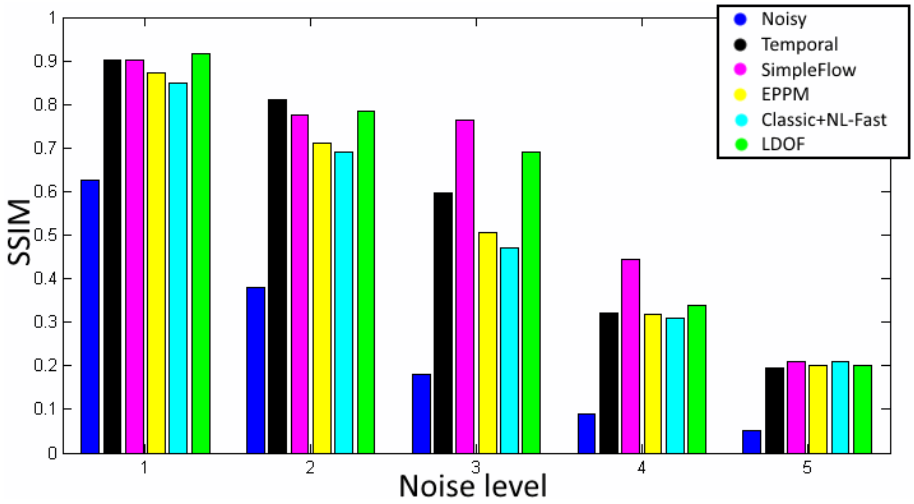


Figure 7.3: The result of the noise filtering of the image sequence real_1.

always beats the temporal filter, leading to the conclusion that optical flow can improve noise filtering, at least if a perfect flow and occlusion detection are obtained.

Regarding the noise filter with optical flow, it usually beats the temporal filter without optical flow. For the sequence market_2, the temporal filter always performs the worst. For the sequence bandage_1, it is only Classic+NL-Fast that is worse, and that only occur on the noise levels full well 200, 100 and 50 electrons. On the sequences captured at Axis, the noise filter with optical flow faces greater challenges. When it comes the sequence

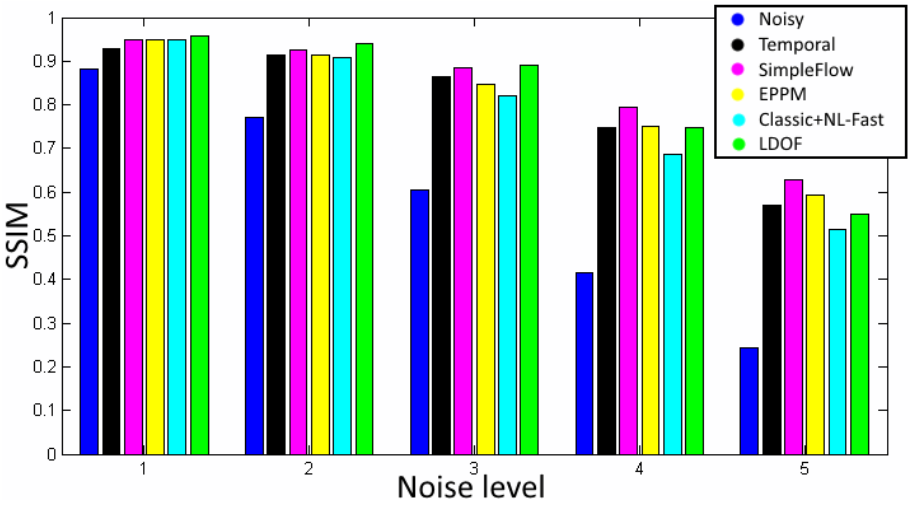
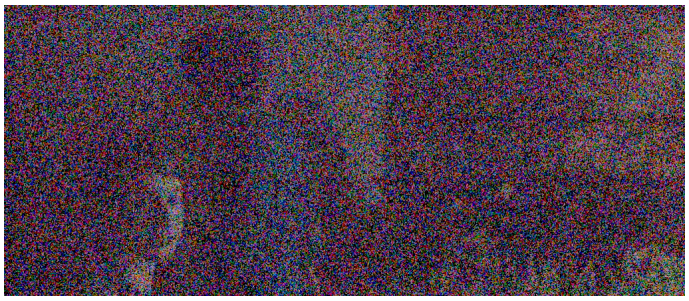


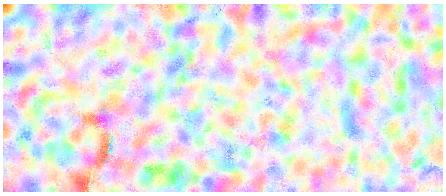
Figure 7.4: The result of the noise filtering of the image sequence real_2.

real_1, only LDOF beats the temporal filter on the noise level full well 800 electrons and on the noise level full well 400 electrons, the temporal filter performs best. On the noise levels full well 200 and 100 electrons, LDOF and SimpleFlow win over the temporal filter, but EPPM and Classic+NL-Fast do not reach the same level of denoising. On the noisiest level (full well 50 electrons) both the temporal filter and all algorithms perform equally well. Worth to remark is that the noisy image has an SSIM value of 0.051, which makes it more difficult than for example bandage_1, which has an SSIM value of 0.164 on the same noise level. Figure 7.5 shows the flow, occlusion mask and filtered image for SimpleFlow. Moreover, it shows the filtered image with the temporal filter. The flow and occlusion mask is nearly totally random which leads to the optical flow filter just doing a kind of smoothing of the noisy image. For that reason, the optical flow estimation can be seen as unsuccessful.

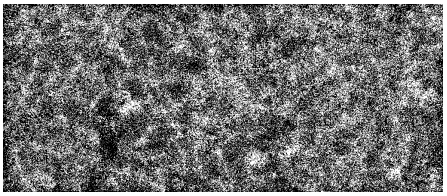
All optical flow algorithms win over the temporal filter on the level with least noise (full well 800 electrons) on real_2. On the other noise levels some of the algorithms beat the temporal filter and some do not.



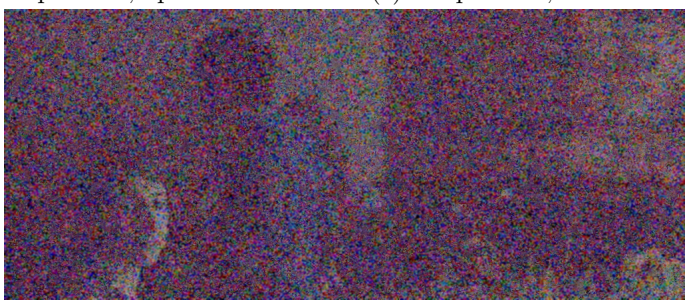
(a) Noisy image (Full well 50, SSIM 0.051)



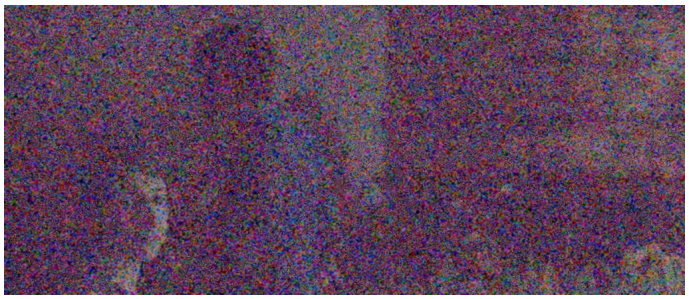
(b) SimpleFlow, optical flow



(c) SimpleFlow, occlusions (finder)



(d) SimpleFlow, filtered image (SSIM 0.209)



(e) Temporal filter (SSIM 0.195)

Figure 7.5: The images are generated from frame 60 of the sequence real_1 with full well 50 electrons. Image (b), (c) and (d) shows the optical flow, occlusions found by the finder step and the filtered image for SimpleFlow. (e) is the image produced by the temporal filter.

7.1.1 Comparison between the optical flow algorithms

The optical flow algorithm that generally performs best is LDOF, which produces the highest SSIM in 12 out of 20 cases. For the lowest noise levels (full well 800 and 400 electrons), it outperforms the other algorithms on all four image sequences. On the noise level with full well 200 electrons, LDOF comes in first place on all sequences except real_1, where it is beaten by SimpleFlow. LDOF is though not as good on the noisiest sequences (full well 100 and 50 electrons) where it mostly is beaten by SimpleFlow and sometimes also by other algorithms.

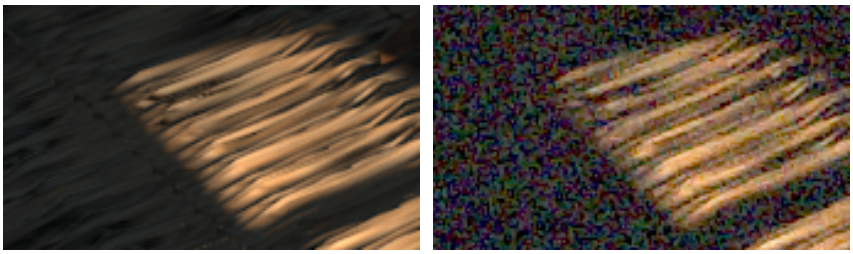
In contradiction to LDOF, SimpleFlow performs best on the noisiest image sequences. In particular, this is evident for the noise level full well 400 electrons on real_1, where SimpleFlow has an SSIM on 0.445, while the closest competitor only has an SSIM on 0.338.

Classic+NL-Fast is overall the worst algorithm, especially without preprocessing. It produces the worst SSIM of the four algorithms in 15 out of 20 cases.

EPPM, finally, is a moderately good algorithm, which never reaches the first place, but at the same time seldom comes last.

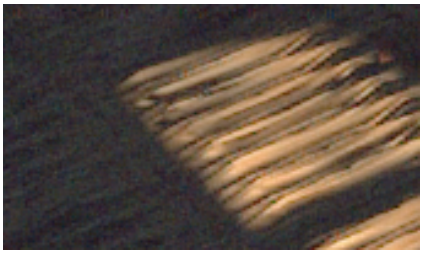
Let us do an analysis of different parts of a filtered image. Figure 7.6 and 7.7 show one part without motion and one part with motion. Figure 7.6 has no motion. The figure shows that ground truth and temporal keep the structure in the image quite well. LDOF and SimpleFlow have less noise but also less structure. Classic+NL-Fast has much noise left after the filtering. It just looks like the image has been smoothed. EPPM is something in between Classic+NL-Fast and SimpleFlow and LDOF. Some parts of the filtered image have less noise and some have much noise.

Figure 7.7 has one big motion. Ground truth, SimpleFlow and LDOF have almost the same result. They keep the structures best of the algorithms and have the least amount of noise. Actually. the frame has one more motion than the big motion, and that is the eye, which has been opened since the previous frame. Because of that, it appears occlusions in the area of the eye. SimpleFlow and LDOF handle the occluded areas better than the ground truth. EPPM and Classic+NL-Fast keep some of the structures of the image, but not as well as ground truth, SimpleFlow and LDOF. For the temporal filter, nearly all structures have disappeared.

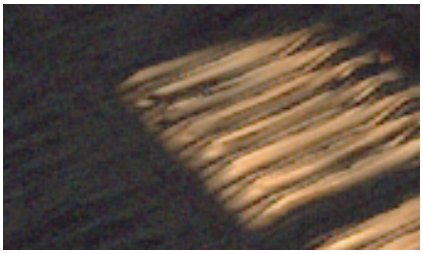


(a) Source

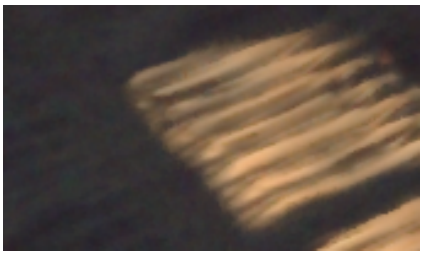
(b) Noisy



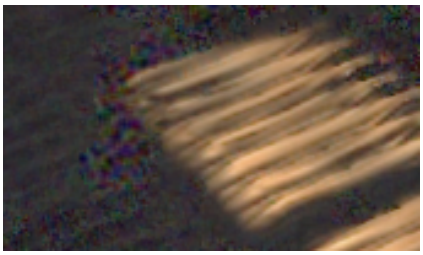
(c) Ground truth



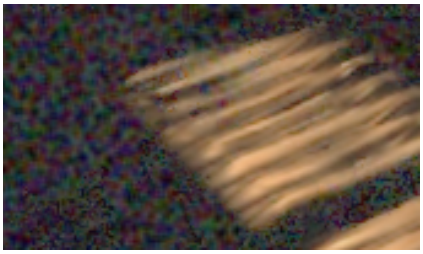
(d) Temporal



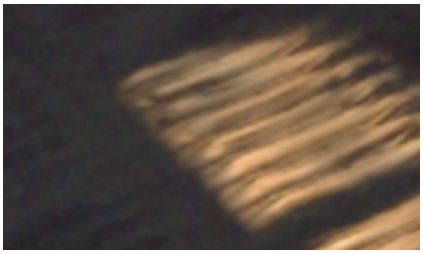
(e) SimpleFlow



(f) EPPM

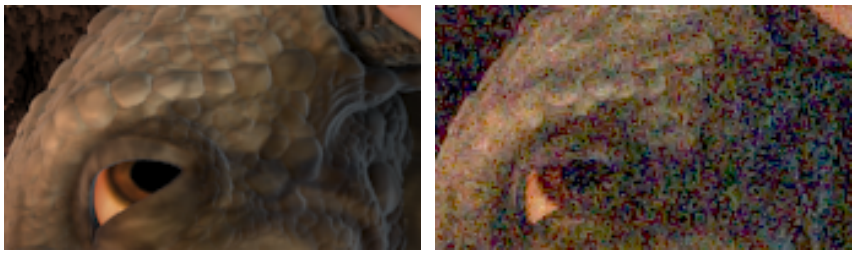


(g) Classic+NL-Fast



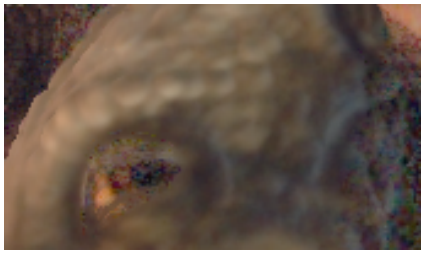
(h) LDOF

Figure 7.6: Filtered images for the different methods for a cropped part of frame 20 of the sequence bandage_1 with full well 200. The frame has no motion.

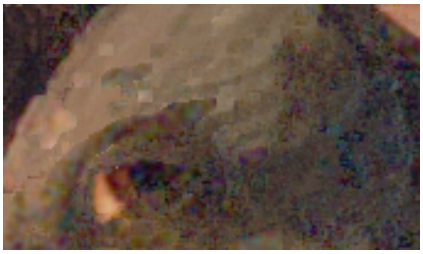


(a) Source

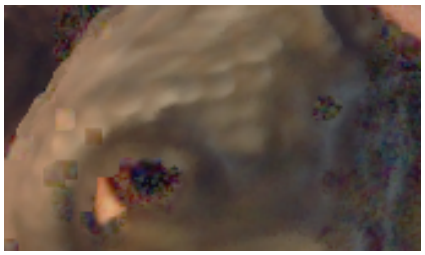
(b) Noisy



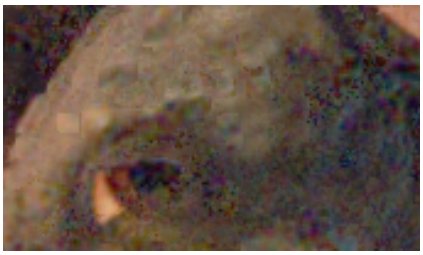
(c) Ground truth



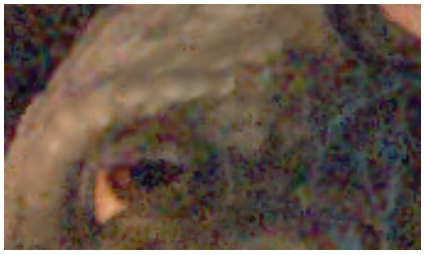
(d) Temporal



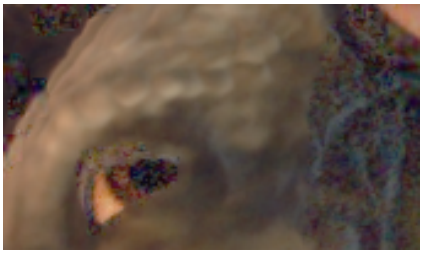
(e) SimpleFlow



(f) EPPM



(g) Classic+NL-Fast

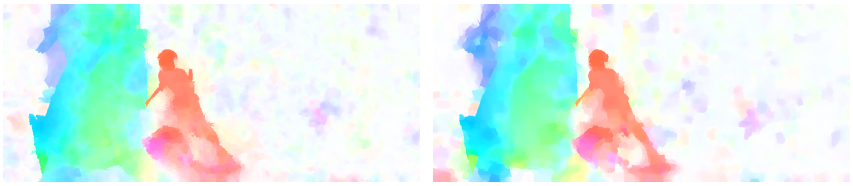


(h) LDOF

Figure 7.7: Filtered images for the different methods for a cropped part of frame 20 of the sequence bandage_1 with full well 200. The frame has one big motion.

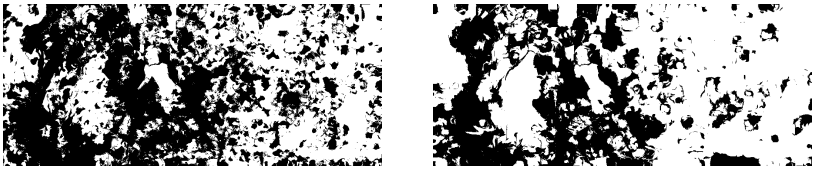
7.2 The usefulness of pre-processing

The algorithm that gains most of using the pre-processing step is Classic+NL-Fast, which always produces a better SSIM when pre-processing is used. Figure 7.8 - 7.10 show an example of the performance gain from pre-processing. By just looking at the flow in Figure 7.8, it is hard to see that the flow from the pre-processing is better. Consider the occluded pixels in Figure 7.9 instead. By using pre-processing the number of occluded pixels decreases from 51% to 37%, which leads to more temporal filtering and thereby less noise. The reason that the number of occluded pixels increases is that the backward flow and the forward flow do not match. It is easy to see that the result becomes better with pre-processing by zooming into the image (see Figure 7.10).



(a) Without pre-processing (b) With pre-processing

Figure 7.8: The generated flow from Classic+NL-Fast for frame 15 of the sequence market_2.



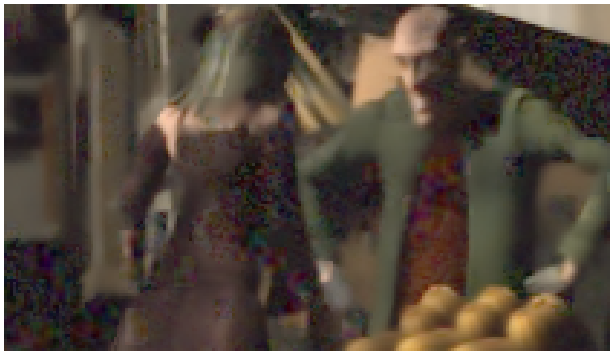
(a) Without pre-processing (51% occluded)

(b) With pre-processing (37% occluded)

Figure 7.9: The occluded pixel found by the finder step by estimating the flow with Classic+NL-Fast for frame 15 of the sequence market_2.

SimpleFlow does not respond very well to pre-processing. In many cases, the result is considerably worse when pre-processing is used. It is only in one out of the twenty cases presented in Chapter 6, that SimpleFlow has a better SSIM in the case of pre-processing, namely for the noisiest sequence, i.e. real_1 with full well 50 electrons. EPPM shows the same pattern. It receives better result without pre-processing except for real_1 with full well 50, 100 and 200 electrons, but the pre-processing does not decrease the SSIM as much as for SimpleFlow for the other cases. LDOF performs better without

(a) Without pre-processing



(b) With pre-processing

Figure 7.10: A filtered part of frame 15 of the sequence market_2 when Classic+NL-Fast is used to estimate the optical flow.

the pre-processing on the lowest noise levels (full well 800 and 400 electrons for all sequences and full well 200 electrons for market_2 and real_2), but better with pre-processing for sequences with more noise.

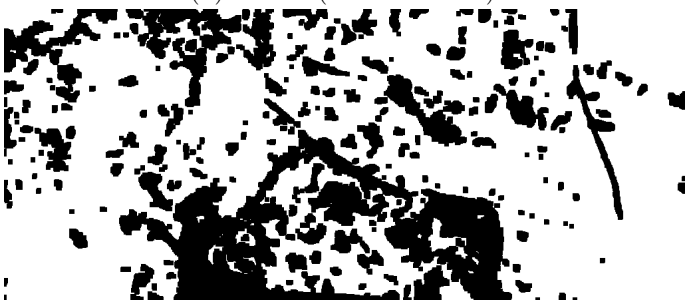
In total, the analysis shows that pre-processing is most useful for Classic+NL-Fast and most devastating for SimpleFlow. Regarding EPPM and LDOF, it is often preferable to skip the pre-processing, but when the level of noise is high, the pre-processing can increase the SSIM, especially for LDOF.

7.3 Occlusions

The four optical flow algorithms use the same technique to find occluded pixels. Some of the algorithms have much difference between the backward and forward flow, which leads to that many pixels are considered occluded after the first finder step (Table 6.4, 6.6, 6.11, 6.13, 6.18, 6.20, 6.25 and 6.27) in the occlusion detection algorithm (see Section 4.5.2). The idea of the last three steps of the occlusion algorithm is to eliminate incorrect flow. Generally, the occlusion algorithm finds a lot of occlusions from the flow generated by Classic+NL-Fast. Both the tables about occlusions in Chapter 6 and Figure 7.11 confirm that.

(a) Ground truth (4% occluded)

(b) LDOF (17% occluded)

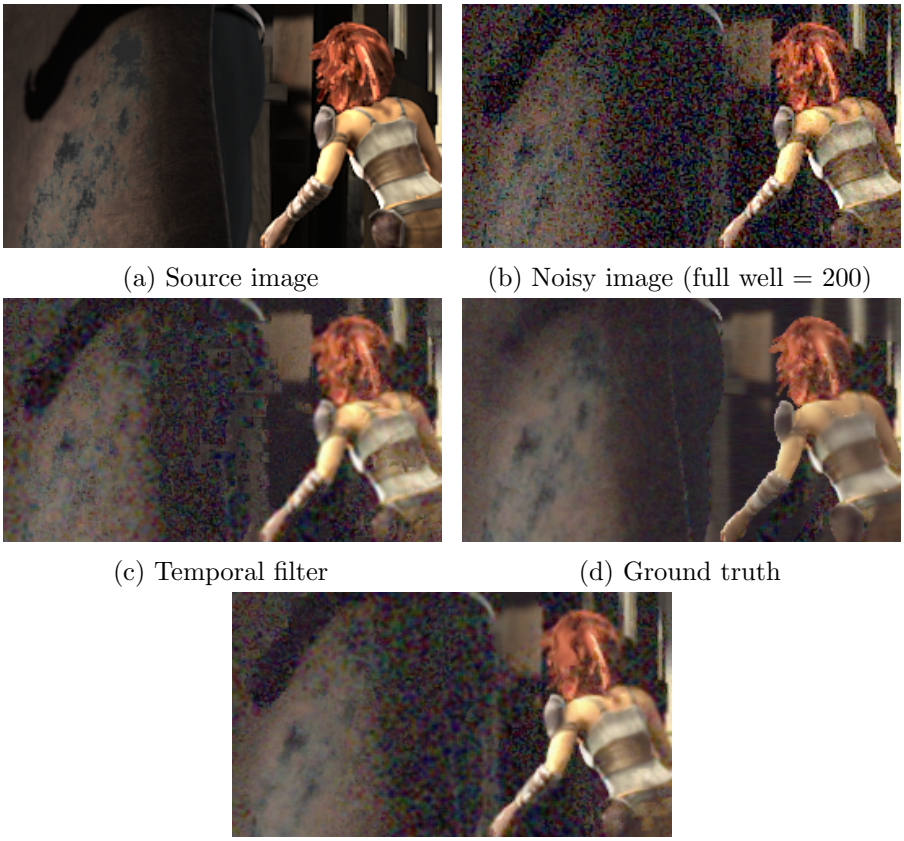


(c) Classic+NL-Fast (31% occluded)

Figure 7.11: Occlusion mask for frame 20 of the sequence bandage_1 with full well 400. For the two optical flow estimation the images show the occlusion map after the last filler step.

7.4 Illumination changes

When objects are moving in a scene, they are going to be affected by illumination changes. Let us consider Figure 7.12 as an example. In (a), the source image, there is a clear shadow in the top left corner. The same shadow appear in both (b), the noisy image, and in (e), the noisy image filtered by LDOF. For the noisy image filtered by the ground truth, (d), the shadow has almost disappeared. Moreover, the girl's back is quite light in the source image, but for the ground truth filter it is a bit darker.



(a) Source image

(b) Noisy image (full well = 200)

(c) Temporal filter

(d) Ground truth

(e) LDOF

Figure 7.12: A part of frame 15 of the sequence market_2 where: (a) is the source image without noise. (b) is the source image with noise. (c) is the noisy image filtered with the temporal filter. (d) is the noisy image filtered with ground truth flow. (e) is the noisy image filtered with the flow from LDOF.

Chapter 8

Discussion

8.1 Discussion

In this chapter, the performance of the algorithms is first going to be discussed, followed by discussions about the algorithm selection, parameter analysis, effect of pre-processing, noise filters, occlusions, illumination changes, measurements and execution speed. The chapter ends with a conclusion and suggestions for future work.

8.1.1 The performance of the algorithms

There are several reasons why the four considered optical flow algorithms produce different results for the noise filter. Algorithms based on color consistency have a drawback for noisy image sequences because a pixel that is noisy in only one of the two frames will differ in color and thus not be considered a match. Gradient based algorithms will probably perform better. Patch matching is generally an advantage because it checks a greater area and not only one single pixel, so noisy pixels will have less effect on the matching.

SimpleFlow is a local algorithm. Therefore, it has problems with larger motions. It is based on patch matching, which makes it suitable for image sequences with high noise level. A disadvantage is that it uses color consistency for the matching.

EPPM has the advantages that it is based on patch matching and that it contains a weighted median filter, which will remove some of the noise. The largest drawback is probably the approximate nature of the algorithm. The self-similarity step chooses the neighbors that should represent the patch based on color consistency. Then, some pixels, which are really representative will be discarded because they are noisy. An even greater risk is that noisy pixels, which actually are very far in color from the center pixel will be picked

because they get a similar color because of the noise. This leads in that the chosen pixels may not be a good representation of the entire patch, which can lead to unsatisfactory tracking. Another problem with EPPM is the cost function. Apart from color consistency, it also contains the census transform. The census transform compares the pixel values in a 9×7 area around the current pixel and a pixel in the other frame. This metric will be misleading when the amount of noise is high and/or when the color differences within the area are small.

Classic+NL-Fast works in more scales than SimpleFlow and EPPM. This ought to be an advantage because the amount of noise decreases when an image is downsampled, provided that a good downsampling method is used. Some other advantages is the usage of smoothing and median filtering of the flow. Like the other algorithms, Classic+NL has the disadvantage of using color consistency. It relies more on color consistency than any of the other algorithms. The greatest drawbacks, which make Classic+NL-Fast produce a worse noise filtering than the other algorithms is the total lack of patch matching and usage of gradients.

LDOF uses even more scales than Classic+NL-Fast. Like Classic+NL-Fast, a good downsampling method could decrease the amount of noise, which makes the optical flow estimation more accurate. One of the terms in LDOF's energy function is based on color consistency, which is a disadvantage for noisy sequences. The other four terms in LDOF's energy function are based on smoothing, gradients and descriptors. The smoothing prevents the effects from the noise by removing discontinuities of the flow. The gradient consistency is not as noise sensitive as the color consistency as long as the amount of noise is not very high. The descriptors are based on patches, and could thereby handle noise quite well. In total, this makes LDOF a suitable algorithm for noisy image sequences.

8.1.2 Algorithm selection

As described in Section 3.1, the selection of optical flow algorithms is based on the MPI-Sintel benchmark. This choice was made after advise from the supervisors of the thesis [26]. Only algorithms with available papers, present code and not too long execution time were considered. Furthermore, the selection took into account that the chosen algorithms should represent different types of optical flow algorithms. Four of the over forty algorithms present at MPI-Sintel [33] have been evaluated and compared in this thesis.

The algorithms' positions at 11th of May 2015 on the MPI-Sintel benchmark is 23 (EPPM), 32 (LDOF), 37 (Classic+NL-Fast) and 40 (SimpleFlow) [33]. In other words, there exist several algorithms which give a more accurate flow.

Many of them are also faster than both LDOF and Classic+NL-Fast [35]. Of course, some of these algorithms may be better suited for finding optical flow in noisy image sequences than the chosen ones. If the choice was based on Middlebury or KITTI instead of MPI-Sintel, the same four algorithms would definitely not have been picked because SimpleFlow is not present on KITTI and Classic+NL-Fast is not present on Middlebury. In the same manner, many algorithms from Middlebury and KITTI do not appear in MPI-Sintel. Therefore, they have not been considered at all, even if they very well can be both faster, more accurate and easier to implement. Furthermore, optical flow is an evolving subject where new algorithms are developed all the time. All this together leads to that there is a great possibility of finding better suited optical flow algorithms than those tested in this thesis.

8.1.3 The effect of parameter analysis

For SimpleFlow and EPPM, a parameter analysis (see Section 3.2) has been performed in order to optimize the parameters for the type of image sequences used in this report. In this parameter analysis, one of the test image sequences, namely bandage_1, was used. Thus, there exist a risk for SimpleFlow and EPPM being overfitted to this sequence and therefore having an extra advantage in the comparison. This can be the reason that EPPM, compared with the other algorithms, performs best on bandage_1. Furthermore, due to the long execution time, no such parameter analysis was performed for Classic+NL-Fast or LDOF, but all parameter choices in the MATLAB implementations remained unchanged. These parameters are most likely not set with noisy image sequences in mind and can therefore potentially lead to Classic+NL-Fast and LDOF having a disadvantage in the total comparison.

8.1.4 The effect of pre-processing

According to the analysis (see Section 7.2), SimpleFlow performs worse on pre-processed images. Most likely, this depends on that SimpleFlow already contains a pre-processing step. In that step, the algorithm performs different actions on the pre-processed image and the original one. By first doing an extra pre-processing, the image SimpleFlow consider as original is already pre-processed, which will result in a small or no difference between the two images and will thus cause the built-in pre-processing in SimpleFlow to have no effect.

Classic+NL-Fast, on the other hand, responds very well on the pre-processing. One of the reasons is that Classic+NL-Fast is mainly based on color consistency and relies more on this property than any of the other optical flow algorithms. By using pre-processing, the main part of the pixels are going to get pixel values which are closer to the noise free image. The color consistency

assumption part in the algorithm is then going to prefer a more correct flow.

In the analysis, one conclusion was that LDOF performs better for sequences with high noise level when pre-processing is used. One possible reason is that the color consistency term in the energy equation for LDOF has too much impact on the result. For the same explanation as for Classic+NL-Fast in the previous section, the pre-processing helps the color consistency term to produce a more correct flow. Another approach to handle sequences with much noise is to change the weight of the color consistency term, and in that way give the other, less noise sensitive terms more impact.

EPPM performs worse for nearly every pre-processed image. An explanation for that is that EPPM is an edge-preserving algorithm. The pre-processing smooths out the image, and therefore makes the edges less distinct.

A recommendation is to use the pre-processing described in Section 3.4.1 for Classic+NL-Fast. For SimpleFlow, only the built-in pre-processing in the algorithm, and no external pre-processing, ought be used. For EPPM, the pre-processing does seldom lead to any improvement. Finally, for LDOF, the pre-processing should be used when the level of noise is high, but not otherwise.

8.1.5 The performance on different image sequences

According to the result, the temporal filter performs best, compared with the optical flow based filtering on real_2 and on some noise levels on real_1. A reason for this is that the motions in real_1 and real_2 are limited to a small part in each image frame. Furthermore, the backgrounds are static, while there are motions in the background in market_2. A sequence with many static parts favors the temporal filter.

A way to further improve the performance of the optical flow steered filter compared to the temporal filter, would be to only choose test sequences with more motion, like market_2. There are mainly two reason to also include image sequences with less motion in the thesis' dataset. Firstly, the aim is to create a dataset that includes many different kinds of motion and thus get a broader and more general analysis about how well the usage of optical flow can improve the performance of a temporal filter. Secondly, real_1 and real_2 are interesting because they are more realistic in a surveillance prospective than an image where the entire background is moving.

8.1.6 Noise filter

Optical flow and noise filtering are both large, broad research fields. To focus on both would be too much work for a master thesis, so the main focus is on the optical flow algorithms. Thus, the noise filter implemented is relatively simple and leaves room for improvements.

The optical flow filter in this thesis is based on the temporal filter. The filter requires both perfect flow and occlusion knowledge to perform a perfect filtering. This is though difficult, if not impossible, to get. The occlusion finder, remover and filler are steps to solve the problem with incorrect flow and occlusions.

An alternative to the currently used noise filter would be to develop a patch based noise filter that uses optical flow. Such a filter would consider a small patch around each pixel in the first frame and search for similar patches nearby the place that the forward flow points at in the second frame. An advantage with this kind of filter is that the flow does not need to be perfect, while a drawback is that the patch search is quite time-consuming.

8.1.7 Occlusions

A perfect occlusion mask can only be obtained if the forward and backward flow is exact. To find exact flow is an unsolved problem. Because of that, the occluded areas need to be estimated instead. In the thesis, one approach for doing so is given, but several other methods could be used and some of them could be better for optical flow noise reduction.

The second step (detect) in the thesis occlusion algorithm will have problems with some edges of moving objects. Imagine the case when the background color next to the border of the moving object has a big color difference between two frames. Then, the mean color difference of that border pixel is going to be too big between the two frames and the pixel is going to be marked as occluded even if it is not.

The occlusion algorithm has a remover and a filler step at the end. An idea is to modify the algorithm to just do the removing and filler step on the pixels detected in the second part of the occlusion algorithm. In the approach of this thesis the remover and filling part is also done on the found occluded pixels in the first step. If one or the other approach is the best is not easy to answer because the first step could also find incorrect occlusions which could be solved by the last two steps in the approach.

The main reason to use a filler step at the end of the occlusion detec-

tion is to increase the area around the uncertain pixels. The probability to use pixels with totally wrong color decreases, by increasing the number of occluded pixels. It leads to that the filtered image get less artifacts.

In the filter for the optical flow algorithms, the output pixel for the occluded pixels has been set to the average of the pixel values of an area around the latest frame's pixel. This is not the only approach to handle occluded areas. Another approach is to apply the temporal filter on the occluded pixels.

8.1.8 Illumination changes

Effects which are hard for the noise filter to handle are illumination changes and shadows (see Section 7.4), i.e. situations where the brightness constancy assumption mentioned in Section 2.5 is not fulfilled.

The analysis shows that the ground truth flow produces a filtered image with illumination changes. Shadows have disappeared and the brightness is different for several parts in the image. The reason is that the ground truth filter averages the corresponding pixels from several frames for a pixel of a moving object and thereby the pixel's illumination is going to change. For example, if the pixel does not have a shadow on it most of the time, the shadow is going to disappear in the filtered image.

Regarding shadows, is it okay to take them away like the ground truth filter does? Maybe for some applications and for others not. If the goal is to noise filter an image, so it becomes as similar to the noise free image as possible, illumination changes need to be handled. The ground truth optical flow does not care about shadows, transparency and specular reflections. It means that the ground truth is not optimal as optical flow for filtering. To solve a part of the problem, one idea to find optical flow for both objects and shadows, and combine the results when filtering.

According to the analysis, the noise filter based on the ground truth just perform slightly better than the other algorithms, when the noise level is low. For the sequence bandage_1 with least noise (full well 800 electrons), LDOF actually beats the ground truth. One of the reasons for this is the problem with illumination changes, described above.

8.1.9 Measurements

In the result tables in Chapter 6, the SSIM error measure (see Section 2.4.1.3) is used to compare the performance of the temporal filter with noise filtering based on the different optical flow algorithms. As has been described in

Section 2.4.1, there exist several other types of metrics that can be used to measure the amount of noise in an image, for instance MSE and PSNR. The reason for choosing SSIM is that it is based on perceptual similarity [27], which means that it favorites image sequence, which look good for the human eye, which is usually the result one want to achieve in imaging.

The SSIM values presented in Chapter 6 are measured between a filtered image sequence and the input sequence with no added noise. To achieve a high SSIM, the filtered image must come as close as possible to the original sequence. But the goal of noise filtering is to remove all noise. Therefore, the assumption that the original sequence contains no noise is made indirectly. This is not true for the sequences real_1 and real_2, which have been filmed at Axis during the thesis and contains some natural noise. A good enough noise filtering algorithm cannot only remove the added artificial noise but also the original noise and thus produce a better image sequence than the original one. The filtered image sequence will then be less similar to the original than a sequence produced by a filter that only removes the added noise and thus get a lower SSIM even if it is a better algorithm.

A way to overcome this unfairness in the measurement would be to first perform a denoising of the original sequence using the best available algorithm. A suggestion is ground truth when it is available and LDOF otherwise. In this way, most of the natural noise is removed. The filtered sequence can then be used as source sequence to put artificial noise on. When this sequence is filtered by one of the noise filters, almost no natural noise can be removed (because there is almost no such noise left), why an image sequence produced by a very good noise filter will be similar to the source sequence and thus get a high SSIM.

The focus in this thesis is to compare how well the optical flow algorithms can increase the performance of a temporal noise filter. An important ability for an optical flow algorithm which should be used in a noise filter is that it produces an accurate flow for noisy image sequences. An optional way to find a suitable flow algorithm would be to measure how accurate optical flow fields the algorithms produce. There are several reasons why this approach has not been used in this thesis. First, it requires a ground truth optical flow to compare with, which are only present for the MPI-Sintel sequences and not for real_1 and real_2. Furthermore, even if a good flow is an advantage, is it not certain that an exact flow is required for the noise filtering. There are already several benchmarks like Barron [10], Middlebury [1], MPI-Sintel [2] and KITTI [3] that compares the accuracy of the flow for different kinds of image sequences, even if none of them focuses on noisy sequences. This kind of measurement are thus already commonly occurring, while the actual improvement an optical flow algorithm can have on a temporal noise filter

has not been very well investigated before.

8.1.10 Execution speed

LDOF is not only the best of the optical flow algorithms considered, but also the slowest. To calculate the forward or backward flow between two image frames take almost 5 minutes, compared with about the half for Classic+NL-Fast and not more than a few seconds for EPPM and SimpleFlow.

Even if the speed of a MATLAB implementation does not say very much about the possibility of implementing LDOF in real time in camera hardware, the long execution time may frighten people to even try. The authors of the LDOF paper say though that there exist GPU implementations of classic warping methods that run in 30 frames per second on VGA images. Furthermore, they say that their approach is also feasible for that and that their additional descriptor matching part fits very well for parallel hardware [9]. In other words, the possibilities to extend LDOF to real time are probably good.

The pre-processing step from Section 3.4.1 takes a very limited amount of time, compared with the total execution time of the algorithms. Thus, it is still a good idea to use it together with LDOF when the noise level is high, even though the execution time should be minimized.

Overall, the execution time of all the considered optical flow algorithms, as well as the noise filter could be decreased. The focus of the thesis has been to optimize the performance and not the execution time of the code. One such example on an easy improvement are to use faster sorting algorithms in the median filters.

8.2 Conclusion

In this thesis, it has been shown that the usage of optical flow increases the performance of a temporal filter. Out of the four optical flow algorithms considered (SimpleFlow, EPPM, Classic+NL-Fast and LDOF), LDOF is suitable to use when the amount of noise is moderate, while SimpleFlow is preferable for image sequences with high noise level. The EPPM algorithm has some problems with noisy image sequences. Classic+NL-Fast performs the worst of the tested algorithms, but get slightly helped if the suggested pre-processing is used together with it.

An optical flow algorithm that should be used to steer a noise filter must produce a very accurate optical flow and find accurate occlusions. To manage this, the algorithm should optimally rely more on gradient consistency than on color consistency. It should use patch matching, but without the

approximation techniques used in EPPM. Finally, it should use descriptors and apply smoothing and a median filter on the optical flow.

8.3 Future work

In the analysis in this thesis, the focus was set on algorithms that can be extended to real time implementations. An idea is to instead use optical flow for post processing of noisy image sequences. Then, the execution speed of the optical flow algorithm used is not as big problem and other, slower, but more accurate algorithms could be used.

LDOF performs well, but was the slowest of the four optical flow algorithms considered. The implementation was written in MATLAB and ran on the CPU. A natural continuation of the work in this thesis would be to try to parallelize LDOF and implement it on the GPU to get a fair performance comparison between the GPU implemented algorithms and LDOF and investigate if it is realistic to run LDOF in real time. There are also a lot of other optimizations of the code that can be done, in order to further speed both LDOF and the other algorithms up. Finally, a parameter analysis to optimize the parameters to noisy image sequences, similar to the one performed on SimpleFlow and EPPM, would probably give an even better accuracy for LDOF.

If LDOF after the improvements described above beats SimpleFlow even on images with high noise level, the future work ought to be concentrated on improving LDOF further. Otherwise, even SimpleFlow is in question for further research and improvements. EPPM and Classic+NL-Fast produce too bad result for the noise filter to be considered interesting to develop further.

Another option is to use the knowledge and analysis from this report and combine the best parts of each algorithm to develop a novel optical flow algorithm, specialized on noisy image sequences. An idea is to let such an algorithm take several frames into account when estimating the optical flow. The state-of-the-art algorithms do not use the flow calculated from previous frames. Since the objects normally move in the same direction for several frames, the optical flow can be used as an initial guess for the next frame. This means that heavy calculations for an initial guess just need to be done for the first pair of frames.

Independently of the choice of algorithm, a final step would of course be to develop and implement a more sophisticated noise filter built on an estimate of optical flow, first in software and then in the hardware on a real camera.

Bibliography

[1] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision*, vol. 92, pp. 1–31, 2011.

[2] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *European Conference on Computer Vision*, pp. 611–625, Springer, 2012.

[3] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *CVPR*, 2012.

[4] B. Horn and B. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[5] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *DARPA Image Understanding Workshop*, pp. 121–130, 1981.

[6] M. Tao, J. Bai, P. Kohli, and S. Paris, "Simpleflow: A non-iterative, sublinear optical flow algorithm," *Eurographics*, vol. 31, no. 2, 2012.

[7] L. Bao, Q. Yang, and H. Jin, "Fast edge-preserving patchmatch for large displacement optical flow," in *CVPR*, 2014.

[8] D. Sun, S. Roth, and M. J. Black, "Secrets of optical flow estimation and their principles," in *CVPR*, pp. 2432–2439, 2010.

[9] T. Brox and J. Malik, "Large displacement optical flow: Descriptor matching in variational motion estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, pp. 500–513, 2011.

[10] J. Barron, D. Fleet, and S. Beauchemin, "Performance of optical flow techniques," *International Journal of Computer Vision*, vol. 12, pp. 43–77, 1994.

[11] A. Buades, B. Coll, and J. Morel, "Nonlocal image and movie denoising," *International Journal of Computer Vision*, vol. 76, pp. 123–139, 2008.

[12] A. Buades, B. Coll, and J. Morel, "Denoising image sequences does not require motion estimation," in *IEEE International Conference on Advanced Video and Signal Based Surveillance*, 2005.

[13] R. Dudek, F. Quintana, and C. Cuenca, "Using optical flow to reduce noise in image sequences," in *International Conference on Image Processing, Computer Vision, and Pattern Recognition*, 2009.

[14] C. Liu and W. Freeman, "A high-quality video denoising algorithm based on reliable motion estimation," in *European Conference on Computer Vision*, Springer, 2010.

[15] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2010.

[16] M. J. Black and D. J. Fleet, "Probabilistic detection and tracking of motion boundaries," *International Journal of Computer Vision*, vol. 38, no. 3, pp. 231–245, 2000.

[17] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, "Bilateral filtering: Theory and applications," *Foundations and Trends in Computer Graphics and Vision*, vol. 4, no. 1, pp. 1–73, 2008.

[18] L. Yin, R. Yang, M. Gabbouj, and Y. Neuvo, "Weighted median filters: A tutorial," *IEEE Transaction on circuits and systems-11: Analog and digital signal processing*, vol. 43, no. 3, pp. 157–192, 1996.

[19] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," in *ACM SIGGRAPH*, 2007. Microsoft Research.

[20] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application*, pp. 331–340, 2009.

[21] Schorsch.com, "Lightning design glossary." `http://www.schorsch.com/en/kbase/glossary/luminance.html`. , accessed: 2015-02-05.

[22] D. Pascale, *A review of RGB color spaces ...from xyY to R'G'B'*. BabelColor, 2003.

[23] Computer Science Department, B. Thomas Golisano College of Computing and Information Sciences, "Color conversion algorithms." `http://www.cs.rit.edu/~ncs/color/t_convert.html`. , accessed: 2015-03-23.

[24] S. Yu, Z. Ping, X. Jiangtao, G. Zhijuan, and X. Chao, "Full well capacity and quantum efficiency optimization for small size backside illuminated cmos image pixels with a new photodiode structure," *Journal of Semiconductors*, vol. 33, no. 12, 2012.

[25] R. Bell, D. Burt, I. Moody, C. D. Mackay, and R. N. Tubbs, "Sub-electron read noise at mhz pixel rates," in *Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications II*, 2001.

[26] F. Olofsson and G. Dahlgren, "Supervisors at Axis Communications AB." personal communication.

[27] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on image processing*, vol. 13, pp. 600–612, 2014.

[28] L. I. Rudin, *Images, Numerical Analysis of Singularities and Shock Filters*. 1987.

[29] G. Aubert, R. Deriche, and P. Kornprobst, "Computing optical flow via variational techniques," *SIAM Journal on Applied Mathematics*, vol. 60, pp. 156–182, 1999.

[30] D. Sun, S. Roth, and M. J. Black, "A quantitative analysis of current practices in optical flow estimation and the principles behind them," *International Journal of Computer Vision*, vol. 106, pp. 115–137, 2014.

[31] D. Fortun, P. Bouthemy, and C. Kervrann, "Aggregation of local parametric candidates with exemplar-based occlusion handling for optical flow," *IEEE Transactions on Image Processing*, vol. 1407.5759, 2014.

[32] F. Opitz, "Markov random fields and the optical flow," in *Informatik*, 2011.

[33] Max Planck Institute for Intelligent Systems, "Mpi sintel dataset, results and ranking." `http://sintel.is.tue.mpg.de/results`. , accessed: 2015-05-11.

[34] Middlebury, "Optical flow evaluation results." `http://vision.middlebury.edu/flow/eval/results/results-e1.php`. , accessed: 2015-03-16.

[35] A. Geiger, "The kitti vision benchmark suite, optical flow evaluation." `http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=flow`. , accessed: 2015-02-20.

[36] Middlebury, "Middlebury optical flow datasets." `http://vision.middlebury.edu/flow/data/`. , accessed: 2015-02-24.

[37] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof, "Anisotropic huber-l1 optical flow," in *British Machine Vision Conference*, pp. 1–11, 2009.

[38] R. Tsuchiyama, T. Nakamura, T. Iizuka, A. Asahara, J. Son, and S. Miki, *The OpenCL Programming Book*. Fixstars, 2012.

[39] P. Leeda, A. Chalmers, T. Troscianko, and H. Seetzen, "Evaluation of tone mapping operators using a high dynamic range display," in *ACM SIGGRAPH*, pp. 640–648, 2005.

[40] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "PatchMatch: A randomized correspondence algorithm for structural image editing," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 28, no. 3, 2009.

[41] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *European Conference on Computer Vision*, pp. 151 – 158, 1994.

[42] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang, "On building an accurate stereo matching system on graphics hardware," in *IEEE International Conference on Computer Vision Workshops*, 2011.

[43] G. Rong and T.-S. Tan, "Jump flooding in gpu with applications to voronoi diagram and distance transform," in *ACM Symposium on Interactive 3D Graphics and Games*, pp. 109–116, 2006.

[44] L. Bao, Y. Song, Q. Yang, and N. Ahuja, "An edge-preserving filtering framework for visibility restoration," in *IEEE International Conference on Pattern Recognition*, pp. 384–387, 2012.

[45] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D*, vol. 60, pp. 259–268, 1992.

[46] R. Vincent and O. Folorunso, "A descriptive algorithm for sobel image edge detection," in *Informing Science and IT Education Conference*, pp. 97–107, 2009.

[47] M. Nielsen, *Graduated non-convexity by smoothness focusing*. Citeseer, 1993.

[48] P. Sand and S. Teller, "Long-range motion estimation using point trajectories," *International Journal of Computer Vision*, vol. 80, pp. 72–91, 2008.

Master's Theses in Mathematical Sciences 2015:E15
ISSN 1404-6342

LUTFMA-3275-2015

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

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