Department of Physics and Astronomy

University of Heidelberg

Diploma thesis

in Physics

submitted by

Joachim Schleicher

born in Rottweil

2011

Image Processing for Super-Resolution Localization Microscopy Utilizing an FPGA Accelerator

This diploma thesis has been carried out by Joachim Schleicher

at the

Interdisciplinary Center for Scientific Computing

under the supervision of

Prof. Dr. Fred A. Hamprecht

Bildverarbeitung für Höchstauflösende Mikroskopie beschleunigt durch FPGA framegrabber:

Optische Mikroskopie ist die Grundlage der Zell-Biologie und erlaubt Einblicke in die Entwicklung lebender Organismen. Höchstauflösende dSTORM Mikroskopie überwindet die Abbe'sche Auflösungsgrenze durch die zeitliche Trennung von Bildpunkten, die räumlich dichter liegen als die halbe Wellenlänge. So lassen sich Auflösungen von 20 nm mit dem Aufbau von gewöhnlichen Fluoreszenz-Mikroskopen erreichen.
Die rechenintensive Rekonstruktion des Bildes aus so aufgenommenen Zeitserien wird in der vorliegenden Arbeit untersucht. Im ersten Teil werden die algorithmischen Grundlagen einer auf Standard-Bildverarbeitungs-Operatoren basierenden Auswertung behandelt. Die halb-automatische Registrierung mehrerer Farb-Kanäle erlaubt die biologische Untersuchung von Interaktionen innerhalb einer Zelle.
Durch den Einsatz konfigurierbarer Hardware ist es möglich, den Datenfluss direkt auf dem Weg von der Kamera zum Host PC zu verarbeiten und die zu speichernde Datenmenge deutlich zu reduzieren. Ein hochauflösendes Bild der Probe stünde dann in Echtzeit zur Verfügung. Die Anwendbarkeit von aktuell auf dem Markt verfügbaren FPGA-Framegrabbern für diese Aufgabe wird im zweiten Teil der Arbeit untersucht und diskutiert.

Image Processing for Super-Resolution Localization Microscopy Utilizing an FPGA Accelerator:

Cell biology gains important insights into living organisms by using optical microscopy. Super-resolution dSTORM microscopy overcomes Abbe's resolution limit by temporal separation of fluorescent markers that are closer to each other than half the wavelength of their emitted light. In this way a resolution of 20 nm is achieved with a standard fluorescence microscope setup.
Processing the data of the acquired image stack is computationally expensive. We present an algorithm to efficiently assemble the underlying image from the measured time series. Standard image processing algorithms allow the precise localization of fluorescent spots without fitting each individual point-spread-function. To align multiple super-resolution images we apply image registration techniques. Multi-color images give biologists the possibility to explore interactions inside cells.
Besides an optimized C++ implementation it is desirable to simplify and accelerate the data flow. We studied the applicability of on the market FPGA framegrabbers to process the raw images on-the-fly. Then a realtime-preview of the sample would be available to the operator of the microscope.

Acknowledgments – Danksagungen

Bei der Erarbeitung der vorliegenden Arbeit haben mich viele Personen unterstützt. Vielen Dank für alle wertvollen Hinweise und Diskussionen! Danke, Ulli für die vielen Tipps und algorithmischen Vorschläge. Danke Michael, für deine Arbeit zur Rauschfilterung der dSTORM Daten.

Ein FPGA-Framegrabber wurde mir freundlicherweise von der Firma SiliconSoftware zur Verfügung gestellt. Danke an Dr. Ralf Lay und Johannes Trein für den Support und motivierende Gespräche. Wolfgang Mischler hat mir den Einstieg in VisualApplets erleichtert und wertvolle Hinweise und Hilfestellungen gegeben.

dSTORM-Daten wurden uns von der Gruppe von Prof. Dr. Mike Heilemann zur Verfügung gestellt. Danke Benni, für die unkomplizierte Zusammenarbeit, schnelles Feedback und Software-Tests.

Frederik und Manfred möchte ich danken für den Austausch und Vergleich von algorithmischen Details zur Datenauswertung sowie Anregungen zum Test auf simulierten Daten.

Nicht zuletzt gilt mein Dank Sven fürs Korrektur-Lesen und konstruktive Hinweise zum Aufbau dieser Arbeit.

Contents

1 Introduction

How do cells develop? Where do diseases come from? What interactions in cells allow viruses to proliferate? These are typical questions that the field of life sciences, in particular biology and medicine are interested in. Deeper insights into cellular processes require progress at several fronts. But better microscopic methods with higher resolutions can certainly lead to a better understanding of how cells develop.

In this thesis we show an example of how such research can profit from interdisciplinary collaboration. We introduce data processing procedures for fast analysis of super-resolution microscopy data that could not be handled without automatic analysis. Huge amounts of data are acquired to reconstruct a single image. Methods from image processing have proven useful to get a rapid preview of the underlying sample structure.

Images with a higher resolution allow the discovery of much more detail. Cellular processes can be found on micrometer and nanometer scales. The size of single molecules reaches from less than one up to tens or even hundreds of nanometers. Super-resolution optical microscopy achieves now a resolution of 20 nm as described below.

If the biological sample of interest does not need to be studied in vitro, there are many other imaging methods apart from optical microscopy. Electron microscopy is the most prominent amongst them and can achieve even atomar resolution. However, the study of living tissue is of particular importance, when studying cell interactions. Thus we stay with light microscopy as a non-destructive method. In general an optical microscope works in trans-illumination or reflective mode. Light is absorbed or reflected by cellular structures and that light is observed with high quality optics. But observing only the reflected light, transparent structures cannot be studied. It is useful to use dyes to mark structures inside cells selectively. *Fluorescence microscopy* uses organic or inorganic molecules or proteins that are coupled to cell structures of interest. With a laser of appropriate wavelength they are activated to their fluorescent state and emit light of a longer wavelength that is then observed.

The optical path of a light microscope was analyzed by Ernst Abbe: Structures smaller than half the wavelength cannot be resolved, since they are diffraction-broadened (Abbe, 1873). Given the wavelength of blue light at $\lambda \approx 400\,\text{nm}$ the resolution limit is at about 200 nm.

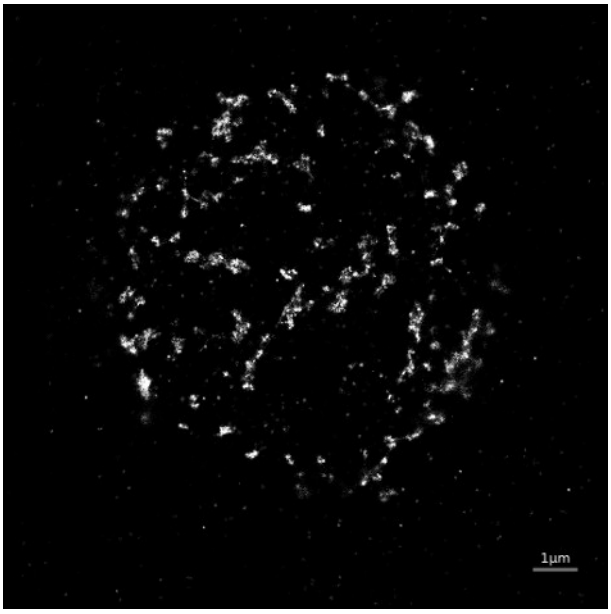

Figure 1.1: T-cell imaged using dSTORM microscopy.

Super-resolution microscopy

Several technologies to overcome this limit were invented during the last decade (cf. Galbraith and Galbraith, 2011; Heilemann, 2010):

Structured Illumination Microscopy (*SIM*) can be mentioned as a first example. A periodic illumination pattern allows for twice the resolution. The pattern is designed to utilize interference phenomena and is rotated at several angles to get an image of the full sample. That means several images have to be taken that are combined to a better-resolved image of the underlying structure.

Single fluorescent molecule microscopy allows to circumvent the Abbe limit with another trick. One diffraction broadened point-spread-function (PSF) can be localized precisely even if its width is blurred. Switching single fluorescent molecules on and off, images of a sample showing only few spots active at a time can be captured. After several thousand switching cycles a complete image of the sample is reconstructed. This data processing task shall be analyzed in detail in this thesis.

Betzig and Hess described this method 2006 independently from each other using photo-switchable proteins (PALM). Simultaneously Rust developed such a method using photo-switchable cyanine dyes and called it stochastic optical reconstruction microscopy (STORM).

Direct STORM (dSTORM) uses conventional fluorescent dyes that are switched reversibly (Heilemann et al., 2008). In contrast to STORM there is no need for an activator fluorophore. The fluorescent markers can be switched on and off directly by laser light of two different wavelengths.

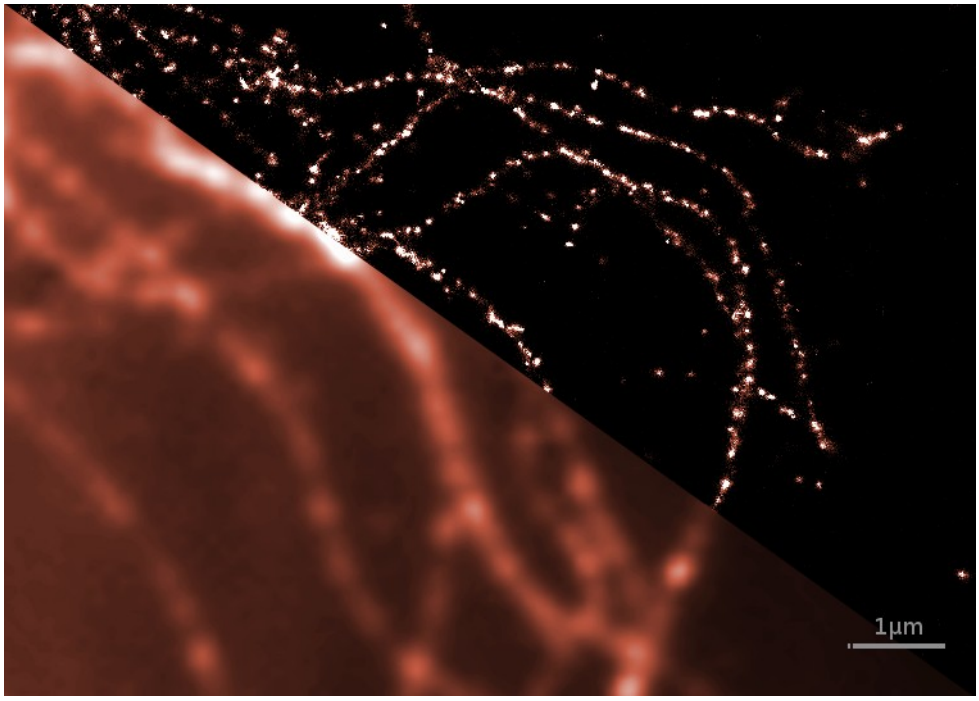

Figure 1.2: Microtubuli stained with Alexa532 at far-field resolution (left) and dSTORM super-resolution image reconstructed with our software (right). The complete image is shown in figure 2.1.

Figures 1.1 and 1.2 show images reconstructed from 10000 single frames. They were acquired using the dSTORM microscope in Bioquant, Heidelberg shown in figure 1.3. All dSTORM images depicted in this thesis have been acquired by members of the group of Prof. Dr. Mike Heilemann from university of Würzburg. Benjamin Flottmann from Bioquant Heidelberg measured the multi-color images, took the bead measurements and reviewed the localizations from our algorithm. By this collaboration we were able to test our algorithm and different processing methods on real world data.

To make the data processing applicable for every-day use of a microscope we had to deal with several challenges. The accuracy of single spot detection limits the resolution of the result image. So the single detections have to be as accurate as possible. Precision and recall should be optimized to improve the quality of the result image and decrease the number of images needed to resolve a certain structure. The image acquisition of an image stack takes several minutes. Afterwards the operator of the microscope should get a rapid feedback to plan the measurement of subsequent regions or details. A realtime-preview during the measurement would be of great benefit for the operation of a super-resolution microscope.

Currently a framegrabber captures the raw images and saves them to the hard

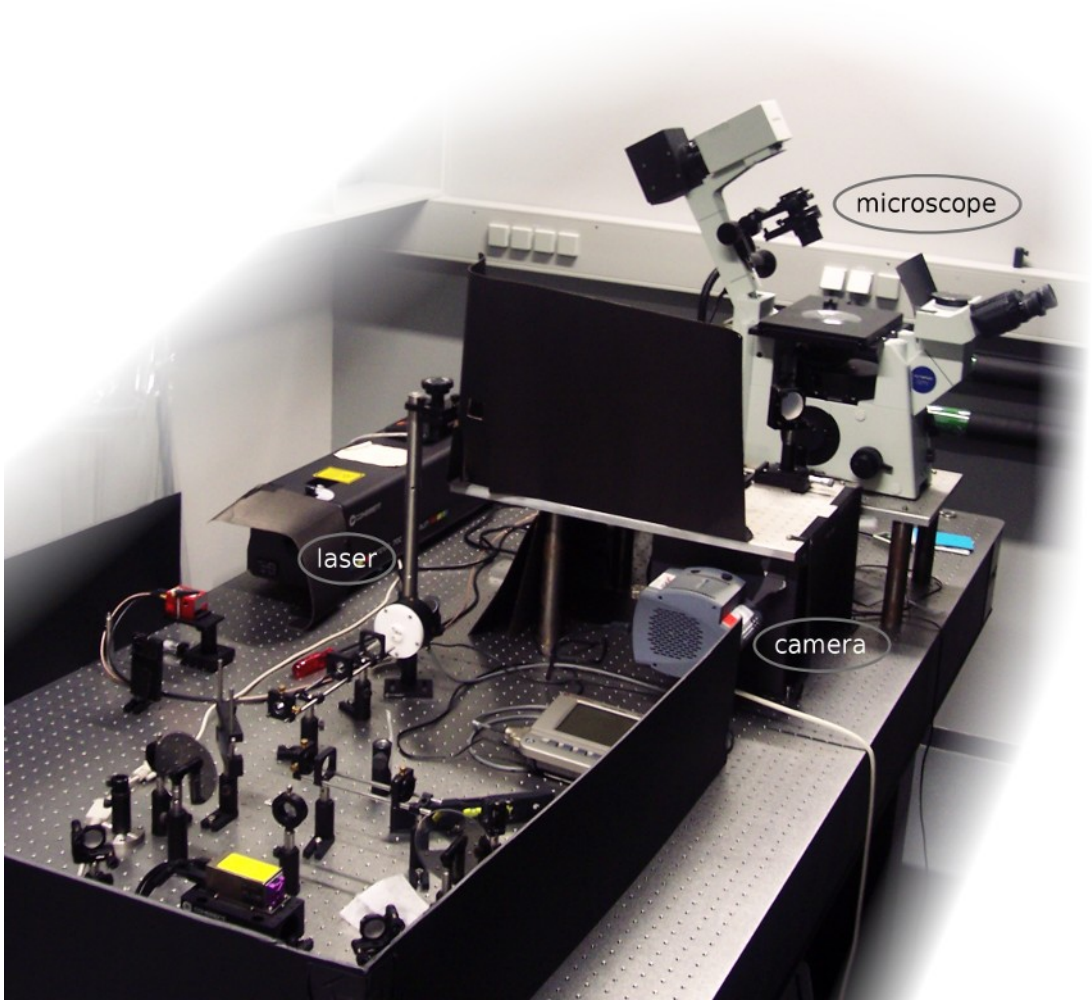

Figure 1.3: Fluorescence microscope used for dSTORM imaging

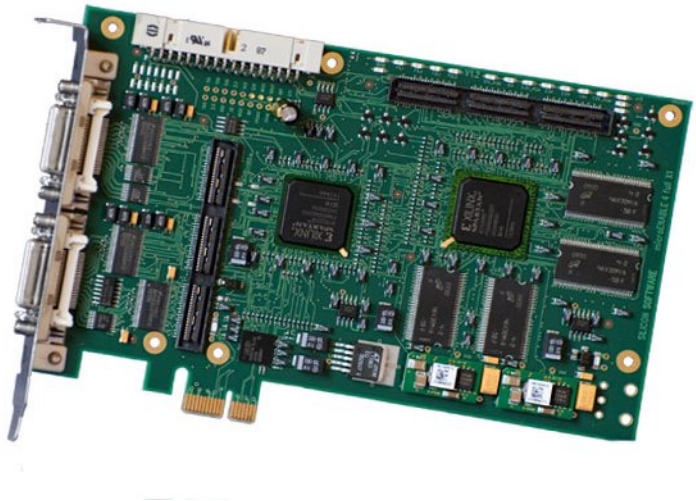

Figure 1.4: microEnable IV VD1-CL framegrabber from SiliconSoftware

disk of the PC. Before discussing different options of acceleration, we first want to take a closer look at computer architecture.

Parallel computer architecture

The central processing unit has an instruction set to execute different commands. Those commands are optimized on transistor level by the chip manufacturer. Following Moore's law the number of transistors inside processor chips doubles every two years. Recently increasing the number of cores has superseded increasing clock speed. Massive parallel computer architectures are available now and software will have to utilize this parallelism to improve processing speeds in the future.

Graphics processing units were initially built for special applications with special instruction sets. Today they have general-purpose computing capabilities (GPGPU) as well and integrate several hundred cores. This opens new possibilities to highly parallel compute-intensive problems in science.

For more special applications custom chips (ASICs) can be designed with much higher flexibility. Field Programmable Gate Arrays (FPGAs) are somewhere between ASCIs and GPUs: A FPGA consists of thousands of logic blocks and the internal connections can be programmed. These chips are much cheaper than a custom ASIC but the developer usually has to know the internal hardware structure and invest longer development times than a CPU-developer to use all the available power.

For processing the data from localization microscopy we started from a CPU implementation. Our algorithm for reconstruction of a dSTORM image is described in chapter 2. It can be applied to data acquired using PALM, STORM and dSTORM,

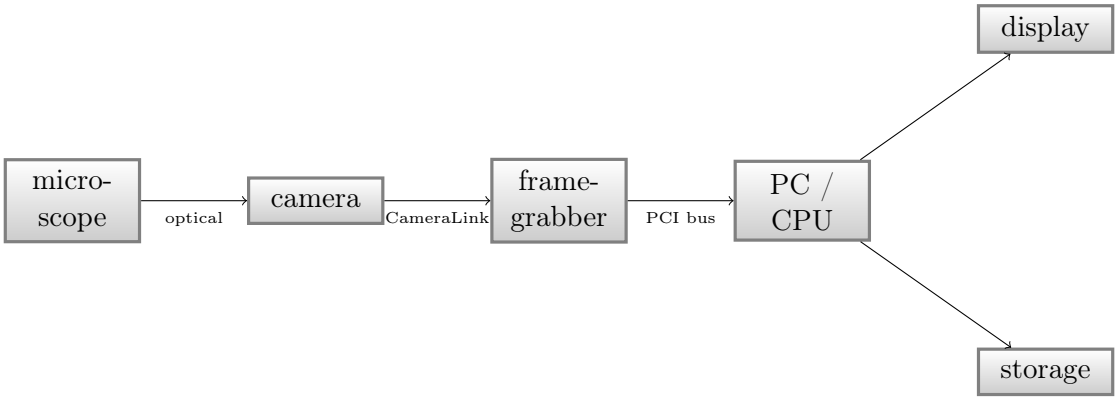

Figure 1.5: Data path for capturing and on-the-fly processing of dSTORM, STORM and PALM data

since they only differ in the use of different fluorescent markers.

Afterwards we show several options to measure the accuracy of single-molecule localization and of measuring the overall image resolution.

As an extension chapter 4 describes the registration of multispectral images. The high resolution of single images causes aberrations between different wavelengths to be non-negligible. We use landmark-based image registration techniques and align the images fitting a linear transformation.

The use of dedicated hardware can lead to significant speedups: Based on the VisualApplets solution from SiliconSoftware we built and tested an FPGA implementation on their microEnable IV VD1-CL shown in figure 1.4 (cf. Silicon Software, 2011a). The proposed data path is shown in figure 1.5. Details are given in the third part of this work.

Part I

Data processing for localization microscopy

2 Data processing

For reconstruction of a super-resolution image from localization microscopy several thousand images have to be analyzed. Hundred thousand single molecule point-spread-functions have to be accurately localized. Our collaborators in biology acquired several time series. We implemented a localization software and analyzed the accuracy and precision. The basics of our algorithm and some problems that had to be solved are discussed in this chapter.

2.1 Goal

The input data are a series of frames each containing a variable number of fluorescing spots. An example frame of input data is shown in figure 2.3 (left image). As output we want to store a coordinate list of detected molecules and create a super-resolved image from that spot positions. The resulting super-resolution image from that input stack is depicted in figure 2.1. It shows microtubulin structures labeled with conventional fluorophores.

During the past few years several algorithms reconstructing STORM, dSTORM or PALM data have been proposed. The first method was an iterative Gaussian-fit that is exact for Gaussian PSFs but the performance is rather slow (cf. Hess et al., 2006). Improvements include the FlouroBancroft algorithm (Andersson, 2007, 2008), rapidStorm (Wolter et al., 2010) and non-iterative methods calculating the center-of-mass (Grüll et al., 2011).

Our algorithm uses basic image processing routines and consists of the following four steps:

- Noise filtering. The images are noisy due to camera readout and photon counting effects. We compensate for that; afterwards the images look smoother and the spots more symmetric as can be seen in figure 2.8.
- Background subtraction. A baseline offset and out-of-focus signals are subtracted.
- Interpolation. To be able to localize the spots with sub-pixel accuracy we apply a spline interpolation to increase the image resolution.
- Maximum detection. After interpolation we simply suppress non-maxima pixels and report the positions of pixels that are larger than all their direct neighbors.

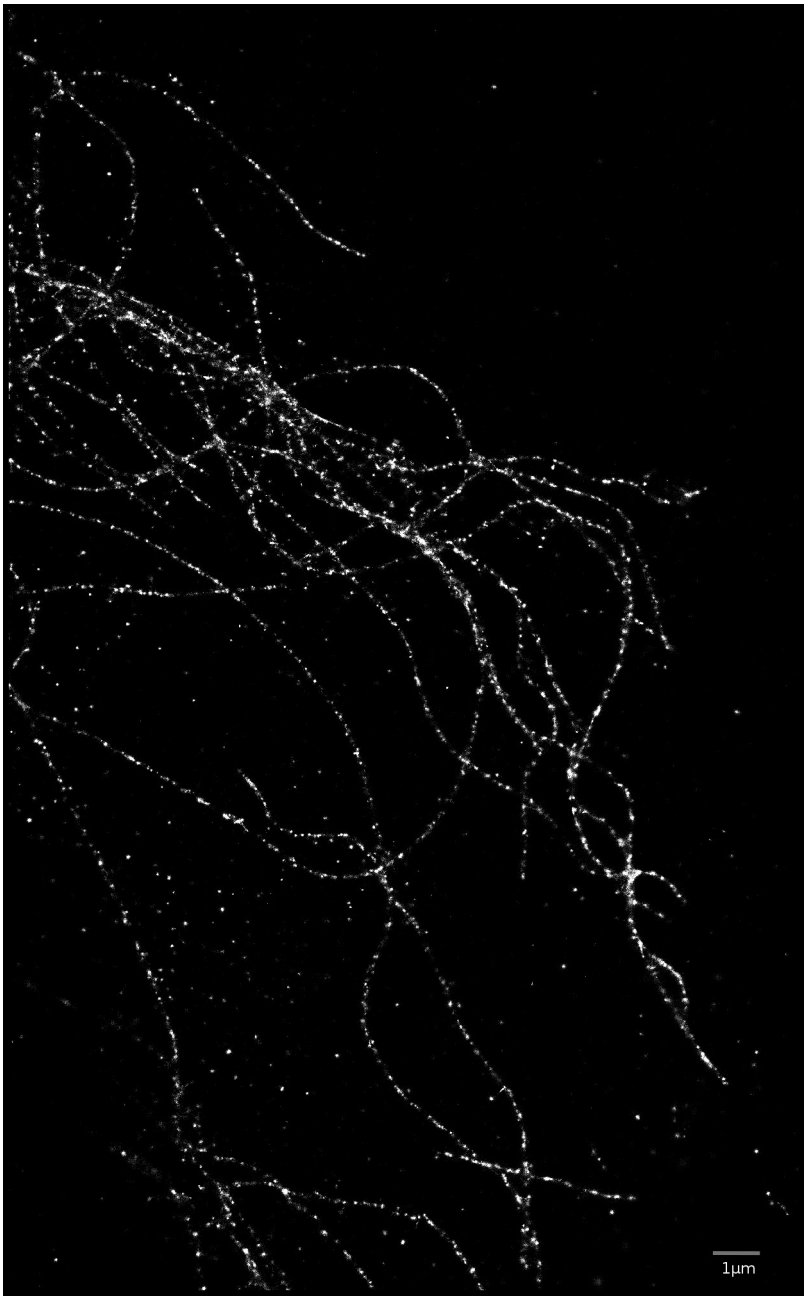

Figure 2.1: dSTORM image: microtubulin labeled with Alexa Fluor 532

To achieve maximum accuracy and minimal runtime we optimized each step of the algorithm and we'll give details in the following sections.

2.2 Noise filtering

One main ingredient for the dSTORM data processing algorithm is a Wiener filter for minimizing acquisition noise and suppressing other signals than the fluorescent single-molecule spots. The Wiener filter is applied to each frame individually; thus all frames can be analyzed in parallel. But before starting the processing step, the whole data set or even several different measurements are used to estimate the noise power and thus the filter characteristics.

Michael Hanselmann compared several other denoising filters for this task; in this section we'll just present the Wiener filter that gave the best results on our data.

Designing the Wiener Filter We assume that the images are corrupted independently from each other by white Gaussian noise

$$I_t(x, y) = J_t(x, y) + V_t(x, y),$$

where $J_t(x, y)$ is the unknown noise-free signal with a known power spectrum (see below). $V_t(x, y)$ is unknown additive noise independent of $J_t(x, y)$ with power density n. We want to determine an inverse filter function $G(x, y)$ such that we can estimate $J_t(x, y)$ from the measured signal

$$\hat{J}_t(x, y) = G(x, y) * I_t(x, y) = G(x, y) * [J_t(x, y) + V_t(x, y)] .$$

Under the assumption of Gaussian noise, selecting a Wiener filter for $G(x, y)$ is optimal in the sense that $\hat{J}_t(x, y)$ is the estimate that minimizes the mean square error (cf. Russ, 2011, page 379)

$$e_t^2(x, y) = \left(\hat{J}_t(x, y) - J_t(x, y)\right)^2 .$$

In Fourier domain, the Wiener filter for denoising with deconvolution with a known PSF $H(u, v)$ is described as

$$\mathcal{G}(u, v) = \frac{1}{H(u, v)} \cdot \frac{|H(u, v)|^2}{|H(u, v)|^2 + \frac{\mathcal{V}(u,v)}{\mathcal{S}(u,v)}}$$

where $\mathcal{G}(u, v)$ is the Fourier transform of $G(x, y)$ at frequencies u and v (cf. Russ, 2011, page 378). $\mathcal{S}(u, v)$ and $\mathcal{V}(u, v)$ represent the mean power spectral density of the undisturbed input signal $J_t(x, y)$ and the noise $V_t(x, y)$. We want to use the Wiener filter only for denoising and omit the deconvolution. Thus we set $H = 1$ and our denoising filter simplifies to

$$\mathcal{G}(u,v) = \frac{\mathcal{S}(u,v)}{\mathcal{S}(u,v) + \mathcal{V}(u,v)}. \tag{2.1}$$

Given the observed image $I_t(x,y)$, an estimate for the denoised image $\hat{J}_t(x,y)$ is obtained by multiplying the filter $\mathcal{G}(u,v)$ in Fourier domain with the input data followed by an inverse Fourier transform

$$\hat{\mathcal{J}}_t(u,v) = \mathcal{G}(u,v) \cdot \mathcal{I}_t(u,v).$$

Estimating the Mean Signal Power Spectrum and Noise Power To estimate the optimal Wiener filter according to equation 2.1, we need to estimate the signal and the noise spectral power. The power spectrum is the magnitude of the complex Fourier transformed image

$$P_t(u,v) = \|\mathrm{FFT}(I_t)(u,v)\|^2.$$

Averaging over a whole stack of images gives us the mean power spectrum $P \in \mathbb{R}^{H \times W}$

$$P(u,v) = \frac{1}{N} \sum_{t=1}^{N} P_t(u,v).$$

Neither the undisturbed signal spectrum nor the noise spectrum can be measured directly; we assume white Gaussian noise $\mathcal{N}(u,v) = n$ ($n \in \mathbb{R}$) distributed equally for all frequencies u, v. We assume further that the high frequency part of the mean power spectrum is pure noise and thus estimate the noise power as

$$n = \frac{1}{K - K'} \left[\sum_{u,v} P(u,v) - \sum_{u,v=-f_{\text{lowpass}}}^{f_{\text{lowpass}}} P(u,v) \right],$$

where the first sum runs over the whole Fourier image and the second sum gives us the low-frequency part. The frequency f_{lowpass} has to be adjusted according to the PSF of the system; for our data with width = height = 128 px a border of 10 px yielded good noise estimates. K and K' denote the numbers of pixels to average over ($K = \text{width} \cdot \text{height}$, $K' = (\text{width} - \text{border}) \cdot (\text{height} - \text{border})$).

Subtracting the noise power from the signal equation 2.1 leads to

$$\mathcal{G}(u,v) = \frac{\mathcal{S}(u,v)}{\mathcal{S}(u,v) + \mathcal{V}(u,v)} = \frac{\mathcal{P}(u,v) - n}{\mathcal{P}(u,v)} \tag{2.2}$$

The resulting Wiener filter is shown in figure 2.2 in the Fourier and in the spatial domain.

If sigma is known for the PSF beforehand, a Gaussian filter with this value is a good alternative to the Wiener filter learned from the data. The accuracy measures described in chapter 3 showed very similar results for the Wiener filter and a Gaussian filter with appropriate width.

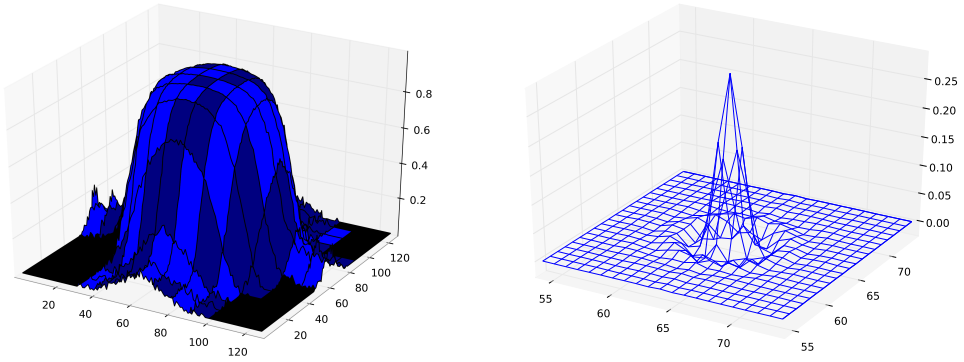

Figure 2.2: Wiener filter constructed from our training data set in Fourier domain (left) and transformed into spatial domain (right)

2.3 Background subtraction

The fluorescent images are acquired using a CCD-Camera. The CCD electronics can add a baseline to every pixel and in inhomogeneous structures we measure illuminations coming from out-of-focus cells. Before detecting the spot positions with a threshold we thus have to estimate the background level.

We decided to use a coarse Gaussian filter as a rough estimate of the background. Our current implementation uses a filter with $\sigma = 10$ but the exact value isn't crucial as our experiments have shown. This yields good results and is computationally easy, since all frames can still be considered independently from each other.

The procedure of subtracting a wide Gaussian from a narrow one is known in literature as *Difference of Gaussians* (Jähne, 2005, page 368) and corresponds to a bandpass filter. Only frequencies of a certain range are maintained and used for the spot-localization.

An example background image showing a structure with inhomogeneous background due to the cell shape is shown in figure 2.3.

2.4 Interpolation

In contrast to other dSTORM data processing algorithms our method does not need any fitting procedure but instead finds local maxima directly in a super-resolved interpolated image. The key ingredient is the filter described above that makes the pixel values measured on the pixel grid smooth and symmetric with respect to the center of the spot. After the filter stage the values can be approximated by a spline.

Following the presentation in Köthe (2007, pp. 68ff) we sum up the basic spline image interpolation methods:

Let the pixel-values of the filtered image be $f(x, y)$ where only pixel values at

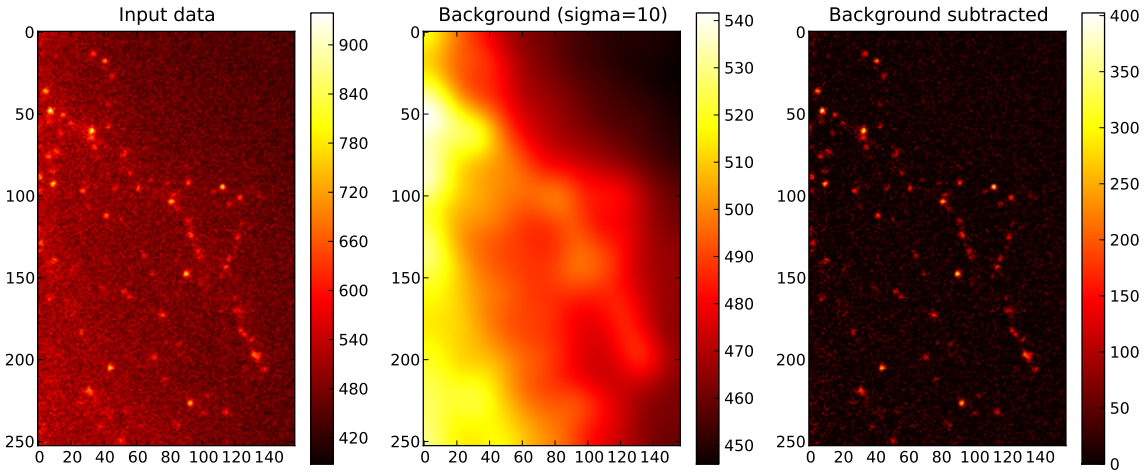

Figure 2.3: In addition to the CCD baseline the left image has an image gradient from the lower left to the upper right corner. A coarse Gaussian smoothing yields a background image displayed in the middle that is subtracted from the original. In the resulting image (right) negative values have been cropped to zero.

integer-positions of x and y are available. The interpolation task is to determine values $f(\tilde{x}, \tilde{y})$ for real-valued positions $(\tilde{x}, \tilde{y})$. The basic idea of spline interpolation is to fit piecewise polynomial splines to the measured pixel values. At integer positions $f(x, y)$ should be reproduced and subsequent splines should be continuous. This can be achieved using a weighted sum of neighboring pixels with one-dimensional interpolators $B(z)$:

$$f(\tilde{x}, \tilde{y}) = \sum_{i=-s}^{s} \sum_{j=-s}^{s} B(\Delta x + i) \cdot B(\Delta y + j) \cdot f(\lfloor \tilde{x} \rfloor + i, \lfloor \tilde{y} \rfloor + j)$$

with denoting $\Delta x = \tilde{x} - \lfloor \tilde{x} \rfloor$ and Δy accordingly. $B(z)$ could be an arbitrary (one-dimensional) interpolation function with support $[-s, s]$. The operation is separable: The interpolation can be applied independently in x- and y-direction.

BSplines One class of interpolators are BSplines of order k that are defined as a recursive convolution of the box function with itself:

$$B_0(z) = \begin{cases} 1, & -\frac{1}{2} \leq x < \frac{1}{2} \\ 0, & \text{else} \end{cases}$$

$$B_n(z) = B_{n-1}(z) * B_0(z) \tag{2.3}$$

By construction these functions are piecewise polynomial of the order k. The coefficients of the splines are given explicitly in table 2.1. The BSpline of order

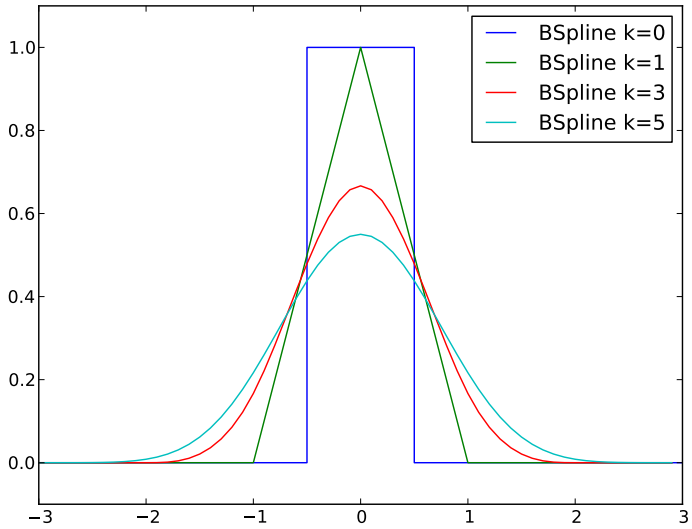

Figure 2.4: First few BSplines, see text for explanation and table 2.1 for the piecewise definitions.

zero corresponds to nearest-neighbor interpolation (nearest pixel values are copied). $B_1(x)$ gives an bi-linear interpolator. For higher order splines the values at integer pixel positions are not reproduced with these splines alone. Figure 2.4 shows that the splines do not fulfill $B(x) = 1$ anymore. They are smoothing the image at the same time as interpolating.

To make this operation an interpolation (reproducing the pixel values at integer positions), a recursive prefilter for sharpening the image is required. The prefilter has to be defined such that the smoothing with the BSpline interpolator is compensated. A first order infinite-impulse-response (IIR) filter is applied forward and backward before BSpline interpolation.

One-dimensional example values and their interpolation with BSplines of order 3 and 5 are plotted in figure 2.6. The support (and thus the computational cost) increases with the order of the spline. On the other hand the precision of the determined maximum is also improved for higher order splines. Figure 2.5 compares the transfer function of different splines with the ideal interpolator.

Cardinal Splines An exact alternative without prefiltering are cardinal splines that evaluate to 1 at $x = 0$ and to 0 at other integer coordinates. The Catmull-Rom spline for example is defined as:

$$\text{CatmullRom}(x) = \begin{cases} 1 - \frac{5}{2}|x|^2 + \frac{3}{2}|x|^3 & \text{if } |x| \leq 1 \\ 2 - 4|x| + \frac{5}{2}|x|^2 - \frac{1}{2}|x|^3 & \text{if } 1 < |x| \leq 2 \\ 0 & \text{else} \end{cases}$$

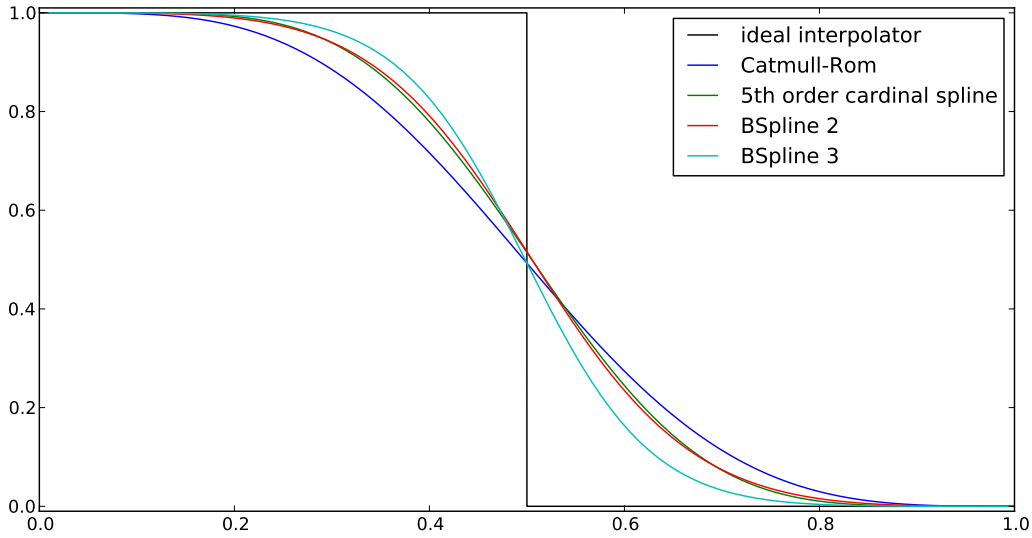

Figure 2.5: In Fourier domain the ideal interpolator copies the low-frequency-part and introduces no higher frequencies than were available in the input image. Our optimized 5th order cardinal spline is as good as a second order BSpline but higher order BSplines approximate the ideal interpolator much better.

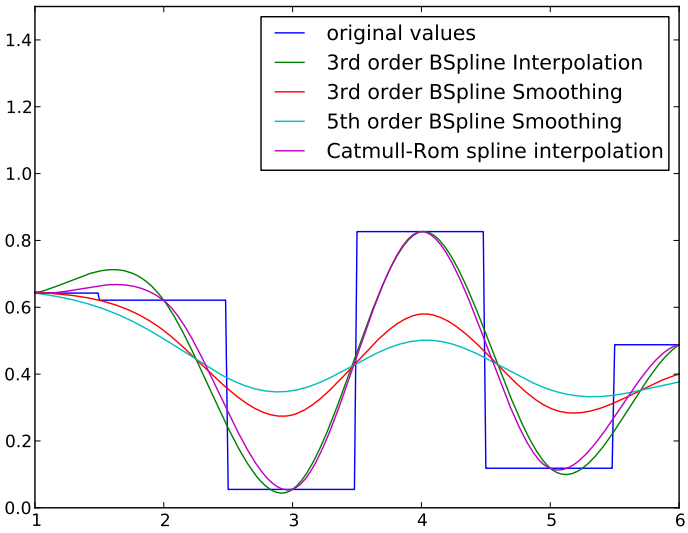

Figure 2.6: One-dimensional values interpolated with different interpolation splines.

$$B_0(x) = \begin{cases} 1 & \text{if } -\frac{1}{2} \leq x \leq \frac{1}{2} \\ 0 & \text{else} \end{cases} \tag{2.4}$$

$$B_1(x) = \begin{cases} 1 - |x| & \text{if } |x| \leq 1 \\ 0 & \text{else} \end{cases} \tag{2.5}$$

$$B_3(x) = \begin{cases} \frac{1}{2}|x|^3 - |x|^2 + \frac{2}{3} & \text{if } |x| \leq 1 \\ \frac{1}{6}\left(2 - |x|\right)^3 & \text{if } 1 < |x| \leq 2 \\ 0 & \text{else} \end{cases} \tag{2.6}$$

$$B_5(x) = \begin{cases} -\frac{1}{12}|x|^5 + \frac{1}{4}|x|^4 - \frac{1}{2}|x|^2 + \frac{11}{20} & \text{if } |x| \leq 1 \\ \frac{1}{24}|x|^5 - \frac{3}{8}|x|^4 + \frac{5}{4}|x|^3 - \frac{7}{4}|x|^2 + \frac{5}{8}|x| + \frac{17}{40} & \text{if } 1 < |x| \leq 2 \\ \frac{1}{120}\left(3 - |x|^5\right) & \text{if } 2 < |x| \leq 3 \\ 0 & \text{else} \end{cases} \tag{2.7}$$

Table 2.1: First few BSpline piecewise-polynomial splines

The transfer function of a cubic BSpline interpolator (after appropriate prefiltering) approximates the ideal interpolator much better than the Catmull-Rom spline. Both alternatives require the same computational effort (number of multiplications and additions). Prefiltering has to be done only once and thus is negligible, if interpolated values at many points of the image are calculated. But if IIR prefilters are not available, the Catmull-Rom spline can be used.

For that particular case we want to generalize the Catmull-Rom interpolation. Can a cardinal spline with support $[-3, 3]$ and 5th order polynomials give a better approximation to the ideal interpolator? In other words, we want to find coefficients for the piecewise-polynomial function:

$$S(x) = \begin{cases} a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0 & \text{if } |x| \leq 1 \\ b_5 \cdot x^5 + b_4 \cdot x^4 + b_3 \cdot x^3 + b_2 \cdot x^2 + b_1 \cdot x + b_0 & \text{if } 1 < |x| \leq 2 \\ c_5 \cdot x^5 + c_4 \cdot x^4 + c_3 \cdot x^3 + c_2 \cdot x^2 + c_1 \cdot x + c_0 & \text{if } 2 < |x| \leq 3 \\ 0 & \text{else} \end{cases}$$

To find these 18 coefficients we set up several conditions. The function values at integer positions are for a cardinal spline $S(0) = 1$ and $S(1) = S(2) = S(3) = 0$. We require $\mathcal{C}^2$ continuity in the spatial domain at the points $x = 0$, $x = \pm 1$, $x = \pm 2$. In analogy to the Catmull-Rom spline we additionally set the derivatives of the transfer function in Fourier domain to 0 at $\omega = 0$ and $\omega = 1$:

$$\lim_{\omega \to 0} \frac{d^2}{d\omega^2}\tilde{S} = 0, \quad \lim_{\omega \to 0} \frac{d^4}{d\omega^4}\tilde{S} = 0, \quad \lim_{\omega \to 1} \frac{d^1}{d\omega^1}\tilde{S} = 0, \quad \lim_{\omega \to 1} \frac{d^2}{d\omega^2}\tilde{S} = 0, \quad \lim_{\omega \to 1} \frac{d^3}{d\omega^3}\tilde{S} = 0$$

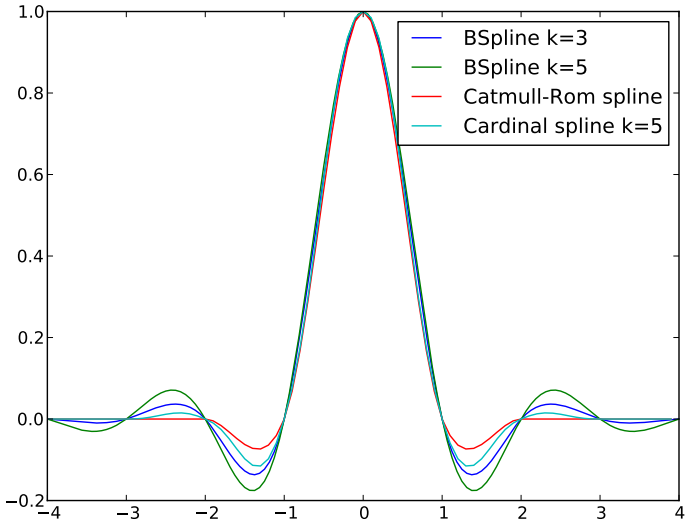

Figure 2.7: Impulse response of BSplines and Cardinal Splines in comparison.

With these 18 conditions we can solve for the coefficients and finally arrive at a new cardinal spline (cf. figure 2.7):

$$S(x) = \begin{cases} \frac{1}{12}\left(-25x^5 + 63x^4 - 35x^3 - 15x^2 + 12\right) & \text{if } |x| \leq 1 \\ \frac{1}{24}\left(25x^5 - 189x^4 + 545x^3 - 735x^2 + 450x - 96\right) & \text{if } 1 < |x| \leq 2 \\ \frac{1}{24}\left(x-3\right)^3 \cdot \left(5x^2 - 18x + 16\right) & \text{if } 2 < |x| \leq 3 \\ 0 & \text{else} \end{cases}$$

The integral can be proven to sum up to $\int_{-3}^{3} S(x)dx = 1$ which means the interpolator introduces no overall gain. A comparison in Fourier domain shows that the transfer function is indeed better than the Catmull-Rom one but still not as good as a 3rd order BSpline (cf. figure 2.5). The explicit transfer functions of the Catmull-Rom and our new cardinal spline are:

$$H_3(\omega) = \frac{e^{-4i\pi\omega}\left(-1 + e^{2i\pi\omega}\right)^3\left(-3 - 2i\pi\omega + e^{2i\pi\omega}(3 - 2i\pi\omega)\right)}{16\pi^4\omega^4}$$

$$H_5(\omega) = \frac{e^{-6i\pi\omega}\left(-1 + e^{2i\pi\omega}\right)^5\left(25 + 24i\pi\omega - 7\pi^2\omega^2 + e^{2i\pi\omega}\left(-25 + 24i\pi\omega + 7\pi^2\omega^2\right)\right)}{64\pi^6\omega^6}$$

Interpolation for dSTORM processing After trying out all the above options we found that a good trade-off for dSTORM processing are BSplines of the order 3. Catmull-Rom splines led to grid artifacts at low signal-to-noise ratios. BSplines with prefiltering showed much less such artifacts. Dropping the prefilter we now

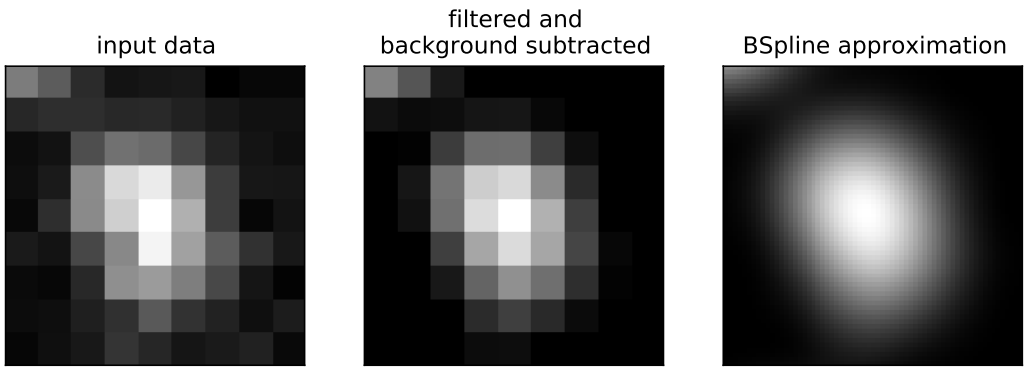

Figure 2.8: After BSpline-Approximation the exact spot positions can simply be determined as the positions of local maxima

use a BSpline *approximation* and get results that normally do no more show a grid structure in the background.

We sample the interpolated image at a multiple of the original rate and find local maxima on this refined pixel grid. An example spot that has been interpolated (smoothed) by a factor of 8 is shown in figure 2.8.

2.5 Maxima detection and post-processing

After interpolation the super-resolved position of each spot is approximated as the local maximum that is above a given threshold. All eight neighboring pixels must have smaller values for a maximum to be valid.

The global threshold is currently the only tuning parameter that has to be varied from data set to data set. With too small thresholds the number of false positives can get very large. With higher thresholds as necessary, low (but valid) signals are lost but a faster preview of the image is possible.

Local threshold relative to the background To minimize the number of spots found due to noise we discard maxima that are smaller than the background determined previously (see section 2.3). The baseline offset contained in the background image is subtracted before this comparison. Then only those spots are kept, whose central intensity after smoothing interpolation is larger than the background value. In other words, the input value has to be at least twice the background to be a spot candidate.

Figure 2.9 shows a typical histogram of detected intensities. The intensity corresponds to the value of the central pixel in the interpolated image. The effect of the local threshold with respect to the background is visualized in figure 2.10. High-intensity spots remain unchanged but in areas with low signal to noise ratios, false

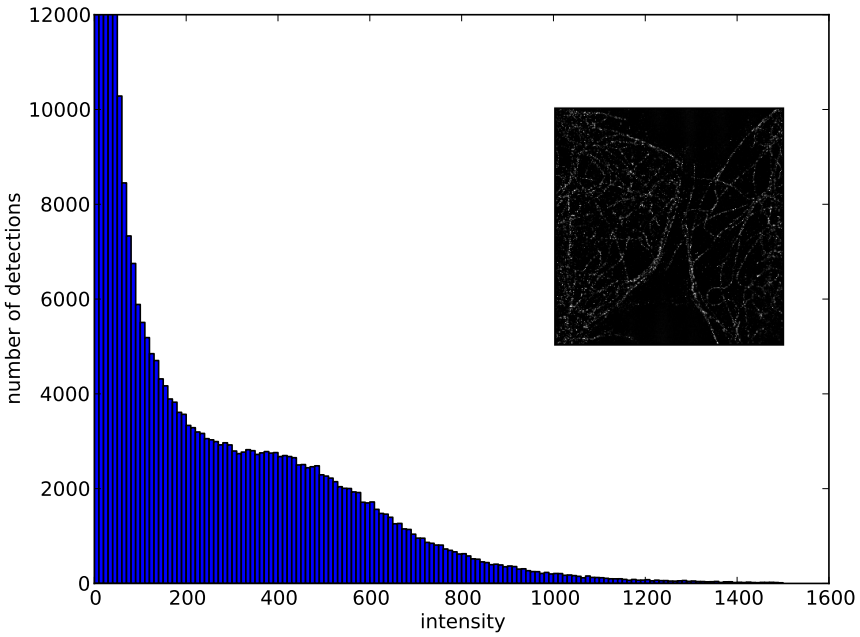

Figure 2.9: Histogram of intensities of the localized spots. The peak for very low intensities are maxima detected from background noise. An appropriate threshold should be selected to get rid of those false detections.

positive detections are efficiently suppressed.

Spot asymmetry Another filter-criterion to get rid of false positives is the spot asymmetry. Ideally a valid spot should be circle-shaped. The asymmetry can be determined in the spline-image by considering the eigenvalues of the hessian matrix:

$$H = \begin{pmatrix} \frac{d^2}{dx^2} & \frac{d^2}{dxdy} \\ \frac{d^2}{dydx} & \frac{d^2}{dy^2} \end{pmatrix}$$

The eigenvalues λ_1 and λ_2 are the curvature in the main directions of the ellipsis at the position of the local maximum. A threshold for too large fractions λ_1/λ_2 removes very asymmetric spots and can improve the quality of the resulting image. In the histogram shown in figure 2.11 this has a noise-suppression effect for low intensity-signals but some high-intensity signals are removed, too.

2.6 Qualitative comparison

Before we come to a quantitative analysis of accuracy, precision and recall we want to show two figures to show the performance of our method. We compared the single spot detections with those found by rapidSTORM (Wolter et al., 2010). Figure

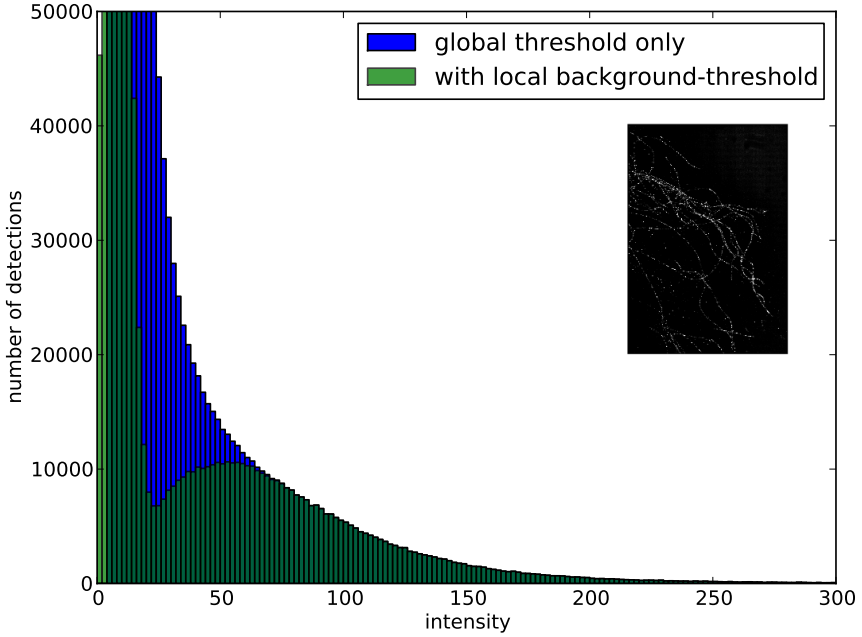

Figure 2.10: Histogram of intensities of the localized spots. The peak for very low intensities are maxima detected from background noise. A local threshold relative to the background removes most of the low-signal false positives.

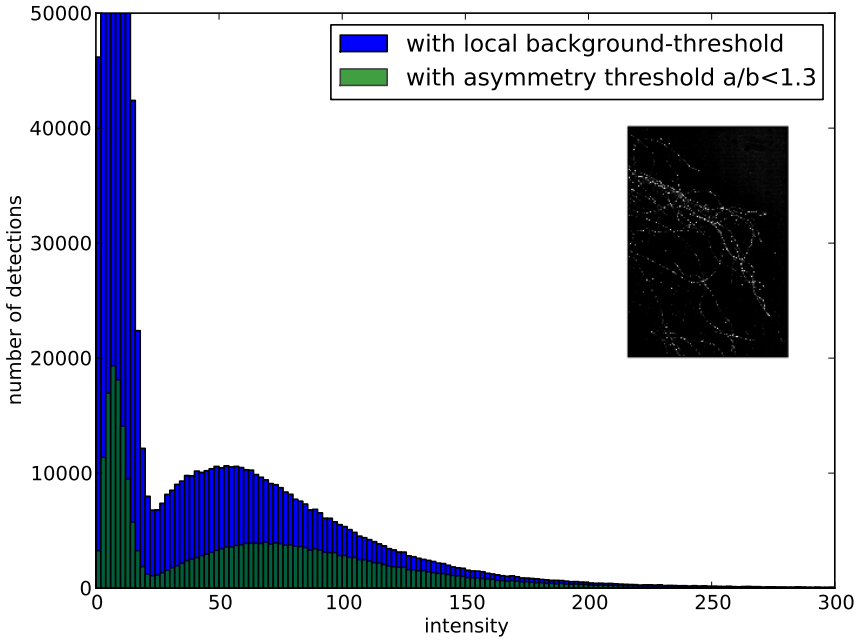

Figure 2.11: Aggressive spot filtering by asymmetry. The fraction of the eigenvalues of the Hessian matrix yield an asymmetry estimate.

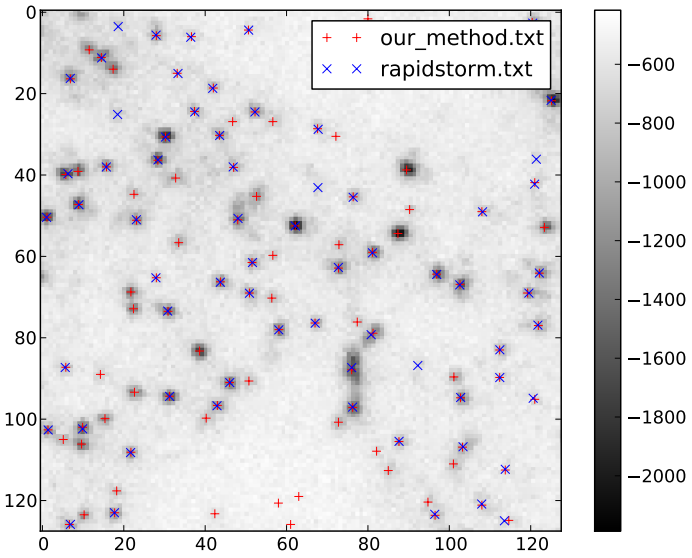

Figure 2.12: Spots localized by rapidSTORM and our maximum detection algorithm. The input data are plotted as gray values in the background. For rapidSTORM we set the threshold for spot asymmetry = 0.1 and an amplitude threshold of 1000.

2.12 shows the detected fluorophore positions found by both algorithms. The input data are plotted in gray in the background. Our visual impression is, that all the detections are valid spots. Both algorithms localize the fluorophores with high accuracy.

In regions, where several spots are near each other, an accurate detection is much more challenging than just a single spot detection. For a detailed comparison we cropped some regions from figure 2.12 and compared them in figure 2.13. It seems, that the simple detection of maxima is much better suited for regions with several molecules near each other, while a Gaussian fit is more accurate, if only single molecules are analyzed.

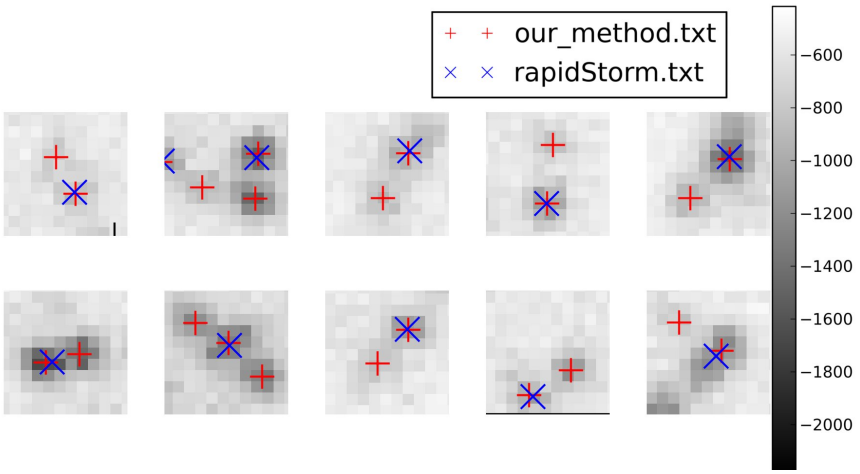

Figure 2.13: Selected regions cropped from the single frame of a dSTORM measurement shown in figure 2.12. The spots localized with our algorithm and with rapidSTORM are marked. The areas show missing detections of rapidSTORM in regions with high spot densities.

3 Accuracy of spot-detection

Besides the qualitative comparison and validation on real data we need a quantitative method to compare different parameter settings and to compare our method with other algorithms. Although localization accuracy for single spots is not necessarily the same as structural resolution, it is obviously useful to reduce the error in single molecule detection. The structural resolution can be determined for example in samples of actin filaments measuring the distance of still separable objects or the width of narrow filaments.

Additionally recall and precision are useful measures although hard to retrieve on real data. Several simulation schemes have been proposed to test the performance of algorithms for data processing for localization microscopy. (Andersson, 2007; Grüll et al., 2011; Wolter et al., 2011).

After comparison on simulated data this chapter discusses bead localization variance as another performance measurement.

3.1 Measuring accuracy using simulated data

Since there is no possibility to get an accurate "ground truth" position for each spot in the measured videos, we use a simulation according to the one described by Grüll et al. (2011). The position of a single spot is chosen randomly (one spot per frame) and a Gaussian PSF is plotted at that position. After adding a constant background the image is disturbed with Poisson-Noise accounting for the photon-counting nature of image acquisition. A stack of 1000 simulated images are then processed with our software.

The detected position $p_i \in \mathbb{R}^2$ is compared to the simulation input position s_i and the standard deviation over all 1000 frames is calculated:

$$\text{stddev} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|p_i - s_i\|^2}$$

This method gives the possibility to compare the accuracy for different signal-to-noise ratios. One example image and our root-mean-square errors are plotted in figures 3.1 and 3.2.

For low intensities our proposed method is as accurate as the estimator from Grüll et al. (2011). For high signal-to-noise ratios their method is marginally better. QuickPALM (Henriques et al., 2010) was run with its default parameters and produced significantly worse localizations.

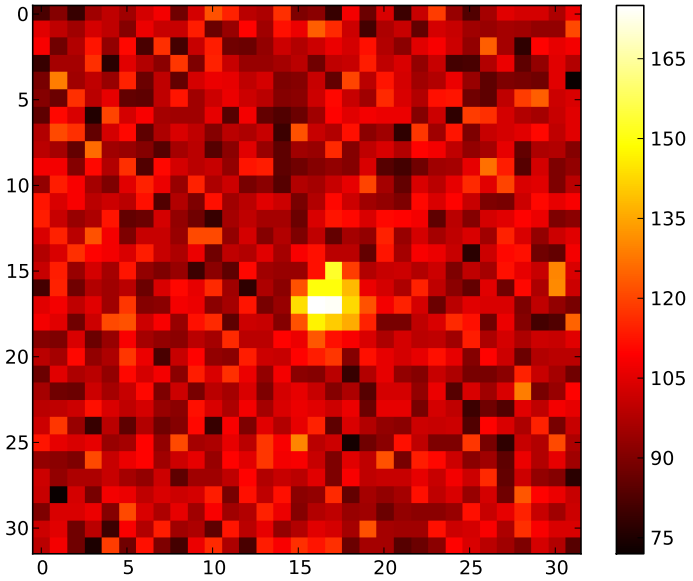

Figure 3.1: Simulated image showing one Gaussian PSF corrupted by Poisson noise. We choose a background level of 100 and $\sigma = 1.4$. The spot shown has a total intensity of $Q = 800$.

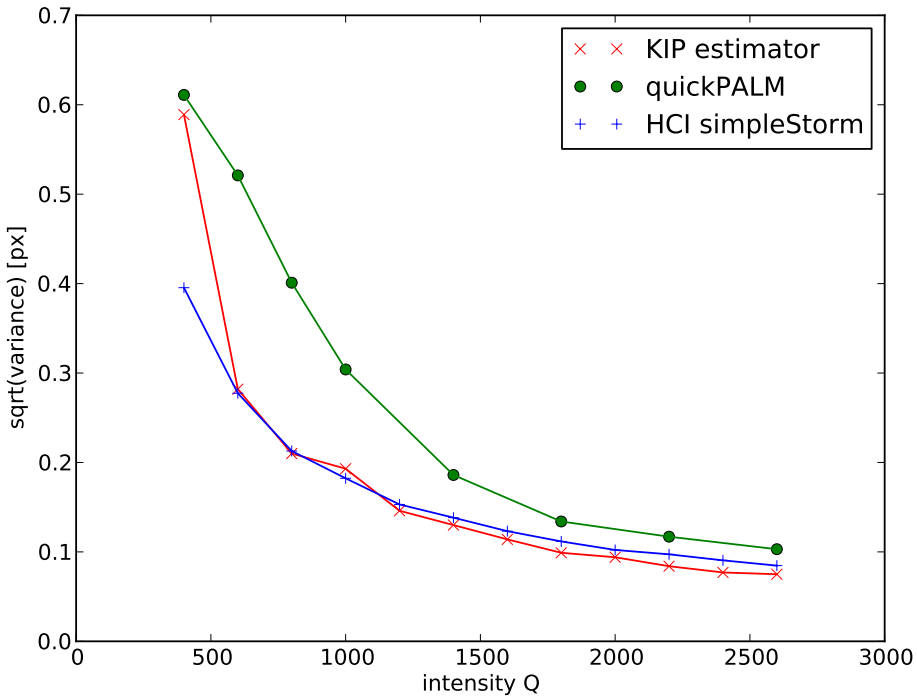

Figure 3.2: Standard deviations of the localized spots from the true positions.

Wolter et al. (2011) have recently proposed an advanced variant of this simulation procedure: Several spots are plotted simultaneously in one frame to determine localization accuracy and recall at the same time. From that results the maximum spot density that the algorithm can handle reliably can be estimated. Good performance on images with very high spot-densities is a challenge. Biological experiments highly benefit from a good multispot performance because the number of images could be decreased and a much better time resolution is possible. For future work it would be a good starting point to test this new simulation with our algorithm.

Using our simulation, the dependency of the estimated maximum intensity I from the total spot intensity $Q = \sum_{i \in \text{spot}} q_i$ can be checked. The spot width σ is fixed. Assuming a Gaussian PSF, the total intensity is:

$$Q = \sum_{i \in \text{spot}} q_i = 2\pi \cdot \sigma^2 \cdot I$$

After the smoothing interpolation and the Wiener filter, the values actually reported by our algorithm are below that theoretical value. The linear relationship between I and Q is verified on the simulated data (see figure 3.3). Here a constant factor of $K = 1.4$ has to be used to compensate for the smoothing filters:

$$\hat{Q} = K \cdot 2\pi \cdot \sigma^2 \cdot \hat{I}$$

3.2 Tetraspeck beads for accuracy evaluation

Another quantitative measurement for accuracy is the comparison of bead localization (non-blinking spots). Tetraspecks are usually used for calibration and multicolor applications (cf. section 4). Those beads are spheres with a diameter of 100 nm that are fixed so that they are at exactly the same position throughout a measured film. Changes in the detected position only result from noise and localization inaccuracy. In those bead measurements we still don't know the actual position of the beads but we can get a good estimate for the true position by averaging all the single detections if the density of beads in the sample is low enough (we used about 0.05 beads/µm^2). The localization performance for bead j is then given by the variance over the determined bead positions:

$$\text{Var}(p^{(j)}) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2 + (y_i - \bar{y})^2$$

A bead sample was prepared by Benjamin Flottmann using invitrogen tetraspeck beads. In close collaboration the imaging parameters were established. We imaged the sample at four different laser excitation energies to vary the measured photon counts and signal to noise ratio. Figure 3.4 shows the variance of several beads

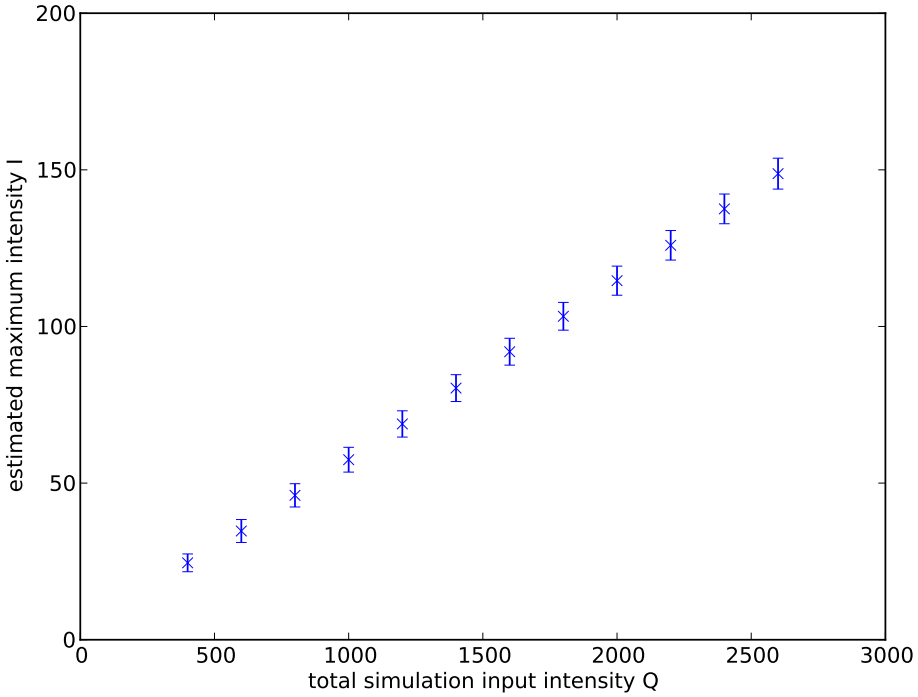

Figure 3.3: Intensity I estimated by our algorithm against the simulation input intensity Q. 1000 simulated images were analyzed for each value of Q.

against the signal intensities. All intensity data were concatenated into one plot. Our intensities from maximum detection were scaled to be comparable with the total spot intensities from the rapidSTORM software.

For the calculation of the average bead center position only spot detections have been selected, that are within a region of 2 px radius and are present in 95 % of all frames.

The variance is few nanometers lower with rapidSTORM than using our method. This is clearly visible for high intensities but the trend continues for low signal intensities as well.

Beads are constantly fluorescent and should be detected in every frame. We counted the number of detections within the region of every single bead. Figure 3.5 shows this number against the intensity. Using our algorithm the recall is constantly above 99 % as soon as the signals are above a minimum intensity. The number of detected spots varies heavily when using rapidSTORM. Especially high intensity signals seem to be discarded by rapidSTORM. A question to the author of rapidSTORM lead to no obvious explanation of the missing detections (until now). Perhaps we set some of their parameters inappropriate.

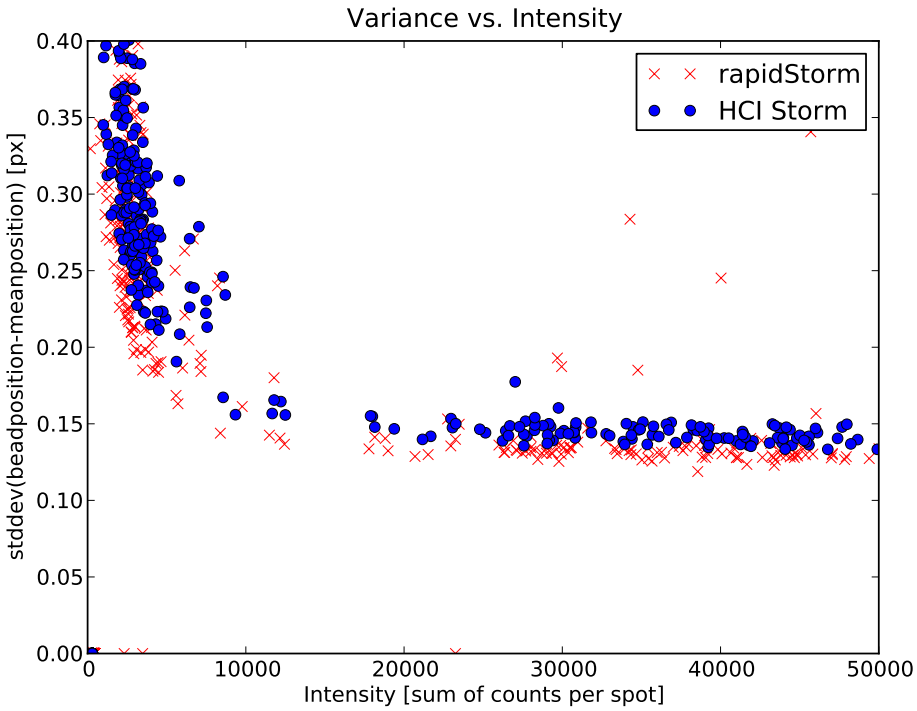

Figure 3.4: The localization accuracy depends on the signal intensities. The variance of detected spots for bead samples was measured at different laser intensities and the data was processed with rapidSTORM(Wolter et al., 2010) and our method.

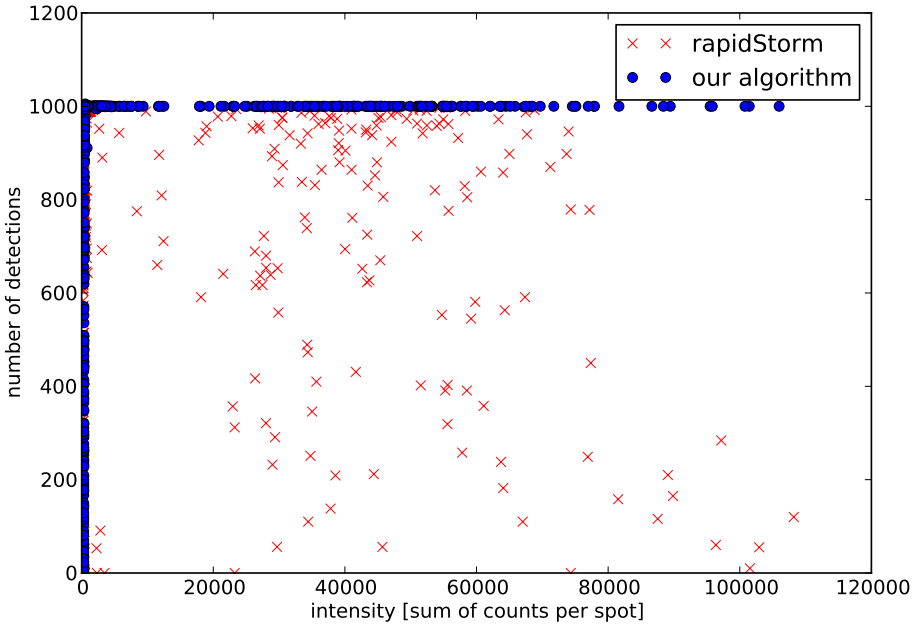

Figure 3.5: Recall of bead detections. Bead samples were imaged at different laser intensities and the data was processed with rapidSTORM (Wolter et al., 2010) and our method. The number of detections out of 1000 frames was counted within a radius of 1.5 px around bead centers.

4 Multicolor registration

In a biological sample different structures can be stained with distinct fluorescent markers. Two or more color channels are then acquired subsequently using different activation lasers and appropriate optical bandpass filters. But as Brauers notes, the optical path of a multispectral camera is distorted by inserting different bandpass filters: "However, since the optical filters exhibit different thicknesses, refraction indices and may not be aligned in a perfectly coplanar manner, geometric distortions occur in each spectral channel" (Brauers et al., 2008). Brauers presents a detailed analysis of the optical path of a multispectral camera. The insertion of a filter wheel between aperture and camera "can cause noticeable distortions in the image" (Brauers et al., 2008). He essentially finds that the misalignment can be well compensated with an affine linear transform.

For our measurements in super-resolution microscopy additional distortions are added by chromatic aberration of lenses inside the optical path. The lenses are corrected for aberrations in the usual optical length scale but not for super-resolution.

Churchman et al. (2005) reported a transformation procedure using a split-screen. They imaged two channels and used a bead calibration measurement to align them within 3.3 nm.

We have no possibility to do a calibration measurement since the rotation of a single filter wheel (forward and back again) already leads to a translation of measured positions, as an example measurement with Tetraspeck beads pointed out.

Thus we add few Tetraspeck beads directly to the sample and determine their positions y_i throughout the data set with variance σ_i. In figure 4.1 beads are highlighted with circles. The translation between different color channels is clearly visible.

We match corresponding beads in the different channels and fit an affine linear transformation to optimally align the channels in a least-squares sense.

4.1 Theory of linear least squares

Currently the setup of our collaboration partners in cell biology can only handle two different color channels (lack of good fluorescent dSTORM markers) but the alignment can generally work on an arbitrary number of color channels. The following theory section is restricted to registration of two color channels. Additional channels can be registered by aligning more different layers to one common background. We therefore choose one channel as background layer that is not changed and transform each other color channel with respect to this fixed image. Note, that this method

does not totally compensate for chromatic aberrations but only matches one channel to all the others. We don't use a calibration pattern, thus distortions present in the reference image remain in all the other channels. Especially non-linear lens distortion effects are not corrected by our procedure. (cf. Brauers and Aach, 2011)

We denote the two-dimensional positions of the background beads with $x_i = (x_{i,1}, x_{i,2})^T$ and the associated foreground beads with $y_i = (y_{i,1}, y_{i,2})^T$.

We assume an affine linear transformation allowing translation as well as rotation and scaling of the image:

$$\hat{y}_i = \beta_0 + \beta_1 \cdot y_{i,1} + \beta_2 \cdot y_{i,2}$$

In matrix notation using homogeneous coordinates this can be written as

$$\hat{y}_i = \beta \cdot y_i, \quad \beta \in \mathbb{R}^{3\times 3}.$$

We estimate β by finding the solution that minimizes the residual sum of squares

$$\mathrm{rss}(\beta) = \sum_i \|x_i - \beta y_i\|^2$$

$$\hat{\beta} = \arg\min_\beta \mathrm{rss}(\beta).$$

The root-mean-square error estimates how far the transformed points are from their "true" position (bias-corrected by subtracting the degrees of freedom of the fit):

$$\mathrm{rms}(\beta, X, Y) = \sqrt{\frac{1}{N - \mathrm{dof}} \sum_{i=1}^{N} \|x_i - \beta y_i\|^2} \tag{4.1}$$

4.2 Graphical User-Interface

For testing the method described above we implemented a prototype in python using PyQt and numpy. The number and positions of the beads used can be selected to evaluate the influence on the exact fit and rms error.

Spot localization results are read from text-files produced by the storm processing software. Every line contains ASCII-encoded x- and y-coordinates, the frame number and the estimated intensity separated by whitespace. The `csv` python module is used to decode the data into a numpy-array. The coordinates are rendered into an image using numpy-functions. We add different channels using distinct colors in rgb color space.

Bead positions are detected automatically by scanning through the localization coordinates and selecting positions that are found in every frame. Localizations belonging to the same bead must be within a local radius of few pixels (we consider a

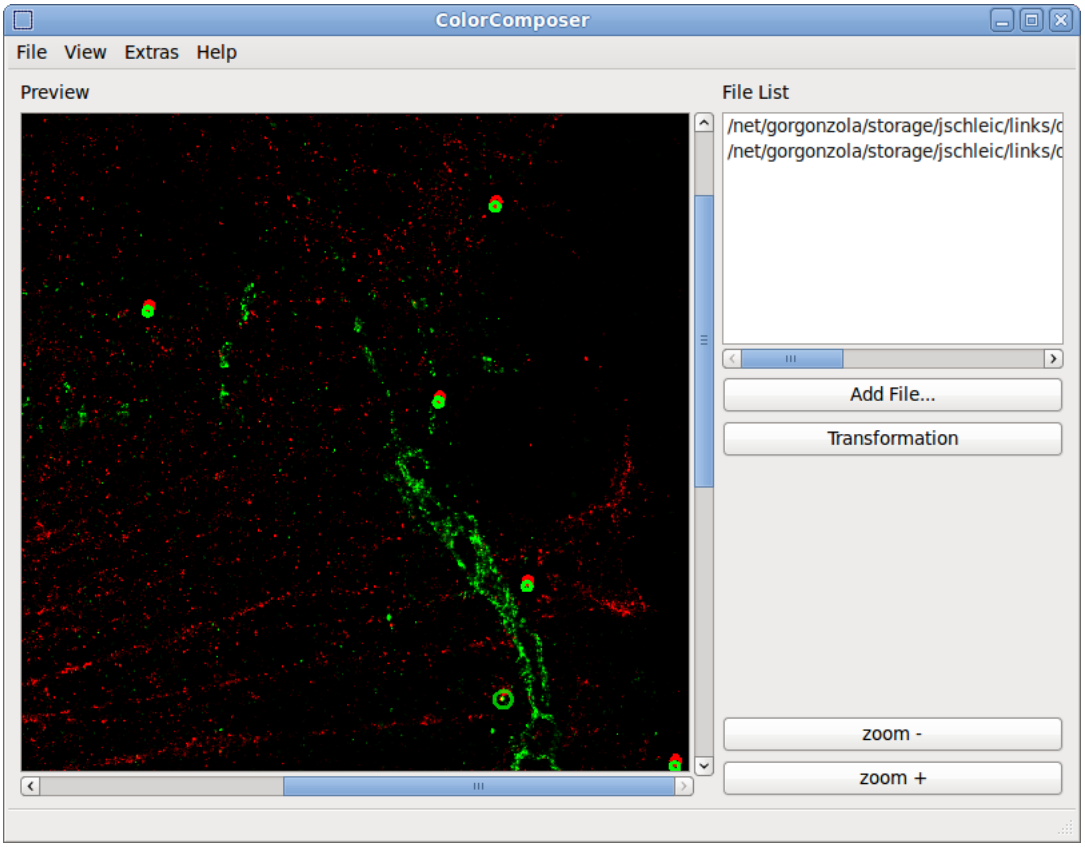

Figure 4.1: Positions of beads added to the sample to align different channels after the measurements are marked with circles. The beads are much brighter than normal signals and should be detected in every frame. The user interface allows selection of beads to calculate an affine linear transformation that compensates for the geometric distortions.

figure	size	# beads	frames	sqrt(Variance) of bead positions	RMS error of fit
4.2	$256 \times 256\,\mathrm{px}^2$	11	7500	0.13 px to 0.27 px	0.0974 px
4.3	$302 \times 236\,\mathrm{px}^2$	23	7500	$0.15 - 0.30$ px	0.2618 px
beads	$512 \times 512\,\mathrm{px}^2$	156	1000	$\approx 0.15\,\mathrm{px}$	$\approx 0.10\,\mathrm{px}$, see figure 4.4

Table 4.1: RMS error for some dual-color measurements

radius of 2 px), since their position differs only due to localization accuracy and noise in the acquisition process. The exact position of the bead is found by averaging over the detections in all frames and the variance indicates the accuracy of the position.

A screenshot with beads found by this procedure is shown in figure 4.1. The alignment was tested with coordinate files produced by our localization algorithm and using rapidSTORM (Wolter et al., 2010). But the rapidSTORM algorithm missed some beads in a significant number of frames. Images processed with our data processing algorithm resulted in much more reliable bead detections afterwards.

The user can select the beads by double-clicking the circles; after calculating the transformation the aligned images are displayed. The transformed raw localization coordinates and the result image can be saved to disk.

Overall the user-interface is a proof-of-concept to evaluate the bead-alignment. It could be easily extended to fully automatic alignment of multi-color measurements. But therefore the spot localization algorithm has to detect the beads reliably.

4.3 Quality measure on real world examples

The dual-color registration has been applied to different samples; as a quantitative quality control we used the root-mean-square error of the fit. Table 4.1 gives some representative numbers. The images in figures 4.2 and 4.3 show corresponding cell structures with endoplasmic reticulum labeled in red and actin fibers labeled with green fluorophores.

Alignment of Tetraspeck Beads To check the quality of the alignment procedure we took an image of a sample containing only Tetraspeck beads and no cell structure. We used a high laser excitation to get a good signal to noise ratio and thus a localization accuracy of about 0.15 px. For the fit only a small fraction of the available beads was used (chosen randomly) and the RMS error was cross-validated with the other points not used for the linear fit. Figure 4.4 shows the results of this experiment.

The fit error was calculated using equation 4.1 with dof = 3 to compensate for a bias when only few points are used for the fit. For three points no fit error can be

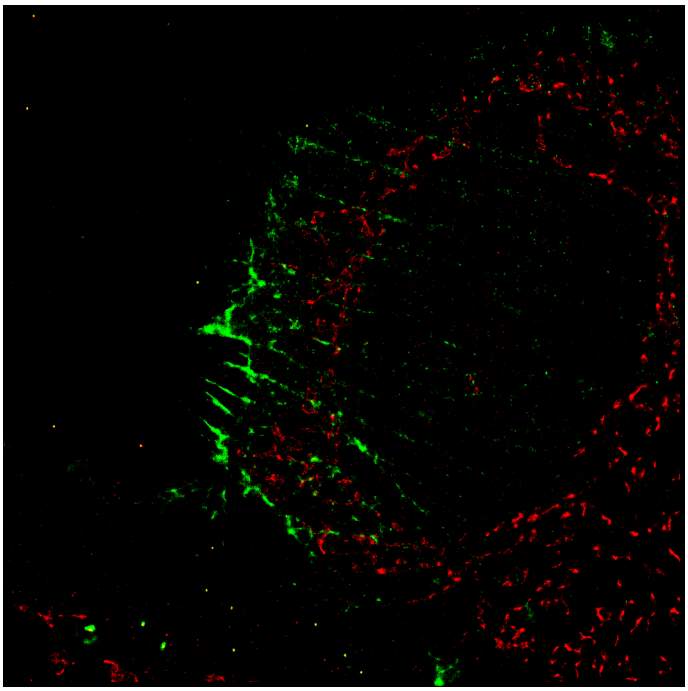

Figure 4.2: ER647 (red) and actinmEOS2 (green) after dual-color alignment of corresponding beads

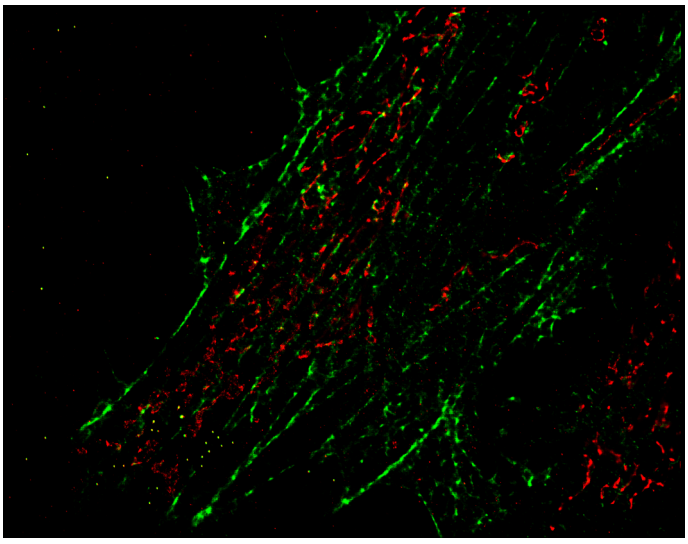

Figure 4.3: ER647 (red) and mEOS2 (green) after dual-color alignment of corresponding beads

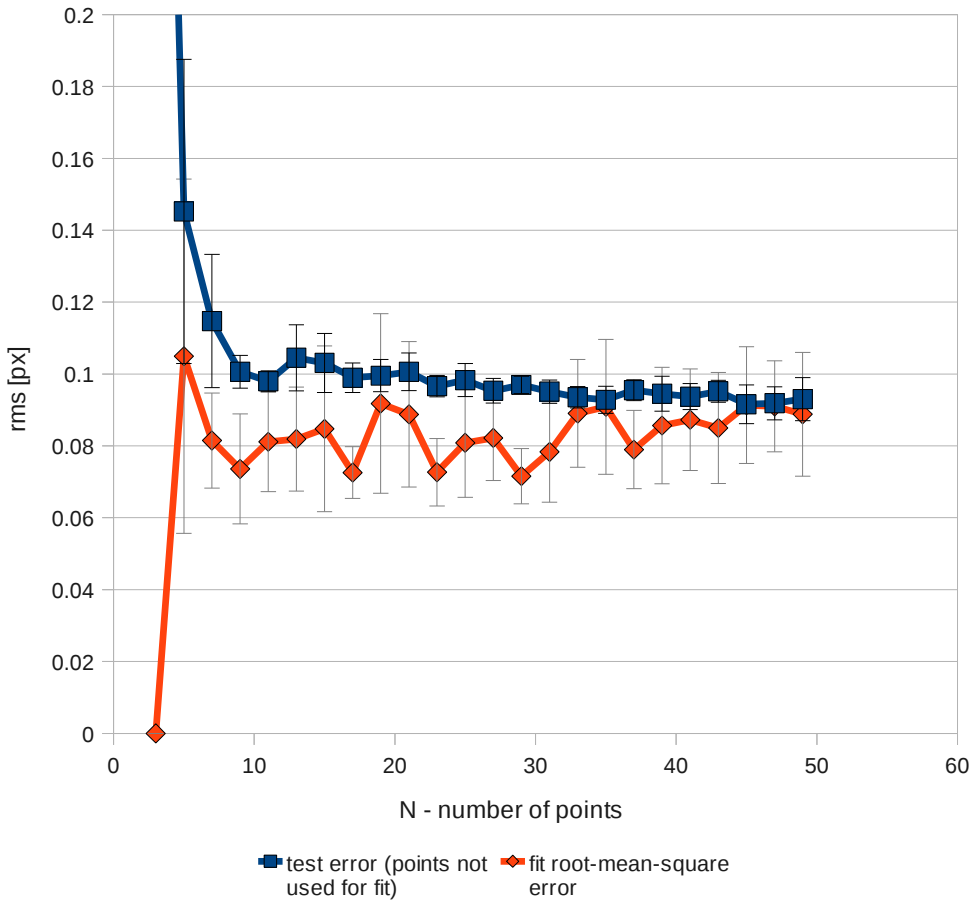

Figure 4.4: Registration of high intensity Tetraspeck beads: the RMS error does not drop significantly further, if more than 15 beads are used for the linear fit

estimated, so we set it to zero. The error bars indicate the spread between different randomly chosen subsets containing the same number of corresponding points.

Using more than 9 points for the fit, the test error decreases only slowly; if one has so many landmarks, more flexible registration algorithms could be applied (e.g. Thin-Plate splines: Bookstein, 1989).

Part II

Accelerated computing

5 CPU

The localization procedure described in the previous chapter has to determine the spots on many successive images. This computationally intensive operation has recently been reported to be solvable in realtime for small image sizes (Wolter et al., 2010; Grüll et al., 2011). Henriques et al. (2010) implemented an ImageJ plugin for PALM data processing.

In the following chapters we will summarize and compare different methods of accelerating our algorithm described above. Besides algorithmic improvements and simplifications (that potentially decrease the image quality while improving the runtime) we look at dedicated hardware. A Field Programmable Gate Array (FPGA) consist of programmable logic that can operate highly parallel and lets us implement the algorithm using the principle of pipelining.

Graphics processing units (GPU) are nowadays optimized for general-purpose computing operating on floating point numbers and are able to process a large number of threads in parallel. We won't give details about this architecture but instead refer to Quan et al. (2010) for a STORM implementation on a GPU with a speedup factor of eight.

5.1 C++ software implementation

After testing the algorithm with a python prototype we implemented the procedure in C++ utilizing the VIGRA image processing library. VIGRA stands for "Vision with Generic Algorithms" and implements templated versions of common image processing algorithms as well as library functions for image import / export (Köthe, 2011).

We compared and evaluated the correctness of our program and the accuracy using different options with the methods described in chapter 3. The raw data was measured by our collaborators Mike Heilemann and Benjamin Flottman using an Andor iXon DU-897 camera. The acquired files are read from a sif-file format as time series using a self-written import routine. Hdf5-input is possible using the vigra-builtin wrapper for the hdf5-libraries.

The data-processing outlined above can be performed with standard vigra function calls and our initial implementation is given in listing 5.1.

The function gets the input data and the parameters as function arguments and uses Vigra-FFTW-wrappers to perform the filtering in Fourier domain. After interpolation the local maxima in the larger image are detected. We wrote an Accessor that is passed to `localMaxima()` to append the coordinates to a `vector`

```
/**
 Localize maxima of the spots and return a list with coordinates
*/
template <class T>
void WienerStorm(MultiArrayView<3, T>& im, BasicImage<T>& filter,
       std::vector<std::set<Coord<T> > >& maxima_coords,
       T threshold=800, int factor=8) {

  unsigned int w = im.size(0);
  unsigned int h = im.size(1);
  unsigned int stacksize = im.size(2);
  unsigned int w_xxl = factor*(w-1)+1;
  unsigned int h_xxl = factor*(h-1)+1;

  // filter must have the size of input
  BasicImage<T> filtered(w,h);
  BasicImage<T> im_xxl(w_xxl, h_xxl);

  for(unsigned int i = 0; i < stacksize; i++) { //over all images
    // access current image as BasicImage
    MultiArrayView <2, T> array = im.bindOuter(i);
    BasicImageView<T> input = makeBasicImageView(array);

    // fft, multiply Wiener filter in frequency domain,
    // inverse fft, take real part
    vigra::applyFourierFilter(srcImageRange(input),
        srcImage(filter),
        destImage(filtered));
    //upscale filtered image with spline interpolation
    vigra::resizeImageSplineInterpolation(srcImageRange(filtered),
        destImageRange(im_xxl));

    //find local maxima that are above a given threshold
    VectorPushAccessor<Coord<T>,
        typename BasicImage<T>::const_traverser>
        maxima_acc(maxima_coords[i], im_xxl.upperLeft());
    vigra::localMaxima(srcImageRange(im_xxl),
        destImage(im_xxl, maxima_acc),
        vigra::LocalMinmaxOptions().threshold(threshold));
  }
}
```

Listing 5.1: Simplest C++ version of our data processing for localization microscopy

instead of marking the corresponding pixels in an image (the default behavior of `localMaxima()`). Afterwards all the returned sub-pixel coordinates are added (weighted with their intensity) into the result image for displaying and stored to a text file for further analysis.

In the following sections we describe some improvements that led to realtime applicability of the C++ version of our algorithm for small and medium image sizes (up to $256 \times 256\,\mathrm{px} \mathrel{\widehat{=}} 25 \times 25\,\mu\mathrm{m}$).

For implementation details please refer to the Code available from the author's webpage[1].

5.2 Algorithmic improvements

Small algorithmic improvements gained large speed-up factors of our C++ version. The most important ones are mentioned here; a complete comparison of runtime measurements including FPGA runtimes is given later in section 7.

Regions of interest The localization of maxima does not require an interpolation of the whole image. It is sufficient to interpolate small regions inside the image where spots are actually located. Depending on the density this gives a speedup factor of about five to ten. As center positions of the ROIs we choose local maxima in the filtered image with intensities larger than the global threshold.

If just single spots are present in the data set, a small ROI of size $5 \times 5\,\mathrm{px}$ is sufficient. If multiple spots are very close to each other, some detections can get lost. In rare cases only the brightest spot is found as ROI center and another spot is outside the ROI window. A typical case is shown in figure 5.1. For measurements with high spot densities (multiple spots inside the ROI) the size of the interpolated region can be increased by a commandline option to find those spots as well. For our test data set the number of additional spots detected by increasing the ROI window size is below 1‰ of the total number. Given the large speedup factor we can even accept a tiny fraction of missed detections in favor of much lower runtime.

Thresholding and Background noise Special care has to be taken if the user sets a very low threshold. Noise should not be counted as spots but actual low signals should be detected. To limit the total number of detections we introduced an additional local threshold relative to the background level that is described in section 2.5.

Memory Usage The first version outlined in listing 5.1 reads the whole data set into main memory before the processing starts. Thus only data sets notably smaller

[1] Webpage: http://hci.iwr.uni-heidelberg.de/Staff/jschleic/

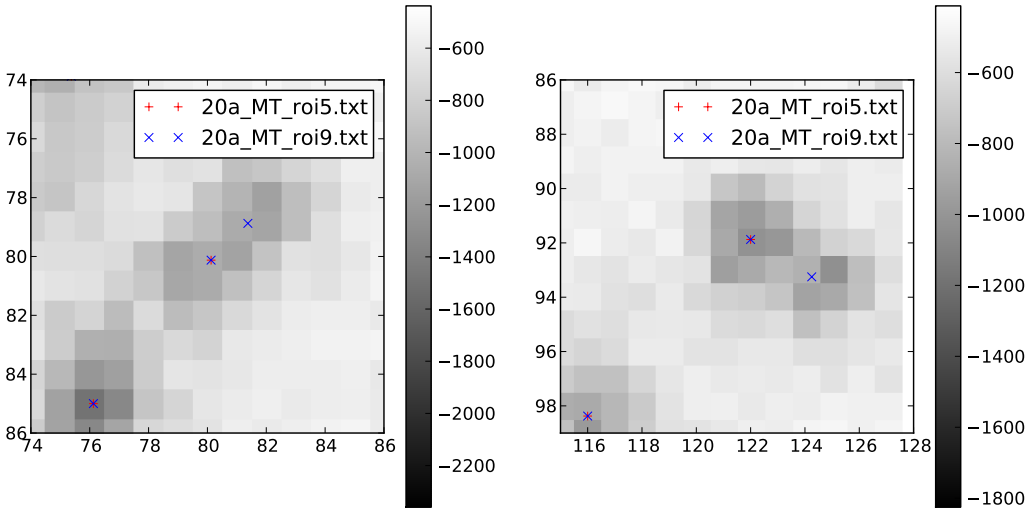

Figure 5.1: Special case, where increasing the window size for regions of interest (ROI) can find additional maxima. Usually one ROI is interpolated per spot candidate. But in high-density multi-spot regions, a larger ROI window size can find some additional spots that are no ROI candidates in the first place.

than the physical size of main memory could be processed. The current version circumvents this limitation and loads only the image frames that are currently processed.

5.3 Parallelization (OpenMP)

Consecutive frames have no data dependencies in our algorithm so that the processing can easily be parallelized. The speedup that can be achieved grows nearly linear with the number of cores of the CPU used while no precision is lost.

Technically the `for`-loop can be processed in parallel by adding `#pragma`s and linking against OpenMP. Then the number of threads is automatically adjusted to the number of cores accessible at runtime. Sequential code sections limit the maximum speedup according to Amdahl's law. Thus we want to avoid critical sections and process all the functions in parallel on different images. The used vigra-function calls are thread safe except `applyFourierFilter` that wraps fftw library calls.

So we rewrote the fftw-wrappers and create one common `fftw_plan` before entering the parallel scope and re-use that plan with the fftw "new-array execute functions". The execute functions of fftw are indeed thread-safe so that the only serial operations are initialization tasks and plotting the found coordinates into one result image. The latter task could be parallelized as well but it does take only a

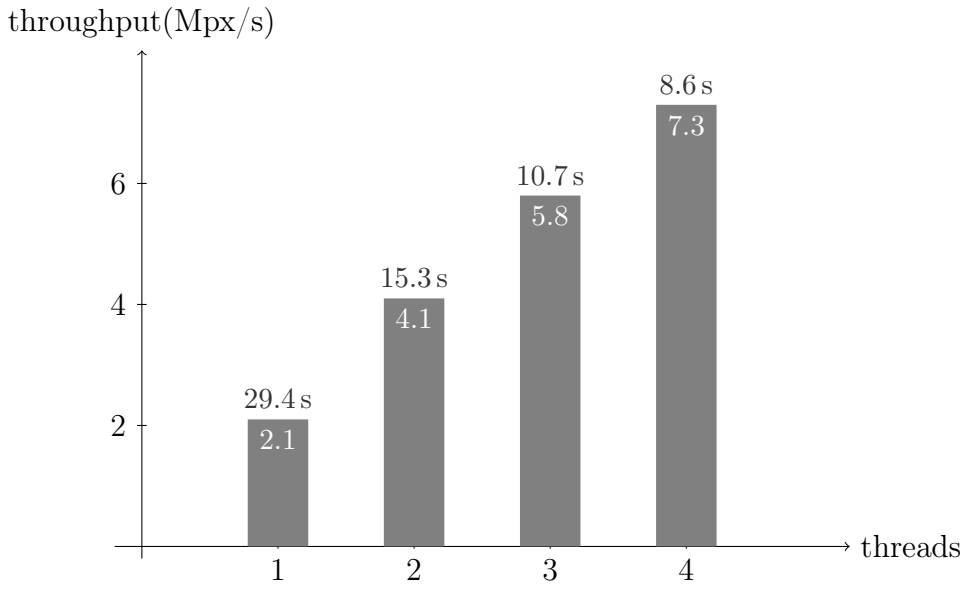

Figure 5.2: Timing of the parallelized command line version utilizing OpenMP. Measured are the runtime and equivalent data throughput in Mega-pixels per second for a 4000 frame measurement with 180 496 spots detected.

small percentage of the total runtime.

We measured runtimes for the data processing part of the algorithm on an Intel(R) Core(TM) i7 CPU 930 with different numbers of threads. The measured values are shown in diagram 5.2.

5.4 User interface

To make the software applicable in daily use we wrote a Qt-based graphical user-interface that asks for the relevant parameters and shows a preview of the assembled result image. A filter has to be built only once after the microscope's setup changed and can then be used for all subsequent measurements. The application uses all available cores by utilizing QtConcurrent API function calls. This gives similar speedups to the OpenMP parallelized commandline version. However, the usability for users not familiar to a command line is heavily increased. The GUI application has an intuitive interface, and asks only for the relevant parameters. The parameter dialog is shown in figure 5.3.

The display of intermediate results takes away some processing power, but gives an immediate feedback of the progress to the operator. If unsuitable parameters have been chosen, the calculation can be aborted in an early stage.

A screenshot of the application rendering the result image at different time steps

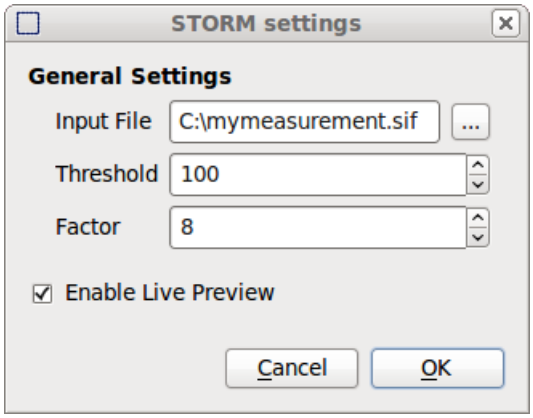

Figure 5.3: *Threshold* is the only parameter that has to be tweaked by the operator. *Factor* chooses the size of the interpolated image and thus the accuracy of sub-pixel detections.

is shown in figure 5.4. The preview is updated once a second, if a significant number of detected spots is available.

5.5 Limits in data throughput

The current approach does measure a complete image stack using the capturing devices and software from the camera manufacturer. Afterwards the data is processed with an analysis software for localization microscopy before the super-resolution result is available. Besides the delay that the operator of the microscope has to wait for the result this procedure is not extensible to very high frame rates because storage media are usually slow. With sufficiently high activation energy some fluorescent markers are able to switch within several milliseconds so that the sample can be imaged with 500 Hz as has been reported recently (Jones et al., 2011). Such frame rates improve the temporal resolution, so that dynamic processes can be imaged.

But retaining the field of view using resolutions of about $256 \times 256\,\mathrm{px}^2$ that produces high data rates of 64 MPix/s that correspond to 256 MB/s. Only fast RAID-systems or solid state disks are capable of storing such data-rates to disk in realtime. Thus we propose to consider realtime-data processing within the framegrabber and directly save the resulting coordinates that take only a fraction of the space of the input data. An FPGA framegrabber is capable of handling such capturing data rates but the algorithm has to be translated onto that hardware and has to be guaranteed to be fast enough to process the data in realtime. In the next chapter we point out some differences between CPUs and FPGAs. We show an FPGA implementation that is as fast as the CPU version and processes the data on the fly.

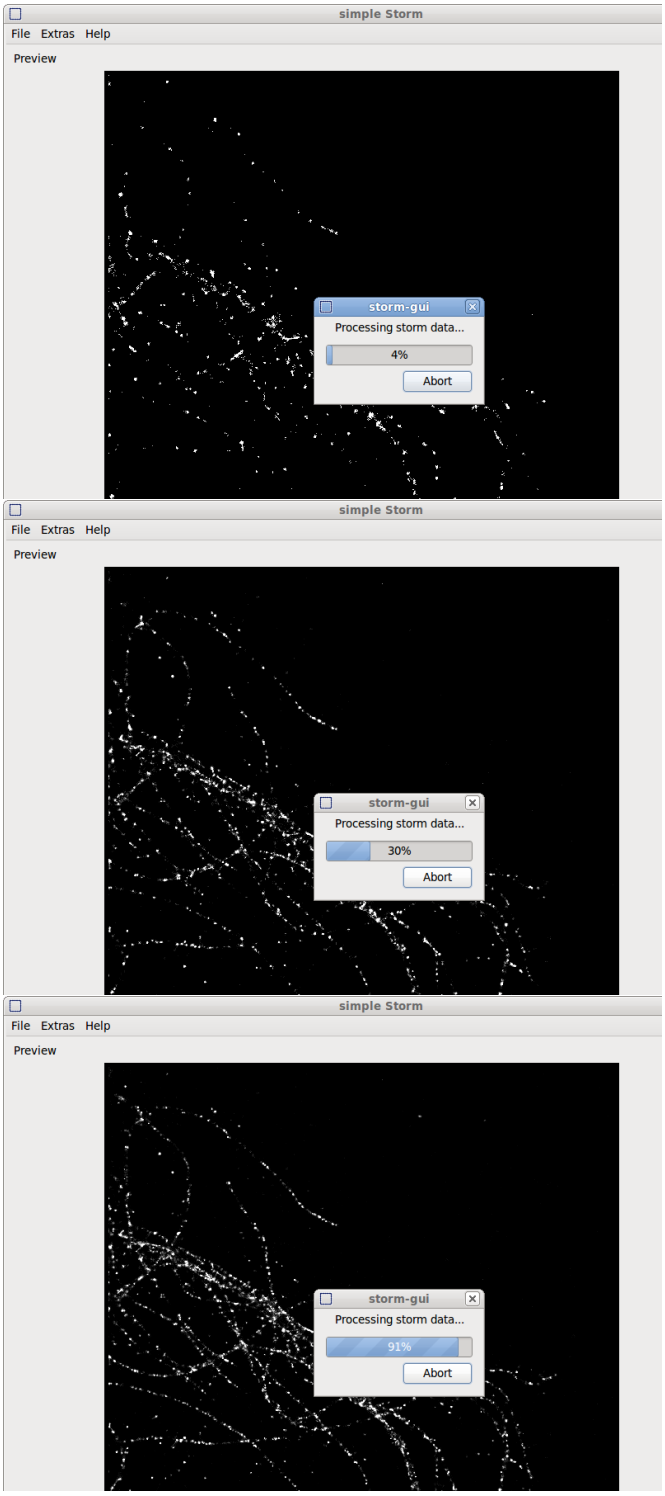

Figure 5.4: The preview image is updated during calculation with the spots already localized.

6 Field Programmable Gate Array

Field Programmable Gate Arrays (FPGA) are integrated circuits consisting of a huge number of configurable blocks operating in parallel. The chips are usually used for low-level data acquisition, control applications and preprocessing of sensor data. But the logic blocks can also be used to accelerate special purpose and highly parallel applications.

6.1 Architecture

FPGAs consist of a large number of configurable logic blocks (CLBs) that can be configured to implement any logical one-bit function. The interconnect network between the blocks is also configurable to build complex functions from those simple blocks. All CLBs work in parallel independently from each other. Figure 6.1 shows a schematic layout with a small number of logic blocks.

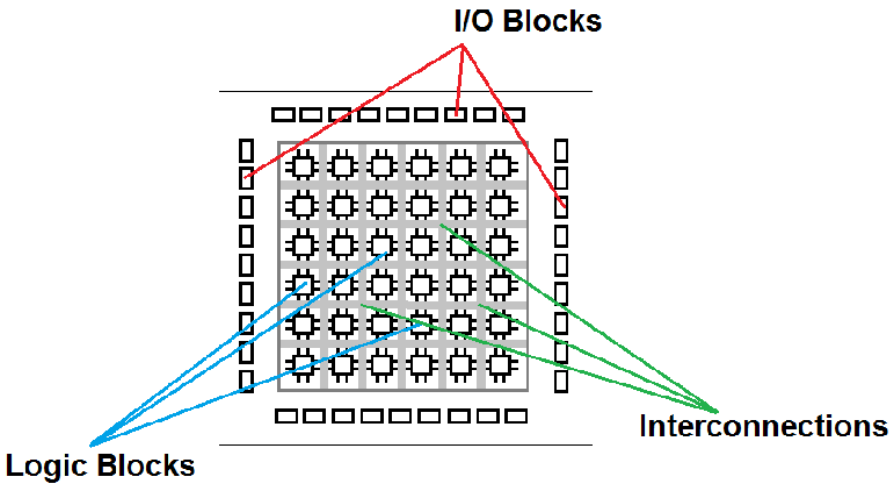

Figure 6.1: Schematic layout of an FPGA (taken from Clifford, 2011)

Additionally the logic blocks contain flip-flop stages to buffer intermediate results and make the design compatible to a global clock. The maximum clock speed in such devices depends on the complexity of the implemented design. The delay through wires and logic from one flip-flop stage to the next determines the clock frequency that can be applied. Additionally modern FPGAs have special purpose units and block-RAM modules that can be allocated by the user (cf. Xilinx, 2009, datasheet).

6.2 FPGA in image processing

FPGAs are "often found as part of a camera back end, used for converting the data coming from the sensor to more meaningful data to be transmitted from the camera" (Clifford, 2011). Such low level applications requiring realtime data processing and transmission are an ideal and relatively old use case for FPGAs. The chips are much cheaper than custom chips (ASICs) and can be easily adopted to new requirements by reconfiguring the same hardware chip.

With the growing number of CLBs and RAM resources per chip current FPGAs are much more powerful and can run complex image processing algorithms on-the-fly. The product "VisualApplets" from SiliconSoftware provides a design-tool to develop image processing applications for the FPGA without dealing with hardware-specific issues and hardware description languages.

6.3 VisualApplets (Silicon Software)

The design tool "VisualApplets" provides a high-level graphical interface to draw data flow diagrams and connect different image processing operators. The operators itself are provided as closed applets that can be parametrized at synthesis- or at runtime. The hardware specific resources are displayed to tell the user how many resources are left on the chip. If no more resources (logic, RAM, flip-flops) are available, the user has to simplify the algorithm.

The FPGA architecture as described above is fundamentally different from general purpose central processing units. CPUs work on 32- or 64-bit numbers and have a limited number of instructions implemented but usually run at much higher clock frequencies (multiple Gigahertz). One major difference between a PC and a FPGA-architecture is the number format: A CPU has built-in, optimized support for floating-point operations, a FPGA works on single-bit level. Floating point operations can be supported in FPGAs as well but integer operations require less resources.

In VisualApplets there are no floating-point operations available so we assume that one knows the input range beforehand and uses integer arithmetic. Fixed-point numbers have to be represented by remembering the number of bits shifted (where to put the decimal place afterwards).

Another major difference is that a FPGA can handle images as a data-flow and is not limited by the number of local registers. Subsequent instructions operating on the same data can be pipelined adding as many register stages as needed. In fact pipelining is the main reason why the FPGA version of an algorithm can run in the same or even less time than a CPU version. The clock frequency of a microEnable IV FPGA frame grabber is 62.5 MHz. A current CPU has a 50 times faster clock rate. But the FPGA pipeline produces one output value in every clock cycle. In contrast

a CPU has to issue up to several hundred instructions to calculate a single result value.

Some drawbacks of FPGAs have to be mentioned as well: The longer development times for synthesis and routing of an FPGA design make it inflexible. Small algorithmic changes can be tested easily in a CPU implementation while the FPGA synthesis takes up to several hours.

6.4 Available operators in VisualApplets

In the following we give a list of well understood algorithms that are implemented in the VIGRA C++ library. On the other hand there are implementations of those or similar algorithms in FPGAs. We refer to the implementation in Visual Applets and give a tabular comparison with some comments about differences between the C++ and the hardware implementation. Table 6.1 on page 53 compares the names and functionality of the different operators. More comments about different types and missing operators can be found below.

Several of the higher-level operators in VIGRA can be composed in VisualApplets using low-level operators. That means VisualApplets is fully featured for basic image processing tasks but it would be appreciable to have such operators as convenience functions available. For example it is much easier to adjust a sigma parameter for a separable Gaussian smoothing than typing all filter coefficients of a lower level FIR-filter by hand.

ROI handling Before describing single operators we want to point out another design difference: VIGRA has functions for combining images that take a binary mask as an additional input image (e.g. `combineTwoImagesIf`). This is a very flexible approach for special applications. VisualApplets implements only operators to select rectangular regions of interest. Masks on pixel level must be implemented by adding a multiplexer after promoting both results in parallel. When using O-type operators this can easily be implemented using a `Case` module connected to both paths and the mask as switch input.

In FPGA designs such concurrent computations do not increase the overall runtime. All logic cells operate in parallel and data paths computing results that are discarded afterwards can help to achieve high overall data rates.

Pixel-wise arithmetics Several functions can be applied pixel-wise to an image. This is the simplest class of operations that are applied in image processing: Given an input image and an output image one can scale by a constant or calculate the squared value at each pixel for example. Combining two images adds no more complexity but simply more storage capacity. The sum or difference of two or three

images is often used. The functionality of `vigra::transformImage` (using functors) is provided by several different operators in Visual Applets (cf. table 6.1).

Pixel-wise color conversion Conversion between different color spaces is supported:

VIGRA	VisualApplets
RGB, XYZ, Polar HSB, LAB, LUV, Y'CbCr, Y'IQ, Y'PbPr	RGB, HSI, XYZ, YUV

Additional functions are white balance (scaling the channels with different factors) and extraction of single channels into separate gray level images. VisualApplets also supports `ColorTransform` as a multiplication with a 3x3-matrix and `Bayer` reconstruction.

Neighborhood-Operations Kernel-based operations are often used to filter images or compute derivatives. Separable and non-separable filters are available and provide the basis for many image processing applications.

Recursive filters (IIR) are less commonly used in practice but provide some additional properties that will be discussed in section 6.5. Recursive Filters are not available in VisualApplets.

Resizing an Image replicates pixels based on local information from the neighborhood of pixels. Different interpolation methods are available in VIGRA; VisualApplets does only provide nearest-neighbor interpolation. The operators `sampleUp` and `sampleDn` increase or decrease the size by powers of two. Some types of interpolation can be built up manually utilizing FIR-filters with appropriate coefficients. We give more details about these compositions in section 6.5.

Global Operations: Image Statistics There are several operations that use a larger or very different part of the image than just neighboring pixels. The only such operation that is implemented in VisualApplets is an operator for the global maximum inside rows / columns or inside the whole frame (see table 6.2).

Other useful global operations such as rotation of an image can be assembled from available operators. They are described as examples in the following section.

6.5 Additional custom operators for VisualApplets

With the closed-source operators listed above one can define his own operators as hierarchical blocks. Many algorithms can be assembled out of these relatively simple operators. Some example implementations of commonly used image processing functions are given in this section.

Table 6.1: Comparison of available basic operators in VIGRA and VisualApplets

	VIGRA function name	VisualApplets	comment
pixel-wise operations	`transformImage`	`abs, arccos, arccot, arcsin, arctan, cos, cot, scale, sin, sqrt, tan, rnd, shiftright, shiftleft`	missing: `log2i`, `exp` or `pow`, `sign`, `arctan2`, `frac`
	–	`and`, `or`, `not`, `xor`, `xnor`, `cmp`, `if`	low-level logic operators
	`combineTwoImages`	`add`, `div`, `mult`, `sub`	missing: `gcd, lcm`
	`combineThreeImages`	–	three or more images can be combined using operators with two inputs consecutively
filters	`separableConvolveX/Y`	`FIRkernelNxM1D`	
	`convolveImage`	`FIRkernelNXM`, `FIRoperatorNxM`	
	`normalizedConvolveImage`	–	normalize by sum of filter coefficients
	`gaussianGradient` etc.	–	convenience functions for smoothing and derivative filters
	`recursiveFilterX/Y`	–	
	convenience functions for smoothing and derivative filters	–	
morphology	`discDilation`	`DILATE`	
	`discErosion`	`ERODE`	
	`discMedian`	`MEDIAN`	
	`discRankOrderFilter`	`MAX, MIN`	`MAX` / `MIN` are special cases of `discRankOrderF.`
resize	`resampleConvolveImage`	–	helper function to resample and filter with a kernel
	`resizeImageNoInterpolation`	`sampleUp`, `sampleDn`	
	`resizeImageLinearInterpolation`	–	easily built with available operators
	`resizeImageSplineInterpolation`	–	
	`resizeImageCatmullRomInterpol.`	–	can be combined from existing operators in VA
	`resizeImageCoscotInterpolation`	–	
	`localMaxima`, `localMinima`	–	can be built with available operators in VA
	`labelImage`	`Blob_Analysis_1D/2D`	

VIGRA function name	VisualApplets	comment
`inspectImage` with corresponding functors	`ColMax, ColMin, ColSum, RowMax, RowMin, RowSum, FrameMax, FrameMin, FrameSum`	

Table 6.2: Some available operators with global image scope

Local Maximum By comparing the neighboring pixels one can mark maxima in an image. For 2-dimensional images the 4- or 8-neighborhood are common choices. Our implementation uses the operator `FIRkernelNxM` to get 3×3 neighboring pixels and compares those with the current (central) pixel. The eight comparisons are independent from each other and are all done in parallel. A 4-neighborhood would be implemented correspondingly. The VA-design is shown in figure 6.2 and has an additional dynamic threshold parameter. With appropriate thresholds maxima in the background can be suppressed.

Rotate / Transpose Image Rotating an image by 90 degrees or transposing it are operations that have to simply copy the pixels in a re-sorted order into the output image. No such functions are implemented in Visual Applets by default but they can be built with `FrameMemory`: On the microEnable FPGA framegrabber whole images can be stored in off-chip DRAM. Our *rotate* and *transpose* operations copy small portions of the image as ROIs from the external DRAM. This small amounts of data are buffered re-ordered in FPGA block RAM so that the appended regions give the transposed image.

An example design is given in figure 6.3. The $128 \times 128\,\mathrm{px}^2$ image with 16 bit is split into four regions of $32\,\mathrm{px} \times 128\,\mathrm{px}$ and the double-buffered `FrameBuffer` has a size of $128\,\mathrm{px} \times 32\,\mathrm{px} \mathrel{\widehat{=}} 64\,\mathrm{kBit}$ thus utilizing 8 block RAMs. To use less memory the single ROIs should have minimal width though full image height is needed (images cannot be concatenated horizontally afterwards).

In cases where "large" separable filters or filters operating only in y-direction are required, transposing the image can be cheaper in terms of internal block RAM. The added delay for transferring the image into external DRAM does not affect the overall data rate but only adds a delay before the first pixel is available.

Spline Interpolation As described in section 2.4 a separable interpolation can be achieved by convolution with a kernel using s neighbors of the original pixel. Figure 6.4 shows an example implementation of an interpolation by a factor of eight with a spline of order 3 (using kernels of size 4×1). The resulting image has $\mathrm{width}_{\mathrm{output}} = 8 \cdot \mathrm{width}_{\mathrm{input}}$; the last 7 pixels have to be discarded since those pixels are no longer an interpolation *inside* the image.

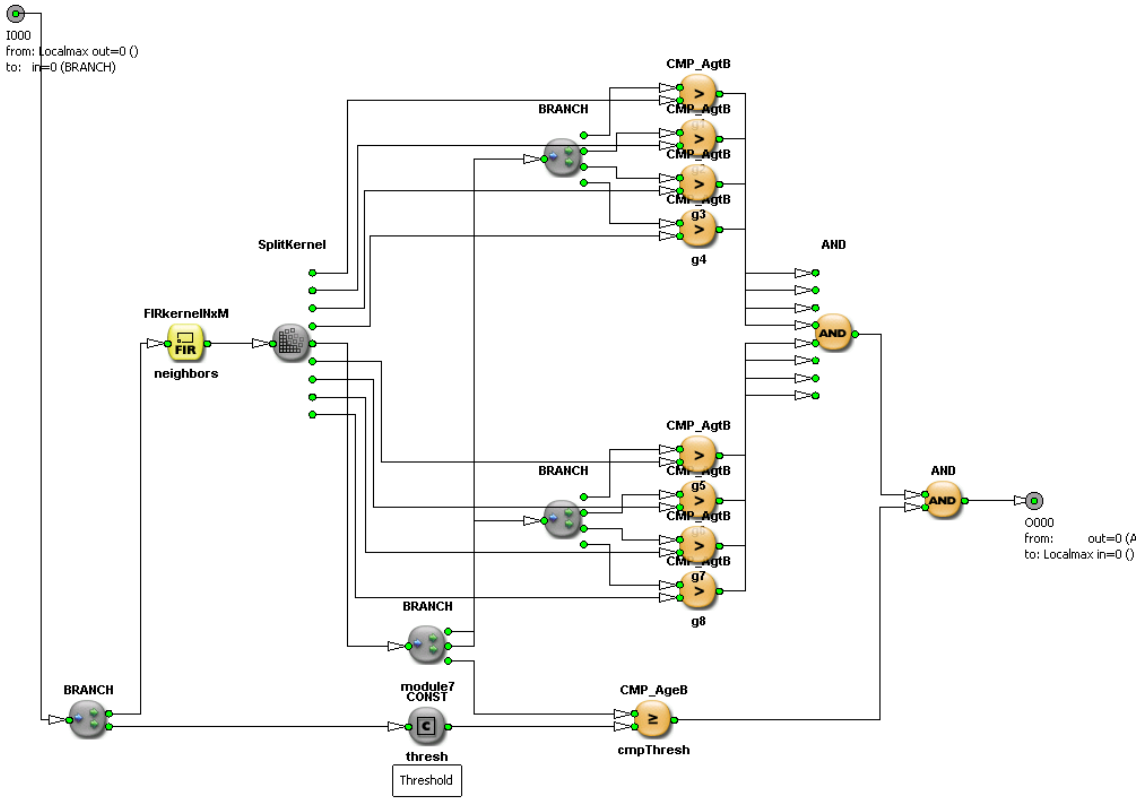

Figure 6.2: VisualApplets design for the detection of local maxima

	p_{l-1}	p_l	p_{l+1}	p_{l+2}
k_0	170	683	170	0
k_1	114	668	242	0
k_2	72	627	323	2
k_3	41	566	408	9
k_4	21	491	491	21
k_5	9	408	566	41
k_6	2	323	627	72
k_7	0	170	683	170

Table 6.3: Filter coefficients for 1-dimensional B-Spline approximation with upsampling by a factor of 8. The filter coefficients sum up to a filter norm of 1024.

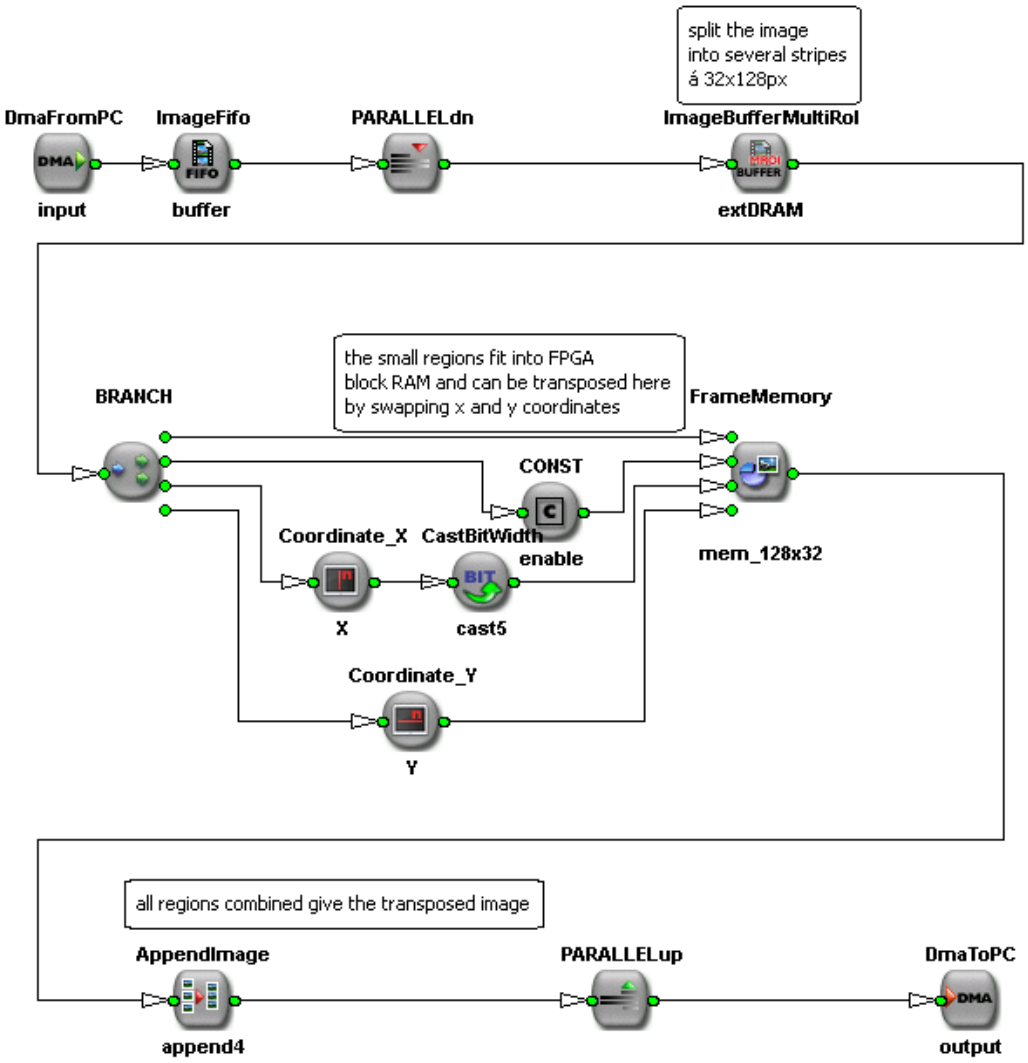

Figure 6.3: Transpose image via external DRAM and internal block RAM

Every subpixel needs a different convolution kernel (weights for the neighboring pixels). The filter coefficients are calculated from sampling $B_3(z)$ (equation 2.6) at appropriate positions. The coefficients are scaled to a filter norm of 1024 (10 bit coefficients) and the kernels used are listed in table 6.3. Since the bit width increased by ten bits after filtering, we need to cast the result back to a reasonable number of bits throwing away the least significant bits. The inherent parallelism of the FPGA allows us to compute all new subpixel values in parallel. Afterwards the module `MergeParallel` is used to increase the image resolution.

The interpolation in y-direction depicted in figure 6.5 is done similarly but there is no operator for merging pixels in y-direction. Thus we calculate the results sequentially line by line and increase the resolution beforehand. `SampleUp` repeats pixels (no interpolation) and has a static parameter to set the resampling factor. The following `FIRKernel` has to buffer $8 \cdot 3 + 1 = 25$ pixels to access four neighboring values, since eight subsequent lines contain identical values after `SampleUp`. A `ModuloCount` operator selects the current row result; the other resulting lines are discarded by the case operator and recomputed. Thus we cannot use the intrinsic parallelism here as we do for the x-interpolation.

Concerning the used *logic elements* the y-interpolation is not more expensive than x-interpolation. The large 25-row `FIRoperator`s contain the same four non-zero elements as the corresponding x-interpolation and are thus reduced during synthesis and only connected to those four buffered values.

But the *RAM* used by this y-interpolation is much higher than for the analogon in x-direction. All 24 intermediate lines have to be buffered in FPGA block RAM. Our overall impression from this implementation is that a specialized operator in VisualApplets could heavily improve RAM usage for the y-interpolation.

Recursive Filter "As convolution requires many operations, the question arises whether it is possible or even advantageous to include the already convolved neighboring gray values into the convolution at the next pixel. In this way, we might be able to do a convolution with fewer operations. [...] However, these filters, which are called *recursive filters*, are much more difficult to understand and to handle" (Jähne, 2005, page 122).

The easiest one-dimensional recursive filter of first order is given by

$$I'(x, y) = \alpha \cdot I(x, y) + \beta \cdot I'(x - 1, y).$$

Because of their large (in general infinite) impulse response recursive filters can approximate FIR filters with huge kernels. For example Young and van Vliet (1995) introduced a recursive implementation of Gaussian filters with a computational effort independent of σ. As another example we described in section 2.4, that a prefilter is required to sharpen an image before BSpline interpolation. Derivatives can be computed using recursive filters as well.

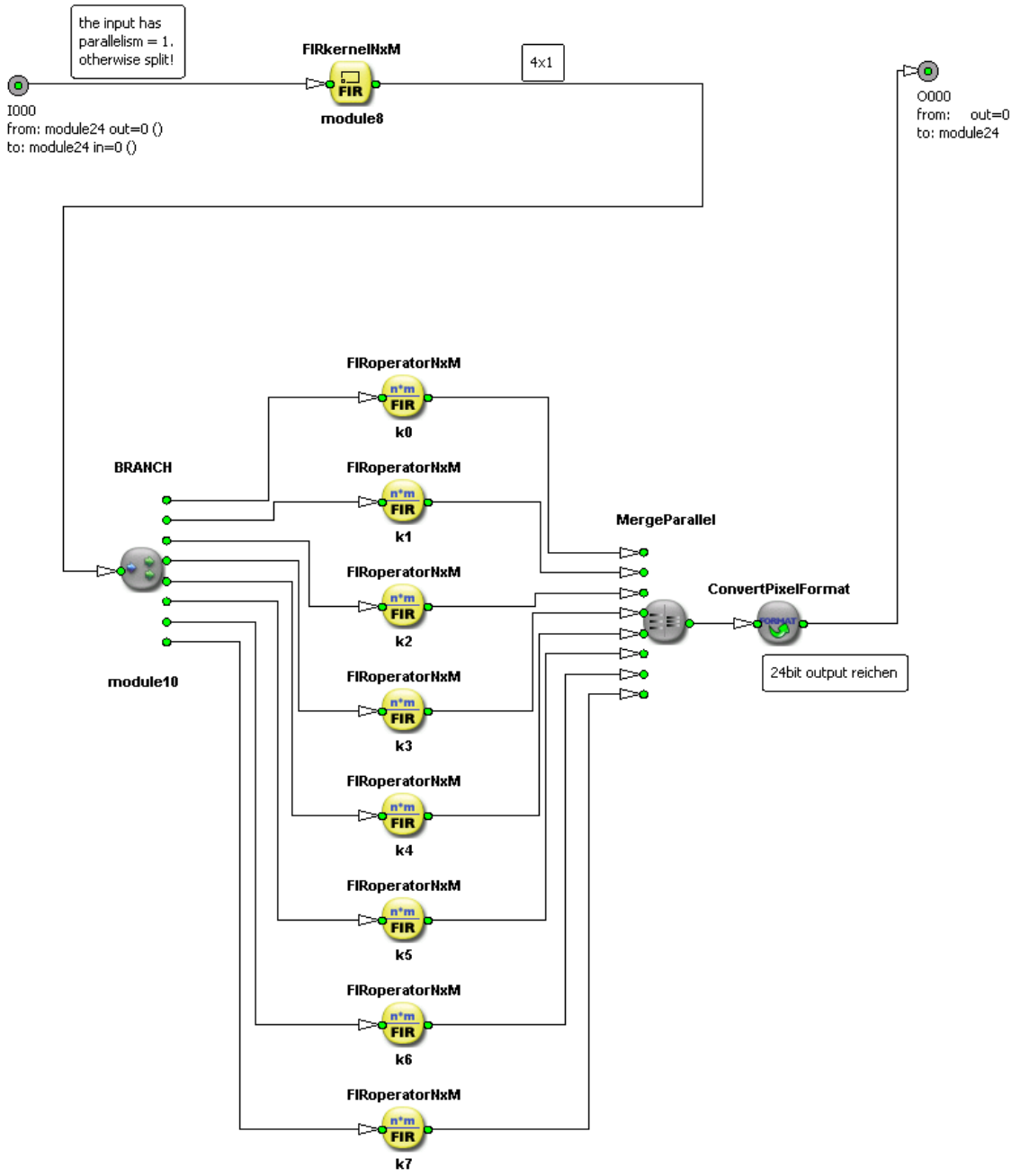

Figure 6.4: Interpolation in x-direction. The actual spline is determined by the coefficients of `FIRoperatorNxM`. By merging eight pixels the resulting image has 8x the original width.

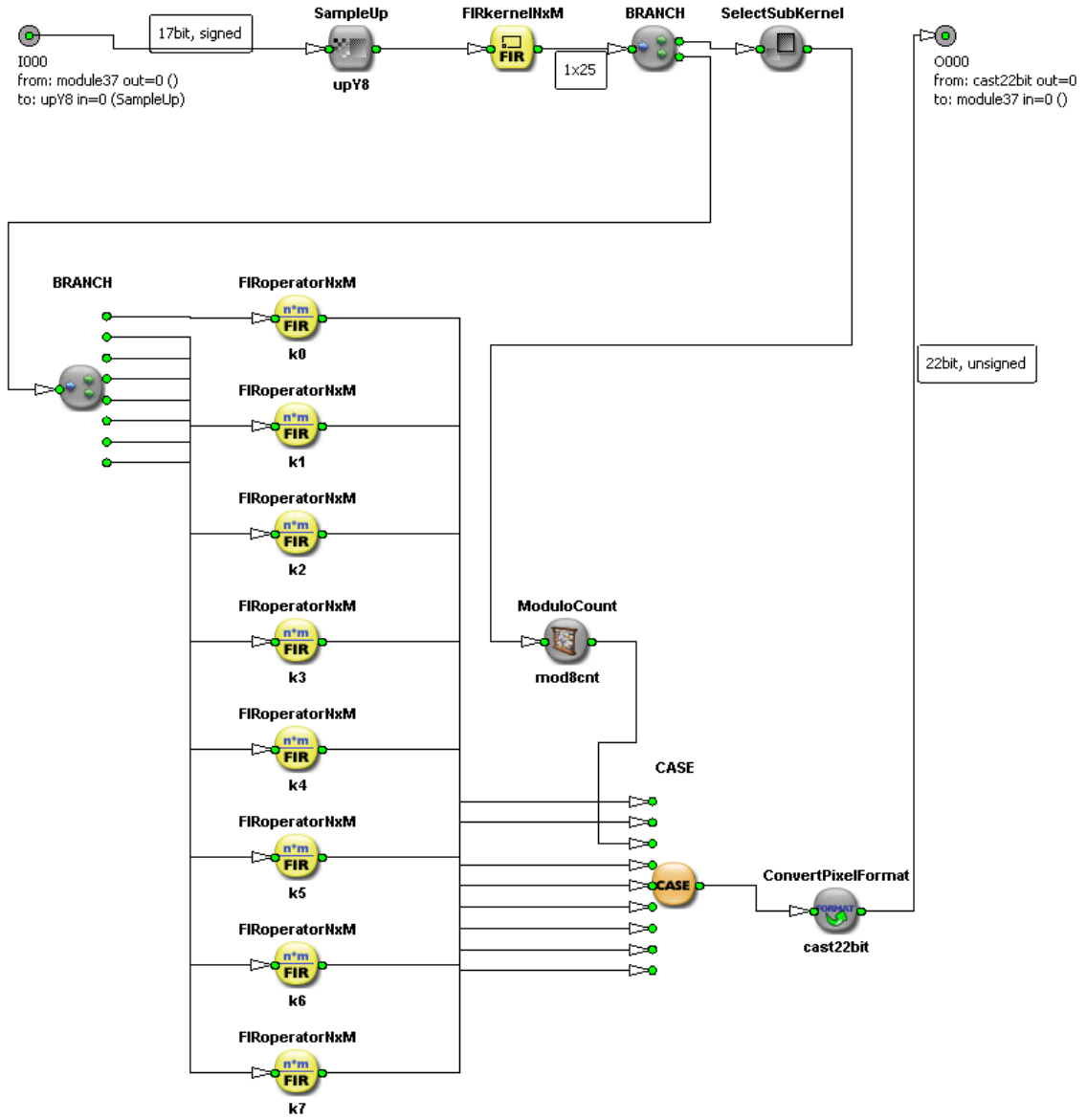

Figure 6.5: Interpolation in y-direction. The spline is determined by four non-zero coefficients of `FIRoperatorNxM`. In contrast to the x-interpolation we need large buffers here (see text for explanation).

Recursive filters just propagate signals into one direction. This is reasonable for time series but not meaningful for spatial data as images. To avoid a phase shift, the filter has to be applied both forwards and backwards. This makes a FPGA implementation more complicated but `FrameBuffer`s can mirror an image so that a filter could be applied first causal and then anti-causal.

Recursive filters are currently not implemented in VisualApplets. We tried an implementation with the available operators but did not succeed because of the following restrictions:

Loops inside a design, connecting the output of a module to its predecessors, are not allowed in VisualApplets. This is a strong constraint that is verified during the design rules check. The prohibition of loops means that we cannot implement a recursive filter in a single VisualApplets design. We tried to build a recursive filter manually using a PixelPlant addon-board. Data streams can be sent from the image processing FPGA on a microEnable to FPGA on the PixelPlant to increase the processing power. This channel can also be utilized to insert already processed data back into the main data path, essentially resulting in a loop. Special care has to be taken for the starting conditions. The data flow has to be started with initial values, otherwise the applet will end in a dead-lock. For a loop on image level the operator `InsertImage` is used, but on pixel level we could not find an equivalent to start the execution.

During synthesis of a VisualApplets design using the PixelPlant several buffers are added so that the maximum data rate can be achieved on the connection between microEnable and PixelPlant. We measured the delay introduced by those buffers to be about 70 clock cycles. This means that a PixelPlant loop would probably not be useful for implementing a recursive filter, since subsequent calculations need the previous results immediately.

However, a vendor implementation as new operator inside VisualApplets seems to be an easy solution. With hardware description languages loops and buffers for previous results are no problem. Appropriate border handling has to be added at the beginning and end of each line.

With such an operator available we would have implemented the complete recursive filter with `LineMirror` and `RotateImage` operations as hierarchical boxes. The design proposed in figure 6.6 needs a large amount of internal FPGA RAM: Following the example of an 128×128 px image from above the `transpose` operators would utilize eight block RAMs each. With two block RAMs for every `LineMirror` a total of 20 RAM blocks is used. This are 56% of the resources available in a Spartan 3E 1600 device and 20.8% of the RAM built into a Spartan 3 4000, respectively. The logic resources used for the filter are small: The filter computation considers very few pixels (only one pixel and its predecessor for a first order filter). The image transformations afterwards, to apply the filter in the correct direction, are pure memory access operations. An optimized implementation of `RotateImage` could reduce the FPGA block RAM requirements. Then recursive filters have only

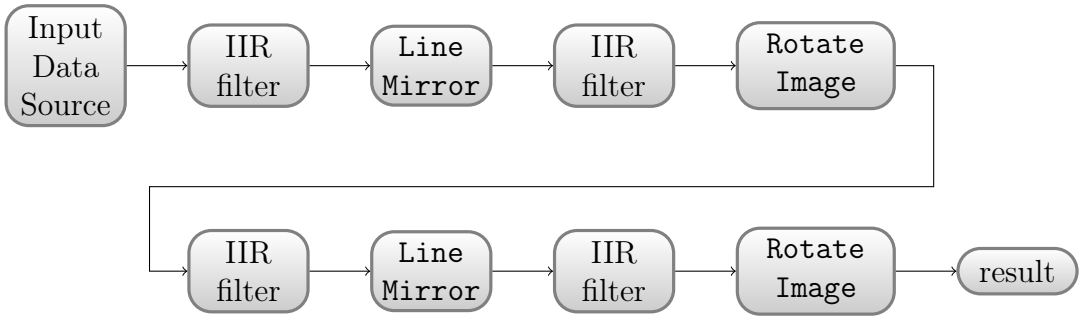

Figure 6.6: Design proposal for a recursive filter using a hypothetical low level IIR filter operator and the hierarchical block `rotateImage` described in this section. The low-level filter operator is applied forwards and backwards both in x- and y-direction.

one drawback left against FIR filters: They add a significant delay for buffering the image in external DRAM. But that does not affect the data throughput. Overall recursive filters are a promising alternative to huge FIR kernels that are impossible to compute with current FPGA framegrabbers.

6.6 dSTORM processing algorithm on FPGA

Using the hierarchical boxes described in the previous section, we now have all the operations to process the storm data completely inside the FPGA framegrabber. Since we have no CameraLink compatible camera available we implemented the test design using Direct Memory Access (DMA) data transfer from the host PC. After further tests the implementation could be switched to the microscopy environment. The framegrabber would be directly connected to the camera of the microscope to get a realtime image of the scene. The operator of the microscope would profit from an immediate feedback of the processed image, so that other parameters of the setup can be adjusted more easily. The implementation was tested with an microEnable IV VD1-CL that has a Xilinx Spartan 3E 1600 FPGA as image processor.

For the algorithmic details see section 2; in the following we give a summary of the FPGA specific modifications only. The implementation is shown in figure 6.7. The Wiener filter is calculated before the actual processing using a sample measurement to determine the noise and signal power spectrum (corresponding to the shape of the PSF). The Wiener filter was transformed from Fourier domain to spatial domain. Finally we cropped the filter to an 9×9 two-dimensional kernel and then synthesized it for the FPGA hardware. There is currently no option to select or modify this pre-filter at runtime.

The BSpline approximation with coefficients from table 6.3 is used to interpolate the image first in y-, then in x-direction. During the x-interpolation the parallelism

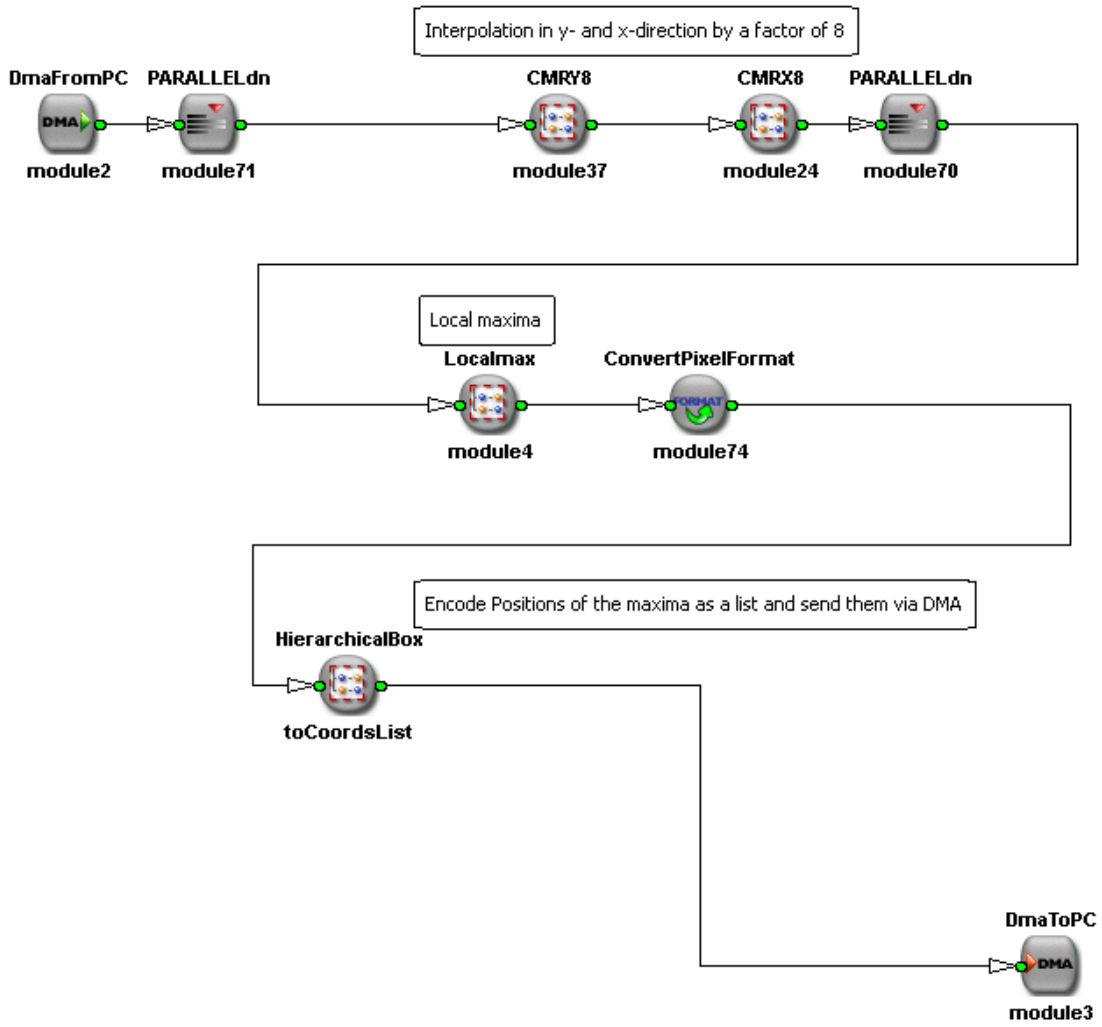

Figure 6.7: The final dSTORM processing module in VisualApplets

is increased to four. Local maxima are detected and encoded as list of coordinates that are then sent to the host for final storage and assembling into a result image. The implementation of the coordinates encoding is depicted in figure 6.8.

Bit width An FPGA has internally a bit-wise architecture; they are not arranged as bytes or even multiple bytes as in a CPU processor. VisualApplets has signed and unsigned integer data types, and the bit width can be set manually. Since the microscopic images have 14 bit resolution we start from that number of bits, and carefully select the bit width after each stage. In every filter or interpolation stage several bits of accuracy are added. We tried to maintain the number of bits as low as possible to reduce the size of needed RAM and logic resources.

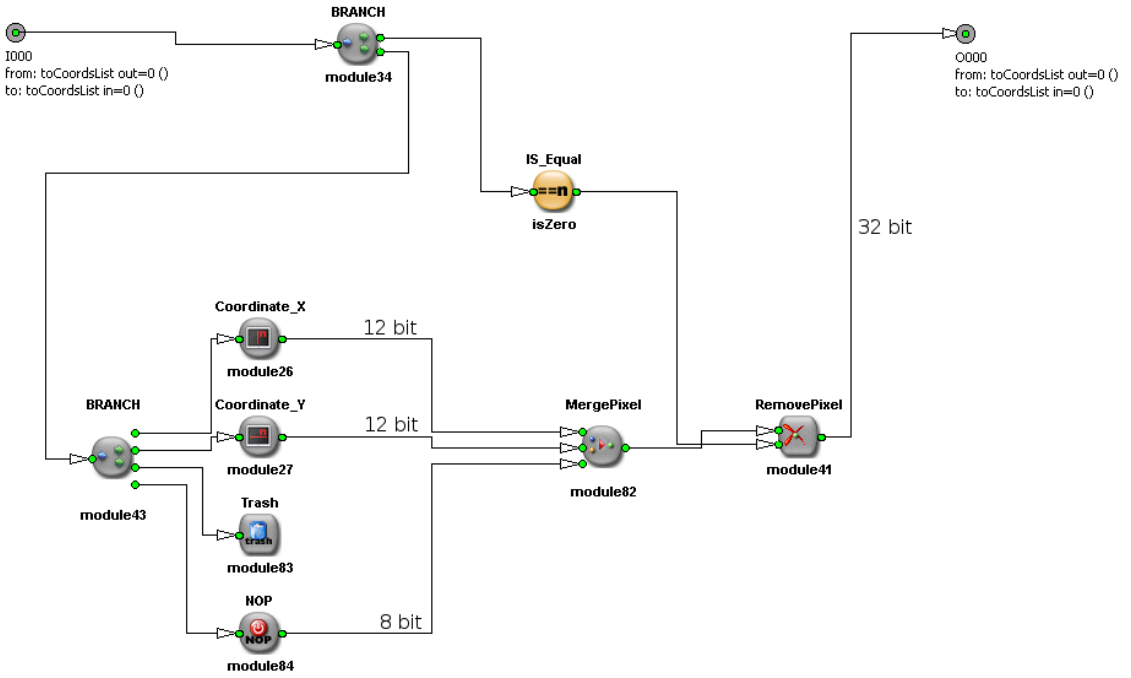

Figure 6.8: Encoding of the positions of pixels that are non-zero to send them as list back to the host PC

After the Wiener filter we crop the results back to 16 bit, the interpolation steps increase that to 22 bit and 24 bit resolution. That is the number of bits an IEEE 754 floating point number has for its mantissa. After determining the maxima we cast the width down to 8 bit, since the exact position is much more relevant than the intensity. The absolute intensity value can be only a rough approximation of the photon count (multiplied with camera gain) since the background and CCD baseline has to be taken into account as well. The interpolated image has a maximal with of 4096 px so that the maxima positions can be encoded with 12 bit numbers. Merging x- and y-coordinate together with the 8 bit intensity into one pixel we have to transfer 32 bit wide numbers to the host PC for each maximum detected.

Although the integer number format used on FPGA has no dynamic range floating point capability, the resulting images and coordinates have a very similar accuracy as the results from our C++ implementation.

DMA data transfer and SDK Silicon Software offers a SDK-Interface for writing a C++-application that communicates with the FPGA device and manages the data transfer. Specially allocated memory is capable to be transferred to the framegrabber device without processor intervention (direct memory access, DMA). We use those function calls to allocate memory, and fill it with an example data set from an hdf5-file. The images are then sent to the device frame by frame. Retrieved result coordinates are saved to a raw file on disk. The post-processing and assembling of the result image from the coordinates file is currently done afterwards with python helper scripts on the CPU.

Runtime The whole design is pipelined so that we can easily calculate the data throughput according to the image dimensions and the parallelism used. The clock frequency of the device is fixed to 62.5 MHz with a processing power of one pixel per clock cycle. In theory in every single clock cycle one pixel is fed into the pipeline, and one result pixel leaves the pipeline. In our case the maximum data rate is only used after the upsampling step. Before the y-interpolation all pixels are delayed until further input can be processed.

Processing an input image of size $w \times h$ we have to process $64 \cdot w \cdot h$ pixels after upsampling with a factor of 8. We process the data with a parallelism of 4 at a clock rate of 62.5 MHz leading to a theoretical input data rate of $62.5 \cdot 4/64$ MPx/s = 3.9 MPx/s.

In comparison to the corresponding single core C++ variant the FPGA implementation gains a speedup factor of ten (cf. table 6.4). Taking the difference in clock frequency into account, the FPGA version must be running a pipeline of several hundred stages.

An overall runtime and speedup comparison with the CPU versions can be found in section 7.

	runtime [sec]	throughput [Mpix/sec]
C++ (without ROIs)	208.3	0.3
C++ (final version, dual core)	18.4	3.6
FPGA (without ROIs), theoretical	16.0	3.9
FPGA (without ROIs), measured	18.0	3.6

Table 6.4: Runtime comparison between our hardware and software variants

Resources used The design fits into an microEnable IV VD1-CL framegrabber FPGA. For tests we split the design into two parts to speedup the Place-and-Route procedure. The algorithm without wiener filter consumes 83% of the internal RAM, 64% of the logic and 74% of all flip-flops. After adding the Wiener filter to the design, the synthesis took several hours until timing-closure was reached. This combined design uses 91% of the RAM modules, 76% of logic and 97% of flip-flop resources.

To further speedup the applet, one could restrict the interpolation to regions of interest as in the CPU version. Therefore a larger FPGA model is needed to implement additional logic. The PixelPlant offers much more internal resources and can be used for more powerful designs. Our simple SDK solution is then no longer applicable, since the number of ROIs in an image is not fixed. One would have to invest much more effort for synchronization and prevention of dead-locks. Since tests with a real camera input were not expected, such a solution would have resulted in a prototype or proof-of-concept only. Thus we dropped this idea in favor of an optimized C++ version, that is now applicable in every-day use.

For applicability of a FPGA applet in real-time one has to keep in mind, that a ROI based implementation has varying runtime depending on the density of input spots. That means, it is hard to guarantee a fixed data rate for real-time capturing. Large input buffers would have to compensate for varying spot density and the experimental setup must make sure, that no buffer overflows occur. Otherwise the size of captured images could be reduced to maintain high frame rates. Our simple FPGA version however, always interpolates the complete image and thus has a fixed data throughput independently of the number of spots within an image.

6.7 Outlook: More complex operators for Visual Applets

The evaluation of VisualApplets showed, that the available low-level FPGA operators allow us to implement most basic image processing functions. The scope of the application is focused on modules that work on pixel level or just local neighborhoods. But some global operations can be built as well.

To make FPGA framegrabbers ready for wider application ranges different steps could be useful: VisualApplets uses pipelining and pixel-level parallelism to speedup execution. Parallelism on higher levels could be implemented as well and speedup

computational expensive problems. Global operators that take larger portions of an image into account are probably difficult to implement but could further improve the applicability of FPGA framegrabbers to more complex tasks. Simultaneously the resources that are available in low-cost FPGAs are increasing so that more involved operators are possible in the near future.

Some concrete example functions that are often used in image processing are:

A Discrete *Fourier Transform* operator would be very useful to analyze or filter signals in frequency domain. For 1D inputs the one-dimensional operators provided by Xilinx as IP cores could be used directly (Xilinx, 2011); for 2D Fourier transforms those could be used as a basis for a row-column decomposition or more sophisticated implementations (cf. Yu et al., 2011).

Image segmentation algorithms as *seeded region growing* or *watersheds* add labels to different image regions minimizing a global cost function. Using a *distanceTransform* one can easily compute a Voronoi tesselation of an image.

For the handling of *image pyramids* a more convenient way of up-/ down-sampling images is needed. Often multi-scale approaches work on the same image at different scales.

Acceleration of other operations would be desirable for science and advanced applications. Statistical learning and prediction procedures and linear algebra functions can be rather slow on CPU architectures and could potentially benefit from the high intrinsic parallelism in FPGAs.

Decision Trees and random forests are commonly used in machine learning. They provide a general framework for selection of relevant information from high-dimensional input data. A random forest uses up to several hundred trees to predict the class of input data based on local features. All trees can be processed in parallel and inside each trees several pipeline stages are possible. But the overall prediction procedure doesn't match the data-flow principle of VisualApplets at all.

7 Conclusion

Super-resolution microscopy can achieve an optical resolution of 20 nm. The data processing algorithms for localization microscopy have to localize hundred thousands of spots in several thousand single frames to reconstruct the image.

We presented a novel data processing algorithm that does not rely to fitting but uses standard image processing operations. For medium image sizes the processing can be completed in the same time as is needed for acquisition. An FPGA implementation using VisualApplets from SiliconSoftware was presented for on-the-fly processing of the images. The currently used camera of our collaborators in biology is not compatible with CameraLink framegrabbers so that realtime tests were not possible. For now the data were recorded and analyzed afterwards.

Speedup and Accuracy

The runtimes of different implementations on CPU and our FPGA implementation are summarized in table 7.1. We measured the execution time on a test data set and computed the corresponding data throughput. The data set contains 4000 frames with 128 px × 128 px each. All CPU measurements are on an Intel(R) Pentium(R) Dual CPU E2180 running at 2.00GHz and the FPGA framegrabber is an microEnable IV-VD1 with Spartan 3E running at 62.5 MHz. Approximately 120×10^3 single spots were detected.

The algorithmic optimizations described in section 5.2 led to a speedup factor of 20 against the first python implementation (using the optimized C++-Vigra bindings). The major parts of this improvement were ROI-Selection and multithreaded execution. With this code medium size images can already be analyzed at the same speed as is required for the acquisition of the time series. To gain another speedup to process the full field of view of the camera or to work with higher frame rates special hardware is necessary. Many-Core systems would be applicable; we tested

	runtime [sec]	throughput [Mpix/sec]
Python prototype	≈ 400	
FPGA (without ROIs)	18.0	3.6
C++	18.4	3.6
Realtime ($(512\,\text{px})^2$ @ 35 fps)		9.2
rapidSTORM (Wolter et al., 2010)	24	2.7

Table 7.1: Runtime comparison between different hardware and software variants

the use of specialized FPGA-framegrabbers. The FPGA version outperformed the equivalent CPU variant by a factor of 10 – although the clock rate of the FPGA board is 40 times lower than the CPU frequency. An extensive use of pipelining is responsible for the high throughput of the microEnable framegrabber devices.

We have shown that the accuracy and precision of our algorithm is comparable to that of other methods. The fixed point data format used in the FPGA version does not decrease the accuracy. Both precision and accuracy were evaluated on simulated data and on bead measurements.

Summary and Outlook

In addition to data processing we analyzed multicolor measurements. Two channels can be aligned with standard affine linear transformations fitted to beads used as landmarks. In our examples less than ten corresponding beads were available per image. We aligned the images with RMS errors of about 15 nm. This is the same order of magnitude as can be achieved for the structural resolution of a single image. If more landmarks are available or a calibration measurement can be afforded without changing the optical settings, more sophisticated registration algorithms could be used. Thin plate splines could be tried for registration.

The evaluation of the FPGA environment has shown, that many basic image processing algorithms can profit from pipelining. On-the-fly data processing makes a real-time preview of super-resolution images possible. Recursive filters and Fourier transformation were not available in the VisualApplets environment. Implementing such additional operators could further improve the applicability of FPGA framegrabbers for image processing tasks.

The C++ implementation is used for dSTORM measurements by our collaboration partners and outperforms other algorithms in runtime and precision. In terms of accuracy our algorithm is nearly as good as the rapidSTORM fitting procedure, and outperforms the QuickPALM ImageJ plugin.

Super-resolution microscopy progressed rapidly during the last decade and it will be fascinating, how this methods evolve. Better time-resolution and simplified handling of localization microscopy have to be combined with rapid data processing. Perhaps these methods can open the way for a better understanding of living organisms and eradicate incurable diseases in near future.

Part III

Appendix

A Lists

A.1 List of Figures

A.2 List of Tables

B Bibliography

Ernst Karl Abbe. Beiträge zur Theorie des Mikroskops und der mikroskopischen Wahrnehmung. *Archiv für Mikroskopische Anatomie*, 9:456, 1873. ISSN 0176-7364. doi: 10.1007/BF02956175.

Sean B. Andersson. Precise localization of fluorescent probes without numerical fitting. In *Biomedical Imaging: From Nano to Macro, 2007. ISBI 2007. 4th IEEE International Symposium on*, pages 252 –255, April 2007. doi: 10.1109/ISBI.2007.356836.

Sean B. Andersson. Localization of a fluorescent source without numerical fitting. *Opt. Express*, 16(23):18714–18724, November 2008. doi: 10.1364/OE.16.018714.

Eric Betzig, George H. Patterson, Rachid Sougrat, O. Wolf Lindwasser, Scott Olenych, Juan S. Bonifacino, Michael W. Davidson, Jennifer Lippincott-Schwartz, and Harald F. Hess. Imaging intracellular fluorescent proteins at nanometer resolution. *Science*, 313(5793):1642–1645, 2006. doi: 10.1126/science.1127344.

F.L. Bookstein. Principal warps: thin-plate splines and the decomposition of deformations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11 (6):567 –585, June 1989. ISSN 0162-8828. doi: 10.1109/34.24792.

J. Brauers and T. Aach. Geometric calibration of lens and filter distortions for multispectral filter-wheel cameras. *Image Processing, IEEE Transactions on*, 20 (2):496 –505, February 2011. ISSN 1057-7149. doi: 10.1109/TIP.2010.2062193.

J. Brauers, N. Schulte, and T. Aach. Multispectral filter-wheel cameras: Geometric distortion model and compensation algorithms. *Image Processing, IEEE Transactions on*, 17(12):2368 –2380, December 2008. ISSN 1057-7149. doi: 10.1109/TIP.2008.2006605.

L. Stirling Churchman, Zeynep Ökten, Ronald S. Rock, John F. Dawson, and James A. Spudich. Single molecule high-resolution colocalization of Cy3 and Cy5 attached to macromolecules measures intramolecular distances through time. *Proceedings of the National Academy of Sciences of the United States of America*, 102 (5):1419–1423, February 2005. doi: 10.1073/pnas.0409487102.

William Clifford. Introduction to FPGA acceleration. Tech-tip, Stemmer Imaging AG, 2011. URL ftp://ftp2.imaging.de/websites/documents/tech-tips/

en_GB-TechTip-%20Intoduction-to-FPGA-acceleration-20110211.pdf. accessed on 31.10.2011.

Catherine G. Galbraith and James A. Galbraith. Super-resolution microscopy at a glance. *Journal of Cell Science*, 124(10):1607–1611, 2011. doi: 10.1242/jcs.080085.

Frederik Grüll, Manfred Kirchgessner, Rainer Kaufmann, Michael Hausmann, and Udo Kebschull. Accelerating image analysis for localization microscopy with FPGAs. In *Field Programmable Logic and Applications, 2011. FPL 2011. International Conference on*, September 2011. doi: 10.1109/FPL.2011.11.

Mike Heilemann. Fluorescence microscopy beyond the diffraction limit. *Journal of Biotechnology*, 149(4):243 – 251, 2010. ISSN 0168-1656. doi: 10.1016/j.jbiotec.2010.03.012.

Mike Heilemann, Sebastian van de Linde, Mark Schüttpelz, Robert Kasper, Britta Seefeldt, Anindita Mukherjee, Philip Tinnefeld, and Markus Sauer. Subdiffraction-resolution fluorescence imaging with conventional fluorescent probes. *Angewandte Chemie International Edition*, 47(33):6172–6176, 2008. ISSN 1521-3773. doi: 10.1002/anie.200802376.

Ricardo Henriques, Mickael Lelek, Eugenio F Fornasiero, Flavia Valtorta, Christophe Zimmer, and Musa M Mhlanga. QuickPALM: 3D real-time photoactivation nanoscopy image processing in ImageJ. *Nat Meth*, 7(5):339–340, May 2010. ISSN 1548-7091. doi: 10.1038/nmeth0510-339.

Samuel T. Hess, Thanu P.K. Girirajan, and Michael D. Mason. Ultra-high resolution imaging by fluorescence photoactivation localization microscopy. *Biophysical Journal*, 91(11):4258 – 4272, 2006. ISSN 0006-3495. doi: 10.1529/biophysj.106.091116.

Bernd Jähne. *Digitale Bildverarbeitung / Digital image processing*. Springer, Berlin; Heidelberg [u.a.], 6. edition, 2005. ISBN 978-3-540-24999-3.

Sara A Jones, Sang-Hee Shim, Jiang He, and Xiaowei Zhuang. Fast, three-dimensional super-resolution imaging of live cells. *Nat Meth*, 8(6):499–505, June 2011. ISSN 1548-7091. doi: 10.1038/nmeth.1605.

Ullrich Köthe. Reliable low-level image analysis. Habilitation Thesis, December 2007. URL http://hci.iwr.uni-heidelberg.de/Staff/ukoethe/papers/habil-koethe.pdf.

Ullrich Köthe. The VIGRA computer vision library, 2011. URL http://hci.iwr.uni-heidelberg.de/vigra. accessed on 31.10.2011.

Tingwei Quan, Pengcheng Li, Fan Long, Shaoqun Zeng, Qingming Luo, Per Niklas Hedde, Gerd Ulrich Nienhaus, and Zhen-Li Huang. Ultra-fast, high-precision image analysis for localization-based super resolution microscopy. *Opt. Express*, 18(11):11867–11876, May 2010. doi: 10.1364/OE.18.011867.

John C. Russ. *The Image Processing Handbook*. Taylor & Francis, Boca Raton [u.a.], 6. edition, 2011. ISBN 978-1-4398-4045-0.

Michael J Rust, Mark Bates, and Xiaowei Zhuang. Sub-diffraction-limit imaging by stochastic optical reconstruction microscopy (storm). *Nat Meth*, 3(10):793–796, October 2006. ISSN 1548-7091. doi: 10.1038/nmeth929.

Silicon Software. Product catalog 2011, 2011a. URL http://www.silicon-software.de/download/archive/SiSo_ProductCatalog_2011.pdf. accessed on 24.10.2011.

Silicon Software. VisualApplets – a graphic-oriented tool, which dramatically simplifies the programming of image processing on FPGA hardware. Software and Documentation, 2011b. URL http://www.silicon-software.de/visualapplets.html. accessed on 2.11.2011.

Steve Wolter, Mark Schüttpelz, Marko Tscherepanow, Sebastian Van De Linde, Mike Heilemann, and Markus Sauer. Real-time computation of subdiffraction-resolution fluorescence images. *Journal of Microscopy*, 237(1):12–22, 2010. ISSN 1365-2818. doi: 10.1111/j.1365-2818.2009.03287.x.

Steve Wolter, Ulrike Endesfelder, Sebastian van de Linde, Mike Heilemann, and Markus Sauer. Measuring localization performance of super-resolution algorithms on very active samples. *Optics Express*, 19(8):7020–33, 2011. doi: 10.1364/OE.19.007020.

Xilinx. Spartan 3E datasheet, August 2009. URL http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf. accessed on 24.10.2011.

Xilinx. LogiCORE IP fast fourier transform v7.1. datasheet, March 2011. URL http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf. accessed on 2.11.2011.

Ian T. Young and Lucas J. van Vliet. Recursive implementation of the Gaussian filter. *Signal Processing*, 44(2):139 – 151, 1995. ISSN 0165-1684. doi: 10.1016/0165-1684(95)00020-E.

Chi-Li Yu, Jung-Sub Kim, Lanping Deng, Srinidhi Kestur, Vijaykrishnan Narayanan, and Chaitali Chakrabarti. FPGA architecture for 2D discrete fourier

transform based on 2D decomposition for large-sized data. *Journal of Signal Processing Systems*, 64:109–122, 2011. ISSN 1939-8018. doi: 10.1007/s11265-010-0500-y.

Erklärung:

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 8. November 2011 .....