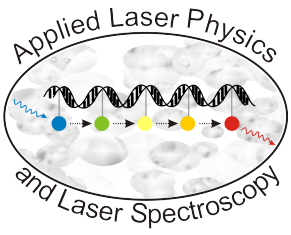
Universität Bielefeld



Diploma Thesis

An Accurate and Efficient Algorithm for Real-Time Localisation of Photoswitchable Fluorophores

Steve Wolter

Examiners: Prof. Dr. Franz Kummert
Prof. Dr. Markus Sauer
Supervisors: Dr. Mark Schüttpelz
Dr. Marko Tscherepanow

Submitted to the Faculty of Technology
Conducted at the Department of Physics
October 2008 to February 2009

Abstract

This work scrutinises the deconfinement problem of photoswitching microscopy. The task of deconfinement is locating the sources of fluorophoric emission with subpixel accuracy in large sequences of images, which are acquired by temporally confining photoswitching microscopy.

The preceding work on this task is summarised and formulated with a standardised terminology. Then, the novel approach of motivational fitting is introduced, which greatly reduces the number of nonlinear fit attempts that are needed. A complete algorithmic concept and an implementation are given for this approach, along with a visualisation based on weighted histogram equalisation.

The resulting implementation in C++ code is verified and used for extensive testing of the chosen algorithms and of the important alternatives. Major parameters are determined experimentally on both real and simulated data. Additionally, it is proven that the deconfinement problem can be solved well within the real-time domain for a wide range of acquisition speeds and parameter choices. This includes the published parameters for the major temporally confining photoswitching microscopy methods.

This real-time capability alleviates the computational burden of temporally confining photoswitching microscopy and can considerably aid wide-spread use of these methods.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

> I have seen the cultivated man craving for travel, and for success in life, pent up in the drudgery of London work, and yet keeping his spirit calm, and perhaps his morals all the more righteous, by spending over his microscope evenings which would probably have been gradually wasted at the theatre. — Charles Kingsley ([29, pg. 42])

1.1 Photoswitching microscopy overview

The optical microscope has been a core instrument of science for centuries. Beginning with the first microscopes built by Galilei [5] and used by early scientists such as Hooke, the microscope has helped revealing countless mysteries. For some time, it was thought that the resolution achievable by microscopic instruments would only be limited by the skill of microscope builders in choosing better and more precise lenses [34, pg. 8].

In 1873, Abbe discovered that the resolution of the standard optical microscope is limited by the laws of physics to roughly 200 nm for visible light. This discovery and the unbowed desire for images of smaller and smaller particles have motivated researchers to develop several methods which achieve higher resolution. Amongst these are prominent methods such as near-field microscopy, which brings the objective very close to the sample to reduce interference, the phase contrast method, which exploits phase information from the light wave, or confocal microscopy, which limits illumination to the focal area.

In the 1990's, researches formulated a fundamentally new way to circumvent Abbe's diffraction-imposed resolution limit: Instead of finding ways to reduce the apparent width of a point light source, they tried to differentiate between close light sources in other ways. The resulting methods, which are summarised under the term *photoswitching microscopy* [22] (or, by some authors, "far-field nanoscopy"), enhance resolution by isolating these close light sources. Once isolated, they can be localised with high precision – at least one order of magnitude better than the classical diffraction limit [53]. This isolation is termed technically as the *confinement*.

The confinement of light sources is achieved by using optically switchable fluor-

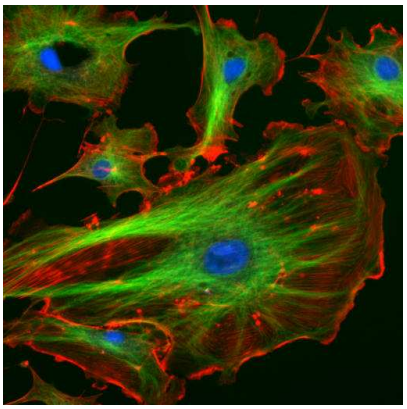


Figure 1.1: Microscope image of cells labelled with fluorescent dyes [41]

ophores[1], i.e. molecules that can be switched between a fluorescent "bright" state and a non-fluorescent "dark" state by irradiation with visible or ultra-violet light. Typically, an optical switch absorbs radiation at three wavelengths: One activation wavelength which switches it from the dark to the bright state, one deactivation wavelength which reverses the process and one readout wavelength[2] which excites fluorescent behaviour if the switch is in the bright state. This switching property can be employed to confine fluorophore emissions by activating only a subset of the total fluorophores.

Photoswitching microscopy has important applications in biological research. Already, "the three-dimensional organisation of distinct nuclear pore complex components has been mapped, and protein clusters on individual synaptic vesicles and in synaptic active zones" [33] have been resolved alongside with a multitude of other phenomenons. Its advantages over other resolution-enhancing methods are so striking that the Nature Methods journal made it its method of the year 2008 [11].

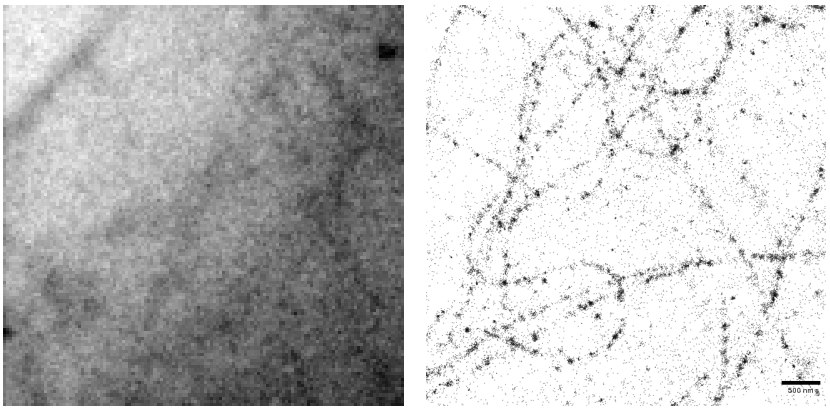
1.2 Temporal confinement and deconfinement

A major group of photoswitching microscopy methods consists of the temporally confining methods, which irradiate the whole sample with a mixture of activating and deactivating light. This effectually activates only a small subset of the total fluorophore population at any time, effectively confining the emissions of the activated fluorophores. If a long sequence of fluorescence images with short exposure times ("stack") is taken, it is likely that each fluorophore will be visible on at least one image in the stack: Fluorophores that were inseparable in space are now separated in time.

To arrive at a target image, the confinement must be reversed. For temporally

[1] A fluorophore is a molecule that absorbs a photon at a short wavelength and subsequently emits a photon of a longer wavelength (so, lower energy). Fluorophores are widely used in biological applications to make cell structures visible, as seen in Figure 1.1

[2] This readout wavelength is not necessarily distinct from the deactivation wavelength, depending on the method used.



(a) Signal averaged over 13 minutes. (b) Image enhanced by photoswitching microscopy

Figure 1.2: Comparison of conventional and photoswitching imaging. Both pictures are from the same source stack depicting fluorescing cell structures.

confining methods, this *deconfinement* process consists of extracting fluorophore position information from a stack S and combining this information into a single image that shows the fluorophores visible in S with enhanced resolution.

1.3 Motivation for and goal of present work

Traditionally, deconfinement had a high computational cost: The several thousand images taken along the time axis must each be searched for fluorophore emissions and the most likely source position of these emissions must be localised, taking hours of computing time even for small stacks. This computational cost is greatly hampering the practical applicability of temporally confining photoswitching microscopy since it delays experimental results and forces buffering of input data. It is rather desirable to have a program that can perform this computation in real time – that is, parallel to and faster than the single images of the stack are acquired.

Therefore, the goal of this work is to devise, develop and deploy a computer program that can reliably perform the deconfinement operation in real time and requires a minimum of user interaction and parameter choice.

1.4 Structure of this Thesis

This work is divided into the following chapters: The theoretical foundations in physics and computer science will be outlined in Chapters 2 and 3, respectively. Chapter 4 describes how I implemented the real-time computer program for deconfinement, while Chapter 5 gives the experimental procedures used for testing the program with real and simulated data. Chapter 6 is dedicated to an in-depth discussion of the results of these tests, while Chapter 7 gives a conclusion of these results and summarises the possibilities for further research.

Chapter 2

Theory of photoswitching microscopy

> Every theoretical explanation is
> a reduction of the intuition.
> – Peter Høeg ([23, pg. 46])

Software engineers emphasise, for good reason, the relationship between software development and the operational domain. Therefore, this chapter gives an overview over the theory of photoswitching microscopy, which is naturally divided into three parts: Firstly the field of microscopy, secondly the field of photoswitchable fluorophores, and thirdly the combination of both to photoswitching microscopy.

2.1 Microscopy

I assume the reader is vaguely familiar with the classical optical microscope, the details of which may be found in [5]. Therefore, I will limit this topic to an overview over the microscope types, the concept of a point spread function and the concept of resolving power.

2.1.1 Typology of microscopes

Microscopes are typified [34, pg. 343] into those that operate within one wavelength of the specimen (near-field) and those that operate from larger distances (far-field). Amongst the far-field microscopes, there are two major concepts: Either the whole specimen is illuminated continuously (wide-field) or the illumination is concentrated through the objective lens (confocal). The wide-field microscopy techniques that operate on an externally illuminated specimen are further divided into bright-field microscopy, which uses all light captured, and dark-field microscopy, which uses only diffracted light[1]. From the major types of near-field,

[1] Contrary to some wide-spread beliefs, dark-field microscopy resolution is limited by the same factors as other far-field techniques are. According to [37], the term $n \sin \theta$, which will be introduced as the numerical aperture in Section 2.1.3, for a dark-field microscope is smaller than 1.4. This is impressive, but by no means revolutionary.

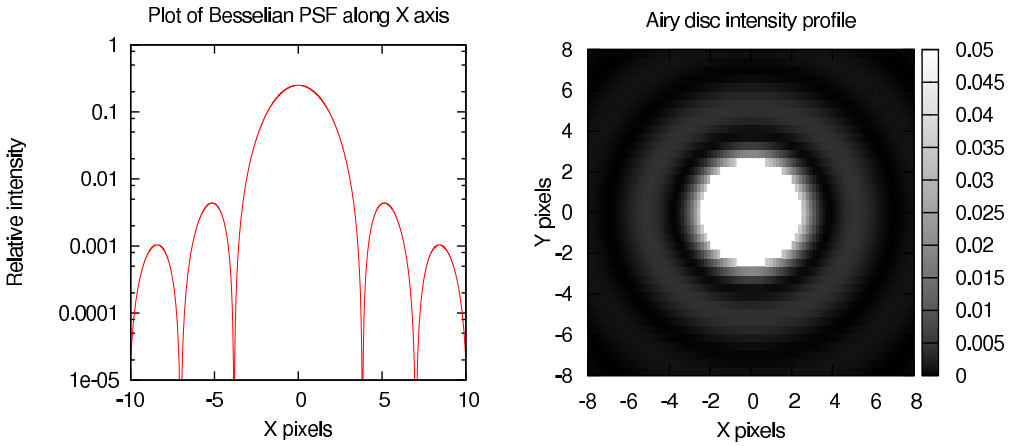


Figure 2.1: Besselian point spread function and Airy disc

confocal and wide-field microscopes, the wide-field microscope is the simplest, oldest, cheapest and most widely applicable[2]; both the near-field and the confocal microscope are more precise, but the near-field microscope is limited to the specimen surface [5] and the confocal microscope is limited by its very narrow field of view, requiring a linewise scanning of a specimen to acquire an image.

2.1.2 The point spread function

To define the notion of "precision", the concept of a point spread function (PSF) is used. A PSF of an imaging device is the intensity distribution of the image generated by a point light source viewed through this device.

The theoretical PSF of a wide-field microscope is called an Airy disc and given by [20]:

$$I(r) \propto \left(\frac{J_1(r)}{r} \right)^2, \tag{2.1}$$

with $I(r)$ denoting the intensity in distance r from the light source's position on the image plane and J_1 denotes the Bessel function of the first kind and first order. Figure 2.1 gives a graph of the resulting point spread function.

2.1.3 Resolving power

The problem of *resolving power* is defined by Smith, King and Wilkins [48] as "the requirement to distinguish light by two similar point objects separated by a small distance". If the optical instrument is sufficient to meet this requirement for a specimen, a clear image is attainable and the image or specimen is said to be *resolved*; if not, distinct point objects will be blurred into a single image entity and the image or specimen is said to be *unresolved*. The resolving power is the prime limitation of optical microscopy for very small entities like the details of cell organelles.

[2] The microscope you have seen in school was probably a wide-field microscope.

There are several criteria for resolving power, the most common of which is the one proposed by Lord Rayleigh [5, pg. 333]. Two light sources are said to be Rayleigh-resolved if the distance between the point sources is greater than the distance between the principal maximum of the PSF and its first minimum. When referring to some entity as "resolved", I will use this criterion.

For a wide-field microscope and an incoherently lighted object (such as a fluorescing sample), the resolving power Y is given by [5, pg. 419]:

$$Y \approx 0.61 \frac{\lambda_0}{n \sin \theta} \tag{2.2}$$

with Y denoting the minimum distance between *resolved* point light sources, λ_0 being the light's wavelength in vacuum, n denoting the refractive index of the lens and θ standing for the angles the marginal rays form with the optical axis. For typical parameters, this results in a resolution of $\lambda/2 \approx 0.2\,\mu\text{m}$ in lateral direction.

2.1.4 The Gaussian approximation to a Besselian PSF

In section 2.1.2 I have described how the PSF of a wide-field microscope is given by a Bessel function. While the Bessel function is, mathematically speaking, a well-behaved and well-assayed function, it is very hard to compute and it is very hard to find parameters for a Bessel function that fits a given intensity profile [28].

This problem has been successfully [44] addressed using a Gaussian model function that approximates the Besselian PSF, even when the intensity profile is distorted by capturing it with a pixelated device. This result has been used and verified by many researchers [53, 4] and is an important precondition for practical deconfinement.

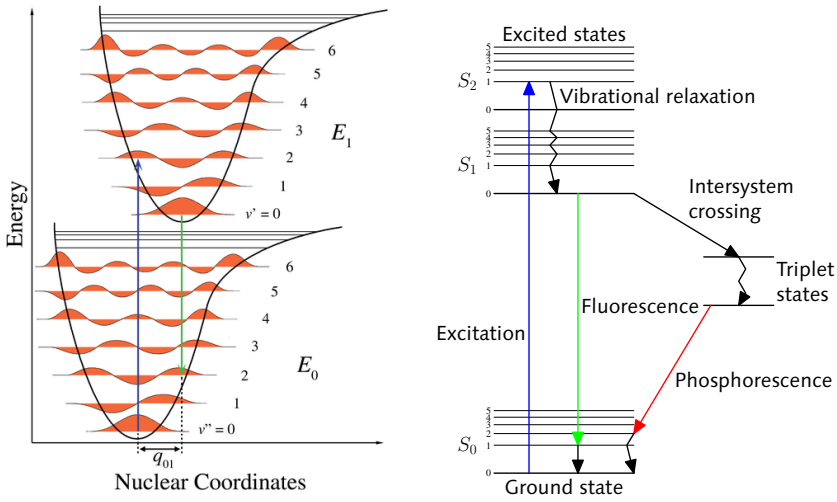
2.2 Photoswitchable fluorophores

2.2.1 Fluorescence

The phenomenon of fluorescence was discovered by Sir George G. Stokes [20] in the 19th century and has become a widely used tool in the time hence. It is characterised by the absorption of light followed within nanoseconds by the emittance of light of a longer wavelength.

Fluorescence has its roots in the mechanics of molecular excitation. When a system absorb or emit light, its quantum state changes from one energy level to another, with the difference between the energy levels given by the energy of the absorbed or emitted photon. Each energy level corresponds to at least one combination of quantum numbers for the system, which are chosen along the system's degrees of freedom. Molecules possess three major types of degrees of freedom [47]: vibration, rotation and electron excitation. These groups correspond to characteristic energies on different scales[3] and thus give rise to a complex and broad spectrum of energy levels.

[3] Typically, 1 to 10 eV for electron excitation, 10^{-3} eV for rotation and 10^{-1} eV for vibration.



(a) Franck-Condon diagram. Image from [50]. The horizontal lines show energy states for an electron in a quantum well, with the form of the standing wave indicated in orange. A transition from the ground state $(E_0, v'' = 0)$ to a matching excited state by fluorophore immission is shown, along with the following emission with lower wavelength.

(b) Typical Jablonski diagram. The horizontal lines show energy states for an electron. The triplet states and intersystem crossing are important for understanding fluorescent behaviour, but outside of the scope of this work.

Figure 2.2: Franck-Condon and Jablonski diagrams illustrating fluorescence. The arrows indicate state transitions, visible transitions in the colour of the necessary emission or immission, invisible transmissions in black.

Not all transitions from this spectrum are equally likely. The International Union of Pure and Applied Chemistry's (IUPAC) compendium of chemical technology [36] states that "the Franck-Condon principle is the approximation that an electronic transition is most likely to occur without changes in the positions of the nuclei in the molecular entity and its environment." Due to this principle, state transitions induced by photons often happen to be so-called *vibronic* transitions, which change both the vibration and the electronic excitation states of the affected molecule. Figure 2.2(a) shows a Franck-Condon diagram illustrating the likely transitions.

A ground-state molecule is thus likely to be excited both electronically and vibrationally by an incoming photon. The vibrational excitation is subsequently lost due to collisions with neighbouring molecules [47], resulting in a lower energy of the photon that is emitted during the electronic de-excitation. This is illustrated in Figure 2.2(a) by the gap q_{01} between the blue and the green transition.

This simple principle of vibronic excitation, vibrational relaxation by collision and electronic relaxation by light emission can result in fairly complex pathways in molecules with multiple likely transitions. A very useful tool in these cases is the Jablonski diagram, which displays the energy levels and transitions in a molecule. Figure 2.2(b) shows the Jablonski diagram for a typical fluorophore.

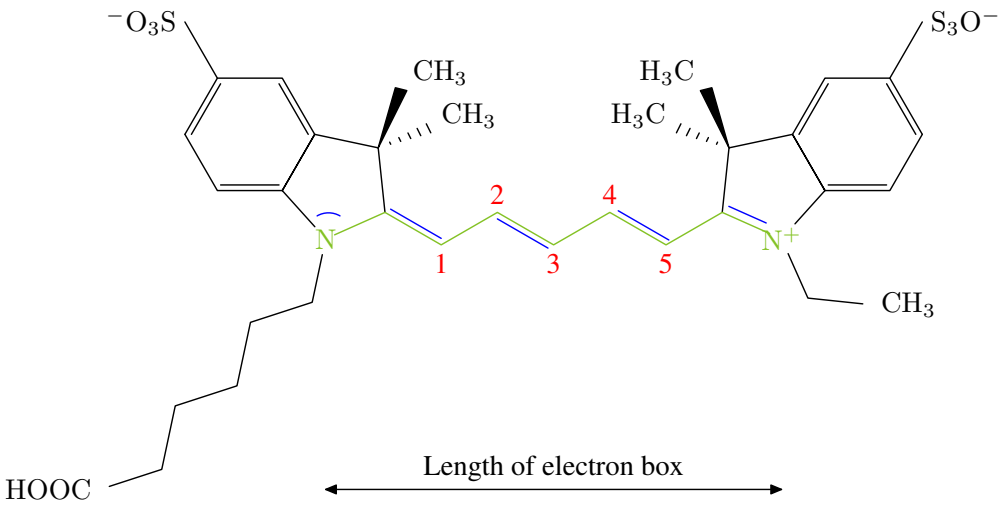


Figure 2.3: Box model for Cy5 molecule. The polymethine chain responsible for fluorescence is displayed in green along with its terminating nitrogen atoms. The blue lines display the free π electrons, and the red numbers show the 5 carbon atoms that give Cy5 its name.

2.2.2 Molecular Photoswitches

Sauer [45] defines a photoswitch as a molecule that "exhibits two stable and selectively addressable states, a fluorescent and a non-fluorescent, which can be conveyed into another in a reversible fashion upon irradiation with different wavelengths of light".

Photoswitches have been subject to large scientific interest since Irie *et al.* synthesised the first artificial photoswitch [25]. Researchers have identified several different photoswitchable molecules, amongst them the green fluorescent protein (GFP)-related Dronpa molecule [45] and the carbocyanine Cy5 [20], and have successfully invoked photoswitching behaviour in organic fluorophores such as Atto-Tec's ATTO 655 [56].

My work has primarily been conducted with images of Cy5-labelled structures, since these have been most readily available. The structure of this dye is sketched in Figure 2.3. In its fluorescent state, Cy5 shows an absorption maximum at 649 nm and an emission maximum at 670 nm [19]. This behaviour can be explained by assuming that the polymethine chain in the molecule centre (displayed in green) allows essentially free movement of the π electrons (displayed in blue) along its axis. The nitrogen atoms at its end form the walls of a one-dimensional box with a length of $L = kb$, where k is the number of chain bonds and b is the length of a chain bond, which can be approximated by the length of a carbon-carbon bond in benzene (139 nm). The energy levels of an electron in a box with length L are given by

$$E_n = \frac{n^2 h^2}{8 m_e L^2}, \tag{2.3}$$

with m_e being the mass of an electron and h being Planck's constant.

If there the number of atoms in the Cy5 chain bond is N, there are $N/2$ electrons

occupying the E_n states. Therefore, the highest occupied state is $n_1 = {}^N/_2$ and the lowest free state is $n_2 = {}^N/_2 + 1$. The smallest possible energy change for a transition is therefore

$$\Delta E = E_{n_2} - E_{n_1} = \frac{h^2(n_2^2 - n_1^2)}{8m_e L^2} = \frac{h^2(N+1)}{8m_e L^2} \tag{2.4}$$

and thus the longest excitation wavelength is

$$\lambda = \frac{c}{\Delta E} = \frac{8m_e L^2}{h^2(N+1)}\, c, \tag{2.5}$$

where c is the speed of light in vacuum.

This formula predicts an excitation wavelength of 579 nm, 70 nm from the measured value and a good guess for such a simple model.

Using radiation with 633 nm wavelength, Cy5 can be switched from its fluorescent state into a dark state showing no fluorescent behaviour and recovered from the dark state by irradiation at around 337 nm [19]. Widengren and Schwille suggested in [58] this behaviour is due to isomerisation and back-isomerisation of the Cy5 molecule.

2.3 Photoswitching microscopy

The photoswitches described in section 2.2.2 can be used to overcome the resolution limit seen in section 2.1.3. Suppose a sample with very fine structures is prepared with switchable fluorophores. If these fluorophores are all emitting photons at a given time and the sample is viewed under an optical microscope, the Airy disks generated by the fluorophores add up and generate a single region of very high signal, making distinction of individual fluorophores impossible under noisy conditions. Figure 2.4 illustrates this effect. This overlapping effect is due to the wave nature of light and principally unavoidable in microscopes with finite lens dimensions [5].

When using photoswitches as light source, this overlap can be avoided by deactivating most fluorophores so that only a small subset of the available fluorophores is measured, independently of the deactivated fluorophores. As seen in the introduction, these methods are called photoswitching microscopy methods.

The members of the first and older group of photoswitching microscopy methods, the spatially confining methods, activate fluorophores based on their locations. These methods use a beam of deactivating light with a donut-shaped profile, which is used to deactivate all fluorophores in a part of the sample except those right in the centre of the donut. If the centre is small enough, it will contain at most one fluorophore at any time, effectively confining the sample emission to the emission of this fluorophore. To measure the emission of the whole specimen, the donut-shaped beam is moved over it, the emission at each point is recorded and combined by a computer into a result image. There are several methods that use this principle, including stimulated emission depletion (STED), ground state depletion (GSD) and saturated pattern emission microscopy (SPEM) [22].

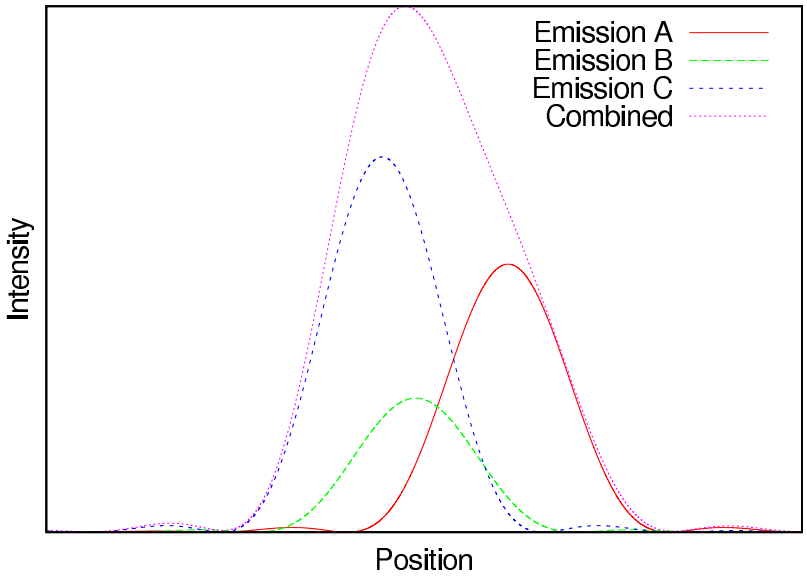


Figure 2.4: Illustration of the emission profile for unresolved light sources. If interference effects are neglected, the three arbitrarily chosen emissions in this image add up to the combined intensity profile. The emissions become unresolvable.

The second and newer group of photoswitching microscopy methods consists of the temporally confining methods. Those methods do not address the fluorophores in a sample by their spatial positions, but rather stochastically by irradiating the whole sample with weak or short-pulsed activating or deactivating light. The low intensity or duration of the irradiation causes only a small subset of the total fluorophore population to be activated, effectively isolating the few activated fluorophores. The exact position of an activated fluorophore can be deducted by fitting a model of the point spread function to the spatial intensity profile of the emission. These positions can be reassembled into a target image. Figure 2.5 shows a sketch of the process. Methods for temporally confining photoswitching microscopy include:

- photoactivated localisation microscopy (PALM)[1], which employs successive phases consisting of a short activating laser pulse and passive detection of the fluorophores activated by the pulse,
- stochastic optical reconstruction microscopy (STORM)[43], which employs pairs of cyanine dyes that exhibit switching behaviour and irradiates these at low activation and deactivation power to achieve stochastic switching, and
- direct stochastic optical reconstruction microscopy (dSTORM)[21], which greatly simplifies STORM by using a single photoswitchable dye instead of pairs.

Generally, the spatially confining methods have the advantage of large noise tolerance [22], but the necessary beam shaping and targeted readout necessitates elaborate optical setups. Also, the speed of acquisition is limited by the scanning

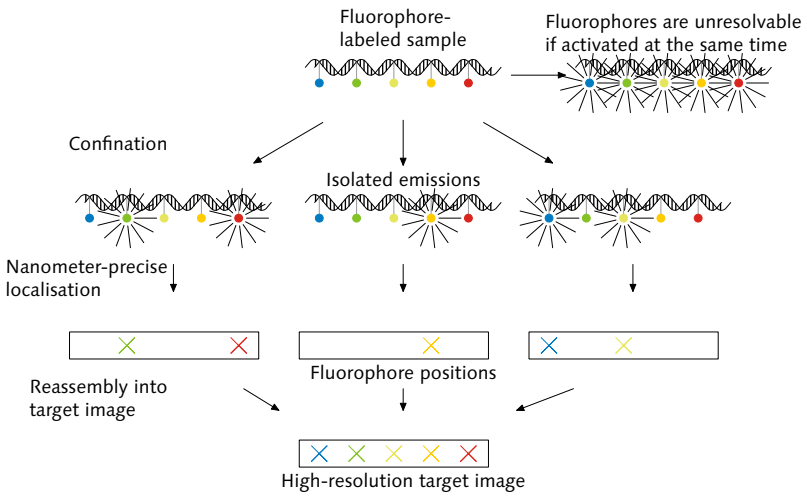


Figure 2.5: Principle of temporally confining photoswitching microscopy. From top to bottom, this figure shows the process of confinement and deconfinement. The deconfinement is shown in two steps here, localisation and reassembly. The columns show three different times during the acquisition, and the rays depict fluorophore activity.

process: The optical switches must be affected by activating light, recover from the dark state and emit a sufficient number of photons before the focus can be moved on. While this time can be reduced by parallelizing the process with multiple readout beams, such an approach introduces even more optical complexity. Additionally, fluorophores and biological samples are subject to high stress in the readout process: When the front of the donut, its centre and its back are moved over a fluorophore, it must be switched off, on and off again in very short time, therefore at a very high rate and with a very high excitation intensity.

In contrast, temporally confining photoswitching microscopy can be performed with standard wide-field microscopes [21]. Since these microscopes measure the complete specimen all of the time, all fluorescence emissions can be captured, requiring fewer switching cycles to achieve the same number of fluorophore localisations. However, temporally confining photoswitching microscopy methods do not provide immediate information about the location of active fluorophores and rely on computational methods to find these emissions in the acquired images, giving rise to the deconfinement problem introduced in the introduction.

Chapter 3

Algorithmic theory

The Feynman Problem-Solving Algorithm:
1: write down the problem
2: think very hard
3: write down the answer
– Murray Gell-Mann ([16, pg. xii])

Computer science offers two fields of research particularly useful to the deconfinement task: image processing and function minimization. While methods from the former field are used to process the input and output images on a pixel basis, the nonlinear parameter estimation from the second field provides the transition from the low-resolution stack images to the high-resolution target image. In addition, the notation and well-known algorithms for elementary statistics will be introduced.

There will occur some linear algebra in this chapter. I will denote vectors and matrices by bold symbols (like $\mathbf{x}$), using lowercase letters for vectors and uppercase for matrices. Specific elements of vectors will be denoted by one or two indices: $\mathbf{x}_n$ is the n-th element of the vector $\mathbf{x}$, and $\mathbf{X}_{jk}$ is the element in the j-th row and k-th column of the matrix $\mathbf{X}$.

3.1 Image processing

This section will give an introduction to digital images and will then introduce the two large classes of image operators, the kernel operators and the rank operators, along with their members important to the deconfinement task.

3.1.1 Images in the space and frequency domains

In the field of computer science images are treated as a multidimensional spatial intensity (or colour, if necessary) distribution. Therefore, a two-dimensional image f is a function of $\mathbb{R}^2 \to \mathbb{R}$ that defines the intensity of the image in every point of the $\mathbb{R}^2$ plane.

The infinite dimension and precision of such an image are, due to technical and physical limitations, impossible to acquire and to represent with current camera and memory technologies. Therefore, a finite and rectangular section of $\mathbb{R}^2$ is divided into equally sized rectangles and the values of the integrals of f over these rectangles are stored into a matrix $\mathbf{P}$. This matrix is the discrete version of an image f used in computational image processing, and a single element of it is called a picture voxel or, for short, *pixel*. I will denote the pixels of an image at the integer position (x, y) as $\mathbf{P}_{xy}$, where $0 \leq x < W$ and $0 \leq y < H$ if $\mathbf{P}$ is a $W \times H$ matrix. Pixels can take integer values between 0 and G; this maximum value G is called the *depth* of an image.

3.1.2 The frequency domain

An image that is defined by a function giving its intensity at each point in space is said to be in the *spatial domain*. There are alternate ways to specify an image, and one of those is the specification of the image-defining function f by its Fourier transform.[1] For the two-dimensional image function $f(x, y)$, the Fourier transform $F(u, v)$ is given by

$$F(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) \exp\left(-2\pi i(ux + vy)\right) dx\, dy \tag{3.1}$$

Almost all functions occuring in practice can be described completely by their Fourier transforms [42, pg. 167]. This implies that the function $f(x, y)$ carries no more information than $F(u, v)$; both are just different ways of representing the same information. Since the parameters (u, v) of the Fourier transform are the frequencies of the sine and cosine terms, the Fourier transform is commonly called the frequency representation of the image, or the image is said to be in the *frequency domain* if it is given by the terms $F(u, v)$.

Of course the same restrictions that limit the treatment of images to matrices apply to the Fourier transforms of images. Therefore, the infinite integral of equation 3.1 is in practise replaced by a finite sum and the Fourier transform of the $W \times H$ image $\mathbf{P}$ is given by $\mathbf{T}$ with

$$\mathbf{T}_{uv} = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} \mathbf{P}_{xy} \exp\left(-2\pi i(\frac{x}{W}u + \frac{y}{H}v)\right). \tag{3.2}$$

Smoothing in the frequency domain

Several image operations that have a smoothing effect on the image, such as those that will be given in the next sections, can be described very elegantly in frequency space. Smoothing operations reach the smoothing effect by reducing the differences between adjacent pixels, but largely keeping the differences between pixels that are far apart from each other. Therefore, the short-range (and thereby highly

[1] The Fourier transformation of a function f is the representation of f by an integral over sine and cosine terms. The weight for each infinitesimal part of this integral is given by the Fourier transform of f, which is usually denoted by F.

frequent) oscillations in the image are reduced and the long-range oscillations are kept. In signal processing terminology, such a filter is called a low-pass filter, and therefore smoothing filters are called low-pass filters in imaging terminology.

A low-pass filter can be useful to enhance the signal-to-noise ratio (SNR) of images. In the common case of stochastically independent noise and comparatively smooth signal, a suitably chosen low-pass filter will remove mostly noise [12] and thereby improve the SNR.

As seen in subsection 3.1.2, the frequency and the spatial domain are equivalent. Therefore, the low-pass filters in the frequency domain should be equivalent to some operation in the spatial domain. According to the *convolution theorem* [10], for filters working by point-wise multiplication in frequency space this equivalent operation is the *convolution*. The convolution of the image matrix $\mathbf{P}$ with some other matrix $\mathbf{W}$ is the scalar product of $\mathbf{P}$ with all possible translations of $\mathbf{W}$, with the result values of the scalar products forming the resulting image matrix. If the matrix $\mathbf{W}$ has few non-zero elements, the convolution can be computed efficiently and low-pass filtering can be achieved without transforming an image into the frequency domain. This class of operations is commonly known as kernel operations and introduced in the following section.

3.1.3 Kernel operators

Definition of kernel operators

A kernel operation [42] is an operation which replaces every pixel with a weighted sum of the pixels in its neighbourhood, with the weights depending on the position of the neighbour relative to the pixel that is to be replaced. The weights for this operation are usually combined into a matrix $\mathbf{W}$, which is called the kernel. If $\mathbf{P}$ denotes the source image for a kernel operation and $\mathbf{W}$ the kernel matrix of width $2m_x + 1$ and height $2m_y + 1$, the application of $\mathbf{W}$ to $\mathbf{P}$ will be defined as

$$K_{\mathbf{W}}(\mathbf{P})_{xy} = \sum_{i=-m_x}^{+m_x} \sum_{j=-m_y}^{+m_y} \mathbf{W}_{ij} \cdot \mathbf{P}_{x+i,y+j} \tag{3.3}$$

Naturally, the time complexity of applying a kernel operator of width m_x and height m_y to an image of width w and height h is $O(w\,h\,m_x\,m_y)$[2]. Especially for large kernel operators (such as those needed for strong smoothing), this is a forbiddingly high computational cost for the real-time domain.

Separable kernel operators

If a kernel operator can be written as the tensor product[3] $\otimes$ of two vectors, it is called *separable*. The distinct advantage of separable kernel operators is the replacability of the operator application by the application of the two producing

[2] The O notation gives the factors to which an algorithm's asymptotic runtime is proportional to. The asymptotic runtime is, informally, the runtime for very large values of all involved factors, thus only considering the largest power for each factor involved.

[3] The tensor product of two vectors $\mathbf{a}$ and $\mathbf{b}$ is by definition a matrix $\mathbf{A}$ with the height of $\mathbf{a}$ and the width of $\mathbf{b}$ and elements $\mathbf{A}_{ij} = \mathbf{a}_i \cdot \mathbf{b}_j$.

vectors. Thereby, computational complexity can be reduced from $O(w\,h\,m_x\,m_y)$ to $O(w\,h\,\max\{m_x, m_y\})$.

Kernel operators for noise reduction

Kernel operators are of great use for noise reduction since they can be used to implement the smoothing low-pass filters in the spatial domain, as I have explained in section 3.1.2.

There are three major candidates for a kernel-based low-pass filter: The *Spalttiefpass*[4], the binomial and the Gaussian low-pass filter.

A *Spalttiefpass* is a local averaging mask. It can be realised as a kernel operator whose kernel elements are all equal to a constant c. For example, a 3 by 5 *Spalttiefpass* is given by

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}. \tag{3.4}$$

It's name is derived from the similarity to looking at the local part of the image through an ideal slit. In effect, the operator averages the local neighbourhood of a pixel to produce the smoothed pixel value. However, the *Spalttiefpass* is somewhat unusual in image processing due to the strong blurring side-effect and the undesirable frequency domain effects [30, pg. 173][42, pg. 57].

On the computational side, the *Spalttiefpass* filter is unique in its property of not only being separable, but even having constant elements in the separated vectors. This fact permits implementation in $O(w\,h)$ time.

The Gaussian low-pass filter is a kernel operator whose matrix values approximate a two-dimensional Gaussian function. Gaussian low-pass kernels are implemented in three variants: as floating-point matrices, as integer matrices or as separable integer matrices. While the floating-point matrices allow the best approximation to the form of the Gaussian function, they carry the large computational cost of floating-point arithmetic. Therefore, Russ [42, pg. 57] uses integer matrices that are optimised to "approximate the smooth analytical curve of the Gaussian, while generally keeping the total of the weights smaller than some practical limit". The results are matrices such as

$$\begin{pmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 7 & 11 & 7 & 2 \\ 3 & 11 & 17 & 11 & 3 \\ 2 & 7 & 11 & 7 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{pmatrix}. \tag{3.5}$$

[4] Direct translation: "slit low-pass"

If some accuracy in this approximation is sacrificed, the matrices may be made separable, as in

$$\begin{pmatrix} 1 \\ 3 \\ 4 \\ 3 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 3 & 4 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 4 & 3 & 1 \\ 3 & 9 & 12 & 9 & 3 \\ 4 & 12 & 16 & 12 & 4 \\ 3 & 9 & 12 & 9 & 3 \\ 1 & 3 & 4 & 3 & 1 \end{pmatrix}. \tag{3.6}$$

The binomial low-pass filter is a separable kernel operator built from two vectors with vector elements from the binomic series. It is useful since it offers a quick way to produce approximations to Gaussian smoothing masks of a given pixel size. It was not used in this work since its application has the same computational complexity as a Gaussian, but the construction method can only construct a number of discrete standard deviations which do not necessarily resemble the PSF. Further details can be found in [30, pg. 173].

3.1.4 Rank operators

The class of rank operators is the second important class of elementary image transformations. Rank operators are not expressed algebraically, but rather by constructing a set of pixels, ordering this set and then taking a pixel with a given rank in that ordered set. Therefore, the basic operations of the class on images $\mathbf{F}$ and $\mathbf{G}$ are identical to the basic set operations: the union $\mathbf{F} \cup \mathbf{G}$, intersection $\mathbf{F} \cap \mathbf{G}$ and complementation $C(\mathbf{F})$ [49, pg. 26]. In grey scale images, these operations translate naturally into the point-wise maximum $\mathbf{F} \vee \mathbf{G}$, the point-wise minimum $\mathbf{F} \wedge \mathbf{G}$ and subtraction from maximum image $\bar{\mathbf{F}}$, respectively. It is important to note that these operations are made on a pixel-by-pixel basis, not on the image as a whole.

Structuring elements

Central to many rank operations is the concept of a structuring element (SE), denoted by the letter B. A SE is a set of translation vectors that is used to establish a neighbourhood relation on the pixels. Therefore, a structuring element is also said to *connect* two pixel positions $\mathbf{a}$ and $\mathbf{b}$ if it contains a vector $\mathbf{v}$ such that $\mathbf{a} + \mathbf{v} = \mathbf{b}$. The translation of all pixels in an image $\mathbf{F}$ by some vector $\mathbf{a}$ is called the *translation* and denoted by $\mathbf{F_a}$. A structuring element can be denoted using a matrix $\mathbf{M}$ and a vector $\mathbf{o}$ (called the origin), where the matrix element M_{ij} is 1 if the vector $\begin{pmatrix} i & j \end{pmatrix} - o$ is in the structuring element. Figure 3.1 shows some structuring elements.

The applications of set operations are manifold and even a rudimentary overview over their uses would exceed the boundaries of this work. Therefore, only the definition is given for most operators here.

1	1	1
1	①	1
1	1	1

(a) SE for 8-neighbourhood

0	1	0
1	①	1
0	1	0

(b) SE for 4-neighbourhood

0	1	0
0	①	1
0	0	0

(c) SE for corner detection

Figure 3.1: Examples for structuring elements (SE) in matrix representation. The circle indicates the origin.

Dilation and Erosion

The dilation $\delta_B(\mathbf{F})$ is defined as the maximum of the pixels connected by a SE B:

$$\delta_B(\mathbf{F}) = \bigvee_{b \in B} \mathbf{F}_b \tag{3.7}$$

The dilation enlarges all features in the image in the directions given by the SE. It can be used to join segments of broken features.

On the other hand, the erosion $\epsilon_B(\mathbf{F})$ is defined as the corresponding minimum:

$$\epsilon_B(\mathbf{F}) = \bigwedge_{b \in B} \mathbf{F}_b \tag{3.8}$$

The erosion shrinks all features in the image, eliminating completely structures unable to cover the SE. This property can be useful to detect elements of a given size and form by using a carefully chosen structuring element.

The structuring elements for dilation and erosion are often square elements of a specified width. For these elements, a shorthand notation is useful:

$$\delta^{(m)}(\mathbf{F}) = \delta_B(\mathbf{F}) \text{ with } B = \{\mathbf{b} | \forall i : -m \leq \mathbf{b}_i \leq m\} \tag{3.9}$$

For example, the SE for $\delta^{(1)}$ is the 8-neighbourhood seen in Figure 3.1(a): the set of vectors that connect a pixel with the 8 pixels neighbouring it horizontally, vertically and diagonally.

Geodesic dilation and erosion

Geodesic transformations are transformations that are not defined by an input image and a SE, but rather on two distinct input images, one called the *marker* $\mathbf{F}$ and one called the *mask* $\mathbf{G}$. While this seems to be a very restricted class of operations at first glance, an efficient choice of marker and mask images yields a great number of interesting transformations.

The geodesic dilation of size 1 is denoted as $\delta^{(1)}_{\mathbf{G}}(\mathbf{F})$ and "defined as the pointwise minimum between the mask image and the elementary dilation $\delta^{(1)}$ of the marker image"[49, pg. 184]:

$$\delta^{(1)}_{\mathbf{G}}(\mathbf{F}) = \delta^{(1)}(\mathbf{F}) \wedge \mathbf{G}. \tag{3.10}$$

Dilations of larger sizes are produced by repeated applications of the elementary dilation:

$$\delta^{(i)}_{\mathbf{G}}(\mathbf{F}) = \delta^{(1)}[\delta^{(i-1)}_{\mathbf{G}}(\mathbf{F})] \wedge \mathbf{G}. \tag{3.11}$$

The geodesic erosion is, *mutatis mutandis*, defined identically.

Opening and closing

While an erosion is a useful tool for erasing all elements smaller than a given SE B from an image, it has the side-effect of reducing the size of all structures in the image. However, if an dilation with a SE $\hat{B}$ equal to the reflection of B across the coordinate origin is performed on the eroded image, structures that survived the erosion are grown by approximately the same amount they were reduced in erosion. This combined process – erosion followed by a dilation with the reflected structuring element – is called an *opening* and is a widely used operator in image processing. If a dilation is performed first and then an erosion, the transformation is called a closing.

Median smoothing

The median operator $M_B(\mathbf{F})$ is defined as the median of all pixels connected by the SE B:

$$M_B(\mathbf{F}) = \text{med}\{\mathbf{F}_b | b \in B\}, \tag{3.12}$$

where the med operator denotes the median operator, which chooses the smallest value greater than half of the elements of a set. Effectively, this operator smooths the image by removing extreme pixel values. It is valued in image processing for it's edge-preserving properties [42, pg. 66].

Morphological reconstruction

The morphological reconstruction by dilation is defined as the iteration of the geodesic dilation until the marker image converges:[5]

$$R^{\delta}_{\mathbf{G}}(\mathbf{F}) = \delta^{(i)}_{\mathbf{G}}(\mathbf{F}) \text{ with } \delta^{(i)}_{\mathbf{G}}(\mathbf{F}) = \delta^{(i+1)}_{\mathbf{G}}(\mathbf{F}). \tag{3.13}$$

The morphological reconstruction by erosion $R^{\epsilon}_{\mathbf{G}}(\mathbf{F})$ is defined analogously.

Fillhole transformation

One specific application of morphological reconstruction by erosion is the fillhole transformation, which is used to fill all local minimums from an image that are not connected to the image border [49, pg. 208]. Mathematically, the fillhole transformation FILL($\mathbf{F}$) is defined as

$$\text{FILL}(\mathbf{F}) = R^{\epsilon}_{\mathbf{F}}(\mathbf{F}^m) \text{ with } \mathbf{F}^m_{xy} = \begin{cases} \mathbf{F}_{xy} & \text{if } (x,y) \text{ lies on the border of } \mathbf{F} \\ G & \text{otherwise} \end{cases} \tag{3.14}$$

The fillhole transformation can be understood by considering how the mostly maximum marker image is slowly eroded from the sides. An example is shown in Figure 3.2.

[5] Convergence is guaranteed by the nature of the transformation [49, pg. 190].

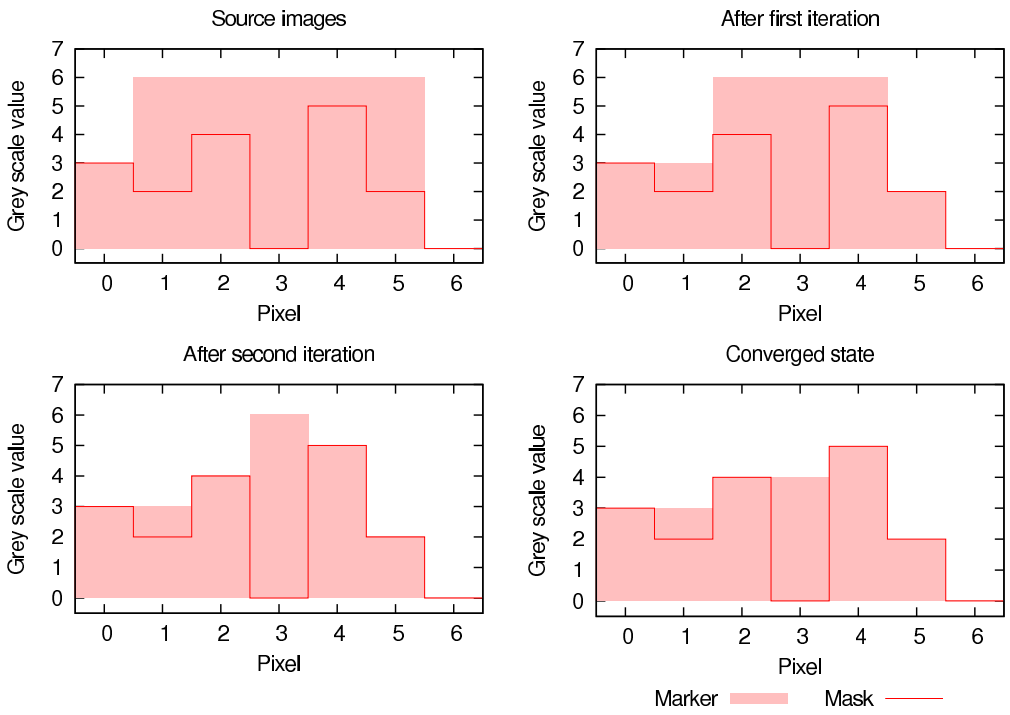


Figure 3.2: Example for 1-dimensional fillhole transformation. The mask image is the input image to the fillhole transformation, the marker in the converged state the output. Each iteration is an application of geodesic erosion.

3.1.5 Non-maximum suppression

The term of non-maximum suppression (NMS) is a difficult term in computer science since different algorithms solving different problems have been named with it. Canny used the term NMS for a part of his edge detection algorithm [8], where he applies NMS to an edge-thinning operation. I will neither use this algorithm nor this meaning of NMS. Neubeck [38] and others use the term for a morphological operator: this operator selects all pixels strictly greater than all other pixels connected to them by a given SE. For clarification, I will call this operation *exclusive* NMS. If the "strictly greater" condition for exclusive NMS is relaxed to greater or equal, an operation I will call *inclusive* NMS is obtained. The difference is subtle, but has far-reaching implications on very homogeneous images; while exclusive NMS will find no maximums in a region of constant intensity, inclusive NMS will declare all pixels as maximums.

Neubeck and van Gool [38] have scrutinised fast implementations for exclusive NMS with rectangular structuring elements of width $2m_x + 1$ and height $2m_y + 1$. According to their results, a fast and simple implementation of NMS utilises the fact that maximum pixels must be maximum pixels within a rectangle of width $m_x + 1$ and height $m_y + 1$ regardless of the alignment of that rectangle relative to the pixel. Therefore, NMS computation can be sped up by dividing the image into rectangles of width $m_x + 1$ and $m_y + 1$, computing the candidates in these blocks and comparing only these candidates against the pixels in neighbouring blocks.

Algorithm 1 Non-maximum suppression

Require: Image of dimensions WxH
Require: Block size m
  for all (w_b, h_b) with $n \cdot (m+1) = w_b$ and $m \cdot (m+1) = h_b$ **do**
    Find maximum pixel p in the block $w_b \leq x \leq w_b + m$ and $h_b \leq y \leq h_b + m$, if any
    if p exists and is maximal in its $\delta^{(m)}$ neighbourhood **then**
      Add p to maximum list
    end if
  end for

The resulting algorithm is shown as algorithm 1; the Neubeck algorithm runs in $O(W\,H)$ time for a $W \times H$ image, regardless of mask size.

If the algorithm is adapted for inclusive NMS, however, multiple maximums can occur within a structuring element. This worsens the worst case complexity (which occurs for regions with constant pixel values) to $O(W\,H\,m_x\,m_y)$ for a $W \times H$ image, which is identical to the naïve implementation.

The inclusive NMS operation can also be implemented by comparing the source image with its dilation by the SE. Basically, any pixel that is left unchanged by dilation is not connected to any pixel greater than it and therefore an inclusive maximum. Since the dilation is a semi-group operation, it can be implemented in $O(W\,H)$ according to an algorithm by Gil and Werman [15].

3.1.6 Intensity transfer function transformations

An intensity transfer function is a function that maps intensity values onto intensity values, such as those functions shown in Figure 3.3. An intensity transfer function T can be used to define an image transformation that replaces every pixel value v by the result of $T(v)$. Formally, the transformation $T(\mathbf{P})$ of the image $\mathbf{P}$ by the intensity transfer function $T(v)$ is therefore defined as

$$(T(\mathbf{P}))_{xy} = T(\mathbf{P}_{xy}). \tag{3.15}$$

Histogram function

Intensity transfer functions can be used to define an important tool for image analysis: the intensity histogram function $h(\mathbf{P}, v)$, which gives the number of pixels in an image $\mathbf{P}$ whose intensity is equal to a value v. The histogram function is often supplemented with the cumulative histogram function

$$h_C(\mathbf{P}, v) = \sum_{i=0}^{v} h(\mathbf{P}, i), \tag{3.16}$$

which gives the number of pixels with a intensity less or equal to v.

The histogram function h can be used to analyse how well an image uses the available range of intensity. A range of intensity values with continuously high

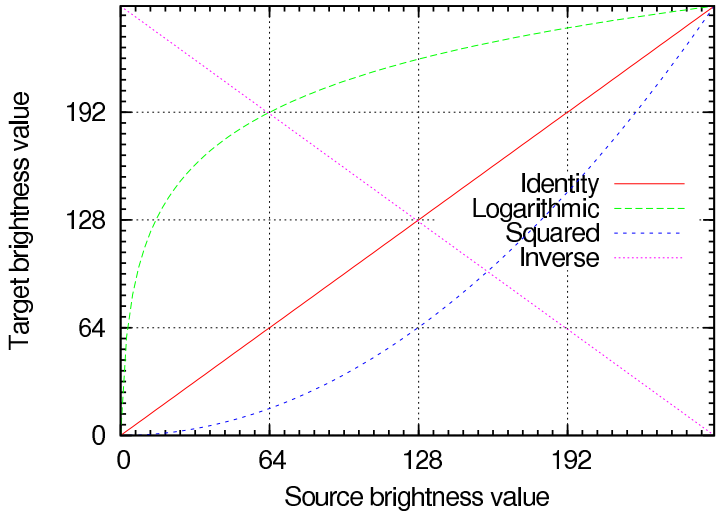


Figure 3.3: Typical transfer functions. The functions transform the abscissa grey values to the values on the ordinate. The transfer functions shown here are static and used in some fields of image processing; for example, the logarithmic transfer function might be used to add contrast to the very dark regions of an image.

values of h can be interpreted as an overused section of the histogram. Such an overused section represents many pixels in the image that are hard to discern from each other because of their small difference in intensity. On the other hand, a range of values in the histogram with continuously low values of h indicates a range of little-used intensity values and thus a waste of dynamic range in the image pixels.

Contrast stretching

When one or both ends of the dynamic range of an image are not used, the contrast is suboptimal. By scaling the intensity values, a higher contrast can be achieved without loss of information. Suppose the low end and the high end of the used histogram range for an image $\mathbf{P}$ are given by the quantities I_l and I_h, respectively, with

$$I_l(\mathbf{P}) = \max_{0 \le v \le G} \{v | \forall w < v : h(\mathbf{P}, v) = 0\} \tag{3.17}$$

$$I_h(\mathbf{P}) = \max_{0 \le v \le G} \{v | \forall w > v : h(\mathbf{P}, v) = 0\} \tag{3.18}$$

Then, the contrast stretching operation is defined as the application of the transfer function T_S with

$$T_S(\mathbf{P}, v) = \left\lfloor \frac{v - I_l(\mathbf{P})}{I_h(\mathbf{P}) - I_l(\mathbf{P})} G \right\rfloor . \tag{3.19}$$

Histogram equalisation

When contrast is to be emphasised over the absolute ratios between pixel intensity levels, the underutilisation of histogram values can be addressed by using the technique of histogram equalization: the intensity values of the image are reassigned

in such a way that the peaks in the intensity distribution are spread out over the whole intensity scale and intensities with small histogram values are merged. If the histogram equalisation is realized as an intensity transfer function, large peaks in the histogram will still be large peaks in the histogram of the equalised image, but the histogram equalisation spaces high peaks far apart from each other in the target histogram to optimise contrast.

The intensity transfer function for histogram equalisation T_{HE} replaces each intensity value by the quotient of the the number of pixels with intensity values smaller or equal than it to the total number of pixels. Formally, the intensity transfer function for histogram equalisation of an image $\mathbf{P}$ is given by

$$T_{HE}(\mathbf{P}, v) = \left\lfloor \frac{h_C(\mathbf{P}, v)}{h_C(\mathbf{P}, G)} \, G \right\rfloor, \tag{3.20}$$

where $\lfloor \cdot \rfloor$ is the Iverson floor function giving the largest integer not larger than its argument. Please note that this makes the output of the histogram equalisation dependent on the source image in two ways: Firstly, the source image is used to define the transfer function, and secondly, it is used as a source image transformed by that transfer function.

An example for a histogram equalisation with good results can be seen in Figure 3.4. You will notice that the overall contrast was enhanced at the cost of new and false visual similarities; the river to the right is now inseparable from the sandbank around it. The similarity is due to the merging of different intensity levels with few pixels to a single, stronger level. It is a danger endemic to histogram equalisation and addressed by the weighted method that is following.

Weighted histogram equalisation

The simple histogram equalisation ensures an approximate equidistribution of values over the histogram. In other terms, the same "share" of the intensity "space" is given to each pixel. However, it is not always desirable to act in such radical fashion when the few pixels in the little-used intensity levels convey important information. In these cases, a more conservative approach is needed: The well-used intensity levels should be expanded, but not too much, ideally according to a parameter that weighs between equalisation and conservation.

This can be accomplished by introducing a weight parameter r into the histogram equalisation formula [30, pg. 130][27]. Let

$$h_w(\mathbf{P}, v, r) = (h(\mathbf{P}, v))^r \tag{3.21}$$

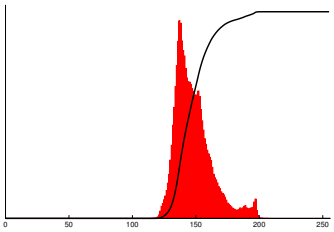
be the weighted histogram function and

$$h_{C,w}(\mathbf{P}, v, r) = \sum_{i=0}^{v} h_w(\mathbf{P}, i, r) \tag{3.22}$$

the cumulative weighted histogram function. Then, the weighted histogram equalisation transfer function T_{WHE} is defined by

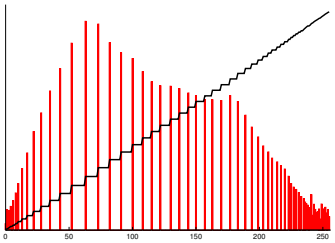
$$T_{WHE}(\mathbf{P}, v, r) = \left\lfloor \frac{h_{C,w}(\mathbf{P}, v, r)}{h_{C,w}(\mathbf{P}, G, r)} \, G \right\rfloor \tag{3.23}$$

(a) Faded Image



(b) Histogram and cumulated hist.

(c) Image after histogram equalisation



(d) Equalised histogram and cumulated hist.

Figure 3.4: Example histogram equalisation. Image from [17]. The histogram is displayed in red, the cumulated histogram in black. The equalised histogram points out how the strong peaks in the histogram are spaced out to enhance the contrast.

and the weighted histogram equalisation operation heq_w as

$$\text{heq}_w(\mathbf{P}, r)_{xy} = T_{WHE}(\mathbf{P}, \mathbf{P}_{xy}, r). \tag{3.24}$$

To understand the effects of different choices for r, a different perspective on the results of the weighted histogram function $h_w(\mathbf{P}, v, r)$ is useful. It is obvious that the term $h_{C,w}(\mathbf{P}, G)$ normalises the sum of all h_w terms to 1. This allows to interpret these terms as *weights* that the source intensity levels v have in the target intensity space: High values of h_w will result in a sharp rise in $h_{C,w}$ and thereby in a sharp rise in the target intensity, thus allocating a greater part of the possible target intensity levels to a source intensity level. Adjusting the weight parameter r changes the balance between the source intensity levels with many pixels and those with few pixels.

A choice of $r = 0$ results in identical weights for all intensity levels regardless of pixel count; thus, no re-weighting occurs at all and the identity transform is performed. Obviously, choosing $r = 1$ yields the simple histogram equalisation, where the relative weight of a intensity level is equal to its pixel count. If $0 < r < 1$ is chosen, the weight of intensity level rises with its pixel count, but sublinearly. This is the weaker form of histogram equalisation desired. For the effects of $r > 1$ or $r < 0$, please refer to [27].

3.2 Levenberg-Marquardt parameter estimation

One of the most fruitful approaches in science is the reduction of observed data by the use of a mathematical "model" for the data: an abstract and mathematical description of a complex system. Many of the successful models offer parameters that must be adjusted to accurately explain and predict the observations for a specific situation. While the ideal choice of these parameters is often difficult and a highly heuristic process, there exist several methods that provide numerical support to the researcher. Amongst these, the *Levenberg-Marquardt method* has "become the standard of nonlinear least-squares routines" [39, pg. 683] and is therefore used in this work.

Generally, the nonlinear least-squares (NLLS) parameter estimation problem is formulated by:

1. A *parameter vector* $\mathbf{a}$, which contains the parameter choices for the model function.
2. A *model function* $y(x;\mathbf{a})$, which computes the model prediction for an input x given the parameters $\mathbf{a}$.
3. A vector of measured data points $\mathbf{D}$ with $\mathbf{D}_i = (\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\sigma}_i)$, where the $\mathbf{x}_i$ are the measurement positions, the $\mathbf{y}_i$ the measurements and $\boldsymbol{\sigma}_i$ the measurement standard deviations.

Given these definitions, the aim of NLLS parameter estimation is to find a vector $\mathbf{a}$ such that the merit function

$$\chi^2(\mathbf{a}) = \sum_i \left[\frac{\mathbf{y}_i - y(\mathbf{x}_i;\mathbf{a})}{\boldsymbol{\sigma}_i}\right]^2 \tag{3.25}$$

is minimal [39, pg. 682]. To achieve this goal, its *gradient vector*

$$\boldsymbol{\beta} = -\frac{1}{2}\nabla\chi^2(\mathbf{a}) \qquad \Leftrightarrow \qquad \boldsymbol{\beta}_j = -\frac{\partial\chi^2(\mathbf{a})}{2\partial\mathbf{a}_j} \tag{3.26}$$

and its *curvature matrix*

$$\mathbf{C} = \frac{\nabla\otimes\nabla}{2}\chi^2(\mathbf{a}) \qquad \Leftrightarrow \qquad \mathbf{C}_{jk} = \frac{\partial^2\chi^2(\mathbf{a})}{2\partial\mathbf{a}_j\partial\mathbf{a}_k} \tag{3.27}$$

can be utilised to iteratively improve an initial guess for the minimum. In the following, this initial guess for the minimum of $\chi^2(\mathbf{a})$ will be denoted with $\mathbf{a}_0$, the current guess with $\mathbf{a}_i$, the improved guess based on $\mathbf{a}_i$ with $\mathbf{a}_{i+1}$ and the step from the current to the improved guess with $\boldsymbol{\delta}_i = \mathbf{a}_{i+1} - \mathbf{a}_i$.

3.2.1 Minimum search by quadratic approximation

Let $\mathbf{a}_i$ be a guess for the minimum of $\chi^2(\mathbf{a})$. If it is sufficiently close to a minimum, the merit function will be approximated by a quadratic form [39, pg. 681]

$$\chi^2(\mathbf{a}) \approx Q_i(\mathbf{a}) = \gamma - 2\boldsymbol{\beta}_i\left(\mathbf{a}-\mathbf{a}_i\right) + \left(\mathbf{a}-\mathbf{a}_i\right)\mathbf{C}_i\left(\mathbf{a}-\mathbf{a}_i\right), \tag{3.28}$$

where β_i and $\mathbf{C}_i$ denote gradient and curvature matrix at $\mathbf{a}_i$. The minimum of the quadratic approximation is the best guess for the minimum of $\chi^2(\mathbf{a})$ and therefore used as $\mathbf{a}_{i+1}$. Since the derivative at an extremum is 0, the minimum of this quadratic approximation is given by

$$\begin{aligned} & 0 &=& \nabla Q_i(\mathbf{a}_{i+1}) \\ & &=& -2\beta_i + 2\mathbf{C}_i \cdot (\mathbf{a}_{i+1} - \mathbf{a}_i) \\ \Leftrightarrow \quad & \boldsymbol{\delta}_i &=& \mathbf{C}_i^{-1}\beta_i \\ \Leftrightarrow \quad & \mathbf{C}_i\boldsymbol{\delta}_i &=& \beta_i. \end{aligned} \tag{3.29}$$

Equation 3.29 is a set of linear equations and can be solved easily since the gradient vector and curvature matrix can be derived from the known form of the model function.

3.2.2 Minimum search by gradient descent

Alternatively to the quadratic approximation, one can iteratively follow the steepest descent of the gradient. While gradient descent has many variations, in the context of the Levenberg-Marquardt method the iteration step $\boldsymbol{\delta}_i$ is proportional to the gradient steepness. The proportionality factor is the inverse of a unit-less number λ, which is to be chosen by the experimenter. Therefore, the gradient descent gives an iteration step of

$$\boldsymbol{\delta}_i = \lambda^{-1}\beta_i. \tag{3.30}$$

However, this equation must be flawed, since the units of measurement[6] of the k-th element in this vector is given by

$$\left[(\boldsymbol{\delta}_i)_k\right] = \left[\left(\lambda^{-1}\beta_i\right)_k\right] = \left[(\mathbf{a}_i)_k\right]^{-1} \neq \left[(\mathbf{a}_i)_k\right]. \tag{3.31}$$

Obviously, since in $\boldsymbol{\delta}_i$ should be added to $\mathbf{a}_i$, they must possess the same units.

This problem may be addressed [39, pg. 683] by including a factor with units of $\left[(\mathbf{a}_i)_k\right]^2$ into equation 3.30; this condition is fulfilled by the inverses of the diagonal elements of $\mathbf{C}$. Therefore, the k-th element of the shift vector for steepest descent is given by:

$$(\boldsymbol{\delta}_i)_k = \frac{(\beta_i)_k}{\lambda\,(\mathbf{C_i})_{kk}} \qquad \Leftrightarrow \qquad \lambda\,(\mathbf{C_i})_{kk}\,(\boldsymbol{\delta}_i)_k = (\beta_i)_k \tag{3.32}$$

3.2.3 Critical comparison of these methods

It is to be expected that both the quadratic approximation and the steepest gradient descent methods will find a minimum of χ^2 if supplied with a good initial guess. The two methods differ chiefly in three respects: flexibility, speed and stability.

Regarding flexibility, the steepest descent method is evidently preferable since the size of the step can be adjusted by a parameter. In an alternative interpretation,

[6] The units of measurement denote the physical units of the quantities dealt with, such as a metre when fitting the position of an object. The $[\cdot]$ notation is used to denote the units used. The $[A]$ operation returns the unit the quantity A is measured in; for example, if t denotes the time, $[t]$ might be seconds (s) or years (a).

this parameter gives the extent of our trust in the local gradient. For well-behaved model functions, this trust may be large and the steps towards the minimum may be large; if, however, the model function changes rapidly, the steps must be chosen small. On the other hand, the steepest descent method is very reliant on a good choice of this parameter and can run into numerous problems when this parameter happens to be wrong, such as slow convergence, oscillations around a minimum and blockage by saddle points.

Regarding speed and stability, the quadratic approximation has a clear advantage. Normally, the computationally costly part of NLLS approximation is the recomputation of χ^2 and its derivatives after each step; minimisation of step number is therefore the prime speed-regarding characteristic of a NLLS scheme. The quadratic approximation can be expected to use fewer steps because of its utilisation of second-order derivatives. In addition, the quadratic approximation needs no parameters and is therefore not prone to wrong parameter choices.

Algorithm 2 Levenberg-Marquardt algorithm

Require: Model function χ^2 with partial derivatives
Require: Initial parameter guess $\mathbf{a}$
Compute $\chi^2(\mathbf{a})$
Pick a small initial value for λ ([39] suggests 10^{-3})
while Approximation is not satisfactory **do**
  Solve linear equation system (3.34) for $\boldsymbol{\delta}$
  Compute $\chi^2(\mathbf{a}+\boldsymbol{\delta})$
  if $\chi^2(\mathbf{a}+\boldsymbol{\delta}) < \chi^2(\mathbf{a})$ **then**
    $\mathbf{a} \leftarrow \mathbf{a}+\boldsymbol{\delta}$
    $\lambda \leftarrow \lambda/10$
  else
    $\lambda \leftarrow 10 \cdot \lambda$
  end if
end while

3.2.4 The Levenberg-Marquardt scheme

The two schemes of quadratic approximation and steepest gradient descent both have unique advantages that complement each other. Levenberg and Marquardt realised that these advantages can both be utilised at the same time when using a mixture of both schemes. To achieve this, the equations 3.29 and 3.32 are combined by forming the new matrix $\mathbf{C}_i'$ with

$$\left(\mathbf{C}_i'\right)_{jk} = \left(\mathbf{C}_i\right)_{jk} \cdot \begin{cases} 1+\lambda & \text{if } j=k \\ 1 & \text{otherwise} \end{cases} \tag{3.33}$$

and result in

$$\mathbf{C}_i' \boldsymbol{\delta}_i = \boldsymbol{\beta}_i. \tag{3.34}$$

The form of $\mathbf{C}'$ depends on the choice of λ; obviously, if λ is large, $\mathbf{C}'$ is dominated by the diagonal entries and its effects closely resemble a steepest descent method.

For small λ, the method resembles the quadratic approximation. According to these similarities, λ should be lowered when the function seems well-behaved and the expected convergence is encountered (indicating usefulness of the quadratic approximation) and should be raised otherwise.

The recommended Levenberg-Marquardt recipe [39, pg. 684] is shown as algorithm 2.

3.3 Elementary statistics

Suppose a number of observations N for some arbitrary quantity has been made, and let $\mathbf{x}_n$ be the outcome of the n-th observation. I will call this vector of all observations the *sample*. Then, the *mean* or *arithmetic average* $\bar{\mathbf{x}}$ of the sample is defined as [6]

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_n \mathbf{x}_n. \tag{3.35}$$

The mean characterises the sample by giving its typical value, not regarding variations. To characterise the magnitude of these variations from the mean, there are three useful quantities which are derived from each other. First, the sum of squared deviations M_2 is defined as $M_2(\mathbf{x}) = \sum_n (\mathbf{x}_n - \bar{\mathbf{x}})^2$. To make this quantity independent of the number of samples, the sample variance $\mathrm{Var}(\mathbf{x})$ is used [6] with

$$\mathrm{Var}(\mathbf{x}) = \frac{M_2(\mathbf{x})}{N-1} = \frac{1}{N-1}\sum_n (\mathbf{x}_n - \bar{\mathbf{x}})^2. \tag{3.36}$$

The sample variance can be normalised to the units of $\mathbf{x}$ by taking the square root; this quantity is called the standard deviation (SD), denoted by $\sigma(\mathbf{x})$ and given by

$$\sigma(\mathbf{x}) = \sqrt{\mathrm{Var}(\mathbf{x})} \tag{3.37}$$

3.3.1 Incremental computation of mean and variance

West *et al.* [32] have developed a recursive computation scheme to quickly compute the mean and the variance of a sample provided that mean and variance of a subset are already known. Suppose that a sample $\mathbf{x}$ with N observations and known mean $\bar{\mathbf{x}}$ and sum of squared deviations $M_2(\mathbf{x})$ is given and should be extended by an observation o to the sample $\mathbf{y}$. The mean $\bar{\mathbf{y}}$ of the extended sample can be computed by

$$\bar{\mathbf{y}} = \frac{1}{N+1}\left(o + \sum_n \mathbf{x}_n\right) = \frac{o + N\cdot\bar{\mathbf{x}}}{N+1} = \frac{o-\bar{\mathbf{x}}}{N+1} + \bar{\mathbf{x}}, \tag{3.38}$$

and the sum of squared deviations M_2 can be computed by

$$M_2(\mathbf{y}) = (o-\bar{\mathbf{y}})^2 + \sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{y}})^2 = M_2(\mathbf{x}) + (o-\bar{\mathbf{y}})(o-\bar{\mathbf{x}}). \tag{3.39}$$

The proof for equation 3.39 can be found in the appendix on page B.

3.3.2 Confidence intervals

Suppose all observations in a sample are stochastically independent and are normally distributed. It is easy to compute the statistical mean for these observations, as I have shown in the previous subsections. Nonetheless, it is not safe to draw conclusions regarding the *stochastic* mean from these observations, that is, the mean value of the normal distribution underlying these observations.

What can be concluded from the sample is an interval that is likely to contain the stochastic mean. This interval is called the confidence interval and can, according to [6], be computed for a sample $\mathbf{x}$ of size n by the expression

$$\overline{\mathbf{x}} \pm t_{\alpha/2;n-1} \frac{\sigma(\mathbf{x})}{\sqrt{n}}. \tag{3.40}$$

α denotes the probability that the interval does not contain the true mean. In other words, if 100 samples are drawn and confidence intervals are computed from them, $100 \cdot \alpha$ of these confidence intervals will not contain the stochastic mean. The values of $t_{\alpha/2;n-1}$ arise as percentile points of the t distribution with $n-1$ degrees of freedom [6, pg. 116].

3.3.3 Classification errors

A classificator is a function $\mathbf{G}(\mathbf{x})$ which maps vectors $\mathbf{x}$ from a space of patterns $\mathbb{C}$ into a finite and discrete space of symbols Ω. The classical example of a classificator is Cinderella's flock of pigeons, which discerned between lentil and ash particles (patterns) by sorting them into bowl or crop (two symbols).

For a given symbol c, a pattern $\mathbf{x}$ and a classificator $\mathbf{G}$, statisticians distinguish two kinds of errors in a classification process: If $\mathbf{G}(\mathbf{x}) = c$ even though $\mathbf{x}$ should not be classified as c, there is a type I error, also termed false positive (FP). If $\mathbf{G}(\mathbf{x}) \neq c$ even though $\mathbf{x}$ should be classified as c, a type II error, also termed false negative (FN), is present. For a good positive or negative classification, I will use the terms true positive (TP) and true negative (TN), respectively.

Chapter 4

Implementation of a deconfinement program

> What I cannot create, I do not understand.
>
> – Richard Feynman ([18])

The operations compiled in Chapter 3 outline the basic building blocks for deconfinement, but they lack the glue of an encompassing algorithm. Since this overall algorithm is too large to be proven theoretically and since the deconfinement is a pressing computational task in contemporary physics, implementing the aforementioned algorithms in computer-executable code was made a major part of this work. In this chapter, I will elaborate about the sources for as well as the methods and results of this implementation.

The implementation project was named `rapidSTORM`, which is an acronym for the `r`apid yet `a`ccurate `p`rogram `i`mplementing the `d`irect `st`ochastical `o`ptical `r`econstruction `m`icroscopy. For ease of reference, I will use this acronym in the following.

This chapter will be divided into four sections: Firstly, I will summarise the state of the art in deconfinement. In the specification, I will exhibit the targets and the means for `rapidSTORM` implementation along with the global deconfinement algorithm that will be used and extended. The third section will contain information about the subalgorithms needed for the deconfinement. The fourth section will be dedicated to the parallelisation and streaming of the data transfer for localisations.

4.1 State of the Art

Several working groups have published pieces of their deconfinement schemes, but there is little comprehensive information.

The most extensive published work known to me and directly related to deconfinement is the U.S. patent 20080182336[59]. In this document, Zhuang *et al.* dedicated several paragraphs to their deconfinement algorithm. Unfortunately,

Figure 4.1: Typical dSTORM source image. This is image 34 of the stack that will be labeled as specimen 1. The fluorescent spots show different intensities and forms; note the blurry residual in the upper right corner, probably some out-of-focus fluorophore, and the strange structure in the lower right corner. The distance between pixels in the object plane was estimated to 71 nm.

no indication was given of either the run-time characteristics or of the false positive rate achieved with the implementation. It is extremely unlikely that Zhuang *et al.* achieved real-time performance since their algorithm performs the computationally expensive task of fitting the Gaussian error function to the fluorescent spots.

While aiming at fluorophore tracing rather than super-resolution, Thomann has done significant research in the issue in his doctoral thesis [52]. His suggested algorithm for fluorophore detection and localisation is very sophisticated and seems to yield excellent results; however, real-time performance is probably lost in the preprocessing stage [52, pg. 54] due to a complex spot detection algorithm using local grey value curvatures and a dynamic threshold based on histogram curvature.

I have had the opportunity to have access to prior work in this field by Schüttpelz, who implemented deconfinement in a MATLAB script [46]. This script is unpublished, but included in this print as Appendix E. It did not achieve real-time performance, but rather required several hours of computation time for input stacks acquired in 13 minutes.

Further prior work concerning deconfinement is mentioned in some articles concerning different super-resolution techniques. Amongst these are articles from Bates *et al.* [2] and Betzig *et al.* [3] which exhibit interesting ideas in deconfinement, but do not elaborate them due to the physical focus of these papers.

The other scientific research groups working on photoswitching microscopy did not include much useful information regarding the deconfinement process into

their recent articles [22, and research referenced therein].

Some useful input can be gained from the research in the field of spot array imaging [7], which faces basically the same problem of finding and judging point light sources. However, spot array imaging is a task significantly different from deconfinement because of different goals; while spot array imaging aims at classification of the spots found on a regular lattice and has no need of a precise localisation, deconfinement needs to detect the exact position of the fluorescent spots.

Last but not least, the scientific field of deconvolution treats the same issues as deconfinement does. However, even Jansson stresses in his related book [26, pg. 30] that "fitting methods", such as the one we are about to implement, have "great value, where applicable."[1] Since the value of fitting methods in deconfinement has been demonstrated by Thompson [53] and others, I have decided to trust in this authority and not explored the ideas stemming from deconvolution in this work.

4.2 Specification

4.2.1 Terminology for deconfinement

To my knowledge, none of the relevant papers have established a clear naming scheme for the deconfinement process. Given the importance of the topic, I will propose such a scheme here.

The process of deconfinement operates on a *source stack* (the sequence of fluorescence images received from the camera), which consists of *source images*. The deconfinement process has the goal of creating a super-resoluted *target image* where, ideally, every visible fluorophore in the sample will be shown at its likely position relative to the other fluorophores. It is important to note that this target images displays the positions of the fluorophore emissions, not their brightnesses or other characteristics.

A group of relatively[2] bright pixels is called a *spot*. For this work, a spot can and will simply be represented by a vector giving its centre. A spot that is likely to be a fluorophore emission is called a *candidate spot* (or, in the following, simply candidate).[3] Spots that were really generated by fluorophores are called *true spots*, those generated by noise *false spots*. The *quality* of a spot is the relative potential of a spot to be a true spot: a spot with higher quality is more likely to be true than one with a lower quality, but the quality is not required to be a probability for spot truth.

If a Gaussian intensity profile is fitted to a spot, the resulting set of parameters is called a *fit*. If a fit is judged by some set of criteria, it can come out to be good (if it is thought to belong to a true spot) or bad (otherwise). Good fits are also called *localisations*; this term also denotes the fluorophore position given in the parameter of a good fit. Figure 4.2 illustrates these terms.

[1] Fitting methods are, in this context, "applicable" if a model is known, which is the case in deconfinement.

[2] Relatively might be relative towards the surrounding area or towards the whole image.

[3] Thomann calls these "potential spots".

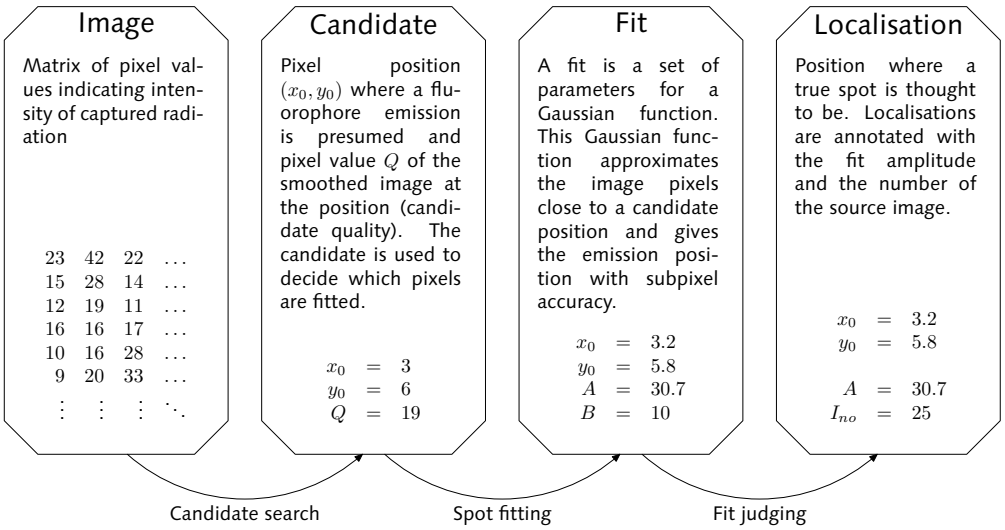


Figure 4.2: Illustration of the terminology used for deconfinement. Each box shows an interdemiate step in the deconfination process. The paramaters *A* and *B* for the fit denote the amplitude of the Gaussian model and the estimated background signal.

The standard deviation(s) of a Gaussian intensity profile that approximates the PSF will simply be called the standard deviation σ in this chapter. I will assume a correlation of 0 between the X and the Y axis; this leaves two independent values σ_x and σ_y. This simplification is not necessarily true, but close enough to the real data encountered in direct stochastic optical reconstruction microscopy (dSTORM) stacks. It was made to simplify the computation of the mask sizes that will be introduced in this chapter.

4.2.2 The deconfinement algorithm

The basic approach for deconfinement (or part of that approach) has been hinted at by Thompson, Zhuang *et al.* and Schüttpelz, but never been formalised. To achieve comparability of approaches, I deem it important to catch up on this issue and present my formalisation of their algorithm in Algorithm 3.

While this algorithm illustrates the basic idea of a realistic deconfinement scheme – namely, filtering the source image for promising candidates and then judging these spots –, it relies on the existence of an absolute criterion for the identification of candidates. This criterion is hard to find, as you will see in section 6.2.1; it is easier to compare two candidates to see which one is more likely to be a true spot. This is exemplified in the existing methods for candidate inspection: Thompson uses all pixels exceeding a threshold and Schüttpelz uses all pixels that additionally are H-maximums[4] of a given height. Comparison of candidates in this schemes simply means comparing the pixel value at the centre of the spot, while the choice of a threshold must be done by hand. Hence, I propose to improve the

[4]H-maximums are local maximums that fulfill a height condition. For details, see [49] or equation 4.1.

Algorithm 3 Basic deconfinement algorithm

Require: Source stack S
Ensure: Output is a target image for S
1: Create empty list of localisations t
2: **for all** source images **I** in S **do**
3: Find set of spot candidates C in **I**
4: **for all** spots s in C **do**
5: Fit PSF to s resulting in fit $\mathbf{a}_s$
6: Evaluate likelihood of $\mathbf{a}_s$ describing to a true spot
7: **if** $\mathbf{a}_s$ likely describes a true spot **then**
8: Insert localisation $\mathbf{a}_s$ into t
9: **end if**
10: **end for**
11: **end for**
12: Insert dot for each element in t into new image **D**
13: **return D**

deconfinement algorithm by

- choosing a low threshold for candidateness,
- sorting the found candidates by decreasing quality and
- fitting the candidates in this list sequentially until
- a sequential number of fits equal to a motivation parameter M_0 fails.

If the truth of a spot corresponds well to its quality, this method will find all true spots.

Furthermore, the result list t filled in lines 1 to 11 is far more flexible than a result image and can be evaluated in wholly different ways – for example, histograms of localisation counts along an arbitrary axis or time-resolved traces of spot localisations. Hence, I propose to let the deconfinement algorithm return the localisation list and defer the image assembly into a later step.

The new algorithm is formalised as Algorithm 4.

4.2.3 Secondary implementation goals

The primary goal of this work is, naturally, to implement Algorithm 4. However, I have identified several secondary objectives.

Firstly, the implementation should be easily extensible and maintainable. The super-resolution microscopy techniques are still in their infancy, and further insights and resulting computational adjustments are to be expected. A well-defined API will greatly facilitate extension, and a well-defined internal structure will facilitate refactoring.

Secondly, the implementation must be portable and possess a graphical user interface. Computers running Microsoft Windows are as commonplace in the lab

Algorithm 4 Motivationally thresholded deconfinement algorithm

Require: Source stack S
Require: Spot standard deviation σ
Ensure: Output is a list of localisations in S

```
Create empty list of localisations t
for all source images I in S do
    Construct list of candidates C in I
    Sort C by decreasing quality
    M ← M_0
    while spots remain in C and M > 0 do
        Fit PSF to first element in C resulting in fit a_s
        Evaluate likelihood of a_s describing a true spot
        if a_s likely describes a true spot then
            Insert localisation a_s into t
            M ← M_0
        else
            M ← M − 1
        end if
        Remove first element from C
    end while
end for
return t
```

environment as personnel inexperienced with command line interfaces is. An easily ported and used implementation will facilitate broad use of the `rapidSTORM` system.

Thirdly, the `rapidSTORM` implementation should be as fast as possible while not sacrificing accuracy. The huge amounts of data, both present and planned for the foreseeable future, processed in `rapidSTORM` applications justifies considerable work on implementation speed.

Fourthly, on-line acquisition of image data directly from the camera is necessary and useful. By this mean, the experimental procedure can be simplified greatly and the buffering of acquired data on storage media can be avoided.

Fifthly, the `rapidSTORM` program should make good use of the current and probably future trend towards multi-core processors. Therefore, good multithreaded performance is required.

4.2.4 Choice of implementation language

While there are many programming languages available which are capable of solving the deconfinement problem, maintainability and extensibility dictate that a widely used language should be used. In computational physics, the languages commonly used include FORTRAN, C, C++, Java, MATLAB and LabView. I have decided to use C++ for the `rapidSTORM` project; in the following, I will give a comparative analysis of the other programming languages with respect to the deconfinement problem.

The core competence of FORTRAN is its set of extremely efficient basic linear algebra subprograms. However, the operations used in `rapidSTORM` make little use of these subprograms, and in addition, the GNU Scientific Library[13] supplies a very fast linear algebra library for C/C++. Thus, this project can not utilise FORTRAN's strength; therefore, I decided that C/C++'s better library and toolchain support makes it the better choice.

Pure C is compiled to very fast code, has excellent library support for numerical and scientific applications and is amongst the languages with the best development tool support. However, it does not support object-oriented design, aggravating both extensible and maintainable design. Using C++ could alleviate these problems while keeping C's advantages.

MATLAB and LabView are interpreted languages widely used in the natural sciences. While they allow rapid development, their interpreters lack the advanced features of the more developed compilers used for C, C++ or Java such as type checking. This severely hampers stability; thus, I decided against using these languages.

Last but not least, Java must be considered for a project of this size. On the one hand, Java's ability to generate intermediate and machine-independent byte code is of little use for the `rapidSTORM` project since the distance between its development and deployment sites is negligible. On the other hand, byte code interpretation carries a high computational cost. This cost motivated me to decide against using Java in the core `rapidSTORM` code, even though it is used for the graphical user interface.

For these reasons, I chose to implement `rapidSTORM` in C++.

4.2.5 Supporting libraries

The implementation of the `rapidSTORM` is a software engineering task, and the prime difficulty in software engineering is complexity. The staple tool used to address this difficulty is modularisation: The division of a large programs into small and easily testable units.

- A general-purpose, cross-platform image processing and viewing library. Amongst the many choices available, I have decided to use the `CImg` library by David Tschumperlé. This C++ template library has the distinct advantage of being easily adapted to all data types, allowing the dSTORM process to save memory by using pixels of 2 byte size and still use the full power of an imaging library.

- The GNU Scientific Library (`GSL`) [13] that provides a range of functions useful for scientific research. It is used in `rapidSTORM` for statistical (mean, variance), stochastic (random number generation) and data fitting purposes.

- A basic multithreading library. The `pthread` library is the established solution for this problem.

  To achieve exception safety, a C++ interface to the `pthread` library is desirable. While the GNU Common C++ library offers this functionality, it's

mutual exclusion locking proved to be faulty in the Microsoft Windows implementation; thus, I reimplemented the public interface of the GNU Common C++'s threading module using the `pthread` library.

- A configuration parameter library. Such a library should offer management and end-user-friendly display of program input parameters such as an input file, an amplitude threshold or the choice of the A/D converter to use on the camera. Kevin Pulo's `simparm` library [40] provides such functions, but was in an early alpha phase at the beginning of the `rapidSTORM` project. The improved and tested version of the library is packaged with the `rapidSTORM` code and documented in the appendix.

- A library to read the SIF image format. The software that comes with the Andor Technology cameras used for dSTORM uses this format for storing source stacks. While Andor Systems thankfully provided code to read some versions of the format, it had no defined user interface, proved to be prone to buffer overflows and was incompatible with newer versions of the file format. For this reason, I extended this code into the `readsif` library.

- A high-level wrapper around the camera driver provided by Andor Technology. This module is called AndorCamera and provides a `simparm`-compatible configuration interface to Andor cameras and an automated image acquisition.

  The Andor camera module, while scientifically trivial, has been one of the most challenging aspects of the `rapidSTORM` system. The problems in its implementation have been caused by particularly bad incompatibilities in the software development kit provided by Andor Technologies for its scientific cameras. While the details of these bugs are not inside the scope of this work, their magnitude must be stated to explain the copious amount of code and work in the Andor camera module. This magnitude is probably exemplified best by the unorthodox combination of calling convention and name wrangling used by the functions in the Andor software development kit, which causes them to overwrite the calling function's memory and to depend on a pre-initialised stack space when called by code compiled by the `mingw` compiler.

- A library for fast non-linear fitting of an exponential function to data points on a 2-dimensional lattice. The GNU Scientific Library provides an implementation of the Levenberg-Marquardt algorithm for non-linear fitting; thus, the task of the library is to provide a concise C++ interface and a fitting function for the GSL implementation. This library is called `fit++`.

- The `CImgBuffer` library used for combining and configuring the different image acquisition methods, for buffering real-time acquisitions and for coordinating concurrent access. This library combines the `readsif` and `AndorCamera` libraries into a common interface.

Table 4.1 shows the versions of the used libraries.

Library	Linux	Windows
CImg	128	129
GSL	1.8	1.8
pthread	NTPL 2.7	pthreads-w32 2.8.0
simparm	0.02	0.02
readsif	1.0	1.0
AndorCamera	1.0	1.0

Table 4.1: Versions of libraries used in rapidSTORM v1.0

While the CImg, GSL and pthread libraries were usable out of the box, I would like to stress that simparm and readsif were in alpha state and the interesting part of the GNU Common C++ version was incompatible to MS Windows so that a copious amount of work was necessary to debug and integrate these libraries. This work is not documented further, but the resulting code can be found in the main rapidSTORM distribution and my share of the code can be determined by comparing these libraries with the original code, both of which are packaged with this thesis. To give a rough estimate of it, roughly 7200 lines of C and C++ code were involved, most of which needed change or were added. I have implemented the CImgBuffer, AndorCamera and libfit++ libraries from scratch, totalling more than 4600 lines of C and C++ code.

4.3 The rapidSTORM engine

The rapidSTORM engine is the module that implements Algorithm 4 and converts a source stack to a set of localisations. Therefore it is the by far most important and computation-intensive part of the rapidSTORM project – typical source stacks for dSTORM are often in excess of 10 gigabytes[5] and must be processed in a matter of minutes to stay real-time. This fact makes it crucial to design the data structures and control flow in a very efficient and overhead-minimising way.

The global structure of the rapidSTORM engine is oriented at Algorithm 4. We see four basic tasks in this algorithm:

1. Line 3, the *candidate search*, reduces the source image to a set of candidates.
2. Line 4 sorts these candidates. While sorting itself is well-studied in computer science, the sorting algorithm can be optimised for this specific case.
3. Line 7 is called the *spot fitting stage*, which reduces a spot to a fit.
4. Line 8 is the *fit judging stage*, which decides whether the spot for a given fit was true or not.
5. The other lines are simple control code. They can be implemented in C++ code almost on a line-by-line basis.

[5] Even though the storage of a complete source stack can be avoided by on-line computation, the sheer amount of data is nonetheless a problem.

Algorithm 5 Spot candidate search

Require: Source image **I**
Require: Suitable data structure C for candidate storage and sorting
1: Smooth **I** with a *Spalttiefpass* of size $2m_s\sigma + 1$ to obtain $\mathbf{I}_S$
2: Find local maximums of $\mathbf{I}_S$ by using Neubeck's NMSalgorithm (see 3.1.5) with multiple maximums of the same height allowed and filter size $2m_s\sigma + 1$
3: **for all** local maximums m of $\mathbf{I}_S$ **do**
4: Insert the candidate (s_m, p_m) into C with s_m being the grey scale value of the maximum and p_m being its position
5: **end for**
6: **return** C

The fit judging can be implemented by a simple thresholding of the fit's amplitude parameter since this value corresponds directly to the suspected number of photons in the fit and false positives should decrease exponentially with it. This leaves items 1-3 for inspection; in the following, I shall propose subalgorithms for these.

4.3.1 Candidate search

The candidate search algorithm has the task of reducing a source image of 10^4 to 10^6 pixels to a small number of spot candidates which should contain all true spots of the image. The central challenge for this stage is the severe restriction of its run-time performance. If the candidate search for a typical image of 256x256 pixels is to be performed in 1 ms, a processor clocked at 2 GHz will have only 300 cycles available for each pixel.

It is obvious that careful algorithm design is needed to perform this task in adequate time. However, the existing work has focused little on this aspect. Schüttpelz used the morphological operator

$$HMAX_h(f) = R_f^{\delta}(f - h) \tag{4.1}$$

followed by thresholding all image pixels and fitting once for each connected region, while Thompson thresholded the unmodified image. Both of these methods have the disadvantage of being very sensitive to single strong pixels, which are a common product of the noise encountered in deconfinement. Only Thomann has given the subject further consideration in [52, pg. 59]: he smoothes the image with a Gaussian low-pass filter and considers the local maximums of the smoothed image. The quality of a maximum in Thomann's scheme is given by

$$\bar{\mathbf{I}} \cdot \det(\mathbf{H}), \tag{4.2}$$

where $\bar{\mathbf{I}}$ denotes the average intensity of a window of large width (5 times the PSF's standard deviation) around the maximum and **H** denotes the local Hessian matrix. Maximum pixels are considered candidates if their quality exceeds a threshold chosen dynamically from the cumulative quality histogram. According to him, this method is "very sensitive" to spot truth, but I assessed it to be computationally very

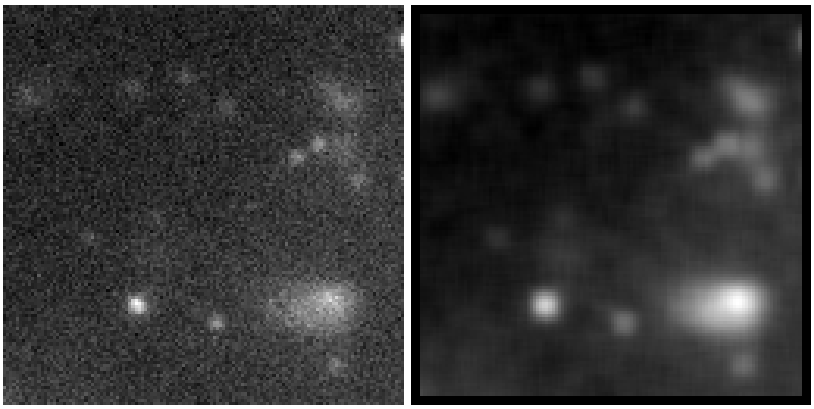


Figure 4.3: Typical dSTORM image smoothed with *Spalttiefpass*. Note that both images have been contrast-stretched independently; the right image has a lower dynamic range. Note how the fluorescent spots in the top left corner became more prominent and how noise maximums vanished mostly, but also how the position of the bottom left spot was blurred.

expensive because of the computation of a Hessian matrix and its determinant. Unfortunately, the scope of my work did not allow to check this assessment.

To find a method less complex than Thomann's, but retaining some of its sensitivity, I have to stress the importance of variance reduction. As you will see in Chapter 6, the primary problem of the candidate search is not to find the spots – in a noiseless environment, they could easily be found by searching local maximums –, but rather to suppress the noise that obscures these spots. For this purpose the *Spalttiefpass* is, according to Klette and Zamperoni[30], superior to the other smoothing filters, especially to the Gaussian filter used by Thomann.

Therefore, I propose to simplify Thomann's candidate search

- by dropping the Hessian matrix,
- by merging the averaging used to compute $\bar{\mathbf{I}}$ into the averaging used to smooth the image and
- by replacing the Gaussian low-pass filter with a *Spalttiefpass* filter.

Every local maximum pixel is considered a candidate with a quality equal to the maximum's intensity. In the implementation, I used Neubeck's fast NMS algorithm introduced on page 19, but modified it slightly to allow inclusive NMS. The resulting algorithm is formalised as Algorithm 5.

This algorithm sets the mask sizes for the *Spalttiefpass* and the NMS proportional to the PSF SD. This choice is motivated by the need to include all strong spot pixels, but as few noise pixels as possible. I therefore deem that the correct choice of a filter size should roughly be equal to the spot size and thus proportional to the PSF SD. Formally, this consideration translates into a kernel with constant elements and a rectangular SE, both with a width of $2m_s\sigma_x + 1$ and a height of $2m_s\sigma_y + 1$, where m_s is a constant and will be determined experimentally.

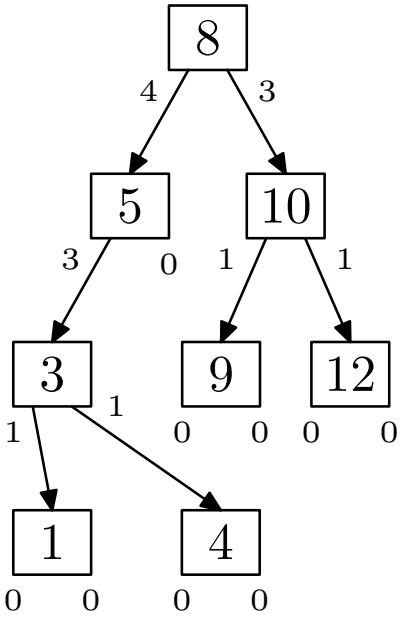


Figure 4.4: Example for a size-limited binary sorting tree. The numbers in the boxes denote integers sorted in this tree, while the numbers below the boxes show the subtree counts, i.e. the number of elements in the respective subtree.

4.3.2 Candidate sorting

Algorithm 5 requires a suitable data structure for storing and sorting the found candidates. Ordinarily, this would indicate a list or vector structure and the usual sorting algorithms.

However, the nature of the problem can be used to reduce the time and space complexity. You will see in Chapter 6 that a well-chosen motivation parameter M_0 will decrease the number of candidates actually fitted to a small fraction of the number of candidates. Thus, the time spent in storing and sorting the tail of the candidate list will be wasted, since these candidates will most likely not be fitted.

In addition, it happens that the smoothed image of a spot has a broad local maximum spreading over several pixels. While inclusive NMS finds multiple maximums for this spot, there should be only one candidate.

To address these optimisation issues I have devised a size-limited merging binary sorting tree. This data structure is a binary sorting tree with two special properties: a size limitation that avoids storing and sorting irrelevant elements and a merging property that will combine equal elements into a single node.

A size-limited tree is a binary sorting tree that stores only the n smallest elements from the input, with n being a parameter of the data structure. The size limitation is implemented by storing the number of elements included in each subtree of a node together with the link to that node (this number is termed the *subtree count*). An example of such a tree is shown in figure 4.4.

When the tree is descended to find an insertion position for a new element, the number of elements smaller than the current element are computed and checked against n. This computation is performed by summing the subtree counts for the

left branch for all nodes at which the right branch was chosen. If this number is greater than n, the element which is to be inserted can be dropped safely since there are at least n elements in the tree smaller than it. For example, consider the tree in figure 4.4: If the number 11 is to be inserted, the insertion code will traverse through the nodes 8, 10 and 12. If the size limit is smaller or equal to 5 (4 nodes on the left-hand side plus the 8-node), the insertion process can be aborted after comparing with the 8-node; if the limit is smaller or equal to 7, after comparing with the 10-node.

The merging property means that a user-supplied merge function is used to merge an element that should be inserted but is equal to an element already found in the tree. This, if combined with a suitable ordering relation, allows quick detection of maximums stretching over several pixels.

To achieve this effect, the ordering relation for maximum pixels representing spots is defined as the function

$$o(x_1,y_1,s_1,x_2,y_2,s_2) = \begin{cases} -1 & \text{if } s_1 > s_2 \\ 1 & \text{if } s_2 < s_1 \\ -1 & \text{if } s_1 = s_2 \wedge x_1 \ll x_2 \\ 1 & \text{if } s_1 = s_2 \wedge \wedge x_1 \gg x_2 \\ -1 & \text{if } s_1 = s_2 \wedge x_1 \approx x_2 \wedge y_1 \ll y_2 \\ 1 & \text{if } s_1 = s_2 \wedge x_1 \approx x_2 \wedge y_1 \gg y_2 \\ 0 & \text{otherwise} \end{cases} , \tag{4.3}$$

with $x_1 \ll x_2 \iff |x_1 - x_2| \geq m_s\sigma_x \wedge x_1 < x_2$ and equivalent definitions for $\gg$ and $\approx$. This function judges a maximum with the coordinates x_1, y_1 and strength s_1 to be smaller, equal or greater than a second maximum with x_2, y_2, s_2 if the result of the ordering function $o(x_1,y_1,s_1,x_2,y_2,s_2)$ is smaller, equal or greater to 0, respectively. o will place maximums with large intensities first in an ordered list; if two maximums have the same intensity, the X coordinate is used to clarify the situation; if the comparison is unclear, the Y coordinate is used. If the intensities and coordinates match closely for two candidates, it is assumed that they belong to the same peak and are thus equal. Please note that the primary ordering relation is reversed from the intuitively found ordering and that the X axis is the preferred ordering axis. The NMS algorithm suggests this ordering since it operates row-wise and finds maximums with low Y coordinate first, a procedure which would result in a structural bias towards the top part of the image if not corrected with this ordering relation.

To make best use of the size-limited tree, the size limit must be chosen as small as possible. This is easily effected by choosing an arbitrary and small limit and raising this limit every time more elements are requested from the tree than are present within the limit. Evidently, this requires a recomputation of the NMS.

4.3.3 Candidate fitting algorithm

When it comes to spot fitting, Schüttpelz, Thomann, Thompson and Betzig agree in fitting the candidate spots with Gaussian functions, which is not surprising considering that Cheezum *et al.* found this method to be superior [9]. Zhuang *et al.*

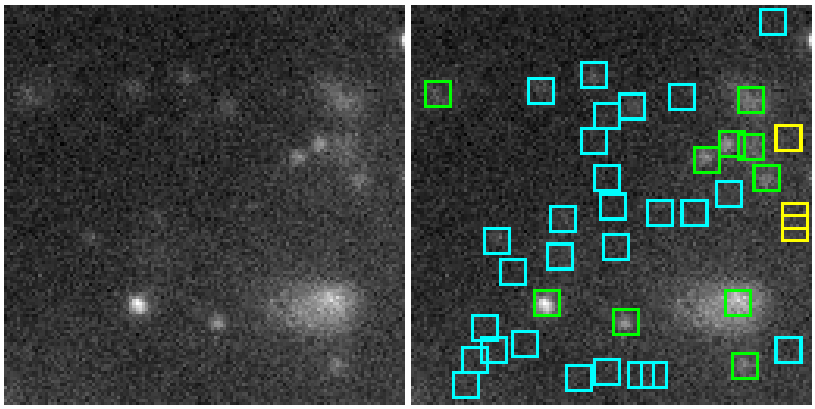


Figure 4.5: Candidate detection. Good fits are shown in green, bad fits are shown in yellow and candidates that were present in the size-limited tree, but not fitted, are shown in cyan.

diverge by using the Gaussian error function; this choice is logically consistent[6], but neither computationally desirable nor supported by published research. The only disagreement amongst the researchers using the Gaussian model persists in the number of free parameters: While Schüttpelz does a full Gaussian fit, including standard deviation parameters, Thomann and Betzig use a fixed standard deviation, and Thompson does not elaborate on this issue (even though he stated that he knew the standard deviation, thus making fixed SD fitting likely).

I expect a fixed SD to be less susceptible to fit to random noise with a strong amplitude, since it has fewer dimensions of play available to do so. While the SD needed to fit a Besselian PSF (possibly distorted by acquisition system imperfections) might differ slightly depending on the amplitude of the signal, I deem the smaller noise susceptibility less important, motivating the choice of fixed SD parameters for fitting.

Algorithm 6 Spot fitting process

Require: Position p in image I

1: **if** p is closer to the image border than $2m_f\sigma$ **then**
2: **return** Invalid fit
3: **else**
4: Cut window W of width $4m_f\sigma + 1$ around p
5: Fit model function ϕ to the pixels in w using initial guess $x_0 = x_p, y_0 = y_p, A = A', B = B'$ resulting in parameter vector a_p.
6: **return** Fit a_p.
7: **end if**

After deciding what model to fit, it has to be decided which pixels to fit the model to. It is obvious that computational speed requires the number of pixels to fit

[6] As I have elaborated in section 3.1, pixel values in an image are local integrals of the intensity distribution. Therefore, if the PSF is approximated by a Gaussian function, the pixel values should be approximated by the local integrals of a Gaussian function.

the model to should be kept as small as possible. It is also necessary to include not only fit as many spot pixels as feasible, but also some surrounding noise pixels, to arrive at a precise estimate for the spot location and strength. Again, this motivates a mask size proportional to the PSF SD, with a proportionality factor m_f that is to be determined experimentally.

For implementing the actual spot fitting procedure, I used the `GSL` implementation of the Levenberg-Marquardt algorithm. The model function was set to a Gaussian PSF in two dimensions with fixed standard deviations σ_x and σ_y and free parameters for the subpixel-precise centre (x_0, y_0), amplitude A and and an additive constant B as given by

$$\phi(x_0, y_0, A, B) = \frac{A}{2\pi\sigma_x\sigma_y} \exp\left(-\frac{(x-x_0)^2}{2\sigma_x^2} - \frac{(y-y_0)^2}{2\sigma_y^2}\right) + B. \tag{4.4}$$

This function has the advantage of having analytically computable derivatives whose values can be computed from the function value with a small number of multiplications. In explicit form, these derivatives are:

$$\frac{\partial\phi}{\partial x_0} = \frac{x-x_0}{\sigma_x^2}\,\phi, \qquad \frac{\partial\phi}{\partial y_0} = \frac{y-y_0}{\sigma_y^2}\,\phi, \qquad \frac{\partial\phi}{\partial A} = \frac{1}{A}\phi, \qquad \frac{\partial\phi}{\partial B} = 1 \tag{4.5}$$

It is crucial to note that these derivatives must be used explicitly in the fitting process. There are NLLS fitting routines that will compute the derivatives of the model function internally and numerically, which is inherently costly.

The initial guess for the fitting coordinates is a Gaussian centred at the candidate spot's centre position (x_p, y_p). The upper left pixel of the fitting window is taken as an estimate of the background value B', and the initial amplitude A' is chosen such that $\phi(p_x, p_y, A', B')$ is equal to the pixel value at p.

Subsequently, this initial guess is iteratively improved using the Levenberg-Marquardt routine. Once the fit has converged, which is according to [39] indicated by a drop in the residues that is below a given threshold, fitting is aborted and the current parameters form the fit.

The `GSL` implementation circumvents the need for the Hessian matrix [13] and the second-order derivatives. Therefore, these first-order derivatives are sufficient and I have implement the spot fitting as shown in Algorithm 6.

4.3.4 Concurrent streaming of localisations

Streaming

Algorithm 4 returns the full list of localisations after processing the whole source stack. This buffering of results is for many applications unnecessary, is wasting storage space and delaying feedback and can be eliminated by an event-driven approach.

This idea can be realised using the publisher/subscriber software design pattern [14, pg. 287]: Algorithm 4 is changed to publish events about the start, the end or the restart of the deconfinement process and to publish one event per source image containing the localisations in that image. Any module that processes

localisations can receive the events by registering with the localisation publisher and providing a method for processing localisations. To implement this scheme, line 18 is erased, and between lines 16 and 17 the lines

1: Publish new localisation event with elements t
2: Erase all elements from t

are inserted. A static view on the resulting system is given in the appendix in Figure C.1(a) and supplemented with a dynamic view in Figure C.1(b).

Parallelization

Given the current trend to multi-core hardware, it is important to consider opportunities for concurrency in algorithm 4. Naturally, the loop iterating over all images in line 2 provides such an opportunity if the publishing interface is implemented with concurrency in mind.

Another chance for concurrency opens in the loop iterating over the spots in line 6. However, parallelizing the algorithm in the image loop is preferable to parallelization in the candidate loop since the former is outermost and thus requires less communication between the concurrent threads.

4.4 The `rapidSTORM` image construction

The `rapidSTORM` engine I have developed and described in the last section reduces a source stack to a set of localisations, which are basically pairs of coordinates in $\mathbb{R}^2$. This is not the desired form of microscopy results: Scientists are used to and existing scientific methods process images[7], and images are discrete matrices with values in a given range. Therefore, the localisation set should be converted to an image showing the localisation positions with a precision that conveys a good visual impression of the real precision attained with photoswitching microscopy.

For this problem, Schüttpelz has employed a simple, yet effective algorithm that histogramised the found localisations in bins $1/10$ pixels wide and high[8] and projected the histogram results on a colour axis ranging from black over red to white. The parameters for this projection were hand-chosen. Zhuang *et al.* seem to have used a similar algorithm, even though they do not state it explicitly. Betzig *et al.* sum the values of Gaussian functions, which are centred at the likely emission position and has a standard deviation equal to the uncertainty estimated by the NLLS fitting method, and most likely project the results on some colour axis.

These methods have a common theme: They construct an intermediate image $\mathbf{D}'$ with undefined dynamic range from the input set of localisations L and then map the received dynamic range into an intensity or colour space to obtain the final image $\mathbf{D}$. I term these two distinct process *pixelisation stage* and *normalisation stage*, respectively. Both of these images have a spatial resolution significantly higher than that of the source image to reflect and accommodate the

[7] Not to mention the strong bias of scientific magazines to colourful images instead of several dozen pages of coordinates.

[8] In computer science terms, Schüttpelz used a nearest-neighbour algorithm.

super-resolution precision. I will denote this resolution enhancement factor by R_e, and a typical choice is 10.

4.4.1 Algorithm for pixelisation stage

While I did not use Betzig's more involved Gaussian spread method because of the uncertainty in the SD estimation done by the NLLS fitting [39], Schüttpelz' simpler method seems to be prone to discretization errors. I deem a linear interpolation of a target image from the localisations distributed in $\mathbb{R}^2$ a workable compromise because it honours the exact fit position while not introducing computational overhead or more parameters for the method.

Therefore, I propose the following pixelisation formula for generating the pixelated image:

$$\mathbf{D'}_{x_0,y_0}(L) = \sum_{l \in L} w(R_e \cdot x(l) - x_0) \cdot w(R_e \cdot y(l) - y_0), \tag{4.6}$$

where $x(l)$ is the X coordinate for the localisation l in the coordinates of the source stack, $y(l)$ the respective Y coordinate and w is the weight function given by

$$w(x) = \begin{cases} 1 - |x| & \text{if } -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}. \tag{4.7}$$

It is obvious that this function approaches the Schüttpelz pixelisation scheme when all localisations are very close to target pixel positions. Furthermore, the contribution of a single localisation to the target image is always 1 and it only affects the four pixels surrounding the localisation.

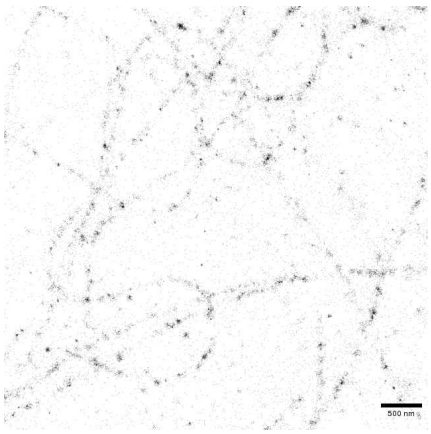
4.4.2 Algorithm for normalisation stage

The pixelisation stage yields a pixelated image with non-negative real numbers as pixel values. In order to obtain a normal digital image, the normalisation stage needs to specify a way to transform these real values into integer values within a given range.

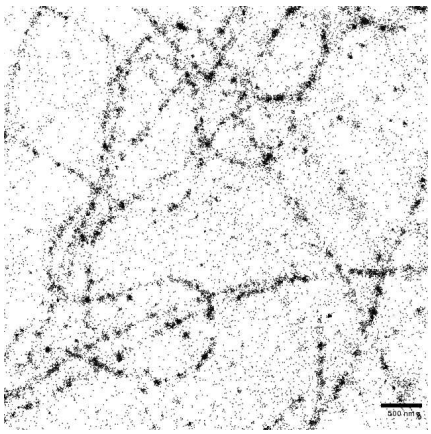
The trivial approach to this problem would be a linear transformation, where the target image is computed as

$$\mathbf{D}_{xy} = \left\lfloor \frac{G}{\max\limits_{a,b} \mathbf{D'}_{ab}} \mathbf{D'}_{xy} \right\rfloor, \tag{4.8}$$

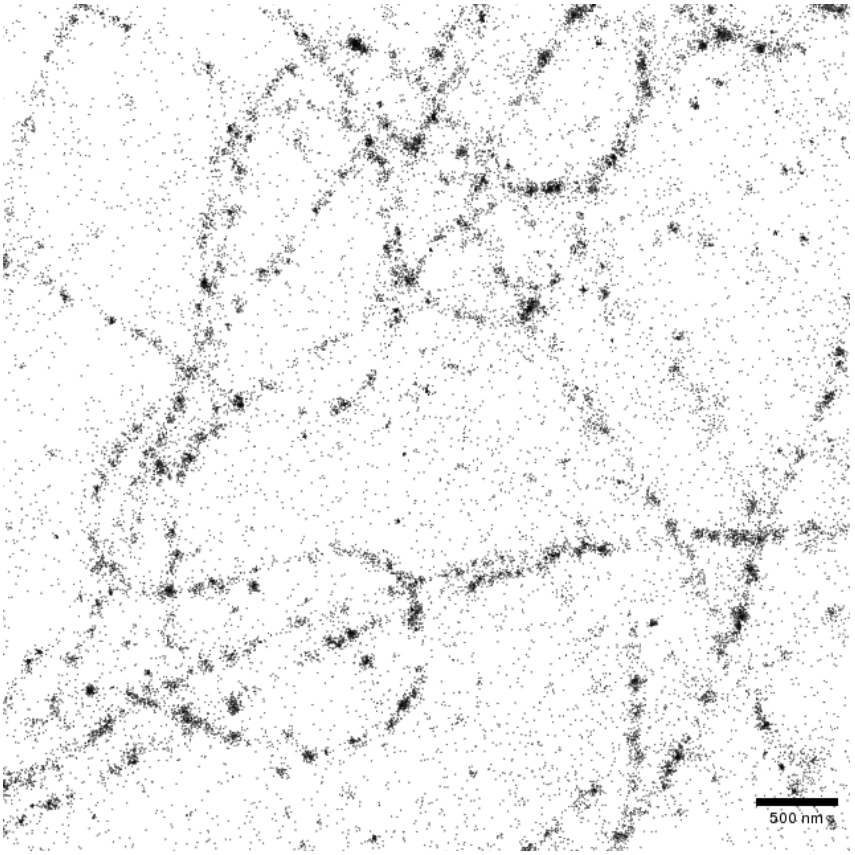
with $\lfloor \cdot \rfloor$ denoting the Iverson floor function, which gives the largest integer not larger than its argument. However, this method yields bad results if a small number of fluorophores are localised very often, which is commonplace in dSTORM practise. In this case, the localisations of these fluorophores culminate in few pixels with very high pixel values, thus ousting all other localisations into the very low grey levels which are almost indiscernible from the background. Thus, this method is suboptimal.



(a) Almost unnormalised ($r = 0.1$)



(b) Completely normalised ($r = 1$)



(c) Mildly normalised ($r = 0.5$)

Figure 4.6: Comparison of histogram normalisation results for different weight parameters. The image shown here is the result image of specimen 1 and shows microtubuli. These result images were all constructed by the `rapidSTORM` image construction, but differ in the weight parameter used for histogram normalisation.

Schüttpelz has improved this method by introducing a hand-chosen threshold M for the linear transformation:

$$\mathbf{D}_{x,y} = \min\left\{ \left\lfloor \frac{G}{\min\left\{M, \max\limits_{a,b} \mathbf{D'}_{a,b}\right\}} \mathbf{D'}_{x,y} \right\rfloor, G \right\}. \tag{4.9}$$

Although this method worked fairly well in practise, it has principally the same limitations as the linear transformation, namely a suboptimal image contrast.

As explained in section 3.1.6, a useful tool for image contrast optimisation is the histogram equalisation. However, the histogram equalisation method is incompatible with the input to the normalisation stage: The former operates on discrete, equidistant values, while the latter consists of pixels with real values.

I addressed this incompatibility by using the transformation from equation 4.8 before applying the histogram equalisation. I minimised the information loss by linearly transforming $\mathbf{D'}$ according to equation 4.8 into a high-depth image $\mathbf{D''}$ with, for example, 16 bits per pixel. $\mathbf{D''}$ is then transformed into $\mathbf{D'''}$ by a weighted histogram equalisation of all non-zero pixels. The weight factor is user-configured with a first estimate of $r = 0.25$; by this method, the microscope operator can easily choose a contrast level that pronounces the important structures in the image. Finally, $\mathbf{D}$ is computed from $\mathbf{D'''}$ by linearly scaling the intensity range to the range allowed for $\mathbf{D}$; if this scaling is reflected in the choice of G in equation 3.23, the intermediate image $\mathbf{D'''}$ can be omitted.

In mathematical terms, let B denote the depth enhancement factor that $\mathbf{D''}$ has over $\mathbf{D'''}$. Then the above algorithm is equivalent to computing

$$\mathbf{D} = \frac{1}{B} \mathrm{heq}_w \left(\left\lfloor \frac{BG}{\max\limits_{a,b} \mathbf{D'}_{ab}} \mathbf{D'} \right\rfloor, r \right). \tag{4.10}$$

4.4.3 Incremental computation

In the laboratory, getting feedback from the super-resolution imaging process as early as possible is highly desirable. Errors can be discovered and corrected faster if intermediate images from the deconfinement process could be displayed parallel to the acquisition and computation process.

A naïve implementation of the algorithms developed in sections 4.4.1 and 4.4.2 for parallel viewing of the deconfinement results is infeasible. At least two complete passes over $\mathbf{D'}$ are needed to compute $\mathbf{D}$ (one pass for the histogram, one pass for the result image), while $\mathbf{D'}$ and $\mathbf{D}$ regularly have megapixel dimensions due to the resolution enhancement.[9] Consequently, if a parallel viewing is possible at all, it must be achieved by an iterative algorithm.

[9] To give an example, the cameras used for the dSTORM method have detectors 1024 pixels wide and high, and dSTORM realises R_e factors of 10. This gives $\mathbf{D'}$ and $\mathbf{D}$ dimensions of 10240x10240 pixels.

A data structure for an iterative algorithm should store sufficient intermediate results of equations 3.23, 4.6 and 4.10 that will be updated iteratively. I chose to store

- $\mathbf{D}'$, $\mathbf{D}''$ and $\mathbf{D}$
- the maximum $M_{\mathbf{D}'} = \max_{x,y} \mathbf{D}'_{xy}$,
- the discretization factor $M'_{\mathbf{D}'}$ that stores the (possibly obsolete) value of $M_{\mathbf{D}'}$ between recomputations of $\mathbf{D}''$,
- the histogram vector $\mathbf{h}$ with $\mathbf{h}_v = h_w(\mathbf{D}'', v, r)$,
- the transfer vector $\mathbf{t}$ storing the weighted histogram equalization (WHE) transfer function used for the last recomputation of $\mathbf{D}$. After recomputation of $\mathbf{D}$, $\mathbf{t}_v = T_{WHE}(\mathbf{D}'', v, r)$ is ensured.

When a new localisation l is incrementally added to these data structures, the operations given in algorithm 7 suffice to update these structures.

Algorithm 7 Incremental update of target image

Require: $\mathbf{D}$, $\mathbf{D}'$, $\mathbf{D}''$, $M_{\mathbf{D}'}, M'_{\mathbf{D}'}, \mathbf{h}, \mathbf{t}$ as defined in text and in consistent state
Require: New localisation $\mathbf{l}$

1: Increase the 4 pixels surrounding $\mathbf{l}$ according to equation 4.6
2: **for all** increased pixels (x, y) **do**
3: $M_{\mathbf{D}'} \leftarrow \max\left\{M_{\mathbf{D}'}, \mathbf{D}'_{xy}\right\}$
4: oldValue $\leftarrow \mathbf{D}''_{xy}$
5: newValue $\leftarrow \left\lfloor \frac{BG}{M'_{\mathbf{D}'}} \mathbf{D}'_{xy} \right\rfloor$
6: $\mathbf{D}''_{xy} \leftarrow$ newValue
7: **if** oldValue and newValue differ **then**
8: Decrement $\mathbf{h}_{oldValue}$ and increment $\mathbf{h}_{newValue}$
9: $\mathbf{D}_{xy} = \mathbf{t}_{newValue}$
10: **end if**
11: **end for**

It is obvious that algorithm 7 performs the update of almost all data structures used in $O(1)$ time and space. However, it does *not* update $M'_{\mathbf{D}'}$ and $\mathbf{t}$ and avoids this update for good reason: Since any change to $M'_{\mathbf{D}'}$ forces a rediscretization of $\mathbf{D}''$ and any change to $\mathbf{t}$ forces a recomputation of $\mathbf{D}$ to ensure consistency, these operations are expensive and should be delayed as long as possible.

To achieve this aim, I have introduced the thresholds for the errors in discretization, in the histogram equalisation transfer function and in the histogram equalisation operation.

The error in discretization can be controlled with a threshold for the quotient $M_{\mathbf{D}'}/M'_{\mathbf{D}'}$ of the actual and the used maximum value. It is checked in line 3 and, if it is exceeded, $M'_{\mathbf{D}'}$ is set to $M_{\mathbf{D}'}$ and $\mathbf{D}''$ is recomputed form $\mathbf{D}'$ by $\mathbf{D}'' = \mathbf{D}'/M_{\mathbf{D}'}$. I have used a value of 1.1 for this threshold.

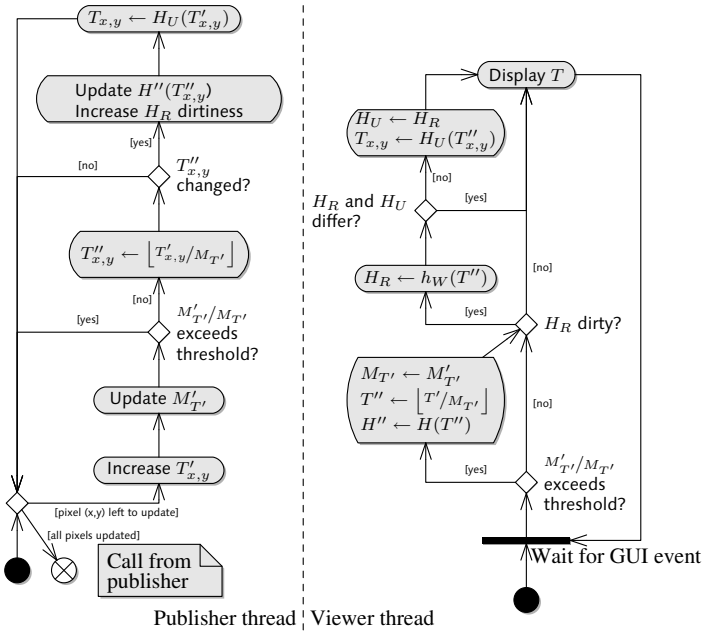


Figure 4.7: Activity diagram of the `rapidSTORM` image viewer. On the left side, the steps necessary for an incremental update of the viewport window are shown. On the right side, the steps taken for a complete recomputation are visible together with the thresholds that control them. The mutual exclusion locks ensuring data integrity are not shown for the sake of clarity.

The histogram equalisation transfer function can be controlled by counting the number of changes in the histogram in line 8. Once a threshold is exceeded, an updated transfer vector $\mathbf{t}'$ is computed from the current histogram. I have used a value of 100 for this threshold.

The error in the histogram equalisation operation can be controlled by comparing the results of the updated transfer vector $\mathbf{t}'$ with $\mathbf{t}$ when the former is computed. If $|\mathbf{t}_i - \mathbf{t}'_i|$ exceeds a given threshold for any i, $\mathbf{t}$ is set to $\mathbf{t}'$ and $\mathbf{D}$ is recomputed from $\mathbf{D}''$ as in line 9; if the threshold is not exceeded, $\mathbf{t}'$ is discarded. I have used a value of 5 for this threshold.

I have chosen these values of all these parameters arbitrarily and not scrutinised the effects of different choices.

4.4.4 Concurrent and event-driven computation

While the algorithm exposed in the last subsection has a very low average time complexity, its worst case complexity is high – if $M'_{\mathbf{D}'}$ must be changed, a full recomputation of $\mathbf{D}''$, $\mathbf{t}$ and $\mathbf{D}$ is likely. This computation takes a long time, and since the incremental update given by Algorithm 7 is naturally executed in the context of the method called by the engine publishing interface, a recomputation will block the localisation publishment and thus the computations of other engine threads. This suggests the number of recomputations should be kept minimal, which can be achieved by recomputing $\mathbf{D}''$, $\mathbf{t}$ and $\mathbf{D}$ only if needed by the graphical

interface.

For these reasons an extra thread is spawned that waits for events from the graphical interface or a user-defined timeout. When any of these events happens, the thresholds defined in the last subsection are checked, any necessary of the recomputations defined there performed and the resulting target image is displayed. The code called by the localisation publisher only performs the incremental updates given by Algorithm 7. Thereby, real-time localisation computation is not hindered unnecessarily by the displaying process while the localisations are still visualised concurrently to the computation process. Figure 4.7 provides a visual representation of the image construction's dynamic behaviour.

4.5 Guessing the spot standard deviation

In section 4.2, you have noticed that the deconfinement algorithm not only takes a source stack as an input, but also the spot standard deviation σ. Under lab conditions σ is not always known and especially not known exactly since minor changes to the experimental setup can introduce subtle changes in σ; consequently, an algorithm for guessing this parameter from a source stack would ease lab use.

I developed such an algorithm, formalised as Algorithm 8. To understand its principle, suppose a set of spots S_σ is given that contains mostly true spots. Then, determining the values of σ_x and σ_y could be done by simply fitting the spots in S_σ with free standard deviation parameters and averaging the results. Unfortunately, this simple approach raises a bootstrapping problem: To construct S_σ, a judgement of spots is needed; to judge spots, σ needs to be known; and to know σ, S_σ needs to be constructed.

This bootstrapping cycle can be broken if a good initial guess σ_0 for σ can be provided. This assumption is most reasonable since Thompson [53] has found a range of standard deviations to yield optimal deconfinement precisions, which experimenters make sure to reach. The good initial guess σ_0 can be used to run the deconfinement routine, and the spots found in that process can be used to find an improved guess for σ that is termed σ_1. This process is iterated for each guess σ_i until σ_j is found with $\sigma_{j+1} \approx \sigma_j$.

Straightforward implementation of this scheme would lead to an increase in algorithm run-time by a factor of j and require buffering of all image data. These costs can be circumvented if the computation with the current guess is aborted when it becomes clear that σ_{i+1} will differ significantly from σ_i. In mathematical terms, a maximum error $\Delta\sigma$ is chosen. After fitting a small set of spots S_σ, the 90% confidence interval (estimated by assuming the elements of S_σ to be normally and independently distributed, a belief that is supported by the success of the estimation method) for S_σ is computed. If the confidence interval is contained in $\sigma_i \pm \Delta\sigma$, $\sigma_{i+1} \approx \sigma_i$ is assumed and the σ estimation process is considered successful. If the confidence interval is disjunct from the interval $\sigma_i \pm {}^{\Delta\sigma}/_2$, [10] the current

[10] The non-rejection interval width is chosen differently from the acceptation interval width to avoid analysis paralysis in border cases. Suppose both intervals equal and $\sigma_{i+1} = \sigma_i + \Delta\Sigma$; then, neither acceptation nor rejection could occur because the confidence intervals stays half in and half out of the acceptation and rejection zone.

Algorithm 8 σ estimation

Require: Initial guess σ_i
Require: Tolerated σ error $\Delta\sigma$
Require: Random number X with $0 \leq X < 1$
Acceptance interval $I_A = [\sigma_0 - \Delta\sigma, \sigma_0 + \Delta\sigma]$
Non-rejection interval $I_R = [\sigma_0 - {}^{\Delta\sigma}/_2, \sigma_0 + {}^{\Delta\sigma}/_2]$
$n \leftarrow 0$, $n_c \leftarrow 23 + \lfloor 100 \cdot X \rfloor$
Initialise estimates for mean and variance of σ according to West on-line computation scheme[32]
while no definitive result was found **do**
  while $n < n_c$ **do**
    Take next localisation l
    if Amplitude is at least 25% of the maximum amplitude encountered and σ differs from σ_0 by no more than a factor of 2 **then**
      Fit spot around l with free deviation to get l'
      if l' is reasonably close to l **then**
        Update means and variances with new measurement $\sigma(l)$
        Increment n
      end if
    end if
  end while
  $n_c \leftarrow n + 1$
  Compute 90% confidence interval $I_C(\sigma)$ for mean of σ as $t_{0.1,n-1} \sqrt{{}^{\mathrm{Var}(\sigma)}/_n}$
  if $I_C \subset I_A$ **then**
    Accept σ_i and break loop
  else if $I_C \cap I_R = \varnothing$ **then**
    $\sigma_i \leftarrow \sigma$
    Restart `rapidSTORM` engine and σ estimation, keeping the current n_c
  end if
end while

guess σ_i is assumed to be wrong. In this case, the current mean of S_σ is chosen as σ_{i+1}, all results accumulated so far are discarded and the `rapidSTORM` engine is restarted. Since there is a 10% probability for the confidence interval being wrong, the next sample is required to be strictly greater than the current to avoid loops with wrong estimates for σ. If neither of the two conditions is true, the sample S_σ is enlarged.

This method works well in most cases, but is prone to errors when localisations with very small photon counts or spots generated by multiple active fluorophores are included. It is therefore necessary to impose limiting conditions on the amplitudes and standard deviations of the used localisations, which are shown in Algorithm 8.

Chapter 5

Experimental material and methods

The success of any experimental testing critically depends on a good choice of experimental objects. In this case, the objects are source stacks and should span the space of data encountered in photoswitching microscopy as well as provide some way to check deconfinement results against a ground truth.

The first demand could be fulfilled with a range of source stacks from different photoswitching microscopy methods. Unfortunately, I had only data for dSTORM available and was therefore limited to testing with this technique, even though I expect data from STORM or PALM to yield qualitatively similar results. To test dSTORM, a small number of source stacks suffices since dSTORM source stacks are basically noise and spots of different photon counts and the same underlying PSF; therefore, the variation in the input data stems from the SNR and the PSF standard deviation, and the SNRvaries wildly in a single source stack because fluorophore activity differs in strength and duration.

The second demand, the availability of ground truth, is harder to meet: By design, photoswitching microscopy measures the position of objects with a precision unattainable by classical microscopy. This problem will be addressed in this section by not only testing with a careful choice of real source stacks, but additionally with source stacks generated stochastically.

5.1 dSTORM image acquisition process

While details may differ, all real specimens were prepared using the dSTORM photoswitching microscopy method. A typical dSTORM work-flow has been described in [56] as follows:

> "African green monkey kidney COS-7 cells were plated in LabTek 8-well chambered coverglass [...]. Microtubules were stained with mouse monoclonal anti-β-tubulin antibodies [...]. Subsequently, cells were stained for 60 min with [fluorophore-]labeled goat anti-mouse $F(ab')_2$ fragments [...].

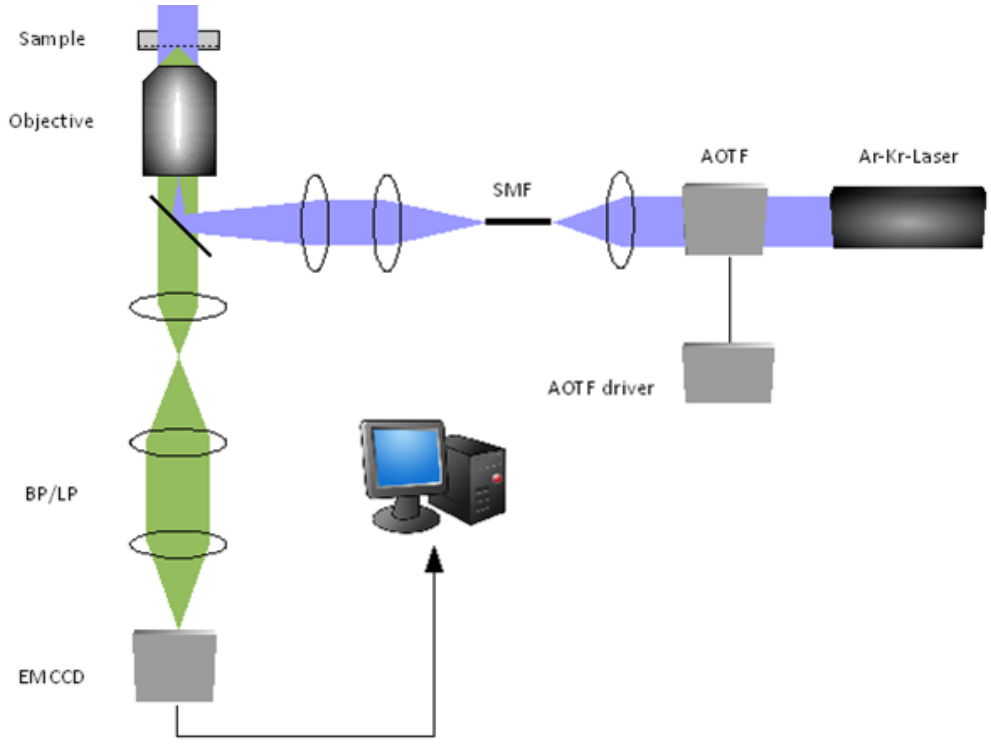


Figure 5.1: Optical wide-field setup used for dSTORM

> "[...] Fluorescence imaging was performed on an Olympus IX-71 applying an objective-type total internal reflection fluorescence (TIRF) configuration equipped with an oil-immersion objective ([60×, NA 1.45]) and was described earlier. The 647 nm laser beam of an argon-krypton laser [...] was selected by an acousto-optical transmission filter (AOTF) and coupled into the microscope objective by a polychromic beam-splitter (532/647 [...]). Fluorescence light was spectrally filtered [...] and imaged on an EMCCD camera [...]. Additional lenses were used to achieve a final imaging magnification of 225, i.e., a pixel size of 70 nm. [...] Photoswitching microscopy was performed with a laser power of 22–40 mW. Typically, 8000–16000 images were recorded at a frame rate of 10–100 Hz corresponding to measurement times of 80 seconds to 26.7 minutes."

An Andor Technologies camera used for dSTORM measures pixel values that are not scaled to some arbitrary interval, but reflect the number of counts received from the A/D converter. I will term that unit A/D count (ADC). According to the specification sheet for this camera [51, pg. 49], roughly 16 ADCs are emitted for each photon that immissions on the detector at the amplification level of 200 used for specimen 1.

No.	Dye	Excit. power (mW)	Integration time (ms)	Source image dimensions (pixels)	Pixel size (nm)	Number of source images
1	Cy5	25	50	128x128	71	8000
2	Cy5	30	10	128x128	85	20000
3	Atto520	50	50	128x128	85	8000
4	QDot655	-	100	512x512	85	500
5	QDot655	-	1	128x128	85	1000
6	QDot655	-	1000	128x128	85	600
7	Cy5	30	50	256x256	85	10000
8	Cy5	20	100	128x128	71	8000

Table 5.1: Real specimens used for `rapidSTORM` verification. Specimen 1 was selected for its high amount of noise, specimen 2 and 8 for their long aquisition and integration times, specimen 3 for the use of an Atto520 dye, and specimen 7 for its large images. Specimens 4–6 were selected to allow localisation precision measurements with reliable statistics, since quantum dots show very strong and reliable fluorescence behaviour.

5.2 dSTORM stacks selected for testing

In Table 5.1, I have compiled a small number of real input images which should span the specimen space. To achieve this aim, I have selected source images with well-separated quantum dots (no. 4-6) and from real biological samples, with different integration times and excitation powers to show different SNRs and with different source stack lengths to demonstrate the limits (or rather, lack of limits) of `rapidSTORM` image processing in the face of gigabytes of data. Additionally, specimen 1 shows fluorophore activity apart from the focal plane that acts as non-independent noise.

5.3 Stochastically generating data sets

As elaborated in the introduction to this section, ground truth data for testing a `rapidSTORM`-driven system is hard to get, but essential to valid testing. In a normal laboratory, no specimen can be produced so precisely that the errors in the deconfinement process clearly surpass the errors in the production process; and comparably exact measurement methods capable of verifying the results of deconfinement are rare and expensive. Therefore, a stochastic simulation system capable of producing realistic source stacks is the only cheap, practical and flexible way to provide a source stack together with ground truth data about the true fluorophore positions and emissions.

To produce such data, I defined both the qualitative and the quantitative characteristics of the signal and the noise and then sampled random data from the GSL Mersenne Twister [35] random number generator with these characteristics and a random seed of 42. Since the noise can be scrutinised independently of the signal, but not vice versa, I started by modelling the noise.

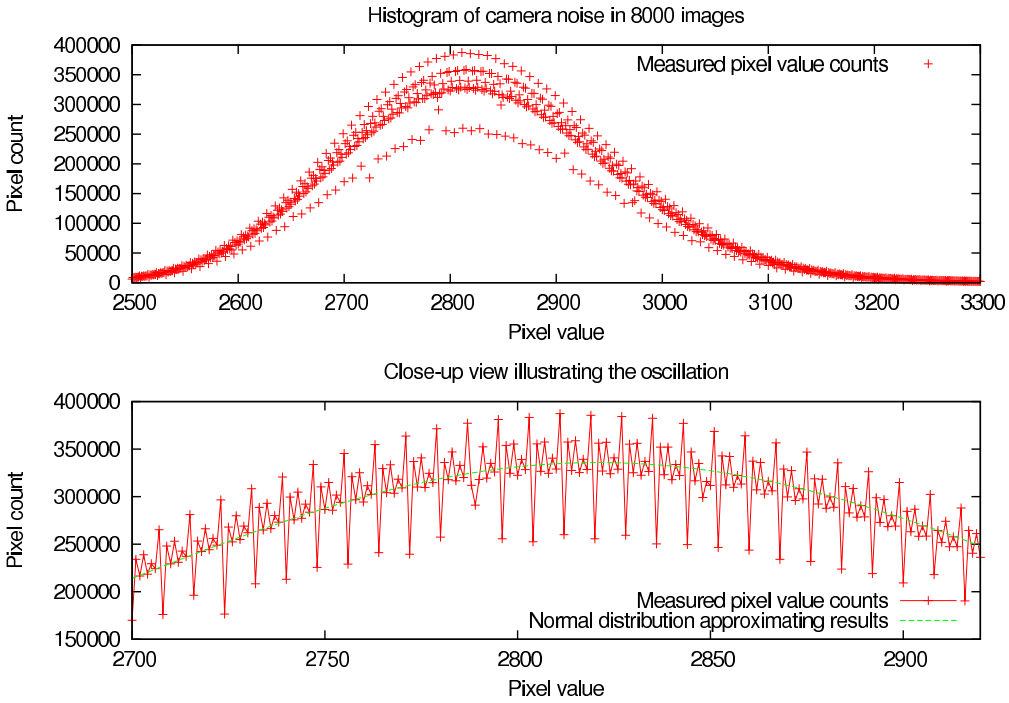


Figure 5.2: Histogram of camera background noise. The short-range oscillation distorting the upper graph becomes clear in the close-up view. This behaviour cannot be explained physically and can be attributed to camera quirks.

5.3.1 Noise model

Thompson[53] identified two principal sources of noise for nanometer-accurate localisation of fluorescent probes: The background noise due to thermal electron noise and stray photons as well as the pixelisation noise due to the limited resolution of a CCD. The pixelisation noise is modelled more easily in conjunction with the signal modelling and will be delayed; therefore, the aim of this subsection is to characterise and quantify the pure background noise that occurs even in absence of fluorophore emissions.

The quantification of background noise depends on a number of factors too large to exhaustively consider in this work. Thompson[53] alone lists four common sources of background noise: “readout error, dark current noise, extraneous fluorescence in the microscope [...] and cellular auto-fluorescence.”

For this reason I decided to use descriptive statistics for the reproduction of camera noise. I did this by reducing the real specimens to noise by pruning every pixel closer than 10 pixels to any localisation. The resulting pixels were histogramised and fitted with a normal distribution, a weighted mean between a normal and a Gamma distribution and finally used unsmoothed as a probability distribution. Figure 5.2 shows a characteristic section of the result for specimen 1.

The bottom part of Figure 5.2 shows how the real noise was overlaid by a short-range oscillation distorting the likelihoods of measured pixel values, probably stemming from the A/D converter. I have corrected this oscillation by rounding each value down to the next lower multiple of the oscillation period, in this case

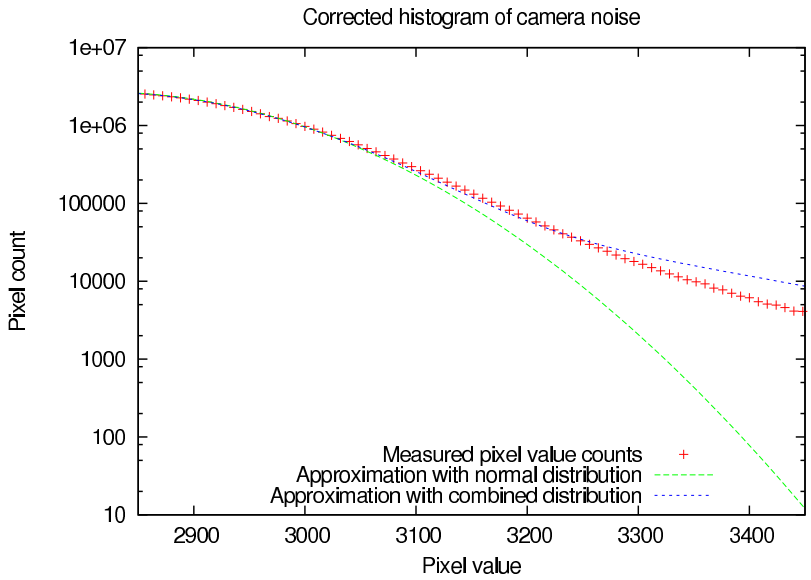


Figure 5.3: Histogram of camera background noise, corrected for A/D converter bias. Note the systematic underestimation of the number of high pixel values that is exhibited by the normal distribution.

8. Effectively, this means that all values in an oscillation period were accumulated to suppress the noise.

The results of this operation are pictured in figure 5.3. Additionally, this figure contains (scaled) graphs of the two probability density function (PDF)s used to model the data.

The first PDF is a normal distribution that was fitted to the data set 1 yielding parameters of mean $\mu = 2817.4$ and SD $\sigma = 127.4$. It is important to note that this distribution, although a good first estimate, fails to explain the high-value tail of the histogram. Assuming a normal distribution of pixel intensity, the probability of encountering a pixel with A/D count 4000 is extremely low; however, the actual number measured is 293 pixels. This severe underestimation of the probability of high pixel values would introduce a high error into the prediction of regions with noise-induced strong signal, which are the most likely candidates for FPs.

I addressed this issue by assuming that only a part of the noise can be attributed to normally-distributed noise, while the other part stems from a Gamma distribution.[1] In explicit terms the PDF for this combined approach is:

$$\Upsilon_{W,\mu,\sigma,x_0,\theta}(x) = W\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}} + (1-W)\gamma_{1,\theta}(x-x_0) \tag{5.1}$$

$$\gamma_{k,\theta}(x) = x^{k-1}\frac{e^{-\frac{x}{\theta}}}{\theta^k\Gamma(k)} \tag{5.2}$$

[1] The Gamma distribution[6, pg. 108] is the generalisation of the Poisson distribution for non-discrete values of the input parameter. It is used here to approximate the Poisson-distributed noise stemming from stray photons with a continuously distributed parameter.

W denotes a weighting factor between 0 and 1 and Γ denotes the Gamma function found in [6]. The resulting PDF Υ was fitted to the data set 1 resulting in the following parameters:

W	μ	σ	x_0	θ
0.96	2734.6	104.6	2582.6	119.7

Υ approximates the data significantly better, with a prediction of 278 pixels for an A/D count of 4000 (with 293 pixels being the measured number). Additionally, the absolute fit residues are reduced from 5.5 to 2.2 billion A/D counts. Therefore, I deem the Υ approximation better than the Gaussian one.

The fit residues of the Υ distribution are, however, still 2.2 billion A/D counts large; such a large error will have a statistical impact. This statistical impact must be weighted against the flexibility inherent in parametrised functional approximations to the noise profile. Therefore, I will use both the Υ distribution and a probability distribution derived directly from the noise data. This direct PDF derived from a source stack S is given by

$$D(x) = \frac{\sum_{\mathbf{I} \in S} h(\mathbf{I}, x)}{\text{Number of pixels in S}}, \tag{5.3}$$

where h denotes the histogram function.

5.3.2 Modelling fluorophores

In section 2.2.2, you have seen that molecular photoswitches have different energy states. It is therefore natural to model a fluorophore as a continuous-time Markov process[2]. For a simplified model, I have introduced two fluorophore states, one fluorescent, one dark, which are decaying into each other with expected lifetimes of 2 seconds for the dark state and 0.1 seconds for the bright state. The fluorescent state emits photons at a constant rate, which is varied in the experiments within the limits of the Cy5 emission rate found by Heilemann *et al.* [19] (5–10 kHz). The PSF for the model fluorophores was taken to be a pixel-integrated Besselian[3] with independent scaling for the X and Y coordinates, resulting in an intensity distribution of

$$I(x, y) = \left(\frac{J_1(x/s_x + y/s_y)}{x/s_x + y/s_y} \right)^2 \tag{5.4}$$

with s_x and s_y denoting the spatial scaling factors. The product of the fluorophore emission rate, the time spent in the on state in a given frame and the normed value

[2] That is, a fluorophore is in one of n states and has in each infinitesimal time unit a certain probability of changing its state. The probabilities of changing from one state to the other are fixed. An example for such a process is given by radioactive atoms, which can be intact or decayed and have a certain probability of changing from intact to decayed in each time unit, resulting in the characteristic exponential decay process.

[3] Please note that the Bessel function is computationally feasible here because the fluorophore position is known. This allows to compute the PSF values only once at the start of the simulation.

Spec. no.	Size	Noise source	Photon rate	Fl. pattern	# Fl.
9	256 px	Spec. 1	10 kHz	random	100
10	256 px	Spec. 1	10 kHz	lattice	100
11	64 px	Spec. 1	10 kHz	lattice	1000
12	256 px	Spec. 1	5 kHz	random	100
13	256 px	Υ	8 kHz	random	100
14	256 px	Υ'[a]	10 kHz	random	100
15	256 px	Υ''[b]	10 kHz	random	100
16	256 px	Spec. 3	8 kHz	random	100
17	256 px	Spec. 4	8 kHz	random	100
18	256 px	Spec. 8	8 kHz	random	100

[a] Υ distribution with $\sigma = 209.2, \theta = 168.8$, which generates twice as much variance in the pixel intensities as the original does.

[b] Υ distribution with $\sigma = 54.7, \theta = 84.6$, which generates half of the variance in the pixel intensities compared to the original.

Table 5.2: Stochastically generated data sets. The data sets were compiled to span a range of different noise distributions, the range of Cy5 photon emittance rates as well as different fluorophore distribution patterns and densities.

of the PSF gave an expected number of photons for each pixel, which was variated with a Poisson distribution.

This fluorophore model and its results were neither checked against the measured specimens in this work nor were real fluorophore characteristics modelled precisely. This was not only due to a lack of time, but also due to the irrelevance of the concrete parameters used. It is obvious that many sets of parameters will yield fluorescent spots with a range of different SNR values, and for the comparison of algorithms, the precise distribution is irrelevant. It will, however, become important for the precise analysis that is to come in future work.

For the experiments, model fluorophores with an average time to deactivation of 0.1 seconds, an average time to activation of 2 seconds and with Bessel function scaling factors of 2.31 (best fitted with a Gaussian with SD 1.8) were used.

5.3.3 Fluorophore distribution pattern

Fluorophores were distributed across the surface of an imaginary, square object slide of given size, either randomly or in a square lattice pattern. For the random pattern, a X and a Y coordinate were chosen independently and randomly. For the lattice pattern, the fluorophores were placed in an equidistant pattern with the outer fluorophores having 4 pixels of space towards the border.

5.3.4 Specimens generated

Table 5.2 lists the parameters for data sets generated stochastically. I have selected the noise from specimen 1 for its high level of background noise and contrasted it with the lower levels of noise from specimens 3, 4 and 8. To demonstrate the viability of this method, Figure 5.4 shows a typical image from specimen 10.

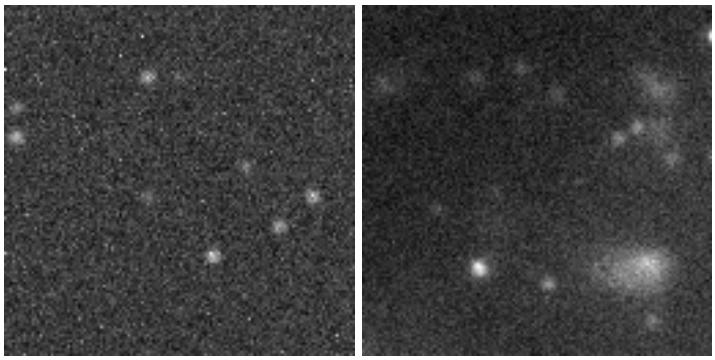


Figure 5.4: Stochastically generated input image. The generated image (left) was captured from the generated data for specimen 10 and is compared, for reference, with the typical `rapidSTORM` image (right). Note that the differing background intensity levels are due to contrast stretching.

5.4 Measuring error rates in candidate search stage

The error rates in the candidate search stage were measured by pairing, for each image, the active fluorophores and the candidates/localisations (called objects in this paragraph). The fluorophore and the object with the lowest distance was paired first, both were removed from their respective set, and the next pair was paired. This means that each fluorophore and each object was paired to at most one object from the other category. Fluorophores and objects that differed more than half the width of the NMS mask size in X or Y direction were not paired. From these pairings, only the candidates were counted that did not have M_0 consecutive unpaired candidates in front of them. This restriction simulates the effects of dynamical thresholding.

The TP count is given by the number of fluorophores correctly paired; the FP count as the number of candidates/localisations not paired with any fluorophore; and the FN count as the number of fluorophores not paired with any candidate/localisation.

5.5 Computational hard- and software

The tests were conducted on an Intel E6550 processor with two cores each clocked at 2.33 GHz. The compiler was the g++ in Debian 4.3.1-9 version, optimising with the O3 flag. Computation times were measured by the GNU time utility, version 1.7, and by the clock() function offered by GNU.

Chapter 6

Results and discussion

> Beware of bugs in the above code; I have only proven it correct, not tested it.
> – Donald Knuth ([31, pg. 5])

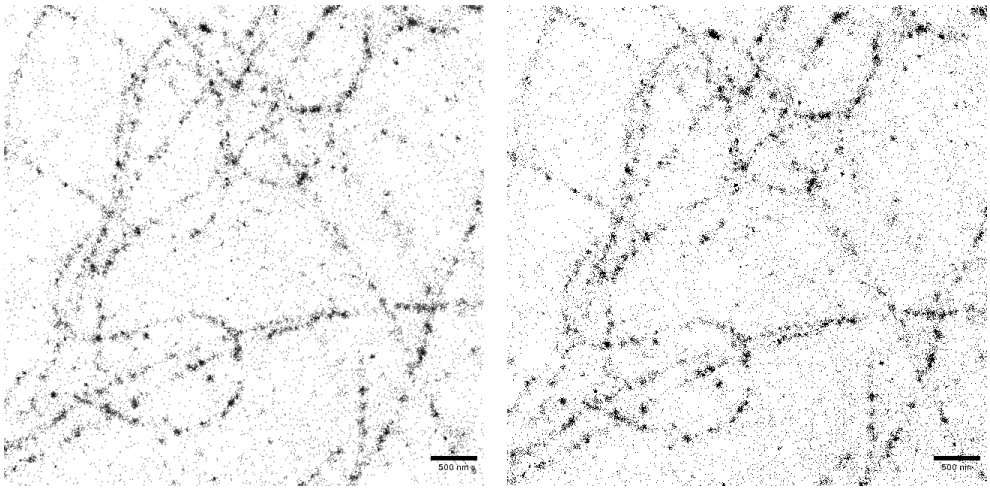
In Chapter 4, I have made many choices in algorithms and parameters that were neither obvious nor proven. Most of these choices can not be supported by traditional algorithmic theory, because all competing algorithms share the same complexity class or even, in some cases, the algorithm optimal in average runtime and space is inferior in real-time space and time complexity. Hence, I have checked most choices experimentally against a range of conceivable alternatives. This approach is, naturally, not exhaustive, but nevertheless a good indicator for the correctness of the involved methods and for the important qualities of the involved algorithms.

This chapter is divided into five sections: Comparing the implementation with the reference Schüttpelz implementation, checking the candidate search algorithm, checking the spot fitter algorithm, checking the σ estimation and evaluating the real-time capability of the system.

6.1 Evaluation by comparison with Schüttpelz implementation

The first and easiest test for a new software system is comparison of its results with existing systems. For this reason, I have tried to check the `rapidSTORM` results on specimen 1 with the results of the pre-existing dSTORM computation method.

To correlate the results, I have paired the localisations found by both methods with the pairing method described in section 5.4, allowing a maximum distance of 1 between the respective coordinates. From the total localisation count of 86839 for `rapidSTORM` and 104256 for the Schüttpelz implementation, 62110 localisations matched with an average Euclidean distance of 0.16 pixels, corresponding to 0.16 nm. These numbers do not account for spots fitted multiple times by the Schüttpelz implementation or for localization attempts rejected due to the amplitude threshold.



(a) `rapidSTORM` (b) Reference

Figure 6.1: Comparison of resulting images for specimen 1

The deviation in spot fitting and fit judging exhibited by these results is considerable. However, a lack of ground truth precludes any conclusion towards the correctness of either the `rapidSTORM` or the Schüttpelz results.

Comparing the images generated by both methods, similarity of the structures can be discerned despite different techniques for localisation visualisation. Figure 6.1 shows matching parts of the result images.[1]

6.2 Candidate search

The following principal questions with respect to the candidate search stage will be answered in this section: Firstly, is a dynamic threshold necessary? Secondly, which of several proposed smoothing algorithms is superior for this task? Thirdly, which of the two NMS operations introduced is superior? Fourthly, can kernel smoothing and a NMS compete with morphological operators such as reconstruction by erosion? And fifthly, what are the ideal parameters for the chosen method?

6.2.1 Necessity of dynamic threshold

Figure 6.2 shows the histogram of good and bad fit counts by candidate quality[2], measured on specimen 9. While the quality proves to be a good predictor of fit truth, it also becomes clear that any threshold choice must be made very precisely to separate true and false spots.

[1] Please note that the image taken from the Schüttpelz implementation was just meant as a first impression and would have been refined for further work or publication. This is not a comparison of imaging techniques, just of the structures that were recognised.

[2] The spot quality was defined in Chapter 4 as the value of the smoothed image at the candidate spot position.

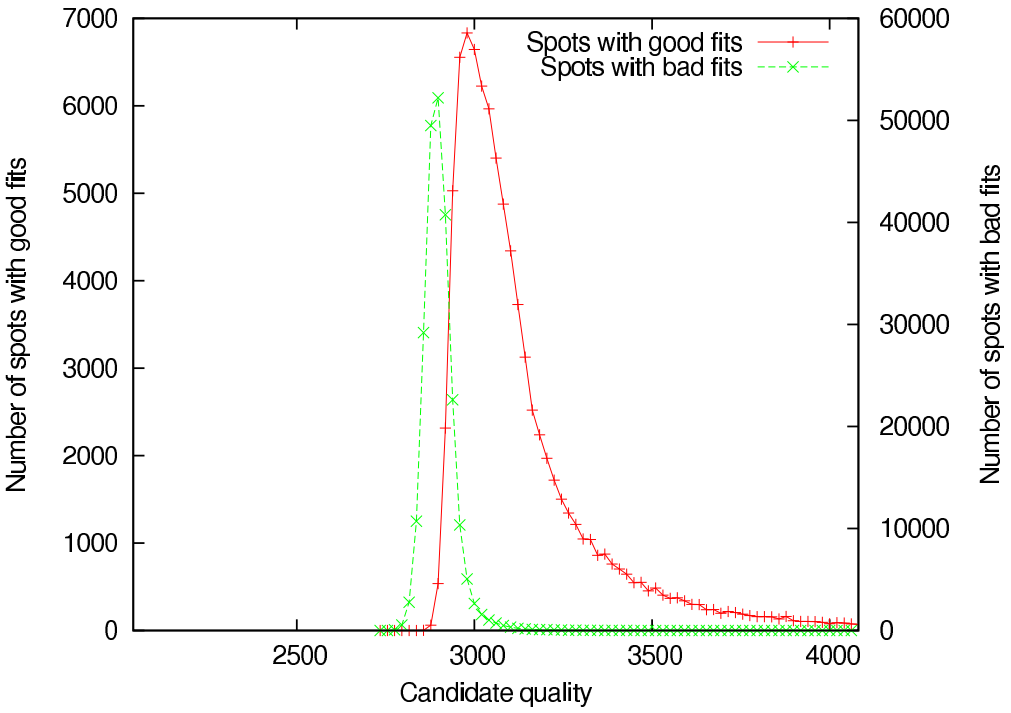


Figure 6.2: Histogram of fit goodness by candidate quality. The scale is chosen differently for good and bad fits to emphasise the principal problem of bad seperability.

6.2.2 Smoothing operations

I have compared three important smoothing methods: Smoothing by *Spalttiefpass*, smoothing by median and smoothing by a Gaussian kernel. I have chosen these methods since the *Spalttiefpass* is one of the fastest smoothing filters available, median smoothing might be useful for shot noise (which might be dominant in input stacks due to the influence of photon shot noise) and a Gaussian kernel due to the high similarity of its weights to the intensity distribution of true spots. The masks were all sized at $2m_s\sigma + 1$, and the Gaussian mask was weighted according to a Gaussian with SD σ to ideally match true spots.

In terms of theoretical execution speed, the *Spalttiefpass* and the Gaussian kernel can be expected to be very fast due to their decomposability, with the median filter able to match these speeds with a fast implementation [24]. In speed measurements on a randomly generated 128x128 image, the pure smoothing times were 0.2, 0.6 and 2.0 ms per image, respectively; the high computation time for the median filter probably stems from inefficient implementation.[3]

On the real specimens, the results of the smoothing filters are very hard to judge. Due to the lack of ground truth there is only one benchmark: The number of localisations found exceeding a certain threshold. Table 6.1 summarises the localisation counts and computation times for the real specimens with different amplitude thresholds for fit judging. It is obvious in these data that the median

[3] I have made no effort to speed up the median filter due to mediocre candidate search performance.

Sp.	θ_A	*Spalttiefpass*		Median		Gaussian	
no.	(ADC)	# loc.	t (s)	# loc.	t (s)	# loc.	t (s)
1	1000	160455	81.16	118876	315.67	425069	152.02
	4000	79300	46.36	75479	294.69	86949	47.73
	9000	52747	37.39	50885	280.70	55715	38.16
2	300	34621	37.74	29198	647.10	67928	53.45
	600	21089	33.24	21144	620.71	21028	37.23
	1200	17597	33.50	17708	636.51	17491	37.39
3	300	28176	30.08	18082	267.05	145556	61.14
	600	5390	21.48	4629	283.50	11035	22.68
	1200	2158	14.22	1983	289.07	2445	16.54
4	1000	15733	10.19	15575	277.32	16372	16.30
	4000	14652	9.65	14799	285.20	15075	15.76
	9000	13437	9.22	13592	287.85	13838	15.33
5	300	3479	2.37	3374	32.28	3499	2.33
	600	2258	2.15	2253	31.90	2262	2.30
	1200	1508	1.90	1509	31.64	1505	2.07
6	1000	2462	5.14	2486	35.66	3223	9.76
	4000	2196	5.12	2204	39.90	2237	5.55
	9000	2175	5.20	2176	35.06	2187	5.58
7	1000	12019	34.19	10712	1248.22	14930	43.76
	4000	6128	31.74	6282	1261.53	6276	42.32
	9000	10	17.37	11	252.53	9	37.69
8	1000	78848	63.19	67087	558.51	201709	106.90
	4000	50310	52.22	47698	516.65	62274	44.18
	9000	25766	32.81	27953	272.92	30459	42.08

Table 6.1: Localisation statistics on real specimens by smoothing algorithm. θ_A gives the amplitude threshold used for computation, and # loc. the number of localisations found in the specimen. The time given here is the processor time used by the `rapidSTORM` process and includes both startup, candidate search and spot fitting times. Therefore, the computation time listed is not a function of the number of localisations, but includes other influences such as stack size.

method is far inferior to the two other methods. It is also obvious that the Gaussian method mostly causes more localisations than the *Spalttiefpass* does; however, this effect is mostly visible for the low thresholds, where it is unclear whether the additional spots are merely false positives.

On the stochastically generated sets, crucial ground truth information is available, since the positions of active fluorophores are known. In Table 6.2, I have compared the *Spalttiefpass* and the Gaussian using different statistics. This table lists not only the candidate statistics, which give the quality of a smoothing method assuming the spot fitting and judging stage to be perfect, but also the localisation statistics, which show the quality of cooperation with the real version of that stage.

It becomes clear that the Gaussian kernel is equivalent or slightly superior to the *Spalttiefpass* if a large amplitude threshold makes false positives in the candidate judging procedure unlikely. However, if the amplitude threshold was chosen

Sp. no.	θ_A (ADC)	candidates						localisations			
		Spalttiefpass			Gaussian			*Spalttiefpass*		Gaussian	
		TPi	FPi	FNi	TPi	FPi	FNi	TPi	FPi	TPi	FPi
9	1000	6.156	3.503	3.301	6.134	3.812	3.198	6.386	7.573	7.539	423.893
	2000	6.179	3.504	3.278	6.123	3.804	3.209	6.290	2.763	7.073	69.679
	3000	6.179	3.504	3.278	6.134	3.812	3.198	6.098	0.960	6.520	13.668
	4000	6.080	3.479	3.276	6.134	3.812	3.198	5.678	0.402	5.928	4.210
	9000	6.124	3.496	3.238	6.117	3.856	3.269	3.515	0.008	3.419	0.054
10	1000	5.770	3.483	3.670	5.589	3.747	3.900	5.973	7.335	6.856	350.375
	4000	5.739	3.473	3.745	5.484	3.723	3.870	5.294	0.395	5.352	3.584
	9000	5.772	3.484	3.668	5.586	3.821	3.854	3.206	0.009	3.136	0.059
11	1000	13.627	1.126	80.350	16.224	0.165	77.846	13.832	0.318	16.233	0.149
	4000	13.627	1.126	80.350	16.471	0.174	77.598	13.734	0.284	16.284	0.116
	9000	13.677	1.121	80.396	17.957	0.183	76.046	12.867	0.219	15.593	0.078
12	1000	4.368	4.047	5.065	4.426	4.120	5.032	5.126	20.022	6.778	419.621
	4000	4.420	3.726	4.994	4.542	4.011	4.794	3.775	0.390	4.367	4.133
	9000	4.357	3.712	5.048	4.439	4.118	4.934	0.054	0.003	0.334	0.047
13	1000	5.595	3.369	3.814	5.600	3.387	3.809	5.851	13.517	6.706	394.745
	4000	5.641	3.356	3.832	5.643	3.402	3.766	4.847	0.054	4.808	0.372
	9000	5.592	3.368	3.817	5.595	3.388	3.784	2.390	0.0	2.357	0.000
16	1000	6.745	3.184	2.731	6.740	3.174	2.736	6.722	2.014	6.940	26.177
	4000	6.745	3.184	2.731	6.740	3.174	2.736	4.780	0.000	4.736	0.000
	9000	6.686	3.188	2.702	6.689	3.193	2.699	2.365	0.0	2.359	0.0
17	1000	7.207	3.090	2.184	7.213	3.083	2.178	6.838	0.001	6.780	0.001
	4000	7.207	3.090	2.184	7.212	3.083	2.179	4.637	0.000	4.628	0.000
	9000	7.207	3.090	2.184	7.212	3.083	2.179	2.283	0.0	2.281	0.0
18	1000	6.279	3.256	3.093	6.348	3.220	3.024	6.392	7.198	6.835	112.876
	4000	6.279	3.256	3.093	6.347	3.224	3.026	4.750	0.000	4.648	0.000
	9000	6.279	3.256	3.093	6.339	3.220	3.033	2.218	0.0	2.195	0.0

Table 6.2: Localisation statistics on the generated specimens by smoothing algorithm. θ_A gives the amplitude threshold used for computation, and TPi, FPi and FNi give the true positive, false positive and false negative count per image, respectively. As defined in section 4.2.1, the candidates shown here are the potential spot locations where fitting commenced, while the localisations are presumed fluorophore locations. The localisation count surpasses the candidate count because of the different counting methods described in section 5.4.

small, the Gaussian kernel's bias towards strong, single pixels (due to the strong weight for the central pixels) causes a large amount of false localisations. The spot finder statistics show that these false localisations were not due to a bad candidate search, but rather due to a high false positive rate in the spot fitter. This is not surprising when considering that the Gaussian mask has a larger weight for the centre pixel than the *Spalttiefpass* and thus gives a higher weight to single strong pixels than the *Spalttiefpass* does. In the spot fitting stage, a single strong pixel with surrounding noise is likely to cause a false positive[4], thereby raising the false positive rate. I consider this effect the cause of the high false positive rate.

Special attention should be directed towards specimen 11, which showed an extremely high spot density in the images. In this case, it can be seen how the smaller smearing effect of the Gaussian function (compared to the *Spalttiefpass*) results in a drastically better candidate recognition for very close fluorophores.

Since low threshold values are very likely to occur under lab conditions[5], the false positive sensitivity of the Gaussian mask is a major issue. For this reason, I deem the *Spalttiefpass* to be preferable for practical use, with the Gaussian smoothing having good future applications if the amplitude threshold can be determined automatically and better experimental techniques can raise spot densities.

6.2.3 Non-maximum suppression comparison

In Chapter 3, two principal approaches to the inclusive NMS have been presented: Firstly, the extension of Neubeck's algorithm for exclusive NMS, and secondly, comparing the image with its dilation.

The comparison between these methods was done by implementing both and comparing on specimens 1 and 4 at an amplitude threshold of 4000. These two specimens were selected to represent both high and low noise conditions. The Neubeck implementation performed better on both specimens, resulting in a total computation time of 46.4 seconds on specimen 1 and 10.6 seconds on specimen 4. The dilation version took 50.9 and 17.7 seconds, respectively.

6.2.4 Morphological operators for spot finding

Since I assume the Gaussian to be inferior due to its vulnerability to single pixel noise, I have tested morphological form operators, which are very tolerant to such noise. The first is a simple erosion with a structuring element $m_s\sigma$ pixels *wide*, thereby half as large as the usual SE and half as large as the usual spot centre.

Considering more advanced morphological operators, I implemented a fillhole scheme proposed by Marko Tscherepanow that uses morphological reconstruction. First, a fillhole transformation (see 3.1.4) was performed on the source image to suppress the noise minimums; secondly, a large (rectangular mask 25 pixels wide) opening produced a background image, which was subtracted from the result of

[4] Any fit attempt that uses a low amplitude to fit to a single strong pixel will have very high residues due to the large error for the single strong pixel. Therefore, single strong pixels tend to cause higher amplitudes in the fit results, generating false positives.

[5] After all, every experimenter has the desire to squeeze every localisation possible from his or her sample.

Sp.	θ_A	candidates									localisations					
		STP			Erosion			Fillhole			STP		Erosion		Fillhole	
no.	(ADC)	TPi	FPi	FNi	TPi	FPi	FNi	TPi	FPi	FNi	TPi	FPi	TPi	FPi	TPi	FPi
9	1000	6.16	3.50	3.30	5.73	3.37	3.70	5.75	3.40	3.68	6.39	7.57	6.31	28.85	6.28	20.16
	2000	6.18	3.50	3.28	5.73	3.37	3.70	5.77	3.40	3.68	6.29	2.76	5.95	4.28	5.97	3.45
	3000	6.18	3.50	3.28	5.72	3.39	3.72	5.74	3.43	3.63	6.10	0.96	5.68	0.47	5.70	0.43
	4000	6.08	3.48	3.28	5.71	3.38	3.68	5.74	3.43	3.63	5.68	0.40	5.47	0.06	5.48	0.05
	9000	6.12	3.50	3.24	5.67	3.39	3.70	5.71	3.43	3.66	3.52	0.01	3.46	0.00	3.46	0.00
10	1000	5.77	3.48	3.67	5.34	3.37	4.11	5.39	3.40	4.06	5.97	7.33	5.91	28.57	5.92	20.06
	4000	5.74	3.47	3.75	5.36	3.38	4.12	5.39	3.40	4.09	5.29	0.40	5.11	0.06	5.13	0.05
	9000	5.77	3.48	3.67	5.38	3.37	4.06	5.39	3.40	4.09	3.21	0.01	3.23	0.0	3.15	0.00
11	1000	13.63	1.13	80.35	27.69	2.98	66.29	26.54	4.31	67.52	13.83	0.32	20.10	0.16	19.76	0.16
	4000	13.63	1.13	80.35	27.69	2.98	66.29	26.52	4.30	67.54	13.73	0.28	19.34	0.12	19.05	0.14
	9000	13.68	1.12	80.40	27.78	2.97	66.69	26.50	4.27	67.43	12.87	0.22	16.36	0.09	16.30	0.10
12	1000	4.37	4.05	5.06	3.69	3.54	5.64	3.76	3.60	5.57	5.13	20.02	4.64	31.19	4.57	20.73
	4000	4.42	3.73	4.99	3.77	3.56	5.68	3.81	3.60	5.60	3.78	0.39	3.37	0.06	3.37	0.05
	9000	4.36	3.71	5.05	3.70	3.63	5.62	3.75	3.65	5.57	0.05	0.00	0.00	0.00	0.00	0.00
13	1000	5.59	3.37	3.81	4.87	3.41	4.60	4.94	3.45	4.53	5.85	13.52	5.54	31.53	5.53	21.21
	4000	5.64	3.36	3.83	4.82	3.40	4.59	4.88	3.44	4.53	4.85	0.05	4.53	0.02	4.58	0.02
	9000	5.59	3.37	3.82	4.82	3.41	4.56	4.80	3.44	4.50	2.39	0.0	2.42	0.00	2.41	0.0
16	1000	6.75	3.18	2.73	6.98	3.27	2.50	6.94	3.29	2.54	6.72	2.01	6.89	1.11	6.88	1.18
	4000	6.75	3.18	2.73	6.98	3.27	2.50	6.94	3.29	2.54	4.78	0.00	4.87	0.00	4.87	0.00
	9000	6.69	3.19	2.70	6.92	3.26	2.47	6.88	3.29	2.51	2.37	0.0	2.41	0.0	2.41	0.0
17	1000	7.21	3.09	2.18	6.93	3.18	2.47	6.95	3.16	2.44	6.84	0.00	6.82	0.00	6.84	0.00
	4000	7.21	3.09	2.18	6.93	3.18	2.47	6.95	3.16	2.44	4.64	0.00	4.74	0.00	4.75	0.00
	9000	7.21	3.09	2.18	6.93	3.18	2.47	6.95	3.16	2.44	2.28	0.0	2.31	0.0	2.31	0.0
18	1000	6.28	3.26	3.09	5.66	3.33	3.71	5.71	3.37	3.66	6.39	7.20	5.98	10.36	6.01	7.87
	4000	6.28	3.26	3.09	5.66	3.33	3.71	5.71	3.37	3.66	4.75	0.00	4.79	0.00	4.79	0.00
	9000	6.28	3.26	3.09	5.66	3.33	3.71	5.75	3.35	3.65	2.22	0.0	2.26	0.0	2.35	0.00

Table 6.3: Localisation statistics for fillhole scheme on generated specimen. θ_A gives the amplitude threshold used for computation, and TPi, FPi and FNi give the true positive, false positive and false negative count per image, respectively. As defined in section 4.2.1, the candidates shown here are the potential spot locations where fitting commenced, while the localisations are presumed fluorophore locations. The localisation count surpasses the candidate count because of the different counting methods described in section 5.4.

the fillhole transformation to isolate the spots on an otherwise black background. In a third step, the spot image was eroded by a rectangular 3x3 mask to erase noise maximums. This image was then subjected to a non-maximum suppression to find single maximums.

Considering the large mask sizes and complex operations included in the fillhole scheme, efficient implementation is crucial. While the fillhole transformation was performed by Tscherepanow's very fast reconstruction implementation of Luc Vincent's algorithm[55], the dilation and erosion operators were implemented using Gil and Werman's algorithm.[15]

Subsequently, I compared this scheme with searching local maximums in a smoothed image. The results on real specimens were inconclusive and can be found in the appendix in Table D.3 on page 101. The morphological operators sometimes caused more localisations to be found than the *Spalttiefpass*, sometimes less, and took significantly more time; however, without reliable data about the false positive rate, no conclusions should be drawn.

Table 6.3 compares the more precise statistics possible on generated data. On the normal specimens, the morphological operators clearly find far less true spots and true positive localisations than the *Spalttiefpass* does. On specimen 11, which features very many and barely separated spots, the morphological schemes perform best, even better than the Gaussian kernel. Notwithstanding the high computational effort, both the erosion and even more so the fillhole scheme exhibit a remarkable tolerance towards false positive localisations for some threshold ranges. This good classification reliability might provide valuable ground truth data if employed correctly.

6.2.5 Parameter choice

Sections 6.2.2 and 6.2.4 showed that a *Spalttiefpass* is the superior choice for candidate search amongst the solutions I have considered. However, the proportionality factor m_s for the size of the *Spalttiefpass* kernel matrix remains to be determined. As I have elaborated in section 4.3.1, the mask size may neither be too large nor too small to maximise the relative weight of the spot contribution.

I measured the impact of changes in m_s for the generated specimens and summarised important results in Table 6.4, with the complete results in Table D.4 in the Appendix on page 102. For the high-noise specimens the choice of 1.5 seems to be a consistently good value for m_s, while the low-noise specimens are almost indifferent to the smoothing mask size. Specimen 11 is once again a special case where the blurring effect of a large average mask is detrimental.

6.3 Spot fitting and judging stage

The spot fitting and judging stage is as important as the spot finding, but has already received substantial attention from Thompson [53] and Cheezum [9]. Therefore, I did not devote as much experimental time to it. Nonetheless, it is important to have an estimate of the spot judging's reliability for different choices of the amplitude threshold parameter θ_A to demonstrate the viability of an ampli-

Sp.	θ_A		candidates			localisations	
no.	(ADC)	m_s	TPi	FPi	FNi	TPi	FPi
9	4000	1.0	7.09	3.60	3.28	5.92	2.04
	4000	1.5	6.80	3.48	3.28	5.68	0.40
	4000	2.0	5.66	3.69	4.28	4.50	0.10
10	4000	1.0	6.59	3.60	3.83	5.42	1.97
	4000	1.5	6.46	3.47	3.75	5.29	0.40
	4000	2.0	5.74	3.65	4.26	4.42	0.09
11	4000	1.0	22.39	1.08	71.44	20.34	0.15
	4000	1.5	13.66	1.13	80.35	13.73	0.28
	4000	2.0	3.35	2.56	91.11	1.67	0.22
16	4000	1.0	7.12	3.19	2.75	4.75	0.00
	4000	1.5	7.09	3.18	2.73	4.78	0.00
	4000	2.0	6.83	3.34	2.91	4.84	0.00

Table 6.4: Effects of smoothing mask size on spot and localisation error rates. I have selected specimens 9–11 to reflect different spot densities and specimen 16 to include a different noise distribution. The results for other fluorophores can be found in Table D.4. θ_A gives the amplitude threshold, m_s the size factor for the smoothing mask, while TPi, FPi and FNi give the average TP, FP and FN count per image.

tude threshold and to be able to reproduce Thompson's results to demonstrate the correctness of the algorithm.

6.3.1 Error rates in spot fitting

I measured the false positive and the false negative rate for the generated specimens by fitting to a single fluorophore that was located in the centre of an image $2m_f\sigma + 1$ pixels wide and high. This fluorophore had the standard dynamics; therefore, most images contained no signal at all and only noise, some images contained a weakly fluorescing fluorophore and some a strongly fluorescing one. The spot fitter was required to accept a spot whenever it emitted a minimum number of photons (30) and reject otherwise. Figure 6.3 shows the error rates measured on the generated samples. As expected, the rate of FPs amongst the localisations drops exponentially, making an amplitude threshold feasible. Moreover, the error rate can be predicted roughly by the variance of the noise, which can be measured easily from acquired data.

6.3.2 Amplitude threshold choice

Table 6.2 shows a sharp decline of false positive rates in a range of spot amplitude thresholds between 1000 to 4000 A/D counts. A good amplitude threshold should suppress most false positives and therefore be chosen near the upper end of this spectrum; this consideration would indicate a good threshold value of 3000-4000 A/D counts. On the other hand, the choice of an amplitude threshold on the basis of these simulations is doubtful because false positive rates cannot be confirmed

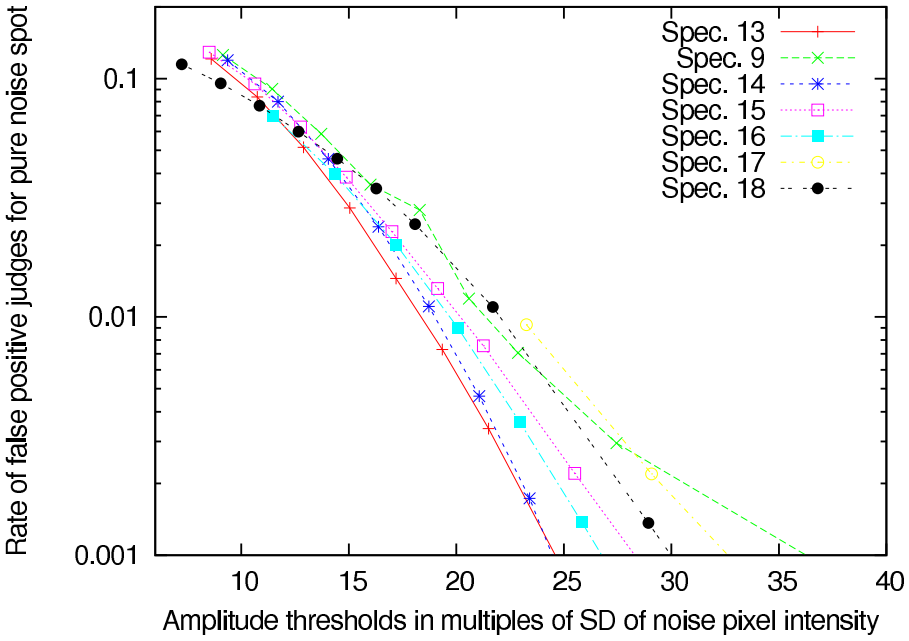


Figure 6.3: Error rates on generated samples by threshold. It becomes obvious that the error rate can be predicted fairly precisely by the standard deviation of the pixel noise. The noise extraction was performed identically as described in section 5.3.1. Specimens 10-12 were left out because their noise is identical to specimen 9.

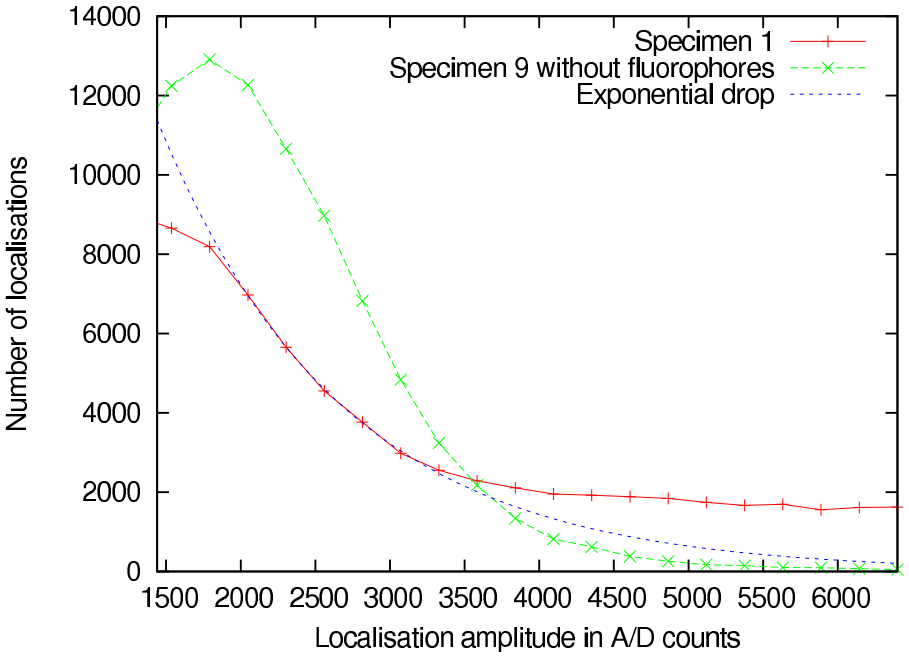


Figure 6.4: Localisation amplitude histogram on real specimen and noise. Note that specimen 9 is a simulation using the noise measured from specimen 1, ensuring comparability.

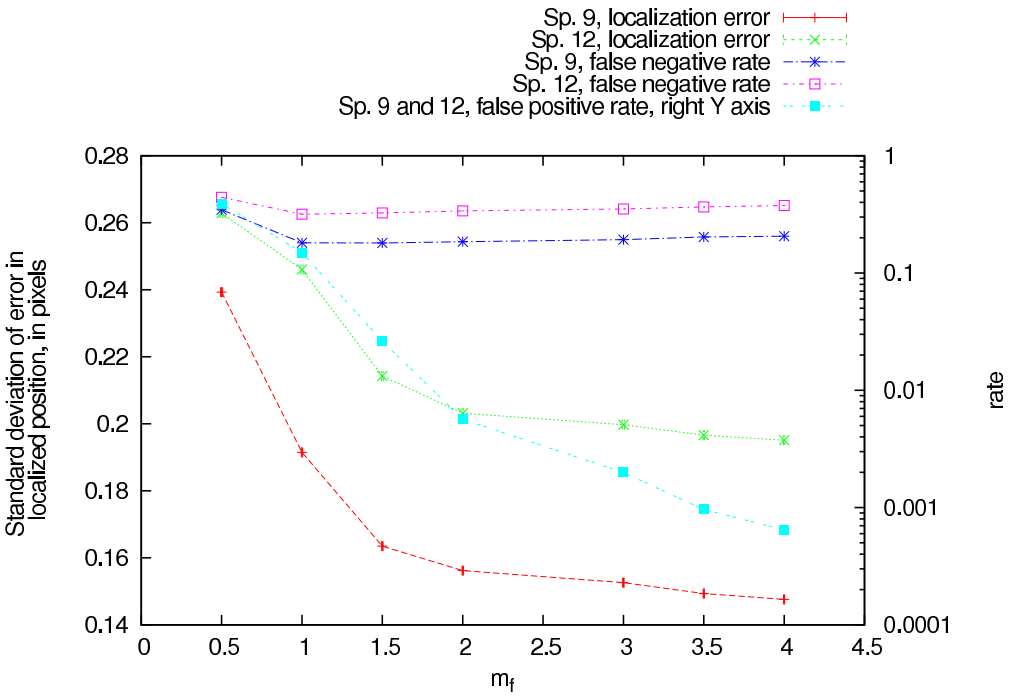


Figure 6.5: Effects of varying spot fitting mask size on spot fitting and fit judging error. Note how an increasing mask size improves reliability and accuracy. Specimens 9 and 12 were selected to represent two different photon emittance rates.

on real samples.

However, the decline of noise spots should manifest itself in the histogram of localisation amplitudes as a transition point between an exponential drop and a less steep part. The exponential drop should be due to false positives, which can be expected to occur with exponentially decreasing probability; and the less steep part would indicate true positives, which should exhibit a significantly different probability distribution.

Figure 6.4 shows the localisation amplitude histogram of specimen 1 and a corresponding histogram for noise measured on specimen 1 without any fluorophores. These data show a transition from one roughly exponential behaviour to a different trend; the histogram acquired from generated noise does not. Moreover, this transition happens in the region of 3000-4000 A/D counts, as predicted. This supports the hypothesis that an amplitude threshold found in simulations can be applied to real specimens.

6.3.3 Fit window size

The spot fitting module, which computes localisations from spots, operates on rectangular windows with a size proportional to the current value of σ. The choice of this window size's proportionality factor m_f is critical for both the speed and the precision of the spot fitting process and thereby the entire software. Since the number of pixels that the model function is fitted to grows quadratically with m_f,

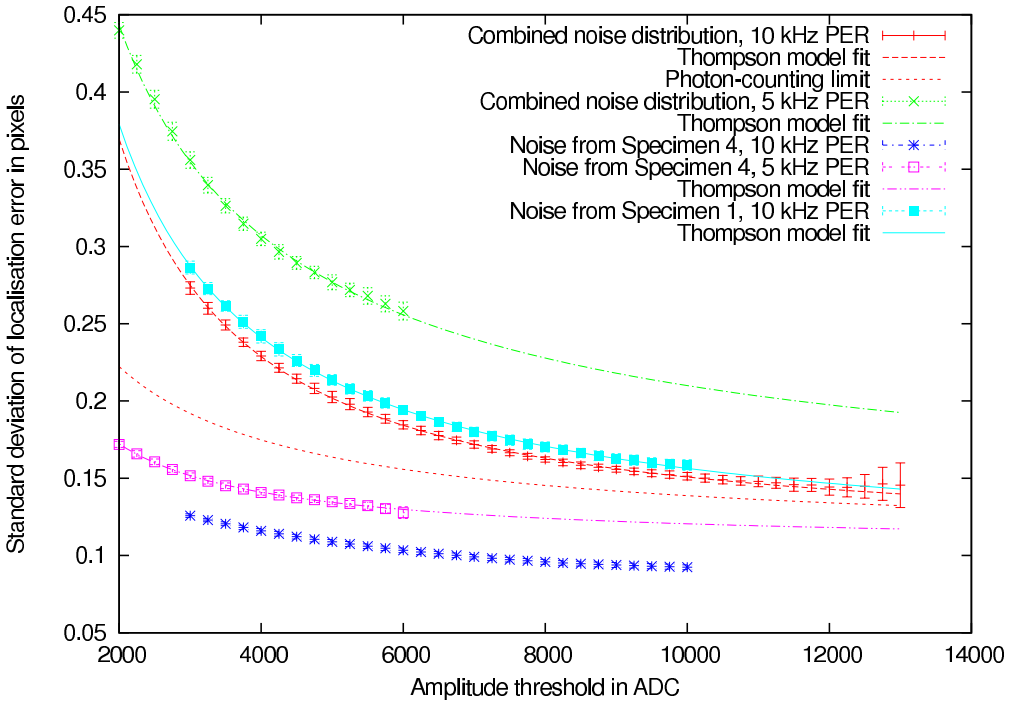


Figure 6.6: Change of localisation precision with threshold and noise. Data points are missing where a low PER made threshold exceedance unlikely.

the computational complexity of the fit grows quadratically with m_f, requiring its value to be chosen as low as possible. On the other hand, the probability of a false positive judge on a fit should drop drastically with higher values for m_f, as well as the precision of the fit should rise due to the inclusion of a larger sample of noise pixels surrounding the spot.

To test this effect, I measured the effect of several mask sizes on spot fitting and shown the results in Figure 6.5. It is surprising how even very large fitting masks[6] still show an improvement in fitting and judging precision. However, it also shows a value of 2 is a viable choice under normal conditions.

6.3.4 Spot fitting precision on generated data

With the same fitting method as in section 6.3.1, I measured the SD of the error in fitting a single spot. The results are shown in Figure 6.6. I have added fits of the Thompson model

$$\left\langle (\Delta x)^2 \right\rangle = \frac{a}{N} + \frac{b}{N^2} \tag{6.1}$$

(with N being the number of photons in a spot and a and b constants motivated and calculated physically in [53]) plus an additive constant to these data where ac-

[6] At normal dSTORM σ values of 2, an m_f of 4 will cause a window 19 pixels wide and high to be fitted.

Sp.	Integration time (ms)	X SD (nm)	Y SD (nm)
5	1	14.0	15.0
4	100	6.3	6.0
6	1000	12.0	12.0

Table 6.5: Localisation precision of real quantum dot specimens. X/Y SD gives the statistical standard deviation of the localisation position in X/Y direction. This SD should be proportional to a linear combination of the inverse and the inverse square of the number of emitted photons, which should be proportional to the integration time. The exhibited deviation from this theory hints at the fluorophore drift effect.

ceptable fit parameters were found.[7] The agreement with the Thompson model is excellent in these cases, and the additive constant I needed to add can be explained with the error inherent in fitting a Gaussian function to a Besselian PSF.

The specimen stacks I used were acquired with pixel sizes between 71 and 85 nm. Therefore, the values acquired here are in good agreement with those published for dSTORM in [21].

6.3.5 Localisation precision on real data

To ensure the applicability of the fit precision results from section 6.3.4 to the real specimen, I have tried to reproduce the results of Heilemann *et al.* [21] using the quantum dot specimens with numbers 4-6. The quantum dots in these specimens should, due to their high photon emittance rate (PER) and continuous activity, allow a very high localisation precision, at least as good as the predictions shown in Figure 6.6.[8] The localisation precision should, according to Thompson [53], improve either with the inverse or the inverse square of the photon number (which is proportional to the integration time), depending on the type of noise dominant under the conditions. Table 6.5 summarises the results.

The results are far from the expectations. Not only are the predictions from simulated data far more optimistic than these data imply, but the localisation precision for 1000 ms integration time is almost equal to that for 1 ms. This implies that there must be an additional effect overlaying the pure noise effects considered in Thompson's work. A time-resoluted view onto the localisations found for a single quantum dot, shown in Figure 6.7, supports this hypothesis: The quantum dot is slowly and erratically drifting from its original position. When this movement is roughly linear for short times, it can be guessed that the localisation precision is definitely higher than indicated by the standard deviation.[9] This move-

[7] The fits to several data sets yielded negative parameters for either a or b, which would be unphysical. For the data sets with 5 kHz photon rate, this fit failure can be explained by an underestimation of the fit error due to rejected fit positions. For the data set with 10 kHz photon rate and noise from specimen 4, I cannot explain the fit failure.

[8] Actually, this high photon count is counteracted by the non-negligible size of a quantum dot. Therefore, an additive constant should arise in equation 6.1.

[9] This is, naturally, only a rough estimate, and it seems statistically unsafe to draw conclusions for the localisation precision from the short time scales in which the drift velocity seemed to be stable.

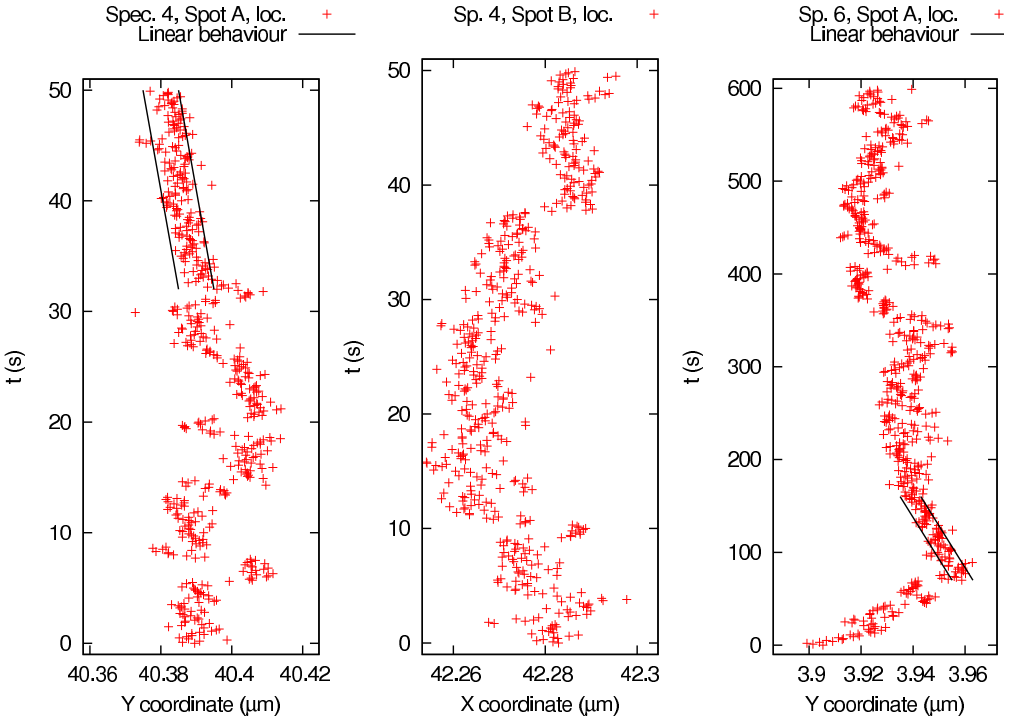


Figure 6.7: Spot time traces showing fluorophore drift. Each data point shows a localisation of the same quantum dot, while the three graphs show three different quantum dots. The black lines indicate sections where the drift was roughly linear and the true, better localisation precision becomes visible. The lines are spaced 10 nm in specimen 4 and 8 nm in specimen 6.

Measured quantity	θ_A (ADC)	Fixed $\sigma = 2$	Free σ	Fixed $\sigma = 1.8$
False negative rate (false negatives per true spot)	2000	0.12	0.19	0.12
	4000	0.24	0.29	0.27
	6000	0.37	0.41	0.4
False positive rate (false positives per 1000 noise spots)	2000	28	68	28
	4000	0.96	6.3	0.96
	6000	0.16	0.52	0.18
Correlation between real and measured spot amplitude	2000	0.97	0.94	0.97
	4000	0.97	0.93	0.96
	6000	0.95	0.9	0.94
Localisation precision (SD of localisation error in pixels)	2000	0.33	0.34	0.31
	4000	0.26	0.26	0.24
	6000	0.21	0.21	0.2

Table 6.6: Comparison between fitting with fixed and variable σ parameter. θ_A gives the amplitude threshold. A fluorophore was, as in the other experiments, simulated with a PSF SD of 1.8 and fitted with a Gaussian function that had either fixed or free σ paramaters. The differences in performance support the choice of fixed σ paramaters for `rapidSTORM`.

ment reduced the localisation precision, and its effect increased with integration time, explaining the bad results for specimen 6. This effect has been mentioned by Zhuang *et al.* in [59], and has been observed in further specimens.

6.3.6 Fixing of σ parameter

To test the hypothesis that fixing of the σ parameter enhances the fit precision, I have repeated the measurements for spot judging error rates and localisation precision on generated data for a spot fitting implementation that also fitted the σ_x and σ_y parameters. The initial guesses for these parameters were set to the real values, and a fit was judged to be good when the amplitude threshold was exceeded and the measured value for both σ components were within ± 1 pixels of the real value.

Table 6.6 summarises the results. Obviously, fixing the fit parameters enhances fit spotting and judging precision and predicts the real number of photons better. However, these results are not surprising, since the photon distributions in the generated were generated by a perfect Besselian PSF. Since this is not necessarily given for real data, these results should be confirmed for real specimens.

6.3.7 Motivation parameter M_0

The motivation parameter M_0 is used in the `rapidSTORM` engine to control the dynamic threshold for spot candidate fitting. Algorithm 4 on page 34 will abort fitting spot candidates for a given image once M_0 consecutive bad spots yielded bad fits.

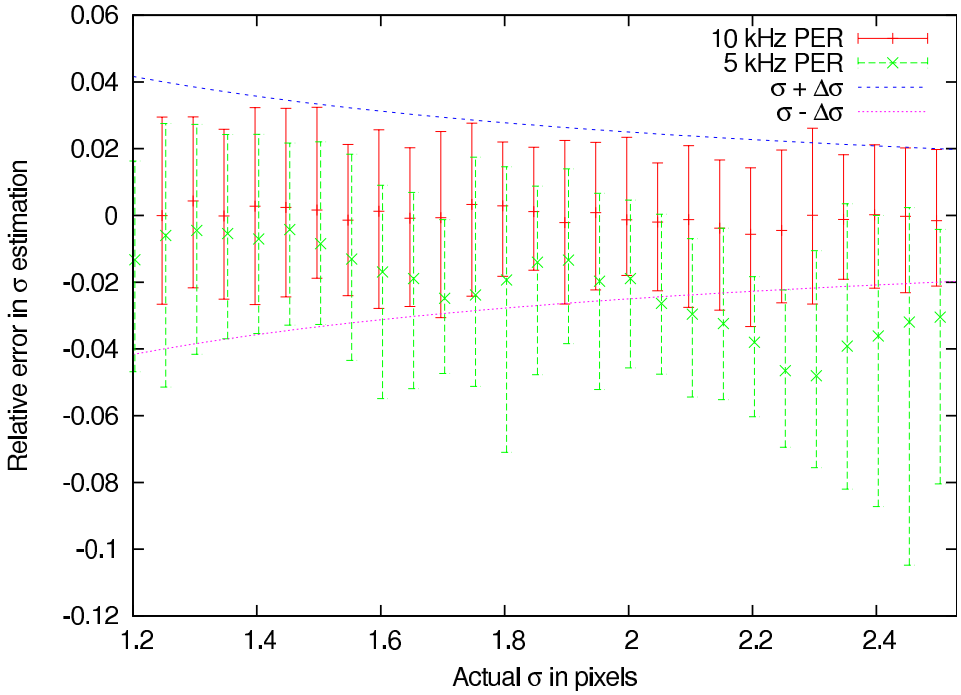


Figure 6.8: Average errors in σ estimation. The values in this graph show the average relative error, while the error bars indicate the minimal and maximal estimations. The blue and the violet line show the error allowed in σ estimation. The photon emittance rate was varied within the bounds given by Heilemann *et al.* in [19].

Naturally, high values for M_0 will result in some additional localisations compared to low values, but also in a much higher number of additional bad fits and in a small amount of additional FN localisations. To estimate these factors, I measured the number of localisations and the computation time needed at different values of M_0 and compiled the results in table D.5 on page 99. To summarise, it shows that a motivation parameter of 3 is a good choice at normal and high threshold levels with few additional localisations to be found with higher values, while at low threshold levels additional localisations are found with each rise in M_0. I deem the latter phenomenon to be caused by false positives. I have not tried to support this belief on generated data because the correct choice of a motivation parameter is naturally strongly dependent on the exact nature of the noise and on background signal, even more so than the other parameters.

6.4 σ estimator

Iterative heuristics are by their nature difficult and troublesome pieces of code. Therefore, special care must be taken to test these for accuracy, stability and convergence speed.

To test the accuracy and stability of the σ estimation code, I modified specimen 9 to use different PSF SDs and tried to estimate those from initial guesses that were

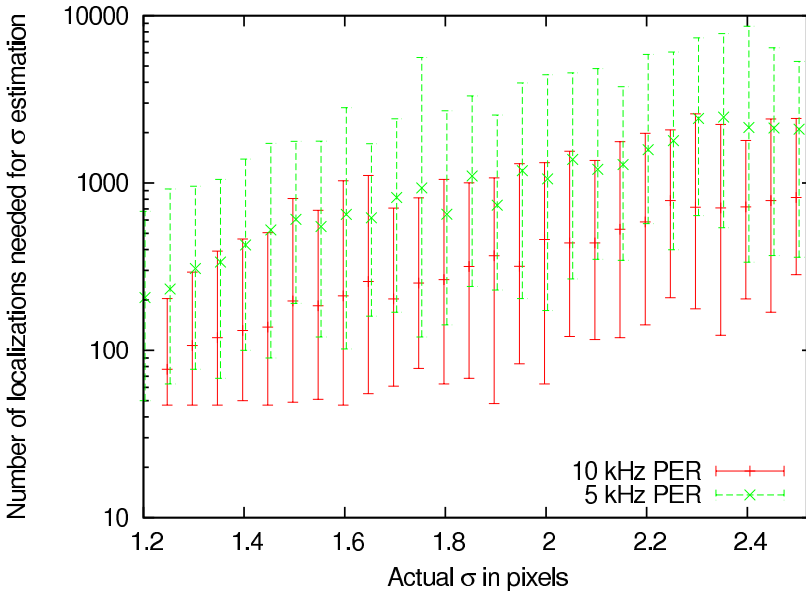


Figure 6.9: Number of localizations used for σ estimation. Error bars indicate minimum, average and maximum value measured in 50 runs on specimen 9.

off by a factor of $\sqrt{2}$. Consistently, the initial guesses for σ_x were chosen too low and the ones for σ_y were chosen too high. For each measured standard deviation, 50 independent samples have been taken at an amplitude threshold of 3000 and the final guesses for the σ parameters have been recorded. Figure 6.8 shows the results.

These results prove the overall fitness and stability of the standard deviation estimation. As evident in the graph, the values of the σ parameters can be guessed within the self-imposed limits of few percents of deviation. Furthermore, of the several thousand estimation attempts made for this graph, none failed to converge and none were off by more than 10%, as Tables D.2 and D.1 on page 100 prove. Since the SNR encountered with the high noise of specimen 9 and the very low PER of 5 kHz is amongst the worst encountered in photoswitching microscopy, the σ estimation algorithm can be expected to work for all photoswitching microscopy inputs.

To test the speed of convergence, I have recorded the number of localisations used for the measurements shown in Figure 6.8, including spots that were not fitted with a free-form Gaussian because of low amplitudes. Localisations that were used in multiple iterations were counted multiple times. Figure 6.9 shows the mean number of localisations needed for acceptance of a σ estimate with the error bars indicating the minimal and maximal value.

It becomes clear that a low SNR exacts fitting very many spots to reach an equilibrium state. This effect grows significantly with the PSF standard deviation. In extreme cases, the σ estimation process is rather unpredictable, with the minimum and maximum number of localisations needed spanning two orders of magnitude.

Sp.no.	θ_A (ADC)	Acq. time (s)	Processor time used (s)	Searching time (s)	Fitting time (s)
1	4000	400	47.0	1.1	43.0
1 ($m_f = 2$)	4000	400	29.7	1.6	25.0
2	600	200	38.6	0.2	30.2
3	1200	400	13.6	1.0	10.5
4	4000	50	10.6	0.6	6.7
5	300	1	2.7	0.1	2.7
6	4000	600	3.2	0.1	1.9
7	1000	5000	30.5	2.6	13.2
8	4000	800	39.3	1.5	35.2

Table 6.7: Comparison of computation and acquisition times. The processor time gives the total runtime of the software including startup and reading of the data files. Searching time is the time spent in candidate search, including smoothing and construction of the size-limited tree, while fitting time gives the time spent on fitting the found spot candidates. The difference between the sum of searching and fitting time and the total processor time probably stems from I/O operations.

6.5 Real-time computation

The experiments described in sections 6.2 to 6.4 were conducted to test the stability and accuracy of the `rapidSTORM` algorithm and implementation. This section tests whether the implementation can achieve real-time computation of dSTORM data and what the limits of real-time computability are.

Table 6.7 summarises the values already presented in Table 6.1. Please note that, in all cases, I have used processor time as the unit of measurement; for modern n-core processors, this number must, at maximum multithreading efficiency, be divided by n to give the wall clock time expended in computation. The multithreading efficiency for `rapidSTORM` computations has been at more than 99% for all real specimens on a 2-core processor, indicating very high multithreading capabilities and thereby large future speedup if the current trend to multi-core processors holds.

It is evident that most specimens were easily deconfined within the real-time domain. Given a 2-core processor, typical specimens such as specimen 1 can be deconfined in less than 25 seconds, which translates into roughly 3 ms per image, and choosing a mildly optimistical m_f value reduces this to less than 2 ms. This is well within real-time range for all photoswitching microscopy methods[10], and actually close to the maximum acquisition rates for many scientific cameras. It is therefore safe to declare `rapidSTORM` as real-time capable system.

[10] PALM uses frame rates of 0.5-1 seconds [3]; for STORM, the acquisition timings are not explicitly stated, but can be derived to be at least 50 ms from the Cy5 emittance rate and the photon counts in [43]; and for dSTORM, 10-100 ms [57] are common.

Chapter 7

Conclusion and Outlook

7.1 Conclusion

In this work, I have examined several aspects of the deconfinement problem of photoswitching microscopy. The existing deconfinement approaches of Betzig *et al.*, Schüttpelz, Thomann and Zhuang *et al.* have been summarised and brought to a coherent terminology. This terminology defines the way of a fluorophore emission from a source image over a *candidate spot*, which gives a likely position of fluorophore emission, and a *fit*, which gives the parameters of a PSF model fitting this spot, to a *localisation*, which is a definitive position and strength of a fluorophore emission.

Building on kernel operators as well as morphological image processing and on non-linear parameter fitting methods, I have found a real-time-capable algorithm for deconfinement. The key insight for this algorithm is the introduction of a motivational method for choosing a spot candidate threshold: Whether or not a spot candidate is fitted is not decided by some pre-chosen value, but by the success or failure of fit attempts to spot candidates with higher qualities.

I have chosen, implemented, tested and benchmarked several spot candidate detection schemes, including those mentioned in prior work. I have found that spot detection based on finding the maximum values in the source image after smoothing the noise by an average smoothing mask (*Spalttiefpass*) locates most fluorophore emissions, is tolerant towards noise and bad parameter choices and yields false positives which are easily discerned from true positives by later stages. With the size-limited merging binary tree, I have found a data structure that allows fast and efficient storage and sorting of the found candidates.

For fitting the PSF model to the candidate spots, I have demonstrated the use of the Levenberg-Marquardt scheme for non-linear least squares parameter fitting as implemented in the GNU Scientific Library. Using this scheme and analytically derived partial derivatives of the parameters, a fitting speed of less than 400 μs processor time per localisation was reached.

In addition, I have demonstrated how the assembly of the individual fluorophores into a target image can be realised concurrently to the deconfinement process and with enhanced contrast by using a series of caching tables for a weighted histogram equalisation.

Along with this work, I have produced a working and stable implementation for deconfinement called `rapidSTORM`. This implementation is capable of completely automatising the process of dSTORM photoswitching microscopy from image acquisition over deconfinement to image assembly with a minimum of user input, but is at the same time highly and easily configurable with a graphical user interface. In addition, this software is flexible and extensible by the means of a publishing interface for found localisations.

Regarding these aspects of terminology, candidate spot finding, spot fitting, fit judging and assembly of localisations into a target image, the real-time deconfinement for photoswitching microscopy has been advanced to a productive level.

7.2 Outlook

While the basic aspects of the deconfinement problem have been tackled, some facets remain questionable and further problems have been raised.

Firstly, Thomann's candidate search scheme based on the Hessian matrix was not implemented in this work. If the scheme can be simplified sufficiently, it might provide additional sensitivity to the candidate search.

Secondly, the fluorophore switching behaviour was greatly simplified in the simulations conducted to test this work. While this does not pose a problem at the moment, further work building on these simulations should carefully check whether this systematic error influences its findings.

Thirdly, the localisation precision tests done to compare fitting methods and parameters have been conducted in simulations only and are not necessarily true for data acquired from real specimens. Once the fluorophore drift problem is solved, these tests should be repeated to check the results.

The fourth point is the fluorophore drift problem itself. This problem is not so much a fault in the deconfinement, but rather the problem of systematic position changes of fluorophores during image acquisition. While Zhuang *et al.* [59] claim to have a solution, there is yet no published method that can detect and correct for fluorophore drift in software. This task is a, perhaps the, major task for deconfinement algorithms.

Fifthly and lastly, the usefulness of the developed algorithm could be enhanced if localisations stemming from the same, continuously active fluorophore could be connected, like Betzig *et al.* [3] did. Not only would this enhance image contrast by avoiding clusters of sometimes hundreds of localisations stemming from a single fluorophore, this information could also be used to enhance the localisation precision or to correct fluorophore drift.

All this leaves much to do in deconfinement. Hence, I would like to conclude this work as it has begun: with a motivation.

> For any man with half an eye,
> What stands before him may espy;
> But optics sharp it needs I ween,
> To see what is not to be seen.
>
> – From the poem McFingal by Trumbull [54]

Appendix A

Credits

Before thanking all the people who have helped me to conduct this work, I have to explain the unusual title of this chapter. I felt that the usual title, "Acknowledgements", does not properly express the gratitude I feel. I admit it is commonly used to express this feeling: but the connotations of acknowledging, which is often used synonymously with admitting or acquitting receipt, do not feel right. Rather, I used the term "credits" because it expresses how I owe my thanks to all of the people listed in this chapter.

First of all, I want to thank the people directly involved in this work. I express my gratitude to Prof. Dr. Markus Sauer and Prof. Dr. Franz Kummert for the interesting topic and the friendly and productive atmosphere in the working groups, and I owe much to my supervisors Dr. Mark Schüttpelz and Dr. Marko Tscherepanow for the advise and criticism that was crucial for transforming a quickly-hacked piece of software into a piece of scientific work. Further credit goes to the other members of the Applied Laser Physics and Laser Spectroscopy group, especially Sebastian van de Linde for many pictures and test runs on the acquisition system, Dr. Mike Heilemann for advice on several topics and my office colleagues Thomas Niehörster, Simon Hennig and Tobias Milde for many helpful comments. Last, but not least, I want to give kudos to Oliver Beine, Katja Heinig, Christian Mertes, Enis Poyraz, Hanna Radzey and Ralph Welsch, whose proof-reading greatly helped this work's comprehensibility.

Without detracting from the value of these contributions, I also want to express my gratitude to the people who have supported me personally during the course of this work. First amongst these are my parents, Michael and Helga Wolter, to whom I owe much more gratitude than this paper can hold and who have, through their emotional and financial support, made my studies and this work possible. My gratitude also goes to my beloved Katja Heinig, who has brightened my days with her love and smile, and who has given me the strength to tame the most meddlesome bits I have ever encountered. Furthermore, I want to thank my flatmates and friends for their patience and support.

Last, but not least, I want to thank all the people who worked on the huge system of public domain software available today. This work has greatly profited from the range of free software available, especially the GNU compiler suite, make and scientific library as well as from the `vim` editor, LaTeX and the countless programs that drive today's professional computing, and I'll try to give some of that use back into the upstream.

Bibliography

[1] Mark Bates, Timothy R. Blosser, and Xiaowei Zhuang. Short-range spectroscopic ruler based on a single-molecule optical switch. *Physical Review Letters*, 94(10):108101, 2005. Available from World Wide Web: http://link.aps.org/abstract/PRL/v94/e108101.

[2] Mark Bates, Bo Huang, Graham T. Dempsey, and Xiaowei Zhuang. Multicolor Super-Resolution Imaging with Photo-Switchable Fluorescent Probes. *Science*, 317(5845):1749–1753, 2007. Available from World Wide Web: http://www.sciencemag.org/cgi/content/abstract/317/5845/1749.

[3] 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. Available from World Wide Web: http://www.sciencemag.org/cgi/content/abstract/313/5793/1642.

[4] Norman Bobroff. Position measurement with a resolution and noise-limited instrument. *Review of Scientific Instruments*, 57(6):1152–1157, 1986. Available from World Wide Web: http://dx.doi.org/10.1063/1.1138619.

[5] Max Born and Emil Wolf. *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*. Pergamon Press, Cambridge, England, fifth edition, 1975.

[6] Albert Hosmer Bowker and Gerald J. Lieberman. *Engineering Statistics*. Prentice-Hall, Inc., Eaglewood Cliffs, New Jersey, second edition, 1972.

[7] Norbert Brändle, Horng-Yang Chen, Horst Bischof, and Hilmar Lapp. Robust parametric and semi-parametric spot fitting for spot array images. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 46–56. AAAI Press, 2000.

[8] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[9] Michael K. Cheezum, William F. Walker, and William H. Guilford. Quantitative comparison of algorithms for tracking single fluorescent particles.

Biophys. J., 81(4):2378–2388, October 2001. Available from World Wide Web: http://www.biophysj.org/cgi/content/abstract/81/4/2378.

[10] James W. Cooley, Peter A. W. Lewis, and Peter D. Welch. The fast fourier transform and its applications. *Education, IEEE Transactions on*, 12(1):27–34, March 1969.

[11] Unknown Editor. Method of the Year 2008. *Nature Methods*, 6(1):1–1, January 2009. Available from World Wide Web: http://www.nature.com/nmeth/journal/v6/n1/abs/nmeth.f.244.html.

[12] J. Doyne Farmer and John J. Sidorowich. Optimal shadowing and noise reduction. *Physica D*, 47(3):373–392, 1991.

[13] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Michael Booth, and Fabrice Rossi. *Gnu Scientific Library: Reference Manual*. Network Theory Ltd., February 2003. Available from World Wide Web: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0954161734.

[14] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Entwurfsmuster*. Addison-Wesley Verlag, München, erste edition, 1996.

[15] J. Gil and M. Werman. Computing 2-d min, median, and max filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):504–507, May 1993.

[16] Samuel Glasson. *Censored Regression Techniques for Credit Scoring*. PhD thesis, RMIT University, 2007.

[17] Image of Hawkes Bay, New Zealand, published by Phillip Capper under Creative Commons attribution license and modified by the anonymous Wikipedia user Konstable, who also created the histogram graphs. Sources are linked at: http://en.wikipedia.org/wiki/Histogram_equalization, June 2006.

[18] Stephen Hawking. *The Universe in a Nutshell*. Bantam Books, New York, 2001.

[19] M. Heilemann, E. Margeat, R. Kasper, M. Sauer, and P. Tinnefeld. Carbocyanine dyes as efficient reversible single-molecule optical switch. *J. Am. Chem. Soc.*, 127(11):3801–3806, March 2005. Available from World Wide Web: http://dx.doi.org/10.1021/ja044686x.

[20] Mike Heilemann. *Design of Single-Molecule Optical Devices: Unidirectional Photonic Wires and Digital Photoswitches*. PhD thesis, Bielefeld University, 2005. Available from World Wide Web: http://bieson.ub.uni-bielefeld.de/volltexte/2005/711/.

[21] 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. Available from World Wide Web: http://dx.doi.org/10.1002/anie.200802376.

[22] Stefan W. Hell. Far-Field Optical Nanoscopy. *Science*, 316(5828):1153–1158, 2007. Available from World Wide Web: http://www.sciencemag.org/cgi/content/abstract/316/5828/1153.

[23] Peter Høeg. *Smilla's Sense of Snow*. Farrar, Straus and Giroux, New York, 1993.

[24] T.S. Huang, G.J. Yang, and G.Y. Tang. A fast two-dimensional median filtering algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 27:13–18, February 1979.

[25] Masahiro Irie, Tuyoshi Fukaminato, Takatoshi Sasaki, Naoto Tamai, and Tsuyoshi Kawai. Organic chemistry: A digital fluorescent molecular photoswitch. *Nature*, 420(6917):759–760, December 2002. Available from World Wide Web: http://dx.doi.org/10.1038/420759a.

[26] Peter A. Jansson. Convolution and related concepts. In Peter A. Jansson, editor, *Deconvolution*, pages 3–37. Academic Press, Inc., Orlando, Florida, 1984.

[27] L.P. Jaroslavskij. *Einführung in die digitale Bildverarbeitung*. Hüthig, Heidelberg, 1990.

[28] Karatsuba, Catherine A. Fast evaluation of bessel functions. *Integral Transforms and Special Functions*, 1(4):269–276, 1993. Available from World Wide Web: http://www.informaworld.com/10.1080/10652469308819026.

[29] Charles Kingsley. *Glaucus*, volume 695 of *Project Gutenberg*. Project Gutenberg, P.O. Box 2782, Champaign, IL 61825-2782, USA, 1996. Available from World Wide Web: http://www.gutenberg.org/dirs/etext96/glcus10.zip.

[30] Reinhard Klette and Piero Zamperoni. *Handbuch der Operatoren für die Bildbearbeitung*. Vieweg, Wiesbaden, 1992.

[31] Donald E. Knuth. Notes on the van emde boas construction of priority deques: An instructive use of recursion. Letter to Peter van Emde Boas, March 1977.

[32] Donald E. Knuth. *The art of computer programming, volume 2: seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, third edition, 1997. Available from World Wide Web: http://portal.acm.org/citation.cfm?id=270146.

[33] Jennifer Lippincott-Schwartz and Suliana Manley. Putting super-resolution fluorescence microscopy to work. *Nature Methods*, 6(1):21–23, January 2009. Available from World Wide Web: http://www.nature.com/nmeth/journal/v6/n1/abs/nmeth.f.233.html.

[34] Stephen G. Lipson, David S. Tannhauser, and Henry S. Lipson. *Optik*. Springer-Verlag, Berlin, Germany, first edition, 1997.

[35] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, January 1998. Available from World Wide Web: http://dx.doi.org/10.1145/272991.272995.

[36] A. D. McNaught and A. Wilkinson. *IUPAC Compendium of Chemical Terminology*. Blackwell Science, 1997.

[37] D.B. Murphy. *Fundamentals of light microscopy and electronic imaging*. Wiley-Liss, New York, 2001.

[38] Alexander Neubeck and Luc Van Gool. Efficient non-maximum suppression. In *ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition*, pages 850–855, Washington, DC, USA, 2006. IEEE Computer Society.

[39] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, second edition, October 1992. Available from World Wide Web: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&path=ASIN/0521431085.

[40] Kevin Pulo. Simparm: A simple, flexible and collaborative configuration framework for interactive and batch simulation software. In *International Supercomputing Conference (ISC07)*, Dresden, Germany, June 2007.

[41] W.S. Rasband. *ImageJ*. U. S. National Institutes of Health, Bethesda, Maryland, USA, 1997-2008. Available from World Wide Web: http://rsb.info.nih.gov/ij/. Fluorescent cell image was published under GNU Public License.

[42] J. C. Russ. *The Image Processing Handbook*. CRC Press, Boca Raton, Florida, 1992.

[43] Michael J. Rust, Mark Bates, and Xiaowei Zhuang. Sub-diffraction-limit imaging by stochastic optical reconstruction microscopy (storm). *Nature Methods*, 3(10):793–796, August 2006. Available from World Wide Web: http://dx.doi.org/10.1038/nmeth929.

[44] Andrés Santos and Ian T. Young. Model-based resolution: Applying the theory in quantitative microscopy. *Appl. Opt.*, 39(17):2948–2958, 2000. Available from World Wide Web: http://ao.osa.org/abstract.cfm?URI=ao-39-17-2948.

[45] Markus Sauer. Reversible molecular photoswitches: A key technology for nanoscience and fluorescence imaging. *Proceedings of the National Academy of Sciences of the United States of America*, 102(27):9433–9434, 2005. Available from World Wide Web: http://www.pnas.org/content/102/27/9433.short.

[46] Mark Schüttpelz. Stormengine.m. Unpublished MATLAB file.

[47] Mark Schüttpelz. *Fluoreszenzspektroskopische Methoden zur Charakterisierung von Protein-Wechselwirkungen*. PhD thesis, Universität Bielefeld, 2006. Available from World Wide Web: http://bieson.ub.uni-bielefeld.de/volltexte/2006/1013/.

[48] F. Graham Smith, Terry A. King, and Dan Wilkins. *Optics and Photonics: An Introduction*. John Wiley & Sons Ltd, Chichester, England, second edition, 2007.

[49] Pierre Soille. *Morphological Image Analysis*. Springer-Verlag, Berlin, 2004.

[50] Mark M. Somoza. Franck-Condon diagram is licensed under Creative Commons attribution and share-alike license and can be found at http://commons.wikimedia.org/wiki/File:Franck-Condon-diagram.png, May 2006.

[51] Andor Technology. *iXon camera manual*. Andor Technology, 7 Millennium Way, Springvale Business Park, Belfast, BT12 7AL, NORTHERN IRELAND, 2008.

[52] Dominik Michael Thomann. *Algorithms for detection and tracking of objects with super-resolution in 3D fluorescence microscopy*. PhD thesis, ETH Zürich, 2003. Available from World Wide Web: http://e-collection.ethbib.ethz.ch/view/eth:26349.

[53] Russell E. Thompson, Daniel R. Larson, and Watt W. Webb. Precise Nanometer Localization Analysis for Individual Fluorescent Probes. *Biophys. J.*, 82(5):2775–2783, 2002. Available from World Wide Web: http://www.biophysj.org/cgi/content/abstract/82/5/2775.

[54] John Trumbull. *The satiric poems*. Univ. of Texas Pr., Austin, Texas, 1962.

[55] Marko Tscherepanow. *Image Analysis Methods for Location Proteomics*. PhD thesis, Bielefeld University, 2008. Available from World Wide Web: http://bieson.ub.uni-bielefeld.de/volltexte/2008/1384/.

[56] S. van de Linde, R. Kasper, M. Heilemann, and M. Sauer. Photoswitching microscopy with standard fluorophores. *Applied Physics B: Lasers and Optics*, 93(4):725–731, December 2008. Available from World Wide Web: http://dx.doi.org/10.1007/s00340-008-3250-9.

[57] Sebastian van de Linde, Ulrike Endesfelder, Anindita Mukherjee, Mark Schüttpelz, Gerd Wiebusch, Steve Wolter, Mike Heilemann, and Markus Sauer. Multicolor Photoswitching Microscopy for Subdiffraction-Resolution Fluorescence Imaging. *Photochemical & Photobiological Sciences*, 2009. Paper submitted and in review process.

[58] J. Widengren and P. Schwille. Characterization of photoinduced isomerization and back-isomerization of the cyanine dye cy5 by fluorescence correlation spectroscopy. *J. Phys. Chem. A*, 104(27):6416–6428, July 2000. Available from World Wide Web: http://dx.doi.org/10.1021/jp000059s.

[59] Xiaowei Zhuang, Wilfred M. Bates, Michael J. Rust, and Bo Huang. Sub-diffraction limit image resolution and other imaging techniques. United States Patent Application 20080182336, July 2008.

Glossary

G	Maximum grey value for image pixel, 47
M_0	Candidate fitting motivation, 33, 40, 58, 74, 86
R_e	Resolution enhancement factor, 44, 45
T_{HE}	Histogram equalization transfer function, 22
T_{WHE}	Transfer function for the weighted histogram equalization, 22, 48
heq_w	weighted histogram equalization, 22, 47
$\lfloor$	Gauss bracket giving the largest integer not larger than the bracketed number, 22, **22**, 45, 47, 48, 50
$\mathbf{D}'''$	Histogram equalization of $\mathbf{D}''$, 47
$\mathbf{D}''$	Preliminary result image produced from $\mathbf{D}'$ by scaling intensity values into a given range, 47–49
$\mathbf{D}'$	Preliminary result image that was discretized into pixels, but has unscaled intensity values, 44, 45, 47–49
$\mathbf{D}$	Result image of photoswitching microscopy, 32, 44, 45, 47–49
θ_A	Amplitude threshold, 63, 65, 67, 74, 77, 86
h_C	Cumulative brightness histogram function, 20, 22
h_w	Brightness histogram function, 22, 23, 48
$h_{C,w}$	Cumulative brightness histogram function, 22, 23
h	Brightness histogram function, 20–22, 57
m_f	Mask size for spot fitting, 42, 68, 70, 71, 77
m_s	Mask size for candidate search, 67, 86

ADC	A/D count: Unit of measurement for source image intensity, 53

classificator Function mapping a large space of pattern vectors onto few symbols, 28

confidence interval Interval containing with some given probability the stochastical mean of the distribution underlying a sample, 28

confinement Separation of close light sources in space or time. Is used to achieve higher resolutions via photoswitching microscopy, 1, **1**, 2

temporal confinement Separation of close light sources in time. Temporally confining methods for photoswitching microscopy activate fluorophores independently of position, but rather by some stochastical low-rate switching process, 2, 3

deconfinement Reversal of the confinement process. The deconfinement finds the confined fluorophore emissions in the source data and re-assembles them into a single, complete image of the specimen, **2**, 3

dSTORM direct stochastic optical reconstruction microscopy: Temporally confining photoswitching microscopy method using standard fluorophores, 10, 31, 36, 37, 45, 47, 52, 53, 60, 71, 72, 78

FN false negative: With respect to a classification symbol, a pattern that was not classified for that symbol even though it should have been, 28, 59, 67, 74

FP false positive: With respect to a classification symbol, a pattern that was falsely classified for that symbol, 28, 56, 59, 67, 68

histogram equalization Image operating enhancing contrast by intensity equidistribution, 21

image translation The translation of an image by a vector is the translation of all pixels by the given vector., 16

marker image In the context of geodesic dilations, the marker image is the image which is dilated in limits defined by the mask image, 17

mask image In the context of geodesic dilations, the mask image is the image which limits the dilation of the marker image, 17

morphological reconstruction Geodesical dilation/erosion iterated until stability is called morph. recon. by dilation/erosion, 18

NLLS nonlinear least-squares: Class of fitting problems, 24, 26, 43–45

NMS non-maximum suppression: Morphological operator finding local maximums of an image, 19, 20, 38–41, 58, 61, 65

opening A morphological operation consisting of erosion and subsequent dilation, 17

PDF probability density function:, 56

PER photon emittance rate: Average number of photons emitted by a fluorophore per second, 72, 76

photoswitching microscopy Collective term for microscopy methods that circumvent the Abbe diffraction barrier by using photoswitches, **1**, 2–4, 9, 11, 30, 44, 52, 76–79

PSF point spread function: Function giving the intensity distribution that a point light source generates on the detector of an optical device, 5, 6, 16, 31–33, 38, 39, 41–43, 52, 57, 74, 76, 78

resolution limit Minimal distance for resolvable light sources. This distance is dependant on the objective used and the wavelength of the measured radiation, 1

resolved An image is said to be resolved if the resolving power of the optical instrument is sufficient to discern between its details., **5, 6**

resolving power The capability of an optical instrument to discern between close light sources, 5, **5**, 6

SD standard deviation: Square root of sample variance, 27, 39, 42, 45, 56, 58, 61, 71, 72, 74, 75

SE structuring element: Set of vectors defining a neighbourhood for pixels, 16–20, 39

seperable matrix A seperable matrix can be expressed by the tensor product of two vectors, 14

SNR signal-to-noise ratio: Ratio between average power of signal and noise, 14, 52, 53, 57, 76

tensor product Product operator producing matrix from two vectors, 14

TN true negative: With respect to a classification symbol, a pattern that was not classified for that symbol correctly, 28

TP true positive: With respect to a classification symbol, a pattern that was correctly classified for that symbol, 28, 59, 67

unresolved An image is said to be unresolved if the resolving power of the optical instrument is insufficient to discern between its details, **5**

WHE weighted histogram equalization: Variant of histogram equalization featuring a weight parameter r, which can be used to scale the effects of histogram equalization, 48

Appendix B

Proof of recursion formula for sum of squares

In section 3.3.1, I have given the West recursion formula for the sum of squares in a sample. This chapter contains the complete proof for this formula.

In this proof, $\mathbf{x}$ denotes a sample that is extended by one observation o to the sample $\mathbf{y}$. $\overline{\mathbf{x}}$ denotes the mean of a vector, $M_2(\mathbf{x})$ the sum of squares for that vector.

$$\begin{aligned} M_2(\mathbf{y}) &= (o-\overline{\mathbf{y}})^2 + \sum_{n=1}^{N}(\mathbf{x}_n-\overline{\mathbf{y}})^2 \\ &= (o-\overline{\mathbf{y}})^2 + \sum_{n=1}^{N}\left((\mathbf{x}_n-\overline{\mathbf{x}})-(\overline{\mathbf{y}}-\overline{\mathbf{x}})\right)^2 \end{aligned}$$

and by using $\delta\overline{\mathbf{x}} = \overline{\mathbf{y}} - \overline{\mathbf{x}} = \frac{o-\overline{\mathbf{x}}}{N+1}$

$$= (o-\overline{\mathbf{y}})^2 + \sum_{n=1}^{N}\left[(\mathbf{x}_n-\overline{\mathbf{x}})^2 - 2\delta\overline{\mathbf{x}}(\mathbf{x}_n-\overline{\mathbf{x}}) + (\delta\overline{\mathbf{x}})^2\right]$$

and by using $\sum_{n=1}^{N}\mathbf{x}_n - \overline{\mathbf{x}} = 0$

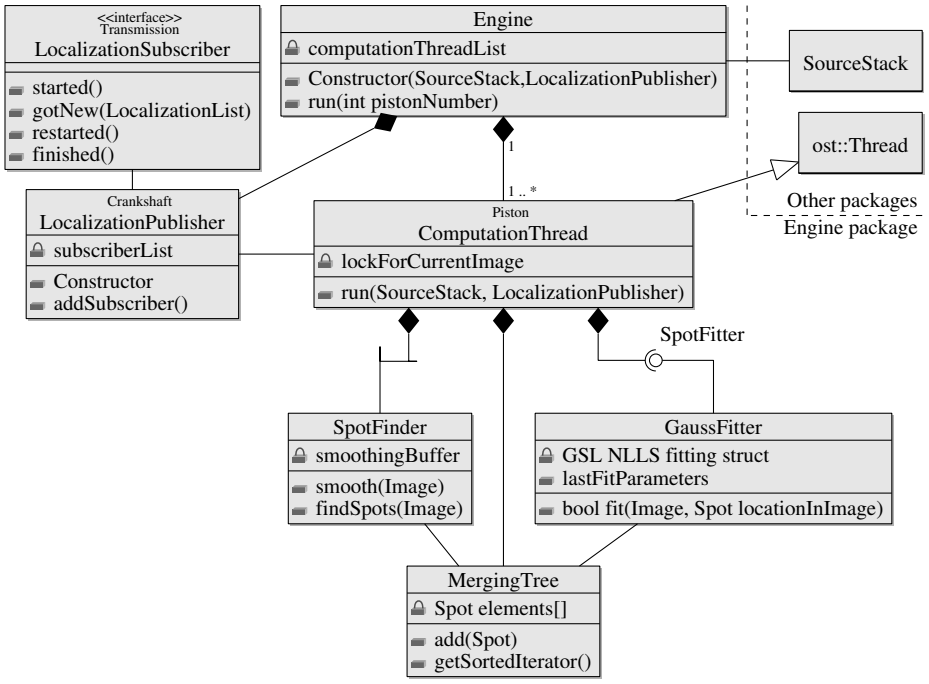
$$= (o-\overline{\mathbf{y}})^2 + \sum_{n=1}^{N}(\mathbf{x}_n-\overline{\mathbf{x}})^2 + N\cdot(\delta\overline{\mathbf{x}})^2$$

and by using $o-\overline{\mathbf{y}} = o-\overline{\mathbf{x}} - \frac{o-\overline{\mathbf{x}}}{N+1} = \frac{N}{N+1}(o-\overline{\mathbf{x}})$

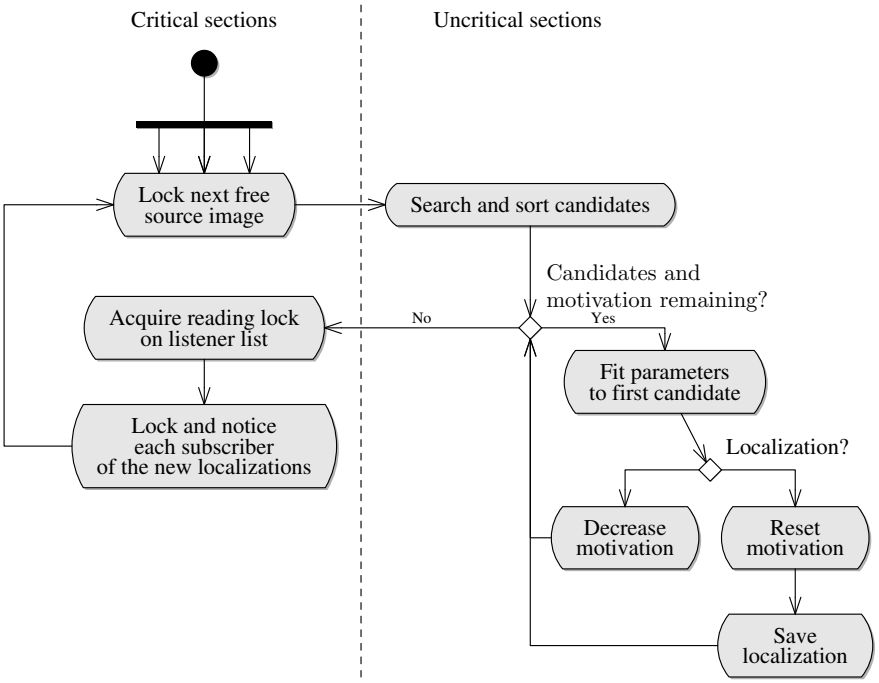
$$\begin{aligned} &= M_2(\mathbf{x}) + (o-\overline{\mathbf{y}})^2 + N\cdot\frac{(o-\overline{\mathbf{x}})^2}{(N+1)^2} \\ &= M_2(\mathbf{x}) + (o-\overline{\mathbf{y}})\left[(o-\overline{\mathbf{x}'}) + \frac{o-\overline{\mathbf{x}}}{N+1}\right] \\ \Leftrightarrow \quad & M_2(\mathbf{y}) = M_2(\mathbf{x}) + (o-\overline{\mathbf{y}})(o-\overline{\mathbf{x}}). \end{aligned} \tag{B.1}$$

Appendix C

Schematic overview over `rapidSTORM` system



(a) Class diagram



(b) Activity diagram

Figure C.1: UML diagrams for parallelized `rapidSTORM` engine

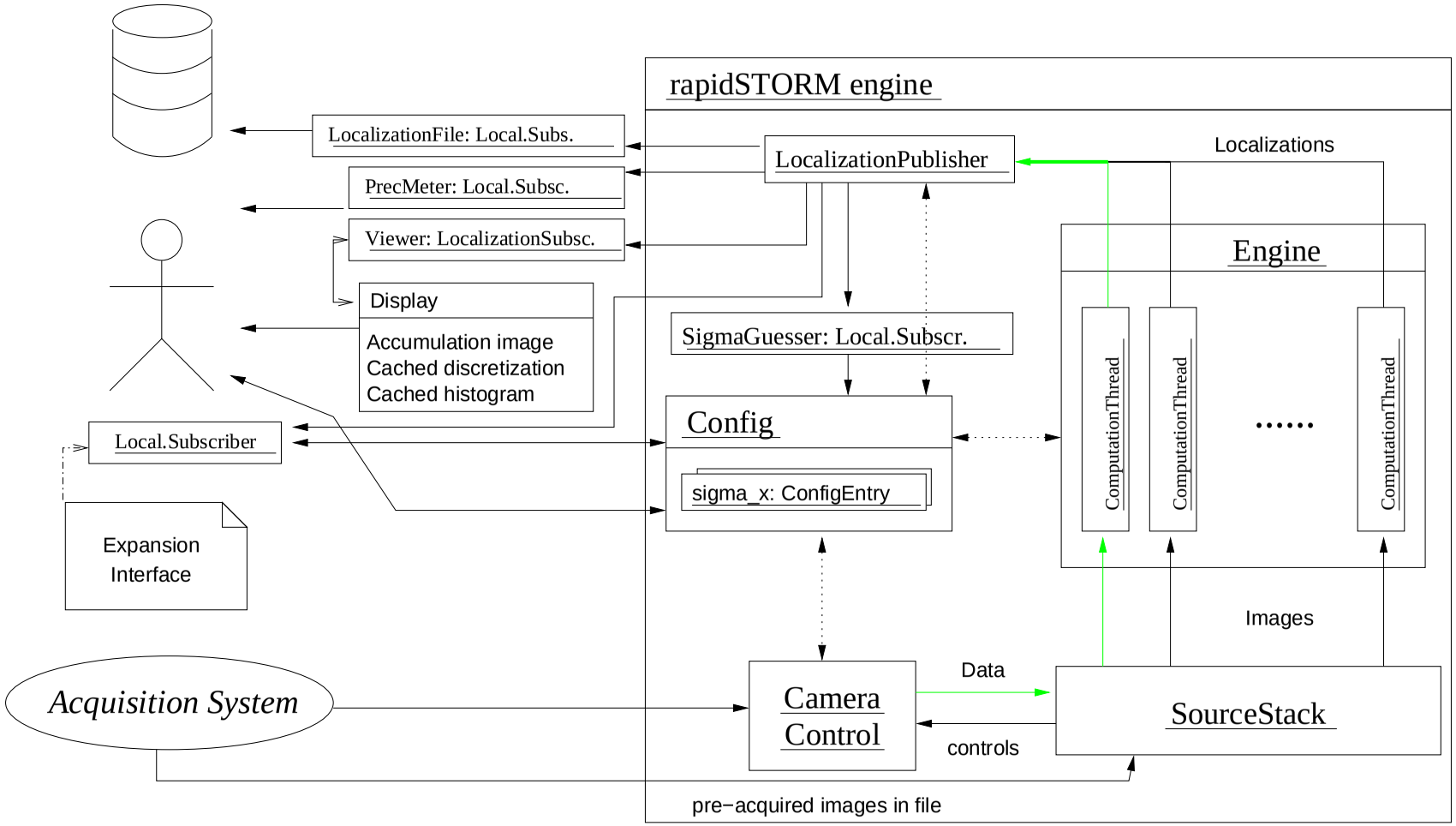


Figure C.2: Overview over the complete `rapidSTORM` system

Appendix D

Tables

	X estimate				Y estimate			
Real value	Mean	Min	Max	σ	Mean	Min	Max	σ
1.2	1.180	1.147	1.219	0.018	1.188	1.141	1.220	0.021
1.25	1.236	1.177	1.276	0.022	1.248	1.194	1.293	0.022
1.3	1.289	1.246	1.338	0.019	1.300	1.246	1.333	0.019
1.35	1.338	1.301	1.378	0.020	1.347	1.299	1.388	0.019
1.4	1.386	1.346	1.431	0.021	1.394	1.355	1.437	0.021
1.45	1.441	1.399	1.481	0.019	1.447	1.405	1.481	0.018
1.5	1.483	1.444	1.532	0.018	1.491	1.458	1.534	0.020
1.55	1.525	1.486	1.570	0.018	1.535	1.480	1.587	0.021
1.6	1.566	1.510	1.610	0.023	1.579	1.514	1.619	0.021
1.65	1.617	1.557	1.662	0.021	1.621	1.572	1.660	0.022
1.7	1.655	1.620	1.695	0.021	1.661	1.619	1.701	0.018
1.75	1.704	1.655	1.785	0.029	1.712	1.666	1.777	0.027
1.8	1.762	1.671	1.826	0.034	1.769	1.674	1.827	0.033
1.85	1.825	1.778	1.869	0.022	1.823	1.746	1.864	0.022
1.9	1.872	1.825	1.916	0.024	1.877	1.829	1.937	0.023
1.95	1.912	1.843	1.967	0.026	1.911	1.853	1.959	0.021
2	1.962	1.913	2.003	0.023	1.963	1.904	2.015	0.024
2.05	1.994	1.952	2.048	0.020	1.998	1.953	2.054	0.024
2.1	2.036	1.992	2.087	0.025	2.040	1.980	2.084	0.027
2.15	2.074	2.022	2.142	0.030	2.087	2.041	2.142	0.026
2.2	2.114	2.057	2.159	0.026	2.119	2.078	2.160	0.018
2.25	2.142	2.089	2.197	0.027	2.149	2.099	2.203	0.027
2.3	2.187	2.126	2.271	0.031	2.192	2.126	2.281	0.034
2.35	2.251	2.154	2.344	0.052	2.265	2.160	2.373	0.051
2.4	2.301	2.192	2.389	0.049	2.325	2.190	2.411	0.050
2.45	2.368	2.209	2.461	0.046	2.375	2.177	2.450	0.048
2.5	2.411	2.286	2.485	0.040	2.436	2.311	2.494	0.043

Table D.1: σ estimator error statistic for 5 kHz photon emittance rate

Table D.5: Effect of M_0 on localization count and computation time

Sp.	θ_A	M_0	# of local.	t (s)	t / loc. (μs)
1	1000	1	88308	38.91	0.44
1	1000	2	125051	58.86	0.47
1	1000	3	160455	80.89	0.50
1	1000	4	201584	109.29	0.54
1	1000	5	239795	136.30	0.57
1	4000	1	63475	31.37	0.49
1	4000	2	74743	40.19	0.54
1	4000	3	79300	46.07	0.58
1	4000	4	81941	51.52	0.63
1	4000	5	83767	56.50	0.67
1	9000	1	37915	21.87	0.58
1	9000	2	48523	31.0	0.64

Table D.5: Effect of M_0 on localization count and computation time

Sp.	θ_A	M_0	# of local.	t (s)	t / loc. (μs)
1	9000	3	52747	37.20	0.71
1	9000	4	54646	42.02	0.77
1	9000	5	55619	45.88	0.82
2	1200	1	17503	21.61	1.23
2	1200	2	17596	27.65	1.57
2	1200	3	17597	33.68	1.91
2	1200	4	17597	39.41	2.24
2	1200	5	17597	45.50	2.59
2	300	1	26605	21.94	0.82
2	300	2	30788	28.82	0.94
2	300	3	34621	37.75	1.09
2	300	4	37514	46.38	1.24
2	300	5	41707	57.02	1.37
2	600	1	21014	21.47	1.02
2	600	2	21079	27.06	1.28
2	600	3	21089	32.85	1.56
2	600	4	21094	39.21	1.86
2	600	5	21097	44.63	2.12
3	1200	1	1475	8.72	5.91
3	1200	2	1971	10.71	5.43
3	1200	3	2158	14.30	6.63
3	1200	4	2248	15.36	6.83
3	1200	5	2238	20.57	9.19
3	300	1	9518	11.14	1.17
3	300	2	18301	15.87	0.87
3	300	3	26379	21.80	0.83
3	300	4	32421	36.05	1.11
3	300	5	42743	40.92	0.96
3	600	1	3152	9.60	3.05
3	600	2	4435	15.14	3.41
3	600	3	5390	21.44	3.98
3	600	4	6121	23.65	3.86
3	600	5	7414	27.43	3.70
4	1000	1	15297	9.58	0.63
4	1000	2	15619	10.06	0.64
4	1000	3	15733	10.20	0.65
4	1000	4	15835	10.49	0.66
4	1000	5	15910	10.80	0.68
4	4000	1	14523	9.27	0.64
4	4000	2	14651	9.50	0.65
4	4000	3	14652	9.72	0.66
4	4000	4	14652	9.88	0.67
4	4000	5	14652	10.04	0.69
4	9000	1	13308	8.90	0.67

Table D.5: Effect of M_0 on localization count and computation time

Sp.	θ_A	M_0	# of local.	t (s)	t / loc. (μs)
4	9000	2	13437	9.16	0.68
4	9000	3	13437	9.36	0.70
4	9000	4	13437	9.47	0.70
4	9000	5	13437	9.59	0.71
5	1200	1	1508	1.28	0.85
5	1200	2	1508	1.62	1.07
5	1200	3	1508	1.92	1.27
5	1200	4	1508	2.14	1.42
5	1200	5	1508	2.41	1.60
5	300	1	3469	1.69	0.49
5	300	2	3479	2.02	0.58
5	300	3	3479	2.30	0.66
5	300	4	3479	2.51	0.72
5	300	5	3479	2.68	0.77
5	600	1	2257	1.56	0.69
5	600	2	2258	1.90	0.84
5	600	3	2258	2.14	0.95
5	600	4	2258	2.36	1.05
5	600	5	2258	2.67	1.18
6	1000	1	2357	4.91	2.08
6	1000	2	2442	5.07	2.08
6	1000	3	2455	9.28	3.78
6	1000	4	2463	9.20	3.74
6	1000	5	2467	9.20	3.73
6	4000	1	2193	4.88	2.23
6	4000	2	2196	5.10	2.32
6	4000	3	2196	5.12	2.33
6	4000	4	2196	5.19	2.36
6	4000	5	2196	5.20	2.37
6	9000	1	2175	4.93	2.27
6	9000	2	2175	5.07	2.33
6	9000	3	2175	5.16	2.37
6	9000	4	2175	5.13	2.36
6	9000	5	2175	5.20	2.39
7	1000	1	10477	16.87	1.61
7	1000	2	11437	26.50	2.32
7	1000	3	12019	34.13	2.84
7	1000	4	12487	37.83	3.03
7	1000	5	12900	39.52	3.06
7	4000	1	6178	13.92	2.25
7	4000	2	6128	22.34	3.65
7	4000	3	6207	32.79	5.28
7	4000	4	6205	37.20	5.00
7	4000	5	6208	38.16	6.15

Table D.5: Effect of M_0 on localization count and computation time

Sp.	θ_A	M_0	# of local.	t (s)	t / loc. (μs)
7	9000	1	10	6.68	667.00
7	9000	2	10	9.10	909.00
7	9000	3	10	16.94	1694.00
7	9000	4	10	27.93	2793.0
7	9000	5	10	32.63	3263.00
8	1000	1	43779	30.14	0.69
8	1000	2	66597	48.56	0.73
8	1000	3	78848	63.17	0.80
8	1000	4	90437	75.42	0.83
8	1000	5	102738	93.52	0.91
8	4000	1	36750	30.98	0.84
8	4000	2	48475	44.85	0.93
8	4000	3	50444	47.26	0.94
8	4000	4	51728	53.18	1.03
8	4000	5	52102	57.04	1.09
8	9000	1	23077	50.16	2.17
8	9000	2	26446	30.93	1.17
8	9000	3	25766	32.77	1.27
8	9000	4	27799	53.16	1.91
8	9000	5	27931	57.52	2.06

	X estimate				Y estimate			
Real value	Mean	Min	Max	σ	Mean	Min	Max	σ
1.2	1.192	1.164	1.222	0.014	1.201	1.176	1.230	0.014
1.25	1.249	1.214	1.284	0.017	1.250	1.220	1.290	0.017
1.3	1.305	1.275	1.339	0.016	1.306	1.269	1.338	0.018
1.35	1.348	1.312	1.389	0.018	1.351	1.320	1.381	0.014
1.4	1.398	1.359	1.440	0.020	1.410	1.367	1.450	0.019
1.45	1.452	1.411	1.498	0.020	1.455	1.418	1.495	0.018
1.5	1.501	1.471	1.553	0.017	1.504	1.472	1.545	0.017
1.55	1.544	1.510	1.582	0.018	1.552	1.515	1.584	0.019
1.6	1.601	1.557	1.639	0.021	1.603	1.554	1.643	0.020
1.65	1.644	1.597	1.677	0.020	1.653	1.613	1.690	0.018
1.7	1.693	1.646	1.739	0.020	1.705	1.650	1.747	0.022
1.75	1.752	1.700	1.802	0.024	1.760	1.715	1.795	0.021
1.8	1.801	1.764	1.828	0.017	1.809	1.770	1.851	0.020
1.85	1.850	1.814	1.889	0.017	1.854	1.825	1.886	0.016
1.9	1.891	1.854	1.932	0.020	1.901	1.845	1.953	0.022
1.95	1.948	1.902	1.989	0.019	1.956	1.911	1.996	0.023
2	1.993	1.962	2.041	0.021	2.002	1.967	2.053	0.021
2.05	2.042	2.008	2.069	0.016	2.049	1.999	2.096	0.024
2.1	2.097	2.046	2.150	0.024	2.098	2.038	2.138	0.021
2.15	2.140	2.093	2.171	0.019	2.144	2.085	2.200	0.024
2.2	2.184	2.128	2.224	0.021	2.191	2.125	2.239	0.024
2.25	2.235	2.194	2.294	0.026	2.244	2.189	2.294	0.023
2.3	2.297	2.231	2.364	0.028	2.303	2.247	2.356	0.023
2.35	2.341	2.294	2.394	0.024	2.354	2.316	2.392	0.017
2.4	2.392	2.343	2.452	0.023	2.409	2.353	2.450	0.024
2.45	2.445	2.400	2.497	0.023	2.454	2.387	2.502	0.025
2.5	2.491	2.444	2.535	0.022	2.501	2.450	2.563	0.025

Table D.2: σ estimator error statistic for 10 kHz photon emittance rate

Sp.	Ampl.	*Spalttiefpass*		Erosion		Fillhole	
no.	thres.	# loc.	t (s).# loc.	t (s)	# loc..t (s)		
1	1000	160455	81.16	99279	79.32	95901	94.30
	4000	79300	46.36	79044	66.66	78066	80.46
	9000	52747	37.39	56559	54.79	56003	69.02
2	300	34621	37.74	32319	64.51	31468	98.43
	600	21089	33.24	20985	61.64	21081	92.54
	1200	17597	33.50	17631	62.25	17598	91.98
3	300	28176	30.08	24618	39.84	19913	56.50
	600	5390	21.48	4895	30.25	4808	49.78
	1200	2158	14.22	2016	24.33	2018	42.33
4	1000	15733	10.19	15834	24.43	15787	36.98
	4000	14652	9.65	15167	23.85	15164	36.57
	9000	13437	9.22	13851	22.97	13851	35.40
5	300	3479	2.37	3348	3.92	3351	6.84
	600	2258	2.15	2254	3.65	2248	5.54
	1200	1508	1.90	1506	3.65	1508	5.52
6	1000	2462	5.14	2207	9.04	2223	9.06
	4000	2196	5.12	2192	8.94	2189	8.97
	9000	2175	5.20	2185	8.93	2176	8.94
7	1000	12019	34.19	11140	85.25	11092	139.68
	4000	6128	31.74	6117	84.66	6128	138.16
	9000	10	17.37	10	80.07	10	132.60
8	1000	78848	63.19	78309	83.28	77487	127.94
	4000	50310	52.22	59223	97.39	60957	79.27
	9000	25766	32.81	30286	48.43	29565	57.51

Table D.3: Spot finder statistics for erosion operator

Sp.	θ_A	m_s	TPi	FPi	FNi	TPi	FPi
9	4000	1.0	7.09	3.60	3.28	5.92	2.04
	4000	1.5	6.80	3.48	3.28	5.68	0.40
	4000	2.0	5.66	3.69	4.28	4.50	0.10
	4000	2.5	3.94	4.29	5.83	2.72	0.15
10	4000	1.0	6.59	3.60	3.83	5.42	1.97
	4000	1.5	6.46	3.47	3.75	5.29	0.40
	4000	2.0	5.74	3.65	4.26	4.42	0.09
	4000	2.5	3.36	4.28	6.46	1.60	0.01
11	4000	1.0	22.39	1.08	71.44	20.34	0.15
	4000	1.5	13.66	1.13	80.35	13.73	0.28
	4000	2.0	3.35	2.56	91.11	1.67	0.22
	4000	2.5	0.79	0.99	93.20	0.11	0.04
12	4000	1.0	6.0	3.88	5.08	4.06	1.94
	4000	1.5	5.58	3.73	4.99	3.78	0.39
	4000	2.0	3.77	3.90	6.23	2.36	0.09
	4000	2.5	2.47	3.96	7.48	1.42	0.09
13	4000	1.0	6.38	3.38	3.78	4.87	0.25
	4000	1.5	6.28	3.36	3.83	4.85	0.05
	4000	2.0	5.32	3.67	4.61	3.84	0.01
	4000	2.5	3.86	4.19	5.91	2.39	0.00
16	4000	1.0	7.12	3.19	2.75	4.75	0.00
	4000	1.5	7.09	3.18	2.73	4.78	0.00
	4000	2.0	6.83	3.34	2.91	4.84	0.00
	4000	2.5	5.75	4.09	3.90	3.71	0.0
17	4000	1.0	7.38	3.10	2.17	4.63	0.00
	4000	1.5	7.35	3.09	2.18	4.64	0.00
	4000	2.0	7.31	3.18	2.19	4.82	0.00
	4000	2.5	7.31	3.18	2.19	4.82	0.00
18	4000	1.0	6.80	3.23	3.03	4.68	0.00
	4000	1.5	6.70	3.26	3.09	4.75	0.00
	4000	2.0	6.49	3.84	3.28	4.72	0.45
	4000	2.5	6.14	3.50	3.57	4.44	0.00

Table D.4: Effects of smoothing mask size on spot and localization error rates

Appendix E

Schüttpelz implementation

Several times in my work, I have mentioned the important contribution of Mark Schüttpelz to the deconfinement algorithm. Because his pioneering implementation has not been published, I include it here to give proper credit and reference. This chapter will list the Matlab script files written for dSTORM deconfinement. I want to stress that all of this code is quoted and not my own work.

STORMconv.m

```
clear all
[datei pfad]=uigetfile;

if datei ~= 0
sizex=input('x size [128] = ');
sizey=input('y size [128] = ');
if size(sizex) == [0 0]
    sizex=128;
end;
if size(sizey) == [0 0]
    sizey=128;
end;
    load([pfad datei]);
    save fitresults fitresults -V6;
    a=fitresults(:,1);
    b=fitresults(:,3);
    c=[a b];
    histim=hist3(c,'Edges',{1:.1:sizex 1:.1:sizey});
save histim histim -V6;
end
```

STORMengine.m

```
function x = STORMengine()
```

```
2  global filename threshold startim endim roi1 roi2 roi3 roi4
       magnification
3  while 1~=0
4      datei=dir('C:\StormIn\ control.txt');
5      if size(datei,1) > 0
6          dostorm(['C:\StormIn\' datei(1).name]);
7          dos(['rename "C:\StormIn\' datei(1).name '" "'
               datei(1).name '.done"']);
8          'Done... STORMengine V0.4'
9      else
10         pause(10)
11     end
12 end
13
14 function dostorm(controldatei)
15 global filename threshold delta startim endim roi1 roi2
       roi3 roi4 magnification tolerance
16 fid=fopen(controldatei);
17 filename=['C:\StormIn\' fgetl(fid)]
18 threshold=str2num(fgetl(fid)); %edit3
19 delta=str2num(fgetl(fid)); %edit11
20 startim=str2num(fgetl(fid)); %edit4
21 endim=str2num(fgetl(fid)); %edit5
22 roi1=str2num(fgetl(fid)); %edit7
23 roi2=str2num(fgetl(fid)); %edit8
24 roi3=str2num(fgetl(fid)); %edit9
25 roi4=str2num(fgetl(fid)); %edit10
26 magnification=str2num(fgetl(fid)); %edit6
27 tolerance=str2num(fgetl(fid)); %edit12
28 fclose(fid);
29
30 fitresults=[1 1 1 1 1 1 1 1];
31 imavg=zeros((roi4-roi3)+1,(roi2-roi1)+1);
32
33 [data,back,ref]=sifread(filename);
34 moviein=data.imageData;
35
36 for i=startim:endim
37     i
38     subim=moviein(roi3:roi4,roi1:roi2,i);
39     temp=calcstorm(subim);
40     fitresults=[fitresults;temp, ones(size(temp,1),1) i];
41     imavg=imavg+subim;
42 end
43 fitresults=fitresults(2:end,:);
44 temp=[getfilename() ' fitresults.mat'];
45 save(temp,'fitresults');
```

```
46 imavg=imavg/(endim-startim+1);
47 imwrite(mat2gray(abs(max(max(imavg))-imavg)),[getfilename()
       '␣Average.tif'],'TIFF');
48 histim=hist3(fitresults(:,[1 3]) magnification,'Edges',{1:(
      roi4-roi3+1) magnification 1:(roi2-roi1+1) magnification
      });
49 imwrite(abs(1-mat2gray(histim,[0 1])),[getfilename() '␣'
      num2str(magnification) 'x␣Image.tif'],'TIFF');
50
51 function fitresults = calcstorm(subim)
52 global threshold delta tolerance
53 options=optimset('TolFun',tolerance);
54 sizex=8;
55 sizey=8;
56 fitresults=[1 1 1 1 1 1 1];
57
58 imlabel=bwlabel(imhmax(subim,delta)>threshold);
59 nrspots=max(max(imlabel));
60 imregionprops=regionprops(imlabel,'Area','BoundingBox','
      Centroid');
61 imspotcenters=[imregionprops.Centroid];
62
63 for i=1:nrspots,
64     posy=round(imspotcenters(2*i-1));
65     posx=round(imspotcenters(2*i));
66     if posx-sizex/2>0 && posx+sizex/2 < size(subim,1) &&
         posy-sizey/2>0 && posy+sizey/2 < size(subim,2)
67         imfit=double(subim(posx-sizex/2:posx+sizex/2,posy-
             sizey/2:posy+sizey/2));
68         [x,y]=meshgrid(1:sizex+1,1:sizey+1);
69         grid=[x y];
70         background=mean(mean(imfit));
71         peak=max(max(imfit))-background;
72         startparam=double([peak,sizex/2,2,sizey/2,2,
             background]);
73         [a res]=lsqcurvefit(@gauss2d,startparam,grid,imfit
             ,[0 0 0 0 0 0],[],options);
74         if res/a(1) < 1e4 && a(3)<sizey/2-1 && a(5)<sizex
             /2-1
75             fitresults=[fitresults;posx-sizex/2+a(4)-1,a(5)
                 ,posy-sizey/2+a(2)-1,a(3),a(1),a(6),res];
76         end
77     end
78 end
79 fitresults=fitresults(2:end,:);
80
81 function f = gauss2d(a,data)
```

```
82  x = data(:,1:size(data,2)/2);
83  y = data(:,size(data,2)/2+1:end);
84  f = a(1) exp(-0.5(x-a(2)).^2/(a(3)^2)-0.5(y-a(4)).^2/(a
        (5)^2)) + a(6);
85
86  function filenameout = getfilename()
87  global filename threshold delta startim endim roi1 roi2
        roi3 roi4 magnification tolerance
88  filenameout=['C:\StormOut\' filename(12:length(filename)-4)
         '␣Threshold␣' num2str(threshold) '␣Delta␣' num2str(
        delta) '␣Tolerance␣' num2str(tolerance) '␣Images␣'
        num2str(startim) '-' num2str(endim) '␣ROI␣' num2str(roi1
        ) '-' num2str(roi2) '-' num2str(roi3) '-' num2str(roi4)
        ];
89
90  function [data,back,ref]=sifread(file)
91  f=fopen(file,'r');
92  if f < 0
93     error('Could␣not␣open␣the␣file.');
94  end
95  if ~isequal(fgetl(f),'Andor␣Technology␣Multi-Channel␣File')
96     fclose(f);
97     error('Not␣an␣Andor␣SIF␣image␣file.');
98  end
99  skipLines(f,1);
100 [data,next]=readSection(f);
101 if nargout > 1 & next == 1
102    [back,next]=readSection(f);
103    if nargout > 2 & next == 1
104       ref=back;
105       back=readSection(f);
106    else
107       ref=struct([]);
108    end
109 else
110    back=struct([]);
111    ref=back;
112 end
113 fclose(f);
114
115 function [info,next]=readSection(f)
116 o=fscanf(f,'%d',6);
117 info.temperature=o(6);
118 skipBytes(f,10);
119 o=fscanf(f,'%f',5);
120 info.exposureTime=o(2);
121 info.cycleTime=o(3);
```

```
122 info.accumulateCycles=o(5);
123 info.accumulateCycleTime=o(4);
124 skipBytes(f,2);
125 o=fscanf(f,'%f',2);
126 info.stackCycleTime=o(1);
127 info.pixelReadoutTime=o(2);
128 o=fscanf(f,'%d',3);
129 info.gainDAC=o(3);
130 skipLines(f,1);
131 info.detectorType=readLine(f);
132 info.detectorSize=fscanf(f,'%d',[1 2]);
133 info.fileName=readString(f);
134 skipLines(f,3);
135 skipBytes(f,14);
136 info.shutterTime=fscanf(f,'%f',[1 2]);
137 skipLines(f,8);
138 if strcmp('Luc',info.detectorType) || strcmp('DV897_BV',
       info.detectorType)
139    skipLines(f,2);
140 end
141 info.frameAxis=readString(f);
142 info.dataType=readString(f);
143 info.imageAxis=readString(f);
144 o=fscanf(f,'65538␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣%d␣65538␣%d␣%d␣%d␣%d
       ␣%d␣%d',14);
145 info.imageArea=[o(1) o(4) o(6);o(3) o(2) o(5)];
146 info.frameArea=[o(9) o(12);o(11) o(10)];
147 info.frameBins=[o(14) o(13)];
148 s=(1 + diff(info.frameArea))./info.frameBins;
149 z=1 + diff(info.imageArea(5:6));
150 if prod(s) ~= o(8) | o(8) z ~= o(7)
151    fclose(f);
152    error('Inconsistent␣image␣header.');
153 end
154 for n=1:z
155    o=readString(f);
156    if numel(o)
157       fprintf('%s\n',o);
158    end
159 end
160 info.timeStamp=fread(f,1,'uint16');
161 info.imageData=reshape(fread(f,prod(s) z,'single=>single')
       ,[s z]);
162 o=readString(f);
163 if numel(o)
164    fprintf('%s\n',o);
165 end
```

```
166 next=fscanf(f,'%d',1);
167
168 function o=readString(f)
169 n=fscanf(f,'%d',1);
170 if isempty(n) | n < 0 | isequal(fgetl(f),-1)
171    fclose(f);
172    error('Inconsistent string.');
173 end
174 o=fread(f,[1 n],'uint8=>char');
175
176 function o=readLine(f)
177 o=fgetl(f);
178 if isequal(o,-1)
179    fclose(f);
180    error('Inconsistent image header.');
181 end
182 o=deblank(o);
183
184 function skipBytes(f,N)
185 [s,n]=fread(f,N,'uint8');
186 if n < N
187    fclose(f);
188    error('Inconsistent image header.');
189 end
190
191 function skipLines(f,N)
192 for n=1:N
193    if isequal(fgetl(f),-1)
194       fclose(f);
195       error('Inconsistent image header.');
196    end
197 end
```

STORMhist.m

```
 1 clear all
 2 sizex=128;
 3 sizey=128;
 4
 5 [datei pfad]=uigetfile;
 6
 7 if datei ~= 0
 8
 9 load([pfad datei]);
10 output=zeros(200,1);
11
12 a=fitresults(:,1);
```

```
13 b=fitresults(:,3);
14 c=[a b];
15 histim=hist3(c,'Edges',{1:.1:sizex 1:.1:sizey});
16
17 imagesc(histim);
18 [xpts,ypts]=getpts;
19 close;
20
21 for i=1:floor(size(xpts,1)/2)
22     x(1)=xpts(i*2-1);
23     x(2)=xpts(i*2);
24     y(1)=ypts(i*2-1);
25     y(2)=ypts(i*2);
26
27     if y(1) > y(2)
28         x=fliplr(x);
29         y=fliplr(y);
30     end;
31
32     deltax=x(2)-x(1);
33     deltay=y(2)-y(1);
34
35     alpha=-atan(deltax/deltay);
36     R=[cos(alpha),-sin(alpha);sin(alpha),cos(alpha)];
37
38     c2=(R*c')';
39     histrectx=R*([y(1);x(1)]/10);
40     histrecty=R*([y(2);x(2)]/10);
41     rsizex=round(sqrt(2)*sizex);
42     rsizey=round(sqrt(2)*sizey);
43     histim2=hist3(c2,'Edges',{-rsizex:.1:rsizex -rsizey:.1:
           rsizey});
44     tempx=(rsizex+round(histrectx)+1)*10;
45     tempy=(rsizey+round(histrecty)+1)*10;
46
47     histdata=sum(histim2(tempx(1):tempy(1),tempx(2)-50:
           tempy(2)+50));
48     [histmax maxindex]=max(histdata);
49     output(-maxindex+101:-maxindex+201)=output(-maxindex
           +101:-maxindex+201)+histdata';
50 end;
51 bar(output);
52 csvwrite([pfad datei '.txt'], output');
53
54 end;
```

STORMview.m

```
1  function varargout = STORMview(varargin)
2  % STORMview M-file for STORMview.fig
3  %      STORMview, by itself, creates a new STORMview or
       raises the existing
4  %      singleton.
5  %
6  %      H = STORMview returns the handle to a new STORMview
       or the handle to
7  %      the existing singleton.
8  %
9  %      STORMview('CALLBACK',hObject,eventData,handles,...)
       calls the local
10 %      function named CALLBACK in STORMview.M with the
       given input arguments.
11 %
12 %      STORMview('Property','Value',...) creates a new
       STORMview or raises the
13 %      existing singleton.  Starting from the left,
       property value pairs are
14 %      applied to the STORMview before STORMview_OpeningFcn
        gets called.  An
15 %      unrecognized property name or invalid value makes
       property application
16 %      stop.  All inputs are passed to STORMview_OpeningFcn
        via varargin.
17 %
18 %       See STORMview Options on GUIDE's Tools menu.
       Choose "STORMview allows only one
19 %      instance to run (singleton)".
20 %
21 % See also: GUIDE, GUIDATA, GUIHANDLES
22
23 % Edit the above text to modify the response to help
       STORMview
24
25 % Last Modified by GUIDE v2.5 03-Mar-2008 15:13:42
26
27 % Begin initialization code - DO NOT EDIT
28 gui_Singleton = 1;
29 gui_State = struct('gui_Name',       mfilename, ...
30                    'gui_Singleton',  gui_Singleton, ...
31                    'gui_OpeningFcn', @STORMview_OpeningFcn,
                           ...
```

```
32                    'gui_OutputFcn',  @STORMview_OutputFcn,
                          ...
33                    'gui_LayoutFcn',  [] , ...
34                    'gui_Callback',   []);
35 if nargin && ischar(varargin{1})
36     gui_State.gui_Callback = str2func(varargin{1});
37 end
38
39 if nargout
40     [varargout{1:nargout}] = gui_mainfcn(gui_State,
           varargin{:});
41 else
42     gui_mainfcn(gui_State, varargin{:});
43 end
44 % End initialization code - DO NOT EDIT
45
46
47 % --- Executes just before STORMview is made visible.
48 function STORMview_OpeningFcn(hObject, eventdata, handles,
       varargin)
49 % This function has no output args, see OutputFcn.
50 % hObject    handle to figure
51 % eventdata  reserved - to be defined in a future version
       of MATLAB
52 % handles    structure with handles and user data (see
       GUIDATA)
53 % varargin   command line arguments to STORMview (see
       VARARGIN)
54
55 % Choose default command line output for STORMview
56 handles.output = hObject;
57
58 % Update handles structure
59 guidata(hObject, handles);
60
61 global moviein moviesizex moviesizey fitresults
62 colormap(gray);
63
64 % UIWAIT makes STORMview wait for user response (see
       UIRESUME)
65 % uiwait(handles.figure1);
66
67 % --- Outputs from this function are returned to the
       command line.
68 function varargout = STORMview_OutputFcn(hObject, eventdata
       , handles)
```

```
69 % varargout  cell array for returning output args (see
       VARARGOUT);
70 % hObject    handle to figure
71 % eventdata  reserved - to be defined in a future version
       of MATLAB
72 % handles    structure with handles and user data (see
       GUIDATA)
73
74 % Get default command line output from handles structure
75 varargout{1} = handles.output;
76
77
78 function edit1_Callback(hObject, eventdata, handles)
79 % hObject    handle to edit1 (see GCBO)
80 % eventdata  reserved - to be defined in a future version
       of MATLAB
81 % handles    structure with handles and user data (see
       GUIDATA)
82
83 % Hints: get(hObject,'String') returns contents of edit1 as
       text
84 %        str2double(get(hObject,'String')) returns contents
       of edit1 as a double
85 set(handles.slider1, 'Value', str2num(get(hObject,'String')
       ));
86 updateim(handles);
87
88 function edit3_Callback(hObject, eventdata, handles)
89 % hObject    handle to edit3 (see GCBO)
90 % eventdata  reserved - to be defined in a future version
       of MATLAB
91 % handles    structure with handles and user data (see
       GUIDATA)
92
93 % Hints: get(hObject,'String') returns contents of edit3 as
       text
94 %        str2double(get(hObject,'String')) returns contents
       of edit3 as a double
95 updateim(handles);
96
97 function edit6_Callback(hObject, eventdata, handles)
98 % hObject    handle to edit6 (see GCBO)
99 % eventdata  reserved - to be defined in a future version
       of MATLAB
100 % handles    structure with handles and user data (see
       GUIDATA)
101
```

```
102 % Hints: get(hObject,'String') returns contents of edit6 as
        text
103 %        str2double(get(hObject,'String')) returns contents
        of edit6 as a double
104 global fitresults
105 histim=hist3(fitresults(:,[1 3]) str2num(get(handles.edit6,
        'String')),'Edges',{1:(str2num(get(handles.edit10,'
        String'))-str2num(get(handles.edit9,'String'))) str2num(
        get(handles.edit6,'String'))+1 1:(str2num(get(handles.
        edit8,'String'))-str2num(get(handles.edit7,'String')))
        str2num(get(handles.edit6,'String'))+1});
106 axes(handles.axes3);
107 imagesc(histim);
108
109 function edit11_Callback(hObject, eventdata, handles)
110 % hObject    handle to edit11 (see GCBO)
111 % eventdata  reserved - to be defined in a future version
        of MATLAB
112 % handles    structure with handles and user data (see
        GUIDATA)
113
114 % Hints: get(hObject,'String') returns contents of edit11
        as text
115 %        str2double(get(hObject,'String')) returns contents
        of edit11 as a double
116 updateim(handles);
117
118 function edit12_Callback(hObject, eventdata, handles)
119 % hObject    handle to edit12 (see GCBO)
120 % eventdata  reserved - to be defined in a future version
        of MATLAB
121 % handles    structure with handles and user data (see
        GUIDATA)
122
123 % Hints: get(hObject,'String') returns contents of edit12
        as text
124 %        str2double(get(hObject,'String')) returns contents
        of edit12 as a double
125 updateim(handles);
126
127 % --- Executes on slider movement.
128 function slider1_Callback(hObject, eventdata, handles)
129 % hObject    handle to slider1 (see GCBO)
130 % eventdata  reserved - to be defined in a future version
        of MATLAB
131 % handles    structure with handles and user data (see
        GUIDATA)
```

```
132
133 % Hints: get(hObject,'Value') returns position of slider
134 %        get(hObject,'Min') and get(hObject,'Max') to
        determine range of
135 %        slider
136 set(handles.edit1, 'String', num2str(round(get(hObject,'
        Value'))));
137 updateim(handles);
138
139 % --- Executes on button press in pushbutton1.
140 function pushbutton1_Callback(hObject, eventdata, handles)
141 % hObject    handle to pushbutton1 (see GCBO)
142 % eventdata  reserved - to be defined in a future version
        of MATLAB
143 % handles    structure with handles and user data (see
        GUIDATA)
144 global fitresults
145 [filename,pathname,filterindex] = uigetfile({'.mat','MAT-
        files␣(.mat)';'.', 'All␣Files␣(.)'},'Pick␣an␣MAT-
        file','MultiSelect', 'on');
146 if iscell(filename)
147     temp=load(filename{1});
148     fitresults=temp.fitresults;
149     for i=2:size(filename,2)
150         temp=load(filename{i});
151         fitresults=[fitresults; temp.fitresults];
152     end
153 else
154     temp=load(filename);
155     fitresults=temp.fitresults;
156 end
157 set(handles.edit8,'String',moviesizex);
158 set(handles.edit10,'String',moviesizey);
159 updateim(handles);
160
161 % --- Executes on button press in pushbutton3.
162 function pushbutton3_Callback(hObject, eventdata, handles)
163 % hObject    handle to pushbutton3 (see GCBO)
164 % eventdata  reserved - to be defined in a future version
        of MATLAB
165 % handles    structure with handles and user data (see
        GUIDATA)
166 global moviein moviesizex moviesizey
167 set(handles.checkbox1,'Value',0);
168 imavg=zeros(moviesizey,moviesizex);
169 for i=str2num(get(handles.edit4,'String')):str2num(get(
        handles.edit5,'String'))
```

```
170       imavg=imavg+moviein(:,:,i);
171   end
172   imavg=imavg/(str2num(get(handles.edit5,'String'))-str2num(
          get(handles.edit4,'String'))+1);
173   axes(handles.axes3);
174   imagesc(imavg);
175
176   % --- Executes on button press in pushbutton4.
177   function pushbutton4_Callback(hObject, eventdata, handles)
178   % hObject    handle to pushbutton4 (see GCBO)
179   % eventdata  reserved - to be defined in a future version
          of MATLAB
180   % handles    structure with handles and user data (see
          GUIDATA)
181   axes(handles.axes1);
182   roi=round(getrect);
183   set(handles.edit7,'String',roi(1));
184   set(handles.edit8,'String',roi(1)+roi(3));
185   set(handles.edit9,'String',roi(2));
186   set(handles.edit10,'String',roi(2)+roi(4));
187
188   % --- Executes on button press in pushbutton7.
189   function pushbutton7_Callback(hObject, eventdata, handles)
190   % hObject    handle to pushbutton7 (see GCBO)
191   % eventdata  reserved - to be defined in a future version
          of MATLAB
192   % handles    structure with handles and user data (see
          GUIDATA)
193   global moviesizex moviesizey
194   set(handles.edit7,'String',1);
195   set(handles.edit8,'String',moviesizex);
196   set(handles.edit9,'String',1);
197   set(handles.edit10,'String',moviesizey);
198
199   % --- Executes on button press in pushbutton8.
200   function pushbutton8_Callback(hObject, eventdata, handles)
201   % hObject    handle to pushbutton8 (see GCBO)
202   % eventdata  reserved - to be defined in a future version
          of MATLAB
203   % handles    structure with handles and user data (see
          GUIDATA)
204   global moviein fitresults
205   imavg=zeros(str2num(get(handles.edit10,'String'))-str2num(
          get(handles.edit9,'String'))+1,str2num(get(handles.edit8
          ,'String'))-str2num(get(handles.edit7,'String'))+1);
206   for i=str2num(get(handles.edit4,'String')):str2num(get(
          handles.edit5,'String'))
```

```
207       imavg=imavg+moviein(str2num(get(handles.edit9,'String')
              ):str2num(get(handles.edit10,'String')),str2num(get(
              handles.edit7,'String')):str2num(get(handles.edit8,'
              String')),i);
208   end
209   imavg=imavg/(str2num(get(handles.edit5,'String'))-str2num(
          get(handles.edit4,'String'))+1);
210   imwrite(mat2gray(abs(max(max(imavg))-imavg)),[getfilename(
          handles) '␣Average.tif'],'TIFF');
211   histim=hist3(fitresults(:,[1 3]) str2num(get(handles.edit6,
          'String')),'Edges',{1:(str2num(get(handles.edit10,'
          String'))-str2num(get(handles.edit9,'String'))+1)
          str2num(get(handles.edit6,'String')) 1:(str2num(get(
          handles.edit8,'String'))-str2num(get(handles.edit7,'
          String'))+1) str2num(get(handles.edit6,'String'))});
212   imwrite(abs(1-mat2gray(histim,[0 1])),[getfilename(handles)
           '␣' get(handles.edit6,'String') 'x␣Image.tif'],'TIFF');
213
214   % --- Executes on button press in checkbox1.
215   function checkbox1_Callback(hObject, eventdata, handles)
216   % hObject    handle to checkbox1 (see GCBO)
217   % eventdata  reserved - to be defined in a future version
          of MATLAB
218   % handles    structure with handles and user data (see
          GUIDATA)
219
220   % Hint: get(hObject,'Value') returns toggle state of
          checkbox1
221   updateim(handles);
222
223   % --- Executes on button press in checkbox2.
224   function checkbox2_Callback(hObject, eventdata, handles)
225   % hObject    handle to checkbox2 (see GCBO)
226   % eventdata  reserved - to be defined in a future version
          of MATLAB
227   % handles    structure with handles and user data (see
          GUIDATA)
228
229   % Hint: get(hObject,'Value') returns toggle state of
          checkbox2
230   if get(hObject,'Value')==1
231       colormap(flipud(gray));
232   else
233       colormap(gray);
234   end
235
236   function updateim(handles)
```

```
237 global moviein fitresults roi1 roi2 roi3 roi4
238 %subim=moviein(:,:,round(get(handles.slider1,'Value')));
239 axes(handles.axes1);
240 %imagesc(subim);
241 axes(handles.axes2);
242 %imagespots=imhmax(subim,str2num(get(handles.edit11,'String
    ')))>str2num(get(handles.edit3,'String'));
243 %imagesc(imagespots);
244 %set(handles.text11,'String',max(max(bwlabel(imagespots))))
    ;
245 axes(handles.axes3);
246 histim=hist3(fitresults(:,[1 3]) str2num(get(handles.edit6,
    'String')),'Edges',{1:(roi4-roi3+1) str2num(get(handles.
    edit6,'String')) 1:(roi2-roi1+1) str2num(get(handles.
    edit6,'String'))});
247 axis([1 (roi4-roi3+1) str2num(get(handles.edit6,'String'))
    1 (roi2-roi1+1) str2num(get(handles.edit6,'String'))]);
248 view([90 90]);
249
250
251 function filename = getfilename(handles)
252 filename=get(handles.activex1,'FileName');
253 filename=[filename(1:length(filename)-4) '␣Threshold␣' get(
    handles.edit3,'String') '␣Images␣' get(handles.edit4,'
    String') '-' get(handles.edit5,'String') '␣ROI␣' get(
    handles.edit7,'String') '-' get(handles.edit8,'String')
    '-' get(handles.edit9,'String') '-' get(handles.edit10,'
    String')];
254
255 % --- Executes during object creation, after setting all
    properties.
256 function slider1_CreateFcn(hObject, eventdata, handles)
257 % hObject    handle to slider1 (see GCBO)
258 % eventdata  reserved - to be defined in a future version
    of MATLAB
259 % handles    empty - handles not created until after all
    CreateFcns called
260
261 % Hint: slider controls usually have a light gray
    background.
262 if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
263     set(hObject,'BackgroundColor',[.9 .9 .9]);
264 end
265
266 % --- Executes during object creation, after setting all
    properties.
```

```
267 function edit1_CreateFcn(hObject, eventdata, handles)
268 % hObject    handle to edit1 (see GCBO)
269 % eventdata  reserved - to be defined in a future version
        of MATLAB
270 % handles    empty - handles not created until after all
        CreateFcns called
271
272 % Hint: edit controls usually have a white background on
        Windows.
273 %       See ISPC and COMPUTER.
274 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
275     set(hObject,'BackgroundColor','white');
276 end
277
278 % --- Executes during object creation, after setting all
        properties.
279 function edit2_CreateFcn(hObject, eventdata, handles)
280 % hObject    handle to edit2 (see GCBO)
281 % eventdata  reserved - to be defined in a future version
        of MATLAB
282 % handles    empty - handles not created until after all
        CreateFcns called
283
284 % Hint: edit controls usually have a white background on
        Windows.
285 %       See ISPC and COMPUTER.
286 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
287     set(hObject,'BackgroundColor','white');
288 end
289
290 % --- Executes during object creation, after setting all
        properties.
291 function edit3_CreateFcn(hObject, eventdata, handles)
292 % hObject    handle to edit3 (see GCBO)
293 % eventdata  reserved - to be defined in a future version
        of MATLAB
294 % handles    empty - handles not created until after all
        CreateFcns called
295
296 % Hint: edit controls usually have a white background on
        Windows.
297 %       See ISPC and COMPUTER.
298 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
299     set(hObject,'BackgroundColor','white');
```

```
300 end
301
302 % --- Executes during object creation, after setting all
        properties.
303 function edit4_CreateFcn(hObject, eventdata, handles)
304 % hObject    handle to edit4 (see GCBO)
305 % eventdata  reserved - to be defined in a future version
        of MATLAB
306 % handles    empty - handles not created until after all
        CreateFcns called
307
308 % Hint: edit controls usually have a white background on
        Windows.
309 %       See ISPC and COMPUTER.
310 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
311     set(hObject,'BackgroundColor','white');
312 end
313
314 % --- Executes during object creation, after setting all
        properties.
315 function edit5_CreateFcn(hObject, eventdata, handles)
316 % hObject    handle to edit5 (see GCBO)
317 % eventdata  reserved - to be defined in a future version
        of MATLAB
318 % handles    empty - handles not created until after all
        CreateFcns called
319
320 % Hint: edit controls usually have a white background on
        Windows.
321 %       See ISPC and COMPUTER.
322 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
323     set(hObject,'BackgroundColor','white');
324 end
325
326 % --- Executes during object creation, after setting all
        properties.
327 function edit6_CreateFcn(hObject, eventdata, handles)
328 % hObject    handle to edit6 (see GCBO)
329 % eventdata  reserved - to be defined in a future version
        of MATLAB
330 % handles    empty - handles not created until after all
        CreateFcns called
331
332 % Hint: edit controls usually have a white background on
        Windows.
```

```
333 %       See ISPC and COMPUTER.
334 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
335     set(hObject,'BackgroundColor','white');
336 end
337
338 % --- Executes during object creation, after setting all
        properties.
339 function edit7_CreateFcn(hObject, eventdata, handles)
340 % hObject    handle to edit7 (see GCBO)
341 % eventdata  reserved - to be defined in a future version
        of MATLAB
342 % handles    empty - handles not created until after all
        CreateFcns called
343
344 % Hint: edit controls usually have a white background on
        Windows.
345 %       See ISPC and COMPUTER.
346 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
347     set(hObject,'BackgroundColor','white');
348 end
349
350 % --- Executes during object creation, after setting all
        properties.
351 function edit8_CreateFcn(hObject, eventdata, handles)
352 % hObject    handle to edit8 (see GCBO)
353 % eventdata  reserved - to be defined in a future version
        of MATLAB
354 % handles    empty - handles not created until after all
        CreateFcns called
355
356 % Hint: edit controls usually have a white background on
        Windows.
357 %       See ISPC and COMPUTER.
358 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
359     set(hObject,'BackgroundColor','white');
360 end
361
362 % --- Executes during object creation, after setting all
        properties.
363 function edit9_CreateFcn(hObject, eventdata, handles)
364 % hObject    handle to edit9 (see GCBO)
365 % eventdata  reserved - to be defined in a future version
        of MATLAB
```

```
366 % handles    empty - handles not created until after all
        CreateFcns called
367
368 % Hint: edit controls usually have a white background on
        Windows.
369 %       See ISPC and COMPUTER.
370 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
371     set(hObject,'BackgroundColor','white');
372 end
373
374 % --- Executes during object creation, after setting all
        properties.
375 function edit10_CreateFcn(hObject, eventdata, handles)
376 % hObject    handle to edit10 (see GCBO)
377 % eventdata  reserved - to be defined in a future version
        of MATLAB
378 % handles    empty - handles not created until after all
        CreateFcns called
379
380 % Hint: edit controls usually have a white background on
        Windows.
381 %       See ISPC and COMPUTER.
382 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
383     set(hObject,'BackgroundColor','white');
384 end
385
386 % --- Executes during object creation, after setting all
        properties.
387 function edit11_CreateFcn(hObject, eventdata, handles)
388 % hObject    handle to edit11 (see GCBO)
389 % eventdata  reserved - to be defined in a future version
        of MATLAB
390 % handles    empty - handles not created until after all
        CreateFcns called
391
392 % Hint: edit controls usually have a white background on
        Windows.
393 %       See ISPC and COMPUTER.
394 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
395     set(hObject,'BackgroundColor','white');
396 end
397
398 % --- Executes during object creation, after setting all
        properties.
```

```
399 function edit12_CreateFcn(hObject, eventdata, handles)
400 % hObject    handle to edit12 (see GCBO)
401 % eventdata  reserved - to be defined in a future version
        of MATLAB
402 % handles    empty - handles not created until after all
        CreateFcns called
403
404 % Hint: edit controls usually have a white background on
        Windows.
405 %       See ISPC and COMPUTER.
406 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
407     set(hObject,'BackgroundColor','white');
408 end
```

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Diplomarbeit „ An Accurate and Efficient Algorithm for Real-Time Localisation of Photoswitchable Fluorophores “ selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe sowie alle Zitate entsprechend kenntlich gemacht habe.