Charles University in Prague

Faculty of Mathematics and Physics

**MASTER THESIS**

Jakub Tomek

# Processing data from two-photon microscope

Department of Theoretical Computer Science and Mathematical Logic

| | |
|---|---|
| Supervisor of the master thesis: | Ondřej Novák |
| Study programme: | Computer science |
| Specialization: | Theoretical computer science |

Prague 2013

Many thanks to the supervisor of this thesis, Ondřej Novák, for endless hours spent discussing and refining this thesis with me. I am also very grateful to Markéta Popelová with whom I spent another set of endless hours spent in discussion of the this thesis. Last, but not least, I thank Ondřej Zelenka and Cyril Brom for further consultations.

Název práce: Processing data from two-photon microscope

Autor: Jakub Tomek

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: Mgr. Ondřej Novák, Ústav experimentální medicíny AV ČR, Oddělení neurofyziologie sluchu

Abstrakt: Dvoufotonová mikroskopie je velmi moderní metodou neurofyziologického výzkumu *in vivo*, umožňující zobrazovat až stovky neuronů zároveň. Tato metoda však produkuje značný objem dat, která je obtížné zpracovat a analyzovat ručně. V této práci představujeme Two-Photon Processor, sadu nástrojů pro komplexní zpracování dat z dvoufotonového mikroskopu. V rámci práce jsme navrhli algoritmus SeNeCA pro segmentaci neuronů ve full-frame nahrávce z dvoufotonového mikroskopu. SeNeCA kombinuje vysokou rychlost zpracování a vysokou kvalitu segmentace a dle naší evaluace je nejlepším dostupným algoritmem pro segmentaci neuronů v *in vivo* datech. Nástroj je již rutinně používán v Ústavu experimentální medicíny AV ČR, Oddělení neurofyziologie sluchu, a byl publikován v Journal of Neurophysiology.

Klíčová slova: dvoufotonový mikroskop, zobrazování, zpracování, segmentace

Title: Processing data from two-photon microscope

Author: Jakub Tomek

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Ondřej Novák, Institute of Experimental Medicine of the ASCR, Department of Auditory Neuroscience

Abstract: Two-photon laser scanning microscopy is a modern method of *in vivo* neurophysiological research, capable of imaging up to hundreds of neurons at once. However, this method produces a large amount of data, difficult to process and analyze manually. This thesis presents Two-Photon Processor, a new toolkit for complex processing of data from two-photon microscope. During the work on this thesis, we designed the SeNeCA segmentation algorithm for detection of neurons in full-frame recording from a two-photon microscope. SeNeCA combines high speed and high quality of segmentation and, according to our evaluation, it currently is the best algorithm for segmentation of neurons in *in vivo* data. Two-Photon Processor is already routinely used in the Institute of Experimental Medicine of the ASCR, Department of Auditory Neuroscience, and it was published in the Journal of Neurophysiology.

Keywords: two-photon microscope, imaging, processing, segmentation

# Table of contents

**Introduction** **3**

**1 Imaging techniques** **4**

1.1 Overview of selected neuroimaging techniques . . . 4
1.1.1 Electrophysiology . . . 4
1.1.1.1 Intracellular recording . . . 5
1.1.1.2 Extracellular recording . . . 5
1.1.2 Electron microscopy . . . 6
1.1.3 Fluorescence microscopy . . . 7
1.1.3.1 Wide-field fluorescence microscopy . . . 8
1.1.3.2 Confocal microscopy . . . 8
1.1.3.3 Two-photon microscopy . . . 9
1.2 Related works in processing of microscope data . . . 12
1.2.1 Denoising . . . 13
1.2.2 Extraction of spike trains from video of brain . . . 14
1.2.2.1 Segmentation . . . 16
1.2.2.2 Tracking . . . 18
1.2.2.3 Inferring spike trains . . . 19
1.2.3 Counting cells . . . 19
1.2.4 Reconstructing neuronal connectivity . . . 19

**2 Software materials and methods** **21**

2.1 Feature overview . . . 21
2.2 Discussion of choice of programming language . . . 22
2.3 Software architecture . . . 23
2.4 Software description . . . 23
2.4.1 Extracting calcium traces from full-frame data . . . 23
2.4.1.1 The processing pipeline . . . 24
2.4.1.2 Discussion of the processing pipeline . . . 25
2.4.1.3 SeNeCA . . . 27
2.4.1.4 Defusion algorithms in TPP . . . 30
2.4.1.4.1 Type 1 defusion . . . 30
2.4.1.4.2 Type 2 defusion . . . 33
2.4.2 Planning line-scan path . . . 35
2.4.2.1 The processing pipeline . . . 37
2.4.2.2 Discussion of the processing pipeline . . . 37
2.4.2.3 Circumvention algorithm . . . 38
2.4.3 Extracting calcium traces from line-scan data . . . 39
2.4.3.1 Processing pipeline . . . 39
2.4.3.2 Discussion of processing pipeline . . . 40
2.4.3.3 Linescan-SeNeCA . . . 41
2.4.4 Noise removal and spike mining from calcium traces . . . 44
2.4.4.1 Processing pipeline . . . 45
2.4.4.2 Discussion of the processing pipeline . . . 46
2.4.4.3 Advanced analysis of experiments . . . 46

2.4.5 ZScan processing . . . . . . . . . . . . . . . . . . . . . . . . 47
2.4.5.1 Processing pipeline . . . . . . . . . . . . . . . . . . 47
2.4.5.2 Discussion of the processing pipeline . . . . . . . 48
2.4.6 Extension 1: Realtime worker . . . . . . . . . . . . . . . . . 48

**3 Evaluation and discussion** **51**
3.1 Evaluation of SeNeCA . . . . . . . . . . . . . . . . . . . . . . . . 51
3.1.1 On time consumption of segmentation using SeNeCA . . . 51
3.1.2 Evaluation on *in vitro* data . . . . . . . . . . . . . . . . . . 52
3.1.2.1 Description of evaluation . . . . . . . . . . . . . . . 53
3.1.2.2 Results . . . . . . . . . . . . . . . . . . . . . . . . . 53
3.1.3 Evaluation on *in vivo* data . . . . . . . . . . . . . . . . . . . 54
3.1.3.1 Description of data . . . . . . . . . . . . . . . . . . 54
3.1.3.2 Description of evaluation . . . . . . . . . . . . . . . 54
3.1.3.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . 55
3.1.4 Evaluation on *in silico* data . . . . . . . . . . . . . . . . . . 55
3.1.4.1 Artificial data modeling . . . . . . . . . . . . . . . 55
3.1.4.2 Used data . . . . . . . . . . . . . . . . . . . . . . . 56
3.1.4.2.1 SeNeCA versus human . . . . . . . . . . 56
3.1.4.2.2 Absolute performance of SeNeCA . . . . 56
3.1.4.2.3 Performance on data with uneven background brightness . . . . . . . . . . . . 56
3.1.4.3 Description of evaluation . . . . . . . . . . . . . . . 56
3.1.4.3.1 SeNeCA versus human . . . . . . . . . . 58
3.1.4.3.2 Absolute performance of SeNeCA . . . . 58
3.1.4.3.3 Performance on data with uneven background brightness . . . . . . . . . . . . 59
3.1.4.4 Results . . . . . . . . . . . . . . . . . . . . . . . . . 59
3.1.4.4.1 SeNeCA versus human . . . . . . . . . . 59
3.1.4.4.2 Absolute performance of SeNeCA . . . . 60
3.1.4.4.3 Performance on data with uneven background brightness . . . . . . . . . . . . 60
3.2 Comparison of TPP to other software tools . . . . . . . . . . . . . 62
3.2.1 BioImageXD and ImageJ . . . . . . . . . . . . . . . . . . . . 63
3.2.2 ISA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
3.2.3 CellSort . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64

**Conclusion** **67**

**Bibliography** **68**

**List of Abbreviations** **75**

# Introduction

One of the state of the art techniques of brain research, two-photon laser scanning microscopy, has quickly become a powerful tool for revealing structural and functional organization of cortical neurons in brains of living animals (Kerr et al., 2005; Knott et al., 2006; Schneider et al., 2012).

However, measurements performed with two-photon (2P) microscope yield gigabytes of video and/or image data. Certain questions may be answered by a researcher after simply looking at the data (e.g., are there any astrocytes in the observed part of brain?). On the other hand, answering more complex questions (e.g., how many neurons are in the observed part of brain and when are they active?) calls for automated, or semi-automated processing.

Recently, several tools which automate many tasks in processing *in vitro* data[1] have been created, such as (Collins, 2007; Kankaanpää et al., 2012; Schneider et al., 2012). Unfortunately, processing *in vivo* data[2] is a lot more difficult, due to, e.g., higher amount of noise, uneven staining of the tissue and 3D motion artifacts.

Usually, *in vivo* data are processed by custom-written software pieces (Bandyopadhyay et al., 2010; Cohen et al., 2011; Jarosiewicz et al., 2012), with the code not publicly available. This means that the published experiments can not be replicated and as it is impossible to critically judge the quality of the used software, the credibility of such research is questionable.

While there are publicly available tools for processing data from *in vivo* experiments, they are are restricted to a specific task, such as correction of motion artifacts ((Greenberg and Kerr, 2009)), or image segmentation ((Miri et al., 2011; Valmianski et al., 2010)). To our knowledge, there is no publicly available, open source software toolkit offering wider array of tools used in the analysis of the data from *in vivo* experiments.

This thesis addresses the need of a software toolkit for complex processing of various types of data from the 2P microscope. We designed and implemented the toolkit Two-Photon Processor (TPP), with primary aim to measure calcium traces of neurons in a full-frame[3] video and using these, infer times of neuronal spikes. The main problem was to find a suitable algorithm for automated segmentation of neurons in a video. We therefore developed a new segmentation algorithm SeNeCA (Search for Neural Cells Accelerated), able to successfully segment even *in vivo* data. Our main results have already been accepted for publication in the Journal of Neurophysiology (Tomek et al., in press).

In the first chapter of this thesis, we briefly describe several relevant methods of neuroimaging and related work in the field of processing microscope data. The second chapter describes the resulting software toolkit TPP. The third chapter contains the comparison of TPP to other relevant tools and the evaluation of SeNeCA. We conclude the thesis with the discussion of the results achieved.

[1]Obtained from parts of brain which have been isolated from their usual surroundings.
[2]Obtained from living brain.
[3]2D section of brain

# 1. Imaging techniques

Due to specialized nature of this thesis, we provide an introductory chapter, both on methods of extracting data from brain and on processing these data, listing related work in these fields.

The first section aims to put the 2P microscope into a broader context describes several relevant small-scale functional[1] recording techniques, which are widely used in neurophysiology research.

The second section of this chapter is dedicated to the description of several software techniques of processing data from microscopy. It contains a summary of related work in the fields which we ourselves worked in when developing Two-Photon Processor(TPP).

## 1.1 Overview of selected neuroimaging techniques

### 1.1.1 Electrophysiology

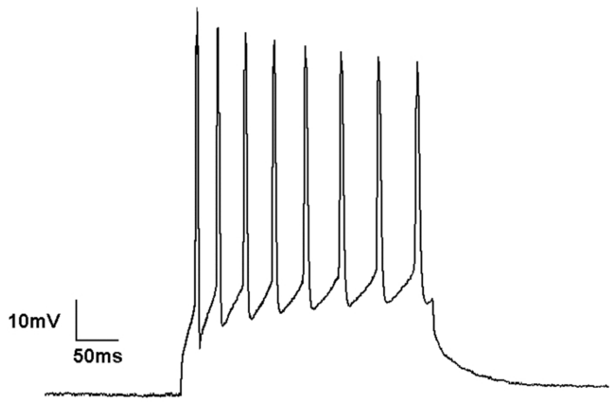


**Figure 1.1:** An example of single neuron recorded and its membrane potential plotted over time. From (Wikipedia.org, 2009)

.

Electrophysiology is one of the oldest methods of brain research (used as long ago as in 1950s in, e.g., (Hodgkin et al., 1952) or (Hubel and Wiesel, 1959)), which is still used and improved today. In contrast with microscopy, which measures optical properties of the observed tissue, electrophysiology directly measures electrical properties of cells. Such an approach is very useful for measuring neuronal activity, see Fig. 1.1 for an example. There are two ways of recording such properties: from within neurons (intracellular) or from the area around neurons (extracellular).

[1]Used to analyze the processes happening in brain, rather than pure morphology. Therefore, we do not describe CT, MRI, etc. The small-scale means that single neurons or small groups are observed. Therefore, we omit EEG, PET or fMRI.

#### 1.1.1.1 Intracellular recording

Traditionally, a microelectrode was used to impale a cell and measure voltage or current across the membrane. Hodgkin and Huxley used this technique to observe the mechanisms driving the generation of action potentials (they have been awarded Nobel prize in 1963). They used their observations to create a well known model of spiking neuron (Hodgkin and Huxley, 1952).

The original method of penetrating a cell's membrane has been improved by the introduction of patch clamp technique. This technique is less invasive and significantly improved the accuracy of measurement. Developed by Nehr and Sakman in late 1970s (they received a Nobel prize in 1991 for it), this technique does not use a sharp-tip electrode as the traditional approach, but a smooth-tip electrode instead. The tip is pressed against the cell membrane and a suction that forms a powerful seal is applied. The seal is often called a "gigaohm seal", due to high electrical resistance of the seal. Because of the high resistance, the currents across the ion channels in the membrane may be isolated and measured. Further reading, e.g., in (Hamill et al., 1981).

While intracellular recording offers excellent temporal resolution and may be performed both *in vivo* and *in vitro*, it is limited to a single cell. Therefore, its possible contribution to the research of large-scale networks is limited.

#### 1.1.1.2 Extracellular recording

In this case, a microelectrode is inserted into extracellular space of the tissue.

It is possible to record a single cell if a thin-tipped electrode (1 micrometer) is used.

When a thicker electrode is used, signal from several neurons is being integrated, which allows larger scale of observation. However, it is often difficult to tell how many neurons are measured and assign putative spikes to the putative neurons (spike sorting). This problem may be partially ameliorated when multielectrodes (e.g., tetrodes) are used (Harris et al., 2000). It is also possible to use an array of multielectrodes and insert it in the brain, thus being able to record a larger area of the neuronal network. This approach may bring better spatial resolution than extracellular recordings made by single multielectrodes (Lei, 2011). However, the process of insertion (especially chronic implantation) into a living brain leads to unfavourable changes in the tissue(e.g., loss of cells and glia) and immune system tends to react by inflammation (Biran et al., 2005); this suggests that the measured activity of the given part of brain may not be the same as in its natural state. Another disadvantage of *in vivo* use of multielectrode arrays (and extracellular recording in general) is the fact that the neurons may move relatively to the recording sites and it is difficult to tell, whether the same set of neurons is measured throughout the experiment. More about recent challenges and advances in this sort of extracellular recording may be found in (Chorev et al., 2009).

The thickness of electrode may be further increased. Then, while the possibility of telling apart different neurons is lost, local field potentials (LFP) may be measured[2]. LFP is believed to represent a synchronized input and output of the measured area.

[2]More about structure of LFP may be found in (Buzsáki et al., 2012).

### 1.1.2 Electron microscopy

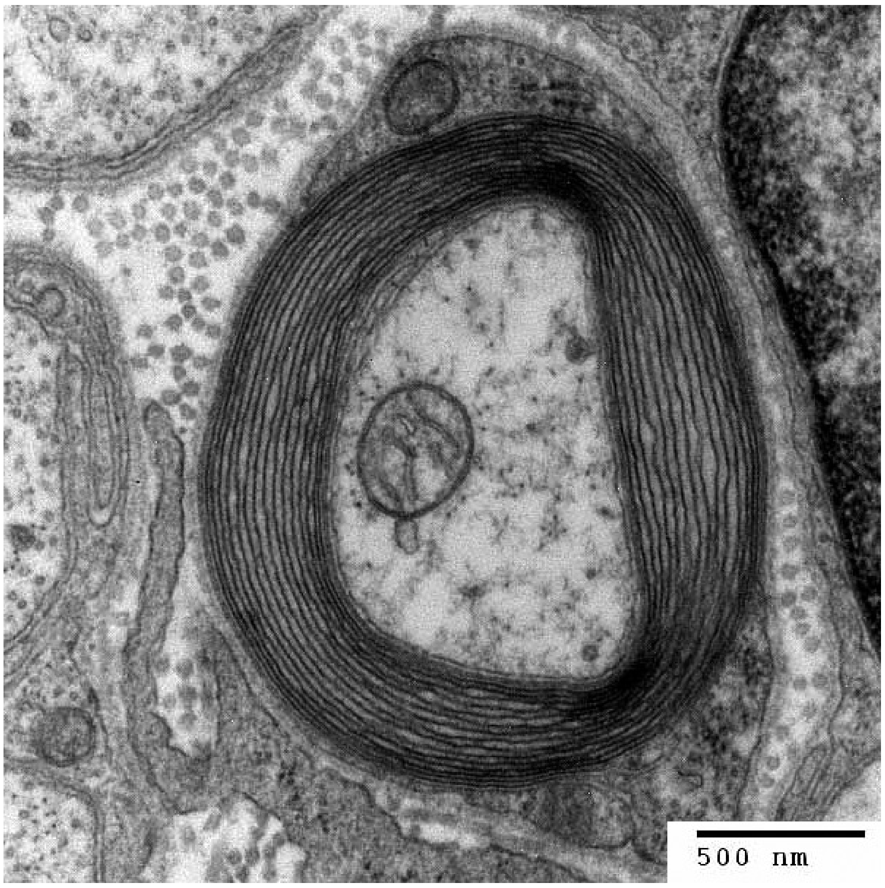


**Figure 1.2:** A section of myelinated axon, as seen by an electron microscope. From (Wikipedia.org, 2008).

The electron microscope (EM) is essentially similar to ordinary light microscopes, except it uses rays of electrons, rather than photons. This leads to much higher resolution (sub-nanometer), very fine structures may be therefore observed, see Fig. 1.2 for an example. Two basic types of EM exist: The first type is transmission electron microscope (TEM), which uses an electron gun to fire electrons through the observed sample (a thin sample must be used). The second type is scanning electron microscope (SEM), which raster-scans the sample with a focused electron beam and collects secondary and backscattered electrons to reconstruct the image. An advantage of SEM over TEM is that it is possible to scan a block of tissue, not necessarily a thin sample. TEM offers somewhat better resolution, however, for observing neurites and other thin structures, even SEM offers sufficient resolution. Both TEM and SEM require the sample to be observed in vacuum, as the molecules in air could scatter the beam of electrons. However, this means that the sample must be dry and fixated, as water in vacuum would become a gas and the sample would be destroyed. The environmental scanning electron microscope (ESEM) is a reaction to this disadvantage. At the cost of slightly decreased resolution, it is possible to observe samples containing water in low vacuum.

Even though electron microscopes are limited to *in vitro* observation and they do not detect electrical activity of neurons, they may still be an important tool for deciphering the functionality of brain. In volume electron microscopy, a voxel of brain may be observed and neurite connectivity reconstructed (thin neurites are still at least 50 nm thick, therefore easily observed by EM). The volume electron microscopy is still a novel approach, being worked on and improved. See (Briggman and Bock, 2011) for a review of current techniques, their advantages and shortcomings.

Recently, Bock et al. have combined *in vivo* imaging of neuronal activity with later *in vitro* reconstruction of neurite connectivity (Bock et al., 2011). Such combined approaches bring further understanding of fine organization and functionality of neuronal networks.

### 1.1.3 Fluorescence microscopy

Fluorescence microscopes rely on illumination of the observed tissue and collection of photons emitted via fluorescence. Due to Stokes shift[3], it is possible to separate the reflected illuminating light from the light coming from fluorescence. One of crucial applications of fluorescence microscopy (especially confocal and 2P) in neuroscience is calcium imaging, with the use of fluorescent calcium indicators. Fluorescent calcium indicators are molecules which make cells react, with the property that the higher concentration of calcium is in the cell, the more light the neuron emits via fluorescence. Due to near-absolute correlation(Kerr and Denk, 2008) between neuronal spikes and large increase in concentration of calcium, it is often possible to see when an observed neuron fires an action potential, because the firing neuron becomes brighter.

In this paragraph, we describe three principally different techniques of deploying calcium indicators to cells.

1. Chemical indicators: In general, the molecules of calcium indicators are injected into the target tissue by a pipette and dispersed. In the most frequently used technique, acetoxymethyl (AM) ester based multi-cell bolus loading (MCBL), indicators based on BAPTA (e.g., fura-2(Grynkiewicz et al., 1985), fluo-4(Gee et al., 2000), etc.) are used as the dye. There is a neat trick allowing better distribution of the dye into neurons: Binding alcohol groups to the carboxyl groups of BAPTA-based indicators, creates an (AM ester)-version of the indicator. The esterized indicators are lipophilic, thus being able of entering cells through their membranes. However, once the indicators enter a cell, cellular esterases cleave off the alcohol groups. This has two consequences: First, the indicator becomes capable of binding calcium (i.e., it becomes a calcium indicator again). Second, the indicator without the alcohol groups becomes negatively charged and therefore it can not exit the neuron anymore via diffusion[4]. Therefore, the calcium indicators are active only after they enter cells and once they do so, they stay there for at least 3 hours. This leads to sufficiently high concentration of calcium indicators in cells and good quality staining. However, the staining is

[3]When a particle absorbs a photon and gets into excited state, it may get rid of its excessive energy by emitting a photon with lower energy back. Stokes shift is the difference between the energy of the absorbed and emitted photon.

[4]It will be eventually pumped out by unspecific ABC pumps after 3-5 hours.

only temporary, due to unspecific ABC pumps transporting the fluorophores out. Another problem is photobleaching: when the tissue is illuminated, the calcium indicators are gradually damaged and they eventually cease to be fluorescent. For a review of AM ester based MCBL, see (Eichhoff et al., 2010).

2. Genetic knock-in: Using the technique of gene knock-in, it is possible to genetically deliver fluorescent proteins into the cells which should be observed. The cells themselves then express the fluorescent protein. For a review of this technique, see (Kotlikoff, 2006).

3. Viral transfection: This technique is somewhat similar to the previous one, but instead of genetic knock-in, a modified virus is used to make cells express fluorescent proteins. This leads to much higher rate of expression than the genetic encoding of proteins via knock-in. This technique leads to very good image quality, enabling imaging of finer structures, e.g., axons (Kuhlman and Huang, 2008). See (Teschemacher et al., 2005) for a review of this technique; see (Heider et al., 2010) for its more recent application.

When compared to electrophysiology, calcium imaging offers better spatial resolution (i.e., it is obvious where neurons are), but somewhat worse temporal resolution of electric signals (the sampling frequency of recording is lower). Similarly to electrophysiology, calcium imaging may be performed *in vivo* and it is less invasive than insertion of multielectrode array. A disadvantage of calcium imaging is the limited depth it can visualize. It is possible to measure certain data from tissue up to 1.6 mm deep (Kobat et al., 2011), but functional neuroimaging is still limited to up to 1 mm (Helmchen and Denk, 2005). As the result, it is still impossible to observe, e.g., thalamus of living mice, using calcium imaging with conventional objectives.

We describe three types of fluorescence microscopes in the following sections.

#### 1.1.3.1 Wide-field fluorescence microscopy

This is the oldest technique of fluorescence microscopy, when the whole observed sample is uniformly illuminated. See Fig. 1.3 for a schema and description of a wide-field microscope, see 1.5 for an example of how data from wide-field microscopy may look. A large disadvantage of this technique is extremely poor z-axis resolution. I.e., neurons 0.1 mm deep will be seen in the image, as well as neurons 0.2 mm deep, etc. This is not a problem when thin sections of tissue are observed, but it makes the technique ill suited for *in vivo* recording.

#### 1.1.3.2 Confocal microscopy

The confocal microscope takes a different approach than a wide-field microscope. Instead of illuminating the whole sample at once, it scans the sample point by point and collects photons via a photomultiplier tube. See Fig. 1.4 for a schema and description of principle of a confocal microscope.

Compared to a wide-field microscope, a confocal microscope has much better z-axis (depth) resolution, see 1.5 for a comparison. This offers several advantages over the wide-field approach. First, signals from various depths of specimen are

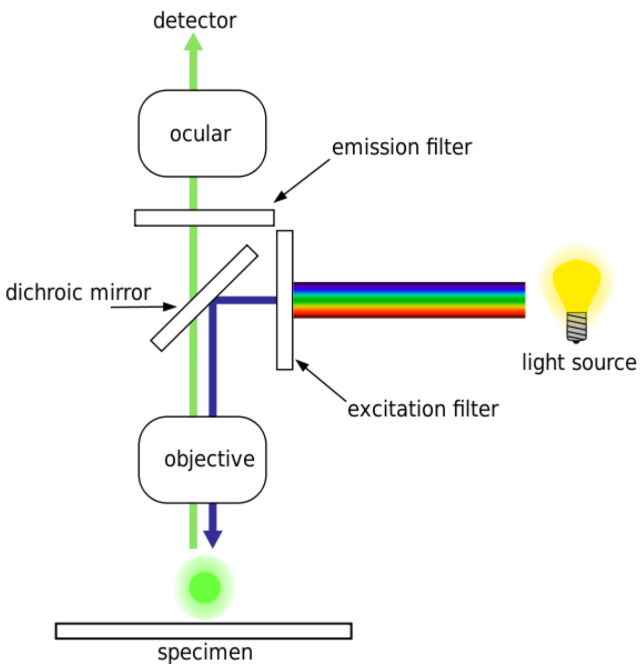


**Figure 1.3:** The arc lamp produces white light, filtered to produce light with a wavelength which evokes fluorescence in the observed sample (the exact color depends on used fluorescent dye). The light with selected color is then reflected at the sample through an objective. This should lead to excitation of fluorophores, with the emitted photons being collected. A portion of the arc lamp light is also reflected back. The photons from the arc light and from fluorophores then hit the dichroic mirror and only the lower-energy photons from fluorophores may pass through, are further filtered and then seen by an eye or recorded by a camera. From (Mühlpfordt, 2008).

not mixed together, only one focal plane is observed at a time. This makes observing single cells much easier than when using a wide-field microscope. Second advantage is the possibility of reconstruction of a voxel of the observed tissue. If a stack of images from various depths[5] is recorded, it may be assembled together into a 3D image of the tissue. An electron microscope may be also used for reconstruction of 3D voxel, but it needs the tissue to be reconstructed to be sliced and recorded slice after slice. A confocal microscope does need such physical slicing and therefore is easier to use for the task.

#### 1.1.3.3 Two-photon microscopy

Developed in the 1990s by Denk (Denk et al., 1994), the 2P microscope offers similar functionality as the confocal microscope. However, it improves its features in several ways and therefore it became an important tool in modern research of *in vivo* brain functionality via calcium imaging, e.g., (Bandyopadhyay et al., 2010; Rothschild et al., 2010). Similarly to a confocal microscope, a 2P microscope is a scanning microscope, but it differs from the confocal microscope in two main ways. First, a two-photon microscope uses pulsed infrared lasers. and two infrared photons are necessary to excite the fluorophore. Second, the two-photon excitation

[5]However, the maximum depth of scanning is still very limited in confocal microscope.

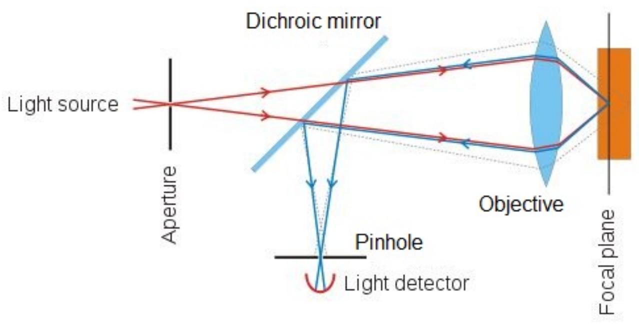


**Figure 1.4:** The laser emits a focused beam of photons, which is targeted at a certain point of the observed tissue in the chosen focal plane (the diagram shows how three different focal planes may be focused). Then, the fluorophores in the tissue emit some photons back. However, not only the fluorophores from the focal plane are hit by the laser ray; also fluorophores above and below the focal plane may be excited. At the first sight, confocal microscopy suffers from the same problem as wide-field microscopy. However, this is where the central trick of confocal microscope comes into play: the detector pinhole. It is crucial to realize that only the photons emitted from the focal plane have such a direction to pass through the pinhole. The photons emitted by fluorophores from above and below the focal plane hit the solid part of the pinhole mechanism and therefore do not pass through to the detector. As a consequence, only the chosen focal plane is observed, when the tissue does not scatter the photons too much. However, when scattering is an issue (e.g., deeper in brain), the scattering leads to some photons originated from the focal plane missing the pinhole, while some photons outside the focal plane may enter the pinhole. Adapted from (Wikipedia.org, 2006b)

principle itself is sufficient to obtain a good z-axis resolution, rather than using a pinhole as in the case of confocal microscope. See Fig. 1.6 for a schema and description of a 2P microscope. We now provide a simplified description of two-photon excitation and then give an overview of the advantages this approach brings.

A fluorophore needs to receive a certain amount of energy from photons be excited and subsequently emit a photon back. In single-photon excitation, such an energy is supplied by a single photon. In two-photon excitation a beam of photons with approximately half the energy, needed for excitation, is focused at a point of the observed tissue. If the concentration of photons is sufficient, it may happen that a fluorophore at the focal point receives energy from more photons at once, thus gaining enough energy to become excited and emit a photon. As the probability of such a simultaneous transfer of energy to fluorophores drops rapidly outside the focal plane, only those molecules in the focal plane are excited and produce signal.

A 2P microscope usually also offers a line-scan recording mode, an alternative to "classical" full-frame recording. Line-scan recording may also be successfully used in calcium imaging of neuronal activity. Illustrated in Fig. 1.7, a laser beam does not raster-scan the entire field of view, but rather follows a specified path instead. I.e., one pass of the scan laser in full-frame recording provides a 2D image, while one pass of the scan laser in line-scan recording provides only a 1D image

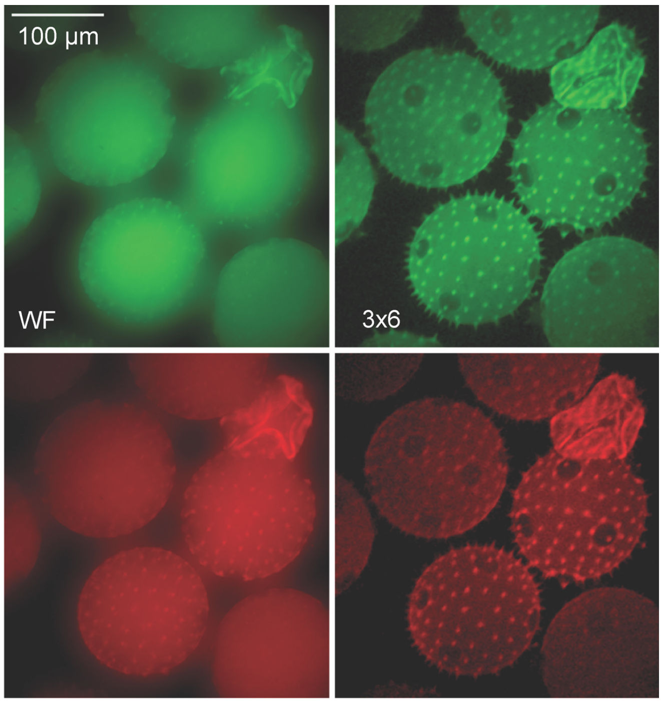


**Figure 1.5:** The same sample (cucurbita pollen grains) as seen by a wide-field microscope (to the left) and by a confocal microscope (to the right). From (Martial and Hartell, 2012).

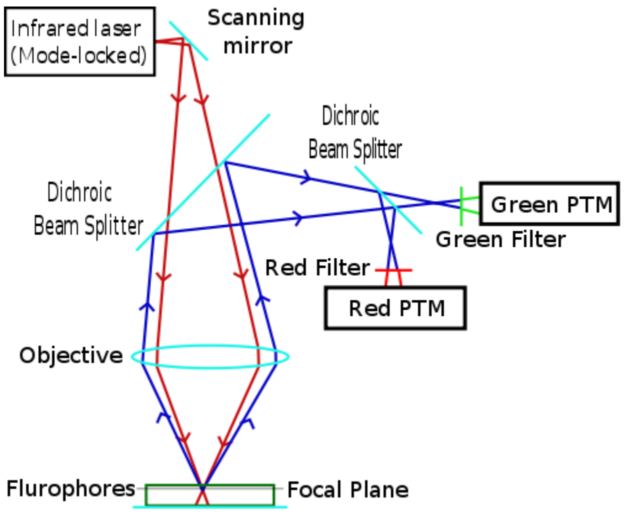


**Figure 1.6:** The infrared laser emits a beam of photons (with lower energy than needed for a single photon excitation) focused at a point in the chosen focal plane. Then, the fluorophores in the tissue emit some photons back. The photons emitted by fluorophores are separated from backscattered photons from the illuminating laser via a dichroic mirror and then further processed. In the case of the microscope on this particular image, the tissue presumably contains tissue stained by two different dyes; the photons emitted by these two types of fluorophores are later separated via another dichroic mirror and they may be processed separately. From (Wikipedia.org, 2006a).

– a single line of pixel intensities. While the outcome of the raster scanning is usually a set of images, using the line-scan mode, a stripe-like image is obtained

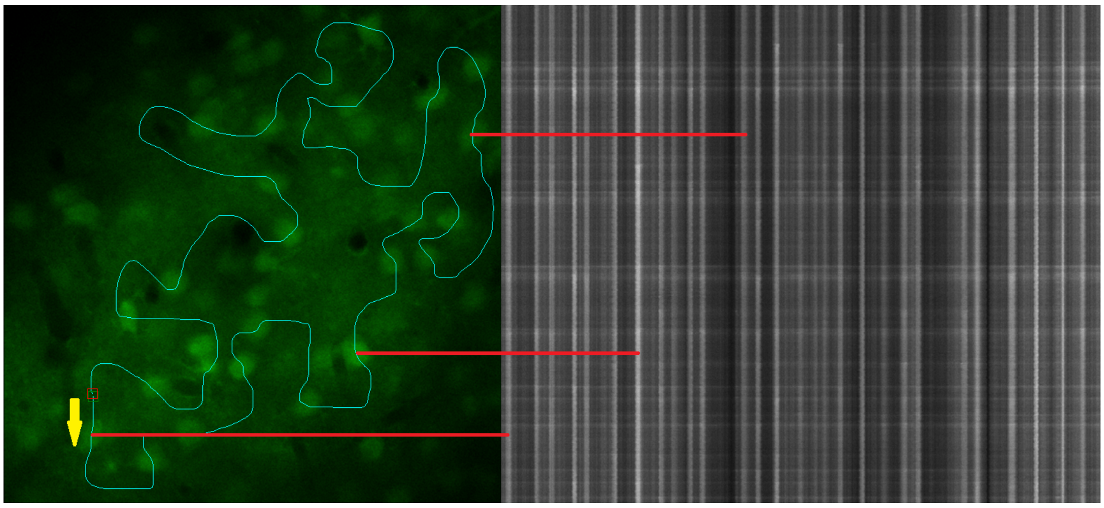


**Figure 1.7:** a) contains a full-frame image of tissue, with scan path demarcated by blue line; b) contains the respective line-scan recording over time, where one line of the data corresponds to a single pass of scan laser through the scan path. The red lines connecting a) and b) show the correspondence between several neurons in the full-frame image and the line-scan recording. The yellow arrow shows the direction of scan path.

(Fig 1.7). Each row of such an image corresponds to a single pass through the path, the numbers of rows therefore correspond to a timeline.

On one hand, the information from line-scan data is not as complete as in full-frame imaging, on the other hand, the data may be collected at a much higher rate. Both full-frame and line-scan recording are viable approaches to calcium imaging and both these methods have their uses.

2P microscopy has two key advantages when compared to confocal microscopy (Helmchen and Denk, 2005). First, the signal is stronger as more photons emitted by the tissue may be collected (the pinhole of a confocal microscope rejects the in-focal photons which were scattered by the tissue). Second, due to longer wavelength of light used in 2P microscopy, it is possible to scan deeper tissue as longer wavelength light is less scattered and absorbed than light with shorter wavelength.

See (Lütcke and Helmchen, 2011) for further information on the technique and recent developments of 2P microscopy.

## 1.2 Related works in processing of microscope data

In this section, we introduce several key problems in computer processing of data from microscopy and list the works related to the topic of this thesis (what we decided to use is to be found in 2.4. The discussed problems are: denoising, extraction of spike trains from video of brain, counting cells and reconstruction of neuronal connectivity. How are the topics relevant to this thesis? We deal with the first two problems directly in our tool, Two-Photon Processor (TPP). While TPP is not aimed at the third, counting cells, we believe it may be used for that purpose too, via its segmentation functionality. Despite the fact that TPP has no functionality for reconstruction of neuronal connectivity, we mention recent work done in that field as it may, in future, bring great insight into how the brain works, which is one of higher purposes of our tool.

### 1.2.1 Denoising

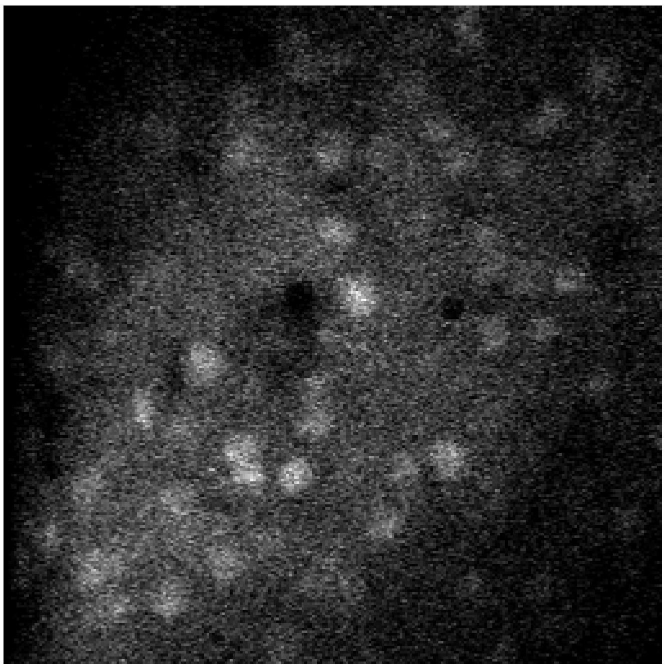


**Figure 1.8:** A sample image from two photon microscope, showing how the noise may look.

Image noise is a random perturbation of intensity and/or color information in the original image. Noise occurs naturally in fluorescence microscopy (and especially in calcium imaging) see Fig. 1.8 for an example of a noisy image.

There is a tradeoff between noise and data acquisition speed: noise can be significantly reduced if slower methods of acquisition are used, but, e.g., in calcium imaging of neuronal activity, the rate of acquisition must be high enough to capture action potentials, which leads to higher levels of noise. Another tradeoff is between the intensity of the excitation beam and noise. If an excitation beam with high intensity is used, more photons are emitted by fluorophores and it is not necessary to rely on photomultiplier tubes[6] so much. However, stronger laser used leads to higher degree of photobleaching (destruction of fluorophores). This may not be an issue when the recording from brain lasts only a few minutes, but in prolonged experiments, it is necessary to use less powerful laser in order to conserve the fluorophores.

For most sorts of automated analysis, denoising of the data is necessary. In the following paragraphs, we list several approaches to denoising.

The basic methods of denoising, Gauss filtering, mean filtering and median filtering rely on a window sliding over the image, using the local information in the window to reconstruct a given pixel (Dangeti, 2003). These techniques are

[6]Vacuum tubes detecting light and amplifying its energy.

very fast to perform. Median filtering preserves the edges of objects rather well (which is useful, e.g., for visualization of morphology of brain), but it does not preserve certain properties of the original image. E.g., if we are interested in average fluorescence of neurons in brain (which is crucial for analysis of calcium traces over time), median filtering may alter the original intensities of objects considerably. On the other hand, averaging algorithms (Gauss, mean filtering) do not preserve edges of objects as well, but they tend to keep the information about intensity in the image.

More advanced techniques include, e.g., anisotropic filtering (Perona and Malik, 1990), Wiener filtering (Yaroslavsky and Yaroslavskij, 1985), Optimal spatial adaptation (Kervrann and Boulanger, 2006) and non-local means (Buades et al., 2010). Especially the non-local means filtering is widely used and algorithms building upon it are being created, e.g., (Marim et al., 2010). While non-local approach leads to significant reduction of noise (with edges of objects well preserved), it may change the information of fluorescence intensity and as a result, it is not too well suited for extracting calcium traces from denoised data. The reason for this is that the non-local means algorithm, when denoising a given pixel, looks for areas in the image with similar surrounding region and bases the reconstruction of the pixel's intensity on these regions. Thus, when a pixel in a neuron is reconstructed, it is reconstructed using intensity information from other neurons (possibly with different intensity or gradient of intensity), which can change how the neuron looks considerably.

The third family of approaches uses temporal information, i.e., it uses a sequence of images instead a single image. If the recorded area is stationary, simple averaging of more frames leads to good reduction of noise (without altering the edges in any way). However, if occurrences of a very short phenomena are to be observed in the sequence of images (i.e., phenomena present in a very few consecutive images in the sequence), this approach obviously leads to loss or blur of such information. Another recent approach to noise reduction using temporal information (even taking the effect of photobleaching into account) may be found in (Boulanger et al., 2009).

### 1.2.2 Extraction of spike trains from video of brain

The technique of calcium imaging makes it possible to record calcium traces of neuronal activity over time, however, this alone is not sufficient for understanding the signallization between neurons. The brain uses action potentials (spikes) in its computation, which makes it vital to infer spike trains[7] from the calcium traces.

Traditionally, this task is solved partially manually. Usually, the researcher takes the first image from the video and encircles regions of interest (ROI) containing neurons (Dombeck et al., 2009; Kerr et al., 2005; Niell and Smith, 2005). Then, calcium traces (intensities over time) are recorded from these ROI and specialized algorithms for inferring spike trains are used to obtain spike trains from the ROI. Such an approach suffers from two general problems: First, it is rather time-consuming. Second, it is subjective, as it is not always obvious what is a neuron and what is not. Therefore, the overall result of a measurement of brain is highly dependent on the researcher who selected the ROI.

[7]Times of action potentials assigned to neurons.

Despite the two problems described above, the traditional approach using ROI is widely used and it is feasible when used with *in vitro* data. However, we argue that it is not nearly as suitable for work with *in vivo* data. The main problem is with movement of the observed tissue. Motion artifacts, both in z-axis and in focal plane occur and there is no way of avoiding them entirely. Therefore, if a researcher draws tight circles around neurons and the neurons later move, a major part of the measured intensities will be background noise as the neurons slide away from the ROI. The resulting SNR (signal to noise ratio) may be so low that it renders the measured data useless. With larger shifts in the tissue, it is possible for the neurons to leave the drawn ROI entirely.

A researcher may draw loose circles around neurons, making sure that even if the neurons move a bit, they will be still in their respective ROI. This partially solves the problem with neurons leaving the ROI during the experiment[8]. However, a new problem appears instead. When too large ROI are used, the signal from neurons is diluted by the surrounding regions and the SNR ratio drops significantly, because a large part of measured information is noise from the area around the neuron. Furthermore, the area around neurons usually contains other neurites, even from other nearby neurons; therefore, if nearby neurons fire action potentials, the area around them may become brighter due to increased concentration of calcium. Therefore, if a neuron with large ROI drawn around it does not fire, but near neurons do, the whole area may become brighter, which will lead to a false positive detection of an action potential in the calcium trace of the original neuron.

One could propose to discard the thought of using large ROI, draw tighter ROI and then use automated tracking methods to track these objects' movement throughout the video. This is probably an approach which would work much better than the idea of static ROI, but it suffers from the fact that due to deformation of the tissue[9], a neuron may disappear from the video (leaving the focal plane) or a new neuron may appear (a neuron originally being outside the focal plane entering it). Thus, the idea of simple tracking of manually drawn ROI is not without major faults either.

With new objects appearing and some objects disappearing in the image, it would probably be necessary to draw the ROI in every frame of the video (or whenever an object appears or disappears), which is too time consuming to be done manually on a larger dataset. This calls for automated processing . Such a need has been recognized in (Kwan, 2010) and partially addressed in CellSort software(Mukamel et al., 2009) and in ISA (Wong et al., 2010). Both works leave much to be desired from the point of view of functionality and especially with CellSort, it is not always easy to find a set of parameters which does not lead to a crash of the software. We discuss the problems of the CellSort in section 3.2.3 and problems of ISA in 3.2.2. To the best of our knowledge, there is no other tool which could automatically or semi-automatically detect neuronal bodies in a video, measure their calcium traces over time and then infer spike trains from these traces.

There are several nontrivial subproblems of automated extraction of spike

[8]This approach is not robust to larger shift of the observed tissue though.

[9]Such a deformation may be caused, e.g., by blood being pumped in a artery nearby, or a hiccup of the animal.

trains from a video: segmentation (determining what is a neuron and what is not), tracking (keeping track of neurons over time and taking newly appeared neurons into account) and inferring spike trains from calcium traces (determining which increase in calcium traces is a spike and which is only noise). While there are works solving one of these subproblems, the only attempts to connect them into a processing pipeline seems to be the above mentioned CellSort and ISA. The following sections discuss related work in the fields of these subproblems.

#### 1.2.2.1 Segmentation

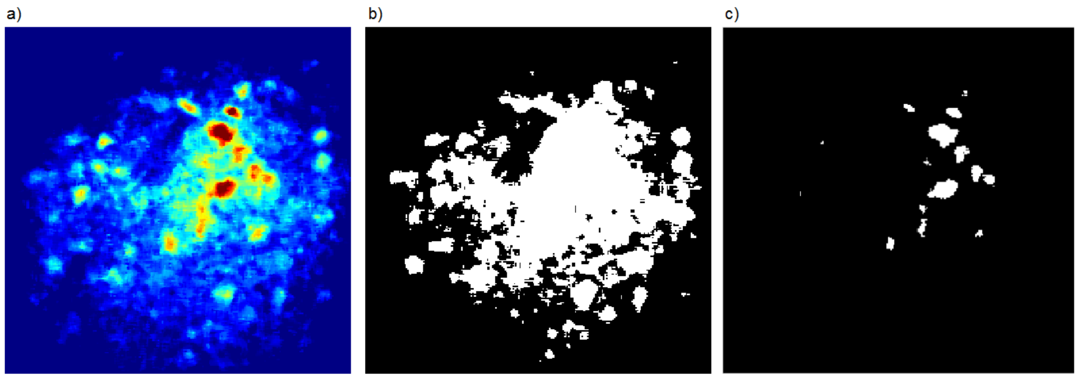


**Figure 1.9:** A denoised image from 2P microscope is displayed in a) as a heatmap. The image thresholded using lower threshold is in b) and the image thresholded using higher threshold is in c). The situation when a lower threshold is used leads to the relatively dark border neurons of the neuronal group being segmented correctly. However, central neurons of the group are merged into one and the segmentation is completely wrong there. On the other hand, when a higher threshold is used so that the bright neurons of the central neuronal group are segmented correctly, the darker neurons are missed entirely and the overall quality of segmentation is also very low. It is important to realize that with as uneven background intensity as in the original image, there is no way of finding a single threshold that could properly segment such an image.

If calcium traces of neurons are to be measured, it is crucial to segment the image and obtain the information which parts of the recorded tissue are neurons and which are not. We start with the description of two traditional, “elementary” approaches. Then, we mention two important toolkits for segmentation of *in vitro* data, followed by several recent approaches to cell segmentation.

Traditional, “elementary” approaches are *thresholding* and *watershed* techniques. Thresholding is historically dominant approach and it is still widely used today (Meijering, 2012). It relies on the fact that in calcium imaging, cells are usually brighter than their surroundings. Therefore, a threshold value is chosen (or detected automatically) and all pixels lighter than the threshold are considered to be neurons and the rest to be background. Examples of thresholding algorithms are, e.g., Ridler-Calvard (Ridler and Calvard, 1978), Otsu (Otsu, 1975) or mean thresholding. Thresholding approaches may work very well with *in vitro* data with uniformly dark background, as demonstrated in (Coelho et al., 2009). However, this technique is much less suitable for *in vivo* data, where the intensity of background may be uneven. The problem is illustrated in Fig. 1.9.

The idea of watershed is based on seeing the image as a scenery with peaks (dark areas) and valleys (light areas). In such a scenery, seeds are placed into local minima (valleys), representing sources of water. Then, water starts springing from the sources and starts "filling the basins", forming lakes. When two such lakes meet, a border is drawn between them. Later, the areas between borders are labelled as "cells". The original algorithm is described in (Beucher and Lantuejoul, 1979). A variant, *marker-controlled watershed*, may be used to segment background of cells specifically, so that the whole image is not completely divided into cells, but it is split into putative cell objects and background. Unfortunately, the task of automated placement of seeds is difficult in images with uneven intensity of background (similarly to thresholding). Furthermore, even with near-ideal data with constant background intensity, marker-controlled watershed leads to severe oversegmentation (Coelho et al., 2009).

There are certain toolkits which contain several different segmentation techniques, e.g., ImageJ (Collins, 2007) or BioImageXD (Kankaanpää et al., 2012), however, these are mostly suitable for *in vitro* data, as according to our experiments, even the techniques supposed to cope well with uneven background tend to perform badly.

Apart from the simple approaches, there are more sophisticated techniques, some of which we mention in the following paragraphs.

The algorithm used in (Mukamel et al., 2009) relies on independent component analysis of temporal signal in a video. Unfortunately, the approach suffers from several problems, described in 3.2.3.

The regression-based algorithm described in (Miri et al., 2011) tracks the intensity of pixels over time and from that, it determines which pixels belong to neurons and which belong to background. Only the active neurons are therefore observed. The approach relies heavily on a stationary image and as motion artifacts often appear in *in vivo* data, it is uncertain how this approach works with them. Also, the brightness of neuropil[10] is often tightly coupled with brightness of nearby neurons, which poses a question, whether the algorithm can tell brightness changes in neuronal somata from brightness changes in nearby neuropil.

Algorithms based on machine learning (Valmianski et al., 2010; Lin et al., 2003) should cope well with the problem of uneven background intensity, but they require annotated training sets and the training and performing the segmentation tend to take a lot of time (at least $10^3$ more times than simple thresholding).

The more sophisticated techniques described above have a nontrivial time complexity (up to 1 minute per 512x512 image in (Valmianski et al., 2010)). Even though they may significantly reduce the amount of manual work, they are insufficient, e.g., for real time processing, which may become an important application of segmentation in future (we discuss this matter in 3.2.3). Bearing in mind the necessity of speed, as well as with the necessity of being able to cope with uneven background intensity and other problems inherent to *in vivo* data, we designed the algorithm SeNeCA (Search for Neuronal Cells Accelerated), described in 2.4.1.3 and evaluated in 3.1.

[10]Synaptically dense area of brain with low concentration of actual neurons. It may contain, e.g., axons and dendrites in high concentration.

#### 1.2.2.2 Tracking

Once we have obtained the locations of cells in different frames of video via segmentation procedures, we might be interested in relating objects found in a single frame to the objects found in the next frame. There are two basic approaches to tracking cell movement: Segmentation-based tracking (where each frame of video is segmented and the objects are assigned to one another across frames) and segmentation-free tracking (the segmentation is performed only at the beginning of the video and the objects are then moved, without need of complete segmentation)(Meijering et al., 2006). However, in *in vivo* data, neurons may leave or enter the imaged focal plane; that is a problem that calls for segmentation of every frame as it is necessary to keep track of newly appeared neurons.

The choice of tracking algorithm depends on several criteria and we shall describe several options and discuss their usefulness for *in vivo* calcium imaging.

A very simple tracking algorithm is *nearest neighbor tracking*. The basic idea is the following: Given two consecutive frames of video (in times *i,i+1*), find the nearest neighbor of every neuron in time *i* in the frame in time *i+1*. These two objects will be considered the same. With *in vivo* data, this alone could work rather badly. If a neuron which was present in the frame *i* disappears in frame *i+1*, the nearest neighbor of the former object may be found very far and it is, obviously, a different object. There is a simple remedy to this problem: Nearest neighbors are searched for in a given radius around the original object and if no suitably near neighbor is found in time *i+1*, it is presumed that the object has disappeared. Nearest neighbor tracking is a robust and useful technique, if the rate of image acquisition is sufficiently high to assure that a single object may no move too far between two frames. When objects can move too far between images, it is possible, e.g., that cells A,B move between frames in such a way, that A takes the position of B, while B moves further away. The simple nearest neighbor tracking would incorrectly output that A has disappeared, B has stayed in its place, and a new cell has appeared (while it is in fact B).

If objects in the image move in the same direction (e.g., blood cells in an artery), it is possible to determine the best cross-correlation of frames to each other and find the optimal alignment according to that. However, when measuring neurons, this sort of movement is not very frequently seen.

If cells retain their shape and/or intensity across frames and they are sufficiently differentiated in these features between one another, it is possible to take a sub-region of the frame in time *i* and, using, e.g., cross-correlation, find the best local alignment to a region in the frame of time *i+1*. Unfortunately, the neurons observed via calcium imaging move in and out the focal plane, thus changing both shape and intensity. Therefore, the technique of local alignment is not particularly suited for tracking neurons in living brain.

A recent graph-based solution tries to find the optimal assignment of objects between two frames as a maximum weighted matching in a bipartite graph (where each partite corresponds to a single frame) (Mosig et al., 2009). For nondeterministically moving cells (which is what happens in *in vivo* data), this technique looks very promising.

For a further overview of tracking techniques, see (Miura, 2005). See (Hand et al., 2009) for a review of recent state of the art software tools for tracking.

#### 1.2.2.3 Inferring spike trains

Once the neurons in a video are segmented and tracked, it is trivial to measure some form of their calcium traces (e.g., average intensities of objects over time). However, due to amount of noise in the signal, it is not a trivial matter to infer spike trains from such calcium traces, i.e., determine which increases in a calcium trace is a spike and which is only noise. The are not many methods for spike train inference and these methods are generally quite complicated. Published algorithms may be found in (Grewe et al., 2010; Vogelstein et al., 2009; Dyer et al., 2010). As developing new methods for spike train inference is out of the focus of this thesis, we do not discuss the topic further.

### 1.2.3 Counting cells

It is often important to know the concentration of cells in a tissue and, unlike other techniques mentioned in this section, this is not limited to neurons. A common task is measuring concentration of bacteria in blood of an organism over time, which may give a crucial information on progression of a disease. Another common task is measuring the amount of different cell types in blood (red cells, white cells, etc.). This may be used to detect pathological conditions of the organism, e.g,, inflammation. In conditions with loss of cells, e.g., Alzheimer's disease, counting living neurons cells over time gives information on the progress of the disease. Counting cancer cells over time, as they multiply, may give us insight into the speed of tumor growth. Counting cells may also give important grounding to computational models of cell systems.

Counting is also not limited to a single type of microscope; it may be performed with data from electron microscopy, fluorescence microscopy, or any other type able to visualize single cells. There are even automated cell counters which rely on other than microscopic methods to extract the number of cells in a tissue, however, these are limited only to *in vitro* samples. On the other hand, fluorescence microscopy makes it possible to count cells *in vivo*. This may become a powerful technique in research of development of brain: chronically implanted two-photon microscope observing the brain stained via viral transfection may prove an insight into development of brain, into the evolution of concentration of neurons and glia.

### 1.2.4 Reconstructing neuronal connectivity

Determining the 3D connectivity of neurons in brain is an important step in successful reverse engineering of brain (Peng et al., 2010; Wu et al., 2011). Due to development of chronically implanted[11] two-photon microscope, it may even become possible to observe the evolution of 3D morphology and connectivity in living brain. However, the task of structural reconstruction is still often solved manually, using single-unit recording, e.g., (Vogt et al., 2005). Despite recent effort and advances, the problem of larger automation of the reconstruction process is still not satisfactorily solved. In the following paragraphs, we list recent contributions to the topic.

[11]Permanently fixed in the skull of observed animal in such a way that the animal may live and move freely.

An electron microscope may be used to reveal even very fine morphological properties, but it is difficult to obtain a voxel representing the measured sample. The sample must be fixated and sliced, before the single slices may be observed via the microscope. However, the technique may be successfully used, as in (Bock et al., 2011).

With developments in fluorescence microscopy, it became possible to visualize axons and dendrites. The key is to use good staining techniques and smaller depth of scanning, so that noise is reduced. In recent years, several tools were created to facilitate the reconstructions of neural circuitry from fluorescence microscopy (Yuan et al., 2009; Rodriguez et al., 2009; Vasilkoski and Stepanyants, 2009; Zhao et al., 2011; Peng et al., 2010).

# 2. Software materials and methods

This chapter is dedicated to the Two-Photon Processor (TPP) toolkit. The first section contains a brief overview of its features. The second section discusses the choice of programming language. The third section introduces its software architecture. The fourth section explains the mechanisms and key algorithms which TPP uses when solving various tasks related to its purpose. The content of this chapter is supplemented by the user guide and technical documentation.

The author of this thesis would like to note that Mgr. Ondřej Novák implemented the step 5 of 2.4.2.1[1], steps 3 and 4 of 2.4.4.1[2] and steps 1,3,4 of 2.4.5.1; the author of this thesis incorporated the functionality into the software architecture and GUI of TPP. Mgr. Ondřej Novák also implemented the generator of artificial data used in 3.1.4.

## 2.1 Feature overview

TPP provides:

- Automated extraction of calcium traces of neurons from both full-frame and line-scan data. Fast mode may be used, which allows the extraction of calcium traces in delayed real time[3].
- Algorithms for denoising the measured calcium traces.
- Algorithms for inferring spike trains from calcium traces.
- A tool for planning a near-optimal scanning path if line-scan mode is to be used.
- Several methods of visualization of measured data:
  - Calcium traces and inferred spike trains of observed neurons.
  - Receptive fields of observed neurons.
  - Colored dots painted over neurons, based on selected neuronal features (e.g., the neuron's "favourite" frequency).
- Other qualities:
  - Possibility of controlling the software both from GUI and from Matlab console.

[1]This feature was added later, in reaction to one of reviews in Journal of Neuroscience. The author of this thesis could not code the task himself due to time constraints

[2]It was necessary to have a working version of TPP in early 2012 and it would not be possible for the author of this thesis to finish the code in time.

[3]The processing speed is sufficient, but there is a several milliseconds long lag, before a video frame is saved to HDD by a microscope.

  - Well commented and documented code.
  - A comprehensive user guide.
  - Automatically tested code.

## 2.2 Discussion of choice of programming language

TPP has been mostly[4] written in the Matlab programming language. The main reason for choosing Matlab was the fact that it is a standard tool for scientific computations and most laboratories interested in our software should be able to use, understand and modify it with ease, compared to more "traditional" programming languages, such as Java or C++. Furthermore, Matlab contains a wide array of ready-to-use libraries from many different disciplines, which we expected to facilitate the development of TPP.

The choice of Matlab had several positive sides, which we discovered throughout the process of developing TPP: Matlab is very well documented and offers many useful functions out of the box and many more are created and shared by its users. The author's subjective feeling is that using Matlab libraries and content shared by users is much easier than in Java, C, or C#. Due to ease of using pre-prepared functions and easy-to-use inbuilt debugger and profiler, the development of software in Matlab was rapid and generally effective (we mention certain problems in the next paragraph). Despite the fact that the initial background research on the performance of Matlab computing suggested that Matlab could be very slow, compared to, e.g., Java, we found out that the aspects in which Matlab is noticeably slower (mainly the work with dynamic objects) are not an issue, as such problems will not be encountered at all during the software development. On the other hand, in functionality relevant to our work (e.g., filtering images, matrix operations), our preliminary experiments have shown that while Matlab is mostly slower than pure C, it is often faster than Java or C++.

However, Matlab also has numerous negative sides: A major problem is the difficulty of developing larger projects. While Matlab IDE is not downright bad, it lacks many useful features for code organization and navigation, present, e.g., in Netbeans, Eclipse or MS Visual Studio. Some features are too slow to be used efficiently (e.g., code completion). While Matlab does support objects, the work with dynamic objects is extremely slow. All this combined makes it difficult to write larger amount of organized code. Another problem that Matlab suffers from, is the absence of several "common" programming concepts, e.g., pass-by-reference, work with pointers and common dynamic data structures, such as linked lists or self-balancing trees[5]. The last major issue we discovered was very difficult

[4]When it became obvious that if a certain crucial function was sped up, real time extraction of calcium traces would be possible, we wrote a version of the function in C and connected it to Matlab via MEX interface.

[5]Except the work with pointers, there are workarounds to these problems. Passing by reference may be done when the structure to be passed is wrapped in a dynamic class, inheriting `handle` class. However, doing this slows the passing of parameters considerably. Second, the absence of common data structured may be worked around by using Java classes, which are rather easy to use from Matlab.

and time-consuming development of programmatic GUI.

Overall, we believe that even when ignoring the factor of usefulness of the tool to other laboratories, the choice of implementing TPP in Matlab was a right one, despite all the problems. Most importantly, we believe that had we used a different programming language, the resulting amount of functionality would be much lower as much more time would have been spent on writing functions which are natively present in Matlab.

## 2.3 Software architecture

The general architecture of TPP is shown in Fig. 2.1. We strictly separated computational procedures from the GUI as it is widely considered to be a good practice and it leads to better organized code. In terms of common software architectures, our software uses MVVM architecture (Model View View-Model, a specialization of Model-View-Controller). However, there is no clear distinction between the View and View-Model components as it would be difficult (and not very useful) to separate these in Matlab.

The core functionality of TPP is represented by several static classes, each taking care of a family of related problems the software solves. Even though using static classes to group related functions is not a widely used concept in Matlab programming, we believe that it is very useful in organization of the code and that in developing this large project in Matlab, it is almost a necessity. We considered basing our software on dynamic objects at the design stage of the project, but we rejected the idea for several reasons: First, Matlab style of work with dynamic objects is not very comfortable. Second, computations with dynamic objects may be very slow. Third, it seemed not necessary to use dynamic objects in our project at all.

There are two extensions to the core functionality. For the reasons listed below, we did not include them in the GUI, but they are runnable as Matlab scripts.

## 2.4 Software description

The TPP toolkit offers a functionality for several families of tasks. This section brings insight into how the given families of problems are treated. In each of the family of problems, we start with a short introduction to the problem, followed by description of how we solve it - the *processing pipeline*. The we discuss the choice of algorithms and explain more complex steps of the processing pipeline in greater depth. If there are nontrivial algorithms of our own design which we had used in the processing pipeline, we describe them after the discussion.

### 2.4.1 Extracting calcium traces from full-frame data

Recording full-frame data is probably the most used format of recording brain activity in calcium imaging. An example of a frame of the recorded video is in Fig. 1.8. Below, we describe the general process of extracting calcium traces from such a sort of data and we discuss the choices we made, concerning the choice of algorithms. We conclude this section with the description and discussion of

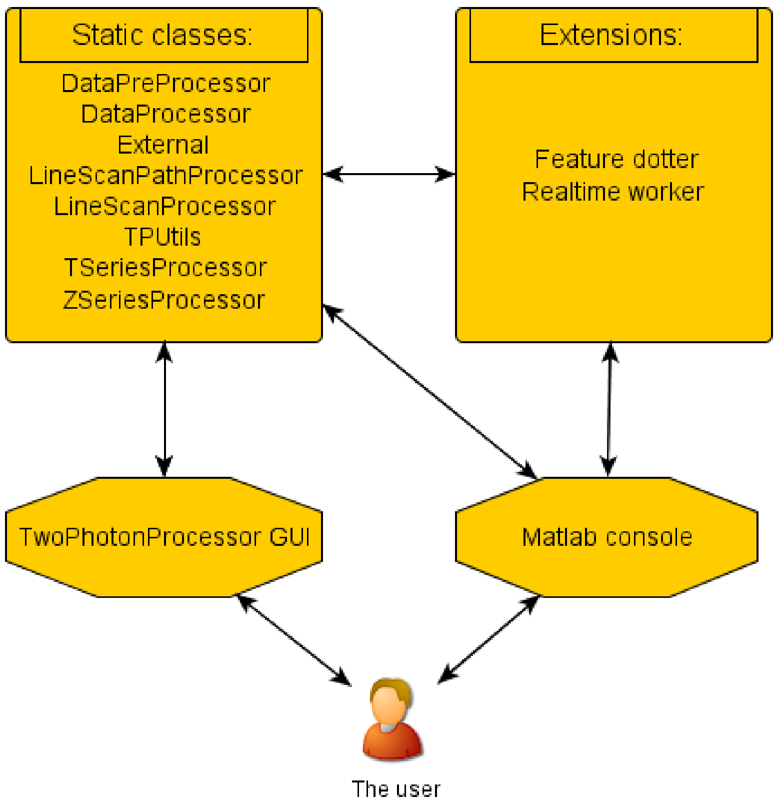


**Figure 2.1:** The user may communicate with core classes of TPP either via Matlab console or GUI. Certain extensions of the original software are runnable only via the console.

three nontrivial algorithms we had designed: SeNeCA, Type 1 defusion and Type 2 defusion.

#### 2.4.1.1 The processing pipeline

1. Load a video represented by TIFF images into memory.
2. Choose the method and parameters of denoising (disk,square or rectangle blur, or median filtering).
3. Set the parameters for future segmentation by SeNeCA.
4. Process the video sequence to obtain calcium traces of neurons. For each frame (in time $i$):
    (a) Average the frames $i-j, i-j+1, ..., i, ...i+j-1, i+j$; $j$ is a parameter, it may be 0.
    (b) Find putative neuronal somata in the image from previous step via SeNeCA segmentation. It is possible to attempt splitting fused neurons. Too small or too large objects may be filtered out.

```
        (c) Assign the neuronal objects from the previous step to
            the neuronal objects found in frames $1, 2, ..., i - 1$ via
            tracking.
        (d) Save average and sum intensities of the neuronal
            objects found in frames $1, 2, ..., i$.
    5. Return the intensities of neuronal objects over time.
```

**Pseudocode 2.1:** The process of extracting calcium traces from full-frame data.

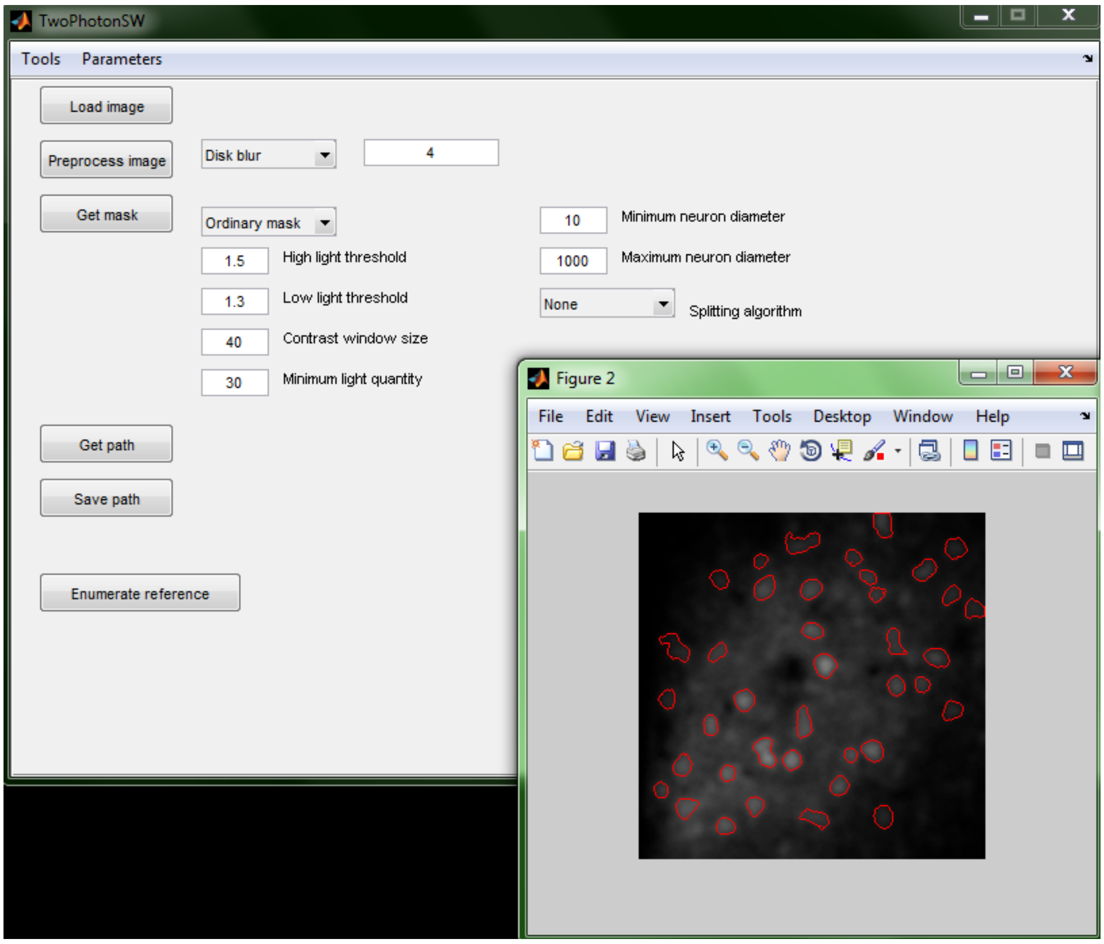
#### 2.4.1.2 Discussion of the processing pipeline

In the step 2, the user may choose between median filtering and disk, square or rectangle averaging. This step is used to get rid of noise which would confuse the later used segmentation algorithm. Despite the existence of more complicated algorithms of denoising, these algorithms may not be suited to our needs (see 1.2.1. The algorithms included in TPP work well enough and they are computationally very cheap.

In the step 3, the researcher is supposed to determine a good set of parameters for the segmentation to be done, see Fig. 2.2 for an example of the result. A pre-prepared recommended set of parameters which works reasonably well for a wide scale of data may be used, but manual tuning of the parameters is possible as well. This is not as comfortable as fully automated segmentation, however, our parameters have clear meaning and we supply a description of how to tune them to get the desired result in the User guide (such a tuning obviously can not be done with fully automated methods). Furthermore, fully automated segmentation algorithms with good enough performance on *in vivo* data do not exist, according to our knowledge. After the parameters are tuned on an image from the video, the given set of parameters will be used for the segmentation of all the frames. This relies on the presumption that important properties of the video will be generally consistent across time; the presumption generally holds though.

The step 4a serves to make the segmentation algorithm more stable. If $j$ is chosen larger than 0, the algorithm does not segment a single frame of the video, but an average of several frames around the given frame instead. This leads to elimination of certain motion artifacts, but it may lead to loss of certain signal too. Setting $j$ larger than 0 is suggested only in the case of data heavily corrupted by noise, when the averaging of several frames is the only way of obtaining valid calcium traces.

In the step 4b, the image is segmented and locations of neurons thus obtained: a *mask* is computed. The mask is a matrix of the same size as the original image and it contains $-x$ inside the $x$-th found neuron and $x$ on the border of the $x$-th neuron. The SeNeCA algorithm is used for the segmentation and it is described in 2.4.1.3. As none of the algorithms described in 1.2.2.1 was suited for the task of fast segmentation of *in vivo* data, we had no choice but to create our own



Disk blur 4

Ordinary mask

1.5 High light threshold

1.3 Low light threshold

40 Contrast window size

30 Minimum light quantity

10 Minimum neuron diameter

1000 Maximum neuron diameter

None Splitting algorithm

**Figure 2.2:** An example of a segmented video frame. The red curves demarcate neuronal borders. The parameters used are displayed in the GUI behind the segmented image.

algorithm which would fit our needs.

After the segmentation has been performed, it is possible (if the user desires so) to attempt splitting two or more neuronal somata which have been labeled as one (this happens, e.g., if two neurons are very close to each other). Two algorithms may be used: Type 1 (splitting based on shape), described in 2.4.1.4.1 and Type 2 (splitting based on intensity) described in 2.4.1.4.2. After the splitting is done, it is possible to filter objects according to their size: Too small or too large objects may be removed (as these are likely not neurons and therefore it is not necessary to record them). The minimum and maximum size of objects are both selected by the user.

The step 4c makes sure that objects found in the current frame are correctly assigned to the objects found previously: this is crucial for coherence of calcium traces over time; the trace of a single neuron must be assigned to a single object throughout the processing of video. We decided to use an algorithm based on *nearest neighbor tracking*, because of the sufficiently high acquisition rate of data. With the acquisition frame rate of 50-100 Hz, a neuron may not move more than few pixels between two frames, therefore the nearest neighbor tracking has good enough performance and it is computationally cheap. As described in 1.2.2.2, we modified the basic nearest neighbor to be able to cope with new objects appearing.

The modified nearest neighbor rule is simple: For a newly discovered object, find the nearest object found in previous frames (the distance is defined as the distance of centroids). If the nearest neighbor is closer than a given threshold (specified by the user), it is considered to be the same object. If it is further, it is considered to be a newly found object.

In `4d`, the intensities of objects found in the current frame are recorded and assigned to previously found objects according to tracking. While the objects are searched for in the denoised image, the intensities are recorded from the original image, to prevent distortion of the data by the used denoising algorithm. Two sorts of intensities are recorded: average intensity of an object (an average intensity of its internal pixels) and sum intensity of an object (sum intensity of its internal pixels). It is a matter of discussion what should be done with the objects which were previously found, but were not found in the current frame. This may mean, e.g., that a neuron moved out of the focal plane. However, it may eventually return and be rediscovered again. For this reason, if a previously object is not found in the current frame, the last location of the object is recalled and recorded. An alternative would be to record the object's intensity as 0 when it does not appear in the current frame and if the object reappears in future, start recording the intensities as usual. However, the large jump at the moment of rediscovery, from 0 to the intensity of the reappeared object could confuse the algorithms for inferring spike trains from calcium traces into thinking that such an event is a spike, which would be likely false. Therefore we decided to record the last known location of an object which does not lead to such a large jump in intensity when an object reappears.

#### 2.4.1.3 SeNeCA

We designed the SeNeCA segmentation algorithm with several considerations in mind. First, it was necessary for the algorithm to handle well uneven background intensity of the processed images. Second, it should be able to cover neuronal bodies well, not only their central parts, or, on the opposite, capture the area outside the neuronal bodies. Third, the algorithm should be fast (less than a second per 256x256 image). Fourth, if possible, the algorithm should be tunable via a set of parameters so that, e.g., only well stained in-focus (i.e., with high contrast to the background) neurons are segmented. The following algorithm, fulfilling all of the four requirements has been created (further explanation is below the box with pseudocode):

```
SeNeCA
Inputs: $image, highLightThreshold$, $lowLightThreshold$,
$contrastWindowSize$, $minimumLight$
Outputs: $mask$

Algorithm:

  1. Precompute the matrix $lightAverages$, of the same size as
     $image$, defined as $image$ filtered by a mean averaging
     filter with height and width $contrastWindowSize$. A mirror
```

```
   projection back to the image is used when a pixel outside
   the image should be taken into account in the computation
   of $lightAverages$.

2. Compute $lightPoints$ as the set of all points $(i, j)$ with the
   following property: $(image(i, j) \cdot highLightThreshold >$
   $lightAverages(i, j))$ && $(image(i, j) > minimumLight)$

3. Initialize $mask$ to be a matrix of the same size as $image$,
   filled with zeros. Initialize $neuronNumber$ to 1.

4. Foreach $(xCor, yCor)$ in $lightPoints$:

   (a) If $(mask(xCor, yCor)! = 0)$, continue with the next point
       in $lightPoints$ immediately.
   (b) Otherwise set $mask(xCor, yCor)$ to $-neuronNumber$ and
       run a 8-connected BFS wave in $image$ from $(xCor, yCor)$,
       setting the visited pixels to $-neuronNumber$. The wave
       may stop at a point $(a, b)$ for three reasons:

         • The following condition holds: $(image(a, b) \cdot$
           $lowLightThreshold < lightAverages(a, b))$
         • $(a, b, )$ is at the edge of the image
         • $mask(a, b)$ contains a positive value, other than
           $numberNeuron$ in its 8-neighborhood.

       When the wave stops at the point $(a, b)$, set $mask(a, b)$
       to $neuronNumber$.
   (c) After the wave has stopped entirely, increment
       $neuronNumber$.

5. return $mask$.
```

**Pseudocode 2.2:** The pseudocode of SeNeCA segmentation algorithm.

Let us explain and discuss the nontrivial steps of the algorithm.

In step 2, the structure *lightPoints* is created to contain points which are *highLightThreshold*-times lighter than the area around them. This is where *lightAverages*, containing the information of the overall lightness of certain surroundings, is used[6]. The second clause in the condition is used to reject overly dark objects in badly stained regions. The *lightPoints* structure should contain at least a few pixels from each neuronal soma in the image. However, if only this step was used (as a sort of dynamic thresholding), the segmentation would not be good enough. The reason is that if too high threshold was used, the somata

[6]To give an idea of magnitude of the constants used, in an image of 256x256 pixels with neurons circa 20-30 pixels wide, we suggest using using *contrastWindowSize* of about 40 and *highLightThreshold* 1.5-2, depending on the contrast of staining.

would be only partially covered (with small area recorded, the resulting calcium traces are more prone to be ruined by noise). On the other hand, with too low threshold used, it is likely that even non-neuronal objects would be labeled as such. Therefore, SeNeCA uses two thresholds - one to find very light points within neurons (providing bad coverage of the somata so far), which will be extended into good covering of somata via appropriately interrupted wave running from the given light points (in a way, SeNeCA may be viewed as a combination of dynamic thresholding and appropriately interrupted watershed).

In step 3, $mask$ will be the resulting matrix determining what are areas inside somata (marked with $-x$ in the $x$-th neuron), what are borders of somata (marked with $x$ in the $x$-th neuron) and what is an area outside (marked with 0). The variable $neuronNumber$ represents a "color" used when a wave is expanded from a neuronal core and paints the entire object segmented.

In step 4a, it is necessary to prevent more waves being ran from a single neuronal core for computational efficiency. More points from $lightPoints$ may be inside a single neuron (it is very likely) and it would be pointless to run the wave from all of them. However, after the wave is ran from a pixel inside a found object, it "paints" the neuronal insides and borders in $mask$. Therefore, after the wave is ran from the first pixel of an object in $lightPoints$, all other pixels from the given object in $lightPoints$ will be nonzero in $mask$.

In step 4b, the wave paints the visited area with the current "color" specified by $neuronNumber$. The wave may stop at a point under three conditions. The first condition is when the wave reaches a pixel which is not sufficiently lighter than its surroundings. This is the most frequent way the spreading of the wave can be terminated. The second condition is when a border of the image is hit. The third stopping condition terminates the wave when it hits the border of another neuron.

In step 4c, the wave demarcating object with number $neuronNumber$ has finished flowing and the segmentation of the object has been therefore finished. The next object will be painted with another color, $neuronNubmer + 1$.

The following theorem analyzes the time complexity of the algorithm.

**Theorem 1.** *The algorithm SeNeCA has time complexity of $\mathcal{O}(p \cdot cws)$, where $p$ is the number of pixels in the source image and $cws$ is $contrastWindowSize$.*

*Proof.* Let us analyze the SeNeCA algorithm step by step. If a sliding window implementation of mean filtering is used in computation of $lightAverages$ in step 1, it is first necessary to compute the window around pixel $(1, 1)$, which takes $cws^2$ time. However, this is trivially dominated by $p$ as it makes no sense to use $cws$ larger than the size of the image. Then, for each pixel other than $(1, 1)$, $cws$ pixels are added to the window and $cws$ pixels are removed. This is done for $p-1$ pixels, therefore, the complexity of step 1 is $\mathcal{O}(p + p \cdot cws) = \mathcal{O}(p \cdot cws)$.

The complexity of step 2 is $\mathcal{O}(p)$ as for each pixel, a single condition is evaluated.

The complexity of step 3 is at most $\mathcal{O}(p)$; even if the matrix $mask$ is not initialized to zeros, it may be done so in one pass through it.

The complexity of step 4 is $\mathcal{O}(p)$, because each pixel is enqueued at most once in the process of running waves from $lightPoints$. Dealing with a single pixel (setting $mask(pixel)$, tests for stopping the wave, enqueuing neighbors) is obviously a constant operation.

Therefore, the complexity of the algorithm is dominated by the first step, which constitutes the resulting complexity $\mathcal{O}(p \cdot cws)$. □

#### 2.4.1.4 Defusion algorithms in TPP

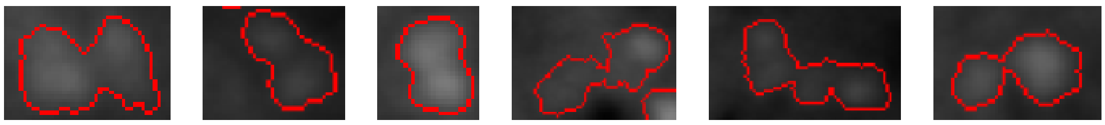


**Figure 2.3:** Several examples of fused neuronal somata.

The SeNeCA segmentation algorithm sometimes considers adjacent neuronal somata to be a single object. This problem, which is common to probably all segmentation algorithms, is called *undersegmentation*. Looking at the Fig. 2.3, we can see that neuronal fusions are generally characterized by a bottleneck between the two neurons and a decreased intensity between the neurons. However, there are cases when one of these conditions (or neither of them) is true, such cases are extremely difficult to segment correctly (e.g., the third fusion from the left in 2.3). Below, we describe two algorithms we had designed as a countermeasure against undersegmentation. Type 1 algorithm splits fusions according to their shape, while Type 2 algorithm splits fusions according to their intensity.

##### 2.4.1.4.1 Type 1 defusion

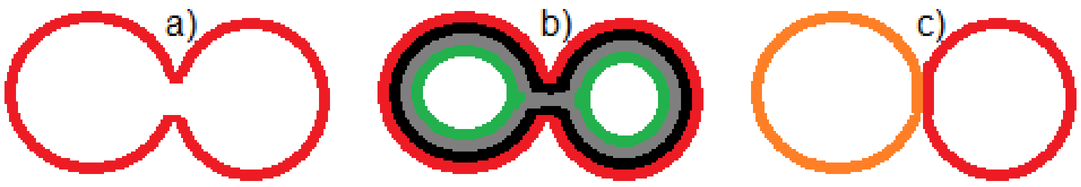


**Figure 2.4:** a) is a border of fused neuronal somata.
b) contains an example cushioning. The first two layers of cushion (black and gray) were 4-connected, but the green one is not.
c) shows the resulting split of objects. The objects are displayed in different colors as they will have different numbers in the resulting mask after defusion.

The basic idea is to take each object in *mask* provided by SeNeCA and cushion its inside, layer after layer from the border. Once an added layer is not connected, it means that there is a bottleneck in the point of disconnection and the object should be split at that place. The idea is illustrated in Fig. 2.4.

```
Type 1 defusion
Inputs: $mask, strength$ {$mask$ has been obtained by SeNeCA,
$strength$ should lie between 0 and 0.5. }
```

```
Outputs: $defusedMask$
Algorithm:

  1. Set $numberNeurons$ to the maximum element in $mask$. Set
     $nextNeuronNumber$ to be $numberNeurons+1$. Initialize empty
     list $candidates$. Add $1, 2, ..., numberNeurons$ to $candidates$.
     Initialize $defusedMask$ to be a zero matrix of the size of
     $mask$.

  2. For each $neuron$ in $candidates$:

     (a) Remove $neuron$ from $candidates$.
     (b) Set $borders, insides$ to contain border points and inside
         points of $neuron$ in $mask$.
     (c) Set $subMask$ to be the smallest submatrix of $mask$
         containing all of the pixels belonging to $neuron$. Let
         $xSize, ySize$ be dimensions of $subMask$.
     (d) Set $minDiameter$ to the minimum of $xSize$ and $ySize$.
         Compute $numberLayers$ as $minDiameter \cdot strength$.
     (e) for $i$=1 to $numberLayers$, add a single layer inside
         the previously laid layer (for $i = 1$, the previous
         layer is the border of the object). Fill hole (or
         holes) inside the newly created layer. If the pixels
         from filled hole (or holes) are not 4-connected,
         define $numComponents$ be the number of disconnected
         components and let $components$ be an array indexed by
         disconnected components, containing the list of pixels
         belonging to the given components; also break from the
         loop.
     (f) If $numComponents$ is not defined, copy $subMask$ into
         $defusedMask$ (at the place where $subMask$ was in
         $mask$). Go to next $neuron$ in the for loop.
     (g) Otherwise: {There is more than one connected
         component, the object can be split.}
          i. Assign numbers 1,2,...,$numComponents$ to the found
             connected components.
         ii. Run ``turn-based'' waves from the connected
             components in $components$. I.e., add a layer around
             the first component, then add a layer around the
             second component, etc. Then go back to the first
             component again. The waves are colored, i.e., it is
             known which pixel was captured by which wave. The
             waves may stop at a point under two conditions:
               • The point is at a border of the original object
                 (before defusion) is hit.
               • Another wave's point is adjacent to the current
               one.
             When all the waves stop, go to the next step.
        iii. Save the object demarcated by the wave of connected
             component number 1 into the appropriate place in
             $defusedMask$, with the number of the original
             object before splitting (saving positive numbers
             at the border and negative numbers inside). Save
             the other objects into $defusedMask$ with numbers
             $nextNeuronNumber$, $nextNeuronNumber + 1$, etc.
             Increment $nextNeuronNumber$ by $numComponents$-1.
         iv. Add all the objects added to $defusedMask$ in the
             previous step to $candidates$ {Splitting of new
             objects will be attempted again.}

  3. return $defusedMask$.
```

**Pseudocode 2.3:** The pseudocode of Type 1 (shape-based) algorithm for splitting of fused neuronal somata.

Let us explain the nontrivial steps in the algorithm. In step 2, the list *candidates* is basically a queue of objects to be split. In the following steps, an object is taken from it: if it can not be split, it is removed; if it can, it is split and the newly created objects are returned to the queue. This is motivated by the observation, that a fusion of at least three neurons may not be splittable by a “cushion” of constant thickness. Therefore, it is first split into two by a thinner cushioning and the component which may be further split will be split in the next pass through *candidates*.

In step 2b, one could propose to determine the borders and insides of an object in a one-way pass through the original mask and then update it when new objects are being added to $defusedMask$, rather than look for the borders and insides of an object again with every *neuron* taken from *candidates*. While this is certainly an improvement in theory, due to relatively slow loops in Matlab and, on the other hand, very fast `find` in-built function, the one-way pass finding of objects with future updates tends to be slightly slower or only marginally faster for common number (30-50) of neurons in an image. If large images with many objects will be processed with TPP in future, adding the variant of one-way pass detection of borders and insides may prove useful.

In step 2c, it would be possible to work with the entire mask, instead of *subMask*, but due to the speed of memory access, we found the solution with working on a submask more efficient.

In step 2d, the number of layers (“thickness of cushioning”) is determined. It is not always desirable to have the maximum possible number of layers, as it may lead to oversegmentation.

In step 2e, the object's inside is gradually cushioned and it is checked whether the inside space of the cushioned object is connected or not. If not, the object will be split into two (or more, if the cushioning creates more than two connected components).

However, it must be determined where to cut the original object and how. We decided to use the "turn-based waves approach" in step 2g, which fairly segments the original object into subobjects. One could argue that most of the work done by the waves only reverses the process of cushioning. While it is true, it is not trivial to design and implement a geometrical solution to where to cut the original object. Furthermore, we believe that this sort of cutting the object is fast enough[7] and leads to good segmentation.

#### 2.4.1.4.2 Type 2 defusion

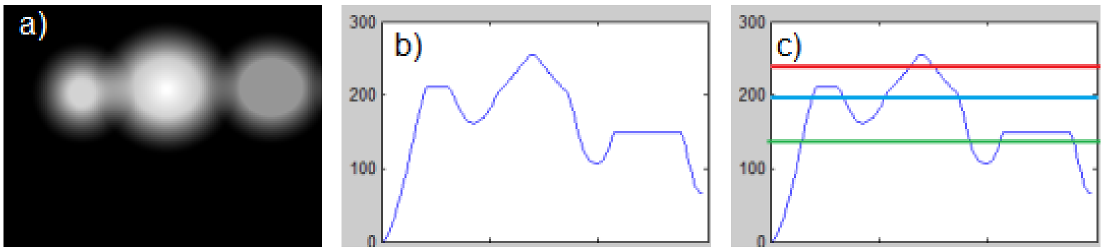


**Figure 2.5:** a) contains a simulated image of fused neurons
b) is a side-view at the scenery given by the fusion in the image in a).
c) shows three possible cuts. The red cut is too high as the area above the cut is obviously connected. However, when one of the other (green or blue) cuts is used, the area above the cut will be split into two. Note that in this case, it is not possible to find a single horizontal cut which would split the object into three disconnected components directly.

Again, each object from *mask* is taken and splitting is attempted. The area containing the object is considered to be a scenery, where light points represent peaks and darker points represent valleys. It is then attempted to cut the scenery by a horizontal cut, starting at the highest point of the scenery. If the scenery above the cut is connected, a lower cut is attempted, etc. Once the scenery above the cut is not connected, the object will be split at the point of disconnection. An illustration of what happens is in Fig. 2.5

The pseudocode of this type of defusion is below. The "outer shell" of the algorithm is the same as of Type 1 defusion, however, we list even the steps taken there for improved readability, rather than just reference them. Note though that only steps 2c, 2d and 2e are different.

```
Type 2 defusion
Inputs: $image, mask, strength$ {$image$ is the image on which SeNeCA ran, $mask$ has been obtained by SeNeCA, $precision$ should lie between 0.01 and 0.2. }
Outputs: $defusedMask$
Algorithm:
```

[7]0.1-0.2s per image with 30-50 neurons. Measured on Intel i7 2.66Ghz processor with 6MB cache.

```
1. Set $numberNeurons$ to the maximum element in $mask$. Set
   $nextNeuronNumber$ to be $numberNeurons$+1. Initialize empty
   list $candidates$. Add $1, 2, ..., numberNeurons$ to $candidates$.
   Initialize $defusedMask$ to be a zero matrix of the size of
   $mask$.

2. For each $neuron$ in $candidates$:

   (a) Remove $neuron$ from $candidates$.
   (b) Set $borders, insides$ to contain border points and inside
       points of $neuron$ in $mask$.
   (c) Set $subImage$ to be the smallest subimage of $image$
       containing all of the pixels belonging to $neuron$.
   (d) for $i$=1 downto 0 with step $precision$: Cut the scenery
       in subImage in height $i * max(subImage)$. If the
       pixels above the cut are not 4-connected, define
       $numComponents$ be the number of disconnected
       components and let $components$ be an array indexed by
       disconnected components, containing the list of pixels
       belonging to the given components.
   (e) Take the cut from previous step which divides the area
       above itself into the most disconnected parts; recall
       its $numComponents$ and $components$.
   (f) If $numComponents$ is not defined, copy the part of
       $mask$ containing $neuron$ into $defusedMask$. Go to next
       $neuron$ in the for loop.
   (g) Otherwise: {There is more than one connected
       component, the object can be split.}
        i. Assign numbers 1,2,...,$numComponents$ to the found
           connected components.
       ii. Run ``turn-based'' waves from the connected
           components in $components$. I.e., add a layer around
           the first component, then add a layer around the
           second component, etc. Then go back to the first
           component again. The waves are colored, i.e., it is
           known which pixel was captured by which wave. The
           waves may stop at a point under two conditions:
             • The point is at a border of the original object
               (before defusion) is hit.
             • Another wave's point is adjacent to the current
               one.
           When all the waves stop, go to the next step.
      iii. Save the object demarcated by the wave of connected
           component number 1 into the appropriate place in
```

```
            $defusedMask$, with the number of the original
            object before splitting (saving positive numbers
            at the border and negative numbers inside). Save
            the other objects into $defusedMask$ with numbers
            $nextNeuronNumber$, $nextNeuronNumber + 1$, etc.
            Increment $nextNeuronNumber$ by $numComponents$-1.
        iv. Add all the objects added to $defusedMask$ in the
            previous step to $candidates$ {Splitting of new
            objects will be attempted again.}

3. return $defusedMask$.
```

**Pseudocode 2.4:** The pseudocode of Type 2 (intensity-based) algorithm for splitting of fused neuronal bodies.

Let us discuss the step 2e, which is the major difference of Type 2 defusion, compared to Type 1 defusion. A cut at gradually decreasing elevation is attempted. The parameter *precision* specifies how fine changes are between cuts. The smaller *precision* is, the more equidistant cuts will be attempted. The cutting of scenery into most disconnected components is then chosen. Note that this step heavily relies on *image* being already somewhat denoised or smoothed. With noise, Type 2 defusion would lead to severe oversegmentation. When a cut is chosen, the scenery above the cut is used as the centers of future "turn-based waves" with the same semantics as the disconnected components inside cushioning in Type 1 defusion had.

Unfortunately, it is not always sufficient to pick a single best cut for correct segmentation, as demonstrated by Fig. 2.5. For this reason, when an object is cut by Type 2 defusion, the resulting objects are returned to the queue for further cutting attempts.

### 2.4.2 Planning line-scan path

Commonly, the scan path is drawn by hand. However, this has two drawbacks. First, it is rather subjective and may lead to a strong bias. Second, the drawn path may be longer than necessary.

We aimed for an automated solution which would automatically find neurons, find the shortest Hamiltonian cycle between them and save the path into a file which could be loaded by the microscope. After the neurons are found (using SeNeCA described in the previous section), the task is close to a Travelling Salesman Problem (TSP) with Euclidean distance as the weight function of edges between vertices (neurons).

Two more requirements should be taken into account when designing an automated solution. First, the scanning laser should never pass through a single neuron more than once in a single cycle. If such a case happened, the neuron would unnecessarily suffer from photobleaching more than other neurons. The second requirement concerns the shape of the scanning path. Certain scanning

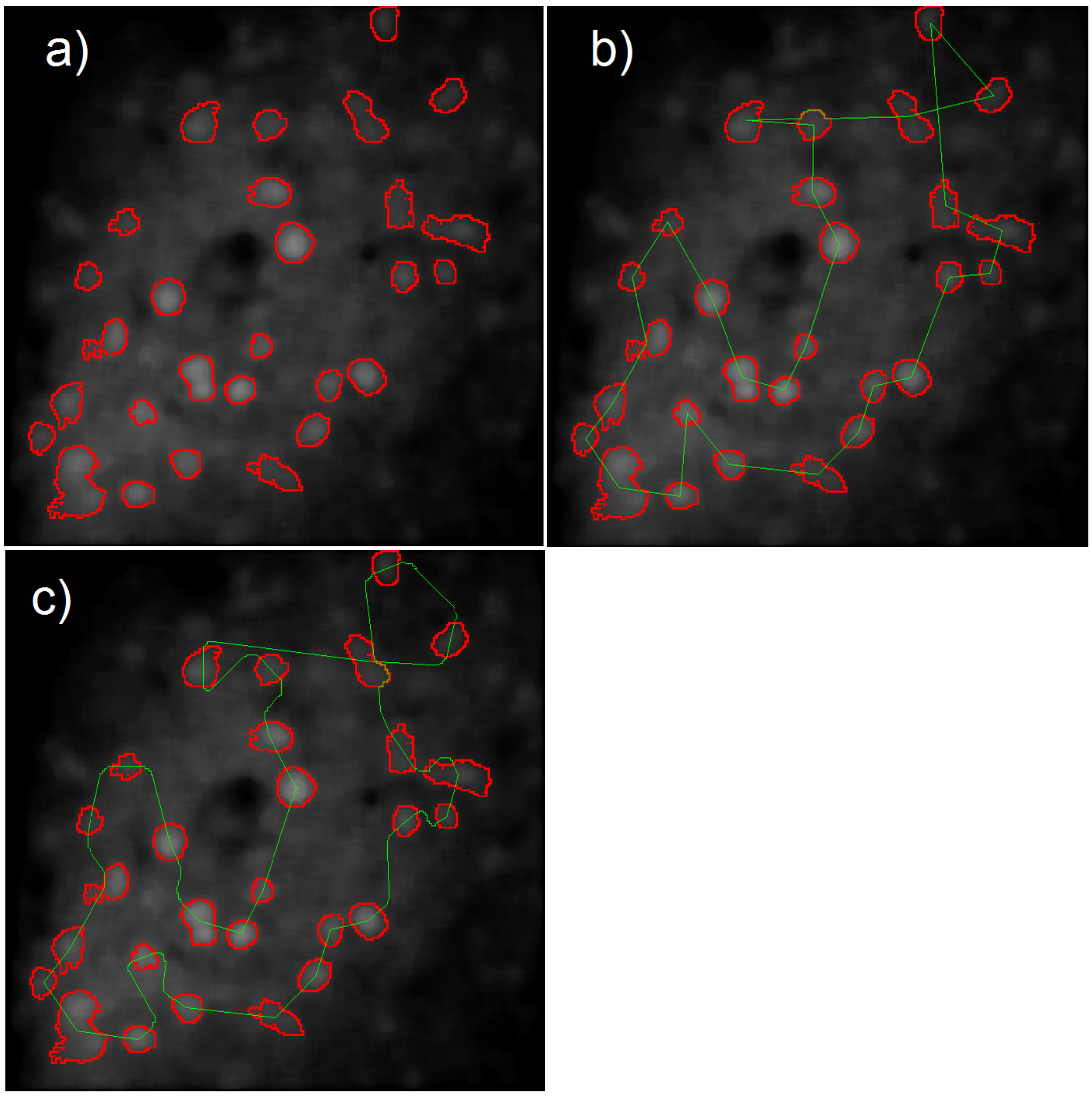


**Figure 2.6:** a) A reference image segmented by SeNeCA with fusions split by Type 1 defusion.
b) A path planned in the reference image, using the algorithm for not entering a single neuron more than once in one cycle. When such a situation should happen, the neuron is circumvented.
c) Another planned path, with the algorithm for reduction of sharp angles used.

galvanometers may have trouble coping with abrupt changes of direction in the scan path (e.g., following angles sharper than 90°). Therefore, the automated solution should have an option to smooth the planned path. Our solution described below fulfills both requirements, see Fig. 2.6 for an example of possible planned path.

To our knowledge, the only tool seriously studying the problem of optimal laser scanning path is HOPS (Sadovsky et al., 2011)[8]. Unfortunately, it does not deal with the first of our requirements. Even though HOPS is open-source,

[8]Some earlier works on the topic of optimizing scanning path may be found in this article.

we came to the conclusion that it will be faster to implement our own solution fulfilling all the requirements, rather than modify the HOPS code and connect it to Matlab.

#### 2.4.2.1 The processing pipeline

```
1. Load a reference full-frame image of the tissue to be
   recorded.

2. Choose the method and parameters of denoising (disk,square
   or rectangle blur, or median filtering) and denoise the
   reference image.

3. Set the parameters for segmentation by SeNeCA and segment
   the image.

4. Find the shortest Hamiltonian cycle in the graph where
   vertices are centroids of objects found in the previous
   step and edges are lines connecting these vertices. Weight
   of the edges is the Euclidean distance between the vertices
   demarcating the edge. The used algorithm is ANT colony
   simulation.

5. (optional) Smooth the path, attempting to remove sharp
   angles of the path.

6. Change the planned path so that no neuron is scanned by the
   scanning laser beam more than once during a single cycle.
```

**Pseudocode 2.5:** The process of of obtaining a path for the scanning beam of a 2P microscope.

#### 2.4.2.2 Discussion of the processing pipeline

The steps 2 and 3 are essentially the same as in processing full-frame data, except only the segmentation part is used here, no intensities are measured.

In step 4, we decided to use ANT colony optimization algorithm, as it was already implemented for Matlab and available from FileExchange[9]. In a preliminary version of the algorithm, we used a hungry algorithm for TSP, then switched to the well known approximation algorithm for solving TSP, but the ANT colony optimization proved to perform better than both of these.

In step 5, which is optional, it is possible to attempt removing sharp angles in the scanning path returned by ANT in the previous step. The galvos deflecting

[9]The function is a part of code of the class External which includes all the code from other Matlab users.

the scanning laser may have trouble with abrupt changes in direction of scanning, which may lead to prolonged dwell times at the places of turn, etc.

In step 6, we make sure that no neuron is visited twice in one cycle. We did not find an already published solution to this problem. A possible solution would be to generate various near-optimal Hamiltonian cycles, waiting for one without any conflicts (a neuron visited twice in a single cycle). However, there are cases when a Hamiltonian cycle without conflict does not exist, therefore we rejected this idea. We designed another algorithm which takes the path from step 4 or 5 and creates detours in places of conflicts. The algorithm is described in 2.4.2.3.

#### 2.4.2.3 Circumvention algorithm

A scan path is defined by *checkpoints* - pixels which, when connected by lines (called *segments*), form the resulting scan path. In case of path from step 4 of the Processing pipeline in 2.4.2.1, *checkpoints* are directly the centroids of neurons to be visited (and segments are the edges between them). If the scan path is smoothed in step 5, the set of *checkpoints* from step 4 is refined to include curves at the places of originally sharp angles.

Even though the implementation of the algorithm is rather long, the basic idea is very simple, described in pseudocode below.

```
Circumvention
Inputs: $path$ {A set of 2D coordinates of $checkpoints$. }
Outputs: $circumventedPath$
Algorithm:
  1. Initialize $circumventedPath$ to empty structure. item For
     each $segment$=($startPoint$, $endPoint$) in $path$:
      (a) Determine $starting$ and $ending$ - the numbers of
          neurons from which and to which the segment leads,
          respectively.
      (b) Add $startPoint$ to $circumventedPath$.
      (c) Iterate over $line$ determined by $segment$, a line
          connecting $startPoint$ and $endPoint$. If a neuron with
          number other than $starting$ or $ending$ is encountered:
            i. Let $x$ be the number of the encountered neuron. Let
               $entryPoint$ be the coordinate when $line$ crossed the
               border of neuron $x$.
           ii. Add $entryPoint$ to $circumventedPath$.
          iii. Divide the border points of neuron $x$ into two
               groups, divided by $line$. Let $shorterDetour$ be the
               smaller set of the two. If the sets have the same
               size, any of them may be taken.
           iv. Add all the points in $shorterDetour$ to
               $circumventedPath$ in correct order (i.e., the member
               of $shorterDetour$ closest to $entryPoint$ is added
               first, etc.).
            v. Set $startPoint$ to be the last point from
               $shorterDetour$ added to $circumventedPath$ and resume
               iterating over $line$ from there.
```

**Pseudocode 2.6:** The pseudocode of circumvention algorithm, which makes sure that no single neuron is visited twice in one scan cycle of a line-scan.

The main idea of the algorithm is the following: When a neuron is visited, while it should not be visited (it is not the starting, nor ending neuron of the given *segment*), divide its border according to *line* cutting the neuron and use the smaller of the groups of pixels as the path for scanner. Therefore, a neuron is bypassed along its border. This could sound like exactly the opposite of what we want to achieve: bypass a neuron. It is vital to realize that when a laser hits a border of a neuron, almost no dye will be hit and no additional photobleaching and/or fluorescence should happen. If problems occur nevertheless (e.g., the tissue moves a little, the detour entering neuron), it is always possible to use *diskblur* with larger parameter as the denoising step before segmentation of the image on which the path is planned. Using such a filtering leads to optical enlargement of neurons. Another option leading to similar result is using lower value of $lowLightThreshold$.

### 2.4.3 Extracting calcium traces from line-scan data

The line-scan data may seem unusual at the first sight and they are further from real looks of brain than full-frame data. However working with line-scan data may be actually easier and faster than working with full-frame data, as there is one dimension less (full-frame recording consists of 2D images over time; line-scan recording consists of 1D images over time). While there are software toolkits for automating various tasks performed with full-frame data, there is no publicly available tool for processing line-scan data to our knowledge. Therefore, we designed and implemented our own processing pipeline for extracting calcium traces of neurons from line-scan data. It is built on similar principles as the extraction of calcium traces from full-frame data, yet it contains several crucial differences. We will discuss these differences in 2.4.3.2

#### 2.4.3.1 Processing pipeline

```
1. Load a line-scan image representing the recording of brain
   over time into memory.

2. Choose the method and parameters of denoising (disk,square
   or rectangle blur, or median filtering).

3. Set the parameters for segmentation and tracking by
   Linescan-SeNeCA and segment the data.
```

```
4. Improve the segmentation and tracking from previous step by
   post-processing: *joining*, *sorting* or *interpolating* the output
   of Linescan-SeNeCA. Also, borders of neurons over time may
   be smoothed.

5. Return the intensities of neuronal objects over time.
```

**Pseudocode 2.7:** The process of obtaining calcium traces from line-scan data.

#### 2.4.3.2 Discussion of processing pipeline

The steps 1 and 2 are very similar to those in processing of full-frame data. However, instead of a sequence of images, only one source image is used[10].

In the step 3, the recording is segmented. However, unlike in full-frame data, a neuron is not anymore represented by a circular object, but by a light column instead. For the segmentation, we used a modified version of SeNeCA, named Linescan-SeNeCA, which, as a side effect, tracks the neurons implicitly. An advantage of this approach is that the set of parameters used for segmentation is a subset of parameters used in segmentation of full-frame data, therefore the user of the software needs to understand only one semantics of parameters for working both with line-scan and full-frame data. Linescan-SeNeCA is described in 2.4.3.3

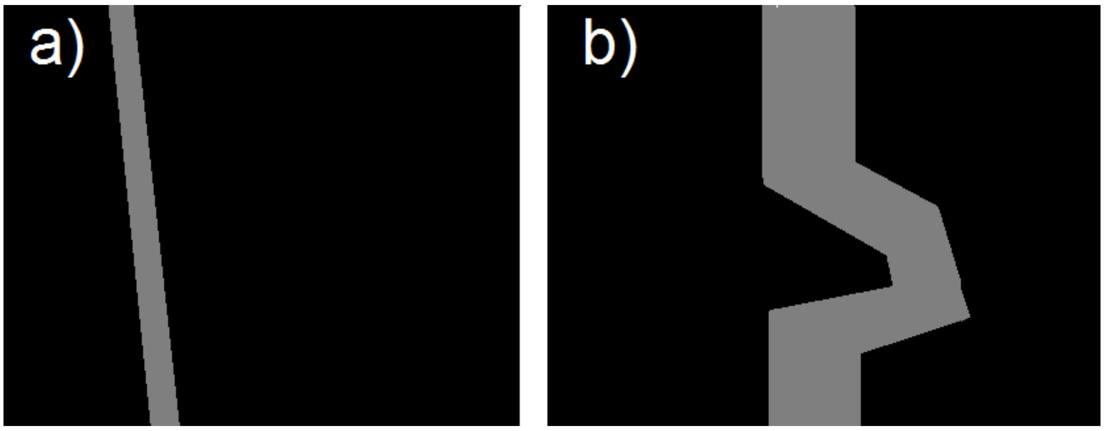


**Figure 2.7:** a) A general shift in neuron's position over time, caused by slow movement of the observed tissue.
b) A brief movement of a neuron, such as caused by a hiccup.

There is a simple alternative approach, utilizing the fact, that neurons resemble light columns. A single sliding column of the image could be taken and its average intensity measured, declaring the columns sufficiently lighter than the average of the given area (using a 1D analogy of *lightAverages* from processing of full-frame data) to be neurons. Unfortunately, this approach would work only

[10]In reality, the Prairie Ultima IV microscope splits the recorded image into images containing 1000 lines, but we join these to produce a single 2D image.

in case of stationary tissue. With in-plane motion artifacts, this could lead to a rather bad segmentation, an example of such problems is in Fig. 2.7, none of which can be properly segmented using a vertical column-like object. Both problem could be seemingly solved by using a wider column which would contain the whole neuron over time, along its in-plane artifacts, however, such an approach would capture too much noise and activity of neuropil, similar to the situation with using larger-than-necessary ROI in 1.2.2.

The step 4 takes care of various improvements to the segmentation and tracking done by Linescan-SeNeCA, which fails to precisely track the badly stained (or out of focal plane) neurons. There are three incremental improvements (the user may choose which one to use) to the tracking and we use an artificial image in Fig. 2.8a to illustrate them.

1. *Joining*: When a neuron temporarily becomes too dark (e.g., it leaves the focal plane), Linescan-SeNeCA will not detect it at the dark stage and when the neuron is rediscovered, it will be assigned a new number. Joining of mask prevents this from happening and it joins neurons “under” one another (their maximum distance in the x-axis is a parameter) into one, as demonstrated in Fig. 2.8d. Aside from joining the neurons, it is possible for the user to state a minimum activity length of neuron in this stage. The neurons which are active less than the given percentage of the measurement will be discarded. This is very helpful in getting rid of small non-neuronal objects and/or noise, which appear only for a short period.

2. *Sorting*: This is a rather technical improvement. It reassigns numbers to found neurons in such a way, that the leftmost neuron (according to its centroid) gets number 1, the second leftmost gets 2, etc. This step serves only to make observing the data more intuitive and comprehensible. The effect of *sorting* is displayed in Fig. 2.8e.

3. *Interpolation*: Even though different parts of neuron may be associated with one another in the process of *joining*, the information between different segments of a neuron is not measured. In the process of *interpolation*, borders of a neuron in time are interpolated so that the neuron will be measured throughout the whole experiment. The process of *interpolation* is depicted in Fig. 2.8f.

Another improvement of the segmentation we implemented is smoothing of neuronal borders, which may be somewhat serrated when too much noise is left in the image. The amount of such smoothing is given by a parameter specified by the user.

In the step 5, the intensities (average and sum, as in full-frame data) are recorded from the found objects, line by line. The format of the outputted calcium traces is the same as from the full-frame recording.

#### 2.4.3.3 Linescan-SeNeCA

The principle of Linescan-SeNeCA is fundamentally similar to the concept of SeNeCA. The core of LineScan-SeNeCA is essentially the same as SeNeCA in 1D instead of 2D. The processing of second dimension of line-scan data (time)

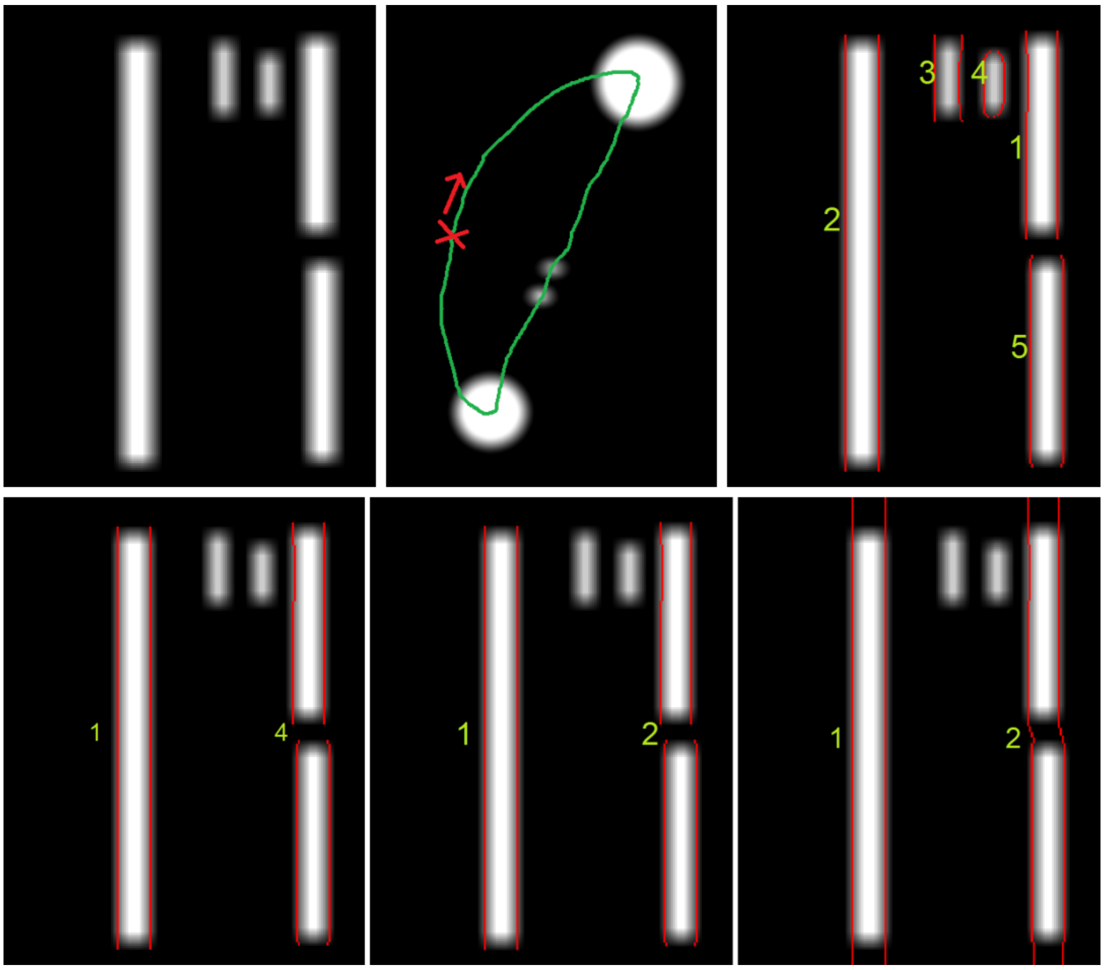


**Figure 2.8:** a) An artificial line-scan measurement. The image emulates one well stained neuron (the leftmost white column), two small non-neuronal objects (e.g., visible crossing of axons) and one neuron which is rather well stained, but has left the focal plane for a while (e.g., due to a deformation of that part of tissue) and moved a bit to the side when returning
b) An artificial full-frame image from which the data were recorded (the full-frame image comes from the time period when even the two small objects were observed), along with a hand-drawn scanning path. The red cross and arrow mark the start of the path and the direction of scanning respectively.
c) A "raw" mask of neuronal borders over time, as produced by Linescan-SeNeCA.
d) A *joined* mask, built upon the previous mask. First, the two parts of the rightmost object were joined into one (because they were close enough). Second, as only the objects active shorter than 50% time were rejected, the two small objects are not segmented anymore.
e) A *sorted* mask with continuous numbering of objects.
f) An *interpolated* mask of the image. While the interpolation may not seem very useful in such a simple and artificial example, it is actually very helpful in real data.

works line by line and is different from how the second dimension is treated in the original SeNeCA algorithm.

```
Linescan-SeNeCA
Inputs: $image, highLightThreshold$, $lowLightThreshold$,
$contrastWindowSize$, $minimumLight$
Outputs: $mask$

Algorithm:

  1. initialize $mask$ to be a zero mask of the same size as
     $image$. Initialize $neuronNumber$ to 1.

  2. Compute matrix $lightAverages$ of the same size as $image$,
     defined as $image$ filtered by a mean averaging filter with
     height 1 and width $contrastWindowSize$. A mirror projection
     back to the image is used when a pixel outside the image
     should be taken into account in the computation.

  3. Compute $lightPoints$ as the set of all points $(i, j)$ with the
     following property: $(image(i, j) \cdot highLightThreshold >$
     $lightAverages(i, j))$ && $(image(i, j) > minimumLight)$

  4. for each $(xCor, yCor)$ in $lightPoints$:

     (a) If $(mask(xCor, yCor)! = 0)$, continue with the next point
         in $lightPoints$ immediately.
     (b) Otherwise set $mask(xCor, yCor)$ to $-neuronNumber$
         and run a 1D wave in the line given by $xCor$
         from $(xCor, yCor)$, setting the visited pixels to
         $-neuronNumber$. The wave may stop at a point $(xCor, b)$
         for three reasons:

           • The following condition holds: $(image(xCor, b) \cdot$
             $lowLightThreshold < lightAverages(xCor, b))$
           • $(xCor, b, )$ is at the edge of the image
           • $mask(xCor, b)$ contains a positive value, other than
             $numberNeuron$ to the left or to the right.

         When the wave stops at the point $(a, b)$, set $mask(a, b)$
         to $neuronNumber$.
     (c) After the wave has finished running, find the centroid
         of the currently found object (at the current line),
         go one line up from that centroid and continue running
         the wave there, etc. Stop when the pixel at the
         above line is either outside the image, or is too
         dark $((image(oneUpX, oneUpY) \cdot lowLightThreshold <$
         $lightAverages(oneUpX, oneUpY))$ ).
     (d) After the process of going to upper lines has stopped,
         return back to the line where the original wave
         started running from $(xCor, yCor)$ and repeat the step
         above, except going down, instead of going up.
```

```
    (e) After the process of going down has stopped, increment
        $numberNeuron$ and go to the next of $lightPoints$.

5. return $mask$.
```

**Pseudocode 2.8:** The pseudocode of Linescan-SeNeCA algorithm for segmentation of line-scan data.

Until when the waves from *lightPoints* start running, the algorithm above is almost the same as SeNeCA, with the exception, that where there was square filter used for obtaining *lightAverages*, only a single-line filter is used now. The reason is that *lightAverages* should contain only spatial, not temporal information.

In step 4b, a 1D wave is used to find borders of the neuron (which is a 1D object in line-scan, if temporal axis is not considered). Other than its dimensionality, the wave behaves exactly the same as the waves used in original SeNeCA.

In steps 4c and 4d, it is attempted to extend the found object across time, i.e., to track it. There are two reasons for this step: First, it provides a tracking algorithm, which would be necessary anyway. Second, because when extending the object across time, *lowLightThreshold* is used instead of *highLightThreshold*, it is possible to find and track even neurons which lack sufficient contrast for most of the time of the experiment[11]. A typical scenario when the second reason is very advantageous is when there is a neuron which has enough contrast for some of its points being added to *lightPoints*, but only when it spikes; no points from its non-spiking periods are contrasting enough to make it there. However, our algorithm starts with the points from spikes of the neuron when the contrast is sufficient and then extend the found object across time, using less demanding threshold, therefore segmenting even the non-spiking periods of the neuron.

### 2.4.4 Noise removal and spike mining from calcium traces

Once the calcium traces of neurons have been obtained (see Fig. 2.9a) from full-frame or line-scan data[12], it is important to infer spike trains produced by the neurons, because information in brain is primarily encoded by action potentials, rather than by calcium traces. We included two algorithms solving this problem, an example of the output of one of these algorithms is shown in Fig. 2.9b. TPP also contains functionality, described in 2.4.4.3, which allows more advanced analysis of neuronal features.

[11]This does not mean necessarily a badly stained neuron. Active areas containing a lot of fluorescent neuropil are another example.

[12]Actually, it is possible to use any sort of data. As long as the traces of neuronal activity are in a correct format, any sort of recording may be processed by this tool. This makes it useful even for analysis of, e.g., data from single-unit electrophysiology.

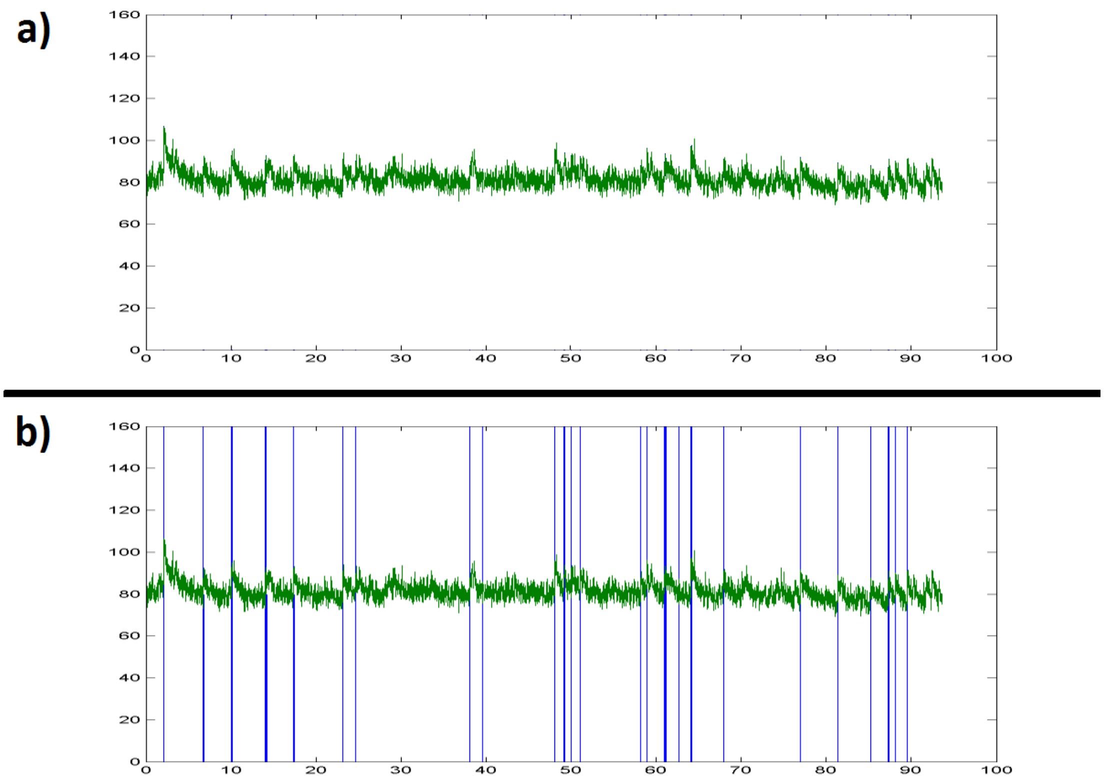


**Figure 2.9:** a) A calcium trace of a neuron.
b) A calcium trace of a neuron with marked occurrences of spikes, according to a spike inferring algorithm.

#### 2.4.4.1 Processing pipeline

```
1. Load a file with selected calcium traces of neurons.

2. Choose the method and parameters of denoising the trace
   (FFT or FIR filtration).

3. Select an algorithm for inference of spike trains and
   run it. More repeats of an experiment may be processed
   together.

4. (optional) Load data describing a specific experiment
   and make the software record and save post-stimulus time
   histograms (PSTH) and receptive fields of neurons.
```

**Pseudocode 2.9:** The process of extraction of spike trains from calcium traces of neurons.

### 2.4.4.2 Discussion of the processing pipeline

In step 2, we included two algorithms for filtering the calcium traces loaded in step 1. There are certain frequencies known to occur in calcium traces, which should be removed, e.g., the pulse introduced by heartbeat. We included FFT filtration in the TPP toolkit as it is easy to understand and often used form of filtration. The second algorithm, FIR filtering, is also frequently used in literature (e.g., (Bandyopadhyay et al., 2010)), which is why we also included it in TPP.

In step 3, two algorithms may be used for inference of spike trains from calcium traces: peeling algorithm (Grewe et al., 2010) and fast non-negative deconvolution (Vogelstein et al., 2009).

In step 4, it is possible to load additional information about an *auditory experiment* and compute more neuronal properties. Both are described in the next section, 2.4.4.3.

### 2.4.4.3 Advanced analysis of experiments

One of the possible designs of an auditory experiment is that the auditory stimuli are presented to the experimental animal while the neurons in its auditory cortex are being imaged. If the order of stimuli and their lengths are saved, it is possible to recall this information via this functionality of TPP and associate it to the reactions of the neurons.

Currently, there are two major types of characteristics that can be obtained and plotted for each neuron.:

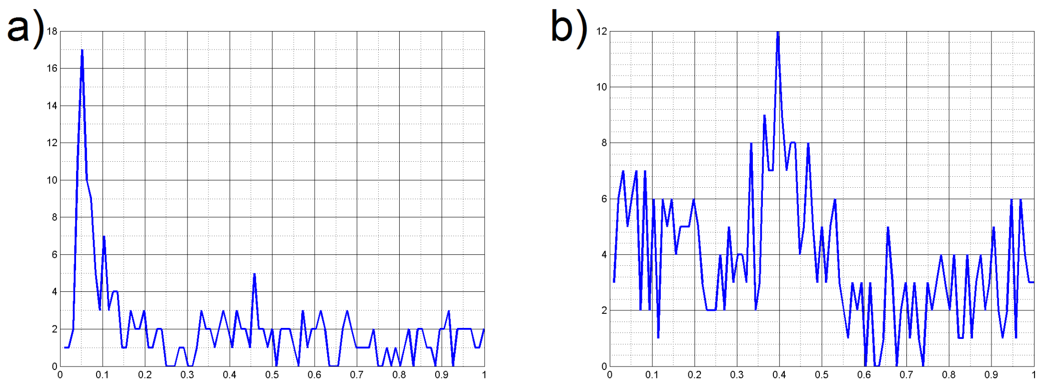


**Figure 2.10:** a ) A post-stimulus time histogram of an onset neuron. b) A post-stimulus time histogram of a rebound neuron.

- Post-stimulus time histogram (PSTH): This is the summed activity of a neuron across repeats of an experiment. It can be used to study, e.g., whether a neuron is an *onset* type (reacting shortly after a stimulus onset), see Fig. 2.10a, an offset type (reacting shortly after the end of the stimulus), or a *rebound* type (reacting longer after the end of the stimulus), see Fig. 2.10b.

- Receptive fields: A battery of sounds with different frequencies and amplitudes may be presented to a mouse several times Then, after associating the inputs with the neuronal responses, a map of neuron's reactions may

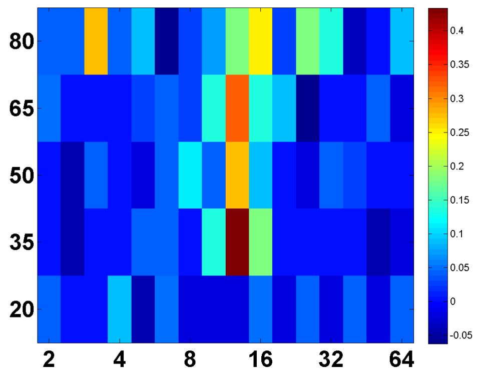


**Figure 2.11:** A receptive field of a neuron of I type (sharply tuned to a single frequency). The warmer colors signal stronger reaction of the neuron to a stimulus with given amplitude and frequency. The displayed neuron tends to be tuned to sounds with frequency around 12000 Hz.

be drawn, see Fig. 2.11. This makes it possible to directly visualize the selectivity of neurons to the given set of stimuli.

A future improvement of TPP worth pursuing would be an inclusion of a tool for automated analysis of measured properties of neurons. It could prove fruitful to perform, e.g., categorial data analysis of a type of neuron (e.g., excitatory/inhibitory or onset/offset) and its properties (e.g., how strictly tuned to its "favorite" frequency it is, whether it is a burster neuron, or a sparse firing one, etc...), discovering currently unknown correlations of properties.

### 2.4.5 ZScan processing

This component plays only a support role in TPP and is very straightforward. Its main purpose is to simply visualize the observed tissue in 3D, facilitate observation of the morphology and concentration of neurons, etc. A stack of images recorded from various depths of brain is the input of this tool, the voxel representing the tissue in 3D is created in the process.

#### 2.4.5.1 Processing pipeline

```
1. Load z-series stack of images, automatically joining
   different color channels.

2. Align the image from the stack.

3. Join the stack into a single voxel, providing correct
   spacing between stack slices.

4. Save, load or view the voxel.
```

**Pseudocode 2.10:** The processing of z-stacks in TPP.

#### 2.4.5.2 Discussion of the processing pipeline

In the step 2, it is necessary to compensate the wiggling of recorded images (caused, e.g., by breathing). We implemented a simple method of aligning images to one another to compensate for the movement artifacts. As the whole image moves due to the movement caused by zooming, we simply take images from depths $i$ and $i+1$, where the $i$-th image is already aligned, try shifting the $i+1$-th image by $\pm x$ pixels in both coordinates, computing the correlation to the $i$-th image. The $x$ is a parameter. The shift the with highest degree of correlation is then taken and aligned along the $i$-th image. Such a simple method works well for movement artifacts in which the entire image is shifted a few pixels to some side.

In the step 4, we included a simple voxel viewer, using which may a user navigate through the voxel and display various sections of the observed tissue. The viewer is shown in Fig. 2.12.

### 2.4.6 Extension 1: Realtime worker

To offer maximum computational speed of segmentation of full-frame data, we included a lightweight version of the processing pipeline from 2.4.1.1 as a runnable Matlab script. Although not as comfortable to use as the standard TPP GUI, it avoids the performance overhead of a running GUI, which improves the speed of computation.

```
1. Read images in a folder. For each image:
   (a) Denoise the image.
   (b) Segment the image using SeNeCA:
       •
       • Compute $lightAverages$ needed for SeNeCA using Matlab
       • Perform the rest of segmentation by C version of
         SeNeCA.
   (c) Remove objects too large or too small to be neurons
       (according to parameters specified by the user).
   (d) Using nearest neighbor tracking, assign found objects
       to the objects found previously (and register the
       newly appeared ones).
   (e) Measure intensities of found objects.
   (f) Report if there is a notable jump in intensity,
       compared to recent history. {A fast heuristics for
       spike detection. }
```

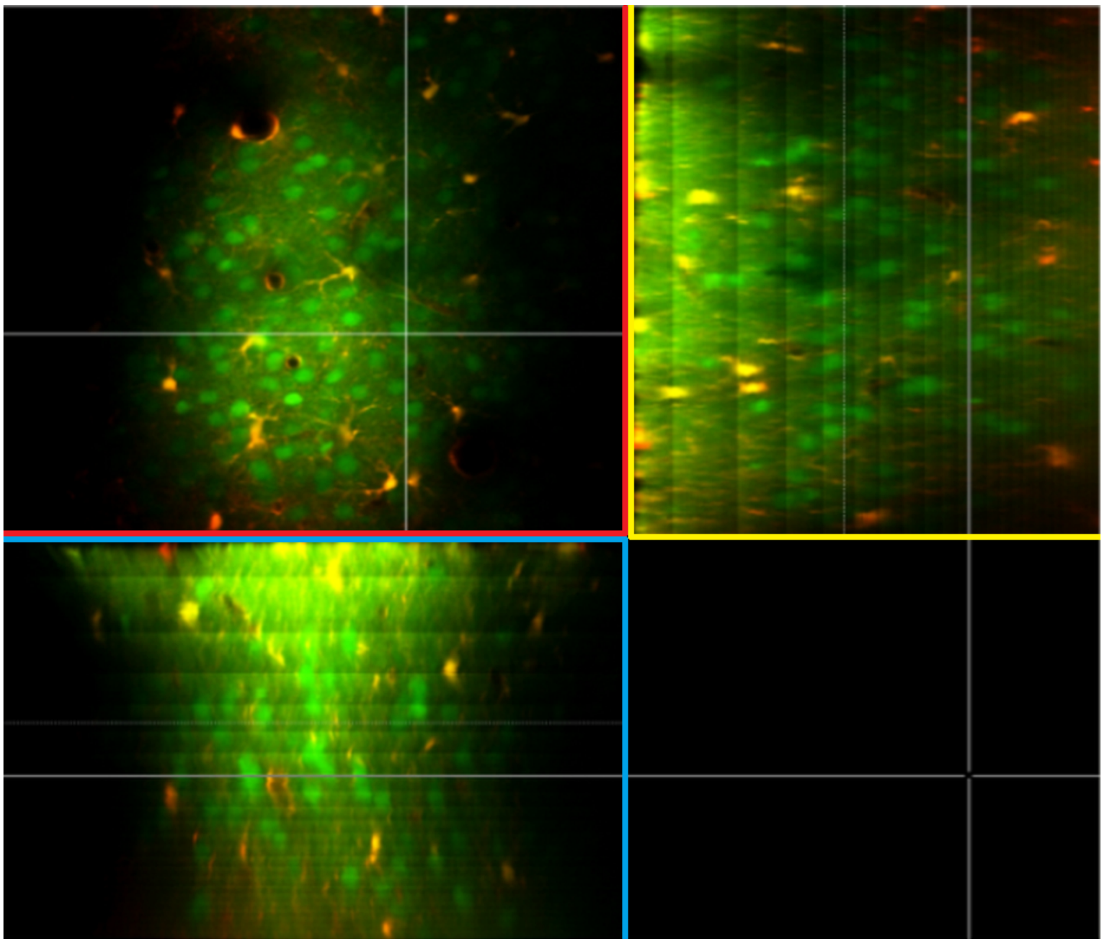


**Figure 2.12:** The voxel viewer included in TPP, showing a voxel of tissue. The section of the screen marked by red lines contains the view from above the brain, while the blue and yellow sections of screen display the sides of the voxel. The screen with the viewer is clickable and clicking on any of the three sections automatically updates the other two sections to match the selected place.

**Pseudocode 2.11:** An overview of the script RealtimeWorker used to record calcium traces of neurons in delayed real time.

Realtime monitoring of single-unit neuronal activity is one of long-term goals in calcium imaging (we discuss the use of such monitoring in 3.2.3 in greater depth). We measured how long would the process of monitoring neuronal activity, described in the pseudocode above take. Using our implementation of SeNeCA in Matlab, about 3-7 images[13] per second could been processed, which, although it allows some sort of near-realtime monitoring of neuronal activity, does not leave a large reserve for further computation done along the segmentation (e.g., a more sophisticated algorithm of spike detection). Profiling the processing of multiple images showed that about 95% time spent in the process is used by SeNeCA. For this reason, we implemented and optimized the part of SeNeCA

[13]With resolution 256x256, containing 30-50 neurons.

(after computation of *lightAverages*) in C and connected it to Matlab via MEX interface. The filtration of too small or too large objects has also been written in C. This Matlab-C hybrid approach led to a significant reduction of time spent on segmentation and it became possible to process more than 100 images per second using an ordinary PC. Computation of *lightAverages* becomes the main bottleneck then, taking circa 40-70% time of the processing, depending on the value of *contrastWindowSize*. We discuss the structure of time consumption of segmentation in 3.1.1.

While there are possible improvements to the current speed of the script Real-time worker, speeding up the largest bottleneck, computation of *lightAverages*, may not be particularly easy, see 3.1.1.

A possible future improvement would be designing an online algorithm for spike inference (or adapting one of the already existing algorithms). Due to fast segmentation and preprocessing, there is a time reserve for a further nontrivial algorithm.

As a side note, the "basic" TPP, controlled from the GUI still uses the Matlab version of segmentation. The reason is that we wanted TPP to be rapidly usable out of the box and making the C version of SeNeCA work may need recompiling the code.

# 3. Evaluation and discussion

This chapter consists of two independent sections. In the first section, we evaluate only the SeNeCA segmentation algorithm, especially its performance on several different datasets. The second section discusses other software tools useful for similar purposes as Two-Photon Processor (TPP).

## 3.1 Evaluation of SeNeCA

We start this section with analysis of time consumption of SeNeCA. Then we describe the results of three analyses which we performed on various datasets: *in vitro*, *in vivo* and *in silico* data. The section is concluded by a short summary of the whole evaluation.

All measurements were performed on an ordinary PC HP Elite 7300 (Intel Core i7 2600 3,4 GHz, 8 MB L3 Cache, 4 cores — 8 threads, 16 GB DDR3 1333 MHz, GPU NVIDIA GeForce GT545, Motherboard Intel H67 Express, HDD 1000 GB 7200 RPM Serial ATA II).

### 3.1.1 On time consumption of segmentation using SeNeCA

Because the speed of segmentation is one of the major advantages of SeNeCA, we describe the structure of the time consumption in greater detail. In this section, we inspect the time consumption of a common use of SeNeCA (256x256 image, ˜40 neurons), described in Pseudocode 3.1. We measured 100 repeats of the code and Matlab profiler was used to determine the time consumption of single steps. Therefore the performance is lower than in real use, due to a certain profiling overhead. Time consumption of SeNeCA on other datasets is described in 3.1.4.

```
1. Load an image with resolution 256x256 from HDD.

2. Denoise the image using a disk blur with parameter 4.

3. Segment the image using SeNeCA:

   (a) Compute $lightAverages$ with $contrastWindowSize$ 20.
   (b) Perform the remaining steps of SeNeCA.

4. Remove objects with diameter larger than 50 pixels or
   smaller than 8 pixels.
```

**Pseudocode 3.1:** The pseudocode of segmentation using SeNeCA.

Table 3.1 summarizes the time consumption of 100 repeats of the sequence of commands in Pseudocode 3.1, according to the language used for the task.

| Task | Matlab [s] | C [s] |
|---|---:|---:|
| Load image from HDD | $0.16 \pm 0.0067$ | - |
| Denoise image | $0.22 \pm 0.012$ | - |
| Segmentation: computation of *lightAverages* | $0.33 \pm 0.0466$ | - |
| Segmentation: remaining steps | $20.09 \pm 0.967$ | $0.16 \pm 0.0052$ |
| Filter objects by size | $1.72 \pm 0.12$ | <0.01 |

**Table 3.1:** The structure of time consumption of segmentation using SeNeCA, measuring the runtime of Matlab code and C code on 100 images. For the first three tasks, C code was not used and therefore it is not listed. The values are given ± standard deviation, measured on 5 repeats.

It is obvious that the hybrid approach to segmentation, using both Matlab and C brings a significant improvement in computational performance. Without the profiling overhead, it is possible to process about 200 images in a single second.

There are several approaches to making the hybrid Matlab-C segmentation even faster and we discuss them below:

- A trivial speedup is using a SSD HDD[1], which would significantly decrease the time spent on reading images.
- Denoising the image and computing *lightAverages* are two computationally demanding tasks and, unfortunately, it may be difficult to improve the speed of these, as they are implemented using fast GPU (graphic processing unit) operations in Matlab. Even when we implemented these operations using efficient sliding window implementation in C, the CPU (Central Processing Unit) computation was much slower. In case of dire need for future increase in computational speed, it is possible to try using CUDA or similar GPU library directly inside the C code performing the segmentation. However, as Matlab itself uses CUDA for the GPU operations, the increase in computational speed is uncertain.
- While it may be possible to further speed up the C code performing the segmentation, it is unlikely to bring a large improvement, as, according to the Visual Studio profiler, 70% time spent in the C code is used for passing parameters and outputs between Matlab and C, therefore even 90% reduction of time spent on actual segmentation would matter very little overall.

### 3.1.2 Evaluation on *in vitro* data

Coelho et al. made a comprehensive comparison of several segmentation algorithms using *in vitro* data of U2OS and NIH3T3 cells (Coelho et al., 2009). See Fig. 3.1 for an example of how the data look. As the datasets and used methodology were published completely, we decided to measure the performance of SeNeCA using the methods described in the publication, despite the fact that SeNeCA was not primarily aimed at use with *in vitro* data.

[1]Solid state drive, a HDD with much faster read access than "ordinary" magnetic drive

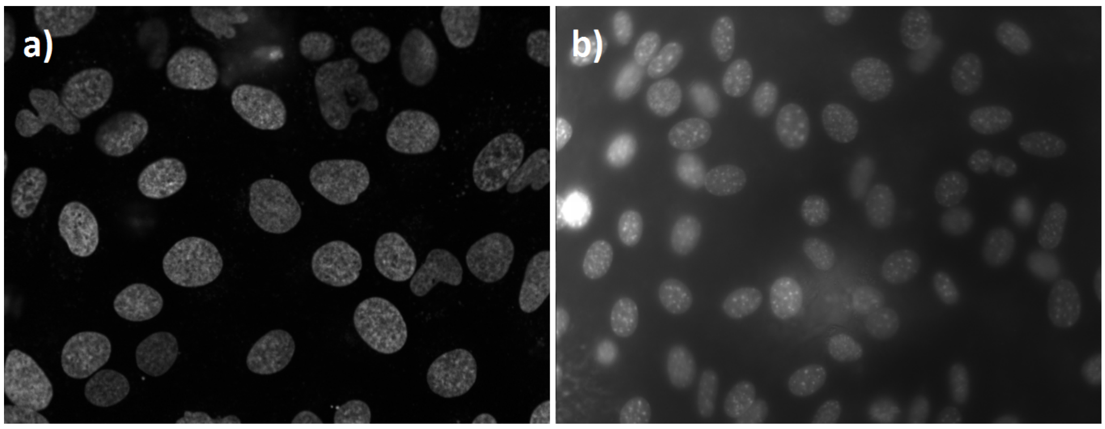


**Figure 3.1:** a) An example of U2OS cells.
b) An example of NIH3T3 cells.

#### 3.1.2.1 Description of evaluation

Both datasets (U2OS and NIH3T3) contain 50 images and their copies with manual annotation of neuronal borders added. The U2OS and NIH3T3 datasets contain, in average, 36.6 and 43.6 neurons per image, respectively. For each algorithm tested in (Coelho et al., 2009), the algorithm was used to segment an image and four classes of errors were counted, with the manual annotation used as a reference: *split* (a single neuronal soma segmented as two or more by the algorithm), *merged* (several neurons segmented as one object), *spurious* (non-neuron wrongly segmented as a neuron) and *missing* (a neuron wrongly not segmented as such). To compare the performance of tested algorithms to human performance, manual segmentation by another researcher was performed by another human annotator (A. Shariff), also from Murphylab.

The segmentation of a single *in vitro* image by SeNeCA was performed in the following way: The image was denoised using disk blur, then segmented using SeNeCA, with too small and/or too large objects filtered out. Parameters of SeNeCA were tuned manually [2], but for each dataset, only a single set of parameters was used. Both filtering inappropriately sized objects and manual tuning of parameters were used in the original work of (Coelho et al., 2009), therefore SeNeCA did not receive any special advantage.

#### 3.1.2.2 Results

The results of SeNeCA were added to those of algorithms measured in (Coelho et al., 2009), see Table 3.2. The table shows that SeNeCA performs very well even on *in vitro* data, being notably better than other measured algorithms, almost rivaling the performance of the human annotator. Especially important is the low number of *missing* objects. In some application (e.g., counting cells), a *spurious* error is just as bad as *missing*, but in other applications (e.g., measuring neuronal activity), *missing* errors are much more important that *spurious*, because the data from *spurious* objects may be simply discarded.

However, let us note that we believe that there are certain errors in the original

[2]The tuning lasted less than a minute. No large search of parameter space was performed.

| Algorithm | Split | Merged | Spurious | Missing | Sum errors |
|---|---|---|---|---|---|
| A.S. Manual | 1.6/1.0 | 1.0/1.2 | 0.8/0.0 | 2.2/3.2 | 5.5 |
| SeNeCA | 0.0/0.0 | 2.2/4.1 | 1.5/1.5 | 1/1.8 | 6.1 |
| Mean threshold | 1.3/1.4 | 3.4/5.1 | 0.9/3.1 | 3.6/4.8 | 12 |
| Merging algorithm | 1.8/1.6 | 2.1/3.0 | 1.0/6.8 | 3.3/5.9 | 13 |
| RC Threshold | 1.1/1.0 | 2.4/2.4 | 0.3/1.9 | 5.5/22.1 | 18 |
| Watershed (gradient) | 7.7/2.6 | 2.0/3.0 | 2.0/11.4 | 2.9/5.4 | 19 |
| Otsu Threshold | 1.1/0.8 | 2.3/2.1 | 0.3/1.7 | 5.6/26.6 | 20 |
| Watershed (direct) | 13.8/2.9 | 1.2/2.4 | 2.0/11.6 | 2.0/5.5 | 21 |
| Active Masks | 10.5/1.9 | 2.1/1.5 | 0.4/3.9 | 10.8/31.1 | 31 |

**Table 3.2:** The errors made by algorithms, given as the mean number of errors of a given type related to the mean number of cells per one image. The numbers before slash are errors on the U2OS dataset, the numbers after are errors on the NIH3T3 dataset. The *sum errors* is an average of sums of all error types between the two datasets.

annotated dataset[3]. First, it is not entirely consistent whether small parts of cells at the borders of the image are segmented as such or not. Second, there are several cases when a pair of adjacent cells is manually segmented as only one cell.

It is interesting that SeNeCA made 0/0 *split* errors, while the second human annotator had 1.6/1.0 *split* errors. We hypothesize that this is actually not a great feature of SeneCA, but a *merged* sort of error in the original annotation of the dataset, which was discovered and correctly segmented by the second human annotator, while SeNeCA was too simple to determine, that there are two fused cells.

### 3.1.3 Evaluation on *in vivo* data

This part of the evaluation attempts to measure the performance of SeNeCA on *in vivo* data, when a human annotation is taken as a reference.

#### 3.1.3.1 Description of data

The data were recorded from the auditory cortex of mouse, layer II/III, loaded with OGB-1 (a green calcium indicator staining neurons and glial cells) and Sulforhodamin101 (used to stain astrocytes), using multi-cell bolus loading. The rate of frame acquisition of 50 frames per second was used, with the resolution of 256x256 and field of view 150x150 µm. Subsequent 100 images of the recording were taken for further processing.

#### 3.1.3.2 Description of evaluation

A human annotator (Ondřej Novák) annotated the 100 *in vivo* images described above. The same images were then annotated by SeNeCA and the same errors as in *in* vitro evaluation were measured: *split*, *merged*, *spurious*, *missing*. The

[3]Nevertheless, wanting to have results comparable to the original published work of Murphy-lab, we took the reference annotation as true, even though we sometimes thought it incorrect.

parameters of SeNeCA were manually tuned on the first image of the sequence, so that no neuron marked by the human annotator was missed. This was motivated by the fact that for measuring neuronal activity, *missing* is the worst type of error and should be avoided preferentially.

#### 3.1.3.3 Results

When writing the numbers of errors of various classes, the numbers are written as a percentage of the number of neurons in the image by the human annotator[4]. After performing the segmentation of the dataset using SeNeCA, the results were: *split*: 0.08%, *merged*: 0.3%, *spurious*: 48.7%, *missed*: 5.9%. Therefore, even though very few neurons were *missed*, the drawback seems to lie in a high percentage of *spurious* objects (the human annotator marked 25.6 objects per image in average, while SeNeCA found 36.5 per image in average).

However, it is vital to realize that of the almost 50% of spurious objects, a nontrivial portion could be real neurons, which were not discovered by the human annotator. It is impossible to determine with certainty, whether SeNeCA or the human annotator were more accurate. More about this problem in 3.1.4, where it was possible to know for sure whether an object is a neuron or not.

### 3.1.4 Evaluation on *in silico* data

When it became obvious that it is difficult to properly compare performance of SeNeCA to human annotator, due to the lack of knowledge who is more accurate, we decided to create an artificial dataset, where it can be determined, what is a neuron and what is not. This *in silico* evaluation has three purposes: First, to measure the performance of SeNeCA compared to a human (neuroscientist) annotator. Second, to measure the absolute performance and computational speed of SeNeCA on several different artificial datasets. Third, to show that SeNeCA can deal with uneven background brightness much better than "conventional" algorithms.

#### 3.1.4.1 Artificial data modeling

The text of this section is taken from (Tomek et al., in press):

To be able to evaluate absolutely the level of errors produced in our algorithmic segmentation, we created artificial datasets simulating real *in vivo* full-frame data. We used MATLAB. For all of the different resolutions and fields of view, we created an orthogonal space whose base had the same resolution as the corresponding dataset and the number of pixels in the z-axis was set to one-third of the base side. Into this space, spherical neurons (filled spheres) were placed randomly (neurons could not be nested within each other). The number of those neurons was calculated according to published neuronal densities (DeFelipe et al., 2002). The diameter was given by normal distribution with an average of 9 µm and a standard deviation of 1 µm. The minimal and maximal diameters of the neurons were set to 6 µm and 12 µm, respectively. Dendrite-like objects were

[4] I.e., 3 missed neurons from 10 marked by the human annotator result in 30% *missed* error.

added into the extracellular space (straight lines oriented randomly). Those lines added zero brightness inside any neuronal body. The brightness of the cells, extracellular space and dendrite-like objects was set according to values we measured on real in vivo data from fast acquisition. Those values were 0.278 for average intensity inside cell somata and dendrite-like objects and 0.15 for the extracellular space; fractions of the intensity of a white point. Subsequently, the intracellular and extracellular spaces were separately corrupted with Gaussian noise (level of added noise: 0.1608 extracellular spaces, 0.1243 intracellular spaces). When the artificial cortical space was prepared, we took 100 randomly chosen sections. To mimic real images that could be obtained using a two-photon microscope, we corrupted the chosen sections with their neighboring sections according to an approximate axial point spread function (PSF). The PSF was modeled as a Gaussian distribution with 1.5 µm standard deviation. Given that we corrupted the chosen sections with their neighboring sections, the values of brightness and noise intensities, now measured from the chosen corrupted sections, inevitably differed from those initially set for the entire artificial space. The initial values of brightness and noise levels were appropriately changed and the process was automatically iterated until the desired values (same as the initial ones) were obtained from the chosen corrupted sections.

#### 3.1.4.2 Used data

Examples of the artificial data used in this experiment, with various values of FOV (field of view) and resolution, are in 3.2.

##### 3.1.4.2.1 SeNeCA versus human

Two datasets were used, both with resolution 256x256, one with FOV 150 µm (50 images), the other one with FOV 300 µm (25 images).

##### 3.1.4.2.2 Absolute performance of SeNeCA

Six different datasets were used to test a variety of possible imaging conditions: 128x128 with FOV 150 µm, 256x256 with FOV 150 µm, 256x256 with FOV 300 µm, 512x512 with FOV 150 µm, 512x512 with FOV 300 µm and 512x512 with FOV 600 µm. See Fig. 3.2a-f for an example of these data.

##### 3.1.4.2.3 Performance on data with uneven background brightness

Two datasets containing 100 images with resolution 256x256 and FOV 150 µm and 300 µm, respectively, were used. They were artificially corrupted in several different ways to model the problem of uneven background brightness.

#### 3.1.4.3 Description of evaluation

In all the evaluations, we considered to be a neuron in-focus (therefore it should be segmented), if its distance of its centroid from the focal plane of the section is less than three fourths of its radius. Again, we measured the number of *split*, *merged*, *spurious* and *missing* errors.

Even though this evaluation tries to determine qualities of SeNeCA algorithm, we perform two support operations nevertheless, as they are an important part of

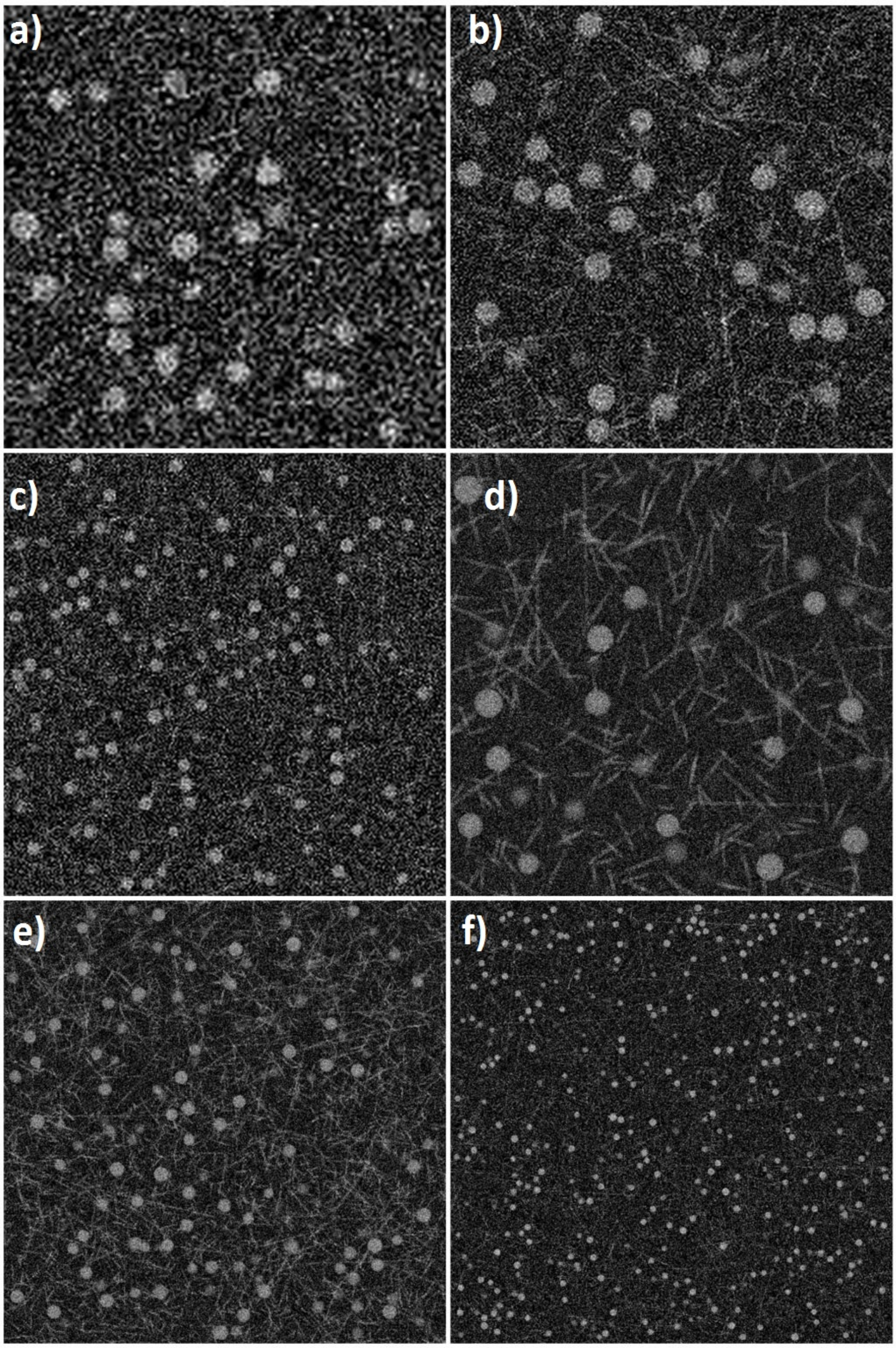


**Figure 3.2:** Examples of artificial data used in evaluation: a) 128x128, FOV 150 µm; b) 256x256, FOV 150 µm; c) 256x256, FOV 300 µm; d) 512x512, FOV 150 µm; e) 512x512, FOV 300 µm; f) 512x512, FOV 600 µm;

the segmentation process[5]: First, the image to be segmented is first blurred using a disk blur to remove excess noise. Second, too large and/or too small objects are filtered out, with the minimum and maximum size being a parameter.

We used an automated solution for counting errors. Let us have a $reference$ image (where locations of neurons are marked) and *mask* image (locations of neurons according to the segmentation algorithm). For each object in $reference$, which does not have a corresponding object at the relevant position *mask*[6], increment the counter of *missing* by one. Similarly, for each object in *mask*, which is not present in $reference$, increment the counter of *spurious* by one. For each object in $reference$ which intersects at least two objects in *mask*, increment *split* by the number of the objects the object in $reference$ intersects. Similarly, for each object in *mask*, which intersects at least two objects in $reference$, increment *merged* by the number of the objects the object in *mask* intersects.

In all parts of the *in silico* evaluation, we performed a search of parameter space to find the optimal set of parameters automatedly. For each dataset, we repeated the following 10 times:

- A random image from the dataset was selected, with the optimal set of parameters found (via search of parameter space) and saved. We allowed use of two metrics of what is "optimal": First, minimizing the sum of errors, second, minimizing the number *missing* errors (in case of the same amount of *missing*, *merged* was minimized).
- Then, the dataset was segmented and the four classes of errors counted.

The resulting performance of the algorithm is given as the average of performance of the algorithm across the 10 repeats[7].

#### 3.1.4.3.1 SeNeCA versus human

A human annotator (Ondřej Novák) has been shown the first five images of both datasets, along with images showing which neurons are considered to be in-plane and should therefore be circumscribed. The annotator then annotated the dataset, concentrating on minimizing the number of *missing* objects. Then, SeNeCA was used to segment the datasets as well. The parameters of SeNeCA were tuned to minimize the overall sum of errors.

#### 3.1.4.3.2 Absolute performance of SeNeCA

SeNeCA was used to segment six different datasets. For each dataset, a search of parameter space has been performed to find the best combination of parameters. Two different metrics were used (and measured separately) to determine "the best" parameters: minimizing the sum of errors and minimizing the amount of

[5]When other algorithms were tested, they were treated in the same way.

[6]It is not necessary for the objects in $reference$ and *mask* to cover each other perfectly. It is sufficient if there is any object in *mask* at the position of an object in $reference$ to prevent the *missing* error.

[7]This emulates the real use of the segmentation algorithm, when parameters are typically tuned on a single image from a dataset and then used for the remainder of the data. To prevent the use of a singular, non-typical image, we took 10 random images instead.

*missing* errors[8]. When measuring the speed of computation, we used the fast Matlab-C version of SeNeCA.

#### 3.1.4.3.3 Performance on data with uneven background brightness

To verify that SeNeCA is indeed better suited for segmenting unevenly stained data than algorithms conventionally used to segment *in vitro* data, we measured the performance of SeNeCA (tuning the parameters to minimize the sum of errors) on the two above described datasets[9], corrupted in four ways:

1. No corruption.
2. Gaussian extrasomatal: Dendrites and background modulated by a Gaussian distribution, such as that pixels at the border of the image were multiplied by 0.5, while pixels at the center of the image were multiplied by 1. This corruption may, in MCBL staining, occur due to poor dispersion of the dye.
3. Gaussian global: The same as case before, but whole image (including neurons) was modulated. This corruption may be a result of, e.g., objective vignetting.
4. Linear ramp: The whole image was modulated by a 0-1 linear ramp in the x-axis. This corruption occurs when one side of the focal plane is deeper in the brain than the other side.

We implemented three algorithms used in (Coelho et al., 2009) to compare SeNeCA to them. We decided to use Mean threshold and Otsu threshold, because they scored high, along with a marker-based watershed (which should be able to cope with uneven background too, to an extent, as it marks its foreground according to local maxima, not global ones). In all these three algorithms, we relaxed as many constants as possible to be parameters and performed a search of parameter space to find the best result minimizing sum of errors.

### 3.1.4.4 Results

#### 3.1.4.4.1 SeNeCA versus human

The Table 3.3 summarizes the results of SeNeCA versus human performance. An important fact is the much lower percentage neurons *missed* by SeNeCA, compared to the human annotator. In this aspect, SeNeCA clearly outperformed the human annotator. The manual annotation led to a lower number of *spurious* objects. However, even the percentage of *spurious* objects segmented by SeNeCA is not critically high. Furthermore, in calcium imaging of neuronal activity, *spurious* information is not such a large problem as it may be simply discarded. The percentage of *merged* and *split* errors is low both in human and SeNeCA, with the human annotator being slightly better.

[8]If more than one combination of parameters led to the same number of *missing* errors, the one minimizing *merged* errors was taken

[9]Stack slices were corrupted, before being added together, in emulation of PSF.

The high number of objects missed by a human suggests that in 3.1.3, a portion of "*spurious* " objects found by SeNeCA could have indeed been real neurons, which were originally missed by the annotator.

| Performer | FOV [µm] | Split | Merged | Spurious | Missing | Neurons |
|---|---|---|---|---|---|---|
| Human | 150x150 | $0 \pm 0$ | $0 \pm 0$ | $0.2 \pm 1.1$ | $18.5 \pm 9.1$ | $27.3 \pm 4.3$ |
| SeNeCA | 150x150 | $0.2 \pm 0.2$ | $1.9 \pm 0.6$ | $7.0 \pm 2.5$ | $4.0 \pm 4.3$ | $27.3 \pm 4.3$ |
| Human | 300x300 | $0.3 \pm 0.5$ | $1.2 \pm 1.3$ | $0.1 \pm 0.3$ | $29.7 \pm 7.3$ | $108 \pm 14$ |
| SeNeCA | 300x300 | $0.6 \pm 0.2$ | $1.9 \pm 0.6$ | $4.8 \pm 1.3$ | $4.6 \pm 2.0$ | $108 \pm 14$ |

**Table 3.3:** The comparison of SeNeCA and a human annotator. Errors (*split, merged, spurious, missing*) are given in %; the number of errors of the particular type related to the mean number of cells per one image. Values are means ± SD.

#### 3.1.4.4.2 Absolute performance of SeNeCA

Table 3.4 shows the best performance of SeNeCA on various datasets, when the sum of errors was minimized. Aside from excellent speed of processing, it shows that all errors can be kept under 10% and occurrence of *split* and *merge* errors is mostly very low. An important result is that it seems that neither the resolution, nor the FOV seem to have a critical impact on the performance of SeNeCA, which suggests that SeNeCA may be used for a large variety of data types.

Table 3.5 differs from the previous one in that number of *missing* errors was minimized[10], instead of sum of errors. This causes a large drop in *missing* error (below 3%), but it leads to a huge increase in *spurious* error percentage. Both *split* and *merged* were slightly increased. Whether the large number of *spurious* errors is feasible or not is dependent on the application of the segmentation. If *missing* objects is very costly, while *spurious* objects may be simply discarded, then the large number of *spurious* objects is not an issue. However, e.g., in counting cells, the large number of *spurious* objects is critical and would cause large errors. The large number of *spurious* objects also leads to slower segmentation (some work is spent on worthless objects) as demonstrated by the 512x512 600 µm FOV dataset, processing of which took the most time from the 512x512 datasets, unlike in Table 3.4.

Of course, this section shows only how well SeNeCA algorithm *can* perform, as we automatically searched the space of parameters and found the best option. However, we believe that due to clear meaning of the parameters and predictable changes in behavior when the parameter values are changed, finding a near-optimal set of parameters is easy even by manual tuning by a user of TPP.

#### 3.1.4.4.3 Performance on data with uneven background brightness

[10]If same number of *missing* errors was made by several combinations of parameters, the one minimizing the number of *merged* errors was taken.

| Resolution | FOV | TPF | Split | Merged | Spurious | Missed | Neurons |
|---|---|---|---|---|---|---|---|
| 128x128 | 150 | 3.1±0.3 | 0.9±0.2 | 3.0±0.7 | 6.1±1.0 | 4.3±1.6 | 30.8±7.3 |
| 256x256 | 150 | 6.3±0.7 | 0.2±0.2 | 1.3±0.7 | 8.2±6.0 | 5.8±3.4 | 27.3±4.3 |
| 256x256 | 300 | 5.7±0.8 | 0.6±0.2 | 1.7±0.5 | 4.8±1.2 | 5.6±1.9 | 108±14 |
| 512x512 | 150 | 27±4 | 0.2±0.1 | 1.2±0.5 | 9.4±3.3 | 3.2±1.9 | 25.2±6.3 |
| 512x512 | 300 | 21±2 | 0.4±0.2 | 2.0±0.7 | 7.9±1.8 | 4.3±1.8 | 99±12 |
| 512x512 | 600 | 21±2 | 0.4±0.1 | 1.4±0.4 | 4.8±1.4 | 9.1±4.2 | 417±24 |

**Table 3.4:** The absolute performance of SeNeCA, when the sum of errors was minimized. FOV is given as a side length of the observed focal plane in µm. TPF means *time per frame* and gives how much time was, in average, spent on a single frame, in milliseconds. Errors (*split, merged, spurious, missing*) are given in %; the number of errors of the particular type related to the mean number of cells per one image. Values are means ± SD.

| Resolution | FOV | TPF | Split | Merged | Spurious | Missed | Neurons |
|---|---|---|---|---|---|---|---|
| 128x128 | 150 | 3.1±0.2 | 1.1±0.1 | 3.5±0.2 | 21±10 | 2.8±0.9 | 30.8±7.3 |
| 256x256 | 150 | 6.5±0.8 | 0.4±0.2 | 1.6±0.6 | 14.8±4.4 | 2.0±1.5 | 27.3±4.3 |
| 256x256 | 300 | 6.5±0.8 | 1.0±0.1 | 2.6±0.4 | 40±15 | 1.4±0.5 | 108±14 |
| 512x512 | 150 | 27±4 | 0.3±0.1 | 1.8±0.3 | 19.0±2.8 | 1.4±0.4 | 25.2±6.3 |
| 512x512 | 300 | 22±2 | 0.5±0.1 | 2.4±0.3 | 27.2±7.1 | 1.6±0.3 | 99±12 |
| 512x512 | 600 | 31±5 | 0.9±0.1 | 2.4±0.2 | 61±11 | 1.3±0.3 | 417±24 |

**Table 3.5:** The absolute performance of SeNeCA, when the number of *missing* errors was minimized. FOV is given as a side length of the observed focal plane in µ. TPF means *time per frame* and gives how much time was, in average, spent on a single frame, in milliseconds. Errors (*split, merged, spurious, missing*) are given in %; the number of errors of the particular type related to the mean number of cells per one image. Values are means ± SD.

Table 3.6 summarizes the performance of the selected algorithms on the two datasets. While Otsu thresholding works almost as good as SeNeCA in unmodulated data, with any modulation of background brightness, SeNeCA becomes clearly the best algorithm. Its performance decreases only slightly, compared to its performance on the unmodulated dataset. For an example of how the algorithms perform, we included an example of segmentation in Fig. 3.3 [11]

However, the numbers need further analysis. The examination of the structure of sum of errors made by Otsu thresholding, mean thresholding and watershed revealed that *missing* errors are the dominant type of error. Why was the percentage of *merged* errors so low, when it is expected for the central group of neurons in a gauss-modulated image to be merged together by these algorithms, as suggested earlier in 1.9? Or why mean thresholding works the worst on unmodulated data, where it should be, in theory, rather successful?

The answer lies in the filtering of objects according to their size. Fig. 3.4

[11] The segmented images are displayed after small objects were filtered out, but before large objects were filtered. In the case of thresholding algorithms, not much would be left in the image if large objects were filtered too. We believe that the selected images with the large central fusion better reflect what the thresholding algorithms do and what are their shortcomings, which is why we have not shown the images with large objects filtered.

| Corruption | FOV [µm] | SeNeCA | Otsu t. | Mean t. | Watershed |
|---|---|---|---|---|---|
| None | 150x150 | 15.0 ± 2.3 | 21.5 ± 2.9 | 121.0 ± 3.3 | 56.7 ± 2.3 |
| None | 300x300 | 14.7 ± 1.6 | 26 ± 0.6 | 107.0 ± 2.4 | 83.1 ± 0.5 |
| Gauss extra. | 150x150 | 19.2 ± 3.1 | 67.6 ± 3.5 | 80.5 ± 1.9 | 59.9 ± 1.0 |
| Gauss extra. | 300x300 | 17.4 ± 0.7 | 70.2 ± 1.4 | 78.1 ± 1.2 | 80.9 ± 0.2 |
| Gauss global | 150x150 | 15.9 ± 1.8 | 73.1 ± 1.8 | 83.0 ± 1.0 | 72.5 ± 0.2 |
| Gauss global | 300x300 | 16.9 ± 2.2 | 70.9 ± 2.2 | 82.6 ± 1.5 | 87.5 ± 0.6 |
| Linear ramp | 150x150 | 15.8 ± 1.3 | 92.8 ± 1.3 | 90.7 ± 0.6 | 79.8 ± 1.3 |
| Linear ramp | 300x300 | 16.9 ± 2.0 | 88.2 ± 0.4 | 90.6 ± 0.8 | 89.6 ± 2.3 |

**Table 3.6:** Performance of chosen algorithms on the given dataset, with various corruptions of the original images, see 3.1.4.3.3. For each algorithm and corrupted dataset, sum of errors is given in %; the number of errors of any type related to the mean number of cells per one image. Values are means ± SD.

shows the segmentation performed by mean thresholding. We see that during filtering objects by size, the whole central fusion is always deleted as it is too large. This deletion, of course, leads to a large amount of *missing* neurons. To address the question, why mean thresholding performs worse on unmodulated data, let us compare Fig. 3.4b and 3.4d. While mean thresholding made slightly less *missing* errors on the unmodulated image, it also made many more *spurious* errors, by segmenting certain dendritic segments as neuronal somata. Therefore, even though the segmentation of unmodulated data is arguably visually not as bad (there are not as many missing cells), it leads to a higher number of errors.

Of course, it may be questioned, whether it is correct to remove too large objects. First, we believe that it is, as we use the segmentation process to record calcium traces of single cells. Thus, having a large neuronal fusion segmented as single cell is useless as it is impossible to extract signals of single cells from such a data. Second, if too large objects were not filtered out, the number of *missing* errors would drop rapidly, but the former *missing* errors made would simply be transformed into *merged* errors, therefore not impacting the sum of all types of errors at all.

Therefore, to conclude this part of analysis, we have shown that three selected simple algorithms fail to correctly segment data with uneven background brightness, while SeNeCA performs the segmentation with a rather low number of errors. This supports our belief that *in vivo* data need to be treated differently than *in vitro* data: e.g., the mean thresholding had the smallest sum of errors in (Coelho et al., 2009), but it performed abysmally on our simulated data with uneven background brightness.

## 3.2 Comparison of TPP to other software tools

In 1.2.2, we mentioned several software tools, functionality of which intersects with TPP. In this section, we compare TPP to the tools which are aimed (or could be used) for automated extraction of calcium traces of neurons in a full-frame video, which is one of the most important features of TPP. To our knowledge, there is no software other than TPP, which would, on top of the recording of calcium traces, offer the step of inferring spike trains. In this aspect, TPP seems to be more self-standing and complete.

### 3.2.1 BioImageXD and ImageJ

Both BioImageXD (Kankaanpää et al., 2012) and ImageJ (Collins, 2007) are large software projects, capable of solving many different tasks in many sorts of data. Compared to this approach, TPP is much more specialized. We believe that if the functionality of TPP is not limiting to a user, the process of learning how to operate it should be faster than in the case of large software tools.

According to our experiments, while both BioImageXD and ImageJ allow segmentation, tracking and measurement of ROI separately, these features are not yet integrated into a processing pipeline. However, we believe that as the features are already separately solved and both projects are open-source, putting them together should be possible. Similarly, both projects could be extended to use one of the published algorithms for inference of spike trains.

Both those projects offer many simple segmentation algorithms based on thresholding and watershed, but they suffer from lack of segmentation tools able to segment *in vivo* data with uneven background brightness. The best algorithm for the task in ImageJ, according to our experiments, was RATS. Unfortunately, its performance was still quite poor (slightly better than, e.g., Otsu thresholding, but much worse than SeNeCA). In BioImageXD, we obtained the best results using Dynamic threshold (probably a very similar algorithm to SeNeCA; with a single threshold), but its performance was not as good as that of SeNeCA and the process of segmentation was extremely slow (more than four seconds per 256x256 image)[12].

BioImageXD is a very recent project and although it offers a lot of functionality and it seems to be extremely well designed, its computations are rather slow and it still suffers from frequent bugs and crashes.

### 3.2.2 ISA

The ISA software is, in contrast to previously discussed projects, aimed purely at automated extraction of calcium traces of neurons in a full-frame video. It suffers from two major problems, which, we believe, makes it ill-suited for processing *in vivo*. The first problem is the only available segmentation algorithm: marker-based watershed. As we have shown in 3.1.4.4.3, marker-based watershed is not able to cope with uneven background brightness, often found in *in vivo* data images. Even its performance on *in vitro* is not very good, as shown in (Coelho et al., 2009). The second problem is the absence of tracking. ISA simply segments the first frame of the video and then measures the intensity of the segmented regions over time. Therefore, ISA is very vulnerable to in-plane motion artifacts and it does not register new neurons appearing in the video.

[12]This may not seem to be "extremely slow", however, let us imagine a 10 minutes long recording at 50 frames per second captured, which yields 30000 frames. It would take more than 30 hours to perform the most basic segmentation using BioImageXD's dynamic threshold, while if SeNeCA was used, the task would be finished in less than three minutes.

### 3.2.3 CellSort

CellSort takes a different approach as compared to the previous three projects, focusing on processing of temporal information in the video during segmentation. Unfortunately, it fails to detect silent neurons (non-spiking) or cells which fire synchronously (Valmianski et al., 2010). Furthermore, according to our experiments, the segmentation algorithm works reasonably well only in very small images (64x64 pixels or smaller). In larger images, it tends to detect noise (even after denoising), rather than neurons. These drawbacks are so significant that we do not describe this tool further.

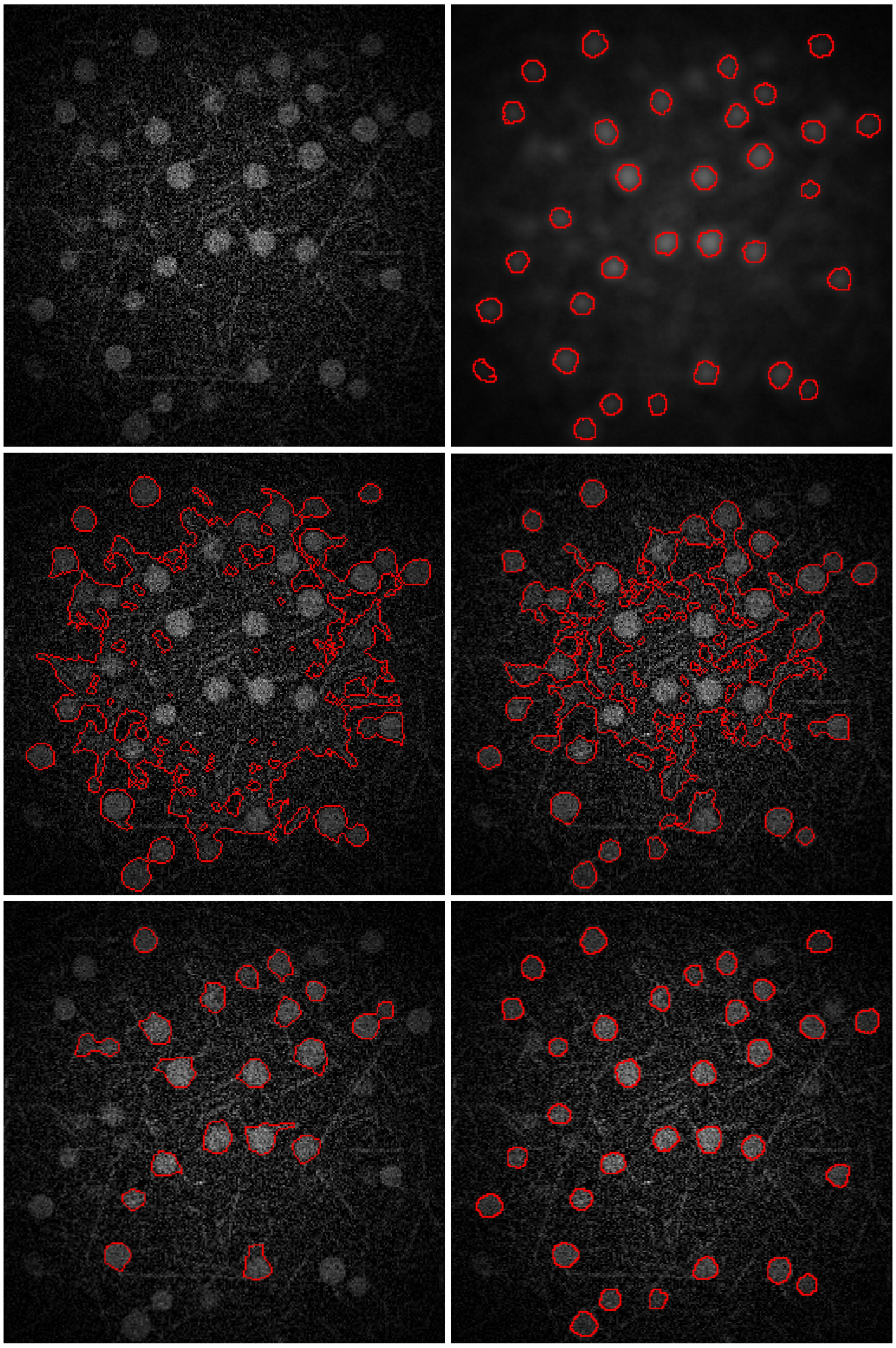


**Figure 3.3:** Examples of segmentation by various algorithms: a) The original image (corrupted by Gauss extrasomatal modulation); b) blurred original image with "true" neurons marked; c) Segmentation by mean thresholding; d) Segmentation by Otsu thresholding; e) Segmentation by watershed; f) Segmentation by SeNeCA.

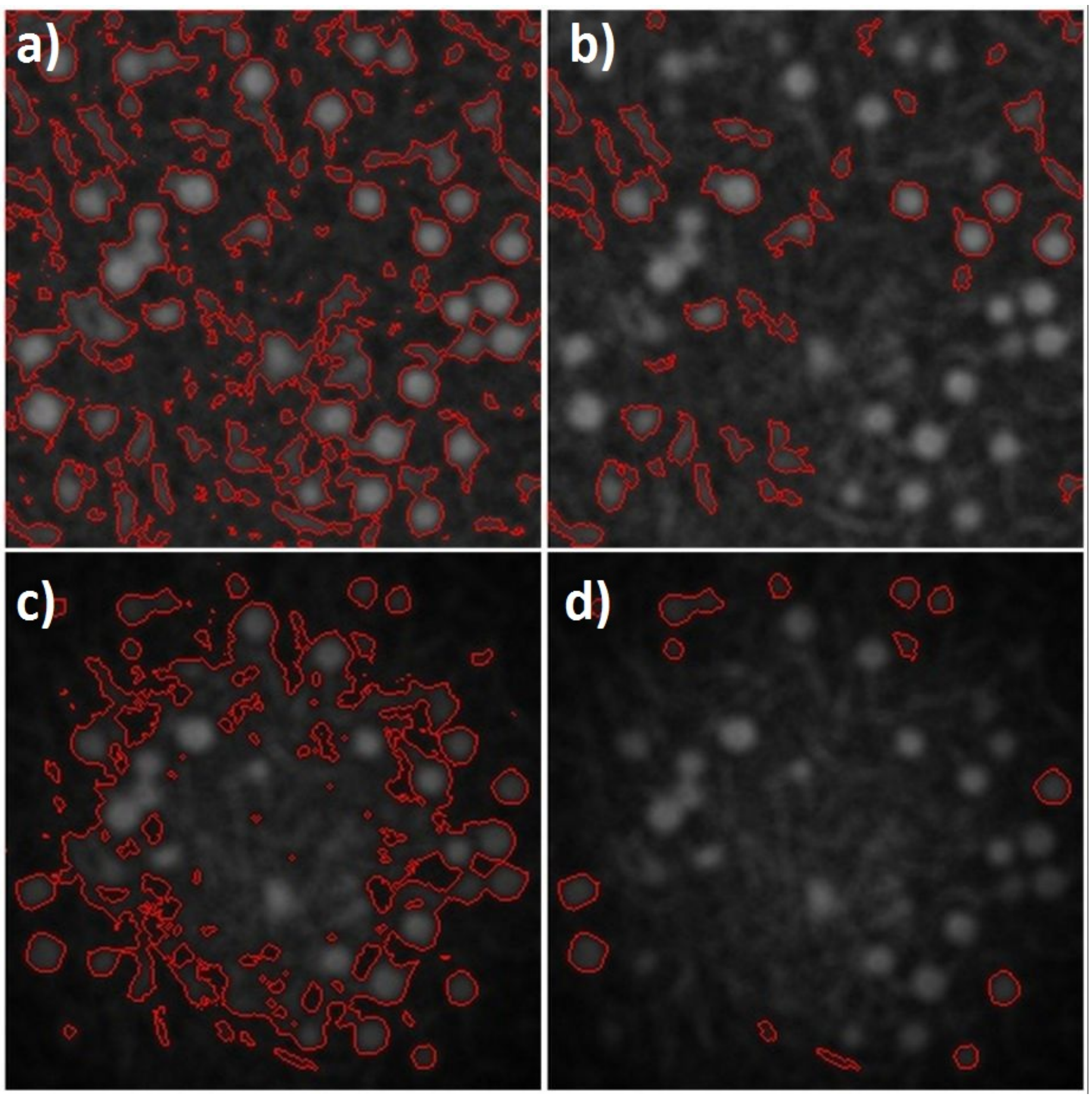


**Figure 3.4:** Performance of mean thresholding under various circumstances: a) No modulation of image, no size filtering; b) No modulation of image, with size filtering (tiny or large objects removed); c) External Gauss modulation of image, no size filtering; d) External Gauss modulation of image, with size filtering (tiny or large objects removed).

# Conclusion

This thesis described the Two-Photon Processor software toolkit for complex processing of data from 2P microscope. It is aimed primarily at automated extraction of spike trains from both full-frame and line-scan *in vivo* data and, according to our evaluation, it seems to be the best tool for this task at the moment (comparison to other tools is in 3.2). An important algorithm introduced in this thesis is SeNeCA: a new segmentation algorithm, capable of processing full-frame *in vivo* data. We published TPP and SeNeCA in (Tomek et al., in press).

The evaluation of SeNeCA (see 3.1) shows that the algorithm is extremely fast, able to process up to 200 frames of 256x256 video per second, when a hybrid Matlab-C implementation is used. We also showed that it provides an excellent quality of segmentation across several different types of data. It seems to be the first algorithm for segmenting neurons, able to rival (3.1.2) or even outclass (3.1.4) a human annotator.

The speed of SeNeCA could be, in future, used for optogenetic experiments (Prakash et al., 2012). Using bacterial opsins (Gradinaru et al., 2010; Gunaydin et al., 2010; Yizhar et al., 2011), it is possible to make neurons fire action potentials when they are targeted by light with a particular wavelength (different from the wavelength employed by 2P microscope). Let us imagine a certain type of experiment with monitoring neuronal activity in an area of brain using SeNeCA, waiting for a specific firing patterns in neurons. After the occurrence of such a pattern, a follow-up routine driving the second laser (used to make neurons fire) may be triggered and within a short time latency given by SeNeCA's segmentation) perturb the firing pattern. This makes many "what if"experiments possible and it may be used to better understand (or even alter) learning and/or other cognitive processes in living animals, although, at this stage, only in a rather small scale of hundreds of neurons at most.

There are two main future extensions of Two-Photon Processor possible. First, the current state of Realtime worker (2.4.6) uses only a simple heuristics for detection of spikes. Its performance would be improved if a more online, in realtime. The second possible improvement of TPP would be to include algorithms for inference of neuronal connectome (i.e., which neuron projects to which neurons) from a reconstructed voxel of tissue. We believe that knowing the connectivity of neurons will facilitate the task of deciphering brain computations in future.

# Bibliography

BANDYOPADHYAY, Sharba, SHAMMA, Shihab A., KANOLD, Patrick O. Dichotomy of functional organization in the mouse auditory cortex. *Nature neuroscience*, **13**(3):361–368, 2010.

BEUCHER, Serge, LANTUEJOUL, Christian. Use of watersheds in contour detection. In *International Workshop on Image Processing*, 1979.

BIRAN, Roy, MARTIN, David C., TRESCO, Patrick A. Neuronal cell loss accompanies the brain tissue response to chronically implanted silicon microelectrode arrays. *Experimental neurology*, **195**(1):115–126, 2005.

BOCK, Davi D., LEE, Wei-Chung Allen, KERLIN, Aaron M., ANDERMANN, Mark L., HOOD, Greg, WETZEL, Arthur W., YURGENSON, Sergey, SOUCY, Edward R., KIM, Hyon Suk, REID, R. Clay. Network anatomy and in vivo physiology of visual cortical neurons. *Nature*, **471**(7337):177–182, 2011.

BOULANGER, Jérôme, KERVRANN, Charles, BOUTHEMY, P., ELBAU, P., SIBARITA, J. B., SALAMERO, J. Patch-based non-local functional for denoising fluorescence microscopy image sequences. *IEEE Transactions on Medical Imaging*, **28**:12, 2009.

BRIGGMAN, Kevin L., BOCK, Davi D. Volume electron microscopy for neuronal circuit reconstruction. *Current opinion in neurobiology*, 2011.

BUADES, Antoni, COLL, Bartomeu, MOREL, Jean Michel. Image denoising methods. a new nonlocal principle. *SIAM review*, **52**(1):113–147, 2010.

BUZSáKI, György, ANASTASSIOU, Costas A., KOCH, Christof. The origin of extracellular fields and currents—eeg, ecog, lfp and spikes. *Nature Reviews Neuroscience*, **13**(6):407–420, 2012.

CHOREV, Edith, EPSZTEIN, Jérôme, HOUWELING, Arthur R., LEE, Albert K., BRECHT, Michael. Electrophysiological recordings from behaving animals—going beyond spikes. *Current opinion in neurobiology*, **19**(5):513–519, 2009.

COELHO, Luís Pedro, SHARIFF, Aabid, MURPHY, Robert F. Nuclear segmentation in microscope cell images: a hand-segmented dataset and comparison of algorithms. In *Biomedical Imaging: From Nano to Macro, 2009. ISBI'09. IEEE International Symposium on*, pages 518–521. IEEE, 2009.

COHEN, Lior, ROTHSCHILD, Gideon, MIZRAHI, Adi. Multisensory integration of natural odors and sounds in the auditory cortex. *Neuron*, **72**(2):357–369, 2011.

COLLINS, Tony J. Imagej for microscopy. *Biotechniques*, **43**(1):25–30, 2007.

DANGETI, Sarita. *Denoising Techniques-A Comparison.* PhD thesis, 2003.

DEFELIPE, Javier, ALONSO-NANCLARES, Lidia, ARELLANO, Jon I. Microstructure of the neocortex: comparative aspects. *Journal of neurocytology*, **31**(3-5):299–316, 2002.

DENK, W., DELANEY, K. R., GELPERIN, A., KLEINFELD, D., STROWBRIDGE, B. W., TANK, D. W., YUSTE, R. Anatomical and functional imaging of neurons using 2-photon laser scanning microscopy. *Journal of neuroscience methods*, **54**(2):151–162, 1994.

DOMBECK, Daniel A., GRAZIANO, Michael S., TANK, David W. Functional clustering of neurons in motor cortex determined by cellular resolution imaging in awake behaving mice. *The Journal of Neuroscience*, **29**(44):13751–13760, 2009.

DYER, Eva, DUARTE, Marco, JOHNSON, Don, BARANIUK, Richard. Recovering spikes from noisy neuronal calcium signals via structured sparse approximation. *Latent Variable Analysis and Signal Separation*, pages 604–611, 2010.

EICHHOFF, Gerhard, KOVALCHUK, Yury, VARGA, Zsuzsanna, VERKHRATSKY, Alexei, GARASCHUK, Olga. *In Vivo Ca2+ Imaging of the Living Brain Using Multi-cell Bolus Loading Technique*, volume **43**, chapter 11, pages 205–220. Humana Press, 2010.

GEE, K. R., BROWN, K. A., CHEN, W. N., BISHOP-STEWART, J., GRAY, D., JOHNSON, I. Chemical and physiological characterization of fluo-4 ca (2+)-indicator dyes. *Cell calcium*, **27**(2):97, 2000.

GRADINARU, Viviana, ZHANG, Feng, RAMAKRISHNAN, Charu, MATTIS, Joanna, PRAKASH, Rohit, DIESTER, Ilka, GOSHEN, Inbal, THOMPSON, Kimberly R., DEISSEROTH, Karl. Molecular and cellular approaches for diversifying and extending optogenetics. *Cell*, **141**(1):154–165, 2010.

GREENBERG, David S., KERR, Jason N. D. Automated correction of fast motion artifacts for two-photon imaging of awake animals. *Journal of neuroscience methods*, **176**(1):1–15, 2009.

GREWE, Benjamin F., LANGER, Dominik, KASPER, Hansjörg, KAMPA, Björn M., HELMCHEN, Fritjof. High-speed in vivo calcium imaging reveals neuronal network activity with near-millisecond precision. *Nature methods*, **7**(5):399–405, 2010.

GRYNKIEWICZ, Grzegorz, POENIE, Martin, TSIEN, Roger Y. A new generation of ca2+ indicators with greatly improved fluorescence properties. *Journal of Biological Chemistry*, **260**(6):3440–3450, 1985.

GUNAYDIN, Lisa A., YIZHAR, Ofer, BERNDT, André, SOHAL, Vikaas S., DEISSEROTH, Karl, HEGEMANN, Peter. Ultrafast optogenetic control. *Nature neuroscience*, **13**(3):387–392, 2010.

HAMILL, Owen P., MARTY, A., NEHER, Erwin, SAKMANN, Bert, SIGWORTH, F. J. Improved patch-clamp techniques for high-resolution current recording from cells and cell-free membrane patches. *Pflügers Archiv European journal of physiology*, **391**(2):85–100, 1981.

HAND, A. J., SUN, T., BARBER, D. C., HOSE, D. R., MACNEIL, S. Automated tracking of migrating cells in phase−contrast video microscopy sequences using image registration. *Journal of microscopy*, **234**(1):62–79, 2009.

HARRIS, Kenneth D., HENZE, Darrell A., CSICSVARI, Jozsef, HIRASE, Hajime, BUZSáKI, György. Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements. *Journal of Neurophysiology*, **84**(1):401–414, 2000.

HEIDER, Barbara, NATHANSON, Jason L., ISACOFF, Ehud Y., CALLAWAY, Edward M., SIEGEL, Ralph M. Two-photon imaging of calcium in virally transfected striate cortical neurons of behaving monkey. *PloS one*, **5**(11): e13829, 2010.

HELMCHEN, Fritjof, DENK, Winfried. Deep tissue two-photon microscopy. *Nature methods*, **2**(12):932–940, 2005.

HODGKIN, Alan L., HUXLEY, Andrew F. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, **117**(4):500, 1952.

HODGKIN, Ao L., HUXLEY, AFj, KATZ, B. Measurement of current-voltage relations in the membrane of the giant axon of loligo. *The Journal of physiology*, **116**(4):424, 1952.

HUBEL, David H., WIESEL, Torsten N. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, **148**(3):574–591, 1959.

JAROSIEWICZ, Beata, SCHUMMERS, James, MALIK, Wasim Q., BROWN, Emery N., SUR, Mriganka. Functional biases in visual cortex neurons with identified projections to higher cortical targets. *Current Biology*, 2012.

KANKAANPää, Pasi, PAAVOLAINEN, Lassi, TIITTA, Silja, KARJALAINEN, Mikko, PäIVäRINNE, Joacim, NIEMINEN, Jonna, MARJOMäKI, Varpu, HEINO, Jyrki, WHITE, Daniel J. Bioimagexd: an open, general-purpose and high-throughput image-processing platform. *Nature Methods*, **9**(7):683–689, 2012.

KERR, Jason N. D., DENK, Winfried. Imaging in vivo: watching the brain in action. *Nature Reviews Neuroscience*, **9**(3):195–205, 2008.

KERR, Jason N. D., GREENBERG, David, HELMCHEN, Fritjof. Imaging input and output of neocortical networks in vivo. *Proceedings of the National Academy of Sciences of the United States of America*, **102**(39):14063–14068, 2005.

KERVRANN, Charles, BOULANGER, Jérôme. Optimal spatial adaptation for patch-based image denoising. *Image Processing, IEEE Transactions on*, **15** (10):2866–2878, 2006.

KNOTT, Graham W., HOLTMAAT, Anthony, WILBRECHT, Linda, WELKER, Egbert, SVOBODA, Karel. Spine growth precedes synapse formation in the adult neocortex in vivo. *Nature neuroscience*, **9**(9):1117–1124, 2006.

KOBAT, Demirhan, HORTON, Nicholas G., XU, Chris. In vivo two-photon microscopy to 1.6-mm depth in mouse cortex. *Journal of biomedical optics*, **16** (10):106014–106014–4, 2011.

KOTLIKOFF, Michael I. Genetically encoded ca2+ indicators: using genetics and molecular design to understand complex physiology. *The Journal of physiology*, **578**(1):55–67, 2006.

KUHLMAN, Sandra J., HUANG, Z. Josh. High-resolution labeling and functional manipulation of specific neuron types in mouse brain by cre-activated viral gene expression. *PLoS One*, **3**(4):e2005, 2008.

KWAN, Alex C. Toward reconstructing spike trains from large−scale calcium imaging data. *hfsp journal*, **4**(1):1–5, 2010.

LEI, Na. *Microsystem Based on CMOS Multielectrode Array for Extracellular Neural Stimulation and Recording.* PhD thesis, 2011.

LIN, Gang, ADIGA, Umesh, OLSON, Kathy, GUZOWSKI, John F., BARNES, Carol A., ROYSAM, Badrinath. A hybrid 3d watershed algorithm incorporating gradient cues and object models for automatic segmentation of nuclei in confocal image stacks. *Cytometry Part A*, **56**(1):23–36, 2003.

LüTCKE, Henry, HELMCHEN, Fritjof. Two-photon imaging and analysis of neural network dynamics. *Reports on Progress in Physics*, **74**(8):086602, 2011.

MARIM, Marcio, ANGELINI, Elsa, OLIVO-MARIN, J. C. Denoising in fluorescence microscopy using compressed sensing with multiple reconstructions and non-local merging. In *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pages 3394–3397. IEEE, 2010.

MARTIAL, Franck P., HARTELL, Nicholas A. Figure 5 a, 2012. URL `http://www.plosone.org/article/info%3Adoi%2F10.1371$%2Fjournal.pone.0043942`.

MEIJERING, Erik. Cell segmentation: 50 years down the road [life sciences]. *Signal Processing Magazine, IEEE*, **29**(5):140–145, 2012.

MEIJERING, Erik, SMAL, Ihor, DANUSER, Gaudenz. Tracking in molecular bioimaging. *Signal Processing Magazine, IEEE*, **23**(3):46–53, 2006.

MüHLPFORDT, Henry. Fluorescencefilters 2008-09-28, 2008. URL `https://en.wikipedia.org/wiki/File:FluorescenceFilters_2008-09-28.svg`.

MIRI, Andrew, DAIE, Kayvon, BURDINE, Rebecca D., AKSAY, Emre, TANK, David W. Regression-based identification of behavior-encoding neurons during large−scale optical imaging of neural activity at cellular resolution. *Journal of Neurophysiology*, **105**(2):964–980, 2011.

MIURA, Kota. Tracking movement in cell biology. *Microscopy Techniques*, pages 1304–1307, 2005.

MOSIG, Axel, JäGER, Stefan, WANG, Chaofeng, NATH, Sumit, ERSOY, Ilker, PALANIAPPAN, Kannap-pan, CHEN, Su-Shing. Tracking cells in life cell imaging videos using topological alignments. *Algorithms for Molecular Biology*, **4**(1):10, 2009.

MUKAMEL, Eran A., NIMMERJAHN, Axel, SCHNITZER, Mark J. Automated analysis of cellular signals from large-scale calcium imaging data. *Neuron*, **63** (6):747–760, 2009.

NIELL, Cristopher M., SMITH, Stephen J. Functional imaging reveals rapid development of visual response properties in the zebrafish tectum. *Neuron*, **45** (6):941–951, 2005.

OTSU, Nobuyuki. A threshold selection method from gray-level histograms. *Automatica*, **11**(285-296):23–27, 1975.

PENG, Hanchuan, RUAN, Zongcai, ATASOY, Deniz, STERNSON, Scott. Automatic reconstruction of 3d neuron structures using a graph-augmented deformable model. *Bioinformatics*, **26**(12):i38–i46, 2010.

PERONA, Pietro, MALIK, Jitendra. Scale-space and edge detection using anisotropic diffusion. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **12**(7):629–639, 1990.

PRAKASH, Rohit, YIZHAR, Ofer, GREWE, Benjamin, RAMAKRISHNAN, Charu, WANG, Nancy, GOSHEN, Inbal, PACKER, Adam M., PETERKA, Darcy S., YUSTE, Rafael, SCHNITZER, Mark J. Two-photon optogenetic toolbox for fast inhibition, excitation and bistable modulation. *Nature methods*, 2012.

RIDLER, T. W., CALVARD, S. Picture thresholding using an iterative selection method. *IEEE transactions on Systems, Man and Cybernetics*, **8**(8):630–632, 1978.

RODRIGUEZ, Alfredo, EHLENBERGER, Douglas B., HOF, Patrick R., WEARNE, Susan L. Three-dimensional neuron tracing by voxel scooping. *Journal of neuroscience methods*, **184**(1):169, 2009.

ROTHSCHILD, Gideon, NELKEN, Israel, MIZRAHI, Adi. Functional organization and population dynamics in the mouse primary auditory cortex. *Nature neuroscience*, **13**(3):353–360, 2010.

SADOVSKY, Alexander J., KRUSKAL, Peter B., KIMMEL, Joseph M., OSTMEYER, Jared, NEUBAUER, Florian B., MACLEAN, Jason N. Heuristically optimal path scanning for high-speed multiphoton circuit imaging. *Journal of neurophysiology*, **106**(3):1591–1598, 2011.

SCHNEIDER, Caroline A., RASBAND, Wayne S., ELICEIRI, Kevin W. Nih image to imagej: 25 years of image analysis. *Nature Methods*, **9**(7):671–675, 2012.

TESCHEMACHER, A. G., PATON, J. F. R., KASPAROV, S. Imaging living central neurones using viral gene transfer. *Advanced drug delivery reviews*, **57** (1):79–93, 2005.

TOMEK, Jakub, NOVAK, Ondrej, SYKA, Josef. Two−photon processor and SeNeCA − a freely available software package to process data from two−photon calcium imaging at speeds down to several ms per frame. *Journal of Neurophysiology*, in press.

VALMIANSKI, Ilya, SHIH, Andy Y., DRISCOLL, Jonathan D., MATTHEWS, David W., FREUND, Yoav, KLEINFELD, David. Automatic identification of fluorescently labeled brain cells for rapid functional imaging. *Journal of neurophysiology*, **104**(3):1803–1811, 2010.

VASILKOSKI, Zlatko, STEPANYANTS, Armen. Detection of the optimal neuron traces in confocal microscopy images. *Journal of neuroscience methods*, **178** (1):197, 2009.

VOGELSTEIN, Joshua T., WATSON, Brendon O., PACKER, Adam M., YUSTE, Rafael, JEDYNAK, Bruno, PANINSKI, Liam. Spike inference from calcium imaging using sequential monte carlo methods. *Biophysical journal*, **97**(2): 636–655, 2009.

VOGT, A. K., BREWER, G. J., OFFENHäUSSER, Andreas. Connectivity patterns in neuronal networks of experimentally defined geometry. *Tissue engineering*, **11**(11-12):1757–1767, 2005.

WIKIPEDIA.ORG. Diagram of a two-photon excitation microscope en, 2006a. URL `http://en.wikipedia.org/wiki/File:Diagram_of_a_two-photon_excitation_microscope_en.svg`.

WIKIPEDIA.ORG. Confocalprinciple in english, 2006b. URL `http://commons.wikimedia.org/wiki/File:Confocalprinciple_in_English.svg`.

WIKIPEDIA.ORG. Myelinated neuron, 2008. URL `http://commons.wikimedia.org/wiki/File:Myelinated$_$neuron.jpg`.

WIKIPEDIA.ORG. Current clamp recording of neuron, 2009. URL `http://upload.wikimedia.org/wikipedia/en/0/01/Current_Clamp_recording_of_Neuron.GIF`.

WONG, Loo Chin, LU, Bo, TAN, Kia Wee, FIVAZ, Marc. Fully-automated image processing software to analyze calcium traces in populations of single cells. *Cell calcium*, **48**(5):270–274, 2010.

WU, Guangying K., TAO, Huizhong W., ZHANG, Li I. From elementary synaptic circuits to information processing in primary auditory cortex. *Neuroscience & Biobehavioral Reviews*, **35**(10):2094–2104, 2011.

YAROSLAVSKY, Leonid P., YAROSLAVSKIJ, L. P. Digital picture processing. an introduction. *Digital picture processing. An introduction.. LP Yaroslavsky*

*(LP Yaroslavskij). Springer Seriesin Information Sciences, Vol. 9. Springer-Verlag, Berlin-Heidelberg-New York-Tokyo. 12+ 276 pp. Price DM 112.00 (1985). ISBN 3-540-11934-5 (FR Germany), ISBN 0-387-11934-5 (USA).*, **1**, 1985.

YIZHAR, Ofer, FENNO, Lief E., DAVIDSON, Thomas J., MOGRI, Murtaza, DEISSEROTH, Karl. Optogenetics in neural systems. *Neuron*, **71**(1):9–34, 2011.

YUAN, Xiaosong, TRACHTENBERG, Joshua T., POTTER, Steve M., ROYSAM, Badrinath. Mdl constrained 3-d grayscale skeletonization algorithm for automated extraction of dendrites and spines from fluorescence confocal images. *Neuroinformatics*, **7**(4):213–232, 2009.

ZHAO, Ting, XIE, Jun, AMAT, Fernando, CLACK, Nathan, AHAMMAD, Parvez, PENG, Hanchuan, LONG, Fuhui, MYERS, Eugene. Automated reconstruction of neuronal morphology based on local geometrical and global structural models. *Neuroinformatics*, **9**(2):247–261, 2011.

# List of Abbreviations

**AM** Acetoxymethyl

**BFS** Breadth-First Search

**CPU** Central Processing Unit

**EM** Electron Microscope

**ESEM** Environmental Scanning Electron Microscope

**FIR** Finite Impulse Response

**FOV** Field of View

**FFT** Fast Fourier Transform

**GPU** Graphic processing unit

**GUI** Graphic user interface

**HDD** Hard Disk Drive

**LFP** Local Field Potential

**MCBL** Multi-Cell Bolus Loading

**MVVM** Model View View-Model

**PSF** Point Spread Function

**PSTH** Post-Stimulus Time Histogram

**ROI** Region of Interest

**SD** Standard Deviation

**SEM** Scanning Electron Microscope

**SeNeCA** Search for Neuronal Cells Accelerated

**SNR** Signal to Noise Ratio

**TEM** Transmission Electron Microscope

**TPP** Two-Photon Processor

**TSP** Travelling Salesman Problem