Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

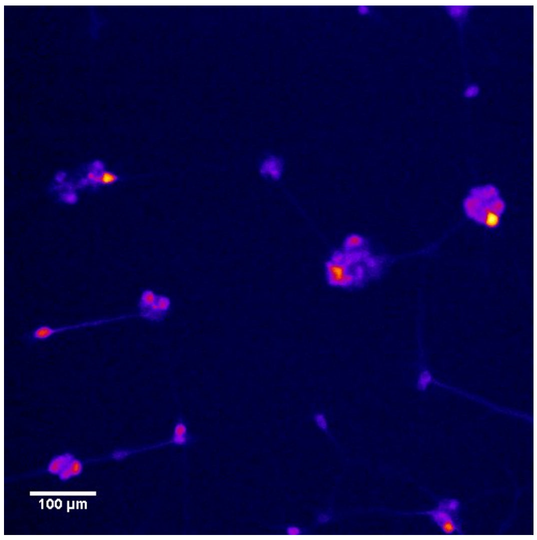
Laboratory of Biosensors and Bioelectronics

Department of Information Technology and Electrical Engineering

Spring Semester 2016

Automated Activity Analysis Of Neuronal Cultures

Semester Project



Conrad Burchert & Moritz Schneider
{bconrad, scmoritz}@student.ethz.ch

May 2016

Supervisor: Mathias Aebersold, aebersold@biomed.ee.ethz.ch
Professor: Prof. Dr. Janos Voeroes, janos.voros@biomed.ee.ethz.ch

Acknowledgements

We would like to thank our supervisor Mathias Abersold for his support in this project. He was always enthusiastic, brought new ideas and created a productive environment. Also big thanks to Adriane Joutang and Greta Thompson-Steckel for the valuable input on missing features and the feedback on our presentation. When we started the project both of us were new to the field of neurobiology, but we were welcomed by the great atmosphere at Laboratory for Biosensors and Bioelectronics (LBB). While we worked on this project, Mathias showed us the cultivation of neural networks *in vitro* and the gathering of data using calcium imaging. In addition we would like to thank all the other members of LBB, who showed us their work and helped us in the main and cell laboratories.

Abstract

Memory storage in the brain, one of the most basic mechanisms, is not yet fully understood. The bottom-up approach of analyzing only a small neural culture with a well-defined structure is a promising approach. Well documented procedures for *in vitro* engineering artificial neural networks allow for analysis of neural cultures with a controlled topology. Calcium imaging can be used to extract the neural activity of the cells in an unobtrusive way. Specialized software has to be used to extract the activity out of video recordings of the experiments. Multiple challenges from image processing and signal analysis appear in the analysis of neural cultures. In this project a solution for the supervised or unsupervised analysis of calcium imaging data has been developed, using current state of the art algorithms from signal and image processing.

Declaration of Originality

We hereby confirm that we are the sole authors of the written work here enclosed and that we have compiled it in our own words. Parts excepted are corrections of form and content by the supervisor. For a detailed version of the declaration of originality, please refer to Appendix B

Conrad Burchert & Moritz Schneider,
Zurich, May 2016

Contents

List of Figures

List of Tables

List of Acronyms

GUI	graphical user interface
LBB	Laboratory for Biosensors and Bioelectronics
ROI	region of interest

Chapter 1

Introduction

The fast growing field of neuroscience tries, among others, to better understand the brain. *In vitro* neuronal cultures are used to investigate basic mechanisms and recent developments in chemical surface patterning enable the creation of engineered neuronal networks of a defined structure[1]. Understanding artificial neuronal networks with a chosen topology might allow for further advancements in the understanding of the brain. Calcium imaging is an unobtrusive way to measure the neuronal activity in such an engineered sample. A flourescent marker for calcium is used to show the activity of neurons under fluoresence microscopy, e.g. the gene encoded calcium indicator GCaMP6[2]. The flourescence of the cells is captured on video, showing the activity over time. An image gathered from a calcium imaging experiment is depicted in figure 1.1.

The analysis of calcium imaging data requires specialized software and manual processing. We used state of the art algorithms from the fields image and signal processing in order to implement a python library that supports unsupervised and supervised analysis of calcium imaging data.

1.1. Using the Library

This section describes how the neuronal activity analyzer library can be used. The usual sequence of calls is described. This sequence is used in `analyzer/batch.py` and in the web based graphical user interface (GUI).

First the video is loaded using a loader. This loader provides access to the video's frames as numpy arrays and metadata: the frame rate and the resolution in pixel per μm.

```
import analyzer
loader = analyzer.open_video("some-video.tif")
```

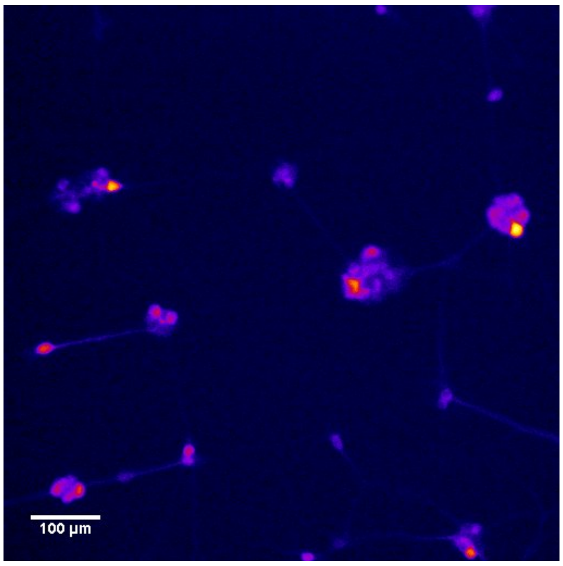


Figure 1.1.: Hippocampus cells on patterned glass, grid topology. Activity recorded with the calcium indicator GCaMP6s. 14 days *in vitro*

Afterwards the video is segmented with the function

```
segmentation = analyzer.segment(loader, config)
```

where **config** is the configration dictionary, that holds all the parameters for all the steps performed. This configuration file is described in more details in section 3.1.1. The segmentation returns a dictionary that contains all intermediate steps: source image, filtered image, thresholded image, segmented image and the borders for the segmented image. These are used in the GUI to display the intermediate results. The segmented image is a numpy array with the same shape as a frame. The value at every pixel stands for the neuron to which this pixel belongs. The segmented image, the loader and the configuration are then used to iterate over the complete video file and calculate the activities of the neurons:

```
activities = analyzer.calculate_activities(
    loader, segmentation['segmented'], config)
```

The return value **activities** is an array containing the activity for every neuron detected in the segmentation. Normally this data should now be normalized:

```
norm_activities = normalize(activities)
```

The activities can then be used for further analysis. To search for spikes use:

```
spikes = analyzer.detect_spikes(norm_activities, config, loader.exposure_time)
```

spikes is a list of the frames at which a spike occurs for every neuron.

Another possibility is to extract correlation functions. Note that this only works on normalized data:

```
correlations = correlate_activities(norm_activities, config, loader.exposure_time)
```

Now **correlations[i,j]** will contain the correlation function of neuron i and neuron j.

Finally all the generated results are saved and plotted with

```
analyzer.save_results(roi, frame, pixel_per_um, time_frame,
                      activities, spikes, 1, videoname, analysis_folder)
```

Chapter 2

Theory

This chapter describes the theory of the algorithms used in this project.

2.1. Thresholding

This section describes the various thresholding algorithms used throughout this semester project. A comparison of all these algorithms is beyond the scope of this project and can be seen in [3]. A thresholding algorithm calculates a threshold value T, that splits all pixels of a grey scale image into two groups.

$$g(x,y) = \begin{cases} 1 & f(x,y) > T \\ 0 & \text{otherwise} \end{cases}$$

Let $n_1, n_2, n_3, ...n_L$ be the number of pixels at each intensity layer $[1, 2, 3, ..., L]$ and $N = n_1 + n_2 + ... + n_L$ the total number of pixels. The histogram of the image is defined as a normalized probability distribution

$$p_i = \frac{n_i}{N} \qquad , \sum_{i=1}^{L} p_i = 1.$$

2.1.1. Otsu Thresholding

One of the most famous thresholding algorithms is the Otsu thresholding algorithm, named after his inventor N. Otsu [4].

Otsu proposed that maximizing the inter-class variance of both groups of pixels results in a good separation of the object and the background. The inter-class variance of the two classes $\{0, 1\}$ is

$$\sigma^2_{inter} = n_0(T)\sigma^2_0(T) + n_1(T)\sigma^2_1(T)$$

where $n_{0,1} = \sum_{i=0,T}^{T-1,N-1} p(i)$ and $\sigma^2_{0,1}$ = variance of cluster 0,1. Calculating the inter-class variance is computationally hard but Otsu showed that maximizing the inter-class variance is the same as minimizing $\sigma^2 - \sigma^2_{inter}$. This term is called intra-class variance and is easily calculated with $\mu_{0,1}(T)$ as the mean of each cluster:

$$\sigma^2_{intra} = \sigma^2 - \sigma^2_{inter} = n_0(T)n_1(T)(\mu_0(T) + \mu_1(T))^2$$

2.1.2. Li's Minimum Cross Entropy

Another unsupervised algorithm to find a threshold is called Li's Minimum Cross Entropy method [5]. It defines the threshold at the value, where the cross entropy is minimized.

The cross entropy is defined as

$$\eta(T) = \sum_{i=0}^{T-1} p_i \log\left(\frac{p_i}{\mu_0(T)}\right) + \sum_{i=T}^{L-1} p_i \log\left(\frac{p_i}{\mu_1(T)}\right)$$

and the resulting threshold T is

$$T = \min_T(\eta(T)).$$

2.1.3. Yen Thresholding

Yen thresholding algorithm is based on the maximum correlation contributed by the object and the background [6]. The correlation of a distribution X is defined as

$$C(X) = -\log \sum_{i \geq 0} p_i^2.$$

The combined correlation is thus

$$TC(T) = C_A(T) + C_B(T) = -\log \sum_{i=0}^{T-1} \left(\frac{p_i}{P(T)}\right)^2 - \log \sum_{i=T}^{L-1} \left(\frac{p_i}{1 - P(T)}\right)^2$$

with $P(s)$ as the cumulative histogram: $P(s) = \sum_{i<T} p_i$

2.2. Segmentation

After the image has been thresholded a separation into foreground and background is available. Now, region of interest (ROI) have to be labeled as neurons or groups of neurons, whose activities will be extracted later. The goal of this step is to create an assignment of every pixel to a ROI or to the background. Results of the different segmentation algorithms are visible in figure 2.1.

2.2.1. Labeling

The trivial approach to segmentation is to just label each continuous area after thresholding as one ROI. This is very similar to using a flood fill tool in an image editing program. While this does not separate neurons near each other it is sufficient for some situations. One such situation is if the total number of spikes over time is of interest, and it is therefore not necessary to look at individual neurons.

2.2.2. Watershed

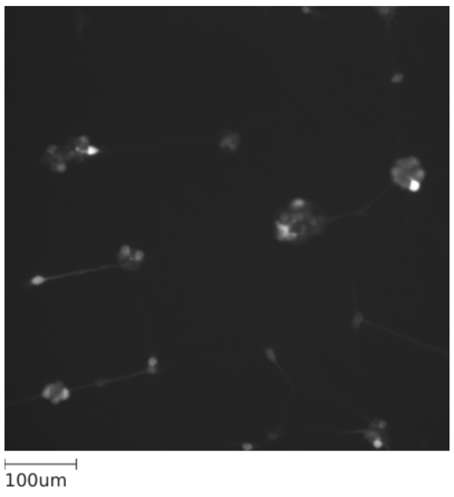
The watershed algorithm is based on the idea of water following gradients[7]. A simple application for the watershed algorithm would be to label mountains. If the mountains each had a water source at the top, the water would flow down the mountains over all surfaces. Boundaries between mountains are made where the water drops from different mountain tops meet.

Another way to look at the same principle is to name valleys instead of mountains. In figure 2.2 the algorithm can be seen applied to such a scenario in one dimension. Starting from the local minima the waterline is raised and everytime two regions meet a boundary is generated there.

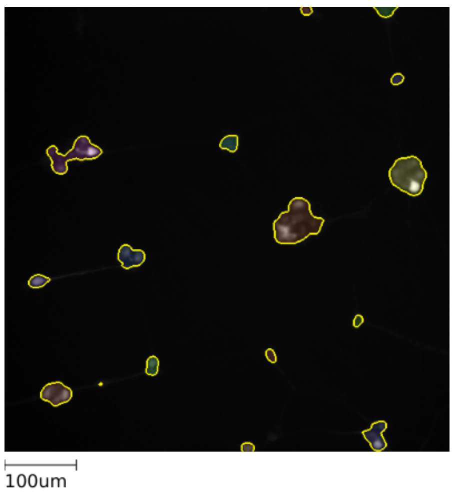
To apply the watershed algorithm to the thresholded image of neurons, the binary valued image needs to be translated to a heightmap. This is accomplished by assigning each pixel the distance to the background as a value. A narrowing in the foreground area will therefore create lower values in the heightmap and create a boundary under watershed. This can be seen in figure 2.3.

2.2.3. Random Walker

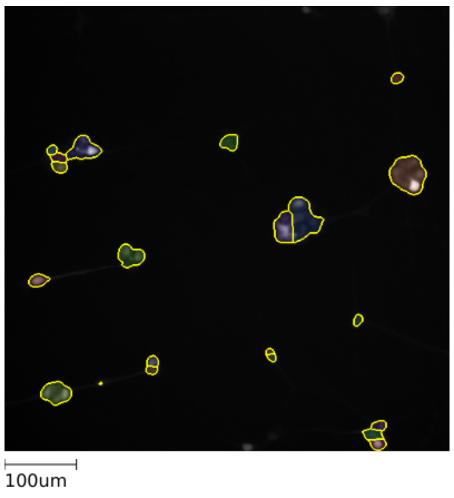
Another method to segment images is the random walker algorithm[9]. Starting at local maxima, random walkers move from pixel to pixel, where transition probabilities depend on the difference in brightness of neighbouring pixels. The probability to move to a neighbouring pixel is chosen higher, if the pixels are similar. Each pixel is then labeled to belong to the region started at that maximum, where a random walker is most likely to



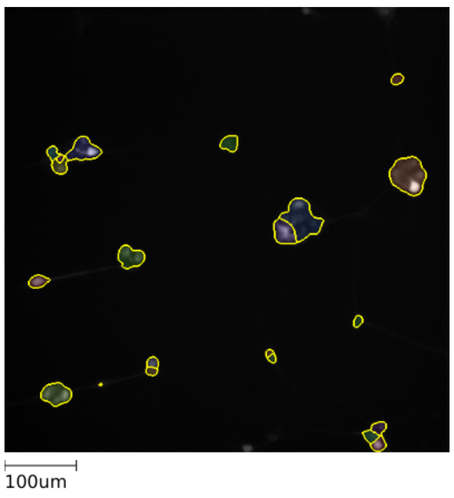
(a) Source image



(b) Segmented using labeling of continuous areas



(c) Segmented using watershed algorithm



(d) Segmented using random walker algorithm

Figure 2.1.: Comparison of the different segmentation algorithms. All used the mean of all frames as a source image, were convolved with a gauss filter($\sigma = 2.0$px) and thresholded using li threshold

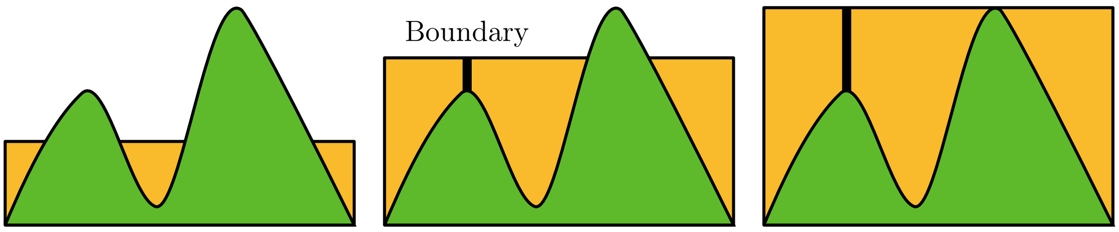


Figure 2.2.: Watershed used to name valleys in one dimension. Regions are generated around local minima, the waterline is raised until two regions met, at that location a boundary is generated. Note that in more than one dimension those do not need to be local maxima.

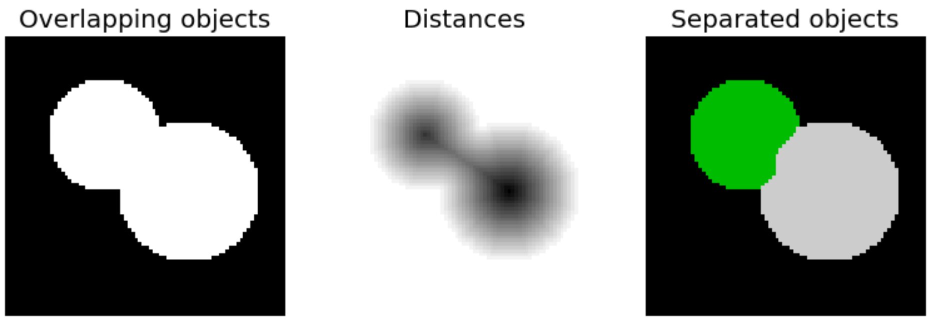


Figure 2.3.: Two overlapping circles converted to a heightmap using the distance to the background as pixel values, and segmented using watershed. Image from scikit documentation [8]. Copyright the scikit-image development team.

have departed from, when it reaches the pixel. Mathematically, the image is interpreted as a graph, where each pixel is a node. Each node is connected to its neighbouring pixels and the weight of the edge is determined based on the difference in brightness. It is common to choose a weight based on the formula:

$$w_{ij} = \exp(-\beta(g_i - g_j)^2)$$

Here β is some constant and g_i is the brightness of the pixel i. On this graph random walkers start at the local maximas. The random walker with the highest probability to arrive at a pixel assigns its origin as the region the pixel belongs to.

To apply the algorithm to the image obtained from thresholding, the same distance to background metric as in the watershed algorithm is used.

2.3. Bleaching

Calcium imaging exposes the sample to photobleaching. The measured brightness of neurons is continuously dropping. To correct the bleaching a second order polynomial is fitted to the measured activity of each neuron, approximating the effect. The approximation is then subtracted from the measured activity data.

2.4. Spike Detection

Researchers are mostly interested in the spikes of the neuronal activity because neurons communicate with electrical impulses. Therefore spike detection plays an important role in analysis of neuronal cultures.

2.4.1. Amplitude Threshold

Amplitude threshold is a trivial approach to the problem. A threshold value relative to the standard deviation has to be set and all parts of the signal above this value are spikes. An example is depicted in figure 2.4. This approach is very robust, it rarely detects false positives but the threshold has to be set by the user. If one wants to manually analyze a neuron this algorithm is very effective, but it cannot be used for unsupervised operation.

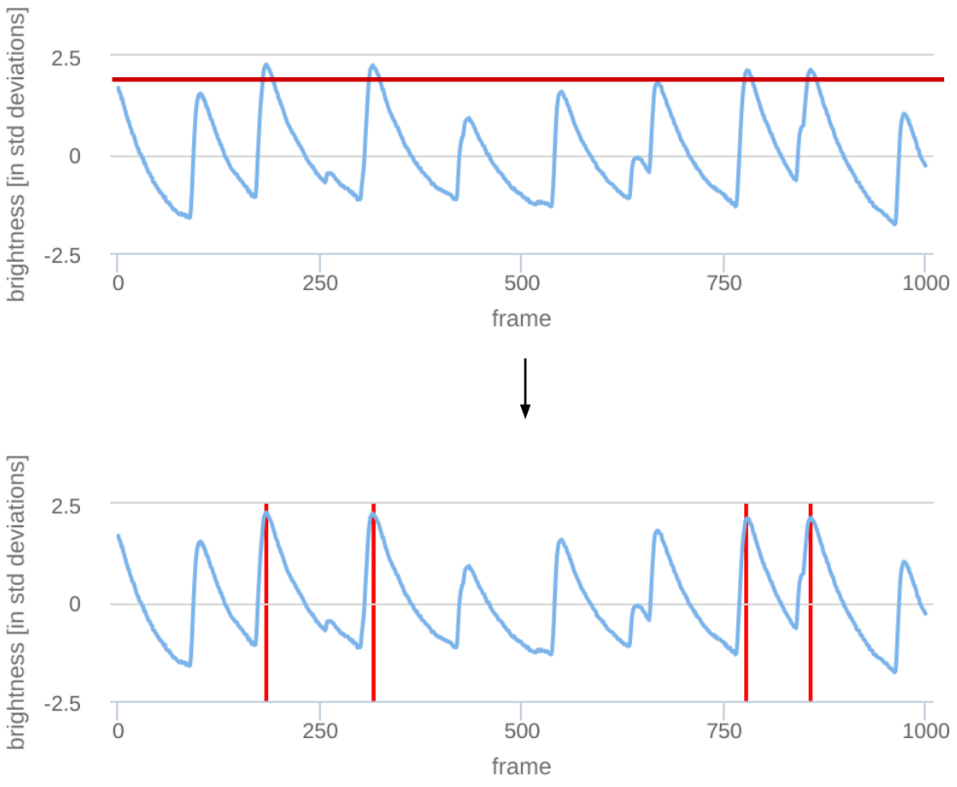


Figure 2.4.: Example: Amplitude threshold. With a parameter n, a threshold is used at the height of n times the standard deviation of the signal. Everything above the threshold is considered a spike and gets labeled as such at its local maximum. The problem is to choose the correct value for n. In this example a parameter of 2 was chosen, which is too high for this signal.

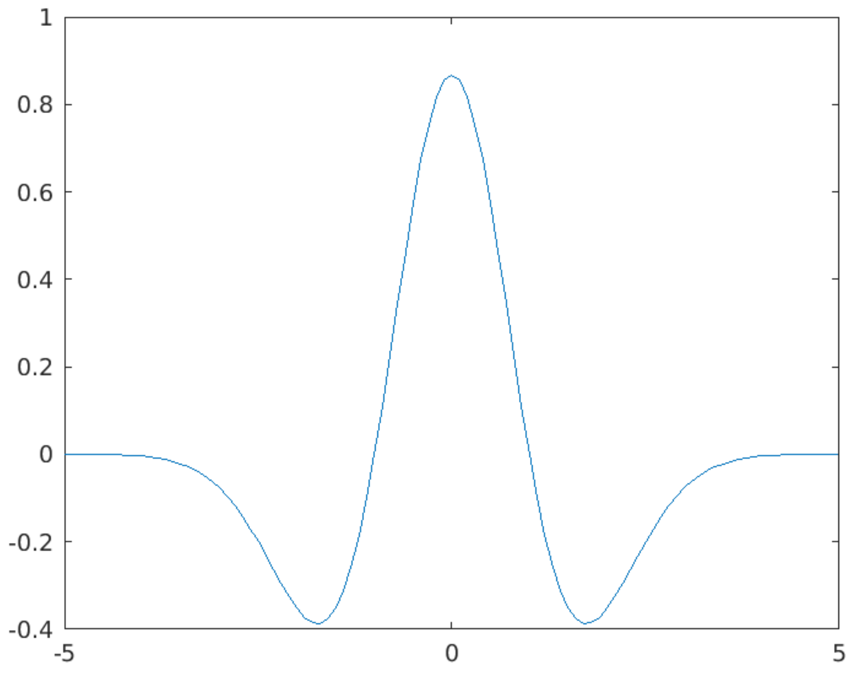


Figure 2.5.: Mexican hat wavelet, a spike is assumed to look similar to this function

2.4.2. Wavelet Spike Detection

Wavelet spike detection tries to detect spikes not by looking at the amplitude but by shape. This method is based on wavelet transformation [10].

A wavelet Ψ is a signal of finite energy and if a wavelet is normalized to $\|\Psi\| = 1$ and centered to the origin it is called mother wavelet. A mother wavelet describes a whole family of wavelets with scale a and translation b:

$$\Psi_{a,b} = \frac{1}{\sqrt{a}} \Psi_{mother} \left(\frac{t-b}{a} \right).$$

The mother wavelet that is used in this project is called "mexican hat wavelet" and depicted in figure 2.5.

Wavelet transformation is used to find the biggest correlation of the investigated signal to the wavelet. For example: If one transforms a music track with a wavelet that looks like a specific tone, then the occurrences of this tone can be calculated. In our case this is used to find spikes. The wavelet transformation is defined as the inner product $\langle , \rangle$

with a wavelet for all spike widths $k \in [a_0, a_1, ..., a_n]$ at every time j:

$$X(j,k) = \langle x, \Psi_{j,k} \rangle.$$

The spike width is not known in advance, therefore a range of equally spaced spike widths $[a_0, a_1, ..., a_n]$ is used. This results in a matrix that represents the correlation with the wavelet. The bigger the value the better the signal correlates with the wavelet at this wavelet width and time.

Afterwards spikes are detected using a threshold that is calculated according to [10].

Chapter 3

Implementation

The implementation was split into two parts, a python library containing all the functionality to analyze calcium imaging videos and a web based GUI to allow easy usage without installation or knowledge about the library. The library is structured to allow easy extension for more algorithms or file formats.

3.1. Basic Usage

3.1.1. Configuration Files

There are two configuration files present in this project. One describes settings of one video and is in the same folder as the video. The other sets parameters for the thresholding, segmentation, spike detection algorithms and further analysis steps and can be used for more than one video file.

Algorithm Parameters

All algorithm parameters used for analysis are stored as a dictionary in a configuration file called `analyzer/config.py`. The default parameters are set in `analyzer/settings.py`, which also contains other hardcoded settings of the library. This file is shown in listing 1. An overview of all possible values in the configuration file is depicted in table 3.1.

```
default_config = {
    'segmentation_source': 'mean',  # name of implementation
    'gauss_radius': 2.,             # sigma for gauss filter in pixels
    'threshold': 'li',              # float from 0 to 1 or name of algorithm
    'segmentation_algorithm': 'watershed',  # name of implementation
    'integrater': 'mean',           # name of implementation
    'spike_detection_algorithm': 'wdm',  # name of implementation
    'min_spike_width': 0.3,         # [s]
    'max_spike_width': 4,           # [s]
    'correlation_max_shift': 5.,    # [s]
    'nSD_n': 3.,                    # float, threshold value in standard
                                    # deviations for ntimesstd implementation
                                    # spike detection
}
```

Listing 1: The default configuration. Excerpt from `analyzer/settings.py`

Table 3.1.: All possible values for the configuration file

Key	**Possible values**
segmentation_source	'first_frame', 'mean', 'variance'
gauss_radius	float
threshold[†]	'otsu', 'li', 'yen', float $\in [0, 1]$
segmentation_algorithm[†]	'label', 'watershed', 'randomwalker'
integrater[†]	'mean'
spike_detetction_algorithm[†]	'ntimesstd', 'wdm'
min_spike_width	float
max_spike_width	float
correlation_max_shift	float
nSD_n	float

[†] Additional implementations for these algorithms can be added with the module system from section 3.3

```
FrameRate = 31.9
PixelPerUM = 0.6466
```

Listing 2: Example structure of a video specific configuration file

Video specific Configuration

The video specific configuration file is located in the same folder as the video and is named after the video file name with an added file extension: *.txt*. For example if the video is named *video.tif* then the configuration file has to be named *video.tif.txt*. This configuration file is searched for lines where the frame rate and the resolution in pixel per μm are set. These lines have to be formatted as shown in listing 2. There can be more information in this file, but if this file does not exist or does not specify the frame rate or the resolution, they are set to the default value. The default values are set to 31.9 fps for the frame rate and 0.6466 for the resolution, which are typical values for videos obtained in this lab.

3.2. Overview of the library files

```
analyzer
 ├── integrater ................................ integrater implementations
 │    ├── integrater.py
 │    └── mean.py
 ├── loader ........................................ loader implementations
 │    ├── loader.py
 │    ├── bioformat_loader.py
 │    └── pillow_loader.py
 ├── segmentation ............................... segmentation algorithms
 │    ├── label.py
 │    ├── randomwalker.py
 │    ├── segmentation.py
 │    └── watershed.py
 ├── spike_detection ........................... spike detection algorithms
 │    ├── ntimesstd.py
 │    ├── spike_detection.py
 │    └── wdm.py
 ├── threshold.....................................thresholding algorithms
 │    ├── li.py
 │    ├── otsu.py
 │    ├── theshold.py
 │    └── yen.py
 ├── __init__.py ............................ main interface of the library
 └── batch.py ..........................functions to invoke a batch process
```

```
│
├── correlation.py ........ calculates correlations between neuron activities
├── normalize.py . Normalizes activity data based on mean and std deviation
├── plot.py .................................. Functions to plot the results
├── roi_reader.py .................. Importer and exporter for ImageJ ROIs
├── settings.py .................. Default parameters and various constants
└── util.py ............................................. Utility functions
```

3.3. Adding modules to implement new algorithms

The analyzer package has a main file `__init__.py`. Most of the functions available there receive a parameter **config**. This parameter holds the dictionary of settings for the current analysis run. The functions in `__init__.py` delegate the work to a specific implementation based on a parameter in the **config**. For example the config value **segmentation_algorithm** chooses which implementation will be used to label the thresholded image with ROIs.

To add new algorithms it is sufficient to create a new module in the corresponding folder. The algorithm is then automatically available and can be selected with the config parameter. To add a new implementation follow these steps:

1. Choose a name for your implementation
2. Create a module <name>.py in the folder for the step you want to reimplement
3. In that module implement a class with the same name, which subclasses the correct baseclass
4. Use the method in your analysis run by setting the correct config parameter to the name of the class

For example, to add a new implementation called "derivative" for spike detection, create a new file `derivative.py` in `analyzer/spike_detection`, which implements a class inheriting from the base class **SpikeDetection** and implement the functions **__init__** and **detect_spikes**, as shown in listing 3.

The process is very similar for thresholding and segmentation algorithms, and methods to generate activity data from ROIs with the base classes **Threshold** in `analyzer/threshold/threshold.py`, **SpikeDetection** in `analyzer/spike_detection/spike_detection.py` and **Integrater** in `analyzer/intergrater/integrater.py` respectively.

3.4. File Loader

All file loaders are in the folder `analyzer/loader` and implement the base class `loader.py`. The base class finds subclasses of itself in **find_loader_class**. Afterwards all

```python
""" Some documentation describing your algorithm. This will be available
to users of the library """

from .spike_detection import SpikeDetection


class Derivative(SpikeDetection):
    """ A maybe not so good algorithm to detect spikes. It might work
    without noise """

    def __init__(self, config, exposure_time):
        """ A variable from the config is loaded, which can be used for
        all activity measurements. Every object of this class will be
        used to analyse one video file. The config parameter is a
        dictionary containing all settings for the analysis.
        """
        self.min_derivative_for_spike = config['min_derivative_for_spike']

    def detect_spikes(self, dataset):
        spikes = []

        last_val = dataset[0]
        for i in range(1, len(dataset)):
            if dataset[i] - last_val > self.min_derivative_for_spike:
                spikes.append(i)
            last_val = dataset[i]

        return spikes
```

Listing 3: A new spike detection algorithm `analyzer/spike_detection/derivative.py`

supported video files can be opened with the function **open** of the base class. The **open** function calls the **can_open** function for every subclass until one of them is able to open the given file. If no subclass is able to open the given file, an exception is thrown.

3.5. Segmentation and Thresholds

Implementations of the segmentation algorithms discussed in section 2.2 are in the folder `analyzer/segmentation`. They implement a base class called `segmentation.py`. This class is not used anywhere, it's only purpose is to make it easier to implement new algorithms.

Thresholding algorithms follow the same file structure as the segmentation algorithms. They can be found in `analyzer/threshold` and implement a base class called `threshold.py`. In this case the base class is implementing a few static methods used to convert from percentage thresholds to absolute ones.

All mentioned segmentation and thresholding algorithms are already implemented in the library called scikit-image[11], which is used in this project. A beta value (see section 2.2.3) of 10 was used for the random walker segmentation algorithm.

3.6. Normalization

Activity data is prepared for further analysis in the **normalize** function in `analyzer/normalize.py`. It first corrects the photobleaching as described in section 2.3. Afterwards the data is shifted to have zero mean and normalized to its standard deviation.

3.7. Spike Detection

Implementations of the two algorithms described in section 2.4 exist in `analyzer/spike_detection/`. They implement a base class called **SpikeDetection**. The constructor of **SpikeDetection** expects two variables: the configuration as in section 3.1.1 and the exposure time. Some algorithms need the exposure time to take a guess at the width of a spike, for example the wavelet spike detection as described in section 2.4. All spike detection algorithms are automatically parallelized when using the **detect_spikes_parallel** in the base class. Spike detection algorithms can be run in parallel by definition because they run independently for every neuron.

```
def _wavelet_transform(self, dataset, gen_wavelet):
    wt = numpy.zeros((self.steps, len(dataset)))
    for j in range(0, self.steps):
        # equally spaces steps for the wavelet widths
        width = self.min_spike_width + j*self.step_size
        wavelet = gen_wavelet(width)
        conv = numpy.correlate(dataset, wavelet, 'same')
        wt[j] = conv
    return wt
```

Listing 4: Wavelet transformation in `analyzer/spike_detection/wdm.py`

3.7.1. Wavelet Spike Detection

The implementation for the wavelet spike detection algorithm is in `analyzer/spike_detection/wdm.py`. One part of this file, the wavelet transform, is shown in listing 4. Wavelet transformation correlates wavelets of different widths with the signal. In code this is implemented as a loop over all possible wavelet widths. These wavelet widths are equally spaced between the minimal and maximal spike width. Afterwards a wavelet is generated with the given wavelet generator. This wavelet is then correlated with the signal and stored in the numpy array **wt**.

For the estimation of the possible widths of the spikes a time range from 0.3 s to 4.0 s is used. There are signals where changing this range yields much better results and changing it is possible in the configuration file from section 3.1.1.

3.8. Correlations

The correlation functions of the activity of neurons are calculated for all pairs of two neurons. This is implemented using the numpy correlate function. Calculation of the complete correlation functions, including all possible time shifts, however creates a high workload. The number of pairs is in the order of $O(n_{neurons}^2)$, the convolution operation to calculate the correlation for each time shift uses $O(n_{shifts})$ multiply and accumulate operations and is executed $O(n_{shifts})$ times. This leads to a total complexity of $O(n_{neurons}^2 n_{shifts}^2)$.

As the number of neurons is typically much smaller than the number of frames in a video, it is interesting to reduce the second term. Correlations with a time shift larger than a few seconds are not interesting, as they do not yield information about the neuronal network's connectivity. Therefore a setting in the analysis configuration called `correlation_max_shift` is intrudoced to limit the number of correlations calculated per pair of neurons. This reduced second term of the complexity to a linear term.

3.9. Web Gui

The web interface consists of a javascript part, that is responsible for displaying the results and a python part, that creates an interface to the analyzer library. In the javascript part the following libraries are used: jquery[1], mpld3[2], d3[3], bootstrap[4], bootstrap-treeview[5] and highcharts[6]. In python the well known flask library[7] is used to communicate with the browser. The data, that is sent from python, is encoded in json[8] and compressed using flask-compress[9].

Apart from creating an interface between the analyzer library and the javascript interface in the browser, the server stores all parameters and intermediate results. To do this a session management system was created. As the name 'session' is already reserved in javascript and flask, such an analysis session is called a run. A run works as a persistent server side storage, implemented as the class **Run** based on a python shelf[10], and is identified by the path of the analysed video file and a name(as opposed to a session in other web services, which is per user). As the webserver can work on many requests in parallel the run needs to be careful about concurrent acccesses to the same data. This is solved using a global lock, a static member of the **Run** class. A consequence is, that functions working on runs need to try to hold a **Run** object as short as possible to avoid delays in other parallel requests.

[1] `https://jquery.com/`
[2] `https://mpld3.github.io/`
[3] `https://d3js.org/`
[4] `https://getbootstrap.com/`
[5] `https://github.com/jonmiles/bootstrap-treeview`
[6] `https://www.highcharts.com/`
[7] `http://flask.pocoo.org/docs/0.10/`
[8] `http://www.json.org/`
[9] `https://flask-compress.readthedocs.io/en/latest/`
[10] `https://docs.python.org/3/library/shelve.html`

Chapter 4

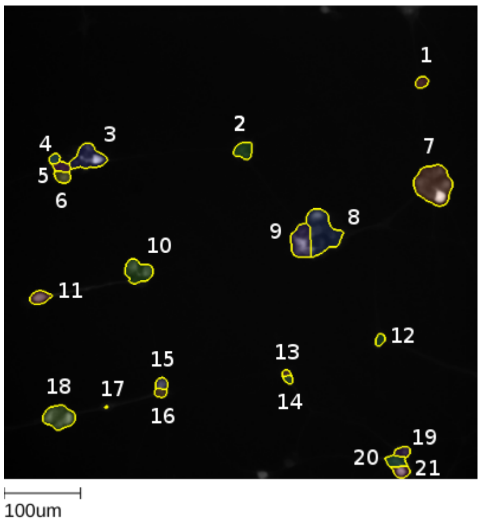
Results

A video containing data collected from a calcium imaging experiment, in which neurons were cultured in a grid structure, was analyzed. The results are visible in figure 4.1. The rasterplot (figure 4.1b) of the spikes shows that activity seems to appear in clusters: multiple neurons have spikes at the same time.

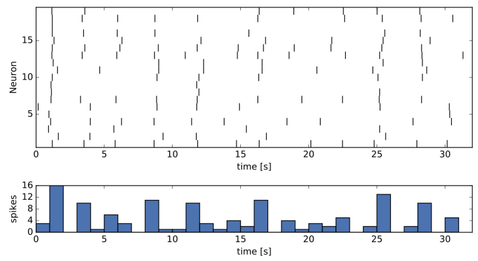
A closer examination of the relationships between neurons can be utilized by the correlations. The correlation matrix in figure 4.1e shows an overview of the maxima of the correlation functions of all pairs of neurons. While the interpretation of this data is not the focus of this work and needs further investigation an outline of possible findings has been created:

It can be seen, that there is a group of neurons 3, 4, 5 and 6, which seem to have a positive maximum of correlation with each other but a negative with most other neurons. This might indicate that these neurons are tightly connected and have inhibitory connections to other neurons. From the segmentation in figure 4.1a it is visible that the neurons directly neighbour each other, which could support this theory.

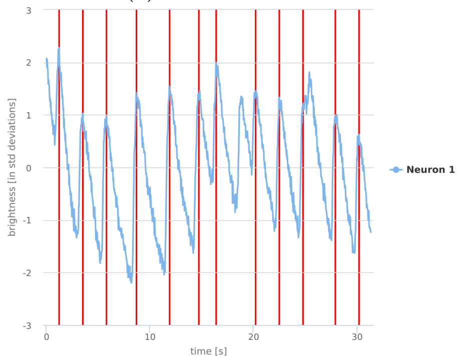
One can also examine the correlation plot of two of those neurons (figure 4.1f) where the time delay is on the x axis and the correlation on the y axis. In this example the maximal correlation is at about a delay of -200 ms from neuron 3 to neuron 4. This might suggest, that there is an axon from neuron 4 to neuron 3. However it can be seen, that there are more local maxima in the correlation function, that have no clear role. These might be related to indirect other connections, possibly involving cycles. Any final interpretation of the correlation data should however make clear what the role of those is.



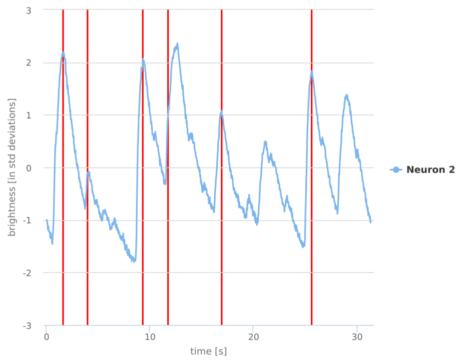
(a) Labeled neurons



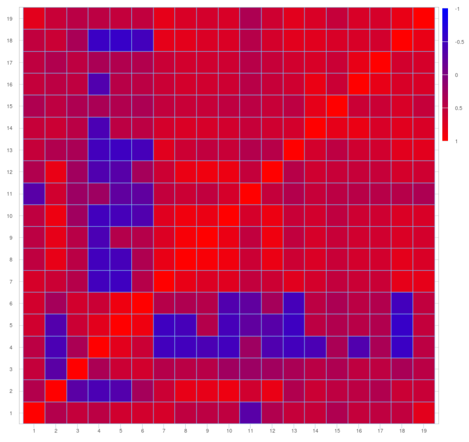
(b) Rasterplot of spikes



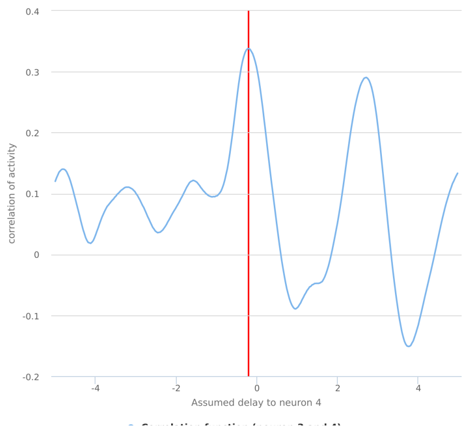
(c) Activity of neuron 1



(d) Activity of neuron 2



(e) Correlation matrix, maxima of the correlation functions



(f) Correlation function of neuron 3 and neuron 4

Figure 4.1.: Analysis of the clusters on grid video

Chapter 5

Conclusion and Future Work

5.1. Conclusion

An easy to use tool was created to analyze calcium imaging data. The tool has a web based GUI, which makes it easy to use without local installation, even for unexperienced users. Extensions for new algorithms can be added by dropping an appropriate implementation into the correct folder. The output consists of activity data in csv format and plots, detected spikes, also in csv format and plots, summmaries of the number of spikes and correlation functions for all pairs of neurons.

The application was installed on a webspace provided by the ISG.EE of ETH Zurich and is available on `neurons.ee.ethz.ch`. The extracted correlation functions of the videos we analyzed seem to provide information about the network connectivity.

5.2. Future Work

The extracted correlation data could be used to construct a graph of the network. However as indirect links also appear in this graph, it will need further reduction to be useful.

It is possible to construct a segmentation algorithm based on correlations using the fact, that pixels belonging to the same neuron will have a higher correlation over the length of the video. The difficult challenge to overcome here is, that this segmentation approach needs to access all data of the video to generate the segmentation, which is computationally expensive.

Using more wavelets in the wavelet spike detection could lead to characterization of spikes because they correlate more with a better fitting wavelet. Wavelets that are similar to all types of spikes and a method to select the best fitting wavelet for each spike could be found. This would allow a more precise timing of the spikes, which could be used in characterisation of connections between neurons.

Appendix A

Task Description

University of Zurich^UZH
ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Biomedical Engineering
Laboratory of Biosensors and Bioelectronics
Prof. Janos Vörös
ETH Zentrum, ETZ F82
Gloriastrasse 35
CH-8092 Zürich
Tel. +41 44 632 59 03
Secr. +41 44 632 45 85
Fax. +41 44 632 11 93
janos.voros@biomed.ee.ethz.ch
www.lbb.ethz.ch

Project Plan

Family Name: Buchert First Name(s): Condrad
E-Mail ETH student: bconrad@student.ethz.ch
Matriculation No.: 12-937-132

Family Name: Schneider First Name(s): Moritz
E-Mail ETH student: scmoritz@student.ethz.ch
Matriculation No.: 11-929-288

Semester Thesis
Department/Institution: ETH D-ITET

Title: Automated activity analysis of neuronal cultures

Start of Project: 22.02.2016
End of Project: 03.06.2016

Project/Thesis Supervisor IBT/LBB: Mathias Aebersold
Project/Thesis responsible Professor IBT/LBB: Prof. Janos Vörös

1. INTRODUCTION

Neurons can be cultured *in vitro* to investigate how they communicate with each other. This bottom-up approach makes it possible to investigate small networks of neurons in a controlled and reproducible manner[1], [2].
In the nervous system, information is transmitted from neuron to neurons as electrochemical waves. As information is passed along the network, the membrane potential of the involved cells is modulated and can be measured optically using fluorescent calcium indicator proteins[3]–[5]
Imaging networks of neurons with high speed cameras is becoming a routine task in many labs. However, the analysis of such data is mostly done using custom scripts. Researches often find themselves with more data than they have the means to analyse. However, there is a trend towards freely available open source tools[6], [7].
The aim of this project is to create a framework for the analysis of calcium imaging data with special attention to expandability and reusability.

2. TASK LIST

- Write a time-table of the work to be performed
- Review the literature relevant to methods of analysing neuronal activity.
- Evaluate potential methods to process calcium activity data.
- Choose and implement at the methods in a modular fashion.
- Learn how to culture networks *in vitro* and obtain calcium activity data.
- Perform a validation or verification framework.
- Present the results in a presentation to the group.
- Write a detailed report of the project.

3. LITERATURE

[1] M. J. Aebersold, H. Dermutz, C. Forró, S. Weydert, G. Thompson-Steckel, J. Vörös, and L. Demkó, "'Brains on a chip': towards engineered neural networks," *TrAC Trends Anal. Chem.*, Feb. 2016.

[2] H. Dermutz, R. R. Grüter, A. M. Truong, L. Demkó, J. Vörös, and T. Zambelli, "Local polymer replacement for neuron patterning and in situ neurite guidance.," *Langmuir*, vol. 30, no. 23, pp. 7037–46, Jun. 2014.

[3] C. Grienberger and A. Konnerth, "Imaging calcium in neurons.," *Neuron*, vol. 73, no. 5, pp. 862–85, Mar. 2012.

[4] T. Knöpfel, "Genetically encoded optical indicators for the analysis of neuronal circuits.," *Nat. Rev. Neurosci.*, vol. 13, no. 10, pp. 687–700, Oct. 2012.

[5] T.-W. Chen, T. J. Wardill, Y. Sun, S. R. Pulver, S. L. Renninger, A. Baohan, E. R. Schreiter, R. a Kerr, M. B. Orger, V. Jayaraman, L. L. Looger, K. Svoboda, and D. S. Kim, "Ultrasensitive fluorescent proteins for imaging neuronal activity," *Nature*, vol. 499, no. 7458, pp. 295–300, Jul. 2013.

[6] J. Freeman, "Open source tools for large-scale neuroscience," *Curr. Opin. Neurobiol.*, vol. 32, pp. 156–163, 2015.

[7] T. P. Patel, K. Man, B. L. Firestein, and D. F. Meaney, "Automated quantification of neuronal networks and single-cell calcium dynamics using calcium imaging," *J. Neurosci. Methods*, vol. 243, pp. 26–38, Mar. 2015.

Appendix B

Declaration of Originality

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Automated Activity Analysis of Neuronal Cultures

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Burchert	Conrad
Schneider	Moritz

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 30.05.2016	C. Burchert
	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Bibliography

[1] M. J. Aebersold, H. Dermutz, C. Forró, S. Weydert, G. Thompson-Steckel, J. Vörös, and L. Demkó, ““brains on a chip”: Towards engineered neural networks,” *TrAC Trends in Analytical Chemistry*, vol. 78, pp. 60–69, 2016.

[2] L. Tian, S. A. Hires, T. Mao, D. Huber, M. E. Chiappe, S. H. Chalasani, L. Petreanu, J. Akerboom, S. A. McKinney, E. R. Schreiter *et al.*, “Imaging neural activity in worms, flies and mice with improved gcamp calcium indicators,” *Nature methods*, vol. 6, no. 12, pp. 875–881, 2009.

[3] M. Sezgin, “Survey over image thresholding techniques and quantitative performance evaluation,” *Journal of Electronic imaging*, vol. 13, no. 1, pp. 146–168, 2004.

[4] N. Otsu, “A threshold selection method from gray-level histograms,” *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[5] C. H. Li and C. Lee, “Minimum cross entropy thresholding,” *Pattern Recognition*, vol. 26, no. 4, pp. 617–625, 1993.

[6] J.-C. Yen, F.-J. Chang, and S. Chang, “A new criterion for automatic multilevel thresholding,” *Image Processing, IEEE Transactions on*, vol. 4, no. 3, pp. 370–378, 1995.

[7] L. Vincent and P. Soille, “Watersheds in digital spaces: an efficient algorithm based on immersion simulations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583–598, Jun 1991.

[8] scikit-image development team, checked out at 2016-05-29. [Online]. Available: http://scikit-image.org/docs/dev/auto_examples/plot_watershed.html

[9] L. Grady, “Random walks for image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1768–1783, Nov 2006.

[10] Z. Nenadic and J. W. Burdick, “Spike detection using the continuous wavelet transform,” *Biomedical Engineering, IEEE Transactions on*, vol. 52, no. 1, pp. 74–87,

2005.

[11] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "scikit-image: image processing in Python," *PeerJ*, vol. 2, p. e453, 6 2014. [Online]. Available: http://dx.doi.org/10.7717/peerj.453