UNIVERSITÀ
DEGLI STUDI
FIRENZE

Computer Engineering

Bachelor Thesis

DETERMINATION OF RADIUS OF MOUSE BRAIN CELLS IN 3D IMAGES

DETERMINAZIONE DEL RAGGIO DI CELLULE CEREBRALI DI TOPO IN IMMAGINI 3D

VITTORIO ZAMPINETTI

Supervisor: *Paolo Frasconi*

Academic Year 2018-2019

Vittorio Zampinetti: *Determination of radius of mouse brain cells in 3D images,* BCMeasure, a tool for measuring of cells soma in confocal light sheet microscopy images, © Academic Year 2018-2019

SUPERVISOR:
Paolo Frasconi

LOCATION:
Firenze

TIME FRAME:
Academic Year 2018-2019

ABSTRACT

Mouse brain 3D images acquisition technologies such as confocal light sheet microscopy offer the possibility to create accurate digital maps of the whole set of cells in a brain. Manual annotation of all features is however a wasteful operation especially when dealing with image sizes in the TeraByte scale at the micron-resolution. A fully automated tool to determine the soma size of each annotated cell in such images would improve efficiency and accuracy in the mapping process. Clean digital images of mouse brains could be used as supervision in state-of-the-art machine learning algorithms which can lead to unlimited improvements in various bioinformatic challenges.

CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I'd like to thank professor Paolo Frasconi for the assistance and the supervision before and during the development of the project.
I wish to acknowledge the essential contribution provided by Dr. Ludovico Silvestri who is responsible for generating the materials (images and annotations) used for this project.
Special thanks also go to professor Thomas Boudier who kindly provided useful clarifications about his ImageJ 3D Suite library.

Last but not least, I would also like to express my gratitude to my family and friends who helped me get through this three-year journey.

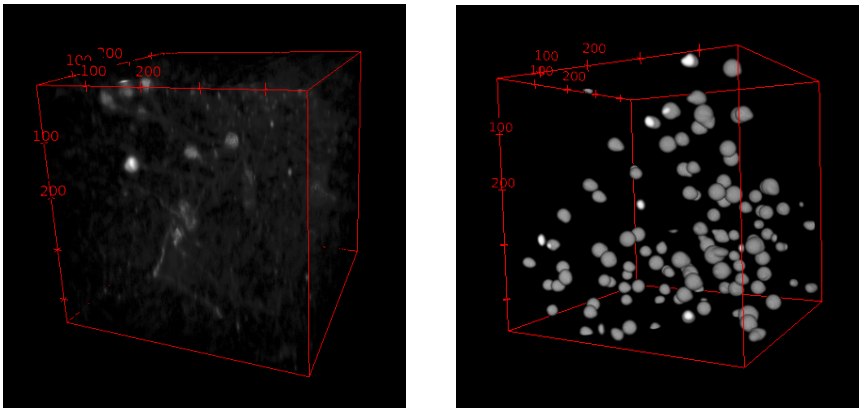
1 INTRODUCTION

1.1 OVERVIEW

Understanding the cytoarchitecture of the mammalian central nervous system on a brain-wide scale is becoming a compelling need in neuroscience. Recent image acquisition technologies make it possible to obtain high-quality snapshots of brain cells at the micron scale resolution. This poses the unprecedented challenge of creating accurate digital maps of the whole set of cells in a brain. Such maps would eventually allow characterizing on a structural basis the physiology and pathology of the central nervous system at various stages, ranging from development to neurodegeneration.

The three main optical approaches used to map mouse brain anatomy are micro-optical sectioning tomography (MOST), serial two-photon tomography (STP) and light sheet microscopy (LSM). The latter (in a particular implementation called Confocal LSM) is the method used to acquire the images which are subject-matter of this project (Silvestri et al. [5]). In this technique, however, different fixation efficiencies within the whole organ and inhomogeneous optical clearing give rise to a *large variability in contrast* throughout the entire volume. Because of this heterogeneity, naïve segmentation or localization methods (e.g. thresholding) cannot be applied to analyze whole-brain datasets obtained with CLSM.
Various 3D cell segmentation tools have already been developed (for example DiAna plugin for ImageJ Gilles et al. [2]) but don't fit these images well enough.

A tool for fully-automated cell localization in such images has already been developed by Frasconi et al. [1]. This, however, doesn't extract any information about the cells morphology. Features like shape and dimension would enrich the resulting digital map, producing an even more detailed representation.
BCMeasure tool takes advantage of previously annotated cell centers to determine the cell radius and therefore make it possible to reconstruct clean digital maps of the brain cells. Clean



(a) Original image (b) Reconstructed image

Figure 1.1: Reconstruction of image stack with determined radiuses in 3D renders

images (like the one in Figure 1.1) can than be used for various purposes.
For example they can be used as supervision for training in machine learning models like Autoencoders. Such algorithms could learn to automatically generate clean maps of brain cells from noisy and low-contrast images.
Moreover, statistics extracted from the results (such as radius length distribution over the volumes, mean shifting distance from the annotated point to the centroid) can give relevant information about the area of the brain being examined or the process reliability on that image. As a side-effect, the application of the mean-shift algorithm (Section 2.4) is useful for centering the manually annotated points which are probably not always accurate.

1.2 MATERIALS

The dataset provided for this project includes many 3D images each one accompanied by a CSV file containing the coordinates of the cells centers recognised and manually annotated.

1.2.1 *Images*

This project is based on the analysis and processing of 3D images obtained with CLSM[1]. The details about the acquisition of the

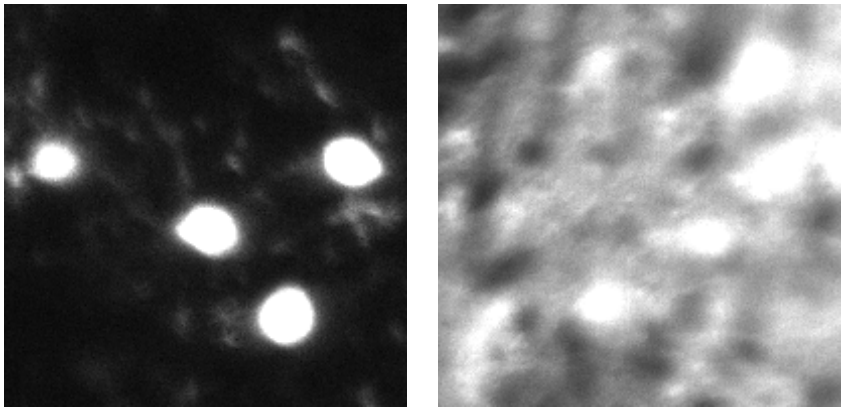
1 Confocal light sheet microscopy

images are presented in Silvestri et al. [5]. After the appropriate brain treatment, single-channel 16-bit TIFF files are produced by the CLSM (see Figure 1.1a for an example of a TIFF image rendered in 3D).

The used dataset comes sparsely from different areas of more than one mouse brain, imaged as clarified above, therefore the files have various sizes. Due to the acquisition technology, the stacks are anisotropic[2], in particular 3μm correspond to 5 pixels in the XY axes and 1.5 pixels in the Z axis. This detail must be considered throughout the process, in fact every operation involving the image depth (from substacking to intensities mean computation of spherical volumes) uses the proportion $\frac{resZ}{resXY}$ as a scale factor.

these TIFF files consist in many stacked 2D images, each one constituting a layer along the Z axis

As anticipated before, the real challenge, reason for the development of this tailored tool, is that the quality of the images varies a lot from one to the other, depending on the area of the brain to which they belong. Some contain well defined spherical-shaped soma cells, for which common global thresholding techniques would be enough, others are low-contrast confused pictures in which sometimes it is difficult to recognize cells even by looking at it (Figure 1.2).



(a) High quality (b) Low quality

Figure 1.2: Example of variability in image quality

1.2.2 *Markers*

The process is enormously facilitated by the availability of annotations of the soma positions, CSV files associated to each stack with the 3D coordinates of every cell recognised. These data are

2 Different resolution in depth with respect to width and height

manually produced by experts who have marked the center of each cell somata. Every 3D point should refer to the center of a cell, but obviously, due to accidental errors, it may not be precise as needed. Moreover, some images were found to be particularly difficult to understand: two independent human labelings on nine substacks disagree on 40 markers on 957, which is almost 5% of the cells.

Nevertheless, during the development of the presented tool the markers were always considered a ground-truth for the location of the cells.

2

METHODS

The process includes four main steps: cell substacking, adaptive thresholding around local max near the seed to find a first radius estimate, mean shift clustering using that estimate as look-distance to find an accurate centroid and again local mean thresholding combined with 3d radial distribution, but this time around the centroid just found, to determine the radius of the cell soma.

2.1 SUBSTACKING

Every annotated cell soma is extracted as a substack of the starting image. The dimensions of the substack are fixed so that the cell is entirely included. Since the visible region of the somata ranges approximately between 15 and 60 voxels of diameter, a cube of 70^3 voxels is big enough. Actually, due to the fact that the resolution along the Z axis is less than that of the XY axes, the substack dimensions are defined as shown below:

$$W \times H \times D, \text{ with } W = H = 70 \text{ and } D = 70\,S$$

with $S = \frac{resZ}{resXY}$ which is the scale of the Z axis with respect to XY axes.

I. e. $S = 1$ if the image is isotropic (which is not our case), otherwise when the resolution of the z axis is less than the resolution of the XY axes, S is between 0 and 1. In our images $S \approx 0.3$.

The substack is centered in the manually annotated 3D coordinates which, as stated in the introduction (**??**), are not guaranteed to be the centroid of the cell somata, therefore the cell might not be exactly centered in the mentioned volume. The centroid is found later with the mean shift algorithm, analysed in detail in Section 2.4.

From here on every operation described is applied locally on a cell substack. The main reason for this choice is that in a single image, cells can be very different from the others, both in intensity and shape, therefore global thresholding would give bad result since it may underestimate the size of some cells or

even exclude them. One of the most important components of the tool is, in fact, the `CellStack` class (see Section 3.3).

2.2 LOCAL MAX CENTER AND THRESHOLDING

First of all the current center of the cell has been shifted from the annotated coordinates to the local maximum (in 3D space) as a small correction. This temporary center seemed to be quite accurate with some images, but of course it could not be used as definitive center due to cases like that of the image in Figure 2.1. Hence the mean shift algorithm.

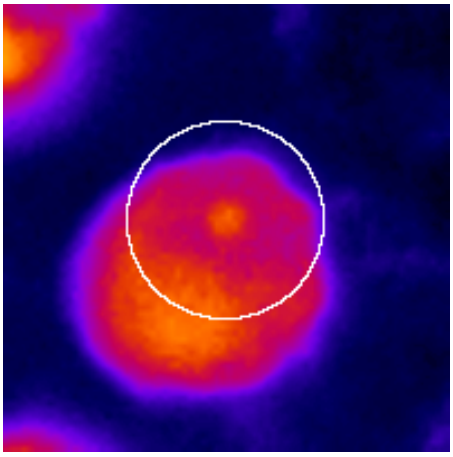


Figure 2.1: Example of wrong radius estimate with *local max* center

As said before, a global thresholding technique would not do the job. Thus, adaptive thresholding was applied for the computation of the intensity value used as a threshold that defines the extension of the object.
The formula used is that of a local mean value (Heck and Boudier [3]) which is computed as follows.
Three circles are drawn around the current center of the cell. The radius of each circle is initially fixed so that the first circle should be located within the object, while the two other circles should be located outside of the object.The mean intensities of the object (within the first circle), and of the background (in between the two other circles) are measured and the threshold calculated. By default, the threshold is almost the mean of the two mean intensity values ($weight = 0.4$). The user can however shift this parameter: for a weight value of 0.25, the threshold will be closer to the background value.

$$meanspot \cdot weight + (1 - weight) \cdot meanback$$

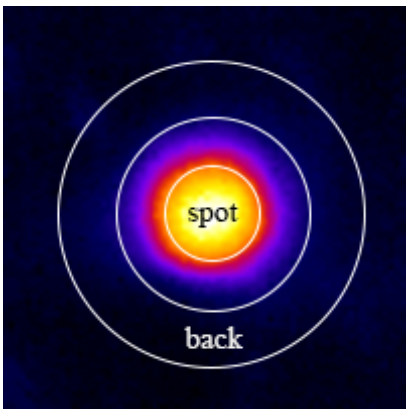


Figure 2.2: Visualization of the *local mean* formula

2.3 3D RADIAL DISTRIBUTION

The measurement of the radius at this point is quite easy (even if we don't have an appropriate center yet) since we can compute the mean of the intensities of the voxels inside each sphere cap of unitary thickness from 0 to a maximum radius value (which the user can define), so that we obtain a 3D radial distribution. The image Figure 2.3 shows how an ideal radial distribution would look like. The measurement is then the maximum radius for which the radial distribution is above the threshold value computed with local mean.

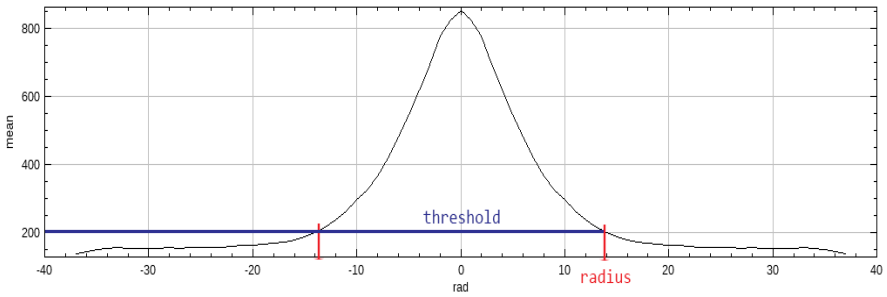


Figure 2.3: Threshold fitting on 3D radial distribution

2.4 MEAN SHIFT

The radius just found is eventually not so erroneous, for this reason it is exploited in the mean shift algorithm as look-distance for the neighboring points. The implementation of the mean shift algorithm in *BCMeasure* is pretty standard but there are some slight differences for which it is worth to deepen some

details.

Mean shift clustering allows finding the centroid of a set of points by shifting each starting point to a weighted mean in a defined neighborhood. The weight is computed using a kernel function of the distance between the starting point and the neighbor. A classic mean shift implementation for a set of data points X can be written like that in Algorithm 1.

Algorithm 1 Naive mean shift implementation

1: **for** $i \leftarrow 0$ **to** $nIteration$ **do**
2: **for all** $x \in X$ **do**
3: $m(x) \leftarrow \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$
4: $x \leftarrow m(x)$
5: **end for**
6: **end for**

Where:

- $K(d)$ is the kernel function of the distance between two points. *BCMeasure* implementation uses the widespread Gaussian Kernel function which is so defined:

$$K(d) = \exp\left(-\frac{d^2}{2\sigma^2}\right)$$

- $N(x)$ is the set of neighboring point of x within a first defined radius (in our case the one found at the end of the previous section).
- $m(x) - x$ is the mean shift for the point x.

Convergence is proved with a differentiable, convex, and strictly decreasing profile function (such as the gaussian kernel function).

Although, if we want more of a center of mass like in our case, we have to weight also with respect to the voxel intensity, not only the distance, therefore the formula at line 3 in Algorithm 1 would rather look like this:

$$m(x) \leftarrow \frac{\sum_{x_i \in N(x)} K(x_i - x) w(x_i) x_i}{\sum_{x_i \in N(x)} K(x_i - x) w(x_i)}$$

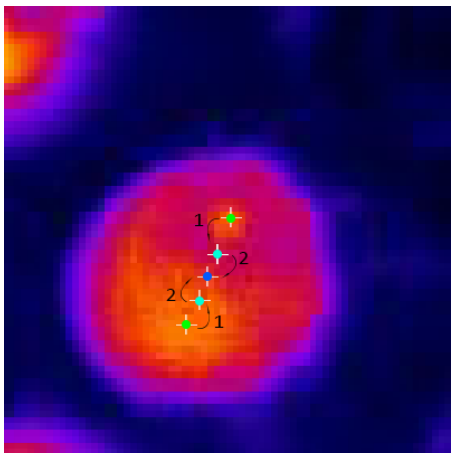


Figure 2.4: Center determination with mean shift clustering

with $w(x_i)$ indicating the intensity value of the voxel in x_i.

Also, it's reasonable to say that the set of datapoint is obviously not the entirety of the image voxels, but just the local maxima in the 3D cell stack, and only those whose intensity is above the threshold previously computed. Then amongst the centroids obtained after running the mean shift algorithm, only the one closer to the original seed is selected as the new current (and final) cell center.

2.5 THRESHOLDING WITH 3D RADIAL DISTRIBUTION (AGAIN)

The steps in Section 2.2 and Section 2.3 (except local max search since we now have the correct center) are repeated just like before but with the new current center.

One small difference is that now the radiuses for the local mean formula can be sized according to the first radius approximation (*i. e.* $old\_radius - 3$ for the first one and $old\_radius + 3$, $old\_radius + 23$ respectively for the second and third) and are no more strictly fixed. This guarantees a slightly more adaptive threshold value because it's almost sure that the conditions are met (first radius within the cell body, second and third in the background).

Apart from that, everything remains the same, and at the end the final radius is found.

3 IMPLEMENTATION

The development of the tool has been carried out mainly thanks to the ImageJ open-source platform[1] which provides an easy-to-use and rich API (Section 3.2). That software is completely written in Java, hence the programming language choice for *BCMeasure* tool. Also the 3D ImageJ Suite core library[2] (Section 3.4) has been widely used, especially for more specific 3D task. In the following sections the main choices concerning the script code will be presented. For further implementation details, the source code is available at `github.com/toyo97/bcmeasurej` accompanied by a README file which, amongst other things, explains the usage of the tool.

3.1 PIPELINE

The script starts parsing the options given from the command line (Section 3.5). Then it reads every marker file in the provided source directory. For each marker file then it looks for the associated image file (which must have the same name) and extract the list of `CellStack` objects with the special static method `getCellStacksFromSeeds()`. Each cell stack is processed locally as described in Chapter 2 and at the end the radius, along with the cell center, is saved in a new marker file. A log with all the execution details is saved in a TXT file in the source directory.

3.2 IMAGEJ LIBRARY

ImageJ provides various abstractions to represent an image. In particular, the most commonly used is the `ImagePlus` object, which contains an `ImageProcessor` (in case of 2D images) or an `ImageStack` (in case of 3D images, just like those in the dataset at issue), which basically is a list of `ImageProcessors`.

3D images in TIFF files are natively supported by the ImageJ platform. After opening a file by giving its path to `IJ.openImage()` static method, it's possible to get voxel intensities just by selecting the slice (Z coordinate) and the XY position on the slice data

1 https://imagej.net/Welcome
2 https://imagejdocu.tudor.lu/plugin/stacks/3d_ij_suite/start

array. The tool also exploits ImageJ GUI for visualizing logs and preview montages in debug mode.

3.3 CELLSTACK

Every core operation of the whole process is applied to a `CellStack` object. As already explained in Section 2.1 every image is decomposed in substacks, each of which contains a cell. Each substack is wrapped by a `CellStack` object and so augmented with new attributes and methods. This class extends ImageJ's `ImagePlus` class so that it's possible to make particular operations while preserving IJ images interface. For example, it embeds a `Box3D` object which stores the relative position of the cell inside the starting stack, making the reconstruction of the initial image possible even after local image processing.

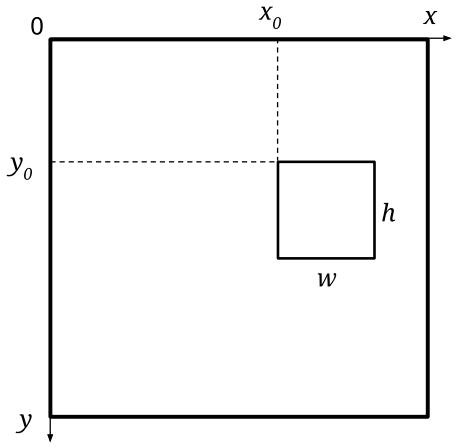


Figure 3.1: Box3D visualization example (in 2D). The outer box represents the initial image stack, the smaller one contains the cell which is about to be extracted and wrapped in a `CellStack` object. The values of x_0, y_0, z_0, w, h and d are stored in the `Box3D` object which is part of the cell stack.

Many operations among those explained in Chapter 2 are part of the `CellStack` class methods because they usually require a variety of cell information and therefore are tightly coupled with the object (see functions like `getLocalMean()`, `findMaxima()`, `computeRadialDistribution3D()`).

3.4 3D IMAGEJ SUITE

One of the biggest library included in the Fiji software[3] (which is just an enhanced version of ImageJ which includes many useful plugins) is the *3D ImageJ Suite* [4], a set of plugins for 3D image analysis and processing.
The core library of the suite[4] offers many useful algorithms that has been included in this project, in some cases with a simple call to the exposed public methods (such as `MaximaFinder`), in other cases slightly editing the code implementation (such as the `Neighborhood` class or `getlocalMean()` formula).

3.5 USER INTERFACE

BCMeasure tool is designed to be executed from command line with the installed JRE[5]. It provides a list of options with which the user can edit the default parameters. After the source code has been compiled, the script can be executed with the `java` command (after a valid Java JRE installation) as shown below:

```
$ java -cp ../lib/*:. bcmeasure -sd /home/user/path/to/source/files
```

The complete usage guide can be found in the README file available at the github page `github.com/toyo97/bcmeasurej`.

3 https://fiji.sc
4 https://github.com/mcib3d/mcib3d-core
5 Java Runtime Environment

4

RESULTS

4.1 ACCURACY EVALUATION

For what concerns the accuracy and correctness of the tool, no ground truth is available. Therefore, the solution proposed was to generate previews of a bunch of cells, among those processed, with the drawing of a circle of the determined radius. From this output is then possible to evaluate the overall accuracy of the process, counting the cells with a bad radius estimate.

As said before in the introduction, the stacks comes from different regions of the mouse cerebellum and for this reason the image quality it's highly variable. As a consequence, it seemed reasonable to make comparisons and evaluation considering cells both from high and low quality images.
However, precise details about the regions of the stacks are unknown - at least in the context of this work -, thus the density of each cells (with respect to the voxel intensities) was computed as a feature indicating the quality of the image. Up to 300 cells with their relative circle are then printed out divided in 3 montages (low, mid and high density) so that the user can estimate the accuracy and eventually retry with different parameters.

As shown in the figures, the tool with default parameters recognises quite well the size of the cells having higher density, while sometimes gets it wrong in low quality images. The overall accuracy, however, is more than tolerable considering that in some cases it's difficult to determine the radius even by closely looking at it (recall Figure 1.2 in Section 1.2.1).

4.2 FILTERS

As a further attempt in trying to improve the accuracy of the tool, filtering can be applied before the cell stacks are being processed. The tested 3D filtering methods are: gaussian blur, median and mean filters.

Figure 4.2 shows the difference between the three filters in terms of visualization.

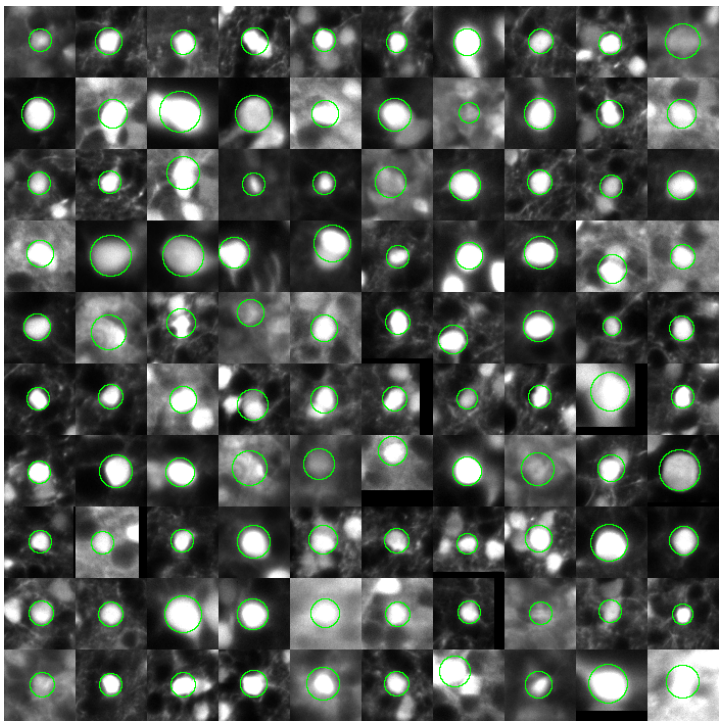


Figure 4.1: Debug mode montage output

The motivation for the use of such filters was to smooth the contour of the cells which sometimes, due to low contrast and bad image quality, have rough shapes. The above-mentioned filters are, in fact, commonly used in edge detection techniques and should denoise images while preserving edges[1].

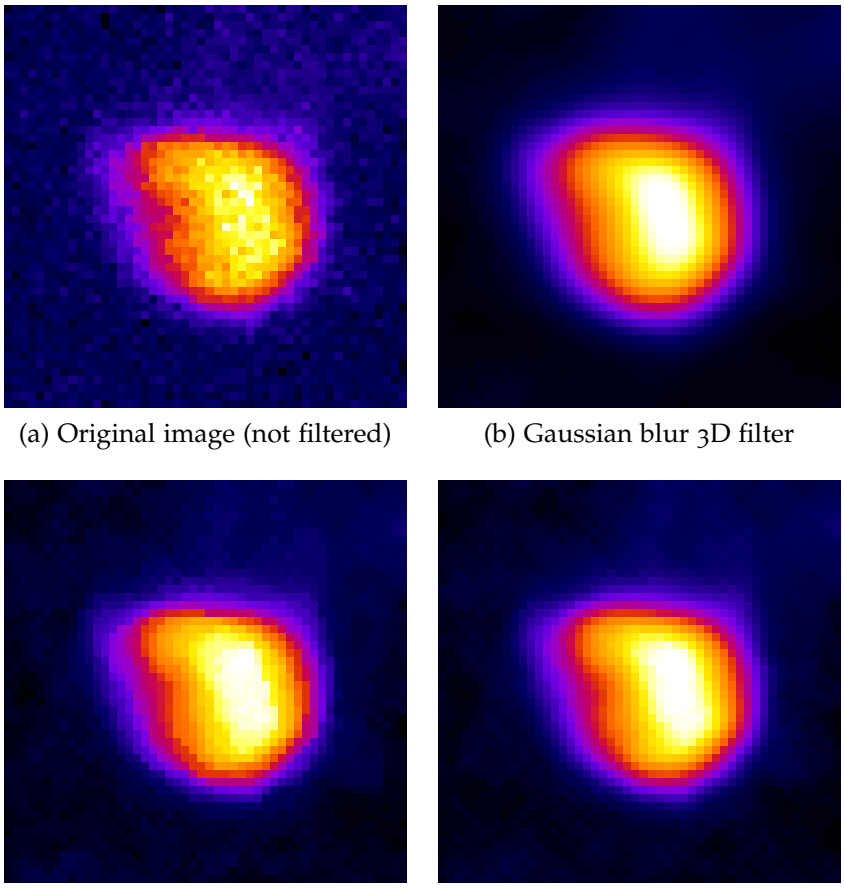
Nevertheless there's no evident improvement after applying the filters, on the contrary some cells are better measured without any preliminary smoothing.

4.3 EDGE CELLS

By default the script discards every cell that is not entirely within the image stack. It is possible, though, to include them, aware of the fact that the radius estimate could be completely wrong.

However, even cells that are partially out of the borders in some cases are measured decently (see Figure 4.3), in particular if they are mostly inside the stack.

1 https://en.wikipedia.org/wiki/Gaussian_blur, https://en.wikipedia.org/wiki/Median_filter



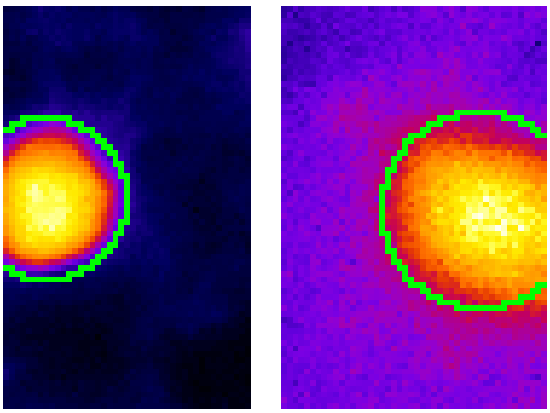
(a) Original image (not filtered)

(b) Gaussian blur 3D filter

(c) Median 3D filter

(d) Mean 3D filter

Figure 4.2: Comparison between tested filters



(a)

(b)

Figure 4.3: Example of good edge cell estimation

5 CONCLUSION

The lack of a ground-truth for the cell radius predictions makes it difficult to develop a quantitative analysis of the results. However, a more qualitative analysis has been made by observing hundreds of differents outcomes (see Section 4.1), always trying not to unconsciously fall in the so called *cherry picking* fallacy.

The overall accuracy can be estimated to be over 90%. This is a good result considering that, due to the bad quality of some images, even finding a simple general algorithm to extract the cell size was, at first, not obvious.

BIBLIOGRAPHY

[1] Paolo Frasconi, Ludovico Silvestri, Paolo Soda, Roberto Cortini, Francesco S. Pavone, and Giulio Iannello. "Large-scale automated identification of mouse brain cells in confocal light sheet microscopy images." In: *Bioinformatics* (2014).

[2] Jean-François Gilles, Marc Dos Santos, Thomas Boudier, Susanne Bolte, and Nicolas Heck. "DiAna, an ImageJ tool for object-based 3D co-localization and distance analysis." In: *Methods* (2017).

[3] N. Heck and T. Boudier. "3D spots segmentation." In: (). URL: imagejdocu.tudor.lu/_media/plugin/stacks/3d_ij_suite/3d_seg_spot_tutorial.pdf.

[4] Jean Ollion, Julien Cochennec, François Loll, Christophe Escudé, and Thomas Boudier. "TANGO: a generic tool for high-throughput 3D image analysis for studying nuclear organization." In: *Bioinformatics* (2013).

[5] L. Silvestri, A. Bria, L. Sacconi, G. Iannello, and F. S. Pavone. "Confocal light sheet microscopy: micron-scale neuroanatomy of the entire mouse brain." In: *Opt. Express* (2012).