Title Page

Title

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.

Authors

Jakub Tomek1,2
The main programmer, the author of the fast somata search algorithm.
Jakub Tomek wrote 70% of the presented software, including the fast somata search algorithm. He wrote 30% of the paper.

Ondrej Novak1 (Ondřej Novák)
The architect of the project, the author of the conception of TPP.
Ondrej Novak wrote 30% of the presented software. He measured the in vivo data and created routines to produce artificial data for the evaluation and performed the evaluation. He wrote 70% of the paper.

Josef Syka1
The supervisor of the project.

Affiliation

1 Department of Auditory Neuroscience, Institute of Experimental Medicine, Academy of Sciences of the Czech Republic, Prague, Czech Republic
2 Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University in Prague, Prague, Czech Republic

Running Head

TPP and SeNeCA - A new toolbox for two-photon microscopy

Contact Information

Ondrej Novak
Department of Auditory Neuroscience, Institute of Experimental Medicine, Academy of Sciences of the Czech Republic, Vídeňská 1083, 14220 Prague, Czech Republic
ondrej.novak@biomed.cas.cz
Phone: +420 241 062 687
Fax: +420 241 062 782
Abstract

Two-Photon Processor (TPP) is a versatile, ready-to-use and freely available software package in MATLAB to process data from in vivo two photon calcium imaging. TPP includes routines to search for cell bodies in full-frame (SeNeCA - Search for Neural Cells Accelerated) and line-scan acquisition, routines for calcium signal calculations, filtering, spike-mining and routines to construct parametric fields. Searching for somata in artificial in vivo data, our algorithm achieved better performance than human annotators. SeNeCA copes well with uneven background brightness and in-plane motion artifacts, the major problems in simple segmentation methods. In the fast mode, artificial in vivo images with a resolution of 256x256 containing ~100 neurons can be processed at a rate up to ~175 frames per second (tested on Intel i7, 8 threads, magnetic HDD). This speed of a segmentation algorithm could bring new possibilities into the field of in vivo optophysiology. With such a short latency (down to 5-6 ms in an ordinary PC) and using some contemporary optogenetic tools, it will allow experiments in which a control program can continuously evaluate the occurrence of a particular spatial pattern of activity (a possible correlate of memory or cognition) and subsequently inhibit/stimulate the entire area of the circuit or inhibit/stimulate a different part of the neuronal system. Two-Photon Processor will be freely available on our public website. Similar all-in-one and freely available software has not yet been published.

Keywords

Algorithm, segmentation, calcium imaging, processing
Introduction

In vivo two-photon laser scanning microscopy (TPLSM) has established itself as a powerful tool to reveal the morphological and functional characteristics of single cortical neurons in living anesthetized animals (Kerr et al. 2005; Knott et al. 2006; Ohki et al. 2005; Wilson et al. 2012). One of the crucial applications of this method is calcium imaging in vivo. To derive a fluorescence signal from each of the observed neurons and glial cells, it is often essential to circumscribe precisely the cellular somata in full-frame data or in line-scan data, respectively (Kwan 2010). In a typical one-day experiment, however, several gigabytes of data are produced. Such an amount of data needs to be processed using an automated software toolkit. Usually, data from two-photon calcium imaging are processed using custom-written software routines (Bandyopadhyay et al. 2010; Cohen et al. 2011; Jarosiewicz et al. 2012) with the codes not being published. To achieve more comparable outcomes from TPLSM, it is necessary to objectify the entire procedure of processing data, including all its inner constants. To the best of our knowledge, there is no open-source software, or any commercial program that is typically used for real-time segmentation of fluorescently stained cells in vivo and complete processing of the data produced in two-photon calcium imaging experiments. There are many well-documented tools available providing the possibility to segment in vitro data (Collins 2007; Kankaanpaa et al. 2012; Schneider et al. 2012). Searching for cells in images from in vivo preparations, such as from in vivo two-photon calcium imaging, is usually a more difficult task, typically due to a worse signal-to-noise ratio (SNR) and the presence of motion artifacts. However, there exist well-documented algorithms for the processing of TPLSM images, but all of them are restricted to a specific task, such as correction for motion artifacts (Greenberg and Kerr 2009) or image segmentation (Miri et al. 2011; Mukamel et al. 2009; Valmianski et al. 2010). Many of the image segmentation algorithms possess certain drawbacks. Simpler thresholding and/or watershed techniques, generally, are not able to cope
with uneven background brightness. Algorithms focused on temporal variation in the signal of each pixel, e.g. (Mukamel et al. 2009), fail to circumscribe non-spiking cells or synchronously firing cells. Machine learning algorithms (Valmianski et al. 2010) need training, while a regression-based algorithm (Miri et al. 2011) relies on the absence of motion artifacts, otherwise it needs a set of images to establish a reference image. However, the main drawback of all the algorithms that have been published so far is the rather slow speed of the procedure. If there exists an algorithm fast enough to allow an online frame-by-frame segmentation, there will be many possibilities how to immediately influence and further investigate the observed neural circuits. For example, one may perturb the function of the neural circuit through optogenetic excitatory or inhibitory stimulation (Boyden et al. 2005; Chow et al. 2010) to examine the causal role of specific neurons (e.g., those recently active) in the observed circuit. Such types of experiments require a very short latency between the moment when the image was taken and the moment when the image was segmented and annotated.

Here we address the development of a well-documented software package for the complete processing of data from TPLSM and the development of an ultra-fast image segmentation algorithm. We have written a versatile, ready-to-use and freely available software package in MATLAB to cope with this task. The program is called Two-Photon Processor and comprises routines for the processing of full-frame data and line-scan data, routines for the automatic creation of the line-scan trajectory, routines for the mathematical preprocessing of the data, spike mining routines and routines for the processing and visualizing of z-stack data. TPP also supports the assignment of any stimuli to the observed spikes. The core of the software package, the cell somata search algorithm, is called SeNeCA (Search for Neural Cells Accelerated) and is based on a combination of watershed, dynamic thresholding and morphometric constraints. All the inner parameters and constants of our software are well-
documented, reasoned and configurable. The latency between the measurement and obtaining
the complete information (segmentation and intensity step detection) could be as little as
several ms per one frame. The performance of our algorithm was evaluated against a human
annotator on real \textit{in vivo} data and against a true reference on synthetic data.

Materials and Methods

Experimental methods

Animal preparation

The entire experimental protocol, including surgery and anesthesia, was reviewed and
approved by the Ethics Committee of the Institute of Experimental Medicine and followed the
guidelines of the Declaration of Helsinki. We used male C57BL/6J mice, aged 4-6 weeks;
their weight was 20-25g. The mouse was anaesthetized with isoflurane (NICHOLAS
PIRAMAL) in pure oxygen (3\% for introduction, 1\% for maintenance; vol/vol). The animal
was placed on a custom built heating pad (38 °C). We applied 1 mg/kg atropine (BB
PHARMA, A.S.) under the skin on the back of the animal and injected 40μl of 2\% (wt/vol)
lidocaine (Egis Pharmaceuticals Ltd.) under the skin covering the skull roof and the right
temporal area. At these locations, the skin was resected and the soft tissue covering the area of
the parietal bones was gently scraped off. The head of the animal was glued to a metal holder
with cyanoacrylic glue. We removed the temporal muscle, and a small plastic reservoir to
hold the immersion was glued to the skull over the auditory cortex, in the middle of the most
ventro-temporal area of the right parietal bone. Then we drilled (OmniDrill, WPI) a small craniotomy (~2 mm in diameter) and removed the dura mater.

**Calcium dye loading and imaging**

Imaging was done using an UltimaIV two-photon laser-scanning microscope (Prairie Technologies) with a Chameleon Ultra II laser (Coherent). The objective used was a LUMPLFLN 40XW (Olympus). The calcium-sensitive fluorescent dye OGB-1AM (Invitrogen), 50 μg, was dissolved in 4 μl of 20% pluronic in DMSO (Invitrogen) and further diluted with artificial cerebrospinal fluid (119 mm NaCl, 26.2 mm NaHCO₃, 2.5 mm KCl, 1 mm NaH₂PO₄, 1.3 mm MgCl₂, 10 mm glucose) containing 50 μM SR-101 (Invitrogen). The dye mixture was pressure injected into the cortex at a depth of 150–300 μm using a pulled glass micropipette (diameter ~5 μm). The craniotomy was covered with 2% low melting point agarose (Roth) and coverslipped. The laser wavelength was set to 810 nm for all measurements. All calcium imaging data were processed using the Two-Photon Processor software package.

**Software work and evaluation**

**Artificial data modeling**

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 (the 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 $\mu$m and a standard deviation
of 1$\mu$m. The minimal and maximal diameters of the neurons were set to 6 $\mu$m and 12 $\mu$m,
respectively. Dendrite-like objects were 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 (Results) from fast acquisition. Those values were
0.278 for the average intensity inside the 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 $\mu$m 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.
We prepared several types of corrupted artificial datasets: different values of SNR, uneven staining of cells, in-plane and out-of-plane motion artifacts and three types of datasets with an uneven distribution of the intensity in the image. We defined SNR as the difference between the neuronal intensity and the background intensity divided by the level of noise outside the cellular somata. Besides the datasets with different values of SNR, all datasets were generated with the above-mentioned values of brightness and noise intensities. In the case of the dataset with uneven staining, the intensities of the cells were given by a Gaussian distribution. Several values of standard deviation were used. The mean value of the distribution was set by the generating procedure during the iteration. The positions of the cells in the datasets corrupted with motion artifacts had two components: the initial position and a small shift in the relevant axis. For each cell and each image in the dataset, the value (two values for in-plane artifacts) of the second component was chosen randomly from a normal distribution with a given standard deviation. This value was set to \(~1.2\mu m\) (1 pixel in images with 256x256 resolution and a 300\mu m wide FOV). Finally, three datasets with an uneven distribution of the intensity were created. Images in the first dataset were modulated by a linear ramp along the x-axis. The second dataset was modulated by a normal distribution with its maximum positioned in the center of the image and half maximum at the centers of the borders of the image. In the case of the third dataset, a Gaussian distribution with the same center and boundary values was used, but only the background and dendrite-like structures were modulated.

We created several uncorrupted datasets with three different resolutions (128x128, 256x256 and 512x512) and three different sizes of the field of view (150x150\mu m, 300x300\mu m and 600x600\mu m). Some of the combinations did not seem to be useful, so we used only six of them (Table 4). All of the corrupted datasets (SNR, uneven staining etc.) were prepared with a resolution of 256x256 and two fields of view: 150\mu m and 300\mu m.
Architecture of the Two-Photon Processor

Two-Photon Processor (TPP) comprises several tools to process data produced in experiments in fluorescently labeled tissues in living animals. The entire software package is written in MATLAB (MathWorks, Inc.), and in addition to MATLAB it requires three MATLAB toolboxes: Image Processing Toolbox, Statistics Toolbox and Signal Processing Toolbox. All the processes are controlled from a user-friendly Graphical user interface (GUI). The TPP can run on computers with a 32bit or 64bit system. We documented and commented all the main inner parameters of TPP throughout the code, and we left most of the parameters adjustable from the GUI. Further description of the software package can be found in the Results section of this article. A detailed description of the key algorithm, SeNeCA, follows below.

Description of SeNeCA

The SeNeCA algorithm is based on a combination of local thresholding and an appropriately interrupted watershed. Because the algorithm is aimed at use with in vivo data, it is necessary that it is able to handle uneven background intensity; this is achieved via using local information throughout the segmentation process. The idea is to identify the neuronal cores via local thresholding and then expand these objects to obtain good coverage of the somata. The algorithm uses two products of the original image: one is used for computing the local thresholds and the other one is used to seed and interrupt the watershed using those thresholds. The images are treated as matrices. An image is first preprocessed with a median
or disk blur filter (the size of the filter is always adjustable) to remove excessive noise (see Appendix). This step proved to be crucial for both speed and accuracy. Then, a new matrix LA (Light Averages; same size as the original image) is created. The LA matrix will be used further for the calculation of the local thresholds. Each of the elements of LA contains the average intensity of the pixels surrounding the analogous element (pixel) in the original image. The surrounding pixels are taken from a square with a defined size of the side, which is centered on this point. The size of the square area for averaging is adjustable as well. Another same-sized matrix, the future output mask, is initialized. The main parameters that need to be set and that are used throughout the algorithm include the higher light threshold coefficient and the lower light threshold coefficient. The actual thresholds at each point of the image are then given as a product of a particular coefficient and the value at the analogous location in the LA matrix. Every point of the denoised original image whose brightness exceeds the higher threshold is considered an intracellular point and represents a possible origin for the watershed wave. All those points are found and stored in a variable. A flooding wave is started from each of this point (Meyer's flooding algorithm (Meyer 1994)). The flooding is stopped by the lower light threshold. The positions are being continuously marked in the third matrix, the output mask. The objects, i.e., the putative cells, are numbered in the process. The edge of the \( n \)th neuron is marked with the number \( n \) in the matrix, while the inside of the \( n \)th neuron is marked by \(-n\). Finally, objects in the output mask are treated with a size filter; the span of the object is tested to be greater or lesser than the minimal or maximal parameter of the filter, respectively. The parameters of the filter are adjustable. Apart from the circumscription of the cells, SeNeCA calculates their centers of mass, reidentifies them between consecutive images based on their location, calculates their mean brightness and evaluates whether a jump in some intensities occurred compared to the previous image (trivial detection of a spike). The pseudocode of the algorithm and the time complexity analysis of the
SeNeCA algorithm can be found in the Appendix. The part of the algorithm after the blurring at the beginning and before the calculations of the centers of mass at the end was transcribed into the programming language C and is called a MEX file.

Performance evaluation of SeNeCA

The performance evaluation of the SeNeCA algorithm was done in three kinds of datasets: in vitro, in vivo and in silico (artificial data). In the case of comparing human and algorithmic performance on real in vitro data, the images annotated by a human annotator were taken as a reference. The levels of error produced by a human annotator were estimated as the difference between two annotations performed by two different human annotators [as was done in (Coelho et al. 2009)]. In the case of comparing human and algorithmic performance on real in vivo data, the images annotated by a human annotator were again taken as the reference; the segmentation performed by the human annotator was considered completely true (purposely). If an unquestionable reference was available (artificial data), the annotated images were evaluated against these reference images. Each image to be annotated had its corresponding reference image. All the cells in a reference image had their own unique identifier filling their entire area. In the case of the evaluation using artificial images, the circumscribed object in the annotated data was assigned to all cells in the reference image whose identifiers occurred at the same corresponding area. We distinguish four classes of errors (Coelho et al. 2009): Split - when two circumscribed cells are assigned to a single reference cell; Merged - when one circumscribed cell is assigned to two or more reference cells; Spurious - when a segmented cell is assigned to a background in the reference image; Missing - when no circumscribed cell is assigned to a reference cell.
Each of the datasets consisted of 100 different images. To evaluate the performance of SeNeCa, we randomly chose one of the images from a dataset and tested all of the combinations of the inner parameters of the algorithm to obey one of two different conditions. The first condition was a minimal total sum of errors. The second condition was to find all of the combinations of the parameters with the minimal possible number of omitted cells (missing), and if there were more combinations than one, subsequently a subset of them with the minimal number of merged cells was chosen. In the case of both of those conditions, the best combination of parameters was taken as the mean values over ten trials; the dataset with 100 images was segmented using that combination, and all types of errors were evaluated. This procedure was repeated 10x for each condition (randomization of the choice). The levels of errors are given in percentages with standard deviations.

The speed of the algorithm was evaluated 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, Motherboard Intel H67 Express, HDD 1000 GB 7200 RPM Serial ATA II). For the purpose of processing time series, SeNeCA in one image circumscribes the cells, calculates their centers of mass, reidentifies them between consecutive images, calculates their mean brightness and evaluates whether a jump in some intensities occurred compared to the previous image. Although we tested the speed of the algorithm on a data set that consisted of different sections rather than on time series, we left all of the mentioned calculations on to measure the speed usable for any real-time feedback intervention that would need such values as its input. To measure the time spent on the processing of one image, we processed 100 images from HDD and measured the spent time on one image using the tic/toc function. Only 90 (11th - 100th) images were taken into the analysis, because the first several times were usually corrupted with HDD delay. All errors are given with standard deviations. The values were evaluated on the entire
sets consisting of 100 images. Times are given with standard deviation as well; 90 durations were taken into account from each dataset.

Performance evaluation of other algorithms

To compare the performance of our algorithm with other (commonly) used segmentation algorithms, we also implemented them and used them instead of the segmentation core of SeNeCA (function segmentation – step 2.b; see Appendix). The chosen algorithms included Mean threshold, Otsu’s threshold and Gradient watershed, algorithms that were also evaluated using *in vitro* data (Coelho et al. 2009). The Matlab codes of these implementations can be found in the Appendix. The implementations included several adjustable constants (size of the blur filter etc.); these constant were optimized in the same way as in the case of the SeNeCA algorithm.

Results

Almost the entire Two-Photon Processor (TPP) was written in MATLAB, except the core of the SeNeCA algorithm. This calls a C function to accelerate the segmentation, but it is still completely controlled from the GUI in TPP via the Matlab wrapper function. The software package requires some toolboxes: Image Processing Toolbox, Statistics Toolbox and Signal
Processing Toolbox. The graphical user interface of TPP (Fig. 1) comprises one context menu listing the actual data processing tools and one context menu of possibilities how to work with the parameters. Each of the tools supports the adjustment of its main parameters and methods. The initial parameters can also be suggested by the software itself.

The scheme of TPP is shown in Fig. 2A. TPP includes 5 principal tools. The first one, *LS* (Line-Scan) *path planner*, is intended to find cells in a reference image and to propose a suitable trajectory for LS measurement. *FF* (Full-Frame) *segmentation and recording* serves as a tool to search for cells in a set of full-frame data in a frame-by-frame manner and to record intensities over time from each particular cell. The core algorithm of this tool is called SeNeCA. This algorithm can run in a fast selfstanding mode to segment full-frame data nearly real-time with a very short latency. The third tool, *LS segmentation and recording*, segments the outcome data from a LS experiment. Intensities over time could then be preprocessed, translated into a spike train and finally exported in *Data processor*. The last tool in TPP is a Z-stack processor that serves to process and display 3D data.

The software package was primarily developed to process data from two-photon calcium imaging *in vivo* (Fig. 2B). In *LS path planner* the reference image is segmented using SeNeCA (Fig. 2C), the centers of mass of the circumscribed cells are calculated and a trajectory for laser scanning is suggested. At first, the trajectory consists of straight lines and is optimized using the Ant Colony Optimization algorithm (Dorigo et al. 1996). The second step (optional) is an implementation of 'detours' (Fig. 2D). Having optimized the order of the objects, the algorithm can choose the optimal way to pass through the area of the cells and optionally to smooth the potential corners of the path. In tissues with a higher cellular density, such as the cerebral cortex, there is a relatively high probability that some of the trajectory
crossing will lie within the area of one of the cells. This would lead to a useless duplication of signal acquisition from cells containing the crossing. More passages through a cell mean more photobleaching. It is useful to avoid this problem, especially when the extra information is completely redundant (a neuron in one line-scan loop would be visited two or more times). In this case, LS path planner will suggest some ‘detours' around those cells involved (Fig. 2D, right middle part). The resulting trajectory can be saved as a text file that contains the points of the trajectory in relative coordinates (in relation to the corresponding field of view). This type of representation is highly transferable.

FF segmentation and recording is a part of the GUI where the segmentation of full-frame data can be controlled. The core of this tool is the SeNeCA algorithm, which was described above. To further improve the outcome of the segmentation, related algorithms can be used to decompose potential cell clusters. TPP includes two such algorithms. Cell cluster decomposition algorithms somewhat slow down the overall process, and they are not suitable for the fastest mode. FF segmentation and recording is intended to process large sets of full frame data. The cells are circumscribed in a frame-by-frame manner and then reidentified between consecutive frames. This implicitly solves the problem of in-plane motion artifacts. However, when motion artifacts are not strong, a certain number of consecutive frames could be averaged to obtain more precise boundaries of the cells. In both cases, the recording of the intensity of fluorescence over time from each cell is done using the original non-blurred data and is taken as the averaged or summary brightness of all of its inner pixels (pixels within its boundaries in the mask) in each frame. For a full-frame time series, the algorithm does not need any training.
**LS segmentation and recording** is a tool to identify brighter stripes in a line-scan data image (Fig. 2E). In these images (full of stripes) each row corresponds to a single passage of the line-scan path, the y-axis is the timeline and each bright vertical stripe corresponds to a single neuron or a glial cell. The positions of the cell boundaries are found by the algorithm in each row separately, so it can partially compensate for in-plane artifacts. The algorithm circumscribes the bright stripes and follows them throughout the entire experimental dataset. Some line-scan data, produced in a relatively long lasting experiment, might be split into several smaller data images by the measuring software (e.g. Prairie View Acquisition Software from Prairie Technologies does so). Those images are first glued together and then segmented as in the previous case. The splitting and re-gluing do not affect the results in any way. The intensity of fluorescence at a particular time point is calculated as the averaged or summary brightness of pixels that are defined by the borders of individual cells on the corresponding row.

*Data processor* serves to process intensity data; it is divided into two parts. The first part is devoted to data preprocessing, where users can enhance the SNR of their data or smooth them using some of the included data-improving filters (Fig. 2F). TPP supports low-pass or band-pass FFT-based filtration and many other filters. Setting the filtering can be first done for a single calcium record and then performed for the entire dataset as a batch process. The second part of the *Data processor* is intended to convert calcium traces to spike train data, to assign the spike responses to individual presented stimuli and to evaluate the responses of individual neurons. All these steps are optional. There are several methods for converting a calcium trace to a corresponding spike train (Lutcke and Helmchen 2011). We implemented two of those methods, the fast nonnegative deconvolution (Vogelstein et al. 2010) and the so-called peeling algorithm proposed by Helmchen (Grewe et al. 2010). The output of the nonnegative
deconvolution algorithm can be further thresholded to eliminate some false positives (Fig. 2H). Many of the inner parameters of those two algorithms are adjustable from the GUI. When desirable, the spike mining algorithms can be completely bypassed, and the user can continue processing with calcium traces. For the correct assignment of data with presented stimuli, it is necessary to accurately synchronize the computer that controls the measurements with the computer that generates the stimulation. The user enters into the computer the set of parameters of the presented stimuli, the order in which they have been presented and the interstimulus interval. When the responses are assigned to stimuli, there are several possibilities for exporting the processed data. In TPP, many types of schemes are supported. Besides some very common ones, such as PSTH (peristimulus time histogram), some rather specialized schemes are also ready to use: TRF (frequency-intensity tonal receptive field) for auditory research (Fig. 2I) and orientation and direction selectivity curves for vision research. The output module is quite open, so any diagram supported by MATLAB can be used when the code of TPP is slightly adapted.

Finally, the Z-stack processor is a useful tool to realign, rescale and display any measured Z-stacks. Realignment helps to compensate for motion artifacts. The process is based on a maximal correlation between consecutive frames. When the Z-stacks are measured, the number of micrometers per pixel in 2D sections may differ from the distance between them. The algorithm can resize the 3D object to correspond with the real size of the imaged specimen. The 3D viewer then offers the main projections.

Generally, we found that the most useful step is to blur the data slightly with a disk or median filter and then segment the preprocessed image. The calcium traces can be taken from raw data or preprocessed data as well. Our software package has a modular character and it is
open. When the data are segmented, processed and plotted, TPP stores the intermediate products in well defined files. At any step of the entire process, the data can be taken out and processed elsewhere.

Evaluation of in vitro data - SeNeCA vs. published algorithms

To determine whether SeNeCA can be used to process in vitro data we allowed it circumscribe freely available data published by the Murphy Lab (Coelho et al. 2009). The data consisted of two datasets; collections of U2OS cells and NIH3T3 cells. The non-bold values of error in Table 1 were counted manually by scientists from the Murphy Lab (Coelho et al. 2009); data used with permission. All the errors before slashes correspond to the U2OS dataset while the errors following slashes correspond to the NIH3T3 dataset. As is described above (Methods), the annotation performed by a human annotator was taken as a true reference. Errors are displayed as mean frequency of occurrence per one image. Both the datasets consisted of 50 images. Other non-bold rows are filed with the error values obtained with other algorithms; the articles describing those algorithms are cited in the first column. Using SeNeCA, we performed a segmentation of the data from the Murphy Lab and evaluated errors in the same way. Each object in the segmented image is assigned to the object in the reference image with which it shares the most pixels. The mean numbers of cells in case of U2OS cells and NIH3T3 cells were 36.6 and 43.6, respectively. As can be seen (Table 1), our algorithm is useful even to segment in vitro data, although it was not primarily created for this purpose. Its performance is only slightly poorer compared to a human annotator, but in the case of larger in vitro datasets, it seems inevitable to accept the slightly poorer performance of
an algorithm than to do the segmentation manually, which could require an unacceptable
amount of time. Our algorithm proved here to be superior to the other algorithms tested.

To further evaluate the performance of SeNeCA, we tested the algorithm on real \textit{in vivo} data
and on artificial data with identical parameters of contrast and noise. The same classes of
errors as described previously were distinguished: \textit{Split, merged, spurious and missing}. In the
following tables, those errors are taken as a percentage of the particular error per neuron and
the average number of neurons per frame. The values are accompanied by standard
deviations.

\textit{Evaluation of in vivo data - SeNeCA vs. a human annotator}

The \textit{in vivo} data were recorded from layers II/III of the auditory cortex of a mouse loaded
with OGB-1 and Sulforhodamin101 using the multicell bolus-loading (MCBL) method
(Stosiek et al. 2003). The frame rate of the acquisition was set to 50 fps, the resolution was
256x256 pixels and the field of view was 150\mu m each side (Fig. 3B). The images were not
averaged and suffered from excessive noise as they were recorded as fast as possible. One
human annotator was left to mark cells in 100 images (time sequence). The instruction was
only to mark the cells, not to circumscribe the cells. The parameters of the algorithm were
then adjusted to find all the cells marked by the human annotator in the first image. In figure
3C, there is an image showing cells marked by the human annotator (white dots) and cells
circumscribed by the algorithm (white line). Empty segmented areas represent errors of the
\textit{spurious} type. In the case of this experiment, the cells marked by the human annotator were
taken as the true reference (purposely). Thus, the results of this experiment do not indicate the
absolute performance of the algorithm, which would need to be based on a real and completely known reference. The percentage of errors (Table 2) is related to the average number of neurons found by the human annotator in one image (N human). The results from 100 in vivo images show that SeNeCA was relatively capable of finding almost all cells that were marked by the annotator, as only 5.9% of the cells marked by the human annotator were omitted (missing). This relatively low number was achieved by adjusting the parameters of the algorithm so that all the reference cells in the first image were circumscribed. The price of this setting is the high percentage of added cells (spurious). The additional objects identified by the algorithm might represent a real error or the algorithm might find some cells that were omitted by the human annotator. However, we cannot say for certain whose segmentation/annotation is better because in general, using in vivo data, we do not know what is a cell and what is not. Some may argue that this information can be obtained from an averaged image or by fine scanning in the z axis, but there are always some motion artifacts, and the average images are not the type of data that are segmented when we want to process the actual data frame after frame. Further insight into the question of whose annotation is better could be provided by the next section of this article, where the human annotator and the SeNeCA algorithm are compared using artificial datasets.

Evaluation of artificial data - SeNeCA vs. a human annotator

To measure the absolute performance of our algorithm, we prepared a set of artificial images that obey all the main attributes of real in vivo data. We modeled a cortical volume, a part of layers II/III of the mouse cortex (Methods). All of the artificial datasets can be downloaded from our web page (Endnotes). The reason for producing the artificial data was to precisely
define the position of each of the neurons. As mentioned in the previous section, we asked the question: which segmentation is more accurate, the one performed by our algorithm or the one performed by a human annotator? To answer this question, we compared both performances on two sets of artificial data. The resolution of both the sets was 256x256 pixels, and the sides of the field of view were 150µm (50 images) or 300µm (25 images) (Fig. 3E, F). We considered a neuron to be in plane if the distance between its center and the focal plane was lower than three-fourths of its radius. In both of the datasets, the human annotator was shown the first five images from each dataset and five corresponding reference images that showed the truly in-plane neurons. Using those images, the human trained what to mark as cells with an emphasis on a low rate of omitted cells (missing). Then, the SeNeCA algorithm was left to circumscribe the cells in the same datasets. The absolute average number of neurons per one image was evaluated from the reference images. Here we used the condition of the minimal sum of errors to optimize the parameters of SeNeCA. Annotation performed by our algorithm seemed better in both of the datasets (Table 3). The human annotator omitted a number of cells in the two datasets (Missing), 18.5% in images with 150µm FOV and 29.7% in images with 300µm FOV, respectively. On the other hand, in the case of SeNeCA, there were often some spurious objects, 7.0% in images with 150µm FOV and 4.7% in images with 300µm FOV, respectively. However, for actual usage in vivo, this is not a serious problem. The algorithm continuously labels the cells with identifying numbers and knows where each cell is. The borders and the signals of the less detectable cells can be simply discarded by the algorithm when a brief history (e.g., the 20 most recent images) of the particular cell is taken into account. For example, if the cell is absent too often in the most recent images, the cell is probably poorly stained or is located out of the focal plane. The only problem that this brings is that the higher number of added cells can slightly slow down the segmentation. The errors of the resting types, split and merged, are at a low level. Taken together, the SeNeCA
algorithm is more sensitive than a human annotator when artificial data are used. This would suggest that at least part of the spurious objects found by the SeNeCA algorithm in in vivo data might be real cells.

**Evaluation of artificial data – performance and speed**

Using the artificial datasets, we were able to measure the absolute performance and speed of the SeNeCA algorithm. The values are given with standard deviations. As can be seen (Table 4), when using the condition of a minimal sum of errors, spurious and missing dominate as expected. Although none of the levels of those two errors is strikingly high, the relatively high level of not found cells (missing) could be a problem, because some information from the neuronal assembly can be omitted. On the other hand, this type of error was rather low when we used the second type of condition (Table 4 - second part). Again, the price was the relatively high level of added cells, but as mentioned earlier this represents only a certain amount of redundant information that can be easily discarded. The most interesting feature is the speed of SeNeCA. The algorithm was continuously circumscribing the cells, identifying them according to their positions, calculating their centers of mass and the average intensities of fluorescence and detecting the potential steps in their intensities from frame to frame. All these values were cyclically overwritten in a global variable. With all of these operations, we tested SeNeCA on a PC (Table 4). The results were surprising. Even such numbers as 80 neurons, 256x256 pixels and 300µm FOV can be circumscribed in less than 6 ms! More than 400 cells, 512x512 pixels and 600µm FOV can be circumscribed between 20 and 25 ms. In general, the times are slightly longer if the second condition is used, but they are still very short.
Evaluation of corrupted artificial data – performance

To prove that SeNeCA is able to cope with uneven staining and background of the images, we tested the algorithm with the use of variously corrupted datasets. Those datasets included: different values of SNR (Fig. 4A), uneven staining of cells (Fig. 4B), in-plane and out-of-plane motion artifacts (Fig. 4C, D) and three types of datasets with an uneven distribution of the intensity in the image – images where only the background and dendrite-like structures were modulated by a Gaussian distribution (Fig. 4F), images modulated entirely by a Gaussian distribution (Fig. 4G) and images modulated entirely by a linear ramp along the x-axis (Fig. 4H). Together with the uncorrupted dataset, we used the last three listed corrupted datasets to evaluate and compare the performance of three chosen algorithms and SeNeCA (Fig. 4E-H). All of these comparisons were performed on the datasets with a resolution of 256x256 and two fields of view: 150µm (black columns) and 300µm (gray columns). For optimizing the parameters of the algorithm, we used the condition of the minimal sum of errors. In figure 4A, three different values of SNR were tested. One might be surprised that there is an optimum for which the SNR is not the highest (SNR=1). The reason is that we kept the intensity of the cell somata and dendrite-like structures equal when the dataset was being generated. In the case of the datasets with the highest SNR, almost the entire noise level, outside the cellular somata, was due to the dendrite-like structures. This caused problems for the algorithm. On the other hand, the level of errors is not unacceptably high, and the algorithm could still be useful even if the SNR is 0.5; for example, a value of ~20% in an image containing ~27 neurons means approx. 5 errors per one image. Secondly, we tested SeNeCA on unevenly stained cells. As can be seen (Fig. 4B), a slight unevenness in staining
did not cause any problems for our algorithm, but as the relative standard deviation of the staining distribution exceeds a certain limit (more than 20% of the mean value), the algorithm will start to err. A specific characteristic of in vivo imaging is the occurrence of motion artifacts can be found in figure 4C. Since our algorithm processes a time series frame by frame, the problem of motion artifacts is implicitly solved. The individual cells are numbered, and their positions are continuously tracked throughout the time series (Fig. 4D). In figure 4C, there is only a slight increase in the error rate when our algorithm processed the dataset corrupted with in-plane motion artifacts instead of the dataset with stationary cells. On the other hand, the out-of-plane motion artifacts still remain a problem for our algorithm.

Finally, we tested the other commonly used algorithms and compared their performance on uncorrupted and corrupted datasets. In clean, unmodulated datasets (Fig. 4E), only the performance of Otsu’s thresholding method was comparable to the performance of SeNeCA, while the others scored substantially worse. In the case of the Mean threshold method, the sum of errors was even higher than the number of neurons in the image (therefore the error level greater than 100% in figure 4E). In all of the remaining datasets (Fig.4 F-H), the performances of all three algorithms with which SeNeCA was compared were at least three times worse than SeNeCA. This confirms that it is absolutely essential that the algorithm interacts with the gross intensity modulation in the image. Our algorithm, SeNeCA, proceeds and adjusts its inner parameters locally; therefore, it copes well with uneven intensity distribution in the images.

Discussion
Although two-photon microscopy is becoming a more and more important tool in neurophysiological research, there is no standard as to how to process the produced data. Many teams use their own routines, but they do not publish the code. Here, we addressed the issue of developing a well-documented software package that would enable users to process the data completely and in a fully objectified manner. To the best of our knowledge, Two-Photon Processor will be the first public software package supporting the complete processing of data from two-photon microscopy. We will publish the entire code together with descriptive documents and tutorials (Endnotes).

One of the most important algorithms of TPP is the algorithm called SeNeCA. This algorithm is intended to segment the full-frame data in a frame after frame manner. Unlike some recently published segmentation algorithms (Mukamel et al. 2009; Ozden et al. 2008), SeNeCA does not rely on the activity of the cells. Due to the fact that our algorithm proceeds locally and its inner parameters are given in relative values, SeNeCA copes well with uneven staining or uneven background brightness (Fig. 4). This could be particularly useful when the multicell bolus loading technique is used (Stosiek et al. 2003).

We showed that SeNeCA could be successfully used in the case of in vitro data as compared with data from the Murphy Lab (Coelho et al. 2009) (Table 1). The mean summary rate of errors performed by SeNeCA in one image (6.1) was close to the error level of human performance (5.5) and nearly half of the rates of the next two most successful algorithms in this experiment [12 errors - Mean Threshold(Coelho et al. 2009); 13 errors - Merging algorithm (Lin et al. 2003)]. In the case of SeNeCA, a substantial part of the sum of error was due to not-separated cell clusters (merged). When some evenly stained cells are closely
attached, it is still rather difficult for the SeNeCA algorithm to split them correctly. However, if SeNeCA is started from the GUI, several cluster-splitting algorithms can be used in addition. On the other hand, as they could be quite time-consuming, we did not use them in the evaluation. Among the types of errors that we evaluated using the in vivo data and artificial data, the missing type could be considered as the worst one. Added neurons (spurious) in many applications do not matter because although they bring redundant information, the mean slowdown of the run of the algorithm is around 10% (extra time). Those false positive regions can be automatically discarded or just left unused. The split and merge types of errors are rather rare. On the other hand, information from the missing neurons is lost, and this could severely influence the outcome of the experiment. In the experiment with the in vivo data, the selection of parameters was carried out with regard to the minimum number of errors of this type as compared with human annotation taken as a reference. Due to the absence of an absolute reference, using the in vivo data it was difficult to judge whose annotation is better. The algorithm omitted ~6% of cells annotated by a human annotator (Table 2). On the other hand, it identified many more putative cells. To evaluate which annotation is better, we tried the algorithm and annotation by a human annotator on artificial datasets (Table 3). The parameters were chosen as those that produce minimal missing errors, and so the eyes of the annotator were set when the annotator had the opportunity to train the annotation on the first five images from the dataset having the corresponding reference images. Unlike the human annotator, the algorithm omitted only a small percentage of neurons. On the other hand, the price was the higher number of added cells (spurious). The SeNeCA algorithm seems more sensitive than a human annotator when artificial data are used. We think that at least part of the additional objects found by the SeNeCA algorithm in in vivo data were real cells. The average image of the in vivo dataset (Fig. 3D) also supports our conjecture. One of the biggest advantages of our algorithm is that it copes well with
modulation of the intensity in images (Fig. 4). Other tested algorithms completely failed in this task. Coping well with uneven brightness in an image should be one of the most important requirements for an algorithm, as this condition is quite often present in images obtained from many different preparations.

Finally, we tried the accuracy of the algorithm on artificial datasets with a broader group of resolutions and measured the mean time the algorithm spent on one image (Table 4). The parameters were selected (in a mutually exclusive manner) with regard to two conditions: 1) minimal sum of errors or 2) minimal missing errors and subsequently a subset with minimal of merged. The values of the error rates are displayed in Table 4. It is important to note that we tested our algorithm here on datasets in which each image corresponds with a different section of its generating artificial space, not on time series. Generally, the results of the evaluation are very similar to those in recently published papers (Valmianski et al. 2010). The main difference is the speed of the algorithm. Our algorithm is capable of processing one image in several milliseconds. Images with a resolution of 256×256 pixels and containing ~100 cells can be segmented in 6 ms, while those with a resolution of 512×512 pixels containing ~400 cells can be segmented in less than 25 ms. Note that the calculation of the position (center of mass) of each cell, its re-identification, the calculation of its mean intensity and any potential jump in intensity between the last images are already included in the given times. The fastest algorithms that have been published so far bring sufficient speed to process the data online when the animal is kept alive, e.g., 70 cells, 512×512 pixels ~ 1 min (Valmianski et al. 2010). The contemporary limit of the speed with which full-frame data can be acquired can be as low as 50 ns per pixel (Chen et al. 2011), meaning ~3 ms per one 256×256 image and ~13 ms per one 512×512 image. Considering that there could be some downsampling in time, the time spent on image segmentation ceases to be a problem if our algorithm is used.
The times in which our algorithm can process one image could be used for experiments from the field of optogenetics (Prakash et al. 2012). In recent years, bacterial opsins have been substantially optimized (Gradinaru et al. 2010; Gunaydin et al. 2010). There are several red shifted variants of channelrhodopsin [e.g., C1V1$_{T-T}$ (Yizhar et al. 2011b)] and bacteriorhodopsin [e.g., eArch3.0 (Yizhar et al. 2011a)] that can be stimulated using a two-photon laser scanning system running at a wavelength (~1040 nm) sufficiently different from those wavelengths used for two-photon calcium imaging (~800 nm). If one uses two separate scanning systems and two separate Ti:S laser systems, the optogenetic tools together with our fast SeNeCA algorithm enable experiments in which an immediate intervention in the observed circuits is desirable. For example, we can imagine such experiments where SeNeCA will be continuously evaluating an occurrence of a particular activation pattern (for example, a neural correlate) among the observed cells, and when it occurs, a follow-up program driving the second scanning and laser system will perturb some part of the activity pattern. When we use this paradigm during a conditioning or learning session in an awake head-fixed animal, we can directly influence the process of learning and further investigate the neural code. We believe that SeNeCA will help in this field of in vivo neurophysiology.

Appendix

Pseudocode of the entire SeNeCA algorithm

Inputs:

folder: the folder containing source images, which may be continually added in real time.
image: an image to be segmented, represented by its intensity matrix. It should be already preprocessed to remove excessive noise (disk blur or median filtering are suggested)
hlt: higher light threshold: Pixels brighter than this will be labeled as parts of the cell somata
llt: lower light threshold: Pixels brighter than this may be labeled as parts of the cell somata
cws: contrast window size: how large an area around a pixel is considered when determining if the pixel is bright enough compared to its surroundings. Square area.
**ml**: minimum light: A technical parameter helping to ignore a too dark somata (badly stained, out of the focal plane, etc.).

**minSize**: Minimum diameter (in pixels) of an object to be considered a neuron.

**maxSize**: Maximum diameter (in pixels) of an object to be considered a neuron.

**jumpSize**: Jump in intensity suggesting an action potential.

**Outputs:**

**mask**: a matrix of the same size as the source **image**. Contains \(-x\) at inside points of the \(x\)th neuron, \(+x\) at the border of the \(x\)th neuron and 0s outside neuronal somata.

1. **foundObjects** = empty set;
2. Read the contents of the specified **folder**, file by file. For each **file**:
   a. **image** = denoise(**file**); {Denoise the image via, e.g., median filtering or disk blur.}
   b. **mask** = segmentation(**image**, hlt, llt, cws, ml);
   c. Discard objects in the mask which have diameter less than **minSize** or larger than **maxSize**.
   d. Using nearest neighbor tracking, assign objects in mask to objects already saved in **foundObjects** (according to the position of their centroids). Add newly discovered objects, not assigned to any previously found object, to **foundObjects**.
   e. Measure and save the sum/average intensities of objects found in **mask**.
   f. Report any jump in intensity higher than **jumpSize**.

**Pseudocode of the part of the SeNeCA algorithm in C (MEX file)**

```c
function segmentation(image, hlt, llt, cws, ml) returns mask

1. Precompute matrix LA (light averages), where LA[i,j] contains the average intensity in a cwsXcws square centered around image[i,j]. This is actually a simple mean averaging.
2. Initialize queue for wave in step 5. The queue holds pairs of numbers (coordinates in a matrix)
3. neuronNumber = 1; {“Painting color”}
4. lightPoints = all pixels (i,j) such that image[i,j] > hlt*LA[i,j].
5. For each (i,j) in lightPoints:
   a. if mask[i,j] == 0 then
      i. enqueue i,j in queue;
      ii. mask[i,j] = -neuronNumber;
      iii. while queue not empty {A wave is ran from (i,j)}
         1. (a,b) = dequeue(queue)
         2. image[a,b] > llt*LA[a,b],
            a. mask[a,b] = -neuronNumber
            b. foreach neighbour In 8-neighbourhood of (a,b)
               i. if mask[neighbour] == 0,
                  1. enqueue neighbor
                  2. mask[neighbor] = -neuronNumber;
            3. if (a,b) lies on image border or image[a,b] < llt*LA[a,b],
```
a. \texttt{mask[a,b] = neuronNumber}

iv. \texttt{neuronNumber = neuronNumber+1}

b. otherwise do nothing

{It is likely that multiple pixels in a single soma will be present in \texttt{lightPoints}; however, we run the wave only once.}

\textit{Time complexity analysis of the SeNeCA algorithm}

An advantage of this segmentation algorithm is its near-linear complexity $O(o^2+p*s)$, where $o$ is the number of objects found in the series of images, $p$ is the number of pixels in one image and $s$ is the value of the parameter determining the side length of the square demarcating a pixel’s surrounding when the LA matrix is computed. There are two observations that can be made, changing the asymptotic complexity of the algorithm:

a) If we consider $s$ a constant (because it does not depend on the size of the image, it is given as a parameter), it makes the complexity $O(o^2+p)$

b) Furthermore, if $o<\sqrt{p}$, i.e., there are less found objects than a side of a square image contains pixels (e.g., there are less than 256 objects found in a 256x256 image), $p$ dominates, which makes the overall complexity linear, $O(p)$. In practice, the assumption b) often holds true. In the following steps, we derive the $O(o^2+p*s)$ complexity:

- In the preprocessing to remove excess noise, it is necessary to perform a sliding window operation with a selected filter with side size $f$. Therefore, the complexity is $O(p*f)$.

- Computation of the LA matrix (essentially a mean averaging) takes $O(p*s)$ time: If a sliding window is used, the computation of the initial window surrounding the pixel $(1,1)$ takes $O(s*s)$, which is trivially dominated by $O(p)$. Then, for every other pixel, $s$ pixels are added to the window and $s$ pixels are removed, which is $O(s)$ per pixel. The whole computation of LA therefore takes $O(p+p*s)=O(p*s)$. Note that as $f$ from the
previous step is practically always smaller than $s$ of the next step, the complexity of the previous step is dominated by the complexity of this step.

- Determining which pixels are lighter than high_light_threshold_coefficient*LA(pixel) is obviously linear $O(p)$ as only a single simple condition is evaluated for each pixel.

- Extending the light pixels from the previous step into full covering of the neuronal somata via the interrupted watershed is also $O(p)$, because no pixel is enqueued twice in the process. Dealing with a single pixel is obviously an operation with constant complexity.

- The assignment of objects to the previously found ones may be trivially done in $O(o^2)$: for each centroid of the currently found objects, the centroids of the objects found in previous frames are iterated over and the nearest one is picked (if no suitably near object is found, the currently processed object is said to be newly appeared). As there are $o$ objects found in total, after SeNeCA ends, both the number of objects found in the current frame and in the previous frames are dominated by $o$, which leads to complexity $O(o^2)$ of this step.

Summing up the complexities of the steps above, eliminating dominated elements, we obtain $O(o^2 + p*s)$. 
Implementation of the other tested algorithms (Matlab code)

Mean threshold

```matlab
function mask=meanThreshold(image, blurSize, MinSize, MaxSize)
  %image – raster image to be segmented; expected in 8bit grayscale
  %blurSize – diameter of the circular blurring filter
  %MinSize – the lower value of the size filter
  %MaxSize – the higher value of the size filter

  %mask – matrix of the same size as image; contains identifying numbers
  % at the positions of boundaries of each particular object

  %TPUtils.blurImageDisk – custom-written function to blur the image
  %TSeriesProcessor.cleanMask – custom-written function to filter the
  %found objects based on their size

  image=TPUtils.blurImageDisk(image, blurSize);
  mask=im2bw(image,double(mean2(image))/255);
  mask=bwlabel(mask,4);
  [mask]=TSeriesProcessor.cleanMask(mask,MinSize,MaxSize);
end
```

Otsu’s threshold

```matlab
function mask=otsuThreshold(image, blurSize, MinSize, MaxSize)
  %image – raster image to be segmented; expected in 8bit grayscale
  %blurSize – diameter of the circular blurring filter
  %MinSize – the lower value of the size filter
  %MaxSize – the higher value of the size filter

  %mask – matrix of the same size as image; contains identifying numbers
  % at the positions of boundaries of each particular object

  %TPUtils.blurImageDisk – custom-written function to blur the image
  %TSeriesProcessor.cleanMask – custom-written function to filter the
  %found objects based on their size

  image=TPUtils.blurImageDisk(image, blurSize);
  mask=im2bw(image,double(graythresh(image)));
  mask=bwlabel(mask,4);
  [mask]=TSeriesProcessor.cleanMask(mask,MinSize,MaxSize);
end
```

Gradient watershed

```matlab
function mask=gradientWatershed(image, blurSize, MinSize, MaxSize, openSize, minPix, supressIntensity)
  %http://blogs.mathworks.com/steve/2006/06/02/cell-segmentation/

  %image – raster image to be segmented; expected in 8bit grayscale
```
%blurSize – diameter of the circular blurring filter
%MinSize – the lower value of the size filter
%MaxSize – the higher value of the size filter
%openSize – size of the filter for the operation imopen – cleaning
%minPix - size of the filter for the operation bwareaopen – cleaning
%supressIntensity – parameter for the function imextendedmax – search
%for extended maxima
%mask – matrix of the same size as image; contains identifying numbers
%       at the positions of boundaries of each particular object
%TFUtils.blurImageDisk – custom-written function to blur the image
%TSSeriesProcessor.cleanMask – custom-written function to filter the
%found objects based on their size

I=TFUtils.blurImageDisk(image, blur);
hy = fspecial('sobel');
hx = hy';
Iy = imfilter(double(I), hy, 'replicate');
Ix = imfilter(double(I), hx, 'replicate');
gradmag = sqrt(Ix.^2 + Iy.^2);
I_eq = adapthisteq(uint8(gradmag));
bw = im2bw(I_eq, graythresh(I_eq));
bw2 = imfill(bw,'holes');
bw3 = imopen(bw2, ones(openSize,openSize));
bw4 = bwareaopen(bw3, minPix);
I_eq=I;
mask_em = imextendedmax(I_eq, supressIntensity);
mask_em = imclose(mask_em, ones(openSize,openSize));
mask_em = imfill(mask_em, 'holes');
mask_em = bwareaopen(mask_em, minPix);
I_eq_c = imcomplement(I_eq);
I_mod = imimposemin(I_eq_c, ~bw4 | mask_em);
m = watershed(I_mod);
m(m==1)=0;
[cleanedMask]=TSSeriesProcessor.cleanMask(m,MinSize,MaxSize);
end

Grants

This work was supported by the Grant Agency of the Czech Republic P303/12/1347 and
P304/12/G069.

Disclosures
No conflicts of interest, financial or otherwise, are declared by the authors.

Endnotes

All of the software presented in this paper will be available for download at uemweb.biomed.cas.cz/tpp/ together with appropriate documentation and tutorials. Matlab functions written to generate artificial data will be possible to find on the same page.

Reference List


FIG.1. Graphical user interface of Two-Photon Processor. The currently active mode shown in the image is FF segmentation and recording.

FIG.2. An overview of the features and capabilities of the TPP software package. A: a scheme of the GUI. B: cells in layers II/III of the auditory cortex stained with OGB-1 (green) and Sulforhodamine101 (yellow). C: cells segmented by our algorithm (red lines - boundaries), same image as in B. D: a scan path (green lines) proposed by the algorithm. Where the laser cross the area of a cell more than once, the laser path is modified to go around the cell, same image as in B and C. E: an example of striped data produced in a line-scan experiment. F: an example of using routines for signal preprocessing. Two calcium traces are shown, one before, and one after the FFT low-pass filtering. In this case, frequencies above one-tenth of the sampling frequency have been removed. G: a look at the 3D object browser. H: transformation of calcium traces (green) into spike representation. Here, the non-negative deconvolution method of Vogelstein was used. The outcome of the procedure (blue spikes) can be subsequently thresholded (red line), and only those spikes, that exceed the threshold are taken into account (red dots). I: characteristics of a single neuron evaluated using Two-Photon Processor - tonal receptive field (left) and peristimulus time histogram (right). Scale bars in B, C and D correspond to 25μm.

FIG.3. Datasets for evaluating the performance of SeNeCA. A: an image from a freely available dataset proposed and shared by the Murphy Lab; a collection of NIH3T3 cells. B: in vivo data taken in fast acquisition. C: an image merging an annotation performed by a human annotator (white dots) and the segmentation performed by the algorithm (white circumscriptions). D: an average image from the in vivo dataset (100 images). E: an example of synthetic data; 300μm FOV, 256x256 pixels. F: the image from E segmented by SeNeCA. Scale bars in B, C and D correspond to 25μm, in E and F the scale bars correspond to 50μm.

FIG.4. Corrupted datasets for evaluating the performance of SeNeCA. Resolution of all the datasets was 256x256 pixels. Black columns correspond to 150μm FOV; gray columns correspond to 300μm FOV. A: images with different values of SNR. B: images with uneven staining of the cells. C: influence of motion artifacts on the performance of SeNeCA. D:
example of continuous segmentation and identification of the cells. E-H: comparison of the
performance of SeNeCA and other tested algorithms - Mean thresholding, Otsu's thresholding
and Gradient watershed. E: Clear datasets with no intensity corruption. F: datasets with
modulated intensity of the background and dendrite-like objects; Gaussian modulation. G:
datasets with modulated intensity of the entire images; Gaussian modulation. G: datasets with
modulated intensity of the entire images; linear ramp modulation along one axis.

Table 1. Results of the evaluation of in vitro data - SeNeCA vs. published algorithms. Errors
given as the mean number of errors of a particular type related to the mean number of cells
per one image. Data from the Murphy Lab were used with permission.

<table>
<thead>
<tr>
<th>Algorithm + reference</th>
<th>Split</th>
<th>Merged</th>
<th>Spurious</th>
<th>Missing</th>
<th>Sum of mistakes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS Manual (Coelho et al. 2009)</td>
<td>1.6/1.0</td>
<td>1.0/1.2</td>
<td>0.8/0.0</td>
<td>2.2/3.2</td>
<td>5.5</td>
</tr>
<tr>
<td><strong>Our algorithm SeNeCA</strong></td>
<td><strong>0.0/0.0</strong></td>
<td><strong>2.2/4.1</strong></td>
<td><strong>1.5/1.5</strong></td>
<td><strong>1/1.8</strong></td>
<td><strong>6.1</strong></td>
</tr>
<tr>
<td>Mean Threshold (Coelho et al. 2009)</td>
<td>1.3/1.4</td>
<td>3.4/5.1</td>
<td>0.9/3.1</td>
<td>3.6/4.8</td>
<td>12</td>
</tr>
<tr>
<td>Merging algorithm (Lin et al. 2003)</td>
<td>1.8/1.6</td>
<td>2.1/3.0</td>
<td>1.0/6.8</td>
<td>3.3/5.9</td>
<td>13</td>
</tr>
<tr>
<td>RC Threshold (Ridler and Calvard 1978)</td>
<td>1.1/1.0</td>
<td>2.4/2.4</td>
<td>0.3/1.9</td>
<td>5.5/22.1</td>
<td>18</td>
</tr>
<tr>
<td>Watershed (gradient) (Coelho et al. 2009)</td>
<td>7.7/2.6</td>
<td>2.0/3.0</td>
<td>2.0/11.4</td>
<td>2.9/5.4</td>
<td>19</td>
</tr>
<tr>
<td>Otsu Threshold (Otsu 1979)</td>
<td>1.1/0.8</td>
<td>2.3/2.1</td>
<td>0.3/1.7</td>
<td>5.6/26.6</td>
<td>20</td>
</tr>
<tr>
<td>Watershed (direct) (Coelho et al. 2009)</td>
<td>13.8/2.9</td>
<td>1.2/2.4</td>
<td>2.0/11.6</td>
<td>2.0/5.5</td>
<td>21</td>
</tr>
<tr>
<td>Active Masks (Srinivasa et al. 2008)</td>
<td>10.5/1.9</td>
<td>2.1/1.5</td>
<td>0.4/3.9</td>
<td>10.8/31.1</td>
<td>31</td>
</tr>
</tbody>
</table>

Table 2. Comparison between the performance of a human annotator and the algorithm. The
human annotator is taken as the reference here. Errors are given in %; number of errors of the
particular type related to the mean number of cells per one image.

<table>
<thead>
<tr>
<th>Performer</th>
<th>Split</th>
<th>Merged</th>
<th>Spurious</th>
<th>Missing</th>
<th>N human</th>
<th>N alg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>0.08</td>
<td>0.30</td>
<td>48.7</td>
<td>5.9</td>
<td>25.6</td>
<td>36.5</td>
</tr>
</tbody>
</table>

Table 3. Evaluation of the absolute performance of a human annotator and the algorithm on
exactly defined artificial data. The known position of the neurons is taken as the reference
here. The resolution of the datasets was 256x256 in both cases. Errors 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>
<thead>
<tr>
<th>Performer</th>
<th>FOV side</th>
<th>split</th>
<th>Merged</th>
<th>Spurious</th>
<th>Missing</th>
<th>Neurons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>150x150μm</td>
<td>0±0</td>
<td>0±0</td>
<td>0.2±1.1</td>
<td>18.5±9.1</td>
<td>27.3±4.3</td>
</tr>
<tr>
<td>Algorithm</td>
<td></td>
<td>0.2±0.2</td>
<td>1.9±0.6</td>
<td>7.0±2.5</td>
<td>4.0±3.4</td>
<td></td>
</tr>
<tr>
<td>Human</td>
<td>300x300μm</td>
<td>0.2±0.5</td>
<td>1.2±1.3</td>
<td>0.1±0.3</td>
<td>29.7±7.3</td>
<td>108±14</td>
</tr>
<tr>
<td>Algorithm</td>
<td></td>
<td>0.6±0.2</td>
<td>1.9±0.6</td>
<td>4.8±1.3</td>
<td>4.6±2.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Evaluation of the absolute performance and speed of the algorithm on exactly
defined artificial data. The known position of the neurons is taken as the reference here.
Errors are given in %; number of errors of the particular type related to the mean number of
cells per one image. Values are means ± SD.

<p>| Condition | resolution [pixels] | FOV side [μm] | neurons | time/frame [ms] |  | Split [%] | Merged [%] | Spurious [%] | Missing [%] |
|-----------|---------------------|---------------|---------|-----------------| |          |            |              |             |             |
| minimal   | 128x128             | 150           | 30.8±7.3| 3.1±0.3         | | 0.9±0.2 | 3.0±0.7     | 6.1±1.0     | 4.3±1.6     |
| sum of    | 256x256             | 150           | 27.3±4.3| 6.3±0.7         | | 0.2±0.2 | 1.3±0.7     | 8.2±6.0     | 5.8±3.4     |</p>
<table>
<thead>
<tr>
<th>Errors</th>
<th>300</th>
<th>108±14</th>
<th>5.7±0.8</th>
<th>0.6±0.2</th>
<th>1.7±0.5</th>
<th>4.8±1.2</th>
<th>5.6±1.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>512x512</td>
<td>150</td>
<td>25.2±6.3</td>
<td>27±4</td>
<td>0.2±0.1</td>
<td>1.2±0.5</td>
<td>9.4±3.3</td>
<td>3.2±1.9</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>99±12</td>
<td>21±2</td>
<td>0.4±0.2</td>
<td>2.0±0.7</td>
<td>7.9±1.8</td>
<td>4.3±1.8</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>417±24</td>
<td>21±2</td>
<td>0.4±0.1</td>
<td>1.4±0.4</td>
<td>4.8±1.4</td>
<td>9.1±4.2</td>
</tr>
<tr>
<td>1) minimal missing error</td>
<td>128x128</td>
<td>150</td>
<td>30.8±7.3</td>
<td>3.1±0.2</td>
<td>1.1±0.1</td>
<td>3.5±0.2</td>
<td>21±10</td>
</tr>
<tr>
<td>2) minimal merged error</td>
<td>256x256</td>
<td>150</td>
<td>27.3±4.3</td>
<td>6.5±0.8</td>
<td>0.4±0.2</td>
<td>1.6±0.6</td>
<td>14.8±4.4</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>108±14</td>
<td>6.5±0.8</td>
<td>1.0±0.1</td>
<td>2.6±0.4</td>
<td>40±15</td>
<td>1.4±0.5</td>
</tr>
<tr>
<td></td>
<td>512x512</td>
<td>150</td>
<td>25.2±6.3</td>
<td>27±4</td>
<td>0.3±0.1</td>
<td>1.8±0.3</td>
<td>19.0±2.8</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>99±12</td>
<td>22±2</td>
<td>0.5±0.1</td>
<td>2.4±0.3</td>
<td>27.2±7.1</td>
<td>1.6±0.3</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>417±24</td>
<td>31±5</td>
<td>0.9±0.1</td>
<td>2.4±0.2</td>
<td>61±11</td>
<td>1.3±0.3</td>
</tr>
</tbody>
</table>

1076