An efficient and extendable Python library to analyze neuronal morphologies

Benjamin Torben-Nielsen*

May 1, 2014

Neuronal morphology has been of interest to neuroscientists since Cajal and Golgi. Due to technical advances and data-sharing initiatives[1] we have access to more neuronal reconstructions than one could accumulate in a lifetime up to recently. It is known that while neuronal morphology is highly diverse and variant[2] it is pivotal for brain functioning because the overlap between axons and dendrite limits the network connectivity (Peters' rule[3]) and dendrites define how inputs are integrated to produce and output signal[4]. Moreover, morphological anomalies and changes are often implicated in neuro-developmental and degenerative diseases[5]. These insights could not have been established without the ability to rigorously quantify neuronal morphologies.

Nowadays quantification is done on reconstructed neuronal morphologies, that is, digital representations of neuronal structures. Reconstruction is done with dedicated software programs such as Neurolucida[6] that turn a "picture" (or a stack thereof) into information usable for quantification by a computer. Neurolucida also comes with some built in functionalities for the analysis of morphologies. However, currently, the *de facto* standard

*Computational Neuroscience Unit, Okinawa Institute of Science and Technology Graduate University, Okinawa, Japan

[1]Giorgio A. Ascoli, Duncan E. Donohue, and Maryam Halavi. "NeuroMorpho.Org: A central resource for neuronal morphologies". en. In: *Journal of Neuroscience* 27.35 (Aug. 2007), pp. 9247–9251.

[2]Ivan Soltesz. *Diversity in the neuronal machine: Order and variability in interneuronal microcircuits*. New-York: Oxford University Press, USA, 2005.

[3]A Peters and BR Payne. "Numerical relationships between geniculocortical afferents and pyramidal cell modules in cat primary visual cortex". In: *Cerebral Cortex* 3.1 (1993), pp. 69–78.

[4]Benjamin Torben-Nielsen and Klaus M. Stiefel. "An inverse approach for elucidating dendritic function". In: *Frontiers in Computational Neuroscience* 4 (2010), p. 128.

[5]W E Kaufmann and H W Moser. "Dendritic anomalies in disorders associated with mental retardation." In: *Cerebral Cortex* 10.10 (Oct. 2000), pp. 981–991.

[6]Jacob R Glaser and Edmund M Glaser. "Neuron imaging with neurolucida - A PC-based system for image combining microscopy". In: *Computerized Medical Imaging and Graphics* 14.5 (1990), pp. 307–317.

file format to digitally store and publicly share neuronal reconstructions is the program-indepedent SWC format[7]. Two widely adopted tools exist to analyse SWC files, L-Measure[8] and the TREES toolbox[9]. L-Measure is the current "golden standard" in morphological analysis and written in Java. It has a web-interface and a standalone version with a graphical user interface (GUI). The TREES toolbox is a Matlab[10] toolbox. Both tools allow users to load and quantify (populations of) digitally reconstructed neurons. The TREES toolbox has the advantage of being implemented in Matlab and hence users can easily integrate it in their own work-flow by scripting in Matlab. Lately, there is a trend in computational neuroscience to use the Python programming language but there is no standalone program or library in Python to perform basic morphological quantification.

We designed and implemented BTMORPH, a Python library that contains a data structure and a set of routines to efficiently represent and analyze neuronal morphologies. The rationale of this library is to provide a solid, well tested backbone in the form of a data structure and atomic morphometric functions that allow users to analyze morphologies in a flexible way. By design, we treat neuronal morphologies as tree structures and all provided morphometrics can be computed on any (sub)tree structure. As such, morphometrics can be performed on the whole structure (as is usually done) or on any subtree; subtrees can be trees made up by a specific neurite type (axon, apical dendrite, ...) or can be selected specifically by, for instance, centrifugal order. This rationale focusing on flexibility and extensibility contrast with the more monolithic approach of existing tools to analyze morphologies and allows users to integrate, and built upon, the functionality of BTMORPH.

The functionality of BTMORPH is available through a documented application programming interface[11] (API). A current list of atomic functions is listed in Table 1. The typical workflow is that a user loads an SWC file into the provided tree-structure. The obtained tree is passed to a statistics class and morphometrics are computed on the fly when users query them. Users can then query morphometric measures of the whole tree, subtrees and even specific nodes (e.g., branching nodes). Because BTMORPH is written in Python users can exploit the full spectrum of tools available in Python (Mat-

[7] RC Cannon et al. "An on-line archive of reconstructed hippocampal neurons". In: *Journal of Neuroscience Methods* 84.12 (Oct. 1998), pp. 49–54.

[8] Ruggero Scorcioni, Sridevi Polavaram, and Giorgio A Ascoli. "L-Measure: a web-accessible tool for the analysis, comparison and search of digital reconstructions of neuronal morphologies." In: *Nature Protocols* 3.5 (Jan. 2008), pp. 866–876.

[9] Hermann Cuntz et al. "One rule to grow them all: A general theory of neuronal branching and its practical application". In: *PLoS Computational Biology* 6.8 (2010), e1000877.

[10] ©The MathWorks, Inc.

[11] http://btmorph.readthedocs.org/en/latest/api.html

plotlib[12], Scipy[13], and sqlite3[14] to name a few) to perform further analysis, visualization and storage of results obtained through BTMORPH

The target audience of BTMORPH consists of researchers who can read and modify Python scripts. However, we also provide straightforward wrappers around the code to perform essential functionality such as analyzing multi-dimensional (e.g., conditional) data in single neurons or to record population statistics. A typical session using BTMORPH is shown in Figure 1: with only a few lines of code a dendrogram is generated as well as a histogram summarizing differences in segment length ending in branching points or terminal points. Several additional visualizations are included as well such as a 3D plot (not shown) and pseudo-3D plot plot with three 2D projections and a population density plot that generates a heat-map of the neurite processes (Figure 2).

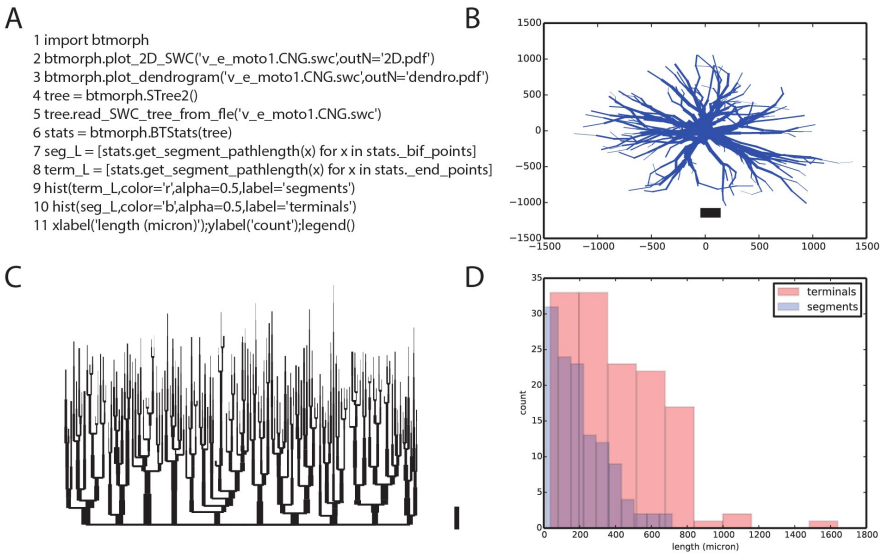


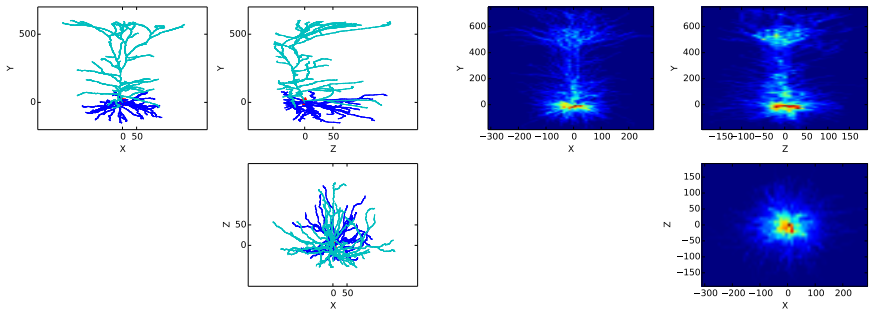
Figure 1: Exemplar use of BTMORPH. A: Code snippet (line numbers added for clarity). B-D: Results of executing the code snippet. B: line 2 (of the snippet)generates a 2D rendering of the morphology and save the file as PDF. C: line 3 generates a dendrogram of the same morphology. D:execution of lines 4-11 load a morphology into the provided data structure (`STree2`) and computes and visualizes the segment length statistics of this neuron.

An important advantage of using an external library is that publicly-available, open-source code is usually subject to extensive testing. BTMORPH is no exception and is bench-marked against L-Measure. An overview of the

[12]http://matplotlib.org/
[13]http://scipy.org/
[14]https://sqlite.org/

Feature	**Explanation**
Topological features	
# stems	Number of trunks sprouting from the soma
# segments	Number of segments. That is stretches of neurite between branching points and between the soma and branching points
# terminals	Number of terminal points
order	Centrifugal order of a point in the tree, i.e., the number of branch points between that point and the soma
degree	Number of terminal points in the subtree starting at any point
partition asymmetry	Topological measure of asymmetry in a tree as defined in[15]
Geometrical features	
total length	Summed length of the complete morphology (in μm)
width	Total extend in the X axis (in μm)
height	Total extend in the Y axis (in μm)
depth	Total extend in the Z axis (in μm)
segment length	Length of the segment (ending in a given branching point)
segment Euclidean distance	Euclidean distance of a segment (ending in a given branching or terminal point)
contraction	Measure of meandering in neurites
bifurcation angle	Angle between two daughter branches in the plane defined by the parent and its daughters
Rall's power	Power p for which the following equation has the smallest error: $D^p = {d_1}^p + {d_2}^p$. Two implementations. One with brute-force and one based on the downhill simplex (Nelder-Mead) search method.
Rall's ratio	$\frac{{d_1}^p + {d_2}^p}{D^p}, p = 3/2$ being Rall's theoretically predicted power that would minimize $D^p = {d_1}^p + {d_2}^p$.

Table 1: Atomic morphometric features presently available in BTMORPH.



(a) `btmorph.true_2D_projections` (b) `btmorph.population_2D_density_projections`

Figure 2: Demonstration of included visualizations. Left: a pseudo-3D plot made up of three 2D projections. Right: a population density plot with a heat map indicating the locations of all neurites in a population again in three 2D projections. The function names are shown in the captions. (Selected L5 pyramidal neurons from the Kawaguchi archive downloaded from NeuroMorpho.org)

comparison between L-Measure and BTMORPH is available on the documenting website[16]. Moreover, we include unit-tests implemented using the Nose testing framework[17]. Currently, only the *de facto* standard SWC-format is supported. When other morphological description formats (e.g., NeuroML) gain more momentum, they can be incorporated in a straightforward fashion by implementing an adapter that loads these description into the BTMORPH tree structure. In the future, we plan to add more morphometric features and wrappers and we hope that users will also contribute their own morphological measures to the open source code.

Information sharing statement

The presented Python library is released as open-source software and is publicly available on BitBucket[18]. This library is cross-platform and runs on any desktop computer with a working installation of Python and the additional packages Numpy, Scipy and Matplotlib. The downloadable package contain several examples, two wrappers and a hands-on tutorial illustrating how to use the library.

[16] http://btmorph.readthedocs.org/
[17] https://nose.readthedocs.org/en/latest/
[18] https://bitbucket.org/btorb/btmorph