3-D Reconstruction of the Connectome Using Dense Correspondence

Vanessa Tan (vanessa.tan@post.harvard.edu)

January 2014

Abstract

The overall objective of this project was to examine automated 3D reconstruction of the connectome from the perspective of feature correspondence. This document outlines techniques and software used for both 2D segmentation and 3D linking of brain tissue EM images. Previous literature is included for reference.

Contents

1 Problem Overview

With recent advances in electron microscopy (EM), new technologies have enabled the production of high-resolution images of thinly sectioned brain tissue. Manually reconstructing a map of neural circuitry (*connectome*) from these images, however, has become a major bottleneck, given the magnitude of new datasets. The development of reliable, automatic algorithms for segmenting and reconstructing neurons in 2D scans would have enormous implications for our understanding of learning and memory.

The data utilized consists of EM image stacks of thinly sliced mouse cortex. Due to the anisotropic data of such datasets – with significantly higher resolution in the x and y dimensions than in the z dimension – the 3D segmentation process is split into two tasks: first segmenting the individual 2D frames, and then linking the corresponding 2D objects to reconstruct neurons. A high-level overview of the workflow is shown in Figure 1.

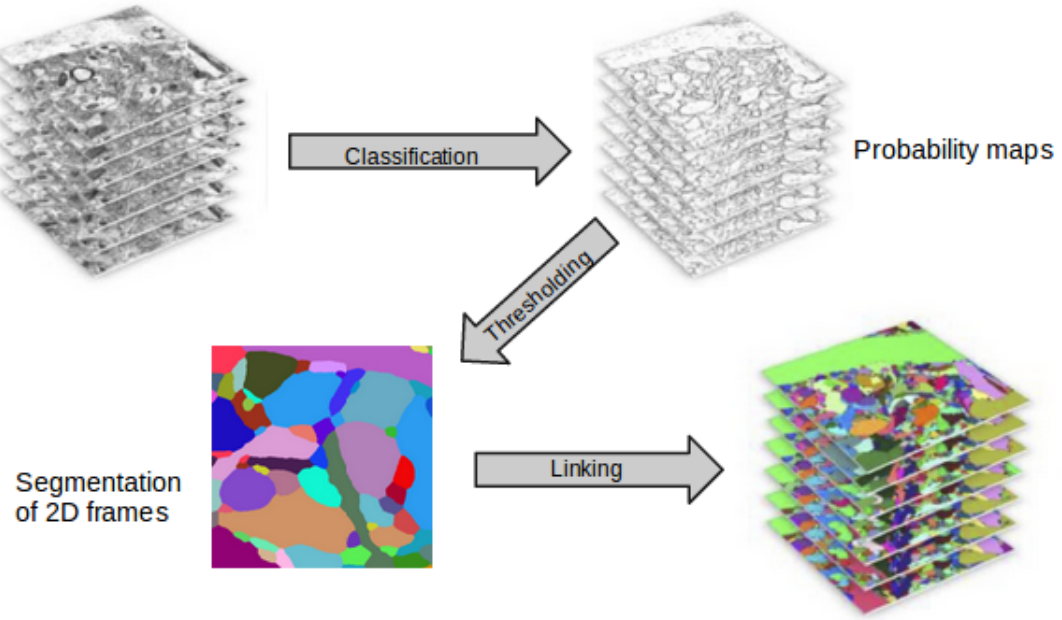


Figure 1: High-level algorithm for 3-D segmentation. Image source: [2].

2 Datasets

2.1 ISBI 2013

The first of two datasets utilized, referred to as here as ISBI, was publicly released by researchers at MIT in conjunction with the IEEE International Symposium on Biomedical Imaging for the

"ISBI Challenge: 3D segmentation of neurites in EM images" [1]. Of the 200 1024x1024 pixel mouse cortex images included, a training set was included for 100, consisting of 2D and 3D ground-truth segmentations labeled by human experts. Sample images, along with their corresponding ground truth delineations, are shown in Figures 2 and 3.

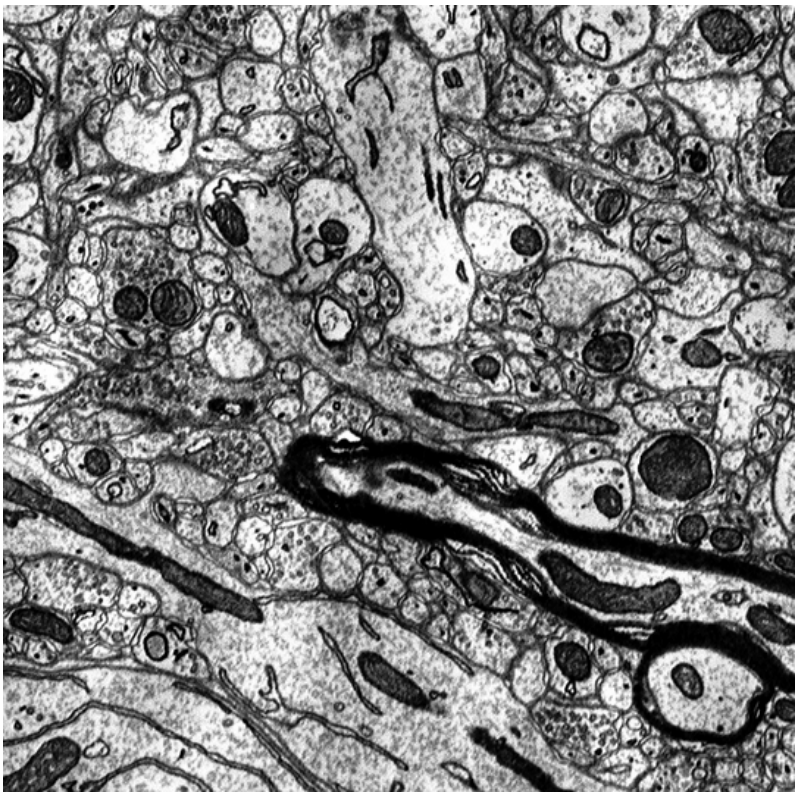
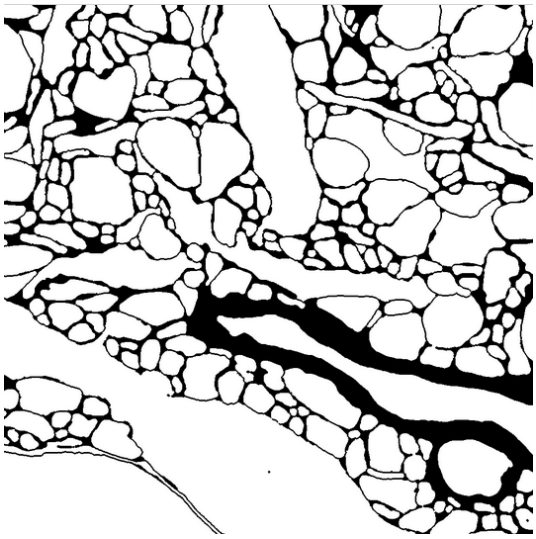
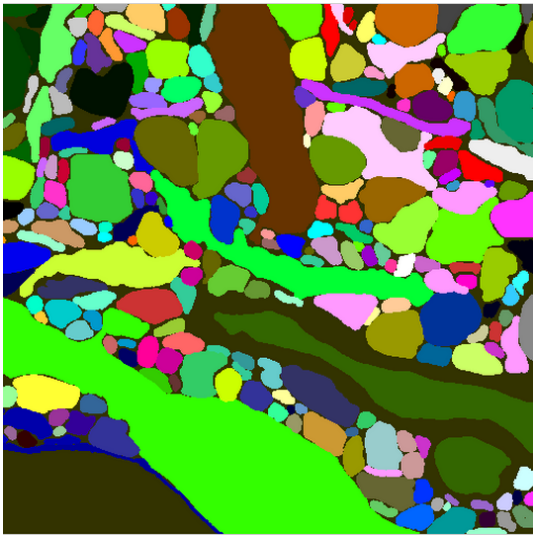


Figure 2: Sample EM image frame from the ISBI 2013 dataset.



(a) 2D segmentation of intracellular space vs. other.



(b) 3D segmentation after final labelings.

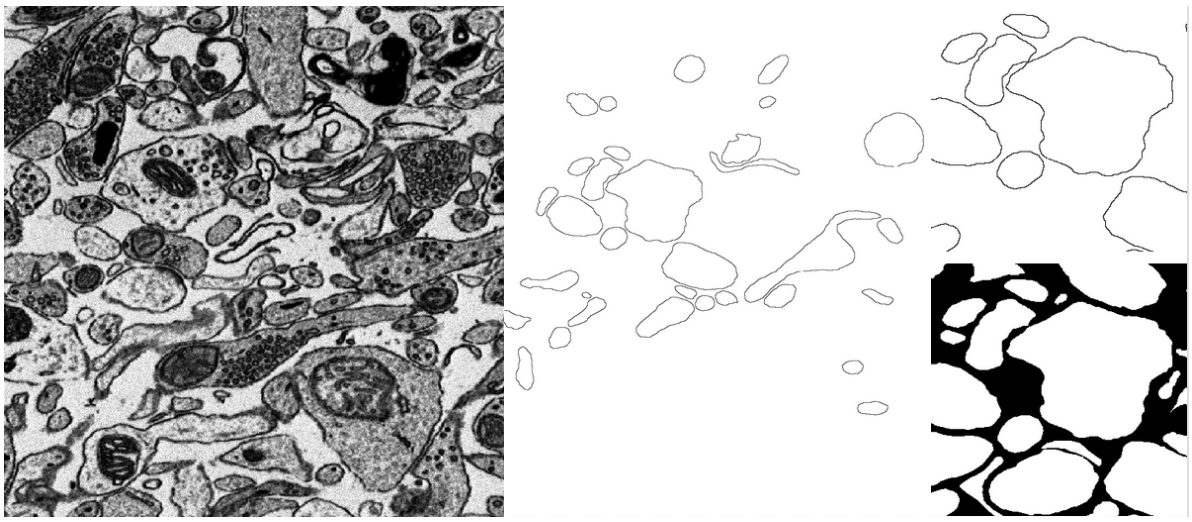
Figure 3: Ground truth labelings for the ISBI dataset.

Note that the images from the ISBI dataset lack large amounts of extracellular space separating each cell. Furthermore, they are relatively artifact-free.

The original images (as PNG files) are located in the `isbi_2013` directory and are also available at the ISBI 2013 Challenge website. For our purposes, the directory `isbi_merged` contains equivalent images merging extracellular space and cell membrane into one label.

2.2 ECS

Recent imaging techniques have been able to preserve the natural spacing between cells, resulting in more visually distinct objects to the human eye. Referred to here as *ECS*, a small dataset of 30 sparsely labeled images has been made available, including a combination of both expert and non-expert annotation. Due to the minimal amount of ground truth, only a 350x350px portion of the stack was used, and I contributed some missing annotation. An example image with annotations are shown in Figure 4.



(a) Sample ECS image frame. (b) Partial membrane labeling with 350x350px insets.

Figure 4: Sample ECS image frame and corresponding ground truth labeling.

Original and cropped images are available in the `ecs` directory. In addition, more data of this variety is being made available by the Lichtman lab, moving away from images as dense as in the ISBI dataset.

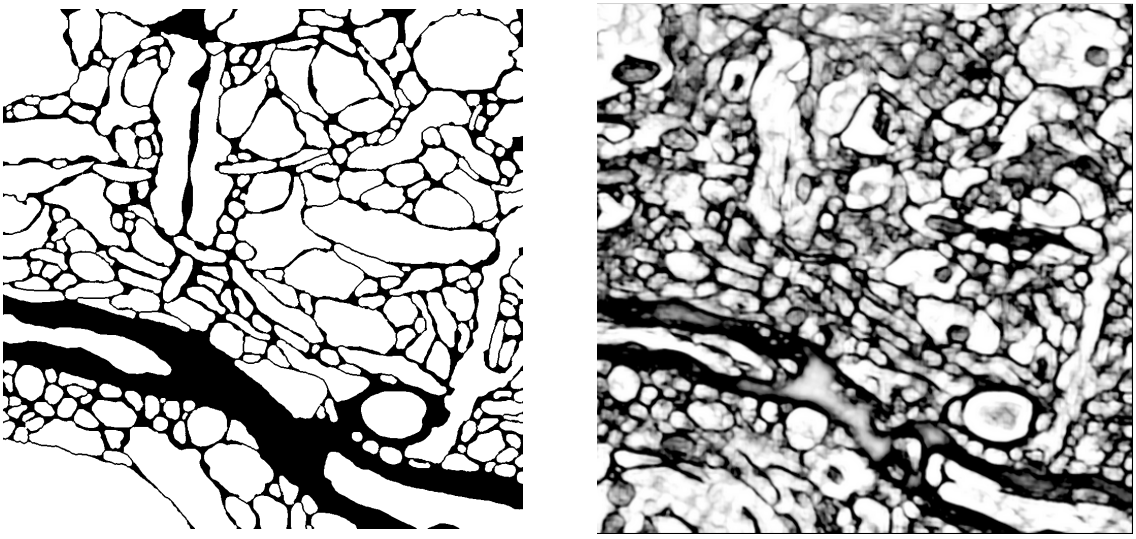
3 2D Segmentation

3.1 SVM Installation and Training

For the 2D segmentation step, I trained the Cox lab SVM and feature generation code included in `connectomics-sandbox.tgz`. In addition to the README instructions, I have included some notes for installation on Odyssey in the Appendix.

Because the SVM uses a binary classification – here, "membrane" vs. "non-membrane" – the output is a probability map where a high probability for a pixel indicates high likelihood of being intracellular space. During training, all extracellular space (which was minimal in the ISBI dataset) was conflated with "membrane" for simplicity.

3.2 Thresholding of Probability Maps



(a) Ground truth membrane labeling.

(b) Probability map output.

Figure 5: Comparison of ground truth and predicted membrane pixels for an image from the ISBI dataset.

A sample probability map output for an image in the ISBI dataset is shown in Figure 5. A respective example is shown for an image from the ECS dataset in Figure 6. Note that because the amount of extracellular space is nontrivial in the ECS dataset, conflating membrane and extracellular space is no longer satisfactory, as intra- and extra-cellular space are not easily distinguished. In the future, the classifier will need to handle three labels in order to achieve good results.

For some informal experiments with the ISBI dataset, both *watershed transforms* and *dual-*

thresholding methods were tested to generate segmentations (`seg2D.m`). Overall, watersheds performed relatively well, but no single threshold produced a satisfactory single segmentation. As a result, the ISBI ground truth 2D segmentations were used in subsequent 3D linking experiments. Using *multiple* possible 2D segmentations in a probabilistic pipeline, as promoted by [2] [3], and [4], is a potential future direction.

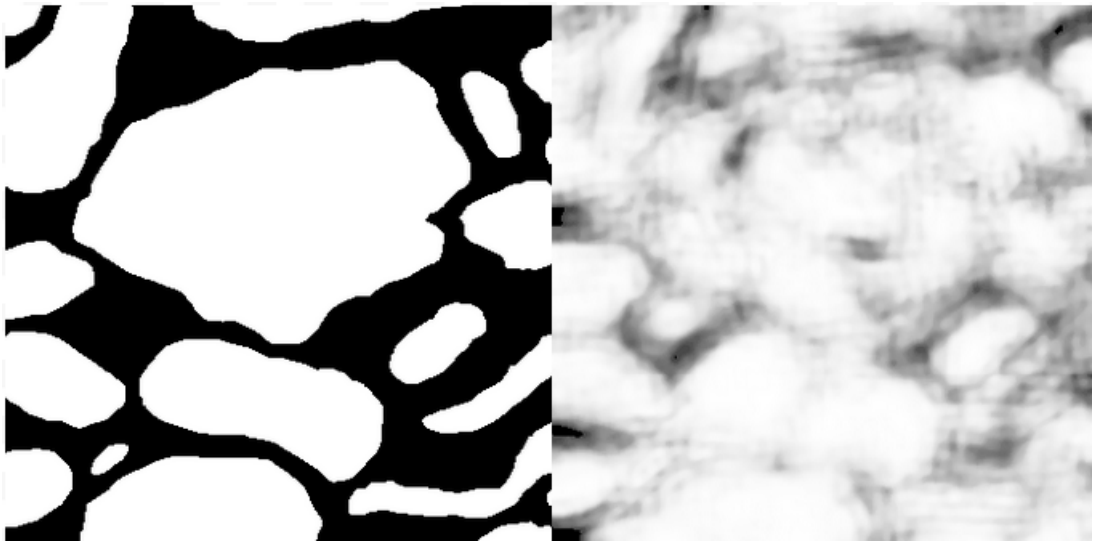


Figure 6: Comparison of ground truth and predicted membrane pixels for an image from the ECS dataset.

4 3D Linking

4.1 KLT Feature Tracking

Initially, my goal was to develop *feature tracking* algorithms for 3D linking. Feature or object tracking is frequently employed in video analysis, and consists of selecting and following the trajectory of an object from frame to frame [7]. The image stack of neuronal pathways can be interpreted as a tracking problem with "motion" of cells in a third spatial dimension instead of motion over time. Because certain previous methods (e.g. [2]) chose to ignore intracellular texture-based information, including the presence of organelles and vesicles, this feature tracking approach aimed to take advantage of such information.

I began by adapting the *KLT feature tracking* algorithm [8], using the C implementation of [9]. Additionally, I experimented with a variation of this tracking framework using SIFT feature vectors for point detection [10]. For both methods, I selected features to track for each

pair of frames, computing how frequently the correspondences correctly linked 2D components belonging to the same 3D object (Figure 7). While both methods performed comparably well linking most objects, they performed poorly with respect to smaller objects, achieving no greater than 65% accuracy on the bottom 25% of objects ranked by size. This posed a significant challenge, since many elongated, thin structures (known as "dendritic spine necks") appear in the data.

The following files pertain to experiments with KLT tracking.

- `klt-orig.c`: Code for standard KLT feature tracking. Parameters: `-n [num frames] -w [window size] -e [min eigenvalue] -s [search range]`. Outputs: feature list files of correspondence between pairs of frames, in `klt_features/features{frame1}-{frame2}.txt`.

- `klt-sift.c`. Code for KLT tracking using SIFT feature selection. Parameters: `-n [num frames] -w [window size] -s [search range]`. Outputs: feature list files of correspondence between pairs of frames, in `sift_features/features{frame1}-{frame2}.txt`.

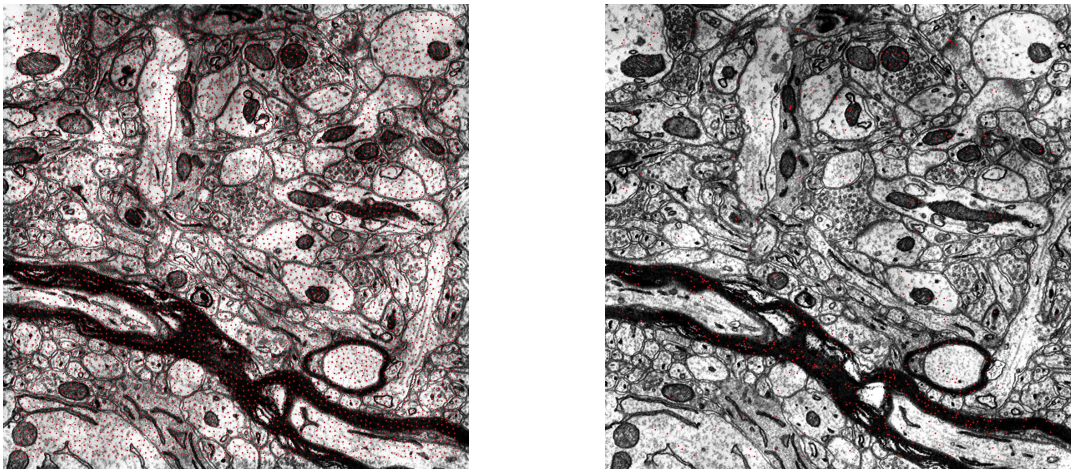
- `extractFeatures.py`. Reformats KLT-outputted feature lists (in .txt format) into .csv format. Parameters: `--feat [klt|sift] --numFeat [num frames]`. Outputs: organized feature lists in `klt_features/features{frame1}-{frame2}.csv`.

- `matlab_script.sh`. Helper script to run Matlab scripts from the command line, without the GUI. Note that arguments are read in as *strings* and may need to be converted (e.g. to ints) within the Matlab code.

- `evalTrackedPoints.m`. Evaluates the accuracy of correspondences reported by tracking. Currently defined against the ISBI 2013 ground truth. Parameter: `[klt|sift]`. Outputs: metrics recorded to `[klt|sift]-metrics.txt` (total number of correct correspondences, percent correct correspondences, percent of objects "hit" by feature selection, percent of objects correctly tracked by at least one point).

- `tracking-klt-batch.sh`: Runs `klt-orig.c`, `extractFeatures.py`, and `evalTrackedPoints.m` with various parameters. Used for parameter tuning. Results stored in `klt-metrics.txt`.

- `tracking-sift-batch.sh`: Runs `klt-sift.c`, `extractFeatures.py`, and `evalTrackedPoints.m`. Results stored in `sift-metrics.txt`.



(a) Interest points identified by KLT tracker. (b) Tracked points from subset of frame (a).

Figure 7: KLT-tracked correspondences in frame (b) for a subset of points from frame (a).

4.2 Optical Flow and SIFT Flow

Initial experiments with KLT revealed that tracking of small objects was maximized by increasing the density of tracked keypoints, without greatly sacrificing overall accuracy. As a result, I turned to approaches for *dense correspondence*. *Optical flow* algorithms, rather than selecting a few keypoints to track, compute the pixelwise motion of the entire image.

Utilizing a variation of this method, *SIFT flow* [6], achieved greater correspondence accuracy than tracking methods and did not miss small structures. The SIFT flow package that accompanies Liu's paper has been downloaded to the directory `SIFTflow`. See the original README for instructions on how to recompile for a different operating system.

A number of experiments with SIFT flow are contained within the `flow_exp` directory. In `siftflow_test.m`, the "accuracy" of SIFT flow correspondences is computed for every pair of frames; in addition, this file can produce flow field images and visualizations of "error" (where an erroneous correspondence is a pair of two points in neighboring frames that are not part of the same ground truth object). `siftflow_multframes.m` is used to produce visualizations of additive flow (e.g. the flow vector of a point 10 frames later).

4.3 Global Graphical Linking Approach

Using flow-based correspondences as indicators for links between frames, I implemented and performed initial testing on a graphical model that determines 3D objects from 2D components. At a high level, the graph includes an edge between every 2D object in one frame and

every 2D object in the adjacent frame; the edge weights between two objects are determined by the proportion of pixelwise correspondences connecting those two objects. Edge weights exceeding a threshold are deemed connections, and the graph is appropriately colored so that objects connected by an edge have the same color. For a more detailed description of the edge weight computations, see Figure 8.

C_i, C_j are 2D objects in frames a and b, respectively;
$\nabla I^{a \to b}$ indicates the flow field between frames a and b;
$\ell_a(x)$ indicates the label of pixel x in frame a.

$$G = (V, E);\ C_i, C_j \in V;\ w_{ij} \in E$$

$$w_{ij} = \frac{p^{i \to j} + p^{j \to i}}{\#C_i + \#C_j}$$

$$\text{with } p^{i \to j} = \sum_{x \in C_i} \mathbb{1}(x, i, j, a, b),$$

$$\mathbb{1}(x, i, j, a, b) = \begin{cases} 1 & : \ell_a(x) = \ell_b(x + \nabla I^{a \to b}(x)) \\ 0 & : \text{otherwise} \end{cases}$$

Figure 8: Equations for computing edge weights (w_{ij}).

Initially, the flow field was generated using SIFT flow as described above. In additional experiments, SIFT features were replaced with Cox lab features (within the SIFT flow algorithm).

The following files pertain to the graphical segmentation approach using both SIFT features and Cox lab features. More detailed documentation is contained within individual Matlab files.

- `seg3D_graphical.m`. Given original EM images and a stack of labels representing a 2D segmentation performs 3D linking using the method described above and returns a stack of labels representing a 3D segmentation.

- `slmsimple_args.py`. Minor modifications made to original file `slmsimple.py`, provided by Kenyon Tsai. Generates Cox lab features for a specified file (and size), with the resulting feature map stored in `fmap.mat` for use in Matlab. The the network based on `train_model.txt`.

- `fullfm.m`. Minor modification of `dispfm.m` provided by Kenyon Tsai, with the network based on `train_model.txt`. Given the results from `slmsimple_args.py` located in `fmap.mat`, the weights are interpolated to produce a full sized feature map (one vector of length 128 per pixel). Some border artifacts result and must be cropped. The feature map can be easily visualized by displaying only one layer.

- `coxlab_feat.m`. Overall Matlab function to generate Cox lab features; calls the Python script `slmsimple_args.py` and calls `fullfm` on the result to produce a full feature map for an image.

- `num_splits_merges.m`. Given a stack of predicted labels and the ground truth labels, computes four metrics: rand error, variation of information, split error, and merge error (see next section).

- `eval_seg3D.m`. The higher level function to predict a 3D segmentation for ISBI 2013 data (using ground truth 2D segmentations) and evaluate based on the metrics described below. Currently, this file needs to be edited in order to change the dataset.

4.4 Evaluation

For evaluating the quality of 3D segmentation predictions, four metrics were used:

- *Rand error.* The file `SNEMI3D_metrics.m`, which computes the official metric "rand error," is provided on the ISBI 2013 Challenge website [1]. A short description is included in the Matlab file.

- *Variation of information.* This measure of similarity between clusterings is also used as a metric in [3].

- *Split and merge error.* Follows the definition of split and merge errors used in [5]:

  > *A perfect solution would assign exactly one label per ground truth cluster. For each additional label a split error is counted. A merge error occurs when two ground truth clusters are assigned the same label. If two ground truth clusters are merged more than once, we follow the definition of [12] and count this as one error as the same two objects are involved.*

"Split/merge error" will refer to the average number of splits or merges *per object*, not the total count.

In addition, `vis3D.m` generates a 3D visualization of (a subset of) objects in any stack of labels. Two examples are shown in Figure 9.

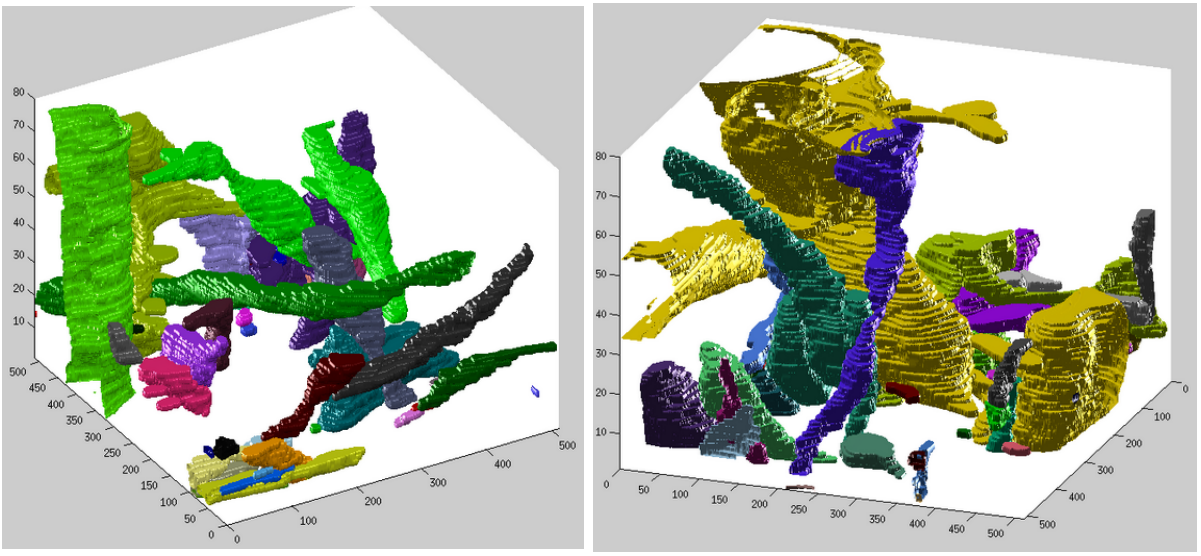


Figure 9: Two partial visualizations of reconstructed neurites.

5 Misc. Notes

- Some original TIFF files (from the ISBI dataset) were omitted from the repository, due to exceeding the GitHub maximum file size. The equivalent stack should be available as individual PNG files. However, a small number of files may reference the TIFF images.

References

[1] Shaar, Nadar. "SNEMI3D: 3D Segmentation of Neurites in EM Images." *IEEE International Symposium on Biomedical Imaging*, 2013. Web. http://brainiac2.mit.edu/SNEMI3D. 8 Nov. 2013.

[2] Kaynig, Verena, Amelio Vazquez-Reina, Seymour Knowles-Barley, Mike Roberts, Thouis R. Jones, Narayanan Kasthuri, Eric Miller, Jeff Lichtman, and Hanspeter Pfister. "Large-scale automatic reconstruction of neuronal processes from electron microscopy images." *arXiv preprint arXiv:1303.7186* (2013).

[3] Vazquez-Reina, Amelio, Michael Gelbart, Daniel Huang, Jeff Lichtman, Eric Miller, and Hanspeter Pfister. "Segmentation fusion for connectomics." In *2011 IEEE International Conference on Computer Vision (ICCV)*, pp. 177-184. IEEE, 2011.

[4] Funke, Jan, Bjoern Andres, Fred A. Hamprecht, Albert Cardona, and Matthew Cook. "Efficient automatic 3D-reconstruction of branching neurons from EM data." In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 1004-1011. IEEE, 2012.

[5] Kaynig, Verena, Thomas J. Fuchs, and Joachim M. Buhmann. "Geometrical consistent 3D tracing of neuronal processes in ssTEM data." In *Medical Image Computing and Computer-Assisted Intervention* 2010, pp. 209-216. Springer Berlin Heidelberg, 2010.

[6] Liu, Ce, Jenny Yuen, and Antonio Torralba. SIFT flow: Dense correspondence across scenes and its applications. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 33.5, pp. 978-994, 2011.

[7] Yilmaz, Alper, Omar Javed, and Mubarak Shah. Object tracking: A survey. In *ACM Computing Surveys (CSUR)* 38, no. 4.13, 2006.

[8] Shi, Jianbo, and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593-600, 1994.

[9] Birchfield, Stan. "KLT: An Implementation of the Kanade-Lucas-Tomasi Feature Tracker." http://www.ces.clemson.edu/ stb/klt/.

[10] Lowe, David G. Distinctive image features from scale-invariant keypoints. In *International Journal of Computer Vision* 60, no. 2, pp. 91-110, 2004.

Appendix: Odyssey build notes

- Modules to load: `python-2.7.3, gcc-latest, cython-0.18, numexpr-2.0`
- For all python setup commands, e.g. scikits.learn, etc.:
  `pip install --install-option="--prefix=/n/home08/vtan/" packagename`
- For "freeimage" dependency:
  - Built Freeimage 3.13.4 from source (3.13.1 from `pip install smc.freeimage` does NOT suffice)
  - In `Source/OpenEXR/IlmImf/ImfAutoArray.h`, add `#include <string.h>` (for memset)
  - Edit Makefile.gnu and change INCDIR and INSTALLDIR at the top to be local directories; then under "install," remove all "-o root" and comment out "ldconfig" line
- Built LCMS 2.3 (NOT 2.4) from source
- Need to pip install PIL
- Commented out instances of "from pymongo import..." (not using MongoDB)
- Set the default Fiji exe path at
  `dependencies/bangmetric/bangmetric/wildwest/isbi12/fiji_metrics.py`
  (in case environment variable `FIJI_EXE_PATH` is not set)
- In `bangreadout/bangreadout/setup.py` and `bangreadout/sthor/sthor/operation/setup.py`, comment out all `-fopenmp` compile flags and run setup again