Segment (Ver.2017-06) Manual

Allison Hsiang

2017-09-18

I Introduction

Segment is the image segmentation module of the *AutoMorph* software package, which was developed by Pincelli Hull and team [1]. *AutoMorph* is available for download on GitHub. The *segment* module is described in detail in Hsiang *et al.* [2].

Segment takes as input a series of z-stack images containing light objects on a dark background and chops out each individual object (contingent on user-provided threshold and min/max size values) into its own stack of images.

II Technical Specification

The z-stack images that are used as input for *segment* must be numbered in order of their height (*e.g.*, image001 is at height 0, image002 is at height 1, image003 is at height 2, image004 is at height 3, etc.). The user is able to test various threshold values and output low-resolution JPGs of the entire input field of view with segmented objects indicated by a red box (Fig. 1) and an automatically assigned object number (Fig. 2) in `sample` mode.

Once the thresholding parameter has been optimized in `sample` mode, the user can then run *segment* in `final` mode, which will chop out the objects identified using the specified threshold value, saving each z-stack to its individual folder labeled by object number (as determined by *segment*'s algorithm). *Segment* also generates a label (Fig. 3) containing metadata for each object that is appended to every z-stack image for a given object. In addition, under `final` mode, *segment* will output a text file listing the parameters used for the run and the low-resolution JPG overview image described above.

Segment, like all *AutoMorph* modules, is run using the command line. On Mac OSX, you must install the GNU Coreutils command line tools (more information here) in order to use the essential UNIX commands (*e.g.*, 'ls' and 'cat') and run

the *segment* binary executable. If you are unfamiliar with the command line, we recommend searching for introductory tutorials online and familiarizing yourself before diving in (here are some suggestions). A good golden rule when dealing with the command line as a beginner is: never input a command if you don't know exactly what it will do!

Segment is run using a plain-text input file, referred to here as a 'control file'. Explanations for the user-controlled variables in the control file, and how to set them, are presented in section V.

III Installation

III.1 Prerequisites

Segment is written in Python and currently is only compatible with Python 2.7. Compatibility with Python 3.x is under development for future release. The following Python modules are required:

- Python v.2.7
- scikit-image
- scipy
- numpy
- pillow
- opencv
- tifffile

We recommend installing the Anaconda distribution of Python, which features easy package managing/installation. Once Anaconda is installed, users need only manually install the opencv and tifffile modules:

```
$ conda install opencv -c conda-forge
```

```
$ conda install tifffile -c conda-forge
```

Note that the -c conda-forge flag is necessary for opencv and tifffile to install correctly. Consult the Troubleshooting section of this manual (section VII) for possible issues that may arise when installing opencv.

If you choose to install the required Python modules manually, note that scikit-image is named skimage in Python repositories.

III.2 Setup

You should first download *AutoMorph* from the GitHub repository by clicking the green "Clone or download" button on the right side of the screen and selecting the "Download ZIP" option. Once you have downloaded the *AutoMorph* software package, you will find the *segment* executable in the AutoMorph/segment. Within this folder you will also find a clean control file; the control file is used to specify the various parameters necessary for running *segment*.

We recommend adding the *segment* executable to your path, so that *segment* can be called from anywhere in your system. To do this on Mac OSX, open the Terminal program (located at /Applications/Utilities/Terminal), and type the following command at the prompt:

```
$ ln -s AMPATH/AutoMorph/segment/segment /usr/local/bin
```

where AMPATH is the location of your *AutoMorph* installation. For example, if *AutoMorph* is located in /Applications, the full command would be:

```
$ ln -s /Applications/AutoMorph/segment/segment /usr/local/bin
```

If you are using Mac OS X, you may encounter a 'Permission denied' error when attempting to create the symbolic link. If this happens, you should use the 'sudo' command like so:

```
$ sudo ln -s /Applications/AutoMorph/segment/segment /usr/local/bin
```

You will be prompted to enter your password; once you have done so, the symbolic link will be created. Note that you should not use 'sudo' unless you know exactly what you are doing!

You can now use *segment* from any location on your computer. Note that you should use absolute paths when creating symbolic links, not relative ones.

IV Quick Run

Once *segment* is installed, it can be run via the command line using the following command (assuming the *segment* executable is in your path):

```
$ segment <path to control file>
```

While running, *segment* will output status messages, including what mode it is running under (**sample** or **final**) and the number of objects found under the current parameters.

V Control File and Parameters

The control file serves as the means by which the user supplies the necessary parameters for *segment* to operate. Some parameters require user input while others can left blank to allow *segment* to use default values. It is recommended that the user read through this section and understand what all the parameters do, and how to set them, before using *segment*.

The control file should be edited using a plain text editor (*e.g.*, BBEdit for Mac OSX) to avoid interpretation issues and must use Unix (LF) encoding.

For parameters that are not required, the default value can be used by leaving the parameter blank, like so:

```
out_directory =
```

The default values of *segment* tend to work well with the slides of foraminifera produced by the Hull lab (see example files in AutoMorph/example_datasets); users are encouraged to use these as a baseline against which to test their own images and parameter values.

A listing of the parameters for *segment* follows, with required parameters marked with an *:

V.1 Global Parameters

*** directory**: The full path to the input directory containing the z-stack images to be processed. This cannot be a relative path.

*** output**: The full path of the folder to which *segment*'s output will be saved. If the folder does not exist, *run3dmorph* will create it. A subfolder (either named 'sample' or 'final' depending on if *segment* is run in `sample` or `final` mode, respectively; see **mode**, below) will be created in the specified directory. This cannot be a relative path.

*** pixel_size_x**: The number of units (as specified by the user under **Unit** in the **Labeling Parameters** (V.2) per pixel in the input images in the x-direction (width), usually determined by the equipment and settings used to capture the original images. For instance, if one pixel in the input images is equal to 0.975 microns, and the pixels are square, then this parameter would be set as:

```
pixel_size_x = 0.975
```

*** pixel_size_y**: Same as above, but in the y-direction (height).

*** mode**: Set as either `sample` or `final`. `Sample` mode is used to optimize threshold and min-max filtering sizes and outputs a low-resolution JPG of all objects identified by *segment* given the test set-

tings. `Final` mode is used when the parameters are optimized; running under `final` mode will output all segmented objects in individual folders in addition to the low-resolution JPG.

skip_last_plane: A boolean (*i.e.*, True or False) specifying whether the last plane in the input z-stack should be excluded in the segmentation process or not. This is used when the last plane is not a true focal plane (for instance, an extended depth of field (EDF) image generated automatically by Surveyor software). By default, this parameter is set to True.

unit: The base unit of measure used in the pixel size conversion; the default value is microns.

V.2 File Format Parameters

input_ext: The file extension for the input images to *segment*. The default setting is `tif`.

output_ext: The file extension for the output object images that will be written (*i.e.*, the individual object z-stack images). The default setting is `tif`.

V.3 Object Recognition Parameters

threshold: The threshold value used to convert the input images into black and white and detect objects. When running *segment* in `final` mode, this should be a single value, *e.g.*:

```
threshold = 0.16
```

When running *segment* in `sample` mode, a range of threshold values can be selected in the following format:

```
threshold = L - H by i
```

where L and H are the low and high end of the range of desired threshold values, respectively, and i is the increment value between each threshold value to be tested. For example, if the user wishes to test threshold values between 0.16 and 0.20 with an increment value of 0.01, they would enter:

```
threshold = 0.16 - 0.20 by 0.01
```

which would test threshold values of 0.16, 0.17, 0.18, 0.19, and 0.20. The default value of **threshold** is 0.20.

minimum_size: The minimum valid width of an object identified using the current threshold setting in the user-defined **unit** (*e.g.*, microns). This parameter is used to filter out background light scatter by excluding identified objects with widths smaller than the specified size. Like the **threshold** parameter, this parameter can be set as a range of values, which will result in segmenting results by each combination of threshold/minimum size values. The default value is 100.

maximum_size: As above, but the maximum valid width. This parameter is used to filter out border effects (e.g., rims, occluded objects, etc.) and other sources of large-scale noise. The default value is 2000.

V.4 Drawing Parameters

box_thickness: A number that sets the thickness (in pixels) of the lines in the red boxes drawn around identified objects in the low-resolution JPG overview images. The default value is 20; the user may change this depending on the images to optimize visibility/clarity of the boxes.

scale_bar_length: The length of the scale bar (in the unit specifed under the **unit** parameter) that is outputted in the metadata labels generated by *segment*. A smaller scale bar that is 1/4 this size is also drawn. The default value is 100.

V.5 Metadata Parameters

*** id**: A name that serves as an identifier for the sample being processed. This name is used as a prefix in the names of output files and labels generated by *segment*.

age: A tag used for reporting the geological age estimation of the sample being processed; this will be printed to the metadata labels generated by *segment*. The default value is `Unspecified age`.

source: A tag used to provide a reference or indicate the source of the age estimation data specified under **age**; this will be printed to the metadata labels generated by *segment*. The default value is `Unspecified source`.

location: A tag indicating the processing location of the images (*e.g.*, Yale Peabody Museum); this will be printed to the metadata labels generated by *segment*. The default value is `Unspecified location`.

author: A tag indicating the group or institution who performed the image processing; this will be printed to the metadata labels generated by *segment*. The default value is empty.

catalog_prefix: a tag indicating the library/museum catalog number prefix (if applicable) for the images being processed; this will be printed to the metadata labels generated by *segment* in combination with the specified **ID**. For example, if the catalog prefix is 'YPM IP', and the ID number is '5067', then the label will identify the sample as 'YPM IP 5067'. The default value is empty.

VI Hands-On Example Run

A set of example images (and associated XML files generated by the Leica DFC450 camera used to generate the images) that can be used to test *segment* can be downloaded on Zenodo here. A mini dataset containing much smaller images is also available in AutoMorph/example_datasets. For the following tutorial, we assume the user is using the larger dataset from Zenodo.

A control file to use with these test images is located at AutoMorph/segment/examples/segment_control_4sq_example.txt. The user need only change the **directory** and **output** parameters to run the example. Step-by-step instructions – which assume that the user's installation of *AutoMorph* is in /Applications, that the user has added the *segment* folder to the path, and that the example dataset is in /Downloads – follow:

1. Open segment_control_4sq_example.txt in a plain-text editor and change the **directory** and **output** parameters to the appropriate paths, and then save the file:

   ```
   directory = /Downloads/segment_4sq_example_slide
   output = /Downloads/segment_4sq_example_slide
   ```

2. At the command line prompt, navigate to the folder containing the control file:

   ```
   $ cd /Applications/AutoMorph/segment/examples
   ```

3. Start *segment* by entering the following command:

   ```
   $ segment segment_control_4sq_example.txt
   ```

The provided control file is set to run in `final` mode; as such, upon running, *segment* will proceed to identify the individual objects in the input slide and chop them out into individual object z-stacks. When *segment* is done running, the command line prompt will reappear. The output files will be located in a folder named 'final' nested inside the directory specified for **directory**. At a threshold value of 0.16 and a valid size range of 100-2000 microns, *segment* should have recognized and extracted a total of 28 objects.

VII Troubleshooting

The most common errors that arise during usage of *segment* occur due to inappropriate control file settings. In particular, *segment* will often quit because it is unable to find any input images; this will be apparent from the status messages outputted by *segment* (*i.e.*, *segment* will say that 0 images were found). If this occurs, the user is advised to double-check their **directory** path, **input_ext**, and **output_ext**, ensuring they are correct.

If *segment* finds the input images but does not extract any objects, the problem likely lies with the **threshold** parameter, **minimum_size** and/or **maximum_size** parameters, or all three. In this case, the user is advised to try out different values for these parameters and determine the optimal values via trial and error.

VII.1 Issues with Installing Opencv

You may run into issues installing opencv with Anaconda, particularly if your system has previous installations of Python and/or other package managers such as Homebrew. The following are issues and solutions that we have come across in-house when installing opencv on Mac OS X.

VII.1.1 Homebrew

If you have previously installed Homebrew, you can install opencv by tapping the homebrew/science repository, like so:

```
$ brew tap homebrew/science
$ brew install opencv
```

VII.1.2 ImportError

If you install opencv with Anaconda, you may encounter an ImportError when trying to import the cv2 library. This may be the result of certain libraries being missing, and you will see an error message similar to the following:

```
Traceback (most recent call last):
  File "/usr/local/bin/segment", line 4, in <module>
    import images
  File "/Automorph/AutoMorph-master/segment/images.py", line 9, in <module>
    import cv2
ImportError: [...] : Library not loaded: @rpath/libopenblasp-r0.2.19.dylib
    Referenced from: somepath/libopencv_hdf.3.2.0.dylib
    Reason: image not found
```

This issue occurs because the dlib library is missing. You can install this library via the following command:

```
$ conda install dlib -c conda-forge
```

You may then need to update other packages (*e.g.*, numpy). Update all your packages like so:

```
$ conda update --all -c conda-forge
```

Alternatively, you may encounter the following error message:

```
ImportError: libopenblas.so.0: cannot open shared object file: No such file
or directory
```

This error can be solved by installing the Openblas library:

```
$ conda install openblas
```

You should now be able to use opencv without issue.

References

[1] *AutoMorph* (https://github.com/HullLab/AutoMorph)

[2] Hsiang AY, Nelson K, Elder LE, Sibert EC, Kahanamoku SS, Burke JE, Kelly A, Liu Y, Hull PM. AutoMorph: Accelerating community morphometrics with 2D and 3D image processing and shape extraction. **Methods in Ecology and Evolution**. *In revision.*

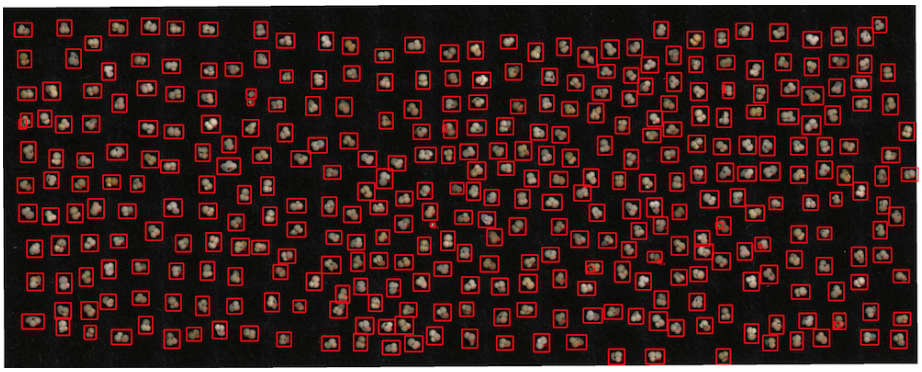

Figure 1: Example of *segment*'s low-resolution JPG output, showing the complete field of view from images inputted into *segment*, with segmented objects indicated by red boxes and numbering (numbering not visible at this scale; see Fig. 2).

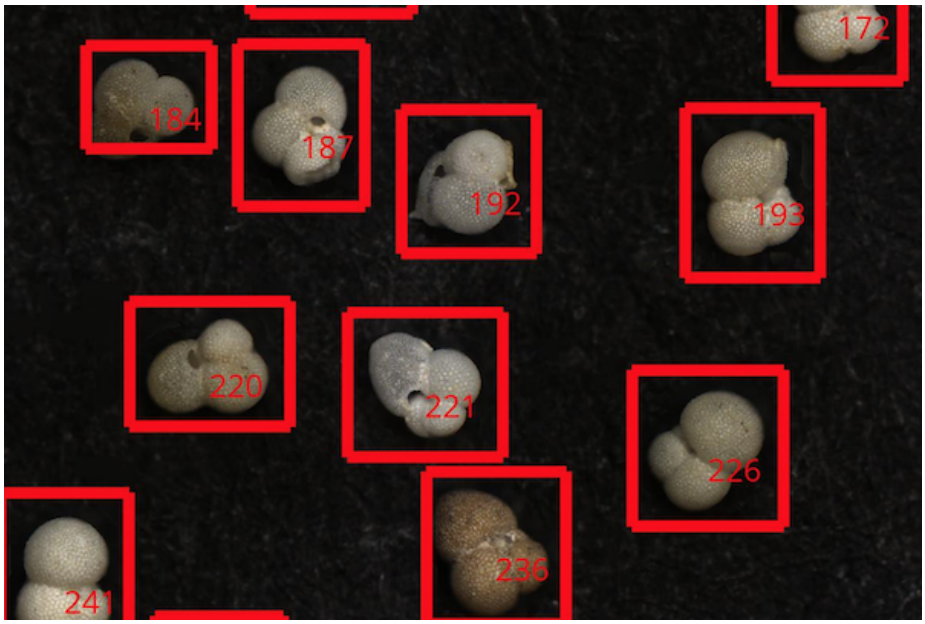

Figure 2: Close-up of *segment*'s output (Fig. 1), showing boxes around individual objects isolated by *segment* and automatic object numbering.

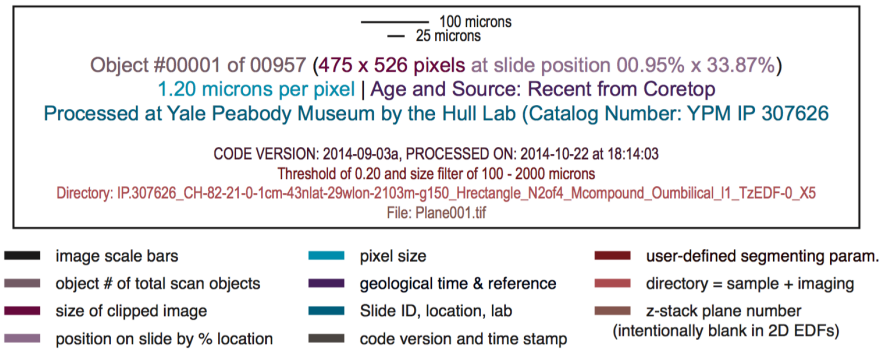

Figure 3: Example metadata label generated by *segment*.