Documentation for DISCO: the swain lab segmentation software

Elco Bakker

January 8, 2018

Contents

0.1 Introduction

Welcome to the DISCO software: a software for segmenting yeast cells in trap like microfluidic devices, tracking them and viewing/editing the results. First, a few notes:

- The software requires that the swain lab general functions package also be installed, which can be found at https://github.com/pswain/GeneralMatlabFunctions.
- The best description of the underlying algorithm is the associated paper published at https://academic.oup.com/bioinformatics/article-abstract/doi/10.1093/bioinformatics/btx550/4103414?redirected
- The software was developed in the swain lab, and you should feel free to contact them with help and support in using the software (http://swainlab.bio.ed.ac.uk/).
- It is recommended that you use the software with MATLAB 2015b. Using it with later versions of matlab requires some extra setup steps that are detailed in section 0.5.

There are broadly two parts to the software: using the software to segment images and training the software to segment unseen image types. The first, segmenting, is relatively straightforward and is how you will use the software day to day. The second, training, requires a little more expertise and effort but should only need to be done once for a given imaging modality/microscope.
We will first describe segmentation, and then describe training, even though it may be necessary for somebody in your group to train a model before anyone is able to segment images.

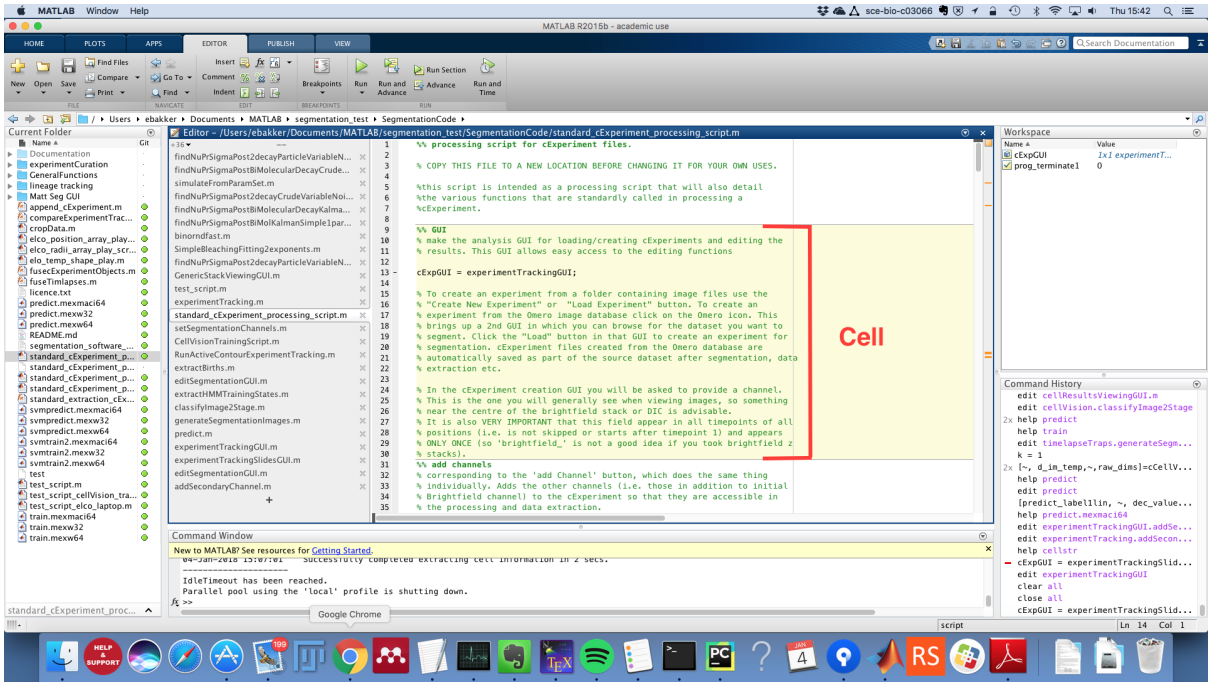

Figure 1: Screen-shot of the standard processing script. The script is broken into cells (indicated) by the %% characters. Each cell contains some code and a description of what the code does, or why you would want to do it (green text). The highlighted (yellow) cell can be run by pressing `cmd Enter` for Mac and `ctrl Enter` for windows.

0.2 Segmenting Timeseries Using the DISCO software

DISCO is a comprehensive (we hope) software for automatically segmenting cells in trap like devices and inspecting/editing the result. The processing is done through a combination of a matlab script, which is run cell by cell, and a collection of GUI's. We have found that this combination allows people to customise their personal work flows and allows the software to be easily updated and maintained. The standard work flow is:

1. The user points the software to the images that make up the experiment and provides some basic information about the experiment.

2. The user checks the identification of traps in the images and sets a number of parameters, mostly by GUI.

3. The software uses a **cellVision Model** and **cellMorphology Model** to automatically identify and track cells in the images.

4. The user can at this stage check and edit the automated cell identification and tracking and select cells to exclude from the data extraction.

5. The software extracts data for the selected cells which can be used in analysis.

To begin open the script called `standard_cExperiment_processing_script.m`[1]; you should see something like the screen shot shown in figure 1.
Run the first cell and a GUI, like that shown in figure 2, should open. This is the primary GUI for processing the software, and is used in combination with the script to process the experiment. By pressing 'h' one can see a help dialogue on all the buttons of the GUI (in general, 'h' will open a help dialogue for GUI's).

[1] in time you will find you want to change parts of this script to suit your own workflow. At this stage, copy the script with a new name and edit this version

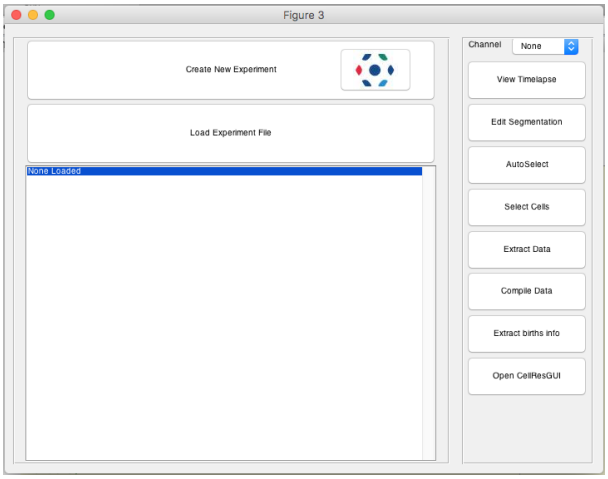

Figure 2: Experiment tracking GUI. This is the primary GUI used in processing experiments. The buttons on the left, are for creating the experiment. The selection screen below is for selecting which positions to process and the buttons on the right are for checking and editing the result of the automated segmentation. By pressing 'h' one can open the help of the GUI.(Note: the colourful Omero Button to the right of Create New Experiment will only appear if you have the Omero software installed and is not necessary for processing experiments.)

To start, press the top left button labelled 'Create New Experiment'. This will open a dialogue by which you point the software at the images you wish to process. The software has been designed for the Swain lab and our image storage structure, which is a top level folder containing one folder for each position imaged during the experiment, with each position folder containing the images for that position. At this stage one needs to select the folder **containing** the position folders. To read about how to get images stored in another way into the correct configuration using ImageJ (or FIJI) see section 0.4.
Now the experiment has been created, the positions can be viewed by pressing the 'View Timelapse' button (the position shown will be the topmost one highlighted on the left). Highlight all the positions you wish to process in the menu on the left and then return to the script. You will now need to run each of the cells in turn until you finally run the cell entitled `Actual long run (Elco standard extraction); run when happy with all the rest!`. This cell will commence the segmentation and data extraction of all the positions.

0.2.1 Notes and pointers:

1. Depending on the length the experiment and the number of position, this last run may take a day or more. Most users run these overnight.

2. The code is parallelised in various places for speed. By default, matlab will use all available cores, which may be frustrating if you want to do other stuff in the meanwhile. You can get matlab to use only some of your cores using the `parpool` command before starting the analysis.

0.2.2 Checking and editing the result

Once the automated segmentation has finished, one can inspect and edit the result in various ways using the buttons on the right of the experiment tracking GUI. By pressing 'h' when the experiment tracking GUI is the focus one can see a brief description of each GUI. When any of the GUI's are open one can get a fuller description by again pressing 'h'.

0.2.3 Accessing the Data

When you are happy with the result, you will want to look at the extracted data. For the DISCO software, this takes the form of numerous statistics extracted for each of the selected channels for each

of the selected cells. To see this go to the command line and type `cExperiment.cellInf`[2]. Each entry in the structure array is a channel, each field is a statistic and each row is the value of that statistic over time for a particular cell. These are stored as sparse arrays, so when you find your desired channel and statistic it is best to copy it to the workspace and run `full(array_name)` to make it a normal array.
To see a description of how each statistic was calculated type `help extractCellDataStandardParfor`.

0.2.4 Diagnosing Poor Performance

If the automated segmentation is performing badly, there are a number of things you can try to improve it. The problem is most likely either due to the cellVision model or the active contour parameters.

cellVision model

You have inspected this as part of the standard script in the section titled `inspect decision image`.One can also see more details of the celLVision output by setting the parameter
`cExperiment.ActiveContourParameters.ActiveContour.visualise = 4`
and running the processing on the first position. If the decision image is not dark and the centres of the cells and bright elsewhere then there is something wrong. You can try changing the channels used for the segmentation by running `cExperiment.setSegmentationChannels`, but if this fails you will need to retrain the cellVision model. It should be noted that the software is generally pretty robust, and retraining should only be necessary if you are using a new imaging modality or a new microscope. If neither of these is the case, you are most likely doing something wrong and it is probably worth contacting the authors or the swain lab (preferably the swain lab) before retraining the cellVision model.

Active Contour Parameters

There are various active contour parameters that can also be played with if the segmentation is not working well. These are stored in various substructures of the structure `cExperiment.ActiveContourParameters`, and I will try and describe the main ones and their effect here. For a description of the use of all the parameters look at `experimentTracking.helpOnActiveContourParameters` (though this is somewhat obscure if you don't know the software quite well).

TrapDetection This is the set of parameters used for detecting the trap. If, when you run the software with `visualise = 4` as described above, the traps are not well blotted out (i.e. either too much or too litte blotting) then you need to change parameters here. One can try changing the channel, and can use negative channel indices to invert the channel. Try and pick a channel with a clear white trap on a black halo.

ActiveContour.alpha This is the shape punishing term applied to new cells. If new cells are don't pay much attention to the image (i.e. are always a similar shape regardless of the image) and are a bit small make this parameter smaller. If new cells are very irregular shapes make this parameter larger.

ActiveContour.beta This controls how much cells are constrained to be the same shape over time. If cells are too rigid (i.e. don't change shape in the way the images suggest) make this parameter smaller. If they change shape too much, make this parameter larger. One can also make this parameter larger if tracking often breaks.

ActiveContour.inflation_weight controls the force that expands cells. If the cells detected are too small, increase this. If they are too large, decrease this (note you should probably try changing alpha first).

CrossCorrelation.ThresholdCellProbability this is the cell probability below which a cell will be discarded as 'not being a real cell'. If a lot of cells are being discarded (as indicated by the 'cells dicarded: ...' text when the software is running) the this is probably too high and should be turned down. If the software persists in keeping cells that don't 'look good', then this is too low and should be increased.

[2]If this is a new instance of matlab you will need to run the first cell of the script again to open the GUI, load the experiment and then run `cExpGUI.cExperiment.cellInf`

If playing with these parameters does not solve the problem, feel free to contact a member of the swain lab for advice in improving performance.

0.2.5 FAQ's

Why do you ask fo the pixel size when creating the experiment? The pixel size is used to determine whether an image has to be scaled to fit with the images used to train the software before processing. For example, our standard cellVision model was trained with 60x image (0.263 μmpixels for our microscope), so if we want to segment 100x images (0.158 μmpixels) the images need to be down sampled by a factor of 0.6 .
The amount of down sampling required is determined by the software by comparing the pixel size submitted when the experiment is created and the pixel size of the experiment used to train the cellVision model. This is done when the traps are selected: the point at which the cellVision and the experiment are committed to each other.
Why do I have to turn my traps to the left? This isn't strictly necessary if you are training a new cellVision model, but it was the orientation in which the first cellVision model was trained and has therefore become our convention. It is easier to share cellVision models between experiments if this convention is maintained.

0.3 Training a cellVision /cellMorphology Model Model for Use With DISCO

The DISCO software uses techniques from supervised machine learning to provide a robust automated segmentation of cell images. This means that before use the software must be trained i.e. provided with a set of images in which cells have been accurately segmented in order to learn the shape and appearance of the cell.

There are two trained components to the DISCO software:

cellVision Model This object is responsible for classifying pixels in an image as either cell edge, cell interior or background. It takes images of the cells (usually a stack of brightfield of phase contrast images), performs transformations to these images to generate a set of features for each pixel and then produces a pixelwise probability of being in the three categories above.

cellMorphologyModel This object encodes the shape of cells, how they change shape over time and how they move in the traps. It receives a collection of curated cell outlines and consecutive timepoints and uses this to learn a probability of a certain cell shape with no reference to the images.

The cellVision model is trained using the script `CellVisionTrainingScript.m` and the cellMorphology model is trained using the `MorphologyModelTrainingScript.m`. In both cases, as with the standard processing, one steps through the well annotated cells of the script to complete the training.
There main work of the training is creating the curated cExperiment (i.e. checking and manually correcting outlines of cells in a collection of images) and can take a few hours for a trained user. This is done at the top of `CellVIsionTrainingScript.m`, and the result can be used for training both the cellVision and cellMorphology model. To get a robust performance you should use images from a number of experiments as explained in the script, but it is not necessary to have more than about 30 timepoints total, so I usually take 10 timepoints (or 5 timepoint pairs if training a cellMorphology model too) from 3 experiments.
There are broadly 2 cases where one would want to train a new set of models:

1. The software is underperforming due to changes in imaging conditions or trap design and you want to retrain an exisiting cellVision model to improve performance.

2. You are training an entirely new cellVision model for a new microscope/imaging modality for which you have no preexisting cellVision model.

In both cases you need to curate a set of cell outlines, but in the first case it is sensible to use an existing cellVision model to do an automated segmentation to correct rather than starting from scratch. In the 2nd case it is often easiest to train a cellVision model on a very small curated data set (5 timepoints) first, and then use this to get an automated segmentation to correct. If you are training an entirely new cellVision model, you will need to set the software to find outlines based on an image transform rather than using the cellVision pixel classification (even if you are just clicking to add cells). How to do this is detailed in the cellVision training script.
It is often not necessary to train both the cellVision and the cellMorphology model. You only need to retrain the cellMorphology model if you are changing trap design, cell type (such that the cell size changes) or magnification (such that the size in the image changes). You might retrain the cellVision model to improve performance on a new microscope or imaging modality without changing the cellMorphology model. If you are only training the cellVision model, and not the cellMorphology model, it is better not to pick timepoints as consecutive pairs (at the time of writing, this meant setting the `pick_pairs` variable to false.)

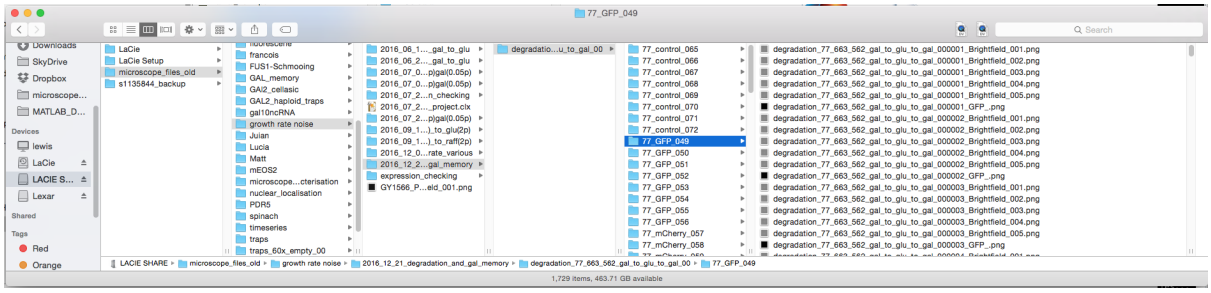

Figure 3: An example of how images are stored in the swain lab image file system. The example shows and experiment called `degradation_77_663_562_gal_to_glu_to_gal_00` in which Brightfield was imaged with 5 z slices and GFP was imaged with only 1.

0.4 Using DISCO with Other Systems

0.4.1 Getting DISCO to read your files

In order to be able to process the images in anyway, DISCO must of course be able to load your files and therefore know where they are.
The DISCO software was designed to be used with files generated by the swain lab microscope control software, which stores files in the form:

`[experiment name]_[6 character timepoint]_[channel name]_[3 character z-slice].png`

with each position being stored in a separate folder. An example is shown in figure 3. This is (I believe) also the convention followed by MicroManager. If you're files are arranged in a different structure an you wish to make the software work, there are two options open to you.

Easy Option: Export Images using ImageJ

The easiest option is to use ImageJ or FIJI to save your images in this format. These programs can handle most image formats and file structures.
First load in an images stack in a way you are happy with. Then save the image stack as an image sequence, choosing a format of PNG and Digits as 6. This will store the image sequence in format appropriate for the software, with channel names given as `c000001,c000002,c000003 ...`
If multiple image stacks are to be processed they can be saved to separate folders in the same parent folder (in imitation of the Swain Lab file structure), allowing the software to be conveniently run on all of them.

Developer Option: Modify `timelapseTraps` Constructor and `timelapseTraps.addChannelsTimelapse` method.

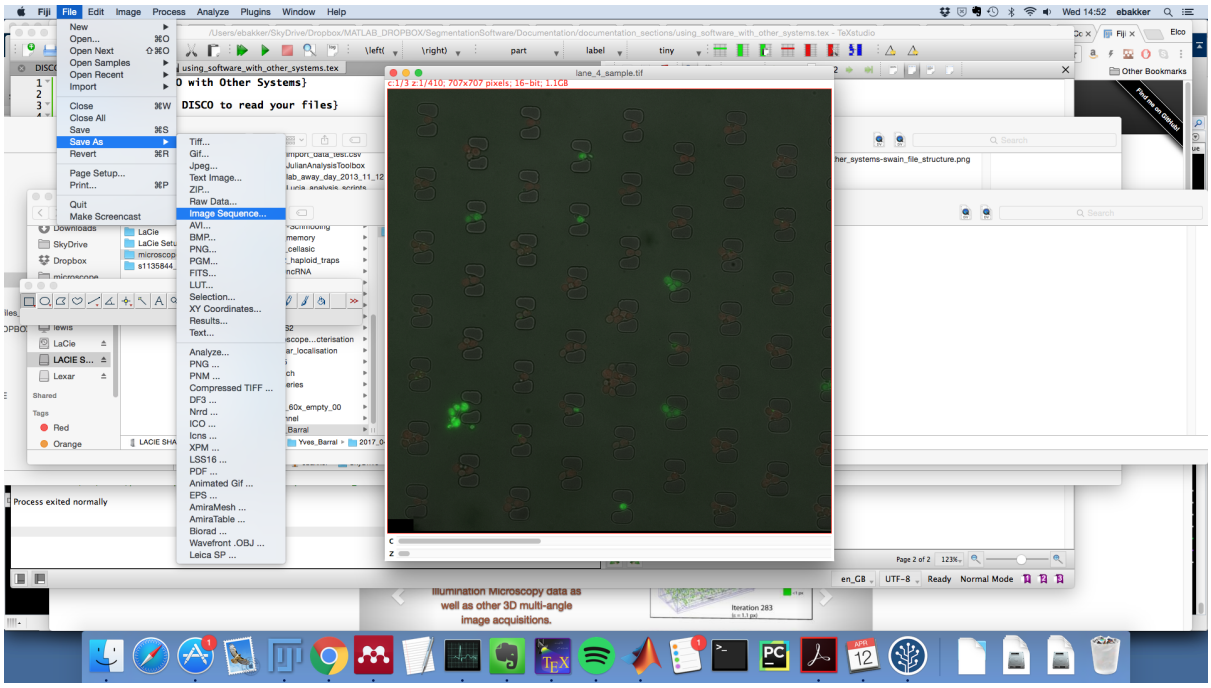

Figure 4: Exporting images for processing from FIJI. Shown is the 'save' options that should be selected for exporting an image stack for processing from FIJI. Selecting this option will open a dialogue box in which 'format' should be set to PNG and 'digits' should be set to 6.

0.5 Using DISCO with Later Versions of Matlab

The core of the cell identification software uses the libsvm library to train and apply linear SVM for pixel classification. The mex files for this are provided, pre-compiled with the software.
Unfortunately the original files that were compiled to create these mex files have been lost, and the libsvm library has moved on such that the function calls have changed. Further, later versions of matlab do not seem to necessarily be able to use mex files compiled by older versions. For this reason, matlab versions beyond 2015b will not be able to use the software without some recoding and retraining.
This has not yet been undertaken, partly due to laziness and partly because recoding the training and classification is a good opportunity to take a serious look at whether some other classifier (random forest, neural networks etc.) wouldn't provide a better result.
When it comes time to retrain, the two parts of the code that will have to be altered are:
`cellVision.classifyImage2Stage`
and the various `cellVision` methods called in `CellVisionTrainingScript.m`:

- `generateTrainingSetTimelapseCellEdge`
- `trainSVMCellToOuterLinear`
- `trainSVMInnerToEdgeLinear`

This is probably all best done by creating a new subclass of `cellVision`, but I leave such structural decisions to the inheritor.

This obviously all goes pretty deep into the code, and as such I recommend liasing with current members of the swain lab, or the other author Matt Crane, when undertaking this.

0.6 Using DISCO when there are no traps

It should be possible to use DISCO when there are no traps present (such as slides, matec dishes or the cellasic device), but since this is rarely used in the lab it is likely to be a little buggy.
To do this the procedure is the same. You **can** use a cellVision and cellMorphology model trained for cells in traps with good results (provided the imaging modality is similar), but there are one or two in the standard script that cells that shouldn't be run and one or two that should be run even though they seem nonsensical given that you have no traps, so read the comments carefully.
If processing slides at single time points (i.e. not timelapses but collections of single timepoints such as for fixed cells) then there is a special GUI (`experimentTrackingSlidesGUI`) expressly for this purpose. The underlying software and procedure is the same but everything can be done via the buttons (so no need to use the script) and the GUI's are a little better suited to single slides. Note, this GUI is rarely used in the swain lab so may be a bit buggy.

Bibliography