Analysis pipeline manual for project titled:

**Retinal direction selectivity maps develop independently of visual input but require retinal waves**

Read before moving on: I wrote a lot of very detailed notes below to make sure that going through the analysis pipeline is as clear as possible. If you want to skip the notes and go straight to the action items, I have highlighted them in yellow.

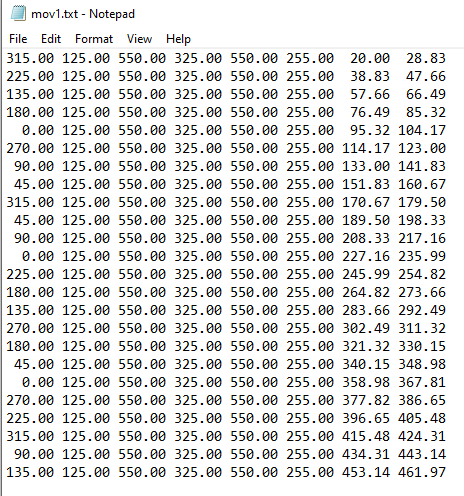
**Starting point:**

After image acquisition, I end up with a folder that has calcium imaging movies (Tif stacks) and text files.

The calcium imaging movies (Tif stacks) are X by Y by T movies. In all cases, X = Y = 256 pixels. Unless otherwise specified in the experimental notes, T = 1400 frames. In the case where GFP was bound to a known cell type, the movies contain both the calcium imaging and the GFP label in a 2800 frame interleaved movie.

The text files have names that correspond to a calcium imaging movie (which movies text files correspond to is found in the experimental notes) and contain information about the stim trials.

Here is an example text file:



Every row is a single trial. The first column is the direction of the stim (in degrees). Note that direction is based on what was shown on the stim computer. There is a transformation that needs to be done to change those directions to match the orientation of the retina in the perfusion chamber. For readers analyzing my dataset, this transformation key can be found in the experimental notes for every retina (it will be used in Matlab codes below). For members of the Feller lab, this transformation can be done with a quick calibration at the end of the experiment.

**Part 1: Analyzing a single movie file**

This section will run you from raw data, to exporting the key aspects of the data to a folder that will compile data across your entire dataset.

**Step 1: correcting for translation and calculate ΔF/F**

No matter whether I acquired a 1 channel movie (just calcium imaging) or 2 channel movie (calcium imaging + GFP label), this next FIJI macro requires the latter configuration. Specifically a 2 channel movie, 1400 frames each and interleaved, for a total of 2800 frames. So for a 1 channel movie, I simply duplicated the movie and interleaved it. This is just because it’s easier to motion correct on a GFP channel. Regardless, the final output will be a dF/F transform of the raw data. Also, this macro has a set number of times that correspond to when the light stims are presented, which remained constant throughout this dataset.

Open “[RegisterAndCalcDFOF\_withGFP\_v2.ijm](https://github.com/atiriac/Alex-DS-project/blob/master/RegisterAndCalcDFOF_withGFP_v2.ijm)” in FIJI, it will open up in the macro editor window. Now open the raw calcium movie file. If it’s just one channel (1400 frames), duplicate the movie and interleave so it bumps to 2800 frames. If the movie is two channel, you don’t need to do anything, it should already be 2800 frames. Run the macro.

Note that this step can take a long time and hogs a lot of computer RAM. Do not attempt on computers with little RAM (<4GB).

Now find the dF/F movie and save it. I typically keep the name of the original movie file and add “\_dFoF” before the “.tif” extension. Also save the mean image of the calcium dye and, if available, the mean image of the GFP.

**Step 2: automatic detection of DS regions**

In Matlab, run findDS. It will ask you for the ΔF/F movie you created in step 1. It will also prompt you for the text file that was generated during the acquisition of that video.

Note that the name of the text file doesn’t necessarily match the movie, you have to look at the experimental notes for that day to figure out which text file is correct.

There will be two output files after running findDS, one raw and one that is smoothed. The code by default saves the smoothed one as whereDS.png, but this can be changed in the findDS code if necessary (I find it doesn’t matter). I typically save one of the matlab images that shows DS ROIs as movieName\_whereDS.png.

**Step 3: match the automated DS ROIs to actual cell locations & draw ROIs**

In FIJI, open the mean image of the calcium imaging that was saved in step 1. If available, also open the mean GFP image. Now also open the whereDS.png that was saved in step 2. Combine all of these images into one by going to image->color->merge channels and assign your open files to different channels. I save this merged file with multiple channels as movieID\_avgs.tif.

Using this new file, you should be able to see the DS regions overlapping the actual location of cells. Some DS regions will match the location of cells, whereas other DS regions will match things that do not make sense (e.g.: a blood vessel or multiple cells). At this point, I draw ROIs in FIJI over the DS regions that are over identifiable anatomical cells. To draw ROIs in FIJI, circle a region and press “t”, the ROI manager should pop up and show you the saved ROIs.

Once all ROIs have been circled, save the ROI manager. It’s important here to make sure you don’t have one ROI selected when you save the ROI manager file or you will just save that one selected ROI. Additionally, make sure to keep the .zip extension in the saved file.

With the ROI manager still open and populated with all the ROIs, create a new 2D blank image (shortcut: Cntrl N) and make it the same XY dimensions as the average image (keep it 8bit and fill with black). With this new image selected, go to ROI manager and click more -> fill. It should fill the circles in white. I like to save this image mask as fileName\_dsMASK.tif.

**Step 4: Manually classify cells as ON, ON-OFF, or bad cells**

In Matlab, run compareDS. This code is designed to allow you to compare a pre-drug and post-drug condition, so when analyzing just one movie, simply load the one movie twice (sorry someday I will be a better coder and write smart code).

A GUI will pop up and ask that you to load two dFoF movies, a mask, and two txt files. When analyzing just one movie, simply load the same dFoF twice, the mask you created in step 3, and the same txt file twice.

At this point I cycle through the cells and keep notes on which cells are ON-OFF, ON, or “bad” cells. I define a cell as “Bad” if the cell exhibited grossly inconsistent responses to each of the 3 trials for the eight different directions. ON-OFF cells respond to both the onset and offset of the light, whereas ON cells only respond to the onset. Below is an example of how I keep notes, the syntax is important for using my future codes.

OnOffCells=[1,4,10,11,14,17,19,20,23,26,28,29,30,31,32,33,35,36,37,38,47,48,50,52,53,56,62,64,69,76,78,80,84,87,92,100,103,104,105];

OnCells=[2,3,6,13,16,21,22,27,34,39,40,57,71,73,79,95,99,107];

badCells=[5,7,8,9,12,15,18,24,25,41,42,43,44,45,46,49,51,54,55,58,59,60,61,63,65,66,67,68,70,72,74,75,77,81,82,83,85,86,88,89,90,91,93,94,96,97,98,101,102,106];

In the above example, there were 107 potential DSGCs. The number refers to the cell ID, which is given in matlab by numbering cells from leftmost to rightmost.

**Step 5: Run the mainDS code to plot results from this FOV**

In Matlab, edit the mainDS code. You will need to copy paste the cell IDs you generated above in the appropriate field in the code under the “inputs” section. It should look like this:



You also need to edit the direction key. Copy paste it in the “inputs” section as well. For my dataset, the directions key is pasted in each experiment’s notes at the end of each retina section. It looks like this in my notes:

real0 = 270;

real45 = 225;

real90 = 180;

real135 = 135;

real180 = 90;

real225 = 45;

real270 = 0;

real315 = 315;

The reason for a key is that retinas are not always positioned the same way in the rig, and directions change depending on left eye and right eye. This key takes care of this post-hoc.

The purpose of this code is to export the ROI data and direction information to a folder that will compile all of your data. Therefore, in the mainDS code, there is a section called “%% Save ROIint and textFile as a variable”. Go to that section and make sure you have a path specified. It should look something like this, though you will likely need to change the path.

pathName = 'C:\Users\Olive\Dropbox\DS project\3\_compiledResults\Data\';

MovieNameNoExt = movie\_name(1:end-4);

save([pathName,MovieNameNoExt,'.mat'], 'roiInt', 'cellCenters', 'textFileArray')

Once you run the code, go to the path you specified and make sure your data was correctly exported. There should a .mat file with the name of the file. Here is an example: “181112\_004\_reg\_dFoF\_binnedInZ.mat”

As you export data, you should populate metadata about the FOV you are currently analyzing into a spreadsheet that will be used later. Read Part 2, step 1 for more instructions.

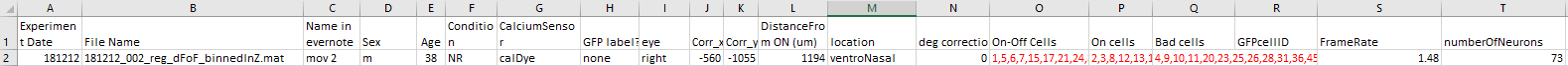
**Part 2: Compiling all data into one table**

Now that you’ve exported data from multiple animals and consolidated them to a single location, we will run code that (1) generates a giant data and metadata table called neuronTable.mat and (2) runs analyses on neuronTable.mat to plot results to answer specific research questions.

**Step 1: create/update a spreadsheet that contains metadata for each FOV**

In the last step of part 1, you created a folder where you exported data from all of your experiments. Before we start compiling your data, we need a spreadsheet that will tell the code which data to compile, and what metadata to append to each FOV. Metadata includes things like age, light rearing condition, genotype, etc…

If it doesn’t already exist, you first need to create a spreadsheet (I call mine “DSmainExperimentsSheet.xlsx”). Leave the first row for column headers, and populate the first row with information on the first FOV. Here’s an example of the headers and first row:



You need metadata for every file you have exported. If you don’t have it, the rest of the code won’t bug out but will not analyze any FOV that isn’t represented in the spreadsheet.

**Step 2: compile data and metadata from multiple FOVs into a big neuronTable**

In the compile folder with all of the data and metadata spreadsheet, run “buildNeuronTable3.m”. This code will take a while to run, especially as you add more and more FOVs. It does have an integrated countdown that will show progress every 10 FOVs it has analyzed. Note that this code should start the parallel computing toolbox in matlab. If you run into issues with this, go into the code and replace “parfor” with “for”.

When this code is done running, it will generate a matlab table called neuronTable and save it in the folder as neuronTable.m

**IMPORTANT NOTE:** Up until this point, the pipeline was linear. Once this neuronTable.m file is created, you can start asking questions. Most of the analyzes I have created from here on start with the neuronTable, so they are essentially parallel analyzes and therefore nonlinear.

**Part 3: Analysis and plots**

As explained above, starting here the analyses are mostly run in parallel. You don’t have to do these in order. Some sections will have titles (usually when multiple codes are required) and other will just start with the code name. Where appropriate, I will include a screenshot of example figures that were generated using the code.

**-Generating polar plots and polar histograms of your data:**

Chart, radar chart

Description automatically generated**polPlots3.m**: This code generates polar plots and polar histograms on neuronTable.m. At the top of the code, you can specify conditions that will restrict which data you want to plot (for example if you just wanted to look at the dark rears). It also does a bunch of other small things, like a silhouette analysis for example.

**silTest.m**: This is a function that runs a silhouette test. It’s called by some functions, like polPlots3.m

**-Functional clustering of data:**

Chart, scatter chart

Description automatically generated

**dsProp.m**: This will run a k-means clustering based on given results from silhouette analyses to functionally cluster the preferred directions. You can then save the newly produced tables that include a column (idxDS) with the functional ID of each cell (1: temporal, 2: dorsal, 3: nasal, 4: ventral).

**-Deviation of functional clusters’ pref dir from cardinal directions:**

Chart, scatter chart

Description automatically generated

**deviationFromCardinal.m**: this code plots each functional grp's preferred direction as a function of distance from optic nerve. It's a good way to quantify the DS map

**-Assess prop and tuning of different fxn clusters:**

**loadProps.m**: This code will look at proportions in WT and B2 (you may need to change some inputs), make plots, and run stats.

**loadTuning.m**: Similar to above but for tuning (DSI and VS).