Clustering analysis

February 2019, Carey lab, Léonard Dupont

1P calcium-imaging recordings were performed in L7::cre(CAG::GCaMP6f.floxed) mice with a cranial window above the right hemisphere of the cerebellum (~3mm, paravermal to lateral). We intend to correlate calcium transients (proxies for neural activity, specifically CF discharge [complex spikes] in this case) with motor behaviour:

(i) Check how overall cell activities vary when the animal is still or moving.

(ii) Attempt to define spatial clusters based on cell activity to validate the long-standing microzone hypothesis.

(iii) Try to further correlate movement phase (stance, swing, amplitudes, phases / temporal and spatial components, etc.) with subregion (microzone) activity.

In this document, we summarise the state of things in our attempt to achieve these three goals, in particular the second one.

0 State of things, things to do (brief)

— Deal with the CNMF-E based demixing (Hoss)
— Fix the tracking algorithm issue to align the rasterplot and belt speed
— Work on the clustering and come up with something

1 Calcium analysis pipeline

a. *Frame registration* : we register all frames onto a queen frame to correct motion artefacts. Resulting files are .tif stacks (from an imaging session) that are cropped to a region of interest and all aligned in space and time.

b. *Stack concatenation* : the Mukamel (2009) algorithm is used to define elongated ROIs, supposedly corresponding to the dendritic tree of Purkinje neurons. To do so, it uses both PCA (separating pixels with high-amplitude variations from those with low-amplitude changes (noise)) and ICA (hypothesis : cells clustered with the first PCs have more or less independent activity) while satisfying temporal and spatial sparseness criteria. To do so, the manier frames we have and the better it is. Using a custom-written function, we concatenate trials (tif stacks) in bigger ones to seed mukamel.

c. *Median filtering:* to remove shutter noise which creates horizontal lines passing through the FOV during the recording and lead to fallacious ROIs in the y direction, we median filter the concatenated stacks with a 5-frame temporal window. We loose temporal resolution, but this is solely to isolate cells: extraction of calcium traces is done on the non-filtered data later on.

d. *ROI extraction using Mukamel:* we extract the ROIs using the method proposed by Mukamel et al. (2009). We are currently using *mu = 0.5.* This might need to be tuned, for space is more important than time in our case (median-filtered). We then convert the ROIs to a *cn (carey neuron)* canonical format for further analysis.

e. *Normalise and baseline* signals using the *zero_and_mean.m* function.

f. *Extraction of traces and spikes:* we map the previously-extracted masks onto the non-filtered data and extract pixel fluorescence as a function of time: we now have our calcium traces. Using the *plot_all_traces* function, the user can visualise these calcium transients and pick a few nice-looking ones containing single-spike events (or what looks like it). Indeed, we are currently using the *MLSpike* (Deneux et al. 2016) algorithm for deconvolution. It follows the steps listed hereunder:

i. Calibrate the algorithm using a trace presenting single-spike events that have been flagged by the user (semi-manually : see *select_single_spikes.m*) - extraction of tau, sigma, amplitude.
ii. Deconvolution of all the roi traces and building a big struct called *deconvolution* containing spiketimes, the roi mask, the roi centroid and the initial calcium trace.
iii. Further build ISI histograms and raster plots (*build_ISI_histo.m*, *rasterplot.m*) respectively.

The next steps all relate to merging the behavioural data and the above-analysed calcium-imaging data.

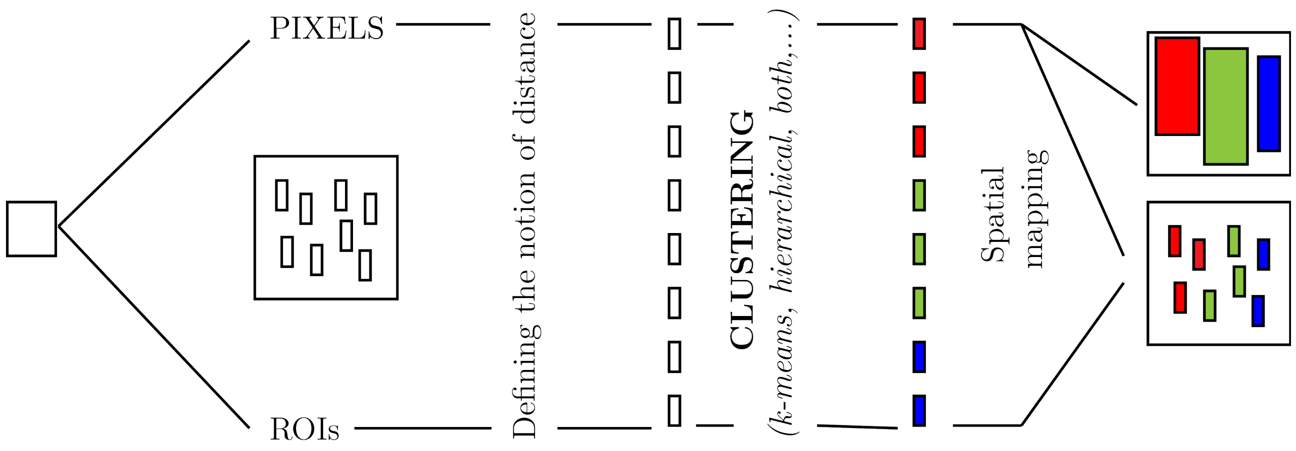
2 Cell clustering

An activity-based cell clustering would be beneficial — if not crucial — to demonstrate once more the microzone hypothesis (Heffley et al. 2018), this time in the context of locomotion. In other words, we would like to prove that clusters of cells sharing common activity patterns also share a common spatial coordinates [*x_1 ; x_2*] on the recorded part cerebellar cortex. We want to group ROIs based on their activity: a single CF innervates a whole bunch of Purkinje cells, which only receive input from this sole olivary input themselves. Right now, it is all about finding which way is best to reach our goal.

i. Defining activity: the initial clustering will be done on activity, which — in our case — corresponds to changes in photon counts as a function of time (GCaMP). The first option would have us use pixels (Heffley et al. 2018) to generate pixel clusters : without taking position into account at first, then plotting the groups back on the frame. Afterwards, we could either use pixel patches (the clusters) alone to define regions or bring it back to ROIs with a mask defining to which cluster each mukamel region would belong to. The second option would consist in directly taking the calcium traces from the ROIs (Dombeck and Tank 2009) to cluster regions of interest based on activity before plotting them back on our frames.

ii. Defining similarities: all clustering algorithm share the notion of distance between two observations. Traditionally, this can be set as the Euclidian distance or correlation using different methods. In the case of calcium activity, we are looking for correlated calcium transients over our N units. This comprises both peak time and peak amplitude. A first approximation could be the use of the Pearson correlation coefficient (uses both covariance and standard deviation). If we were to use ROIs, we could also generate a virtual vector with peaks out of the spiketimes (post-deconvolution) in order to standardise signals and focus on these two features only (time + amplitude).

iii. Choosing the right clustering method: this might be the most tricky part. Clustering algorithms are numerous and all present downsides offsetting their downsides. Most studies either use repeated k-means ++ algorithms or hierarchical clustering based on similarity of activity. Some also mix the two, starting with k-means and eventually merging some clusters using distance matrices. We will probably try both to make sure we converge towards the most sensible option. I started off with a k-means based method, for I had the feeling that it could prove to be statistically compelling.


PIXELS
ROIs
Defining the notion of distance
CLUSTERING
(k-means, hierarchical, both,...)
Spatial mapping

- **k-means:** hereunder, we describe the current state of the clustering programme. It is partly based on Dombeck et al. 2009. The detailed code can be found in *clustering_k_means.m.* This algorithm uses ROI calcium traces and the Pearson correlation for the distance.

Step 1: we have a matrix containing fluorescence values for all ROIs over time. We pass it through *zero_and_mean.m* which brings all traces back to zero (subtract the minimum) and normalise them using their respective mean. Hence all mean values are 1 and this criterion cannot be the one used for clustering.

Step 2: the user defines K (number of clusters) and *runs* (number of times that the k-means algorithm shall run). We use the k-means ++ variant : the seeding of the centroids at *(t=0)* is done uniformly for cluster 1, and then space is weighed with a choice probability based on proximity to already-existing centroids for the (K - 1) remaining clusters. Here we set *runs* > 2e3. For each run, we store the ROI numbers *int([1 , N_{rois}])* belonging to each cluster in a struct named *'clusterresults'* with subfields c_i i from 1 to K.

Step 3 - statistical relevance of clusters: this is the tricky part. Formulated concisely, we want to test for the redundancy of the populations in our K clusters over the runs. I first thought that one could interpret this in terms of intersections. This is the current idea that I am working with:

(i) <u>Finding neighbours :</u> For each of the R rois, find the cluster c_i they are in for each run. Every time, concatenate in a vector *($neighB_R$)* the other regions that were in this cluster too.
(i') The result is an ensemble of R vectors, where *$neighB_R$* contains the neighbours of ROI R over all the runs.
(ii) <u>Building statistical neighbourhoods :</u> For each of the vectors, calculate the occurring frequency of each ROI: in other words, find how reliably each of the remaining *(R-1)* regions have been clustered with it. Build a new vector *$neighB_{R,thr}$* in which only the ROIs such that $f_{occurrence} > 0.8$ have been put. This is now a close-neighbourhood and a statistically significant one. If the length of this vector is smaller than Z (to be defined : probably between 5 and 10 — we can fine-tune that empirically), then the neighbourhood is discarded.
(iii) <u>Merging neighbourhoods :</u> We can now think along the following idea : reciprocal neighbours have similar neighbourhoods and should be clustered. However, we can not simply merge neighbourhoods without a similarity criterion. Hence, we build a distance matrix M (real,

symmetrical) such that M*(j , k)* is the distance between neighbourhoods *j* and *k, $D_{j,k}$*. This distance is defined as follows :

$$D_{j,k} = \left(\frac{card(\ neighB_{j,thr} \cap neighB_{k,thr}\ )}{max(\ card(neighB_{j,thr})\ ,\ card(neighB_{k,thr})\ )} \right)^{-1}$$

We then follow the hierarchical clustering procedures : we merge the two closest neighbourhoods. However, this could both be done by using the intersection and the union of our ensembles. What I propose is to use the union, but that we take into account the fact that some elements of the merged neighbourhoods were not part of the intersection. To do so, I propose the following pipeline: each neighbourhood should initially possess a weighing vector W_R such that :

$$W_R = [w_1,\ \dots\ , w_R] \quad \text{with} \quad w_i = 1 \quad \text{if } i \text{ belongs to the neighbourhood and} \quad w_i = 0 \quad \text{otherwise.}$$

Then, at each merging event of *R1* and *R2*, the new neighbourhood *R'* will be accompanied by a new weighing vector $W_{R'}$ such that:

$$W_{R'}(k) = w_k^{'} = \frac{W_{R1}(k) + W_{R2}(k)}{2} \quad \forall k \in [1, R]$$

Actually, these weights would be calculated beforehand and taken into account in the distance formula. Indeed, we want to favour merging events that will bring little new elements and entertain the core of the neighbourhoods. Weights will decrease for elements coming from the union that do not reappear.

$$D^*_{j,k} = \left(\frac{card^*(\ neighB_{j,thr} \cap neighB_{k,thr}\ )}{max(\ card(neighB_{j,thr})\ ,\ card(neighB_{j,thr})\ )} \right)^{-1}$$

$$\text{with} \qquad card^*(R1 \cap R2) = \sum_{k=1}^{R} W_{R1 \cup R2}(k).ind(k \in R1 \cup R2)$$

where *ind* is the usual indicator function. I have the feeling that the issue with this formula is that it does not take into account the time since last merging, or in other words the stability of the kernel. Perhaps I am wrong, I need to think about it.

(iv) As long as there exists a distance being smaller than the merging threshold, we create bigger clusters. When it is not the case anymore, we stop the merging process and we hopefully have our final groups.

This will have to be tested soon on MATLAB.