Intracellular Receptors

Materials and Methods: Analysis and Processing

Srdjan

February 24, 2021

1 General pipeline

A typical experiment involving imaging of pancreatic slices in our lab concerns a single field of view showing up to hundreds of cells, in a recording of at least several, often dozens, gigabytes. Current tools (ImageJ, ...) rely on loading the recording, or its part, into memory, for viewing, analysis, and processing. It also requires laborious and long human engagement. We have developed a set of interdependent tools to automatize as much as possible the analysis pipeline (see fig. 1). The crucial elements of our pipeline are the following:

- (Semi-)automatic detection of regions of interest (ROIs);
- Transformation of ROI time traces into standard score ("z-score") and correction for filtering distortion;
- Quantification of the phenotype for each ROI in terms of the distribution of events of different durations.

Our toolset is inspired and relies on CaImAn [1], a similar package developed for the purposes in neuroscience research.

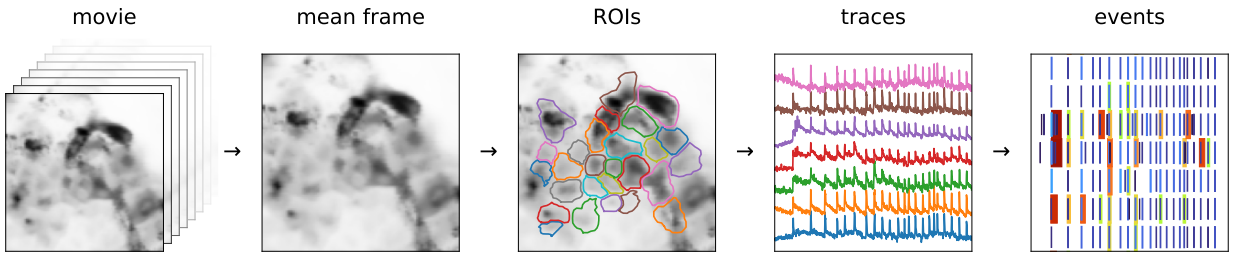

Figure 1: An illustration of our processing and analysis pipeline: (*i*) From a full movie, we calculate the mean (or other statistic) across all frames. (*ii*) We pass the mean image through a band-pass filter and define ROIs by detecting local peaks. (*iii*) We save ROIs with all the important information (time traces, ROI coordinates, movie statistics, recording frequency, pixel size, etc.). (*iv*) Traces contain features at very different timescales—with different timescales presumably important for different cell types. We collect them into separable events for analysis.

1.1 (Semi-)Automatic Detection of Regions of Interest

Once imported, a recording is stored as a 3-dimensional ($T \times x \times y$) numpy array [2]. When the recording is stable, obtaining a mean image, or any other statistic over frame, is rather trivial. In case there is horizontal movement, it can be corrected for by aligning the frames to a template. For this we use the functionality present in CaImAn [1], except that high frequency recordings

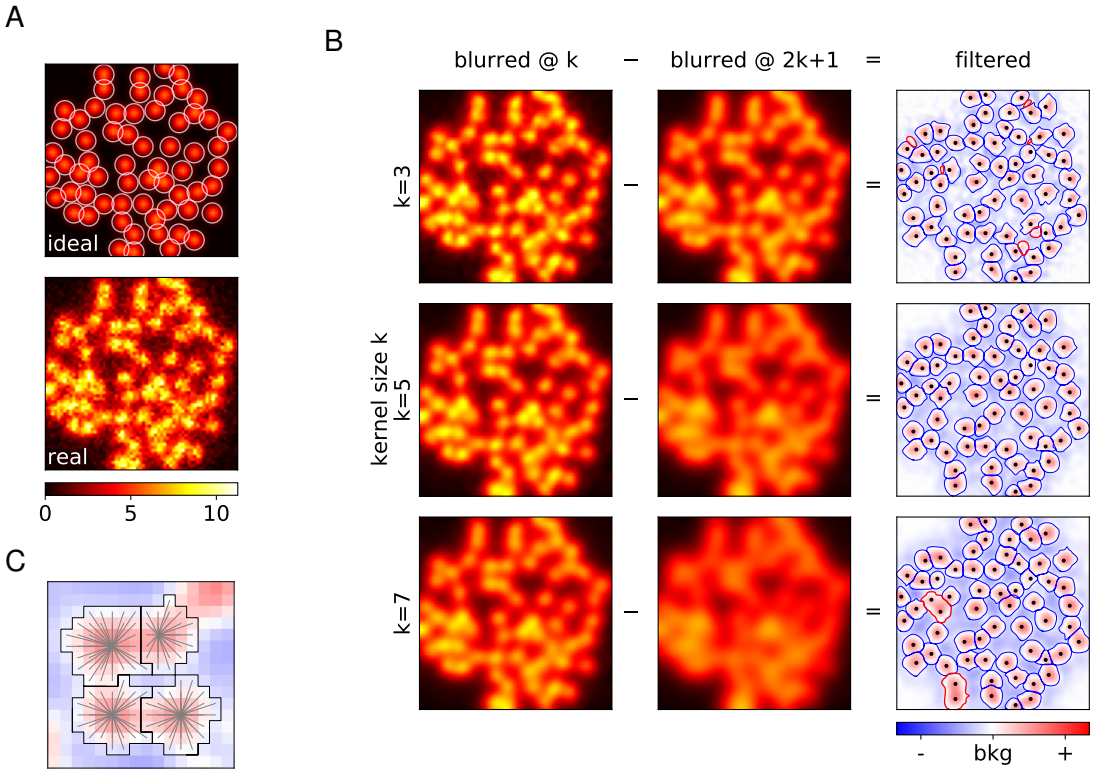

Figure 2: (A) Computationally created template 64×64 image with uniform spherical cells, and with Poisson noise added. (B) Band-pass filtering of the "realistic" image from A, with different kernel sizes, for denoising (*left*), and for approximating background (*middle*). The difference is the filtered image, used to construct ROIs (*right*). The size of the kernel determines the approximate size of the ROIs obtained. In thick red contours we emphasize misidentified ROIs; the dots indicate the real locations of the cells. (C) Each ROI is constructed by explicitly searching for a closest peak in intensity. A pixel can only be part of a single ROI.

need to be rebinned to some moderate frequency (a few Hz), before correcting, in order to reduce the noise level. Once the translation offsets are obtained, we use them to correct the recording in original frequency.

To define regions of interest, we blur the representative image by a kernel of the size we expect cells to be, and at the scale double of that. The difference between these two images represents a band-pass filter of the original, where the local intensity variation are emphasized (fig.2). We then pass through all pixels where the value of the filtered image is positive (or larger than a small positive threshold), and for each pixel we search for a local peak in its vicinity. All the pixels that lead to the same local peak are then grouped into a single ROI.

As we are mainly interested in islet cells, we choose the kernel size to approximately correspond to 10μm, the characteristic length scale of the islets cells. If the pixel size is unknown, the choice of kernel is up to the person running the scripts.

Representative image can be a mean over all frames or any other statistic. In addition our code supports standard deviation, mean and standard deviation of the first derivative of the movie, and a "robust maximum" of the movie. As "robust maximum" we define a very high percentile of the set absolute values of a set, essentially a value close to its maximum, by default it is 10th largest. We avoid the maximum as a means to make analysis robust to outliers. This statistic is sensitive to cells which fire extremely rarely during a recording, so that the mean of those pixels is negligible. By default, we choose an average of the mean and high percentile as a representative image for band-pass filtering and ROI extraction.

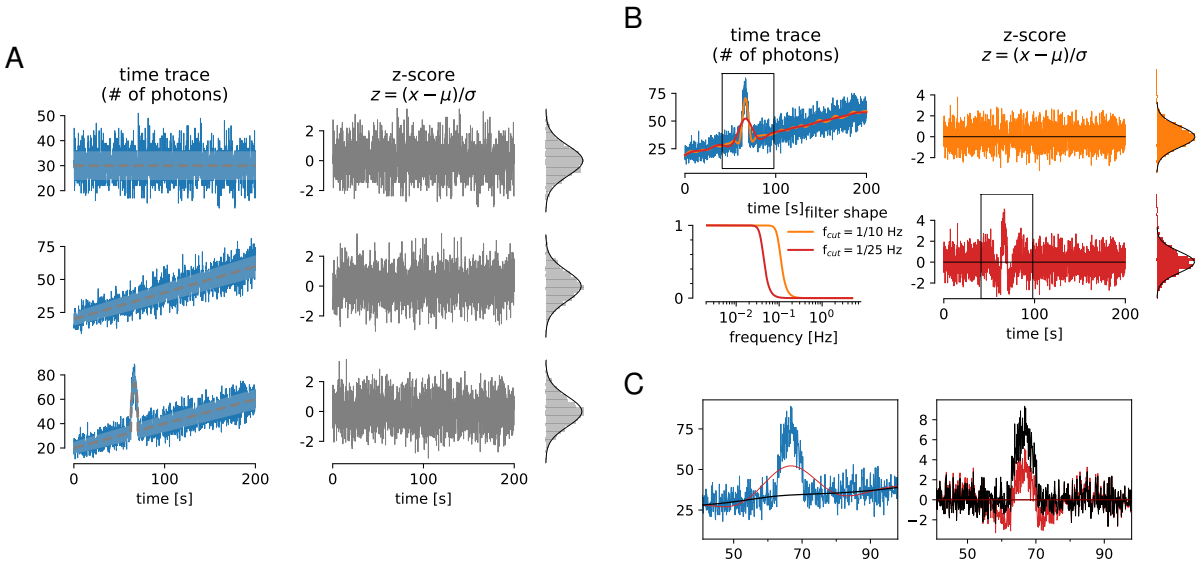

Figure 3: (A) *Left*: Simulated traces with Poisson noise (blue) around different means (grey) with different temporal features. The shaded area represents one standard deviation interval around mean. *Right*: Irrespective of the shape of the mean, z-score transformation behaves exactly as a Gaussian variable with zero mean and unit standard deviation.
(B) The values of z-scores depends on the reference. In subfigure (A), the reference curves are known, but in general they need to be inferred, typically by low-pass filtering. The cut-off frequency f_{cut} of a filter determines the timescale of the events that can be detected in z-score. If filtered with too high cut-off (orange), the resulting smooth curve follows too closely the trace and the event is not visible in z. With $f_{cut} = 1/50$ Hz, the resulting z-score has obvious outliers ($z \geq 3$), visible both in trace and in the histogram.
(C) Zoom to the indicated regions form B. The original (blue) and the filtered trace and z-score (red), and the filtered trace and z-score corrected for distortion (black).

1.2 Trace processing

1.2.1 z-score and filtering

In an ideal detector, recording a time trace of a single pixel in absence of any signal would consist of independent values of the number of photons detected during the dwell time. The values (x) are then distributed according to the Poisson distribution, with standard deviation (σ_1) being equal to the square root of the mean μ_1, $\sigma_1 = \sqrt{\mu_1}$.

A transformation to a new variable

$$z = \frac{x - \mu}{\sigma}$$

is called *standard transformation*, and the new variable standard score or z-score (fig.3). It recasts the initial quantity x in the units of standard deviation from the expected mean. A noisy trace in z spends 95% of the time between -2 and 2, and 99.7% between -3 and 3. Probability of $z > 3$ is very small $p < 0.0013$, which is why it is often considered a threshold value for pointing out the outliers.

In general, the mean slow component needs to be inferred, typically by low-pass filtering. Throughout this project, we use cascaded second-order section (sos) filtering, implemented in `scipy.signal` module [3]. The cut-off frequency f_{cut} of a filter determines the timescale of the events that can be detected in z-score.

Correcting for the filtering distortion Fast Fourier transform naturally distorts signals, but the inferred z-score can be used to correct for it. We constructed an iterative filtering algorithm, where at each iteration, we neglect the outlier values of the original trace, substitute them with the values

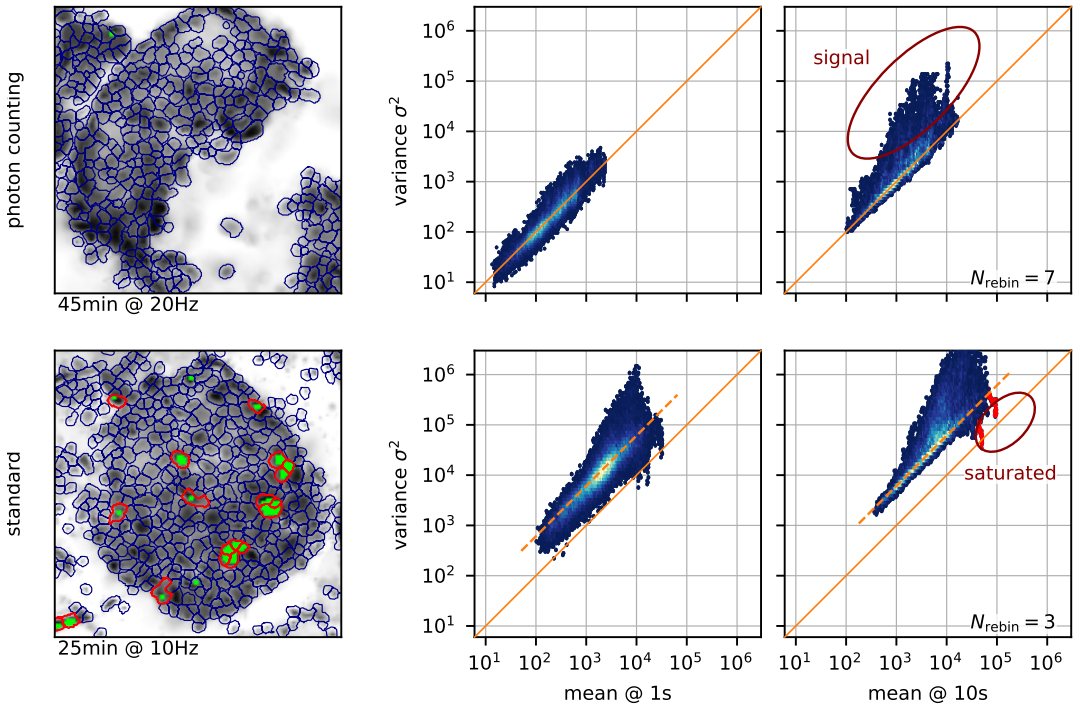

Figure 4: *Left*: Regions of interest for the two experiments recorded using the raw photon-counting mode (*top*, used for Fig.1 in the main text), and the standard mode with large gain (*bottom*). Pixels which had at least 10 saturated values are shown in green. In thick red contours we emphasize ROIs with at least 30 green pixels used to emphasize the influence of saturation (see below).
For each recording, we filter the ROI traces at 1s (*middle*) and 10s (*right*) to separate the trace into slow and fast component. For each ROI we then randomly choose 100 values of the slow component together with the variance of the fast component from a window around the same timepoint, with window size the same as the timescale.
In agreement with our assumptions, the variances and the means are linearly dependent, and in the case of raw photon counts, the dependence is exactly 1 (solid diagonal line $\sigma^2 = \mu$). In the standard mode, the dependence is still linear, yet with a slope larger than one (dashed orange line). Points above the bulk in this view are due to windows with larger variance, and presumably connected with activity (at the appropriate timescale). Points below the bulk are due to undervariate windows. They are concentrated at high values and are due to saturated pixels. In the lower right plot we emphasize this fact by showing the points from saturated ROIs in red.

of the slow component, and reapply the filter. At each iteration, the distortion is less prominent, increasing the z-score. In fig. 3C, we show the result after 10 iterations, though we find three iterations is a more conservative choice, and a reasonable compromise between results and computing time.

ROI traces All above refers also to a sum of pixel traces, but, crucially, not to their mean. A sum of two pixels a and b ($x = x_a + x_b$), with means μ_a and μ_b, would have a standard deviation as expected $\sigma = \sqrt{\mu} = \sqrt{\mu_a + \mu_b}$. But, if we were to consider the average $\tilde{x} = x/2$, standard deviation would be $\sqrt{2}$ times smaller $\tilde{\sigma} = \sigma/\sqrt{2}$. Therefore, when calculating z score for a ROI trace, we always consider the sum, rather than the average trace of the underlying pixels. When we visualize traces, we show them averaged, only to keep the scales comparable.

The same reasoning holds for rebinning a single trace, where the resulting trace, rebinned by a factor of n, has a standard deviation $\sqrt{n}$ times smaller than the original.

1.2.2 Realistic detectors

All? the experiments discussed in this manuscript were recorded on a Leica setup with Hybrid detector in photon counting mode. In this case, we see no significant departure from our assumption of Poisson distribution. Even with non-unit gain, the linear dependence between variance and mean remains, though the slope is different from 1 (fig. 4).

Other types of detectors introduce additional sources of noise other than Poisson (e.g. thermal), but, mostly it is still dominated by Poisson in our experience, at least as long as the values between neighboring pixels and frames are independent.

1.3 Identification of Events

Traces contain features spanning orders of magnitude in time: from tens of miliseconds, to tens of minutes. We aim to investigate how do events at these different timescales interact and a connection between them and the islets' environment. For this, we devised an algorithm based on sequential filtering, followed by . In the first step, we perform a sequential filtering of the traces at timescales starting from 0.5s, and increasing by a factor of $\sqrt[4]{2}$, $\tau = \{2^{-1}, 2^{-3/4}, 2^{-1/2}, 2^{-1/4}, \dots\}$, until the timescale of the longest event we are interested (and in any case it makes no sense to filter at timescale comparable to the length of the experiment itself). At each timescale, we transform the trace to z-score, and identify regions where $z > 3$ as *candidate events*.

An event is characterized by the start time (t_0), it's maximal height, and the width at the half of the height (*halfwidth*, δt), which is our measurement of it's duration. For simplicity, we define end time as $t_{\text{end}} = t_0 + \delta t$, although events arguably last much longer after the intensity drops at half of the peak.

If an event is real, it will be detected at multiple timescales, and will start around the same time and will have approximately same halfwidth. We specify a tolerance of 20% of the halfwidth as to whether two candidate events should be considered equal; if their start and end times are within 20% of the halfwidth, they are considered cognates, having arisen due to the same real event. For a set of cognates, we estimate the start and end time of the real underlying event as a median over the set. If the resulting estimate for the halfwidth is larger than 2s, we require that a set consists of at least 4 candidate events (corresponding to event being detectable when filtered at timescales that differ at least two-fold). For shorter events, we require only that an event is not unique to a single timescale.

We also neglect events that last less than 3 time frames, as well as those too close to the beginning or end of the recording (within $\delta t/2$), which we ascribe to artefacts from zero-padding for the filtering. We term this procedure event *distilling* (see fig. 5).

2 Analyses

2.1 El Toro

2.2 Isradipine

2.3 Low Ca

2.4 etc

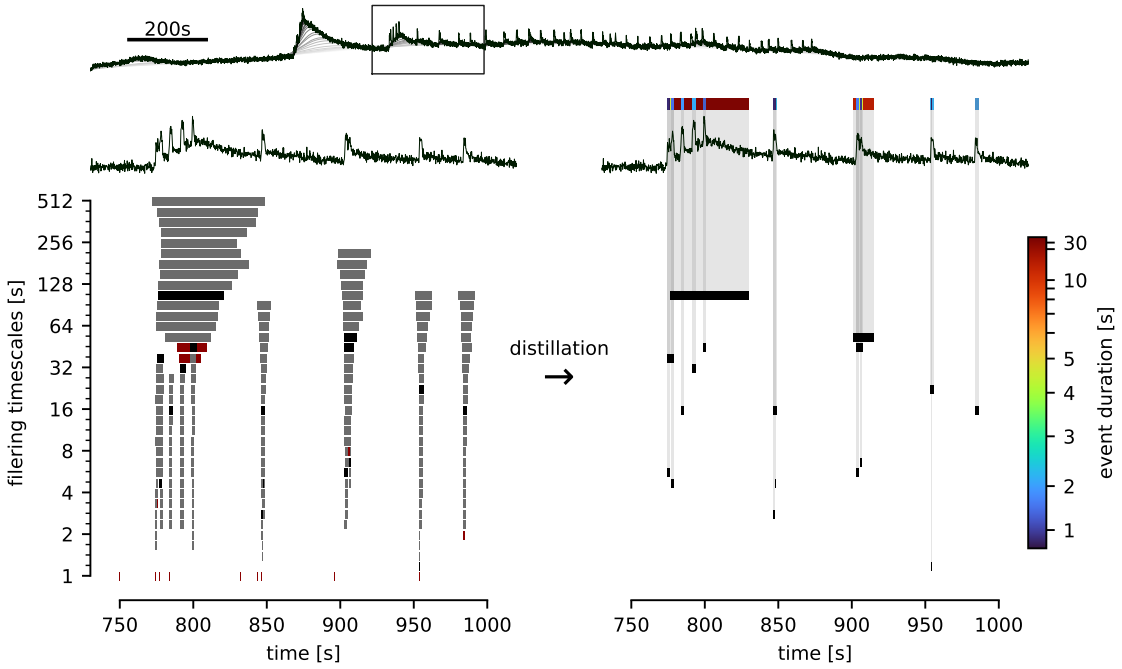

Figure 5: (A) Example of a trace (black) and slow components in sequential filtering (shades of gray). (B) Close-up view of the indicated small region from (A) and a representation of all candidate events in that region detected through sequential filtering. Each bar represents an event. Vertical position corresponds to the timescale at which the event was detected, the horizontal boundaries correspond to beginning and end time. In grey we show events that are deemed false positives, because they are detected at too few filtering timescales. Others events belong to groups of similar events which are then considered real; we show a boundary around the bar of the event closest to the one finally distilled. It's time boundaries are set to median of the whole group, and the color indicates its halfwidth. (C) Distillation greatly removes redundancies and false positive candidates. Only robust events are selected for further analysis (see text for details). In the top part we represent the distilled events at single height, in a more compact view. We choose a color code (mapping to a hill curve) which is nearly linear up to a few seconds, to emphasize the color difference at the timescales most interesting for our analysis.

References

[1] Andrea Giovannucci, Johannes Friedrich, Pat Gunn, Jeremie Kalfon, Brandon L Brown, Sue Ann Koay, Jiannis Taxidis, Farzaneh Najafi, Jeffrey L Gauthier, Pengcheng Zhou, Baljit S Khakh, David W Tank, Dmitri B Chklovskii, and Eftychios A Pnevmatikakis. Caiman: An open source tool for scalable calcium imaging data analysis. *eLife*, 8:e38173, 2019.

[2] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[3] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quin-

tero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.