PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERÍA
Dirección de Investigación e Innovación
Programa IPre de Investigación en Pregrado

Calcium Image Analysis Pipeline for Analysis of Calcium Signals

Gabriel Miranda[a], Jeremiah Zartman[b]

[a] *Department of Chemical and Biological Engineering, School of Engineering, Pontificia Universidad Católica de Chile. 4th year*
ggmiranda@uc.cl

[b] *Department of Chemical and Biochemical Engineering, College of Engineering, University of Notre Dame. Professor,*
jzartman@nd.edu

Abstract

Calcium images has been used in biological science for various reasons, but in this study, we are focus on the usefulness of them to represent physiological changes in the cell or tissue that it is being studied. The study of these images grants great insight of what is happening in the system that we are studying, this is why several analysis techniques have been developed in the past years. In this research, we have developed a pipeline for the analysis of calcium images that uses different programs to generate the extraction of different features that describe the behavior of the tissue that is being studied in a quantitative way. The first software used is called CaImAn and is used to generate the motion correction and source extraction of the calcium images. Following this step, the calcium profile is extracted and analyzed with SciPy. The pipeline shows great performance, and the capability to generate figures to communicate the data generated, all of this in a highly automated way, where only a few parameters have to be tunned by the user.

Palabras clave: *Calcium imaging, Image analysis, Calcium signaling, Feature extraction, Automated analysis.*

1. Introduction

Many processes that occur within the cell use calcium (Ca^{+2}) as second messenger, for example cell division, growth and death present an underlying calcium dynamic that guides this process

(Berridge et al., 2000). The cells can encode intercellular and intracellular signals in Ca^{+2} signals, this signals are characterized by their amplitude, frequency and integrated intensity (Berridge, 1997; Clapham, 2007). For this reason, it is important to study the calcium expression of the cell to understand the effect of certain environmental conditions on the cellular dynamic represented by the calcium signals, changes in the characteristics of the signals may represent the changes of different physiological aspects of the cell. This could help us to tuned the cells to do what we want from the calcium signals, for example tunned the signals to modulate organ size (Soundarrajan et al., 2021). Thanks to technologies like the fluorescent probe GCaMP6f the imaging of the dynamics of Ca^{+2} is possible, revealing interesting dynamics like oscillations, spikes and waves of Ca^{+2} (Gu et al., 1994; Politi et al., 2006; Sanchez et al., 2021; Soundarrajan et al., 2021). In addition to this, recent microscopy insights allow us to see the calcium dynamics in great detail, this is important as the study of this dynamics contain great information of what is happening in the cell; is worth mentioning that recent technologies have allowed us to see this dynamic *in vivo (Friedrich et al., 2021; Stosiek et al., 2003)*.

The great amount of data generated from the imaging of calcium dynamics creates the necessity of the development of analysis technologies to extract quantitative data from them. In the recent years various algorithms for this analysis have been developed, and many implementations have appeared. Algorithms like constrained non-negative matrix factorization (CNMF) and its extended version for 1-photon microscopes (CNMF-E) are used by calcium imaging software to extract the fluorescent profile of the calcium signals (Pnevmatikakis et al., 2016; Zhou et al., 2018). The most used calcium image analysis software are Calcium Image Analysis (CaImAn)(Giovannucci et al., 2019), miniscope 1-photon imaging signal extraction pipeline (Min1pipe) (Lu et al., 2018), EZcalcium (Cantu et al., 2020) and CytoNet (Mahadevan et al., 2022), being CaImAn the most relevant one shown by the great amount of citations and stars in its GitHub profile. From these software we can extract many features, but the most important one is the fluorescent profile of the Regions of Interest (ROIs) identified by the software, in figure 1 you can see the input and

output schema of the analysis tools, also CytoNet generates an analysis of the cellular community that is captured by the image.

Even though these software we can extract the fluorescent profile, none of these generates a robust mathematical analysis of the profile. In this work we present a calcium image analysis pipeline, using CaImAn to extract the fluorescent profile and the softwares SciPy, Numpy, scikit-learn and Matplotlib for mathematical analysis and visualization (Harris et al., 2020; Hunter, 2007; Pedregosa et al., 2011; Virtanen et al., 2020). This pipeline has been tested by analyzing calcium images provided by Zartman Lab probing the usefulness of this tool.

In this document the analysis is going to be done in images of an experiment of the effect of Yoda1 in the Piezo1 channels. Piezo1 is a mechanosensitive non-selective calcium channel, this means that is a channel for calcium that is activated by mechanical force imposed on it (Liao et al., 2021; Volkers et al., 2015). And the drug Yoda1 that activates the Piezo1 channels without the necessity of the mechanical activation, generating an influx of calcium to the cell without imposing a mechanical stress in the Piezo1 channel(Botello-Smith et al., 2019).

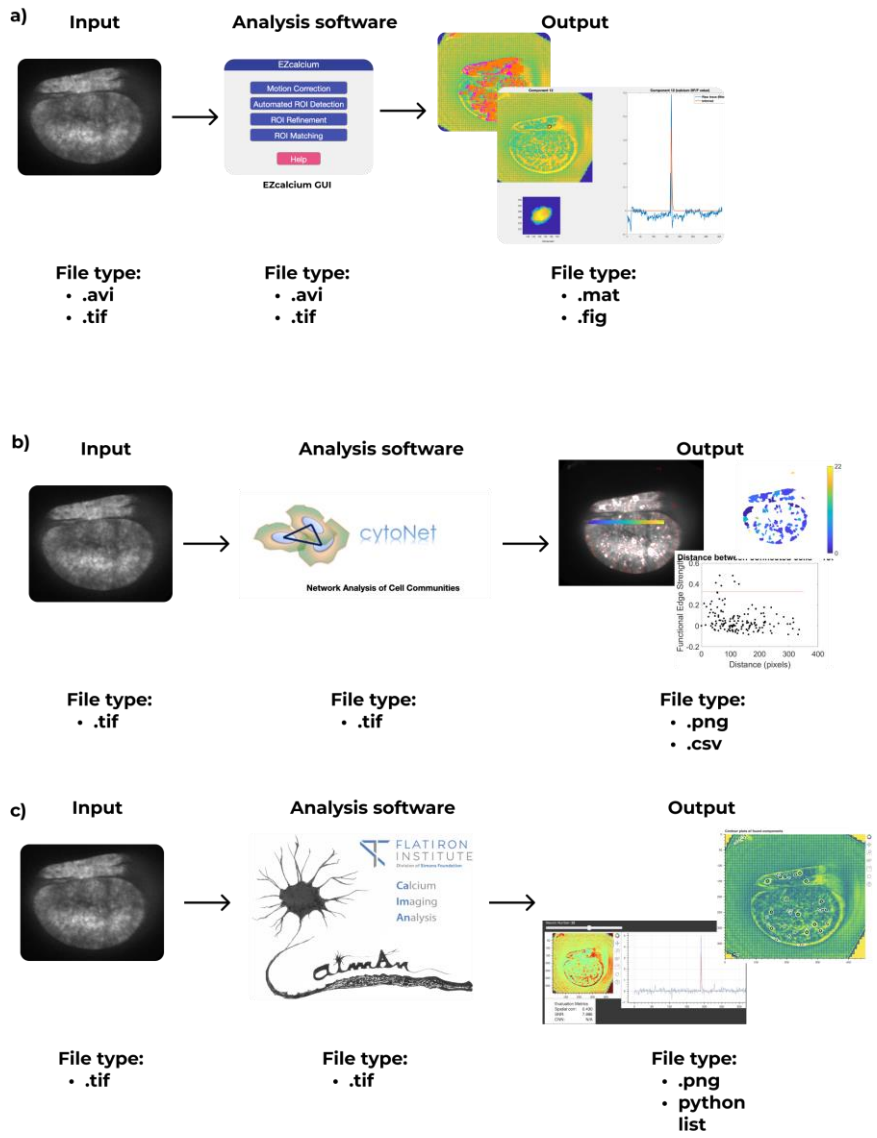

Figure 1: Input and output representation of the different software. a) EZcalcium: The input of this software is a .tif stack containing the microscopy video and the output is the image with the ROIs recognized and the fluorescent profile of the signals of each ROIs with a .mat file that contains the information extracted. b) CytoNet: The input of this software is a .tif stack containing the microscopy video and the output the cellular community analysis done by the software. c) CaImAn: The input of this software is a .tif stack containing the microscopy video and the output is the image with the ROIs recognized and the fluorescent profile of the signals of each ROIs, from the output more information can be extracted, for example the data containing the numerical information of the fluorescent profile.

2. Methodology

The pipeline generated was developed in the environment Google Colab (Bisong, 2019). Figure 2 represents the pipeline by a flow chart.

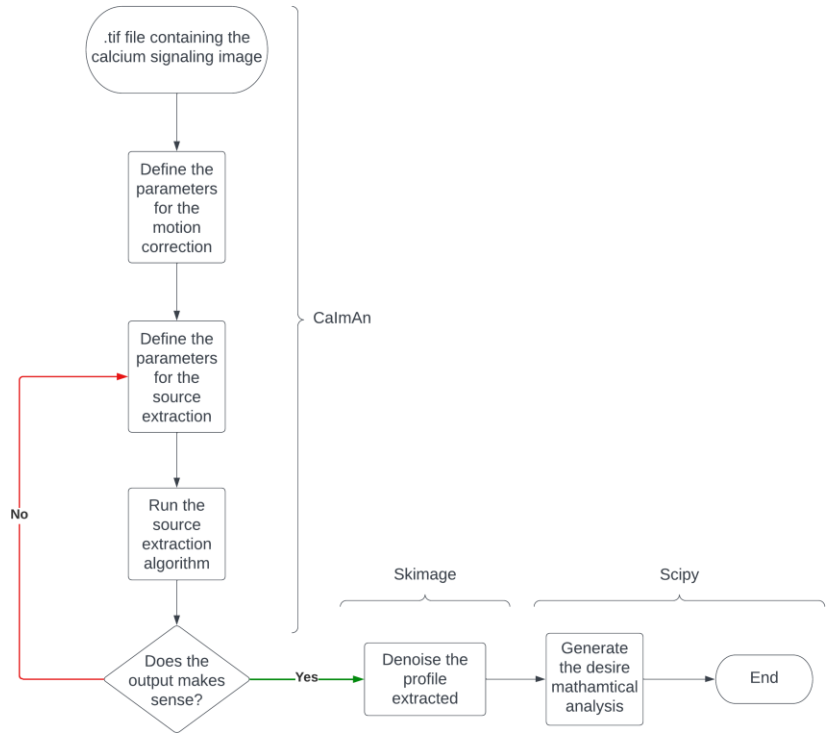

Figure 2: The workflow of the pipeline can be summarized by this flow chart. The software used is listed in the image too. The pipeline was implemented in a .ipynb file using Google Colab, by doing this we take advantage of the shareability of this application. The flowchart was generated with Lucidchart (Faulkner, 2018)

In the following sections we are going to present the pipeline in the order in which is run.

2.1. Initial configurations

The first step of the pipeline is the installation of the required packages and the importation of them to the environment.

```
!pip install git+https://github.com/flatironinstitute/CaImAn.git --quiet #
installation of the package from github
# installation of the packages that this package depends on
!pip install pims --quiet
!pip install pynwb --quiet
!pip install ipyparallel --quiet
!pip install peakutils --quiet
try:
    (magic_name, parameter_s).get_ipython().magic(u'load_ext autoreload')
    (magic_name, parameter_s).get_ipython().magic(u'autoreload 2')
    (magic_name, parameter_s).get_ipython().magic(u'matplotlib qt')
except:
    pass

import logging
import matplotlib.pyplot as plt
import numpy as np
import glob
import scipy
from scipy.special import logsumexp
import caiman as cm
from caiman.source_extraction import cnmf
from caiman.utils.utils import download_demo
from caiman.utils.visualization import inspect_correlation_pnr,
nb_inspect_correlation_pnr
from caiman.motion_correction import MotionCorrect
from caiman.source_extraction.cnmf import params as params
from caiman.utils.visualization import plot_contours, nb_view_patches, nb_plot_contour
from caiman.paths import caiman_datadir
import cv2
from skimage.restoration import denoise_wavelet
import tifffile
import io
import base64
from IPython.display import HTML
import os

from mpl_toolkits import mplot3d


try:
    cv2.setNumThreads(0)
except:
    pass

# these packages will help us to plot the reults. of the analysis
```

```
import bokeh.plotting as bpl
import holoviews as hv
bpl.output_notebook()
hv.notebook_extension('bokeh')
```

Thanks to be working in Google Colab we can access the files in our Google Drive, this is a great advantage because in this way the files are stored in the cloud rather than in the physical storage of the computer, saving memory for other type of files. To connect to Google Drive the following code has to be run.

```
from google.colab import drive
drive.mount('/content/drive')
```

Once initiated the connection with Google Drive, we must define the path to the file that is going to be analyzed, and the path to the folder that contains said file.

```
names = ['path_to_file'] # this variable tells the program where the file is. It is a
list, so if you want to upload one video, you have to
                         # put one path in the list
path_to_model = 'path_to_folder' # this variable stores the path to the folder which
stores the video file. It is also the folder that will store the .mmap files
```

The large number of calculations that the software has to do in order to run the analysis, the usage of parallel computing is needed. In this paradigm the large work is divided in small works that can be solved by parallel processor (Asanovic et al., 2009). To star this process a cluster must be started, to do this we use the following function:

```
# the following code will generate the cluster and will close any pre-existing
cluster, initiating a new one.

if 'dview' in locals():
    cm.stop_server(dview=dview)
c, dview, n_processes = cm.cluster.setup_cluster(
    backend='local', n_processes=None, single_thread=False)
```

2.2. Motion Correction

The first part of the process of image analysis is the motion correction of the images. In this process the motion that the image has due to the movement of the sample is corrected by different algorithms. The algorithm used by the CaImAn package is NoRMCorre (Pnevmatikakis & Giovannucci, 2017). Before running this algorithm, we first have to define the parameters of it.

```
# Parameters which are dependent of the data set
frate = 10                      # The rate of the movie frames
decay_time = 0.4                # This is the lenght of the transient, the units are
seconds

# Parameters of the motion correction process
motion_correct = True    # This will flag the performing motion correction
pw_rigid = False         # This is a flag for the  performing piecewise-rigid motion
correction (otherwise just rigid)
gSig_filt = (3, 3)       # This is the size of high pass spatial filtering, it is used
when we are working with 1p data
max_shifts = (5, 5)      # This parameter marks the maximum allow rigid shift
strides = (48, 48)       # Is going to start a new patch of pw-rigid motion correction
every x pixels
overlaps = (24, 24)      # Quantifies the overlap of patches(size of patch
strides+overlaps)
max_deviation_rigid = 3  # Indicates the maximun deviation allow for a patch with
respect to rigid shift
border_nan = 'copy'      # Indicates the replication along the border
splits_rig = 10          #
shifts_opencv = True  # flag for correcting motion using bicubic interpolation
(otherwise FFT interpolation is used)
# Now we store all the paremeters in a dictionary
mc_dict = {
    'fnames': fnames, # note that here we pass the file path
    'fr': frate,
    'decay_time': decay_time,
    'pw_rigid': pw_rigid,
    'max_shifts': max_shifts,
    'gSig_filt': gSig_filt,
    'strides': strides,
    'overlaps': overlaps,
    'max_deviation_rigid': max_deviation_rigid,
    'border_nan': border_nan,
    'splits_rig': splits_rig,
    'path_to_model': path_to_model, # here we pass tha path to the model's folder
    'shifts_opencv': shifts_opencv
}
```

```
# Finally we store the parameters in an object, this object is going to be read by the
program
opts = params.CNMFParams(params_dict=mc_dict)
```

Once the parameters are set, the motion correction algorithm is run.

```
# If the motion_correct is True then the correction is going to be performed
if motion_correct:
  # thus is the cored that will perform the correction
    mc = MotionCorrect(fnames, dview=dview, **opts.get_group('motion'))
    mc.motion_correct(save_movie=True)
    fname_mc = mc.fname_tot_els if pw_rigid else mc.fname_tot_rig
    if pw_rigid:
        bord_px = np.ceil(np.maximum(np.max(np.abs(mc.x_shifts_els)),
                                     np.max(np.abs(mc.y_shifts_els)))).astype(int)
    else:
        bord_px = np.ceil(np.max(np.abs(mc.shifts_rig))).astype(int)
        # With this we can plot the result
        plt.subplot(1, 2, 1); plt.imshow(mc.total_template_rig)  # % plot template
        plt.subplot(1, 2, 2); plt.plot(mc.shifts_rig)  # % plot rigid shifts
        plt.legend(['x shifts', 'y shifts'])
        plt.xlabel('frames')
        plt.ylabel('pixels')
    # This is going to create the memory map that will store the motion corrected
video
    bord_px = 0 if border_nan == 'copy' else bord_px
    fname_new = cm.save_memmap(fname_mc, base_name='memmap_', order='C',
                               border_to_0=bord_px)
else: # If motion_correct if false, then the video is only going to be stored
      # as a .mmap file
    fname_new = cm.save_memmap(fnames, base_name='memmap_',
                               order='C', border_to_0=0, dview=dview)
```

Finally, we save the motion corrected image in a memory map (.mmap file) to use it later in the source extraction step.

```
# load memory .mmap file
Yr, dims, T = cm.load_memmap(fname_new)
images = Yr.T.reshape((T,) + dims, order='F')
```

2.3. Source Extraction

Source extraction refers to the process in which the Regions of Interest are detected by ROI detection algorithms, in this case the algorithm is GreedyCorr (Zhou et al., 2018). Then the fluorescent profile of these regions is extracted, in this case using the CNMF-E algorithm. In this process a fine tuning of the parameters of the CNMF algorithm has to be done to generate an extraction that fits what is seen in the images. For this process, the amount of noise that the image contains needs to be analyzed, a good way to do this is using the "Plot Z-axis profile" available in the image analysis software ImageJ.

Before the algorithm of is run, the parameters of it have to be defined, to do this the following code block has to be run. The description of these parameters is defined in the commentaries of the code.

```
# Set of paremeters that will define the source extraction and deconvolution
p = 1                # This defines the order of the autoregressive system
K = None             # This defines the upper limit of the number of components per
patch, in general this parameter is define as None
gSig = (3, 3)        # Gaussian width of a 2D gaussian kernel, this approximates a
neuron
gSiz = (13, 13)      # average diameter of a neuron, is useful to define it as 4*gSig+1
Ain = None           # The possibility to seed the analysis with a binary mask
merge_thr = .7       # merging threshold, this threshold is defined as the maximum
correletion
rf = 40              # half-size of the patches in pixels. e.g., if rf=40, patches are
80x80
stride_cnmf = 20     # amount of overlap between the patches in pixels
#                      (keep it at least large as gSiz, i.e 4 times the neuron size
gSig)
tsub = 2             # downsampling factor in time for initialization,
#                      increase if you have memory problems
ssub = 1             # downsampling factor in space for initialization,
#                      increase if you have memory problems
#                      you can pass them here as boolean vectors
low_rank_background = 2  # None leaves background of each patch intact,
#                      True performs global low-rank approximation if gnb>0
gnb = 2              # number of background components (rank) if positive,
#                      else exact ring model with following settings
#                          gnb= 0: Return background as b and W
#                          gnb=-1: Return full rank background B
#                          gnb<-1: Don't return background
```

```
nb_patch = 2         # number of background components (rank) per patch if gnb>0,
#                      else it is set automatically

# the following parameters will help you to optimze the image analysis
# if the image is too noisy is recommendable to lower the paremters
# if there are too many regions of interst (ROI) is recommendable to increase this
values
min_corr = 0.9       # min peak value from correlation image
min_pnr = 0.9         # min peak to noise ration from PNR image, This value should be
grater than 1
                     # to have more signal than noise. This is because the ratio is
signal/noise
                     # So values bigger than 1 express profiles where the signal is
more intense than the
                     # noise generated by the image it self

# this parameter is used to analyze the sample further more and optimze even more the
ROIs,
# is not always advisable to increase it so much, 2 is a good number
ssub_B = 2           # additional downsampling factor in space for background

# this parameter defines the ring size of the ROIs
ring_size_factor = 1  # radius of ring is gSiz*ring_size_factor

# now we change the paramters stored in the previous object created for the motion
correction
# algorithm
opts.change_params(params_dict={'method_init': 'corr_pnr',  # use this for 1 photon
                                'K': K,
                                'gSig': gSig,
                                'gSiz': gSiz,
                                'merge_thr': merge_thr,
                                'p': p,
                                'tsub': tsub,
                                'ssub': ssub,
                                'rf': rf,
                                'stride': stride_cnmf,
                                'only_init': True,    # set it to True to run CNMF-E
                                'nb': gnb,
                                'nb_patch': nb_patch,
                                'method_deconvolution': 'oasis',       # could use
'cvxpy' alternatively
                                'low_rank_background': low_rank_background,
                                'update_background_components': True,  # sometimes
setting to False improve the results
                                'min_corr': min_corr,
```

```
                                'min_pnr': min_pnr,
                                'normalize_init': False,               # just leave as
is
                                'center_psf': True,                    # leave as is
for 1 photon
                                'ssub_B': ssub_B,
                                'ring_size_factor': ring_size_factor,
                                'del_duplicates': True,                # whether to
remove duplicates from initialization
                                'border_pix': bord_px})                # number of
pixels to not consider in the borders)
```

Before running the source extraction algorithm, we can inspect the parameters defined to see if everything is in order.

```
# with this function we generate some summary of the image (correlation and peak to
noise ratio)
cn_filter, pnr = cm.summary_images.correlation_pnr(images[::1], gSig=gSig[0],
swap_dim=False) # change swap dim if output looks weird, it is a problem with tiffile
# inspect the summary images and set the parameters
nb_inspect_correlation_pnr(cn_filter, pnr)
```

To run the algorithm the following code has to be run. This algorithm might take some time, but the analysis generates a variety of ROIs with high quality.

```
# the following code block runs the algorithm

cnm = cnmf.CNMF(n_processes=n_processes, dview=dview, Ain=Ain, params=opts)
cnm.fit(images)
```

To filter the quality of the profile extracted the following code must be run.

```
# parameters that will filter the components identified
# You can change this parameters in order to change the number of componentes accepted
min_SNR = 3            # adaptive way to set threshold on the transient size
r_values_min = 0.3     # threshold on space consistency (if you lower more components
#                        will be accepted, potentially with worst quality). The higher
the value
                       # the higher the probability of the component to be a neuron.
                       # So if the cells are very different from neuron the value
should be lower
```

```
cnm.params.set('quality', {'min_SNR': min_SNR,
                           'rval_thr': r_values_min,
                           'use_cnn': False})
# this will generates the filtering of componentes
cnm.estimates.evaluate_components(images, cnm.params, dview=dview)

# now we can see how many of them were identified and how meny where accepted
print(' ***** ')
print('Number of total components: ', len(cnm.estimates.C))
print('Number of accepted components: ', len(cnm.estimates.idx_components))
```

This code will output the number of components detected by the CNMF-E algorithm and the number of components that were accepted after being filtered.

Finally, the $F/\Delta F$ profile can be extracted using the following code block.

```
cnm.estimates.detrend_df_f(quantileMin=8, frames_window=250)

# If you get the error "RuntimeError: invalid percentile" change the values of the
parameters of the function,
# but not go to far from the defoult values
```

2.4. Mathematical Analysis

The mathematical analysis developed to extract important mathematical features of the calcium signals. The features extracted are the number of peaks, the width of half maximum, height of the peak and the frequency of oscillation of the calcium signals. The code developed for this part relies strongly in mathematical analysis software like SciPy (Virtanen et al., 2020) and numpy (Harris et al., 2020), for the visualization features of the functions generated use the package Matplotlib for the generation of the figure (Hunter, 2007).

The functions generated can be found in the following code block:

```
##### function that generates the amount of peaks of the signal
```

```
def generate_peaks(graph, height=0, plot=False, number=False, distance = 1 ):
  '''
  input:
  graph : nd.array
          the nd.array that stores the temporal values of the DF/F profile
  height: int
          indicates the minimal required to interpretate as a signal
  plot: bool
        Indicates whether you like to output the plot of the peaks and signal or not
  number: bool
          Indicate whether you like to output the number of peaks or not
  distance: int
            This parameters define the distance of the peaks in order to consider them
as peaks.
            for example, if we consider [0,0,0,4,0,8]
            if distance is 1: 4 is considered a peak. But if distance is 2: 4 is not
            considered as a peak, because at a distance of 2 is other number higher
than 4.
  output:
  n_peaks: int
           the number of peaks of the graph in the timeframe of evaluation
  '''

  # we extract the peaks using the function of scipy
  peaks,_= scipy.signal.find_peaks(graph, height=height , distance = distance)
  # calculate the length of the array of peaks in order to know the quantity of them
  n_peaks = len(peaks)
  # now we plot a graph containing the peaks detected
  if plot == True:
    plt.figure()
    plt.plot(graph)
    plt.plot(peaks, graph[peaks],"x")
  # if number is true you can have a string saying the number of peaks
  if number==True:
    print("the number of peaks is "+ str(n_peaks))
  return n_peaks
###### function that generates the seconds and images of the system
#
def seconds_images(fname):
  '''
  input:

  fname: string
         a string containing the path to the .tif file to analyze
  output:
  seconds_interval: int
                    The amount of seconds between to frames
  n_images: int
            The amoun of images in the .tif file

  seconds: int
           the length of the video in seconds
  considerations:
    for a better analysis make sure that your .tif file contains in its metadata
    the interval of seconds between each frame, or in the frames are labeled as the
    second in which it was taken. Other types of metadata cannot be analyze due to the
    lack of the temporal parameter.
  '''
```

```
  with tifffile.TiffFile(fname) as tif: # we read the tiff file of the images to
extract the seconds interval
    volume = tif.asarray()
    axes = tif.series[0].axes
    imagej_metadata = tif.imagej_metadata

  keys_metadata= list(imagej_metadata.keys())

  # if the .tif file stores the amount of seconds between images
  if'finterval' in keys_metadata: # if the metadata contains the interval between the
frames
    seconds_interval=imagej_metadata['finterval'] # the seconds between two frames
    n_images=imagej_metadata['images']
    return seconds_interval, n_images
  # if the .tif file stores the seconds that took to take the video as labels of the
images
  else:  # if the label of the metadata frames contains the amount of seconds that
took
         # to take the pictures
    labs = imagej_metadata['Labels']
    seconds = labs[-1].strip(' s')
    seconds_interval = float(labs[1].strip(' s'))-labs[0].strip(' s')
    n_images = imagej_metadata['images']
    return seconds_interval, n_images



####### this function generates the frequency of peaks of the signal
def generate_frequency(graph, sec_img,height=0, distance = 1):
  '''
  input:
  graph : nd.array
          the nd.array that stores the temporal values of the DF/F profile
  sec_img: tuple
           tuple that contains the seconds between two frames and the total images
           in the .tif file
  height: int
          indicates the minimal required to interpretate as a signal
 distance: int
           This parameters define the distance of the peaks in order to consider them
as peaks.
           for example, if we consider [0,0,0,4,0,8]
           if distance is 1: 4 is considered a peak. But if distance is 2: 4 is not
           considered as a peak, because at a distance of 2 is other number higher
than 4.
  output:
  frequency: int
             the frequency of the peaks
  '''
  # ways to extract and calculate the duration of the calcium image video
  if len(sec_img) == 2:
    duration = sec_img[0] * (sec_img[1]-1) # calculate the duration of the video
  if len(sec_img) == 1:
    duration = float(sec_img[0])
  # generate the number of peaks in order to calculate the frequency
  n_peaks = generate_peaks(graph,height=height, plot=False, number=False, distance =
distance)
  # formula for the number of peaks, this is in peaks/seconds
```

```
  frequency= n_peaks/duration
  return frequency
###### this function generates the average height of the peaks in the signal
expression
def generate_average_hpeak(graph, height=0, distance = 1):
  '''
  input:
  graph : nd.array
          the nd.array that stores the temporal values of the DF/F profile
  height: int
          indicates the minimal required to interpretate as a signal
  distance: int
           This parameters define the distance of the peaks in order to consider them
as peaks.
           for example, if we consider [0,0,0,4,0,8]
           if distance is 1: 4 is considered a peak. But if distance is 2: 4 is not
           considered as a peak, because at a distance of 2 is other number higher
than 4.

  output:
  average: int
            The average height of the calcium peaks
  '''
  # we extract the properties of the peaks using the function of scipy
  peaks,properties= scipy.signal.find_peaks(graph, height=height, distance = distance)
  height_peaks = properties['peak_heights']
  # we calculate the average height of the peaks
  average= np.mean(height_peaks)
  return average
#### this function generates the width of the half of the peak
def generate_whm(graph, sec_img,height=0,distance = 1,plot=False):
  '''
  input:
  graph : nd.array
          the nd.array that stores the temporal values of the DF/F profile
  height: int
          indicates the minimal required to interpretate as a signal
  distance: int
           This parameters define the distance of the peaks in order to consider them
as peaks.
           for example, if we consider [0,0,0,4,0,8]
           if distance is 1: 4 is considered a peak. But if distance is 2: 4 is not
           considered as a peak, because at a distance of 2 is other number higher
than 4.

  output:
  whm: nd.array
       indicates the whm of the peaks
  '''
  # we extract the properties of the peaks, between them the weight
  peaks,properties= scipy.signal.find_peaks(graph, height=height, distance = distance)
  # we extract the width of the peaks
  width_peak=scipy.signal.peak_widths(graph, peaks,rel_height=0.5)
  # if there are no peaks, we define the width as 0
  if len(peaks) == 0:
    whm = [0]
  else:
    whm = width_peak[0]
```

```
  # plot the profile with the width of the medium height
  if plot==True:
    results_half = scipy.signal.peak_widths(graph, peaks, rel_height=0.5)
    plt.figure()
    plt.plot(graph)
    plt.plot(peaks, graph[peaks], "x")
    plt.hlines(*results_half[1:], color="C2")
  whm = np.array(whm) * float(sec_img[0])
  return whm
# this function generates the analysis of the fluorescent expression of the
# components of the tissue
def analyze_df_f(spatial_components,  fname, initial_idx=0, final_idx=0, idx_comp=[],
height = 0, distance = 1,idx = 'all' ):
  '''
  input:

  spatial_components: nd.array
                      an nd.array which contains the graphs of the DF_F profile
  fname: string
         string containing the path to the .tif file to analyze
  initial_idx: int
               intial component of the interval to analyze. Only use if idx ==
'interval'
  final_idx: int
             final component of the interval to analyze. Only use if idx ==
'interval',
             containing this last one
  idx_comp: list
            list of the indexes of the components to analyze
            if you want the component number i use i as input
  height: int
          indicates the minimal required to interpretate as a signal
  distance: int
            This parameters define the distance of the peaks in order to consider them
as peaks.
            for example, if we consider [0,0,0,4,0,8]
            if distance is 1: 4 is considered a peak. But if distance is 2: 4 is not
            considered as a peak, because at a distance of 2 is other number higher
than 4.
  idx: string
       indicate the interval type of the components
          'all': Analize all the components
          'interval': indicate the interval of components to analyze. If this is
                      the case indicate the initial and final component index.
          'specific': indicate the specific number of the components to analyze.
  output:

  analysis: dict
            a nested dictionary that indicates the parameters of each component
            'numnumber_of_peaks': number of peaks in the time frame
            'fraquency_of_peaks': frequency of the peaks over the time
            'whm_of_peaks': whm of the peaks
            'peaks_average_height': the average height of the peaks
  '''
  analysis=dict()
  sec_image=seconds_images(fname)

  # we generate a different array of index in order to analyse the desire components
```

```
  if idx == 'all':
    index = range(len(spatial_components))
  elif idx == 'interval' and (bool(initial_idx)or initial_idx==0) and bool(final_idx)
:
    index = range(initial_idx, final_idx+1)
  elif idx == 'specific' and bool(idx_comp):
    index = np.array(idx_comp)-1
############## analysis loop ##############
  for i in index:
    # we generate the analysis for each component extracted
    graph=spatial_components[i]
    n_peaks=generate_peaks(graph,height=height, distance = distance)
    freq=generate_frequency(graph,sec_image,height=height, distance = distance)
    peak_whm=generate_whm(graph,sec_img = sec_image,height=height)
    peak_aver=generate_average_hpeak(graph, height = height , distance = distance)
    results={'number_of_peaks': n_peaks,
             'frequency_of_peaks': freq,
             'whm_of_peaks':peak_whm,
             'peaks_average_height':peak_aver}
    analysis[i+1]=results
  return analysis
#### this function generates the average of the parameters calculates by the
#### function analyze_df_f
def average_parameters(dict_results):
  '''
  input:
  dict_results: dict
                dictionary containing the information of each cell identified by the
analysis tool.
  output:

  out: dict
       dictionary containing the information asked, it has the average resul of the
       parameters calculated by the function analyze_df_f
       outpus:
          average_number_of_peaks
          verage_frequency_of_peaks
          average_whm_of_peaks
          average_average_heigh_of_peaks
  '''
  # define the initial variables in order to found out the total of them
  num_peaks = 0
  freq_peaks = 0
  whm_of_peaks = 0
  hg_peaks = 0
  comps=len(dict_results)
  for i in range(len(dict_results)):
    # we run through every component in the results dictionary
    num_peaks+=dict_results[i+1]['number_of_peaks']
    freq_peaks += dict_results[i+1]['frequency_of_peaks']
    whm_of_peaks +=
logsumexp(dict_results[i+1]['whm_of_peaks'])/len(dict_results[i+1]['whm_of_peaks'])
    hg_peaks += dict_results[i+1]['peaks_average_height']
  # the output is made of the average number, taking in consideration the total
extracted before
  out = {'average_number_of_peaks':num_peaks/comps,
         'average_frequency_of_peaks':freq_peaks/comps,
         'average_whm_of_peaks':whm_of_peaks/comps,
```

```
        'average_average_heigh_of_peaks':hg_peaks/comps}
 return out
```

With this function we can extract important features of the calcium oscillations, these important features are number of peaks, frequency of peaks, width of half max of the peaks and the height of the peaks. Also, we can calculate the average of the parameters of each of the tissues that we are analyzing.

2.5. Piezo1 and Yoda1 experiments

A set of experiments were performed in the wing disc of *Drosophila melanogaster* involving the interaction between the Yoda1 drug and the Piezo1 channel. These experiments are not part of this study but were provided by the experimental team of Zartman Lab for their analysis.

The experiments were performed in three genetic lines of *Drosophila* flies, one without alterations in the Piezo1 channels, one with the over expression of this channels, and the final one with the knockdown of them by the implementation of the RNAi (interference RNA) called PiezoRNAi. Then this fly lines were cultivated in absence of Yoda1 and in presence of Yoda1 (1uM). This generated 6 different conditions in which we can analyze the interactions of Yoda1 and Piezo1 with the calcium dynamics in the wing disc. The images of the calcium activity were taken using a 1-photon confocal microscope.

3. Results & Discussion

The principal insight of this program is the facility to generate important analysis from the data captured by confocal microscopy and it facilitates the analysis for non-programmer wet biologists.

Several images were analyzed using this pipeline, showing a high put through extraction of the parameters that were required joined with a capability to generate important images for the

publication of the insights. First, the frequency of oscillation of the components of the wing disc of *Drosophila melanogaster,* this analysis is shown in figure 3.

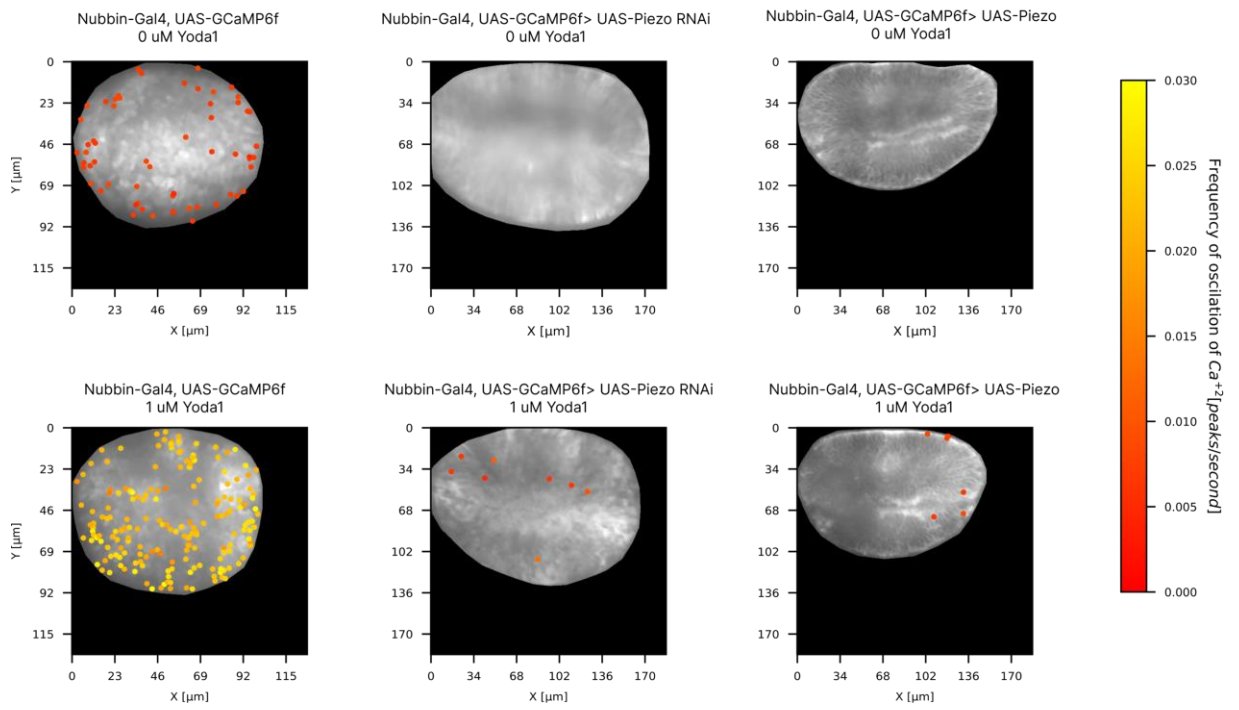

Figure 3: Frequency analysis of the components detected by the program CaImAn. The components detected are represented by the dots. The frequency of said components is represented by the color of them, following the color scheme that is shown in the color bar on the right. Each picture was taken using a confocal 1-photon microscope.

From this analysis we can notice that Yoda1 increases the number of components that are oscillating and the frequency of them, as we can see from the analysis of the tissue with and without yoda1 with no genetic alteration (the ones in the first column), showing that Yoda1 activates the Piezo1 channels, enhancing the entrance of calcium to the cell, generating more oscillation of calcium over time. Also, we can see that, even though the Piezo1 channels are inhibited by PiezoRNAi, in the presence of Yoda1 the activity of calcium increases generating the components that are detected by the program. Furthermore, we can see that the number of components detected in the PiezoRNAi sample is bigger than the one with Piezo1 over expression

sample, elucidating some kind of interaction between Yoda1 and the increased number of Piezo1 Channels that inhibits the activity of calcium oscillations.

Other types of analysis can be generated, figure 4 shows the mathematical analysis of the effects of Yoda1 over time in a sample with normal expression of Yoda1.

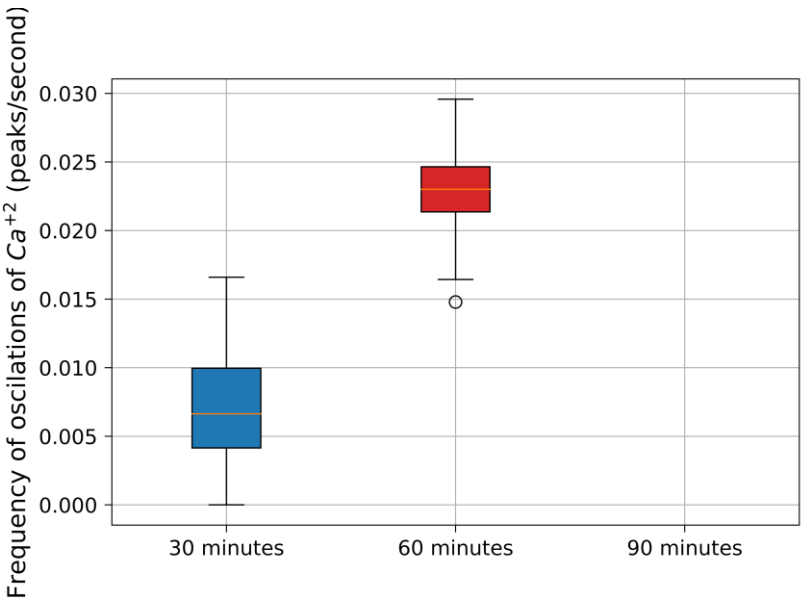

Figure 4: Box plot of the frequency of oscillation of calcium over different time points. The frequency extraction of the tissue was performed by analyzing images that were taken 30 minutes after cultivation, 60 minutes after cultivation and 90 minutes after cultivation. The last column does not present points because the CaImAn program didn't detect any component that presented calcium activity.

In figure 4, we can see that the activity of Piezo1 channels increases over time, having a peak at 60 minutes. But at 90 minutes we can see no activity of calcium, showing that the effect of Yoda1 over the Piezo1 channels do not function during great time spans. We can see then that the activity of Yoda1 increases over time until it reaches a peak, then it decreases rapidly until there is no more calcium activity.

4. Conclusion

We have shown the capability of the pipeline generated to analyze calcium images. From the analysis developed and the data extracted we can generate figures to communicate the insights that the experiments uncover. Specifically, here we have analyzed the experiment of effect of Yoda1 over the Piezo1 channels in the wing disc of *D. melanogaster*, and from the images generated from the data extracted with the pipeline and the capabilities of the package matplotlib (Hunter, 2007), we can generate deep analysis of what is happening in the tissue thanks to the quantitative data. This shows the importance of the tool generated because it will enable wet scientist to generate data analysis without the necessity of having computational skills.

Further work is yet needed in this area due to the lack of tools that analyze phenomena in non-neuron like cells because the software used, CaImAn, could generate the analysis because the cell shape of the cells that compose the wing disc are like neurons. But, in cells with a morphology that differs a lot from neurons the analysis pipeline might fail, this is the case of plant cells. A further study in this area is needed, and the necessity of tools that attain the problem of cell segmentation joined with source extraction in plant cells is high. Other tools that are needed in this area are tools that can analyze calcium waves in tissue or cells, this is important that is has been shown that this waves are important in different processes, for example organ growth (Soundarrajan et al., 2021).

Acknowledgment

The autors give thanks to the team that welcomed me in the University of Notre Dame, specifically to Nilay Kumar, Mayesha Mim, David Gazzo, Maria Unger, Shuman Liu, and special thanks to Dr. Jeremiah Zartman who opened the doors of his laboratory to me. Also, the authors would like to give thanks to Mayesha Mim and Nilay Kumar for providing the images for the analysis and testing

of the pipeline. And a thanks to the School of Engineering of Pontificia Universidad Católica de Chile for financing this opportunity.

Referencias

Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D., & Yelick, K. (2009). A view of the parallel computing landscape. *Communications of the ACM*, *52*(10), 56–67. https://doi.org/10.1145/1562764.1562783

Berridge, M. J. (1997). The AM and FM of calcium signalling. *Nature*, *386*(6627), 759–760. https://doi.org/10.1038/386759a0

Berridge, M. J., Lipp, P., & Bootman, M. D. (2000). The versatility and universality of calcium signalling. *Nature Reviews Molecular Cell Biology*, *1*(1), 11–21. https://doi.org/10.1038/35036035

Bisong, E. (2019). *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Apress. https://doi.org/10.1007/978-1-4842-4470-8

Botello-Smith, W. M., Jiang, W., Zhang, H., Ozkan, A. D., Lin, Y.-C., Pham, C. N., Lacroix, J. J., & Luo, Y. (2019). A mechanism for the activation of the mechanosensitive Piezo1 channel by the small molecule Yoda1. *Nature Communications*, *10*(1), 4503. https://doi.org/10.1038/s41467-019-12501-1

Cantu, D. A., Wang, B., Gongwer, M. W., He, C. X., Goel, A., Suresh, A., Kourdougli, N., Arroyo, E. D., Zeiger, W., & Portera-Cailliau, C. (2020). EZcalcium: Open-Source Toolbox for Analysis

of Calcium Imaging Data. *Frontiers in Neural Circuits*, *14*, 25. https://doi.org/10.3389/fncir.2020.00025

Clapham, D. E. (2007). Calcium Signaling. *Cell*, *131*(6), 1047–1058. https://doi.org/10.1016/j.cell.2007.11.028

Faulkner, A. & Contributor. (2018). Lucidchart for easy workflow mapping. *Serials Review*, *44*(2), 157–162.

Friedrich, J., Giovannucci, A., & Pnevmatikakis, E. A. (2021). Online analysis of microendoscopic 1-photon calcium imaging data streams. *PLOS Computational Biology*, *17*(1), e1008565. https://doi.org/10.1371/journal.pcbi.1008565

Giovannucci, A., Friedrich, J., Gunn, P., Kalfon, J., Brown, B. L., Koay, S. A., Taxidis, J., Najafi, F., Gauthier, J. L., Zhou, P., Khakh, B. S., Tank, D. W., Chklovskii, D. B., & Pnevmatikakis, E. A. (2019). CaImAn an open source tool for scalable calcium imaging data analysis. *ELife*, *8*, e38173. https://doi.org/10.7554/eLife.38173

Gu, X., Olson, E., & Spitzer, N. (1994). Spontaneous neuronal calcium spikes and waves during early differentiation. *The Journal of Neuroscience*, *14*(11), 6325–6335. https://doi.org/10.1523/JNEUROSCI.14-11-06325.1994

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., & Smith, N. J. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(03), 90–95.

Liao, J., Lu, W., Chen, Y., Duan, X., Zhang, C., Luo, X., Lin, Z., Chen, J., Liu, S., Yan, H., Chen, Y., Feng, H., Zhou, D., Chen, X., Zhang, Z., Yang, Q., Liu, X., Tang, H., Li, J., … Wang, J. (2021). Upregulation of Piezo1 (Piezo Type Mechanosensitive Ion Channel Component 1) Enhances the Intracellular Free Calcium in Pulmonary Arterial Smooth Muscle Cells From Idiopathic Pulmonary Arterial Hypertension Patients. *Hypertension*, *77*(6), 1974–1989. https://doi.org/10.1161/HYPERTENSIONAHA.120.16629

Lu, J., Li, C., Singh-Alvarado, J., Zhou, Z. C., Fröhlich, F., Mooney, R., & Wang, F. (2018). MIN1PIPE: A Miniscope 1-Photon-Based Calcium Imaging Signal Extraction Pipeline. *Cell Reports*, *23*(12), 3673–3684. https://doi.org/10.1016/j.celrep.2018.05.062

Mahadevan, A. S., Long, B. L., Hu, C. W., Ryan, D. T., Grandel, N. E., Britton, G. L., Bustos, M., Gonzalez Porras, M. A., Stojkova, K., Ligeralde, A., Son, H., Shannonhouse, J., Robinson, J. T., Warmflash, A., Brey, E. M., Kim, Y. S., & Qutub, A. A. (2022). cytoNet: Spatiotemporal network analysis of cell communities. *PLOS Computational Biology*, *18*(6), e1009846. https://doi.org/10.1371/journal.pcbi.1009846

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., & Dubourg, V. (2011). Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, *12*, 2825–2830.

Pnevmatikakis, E. A., & Giovannucci, A. (2017). NoRMCorre: An online algorithm for piecewise rigid motion correction of calcium imaging data. *Journal of Neuroscience Methods*, *291*, 83–94. https://doi.org/10.1016/j.jneumeth.2017.07.031

Pnevmatikakis, E. A., Soudry, D., Gao, Y., Machado, T. A., Merel, J., Pfau, D., Reardon, T., Mu, Y., Lacefield, C., Yang, W., Ahrens, M., Bruno, R., Jessell, T. M., Peterka, D. S., Yuste, R., & Paninski, L. (2016). Simultaneous Denoising, Deconvolution, and Demixing of Calcium Imaging Data. *Neuron*, *89*(2), 285–299. https://doi.org/10.1016/j.neuron.2015.11.037

Politi, A., Gaspers, L. D., Thomas, A. P., & Höfer, T. (2006). Models of IP3 and Ca2+ Oscillations: Frequency Encoding and Identification of Underlying Feedbacks. *Biophysical Journal*, *90*(9), 3120–3133. https://doi.org/10.1529/biophysj.105.072249

Sanchez, C., Berthier, C., Tourneur, Y., Monteiro, L., Allard, B., Csernoch, L., & Jacquemond, V. (2021). Detection of Ca2+ transients near ryanodine receptors by targeting fluorescent Ca2+ sensors to the triad. *Journal of General Physiology*, *153*(4), e202012592. https://doi.org/10.1085/jgp.202012592

Soundarrajan, D. K., Huizar, F. J., Paravitorghabeh, R., Robinett, T., & Zartman, J. J. (2021). From spikes to intercellular waves: Tuning intercellular calcium signaling dynamics modulates organ size control. *PLOS Computational Biology*, *17*(11), e1009543. https://doi.org/10.1371/journal.pcbi.1009543

Stosiek, C., Garaschuk, O., Holthoff, K., & Konnerth, A. (2003). *In vivo* two-photon calcium imaging of neuronal networks. *Proceedings of the National Academy of Sciences*, *100*(12), 7319–7324. https://doi.org/10.1073/pnas.1232232100

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., & Bright, J. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272.

Volkers, L., Mechioukhi, Y., & Coste, B. (2015). Piezo channels: From structure to function. *Pflügers Archiv - European Journal of Physiology*, *467*(1), 95–99. https://doi.org/10.1007/s00424-014-1578-z

Zhou, P., Resendez, S. L., Rodriguez-Romaguera, J., Jimenez, J. C., Neufeld, S. Q., Giovannucci, A., Friedrich, J., Pnevmatikakis, E. A., Stuber, G. D., Hen, R., Kheirbek, M. A., Sabatini, B. L., Kass, R. E., & Paninski, L. (2018). Efficient and accurate extraction of in vivo calcium signals from microendoscopic video data. *ELife*, *7*, e28728. https://doi.org/10.7554/eLife.28728

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERÍA
Dirección de Investigación e Innovación
Programa IPre de Investigación en Pregrado