tridesclous Documentation

Release 1.4.2

Samuel Garcia, Christophe Pouzat

Jan 27, 2020

Contents

If this is your first time here, after admiring the logo, you should go in *Overview* and *Step by step quickstart*.

CHAPTER 1

Contents

1.1 Overview

tris des clous is a very dishonest translation of **spike sorting** to French.

Pronouce it [tree day clue] in English.

The primary goal of tridesclous was to provide a toolkit to teach good practices in spike sorting techniques. Trideslcous is now mature and can be used to sort spike on tetrode up to neuropixel probe recorded dataset.

Tridesclous is both:

- an offline spike sorter tools able to read many file format
- a realtime spike sorting combined with pyacq

1.1.1 Main features

- template matching based mathod
- offer several alternative methods at several processing steps of the chain
- offer a UI written in Qt for interactive exploration.
- use neo for reading dataset. So many format are available (Raw, Blackrock, Neuralynx, Plexon, Spike2, Tdt, OpenEphys...)
- use hardware acceleration with opencl : both gpu and multicore cpu
- use few memory
- have built in dataset to try it
- quite fast For a tetrode dataset, you can expect X30 speedup over real time on a simple laptop.
- have an simple python API. Easy to write notebook or build custom pipeline.
- multi-platform

- open source based on a true open source stack

The forest of spike sorting tools is dense and *tridesclous* is a new tree. Be curious and try it.

1.1.2 General workflow

Many tools claim to be "automatic" for spike sorting. In tridesclous we don't, the workflow is:

1. Construct catalog. This part is automatic but needs carefully chosen parameters/methods. This is more or less the legacy chain of spike sorting = preprocessing+waveform+feature+clustering This can be done with a small subset of the whole dataset as long as it is stationary.
2. Check and correct the catalog. **This part is manual.** It is done through a user interface. Multiple views in the interface help the end user make good decisions: change the threshold, enlarge waveform shape, change feature method, change clustering algorithm and of course merge and split clusters. This part is crucial and must be performed to clean clusters.
3. "Peel spikes". This is the real spike sorting. It is a template matching approach that subtracts spikes for signals as long as some spike matches the catalog. This part can be run offline as fast as possible or online (*soft* real time) if fast enough.
4. Check the final result with a dedicated user interface. No manual action is required here.

Manual checking part **2** and **4** can be optional if you like to use black-box style spike sorting tools.

Why is it different from other tools:

- Today, some other tools propose more or less the same workflow except the central step: check the catalog before the template matching! This is critical. These tools often over split some clusters and this leads to long, useless and uncontrolled manual merges (or split sometimes) at the end of the process.
- The catalog is built only from a small part of the dataset, let say some minutes. Some other tools try to cluster spike on long recording but there are many chances that signal, noise, amplitude will not be stationary. Clustering on a small part is faster and leads to more stable results.
- The user interface (for catalogue check/correct and peeler check) is part of the same package. So viewers are closely linked to methods and everything is done to alleviate the pathologies of these methods.

1.1.3 Online spike sorting

If you have a pyacq compatible device (Blackrock, Multi channel system, NiDaqMx, Measurement computing, . . .) you can also test tridesclous online during the experiment. See online_demo.py

In pyacq, you can build your own viewers in a custom "Node", so you should be able to monitor during the recording what you need (receptive field, . . .)

1.2 Installation

If your are familiar with python simply install the dependency list as usual.

tridesclous works with python 3 only.

If these are your first steps in the python world there 2 main options:

- install python and dependencies with anaconda distribution (prefered on window or OSX)
- use python from your system (in a virtual environement) and install dependencies with standard pip (prefered on Linux Ubuntu/Debian/Mint)

Note that you are free to install Anaconda on Linux but conda add an heavy layer of package management in parallel of your linux distro pacake management that can be messy.

Here 2 recipes to install tridesclous in an "environement". If you don't known what an "environement" is : remember that it is an isolated installation of python with a fixed version and many libraries (modules) with also fixed version somwhere in a folder in your profile. This folder won't be affected by upgrading your system and so it should work always. This quite important because other spike projects don't use same libraries version (for instance PyQt4 vs PyQt5). If you want to compare them, you will need environement.

1.2.1 Case 1 : with anaconda (prefered on window or OSX)

Do:

1. Download anaconda here https://www.continuum.io/downloads. Take **python 3.7**
2. Install it in user mode (no admin password)
3. Lanch **anaconda navigator**
4. Go on the tab **environements**, click on **base** context menu.
5. **Open Terminal**
6. For the basic:

```
conda install scipy numpy pandas scikit-learn matplotlib seaborn tqdm openpyxl␣
↪numba
conda install -c conda-forge hdbscan
```

7. For GUI and running example:

```
conda install pyqt=5 jupyter
pip install pyqtgraph==0.10 quantities neo
```

8. And finally install tridesclous from github:

```
pip install https://github.com/tridesclous/tridesclous/archive/master.zip
```

Optional if you're up for a fight and you really want fast computing with OpenCL:

1. install driver for GPU (nvidia/intel/amd), this is quite hard in some cases because you need to download some OpenCL (or cuda) toolkit.
2. Download PyOpenCl here for windows : http://www.lfd.uci.edu/~gohlke/pythonlibs/
3. cd C:/users/...../Downloads
4. pip install pyopencl-2019.1.2+cl12-cp37-cp37m-win_amd64.whl

Warning: Some user with windows report strong problems. Anaconda is hard to install and also in the tridesclous GUI, when a file dialog should open python surddenly crash. One possible reason is : on Dell computer an application **Dell Backup and Recovery** is installed. This application also used Qt5. For some versions (1.8.xx and maybe others) of **Dell Backup and Recovery** this Qt5 have bug and theses Qt5 ddl are mixed up with anaconda Qt5, this lead to a total mess hard to debug. So if you have a Dell, you should upgrade **Dell Backup and Recovery** or remove it.

1.2.2 Case 2 : with pip (prefered on linux)

Here I propose my favorite method that install tridesclous with debian like distro in an isolated environement with virtualenvwrapper. Every other method is also valid.

Open a terminal and do:

1. sudo apt-get install virtualenvwrapper
2. mkvirtualenv tdc –python=/usr/bin/python3.6 (or python3.5)
3. workon tdc
4. pip install scipy numpy pandas scikit-learn matplotlib seaborn tqdm openpyxl hdbscan numba
5. pip install PyQt5 jupyter pyqtgraph==0.10 quantities neo
6. pip install https://github.com/tridesclous/tridesclous/archive/master.zip

1.2.3 Big GPU, big dataset OpenCL, and CO.

OpenCL is a language for coding parralel programs that can be run on GPU (graphical processor unit) and also on CPU multi core.

Some heavy part of the processing chain is coded both in pure python (scipy/numpy) and OpenCL. So, TDC can be run in any situations. But if the dataset is too big, you can stop mining cryto-money for while and can try to run TDC on a big-fat-gleaming GPU. You should gain some speedup if the number of channel is high.

Depending, the OS and the hardware it used to be difficult to settle correctly the OpenCL drivers (=opencl ICD). Now, it is more easy (except on OSX, it is becoming more difficult, grrrr.)

Here the solution on linux ubuntu/debian :

1. workon tdc
2. For intel GPU: sudo apt-get install beignet For nvidia GPU: sudo apt-get install nvidia-opencl-XXX
3. sudo apt-get instll opencl-headers ocl-icd-opencl-dev libclc-dev ocl-icd-libopencl1
4. pip install pyopencl

If you have a recent laptop you can also try the new neo-icd for intel GPU.

If you don't have GPU but a multi core CPU you can use POCL on linux:

    sudo apt-get install pocl

Here on windows a solution:

1. If you have nvidia or intel a a recent windows 10, then opencl driver are already installed
2. Download PyOpenCl here for windows : http://www.lfd.uci.edu/~gohlke/pythonlibs
3. Take the pyopencl file that match your python
4. cd C:/users/…../Downloads
5. pip install pyopencl-2019.1.2+cl12-cp37-cp37m-win_amd64.whl (for instance)

1.2.4 Ephyviewer (optional)

With neo (>=0.8) installed, if you want to view signals you can optionally install ephyviewer with:

```
pip install ephyviewer
```

1.2.5 Upgrade tridesclous

There are 3 sources for upgrading tridesclous package depending your need.

For **official** release at pypi:

```
pip install --upgrade tridesclous
```

For **up-to-date** or **new-featured** version get the master version on github:

```
pip install --upgrade https://github.com/tridesclous/tridesclous/archive/master.zip
```

For **work-in-progress** or **in-debug** version, take master version on my personal repo:

```
pip install --upgrade https://github.com/samuelgarcia/tridesclous/archive/master.zip
```

1.3 Launch

There are several ways to launch tridesclous:

- Inside a **jupyter notebook** good option to make example in your lab
- Inside python script, if you construct automatic pipeline
- With a Graphical User Interface (GUI): the least frightening for beginners

Please read carrefully, the how and the why for each methods.

1.3.1 Method 1: Launching tridesclous inside a jupyter notebook or a python script

This is the best method that authors recommend for users. People that never code with python or any language can be scared about this but this is easy!

Keep in mind that for reproducible science you need to keep track of what you are doing and this is the best way.

Also note that magic all-in-one command line and GUI keep you away from deep understanding of your spike sorting tool chain. Writting simple code block can help you a lot to realize and overcome difficulties.

So for this method:

1. Launch jupyter notebook (easy if you have anaconda)
2. Copy/paste:
   - this notebook
   - or this one
3. Read it carrefully.
4. Modify it and do your spike sorting.

Please also explore the [examples folder](https://github.com/tridesclous/tridesclous/tree/master/example) that contains some example on some dataset.

1.3.2 Method 2: Launching tridesclous GUI

Here's the method for lazy people (or people in a hurry).

For demagogical reasons, we wrote a GUI in Qt for launching tridesclous.

Do:

- open a terminal:

```
workon tdc  (or source activate tdc for windows)
tdc
```

- **In the GUI you must:**
    1. File>Initialize example_locust_dataset
    2. Select a channel group in **chan_grp**
    3. **Initialize catalogue**
    4. **Open catalogue Window**
    5. save catalogue when happy
    6. **run Peeler**
    7. **open PeelerWindow**

See *Step by step quickstart* for complete explanation.

1.4 Step by step quickstart

Here is a step by step hands on with the user interface. Mind you, writing a jupyter notebook is a better method to keep track of the spike sorting process. But the *click and play* approach is prefered for beginners.

1.4.1 Launch

In a console (terminal):

```
workon tdc (or activate tdc for windows)
tdc
```

You have a minimalist window with some icons on the left.

1.4.2 Step 1 - Initialize dataset

This step consists of initializing the datasets and configure everything: number of channels, probe geometry, number of channel groups.

To do that, normally, you have to click on "Initialize dataset", but for testing purposes you can simply use menu **"File > download dataset > striatum_rat"**. This will locally download datasets from https://github.com/tridesclous/tridesclous_datasets and configure everything for you.

After that step some information are displayed about the:

- DataIO: details about the dataset (nb channel, nb channel group, n segment, sample rate, durations...)
- CatalogueConstructor: empty for the moment

Here we have a dataset given by David Robbe and Mostafa Safai recorded with a tetrode in sriatum of a rat. The signal is sampled at 20kHz.

1.4.3 Step 2 - Initialize Catalogue

This step contains several sub steps to create the catalogue. This is automatic but needs some parameters.

Click on "Initialize" on parameters dialog popup. Parameters are organized in sections:

- duration
- preprocessor
- peak_detector
- noise_snippets
- extract_waveforms

For complete details on parameters see *Choose parameters* and *Important details*.

Then you have to choose a **feature method**. For tetrode **"global_pca"** with 5 component sounds good. You will be able to change this in catalogueWindow later on.

Then you have to choose a **cluster method**. Here let's choose **"gmm"** (gaussian mixture model) with 3 clusters.

Depending on the dataset and chosen method this can take a while. Here, it should take around 5 seconds for 300s of signal.

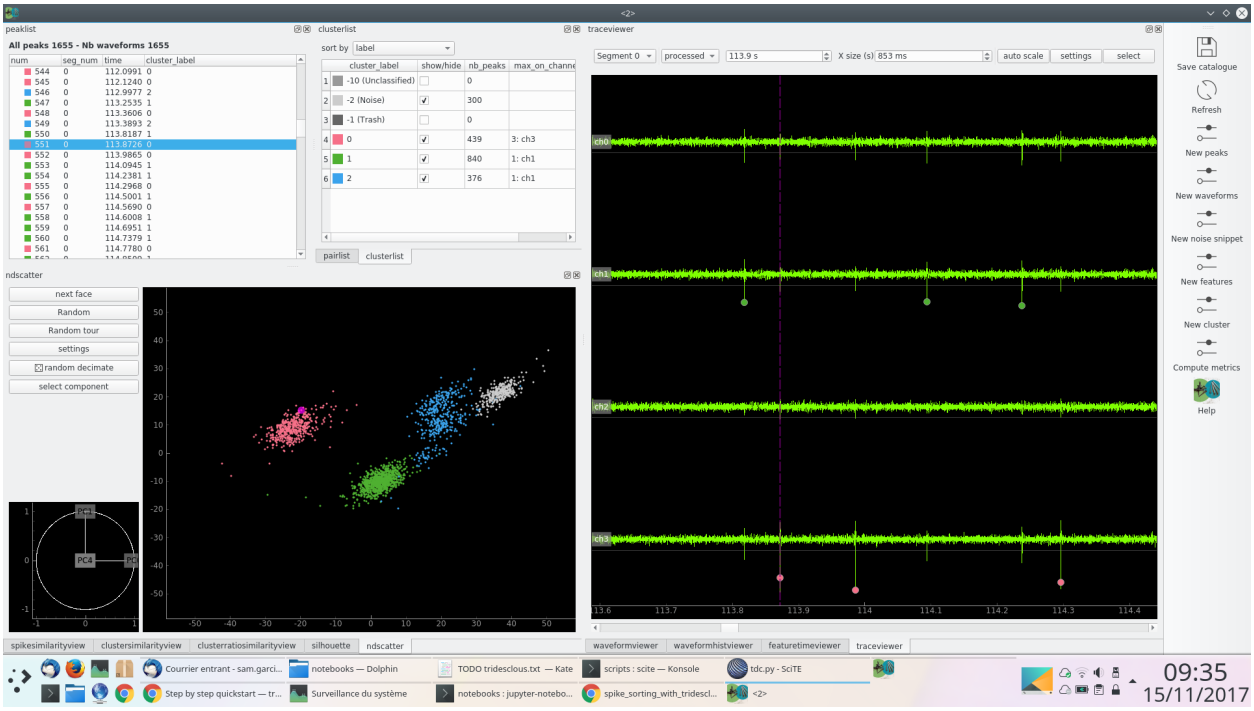
After that step CatalogueConstructor info is updated:

- nb of detected peaks
- waveforms shape
- features shape
- cluster labels

1.4.4 Step 3 - CatalogueWindow

Click on CatalogueWindow. A window with multi view will help with the manual correction of the auto-catalogue (step 2)

This window contain docks than can be arranged as you want. Some of them are organized in tabs, but you can change. with drag and drop. You can event close or move some view on another screen. Righ click on the left toolbar to make them appear again.



On the right toolbar you can manually re-run some sub steps of the previous chain: detect peak, extract waveforms, extract noise snippets, extract features, cluster.

Main views are:

- spike list
- cluster list
- trace view
- ndscatter
- waveform view

For more detail see *Catalogue window*

All views are linked, this means that when you click somewhere it will update the other views. For instance, if you select a spike, the trace view will zoom on that spike and the ndscatter will highlight the spike.

In the trace view you can zoom Y with the mouse wheel and zoom X with right click.

Make visible one by one each cluster [0, 1, 2]. Play with the noise (label -2) and see what happens in each view.

Click on **"random tour"** in ndscatter. It is a dynamic projection that includes many dimensions like in GGobi. It helps a lot to understand how many clusters we have.

Many views can be customized with a settings dialog. Sometimes, you have to double click on the view, sometimes on a button. For instance in **waveformhistviewer** you can choose the *colormap* and the *binsize* with a double click in the black area.

In **pairlist**, select each pair and see what happens on **waveformhistviewer**. Use the mouse wheel to zoom the colormap and right click to zoom X/Y.

Them click on **Compute metric**, this will enable some views: **spikesimilarity**, **clustersimilarity**, **silhouette**.

Go to **waveformviewer**, select "geometry" or "flatten".

Go to **waveformhistviewer**, your best friend same as **waveformviewer** in flatten mode but with historgram density in 2D.

Cluster list contains a context menu that proposes a lot of actions: merge, split, trash. Click on "re label cluster by rms".

Now you can see that cluster 0 and 1 are very well isolated but cluster 2 is very close from our chosen threshold. To simplify we will send it to "trash". This means that the "peeler" (template matching) will not try to get it.

Now do "make catalogue for peeler". We have 2 clusters in our catalogue.

Close the window.

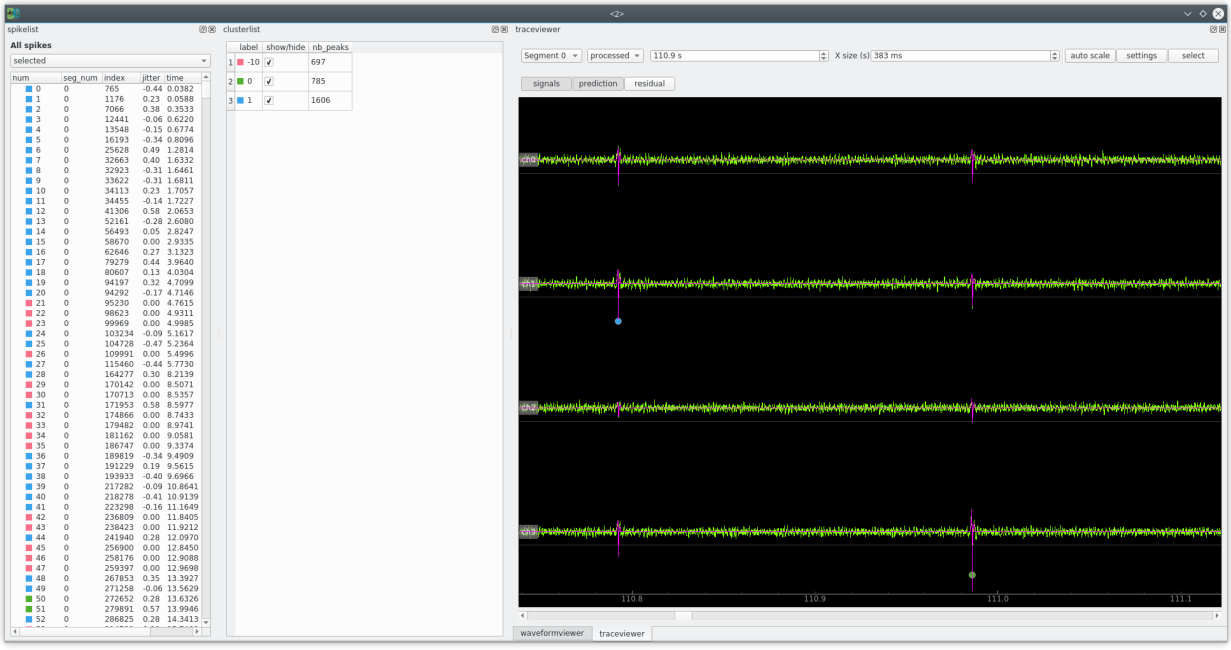
1.4.5 Step 4 - Run peeler

Click "run peeler" and keep parameters.

This should take about 10 seconds (for 500s of signal). The speedup 50x over real time is due to low number of channels and low number of clusters.

1.4.6 Step 5 - PeelerWindow

Click on "open PeelerWindow"



This windows is to check whether peeler has corectly done its job, in other words if the catalogue were OK.

You can click on the spike list and the trace auto zooms on the spike.

On the trace view you can click on "residual".

The most important things to understand here is:

- the green trace is the **preprocessed** signal (filter and normalized)
- the magenta trace is the **prediction** = zero + waveform interpolated in between samples.
- the yellow one is the **residual = preprocess - prediction**

If the catalogue is good and the peeler not buggy, the residual must always stay under the threshold (white line) for all channels.

You can see that some spike are not labelled (-10) this means that:

- we forgot a cluster in the catalogue
- we deliberatly removed this cluster because it is too close from threshold or noise.
- the interpolation between samples is wrong and the remaining noise due to sampling jitter is bigger than standard noise (too bad).

1.5 Catalogue window

CatalogueWindow is important, it helps users to manually validate and correct some mistakes made by the automatic steps (filtering, threshold, feature extraction, clustering, . . .)

All views are linked, this means that some actions interact with other views.

All views can be customised by settings (often with a double in the view).

Here some detail on each view.

Note: A lot of efforts have been put in making this UI as smooth as possible but for big datasets (>100 channels) the CatalogueWindow can be slow for some actions because it trigs a full refresh on other views and no computation can be done (re compute centroid for instance). So be patient and smart.

Note: For some manual actions on catalogue, CatalogueWindow can suddenly crash. While this is annoying, you should not lose any data. Just open again the same dataset and you should be in a previous situation.

In cases of crashes please send an issue on github https://github.com/tridesclous/tridesclous/issues It takes only minutes and helps a lot to make tridesclous more stable. Please copy/paste the error message in the console in the issue and describe very briefly the actions that triggered the crash.

1.5.1 Toolbar on the left

You can:

- **Make catalogue for Peeler**: this construct all waveform centroid + first and second derivative + interpolated waveforms. This is needed by the Peeler. This initial catalogue is valid for one channel group only. This is saved inside the working directory of the DataIO in the appropriate channel_group sub dir. Do this at the end when the clusters seem OK before launching the Peeler on the entire dataset.
- **refresh**: this reloads data from DataIO and refreshes all views. This is extremly useful when you externally in jupyter for instance change or test some methods that transform data. This avoids to restart the UI.
- **New peaks**: re detect peaks, you change the threshold for a new detection here.
- **New waveforms**: take some other (or all) waveforms with other parameters.
- **Clean waveforms**: detect bad (allien) waveform.
- **New noise snippet**: extract new noise snippet.
- **New features**: choose other method or parameters for features extraction.

- **New cluster**: choose other method or parameters for clustering.
- **Compute metrics**: the compute metric on clusters : similarity, ratio_similarity, silhouet, ...
- **Help**: a button that magically telestranport you in that this current page.
- **Savepoint**: duplicate the working dir, that you can manually go back in the past with a funny game of folder renaming.

1.5.2 Peak list

class tridesclous.gui.peaklists.**PeakList**(*controller=None*, *parent=None*)

Peak List show all detected peak for the catalogue construction.

Here pintentionally peaks are not spikes already (even most of them are spikes) because supperposition of spikes are done here in catalogue in Peeler.

Note:

- If there are to many peaks, not all of them will have a extracted waveform. This why some peak are not labeled (-10) and nb_peak != nb_wveforms sometimes.
- Peaks can belong to diffrents segment, a column indicate it. This is th full list of all peaks of all segment.
- A right click open a ontext menu: * move one or several selected spike to trash * create a new cluster with one or several spikes
- When you select one spike, this will auto zoom on **Trace View**, auto select the appriopriate segment and hilight the point on **ND Scatetr**. And vice versa.

1.5.3 Cluster list

class tridesclous.gui.peaklists.**ClusterPeakList**(*controller=None*, *parent=None*)

Cluster list is the central widget for actions for clusters : make them visible, merge, trash, sort, split, change color, ...

A context menu with right propose:

- **Reset colors**
- **Show all**
- **Hide all**
- **Re-label cluster by rms: this re order cluster so that 0 have the bigger rms** and N the lowest.
- **Feature projection with all**: this re compute feature projection (equivalent to left toolbar)
- **Feature projection with selection**: this re compute feature projection but take in account only selected usefull when you have doubt on small cluster and want a specifit PCA because variance is absord by big ones.
- **Move selection to trash**
- **Merge selection**: re label spikes in the same cluster.
- **Select on peak list**: a spike as selected for theses clusters.
- **Tag selection as same cell**: in case of burst some cell can have diffrent waveform shape leading to diferents cluster but with same ratio. If you have that do not merge clusters because the Peeler wll fail. Prefer tag 2 cluster as same cell.

- **Split selection**: try to split only selected cluster.

Double click on a row make invisible all others except this one.

Cluster can be visualy ordered by some criteria (rms, amplitude, nb peak, . . .) This is useless to explore cluster few peaks, or big amplitudes, . . .

Negative labels are reserved:

- -1 is thrash
- -2 is noise snippet
- -10 unclassified (because no waveform associated)
- -9 Alien

1.5.4 Trace viewer

class tridesclous.gui.traceviewer.**CatalogueTraceViewer**(*controller=None*, *signal_type='processed'*, *parent=None*)

Trace viewer allow to browser raw signal and preprocess signals.

Note that this viewer do not load the entire signals in memory but load chunk on demand from HD, that is why depending on the drive it can be quite slow. All zoom and scale factor for signals are computed on CPU and not on GPU (it is not vispy!!), so this is not the fastest veiwer but many tips help user to navigate very efficiently.

What you can do:

- On the bottom there is a slider over time
- On the left there is a slider over channels (if nb_channel>16)
- If several segments you can switch.
- You can select manually jump to any time
- You can zoom the X (time) axis with the spinbox ot by **right click** with mouse.
- **The mouse wheel make a glocal zoom on signal**
- You can "select" manually a spike with "select" button and this will be show in **ND Scatter**
- The threshold is a line for each channel. This is very important to why so peak are not detected.
- Setting button: * "auto_zoom_on_select" : auto zoom when select on ndscatter on peak list * "zoom_size" in s * disable plot threshold

Important:

- Ths "preprocessed" signal are normalized (robust Z-score) so that the noise variance is 1. So the apparent noise **must be** inbetween [-3, 3]

1.5.5 NDScatter

class tridesclous.gui.ndscatter.**NDScatter**(*controller=None*, *parent=None*)

ND Scatter a central beast for the catalogue window.

It try to mimic RGGobi viewer package.

Displaying clusters in high (more than 3!) dimensional space is difficult. To overcome this problem, some people use a combination of 2D plots of all pairs of dimensions. Others use various 3D representations. However in this

two cases the view angle is seriously restricted: it is the projection of a big hyper cube (ND) onto only one face (2D or 3D) of this hyper cube. In real life, two clusters are not necessary well separated in one of this hyper-cube's face but better in a complex hyper plane that brings into play a combination of all features/dimensions.

ND Scatter project randomly N-D space into 2-D space. Furthermore it do it dynamically. This is the **random tour**.

If you want to compare how 2 by 2 components are displayed in other tools you can use **next face**. This will combined (PC0, PC1), then (PC0, PC2), ...

For very high dimention, ND Scatter is not enough, for so ND scatter is guided in feature selection with the probe geometry. In short when you set visible some label with the **Cluster list** this automatically activated the dimenssions that must but visible (and rodomized) and hidden. This is based on **extremum_channel** property of clusters. This is the **auto_select_component** in settings. Note that you can define a radius in micometers arround the channel of the max so that all component in a neighborhood of thsi channel will be also displayed. this is based on the PRB file that **need to have the units in μm** (contrary to phy where units is not important for plotting)

Note:

- for too dense cluster you can limit the nb of dot dsiplayed. It helps a lot when density of clusters are differents. The is the role of **random decimate** button
- you can manually activate/deactivate componant just for to understand.
- mouse wheel will zoom both X et Y
- with right click you can draw a **lasso**. This will select peak inside the poly on the projected hyperplan. This is in fact the secret weappon of tridesclous. It help a lot for au zoom on trace for "strange" point ouside main clouds. It can help to understand if they are in fact superposition of several spikes or very few dense cluster.
- not all peak can be visible here only thosse wich have a waveform and so a feature. remember that not all waveforms are taken for clustering but only a subset. Getting everything is the role of the Peeler.
- This viewer take some CPU ressource since in **random tour** a numpy.dot is done for each step.

1.5.6 Pair list

class tridesclous.gui.pairlist.**PairList**(*controller=None*, *parent=None*)

Pair list is an intuituitive list of pair of cluster : when you click on a pair this make visible only this 2 cluster on all others views.

This help to validate in a fast way clusters.

For convinience this paris can be filtered:

- **all pairs** : all combinaison
- **high similarity**: select only pairs that have similarity over a threshold (in settings)
- **similarity amplitude ratio**! select only pairs that have "similarity ratio" over a threshold (in settings)

And they can be sorted by:

- label
- similarity
- ratio_similarity

WIth a right click you can:

- **merge** a pair

- **tag same cell**, this keep 2 cluster but there tag as same cell.

1.5.7 Waveform viewer

class tridesclous.gui.waveformviewer.**WaveformViewer**(*controller=None*, *parent=None*)

Waveform viewer is undoubtedly the view to inspect waveforms.

Note that in some aspect **Waveform hist viewer** can be a better firend.

All centroid (median or mean) of visible cluster are plotted here.

2 main modes:

- **geometry** waveforms are organized with 2d geometry given by PRB file.
- **flatten** each chunk of each channel is put side by side in channel order than it can be ploted in 1d. The bottom view is th mad. On good cluster the mad must as close as possible from the value 1 because 1 is the normalized noise.

The **geometry** mode is more intuitive and help users about spatial information. But the **flatten** mode is really important because is give information about the variance (mad or std) for each point and about peak alignement.

The centoid is dfine by median+mad but you can also check with mean+std. For healthy cluster it should more or less the same.

Important for zooming:

- **geometry** : zoomXY geometry = right click, move = left click and mouse wheel = zoom waveforms
- **flatten**: zoomXY = right click and move = left click

Settings:

- **plot_selected_spike**: superimposed one slected peak on centroid
- **show_only_selected_cluster**: this auto hide all cluster except the one of selected spike
- **plot_limit_for_flatten**: for flatten mode this plot line for delimiting channels. Plotting is important but it slow down the zoom.
- **metrics**: choose median+mad or mean+std.
- *show_channel_num**: what could it be ?
- **flip_bottom_up**: in geometry this flip bottom up the channel geometry.
- **display_threshold**: what could it be ?

1.5.8 Waveform hist viewer

class tridesclous.gui.waveformhistviewer.**WaveformHistViewer**(*controller=None*, *parent=None*)

Waveform histogram viewer is also a important thing.

It is equivalent to **Waveform veiwer** in **flatten** mode but with a 2d histogram that show the density (probability) of a cluster. So waveforms are flatten from (nb_peak, nb_sample, nb_channel) to (nb_peak, nb_channel*nb_sample) and are binarized on a 2d histogram. Then this is plotted as a map. The color code the density.

This is the best friend to see if two cluster are well discrimitated somewhere or if one cluster must be split.

Important:

- use right click for X/Y zoom
- use left clik to move
- use **mouse wheel** for color zoom.Really important to play with this to discover low density
- intentionnaly not all cluster are displayed other we see nothing. The best is to plot 2 by 2. Furthermore it faster to plot with few cluster.
- don't forget to display the **noise snippet** to validate that the mad is 1 for all channel.

Settings:

- **colormap** hot is good because loaw density are black like background.
- **data** choose waveforms or features
- **bin_min** y limts of histogram
- **bin_max** y limts of histogram
- **bin_size**
- **display_threshold**
- **max_label** maximum number of labels displayed simulteneously (2 by default but you can set more)

1.5.9 Feature on time viewer

class tridesclous.gui.waveformhistviewer.**WaveformHistViewer**(*controller=None*, *parent=None*)

Waveform histogram viewer is also a important thing.

It is equivalent to **Waveform veiwer** in **flatten** mode but with a 2d histogram that show the density (probability) of a cluster. So waveforms are flatten from (nb_peak, nb_sample, nb_channel) to (nb_peak, nb_channel*nb_sample) and are binarized on a 2d histogram. Then this is plotted as a map. The color code the density.

This is the best friend to see if two cluster are well discrimitated somewhere or if one cluster must be split.

Important:

- use right click for X/Y zoom
- use left clik to move
- use **mouse wheel** for color zoom.Really important to play with this to discover low density
- intentionnaly not all cluster are displayed other we see nothing. The best is to plot 2 by 2. Furthermore it faster to plot with few cluster.
- don't forget to display the **noise snippet** to validate that the mad is 1 for all channel.

Settings:

- **colormap** hot is good because loaw density are black like background.
- **data** choose waveforms or features
- **bin_min** y limts of histogram
- **bin_max** y limts of histogram
- **bin_size**
- **display_threshold**

- **max_label** maximum number of labels displayed simulteneously (2 by default but you can set more)

1.5.10 Spike similarity

class tridesclous.gui.similarity.**SpikeSimilarityView**(*controller=None*, *parent=None*)

Spike similarity view dispplay the spike-to-spike similarity. Only visible cluster are shown.

If nothing appear means : metrics are not computed yet or the size of the similarity is too big (over **max__size**).

1.5.11 Cluster similarity

class tridesclous.gui.similarity.**ClusterSimilarityView**(*controller=None*, *parent=None*)

Cluster similarity view dispplay the clsuter-to-cluster similarity.

If nothing appear means : metrics are not computed yet.

1.5.12 Cluster ratio similarity

class tridesclous.gui.similarity.**ClusterRatioSimilarityView**(*controller=None*, *parent=None*)

Cluster similarity ratio view dispplay the clsuter-to-cluster ratio similarity.

If nothing appear means : metrics are not computed yet.

1.5.13 Silhouette

class tridesclous.gui.silhouette.**Silhouette**(*controller=None*, *parent=None*)

Silhouette display the silhouette score.

Implemented with sklearn. Must compute metrics first.

See:

- Silhouette wikipedia
- Silhouette sklearn

1.6 Important details

A list a miscellaneous details, some of them are related to data handling and other to details of methods at different steps of the chain.

1.6.1 Multi segments

Many recording are split into several “Segments”. Segment here refer to the neo meaning.

This is because the protocol has been recorded in several pieces.

There are several cases:

- each file corresponds to one segment

- on file has internally several segments. For instance Blackrock. Because you can make a pause during recording the same file.
- several files that themselves contain several segment. Outch!!

Tridesclous use neo.rawio for reading signals. In neo, reading signals from segments is natural so this notion of segment is the same in tridesclous.

This means that you can feed tridesclous with a list a files (with same channels maps of course) and tridesclous will generate them as a list of segment naturally.

Note that the raw file should store the data in this order: *t0c0 t0c1 t0c2 t0c3... t1c0 t1c1 t1c2 t1c3* where *t{i}c{j}* is sample no. *i* on channel no. *j*. Thus if the data is loaded in a 2D numpy array *x* where each row is the time series data from one channel, you can save it in a raw file with this code: *x.T.tofile(filename)*.

1.6.2 Channel groups, geometry and PRB file

There several situation when you want use channel_group:

- The probe have several shank and you want do do the spike sorting indenpendently on each shank
- Only a subset a channels are used.
- There are dead channels
- The dataset is several tetrode (N-trode) and so several channel group.

Tridesclous manage theses **channel_group** globally, this means that with same DataIO.

But you will need to construct a catalogue for each channel_group and run the Peeler for each **channel_group**. Of course, you can compute each group in parallel if you have enough resources to do it (CPU, RAM, GPU).

If you use probe you naturally know the "geometry" of the probe. For handmade tetrode, you don't know the geometry.

PRB files are simple files easy to edit with a text editor and where introduced in klusta. There are also used in spyking-circus and certainly in other toolbox.

This files describe both **"channel_group"** and there **"geomtery"**. Klusta also needs **"graph"** but it is ignored in tridesclous.

A typical PRB file look like this, here 8 channels (2 tetrodes):

```
channel_groups = {
    0: {
        'channels': [0, 1, 2, 3],
        'geometry': {
            0 : [-50, 0],
            1 : [0, 50],
            2 : [50, 0],
            3 : [0, -50],
    },
    1: {
        'channels': [4, 5, 6, 7],
        'geometry': {
            4 : [-50, 0],
            5 : [0, 50],
            6 : [50, 0],
            7 : [0, -50],
    },
}
```

If some of the channels were dead or picked up excessive noise, they can be skipped:

```
channel_groups = {
    0: {
     #  'channels': [0, 1, 2, 3],    # if all channels were to be used in spike-sorting
        'channels': [1, 2, 3],    # do not use data from channel at index 0
        'graph':  [],  # Used by klusta but we don't care, SpikeInterface fills this␣
↪up automatically
        'geometry':  {
           # 0: [0, 20],
            1: [0, -20],
            2: [20, 0],
            3: [-20, 0],
        }
     },
}
```

Units of geometry must be micrometers

Collections of PRB files exists here:

- https://github.com/kwikteam/probes/blob/master/neuronexus
- https://github.com/spyking-circus/spyking-circus/tree/master/probes

tridesclous can automatically download them with the DataIO:

```
dataio.download_probe('kampff_128', origin='spyking-circus')
dataio.download_probe('1x32_buzsaki', origin='kwikteam')
```

1.6.3 Pre-processing

The pre-processing chain is done as follows:

1. **Filter**: high pass (remove low flucatution) + low pass (remove very high freqs) + kernel smooth
2. **common_ref_removal**: this substracts sample by sample the median across channels When there is a strong noise that appears on all channels (sometimes due to reference) you can substract it. This is as if all channels would re referenced numerically to there medians.
3. **Normalisisation**: this is more or less a robust z-score channel by channel

Important:

- For spike sorting it is important to compute the filter with forward-backward method. This is filtfilt in matlab or in scipy. The method prevents phase problems. If we apply only a forward filter the peak would be reduced and harder to detect. There is a counter part of this filtfilt: if we want to avoid reading the whole dataset forward and then the whole dataset backward, we need a margin at the edge of each signal chunk to avoid bad side effect due to filter. This parameters is controlled by **lostfront_chunksize**. This leads to more computation and potentially, to small errors. Fortunately, when filtering in high frequency (case in spike sorting) 128 sample at 20kHz is sufficent to not make mistakes.
- The smooth is in fact also a filter with a short kernel applied also forward-backward.
- Filters are applied with a biquad method. High pass + low pass + smooth is computed within the same filter with several sections (cascade).
- The normalisation is a robust z-score. This means that for each channel **sig_normed = (sig - median) / mad.** Mad is median absolute deviation So after pre processing chain, the units of each is **signal to noise ratio**. So as the Gaussian low:
  - magnitude 1 = 1 mad = 1 robust sd = 68% of the noise

  - magnitude 2 = 2 mad = 2 robust sd = 95% of the noise
  - magnitude 3 = 3 mad = 3 robust sd = 99.7% of the noise

  This is crucial to have this in minds for settings the good threshold.

- Many software also include a whitening stage. Basically this consists of applying to signals the factorized and inversed covariance matrix. This is intentionally not done in tridesclous for theses reasons:
  - Contrary to what some user think: this does not denoise signals.
  - This must be computed on chunks where there are no spikes. This is hard to do it cleanly.
  - Matrix inversion can lead to numerical error and so some pathological tricks are often added.

1.6.4 Peak detection and threshold

If one understands that the preprocessed signals units are MAD, the threshold become very intuitive.

The best is to have spikes that have the big signal to noise ratio so that all spikes from a cluster do not overlap with noise. This is important because if the threshold is too close to the noise some of the spikes will not be detected and so the cluster will be partial and so the centroids of the cluster will be wrong. Bad practice!!

There is 2 algorithms to detect spike:

- **global** : every local extrema above the threshold on at least one channel is considered as peak.
- **geometrical** same local extrema detection but only on local part given the geometry of the probe.

With high frequency noise the true peak can be noisy and become a double local extremum. When you want to avoid that to not having twice the same peak extracted with some sample delayed. This is the role of the **peak_span_ms** parameters: when 2 local extrema are in the same span, only the best is kept.

1.6.5 Waveforms extraction

For contruction of catalogue, we need to extract **waveforms**. It is a snipet around each peak. The feature and cluster will be based on this array.

- Not all waveforms are extracted in tridesclous only a subset of them. If the **duration** choosen for the cataloque is too long them we could have too much peaks. Gathering them all of them can be too long and lead too memory overflow. So a random subset is choosen. If clusters are clear and dense enough, it is not a problem because it will lead to same centroid if we have took waveforms from all peak. For low firing rate neurons having low dense cluster can be a problem and the user need to keep eye open on this. In the catalogue peaks that have label -11 means: they don't have been chossen, so they have no waveforms.
- catalogueconstructor.some_waveforms shape is (nb_peak, nb_sample, nb_channel). On the sample axis waveforms are aligned of the extrema.
- sample width of each waveforms is controlled by **n_left** and **n_right** parameters. They must choosen carrefully. If it is too long the total dimenssion will be high and there will be too much noise for clustering. If it is too short, the Peeler (template matching) will fail when substracting leading to noisy residual due to borders. A good rule is:
  - the median of each cluster need to be back to 0 on each side
  - AND the mad of a cluster need to be back to 1 (noise) on each side.

1.6.6 Noise snippets extraction

A good practice is to extract also some noise snipet that do not overlap peaks. This will be usefull to compare waveforms of peaks and snippet of noise statistically.

If everything OK, this noise must median=0 and mad=1 because the preprocessed signal is normalized this way. Checking this is important.

Noise snippet can be also projected in the same sub space than waveforms. With this, we can compare distance noise to waveforms in the feature space.

1.6.7 Feature extraction

On that part we enter in the quarrel zone. It is a subject were people having introduce new methods in context of spike sorting stand up to defend religously new ideas.

The problem is pretty simple: the dimenssion of waveforms (nb_sampleXnb_channel) is too big for clustering algorithm, so we need to reduce this "space" to a smaller "space". And course we want to reduce this dimentionality while keeping difference between cluster. The step is so called **feature extraction**.

The most obvious methods is PCA but also SVD, ICA, or wavelet tricks have been proposed.

> For instance., PCA will keep the sample were the variance is the biggest in full space.

Keep in mind, that choosing between PCA or SVD or ICA do not mater so much.

The real problem in fact is how can we do this when we have lot of channels ? Many tools apply a dimenssion reduction by channel (often PCA) and concatenate them all. This a well establish mistake because each channel will have the same weight in the feature space even if it contain noise. A better approach proposed by some tools is to take in a neighborhood some channels, concatenate there waveform and to apply a PCA on it. Doing this will automatically elimate channel with few variance.

Note that when a spike have a clear spatial signature, (for example in dense array a spike can be seen on 10 channels), taking only the amplitude by channel of spike at the extrema is very naive but lead to good result. This is called **meak_max** in tridesclous. This is the fastest method and do not imply alegebric formula.

To not upset anybody we implement several methods, so the user can choose and compare:

- **global_pca** concatenate all waveform and apply a PCA on it. The best for tetrode (and low channel count)!!
- **peak_max** get only the peak by channel. Very fast for dense array and not so bad.
- **pca_by_channel** the most widespread method. Apply a PCA by channel and concatenate them after.
- **neighborhood_pca** the most sofisticated. For each channel we concatenate the waveforms of the neighborhood and apply a PCA on it.

1.6.8 Clustering

Likewise feature extraction, for cluster, imagination and creativity have been large to introduce in the context of spike sorting some well establish or new fashionable methods of clustering. While the field of machine learning is exploding todays the number of sorting algotrithm is naturally become bigger.

Unfortunatly there is a central dilemma : the end user want that the algorithm tell him how many cluster we have but robust clustering algorithm also want that the end user tell him how much cluster there are. Outch!

Of course for very big dataset with tens (or hundreds!) of neurons nobody wants want to try all **n_cluster** possibilities for discovering the best. There is a strong need of automagic cluster number finding. This is possible with many methods, for instance density based approaches. But keep in mind that there are always some parameters somewhere

(often hidden to user) that can dramatically change the cluster number. So don't be credulous when some toolbox propose full automatic spike sorting, some (hiden) parameters can lead to over clusterised or over merged results.

The approach in tridesclous is to let the user choose the method but validate manually the choice with the CatalogueWindow. The user eye and intuition is better a weapon than a pre parametrised algotihm.

As we are lazy, we did not implement any of theses methods but use them from sklean package. However, one home made method is implemented here: **sawchaincut**, be curious and test it. **sawchaincut** is more or less what all we want : a full automated clustering, this works rather well on dense multi-electrode array when there is a high spatial redundancy (a spike is seen by several channels) but need some manual curation (like every automated clustering algorithm).

The actual method list is:

- **kmeans** super classic, super fast but we need to decide **n_cluster**
- **gmm** gaussian mixture model another classic, we need to decide **n_cluster**
- **agglomerative** for trying, we need to decide **n_cluster**
- **dbscan** density based algorithm n_cluster should be automatic. But **eps** parameters play a role in the results.
- **hdbscan** identique with withous **eps**
- **isosplit** : develop by Jeremy Maglang for moutainsort
- **sawchaincut** this is a home made, full automatic, not so good, not so bad, dirty, secret algorithm. It is density based. If you don't known which one to choose and you are in a hurry, take this one. Most beautiful and well isolated clusters should be captured by this one.
- **pruningshears** this is also a home made stuff. Internal it use hdbscan but localy.

1.6.9 In between sample interpolation

A non intuitive but strong source of noise is the sampling jitter. Signals are digitaly sampled between 10kHz and 30kHz so the inter sample interval is between 33 and 100 μs. A spike been a very short event, the true of the signal peak before digitalisation have few chance to be at the same time than the sample. It is in fact in between 2 samples with a random uniform low.

You can observe it easly: you compute the centroid with the median of a cluster and you can see a big overshoot of the variance (done with the mad) around the peak. This is due to high first derivative et poor alignement.

At the Peeler level, we need to compensate this jitter before substract the centroid from the signal otherwise the residual will show strong fake peak around the true peak (like the mad). This is due to jitter. The remain noise amplitude if no jitter compensation can be in order of magnitude of 2 or 3 mad. The phenonem is really clear with spike with big amplitude at 10kHz. At upper sample rate with small peak this is less important.

The method used for jitter estimation and cancellation is describe here. In short this method based on taylor expanssion is fast and acurate enough.

1.7 Choose parameters

Some details on parameters for each step.

1.7.1 Preprocessor

- highpass_freq (float): frequency of high pass filter typically 250~500Hz. This remove LFP component in the signal Theorically a low value is better (250Hz) but if the signal contain oscillation at high freqyencies (during sleep for insctance) theye must removed so 400Hz should be OK.
- **lowpass_freq (float): low pass frequency (typically 3000~10000Hz) This remove noise in high freuqnecy. This help** to smooth the spike for peak alignement. This must not exceed niquist frequency (sample_rate/2)
- **smooth_size (int): other possibility to smooth signal. This apply a kernel (more or less triangle) the *smooth_size** width in sample. This is like a lowpass filter. If you don't known put 0.
- **common_ref_removal (bool): this substracts sample by sample the median across channels** When there is a strong noise that appears on all channels (sometimes due to reference) you can substract it. This is as if all channels would re referenced numerically to there medians.
- **chunksize (int): the whole processing chain is applied chunk by chunk, this is the chunk size in sample. Typically 1024.** The smaller size lead to less memory but more CPU comsuption in Peeler. For online, this will be more or less the latency.
- **lostfront_chunksize (int): size in sample of the margin at the front edge for each chunk to avoid border effect in backward** In you don't known put None then lostfront_chunksize will be int(sample_rate/highpass_freq)*3 which is quite robust (<5% error) compared to a true offline filtfilt.
- engine (str): 'numpy' or 'opencl'. There is a double implementation for signal preprocessor : With numpy/scipy flavor (and so CPU) or opencl with home made CL kernel (and so use GPU computing). If you have big fat GPU and are able to install "opencl driver" (ICD) for your platform the opencl flavor should speedup the peeler because pre processing signal take a quite important amoung of time.

1.7.2 Peak detector

- peakdetector_engine (str): 'numpy' or 'opencl'. See signal_preprocessor_engine. Here the speedup is small.
- peak_sign (str) : sign of the peak ('+' or '-'). The double detection ('+-') is intentionaly NOT implemented is tridesclous because it lead to many mistake for users in multi electrode arrays where the same cluster is seen both on negative peak and positive rebounce.
- relative_threshold (str): the threshold without sign with MAD units (robust standard deviation). See *Important details*.
- peak_span_ms (float) : this avoid double detection of the same peak in a short span. The units is millisecond.

1.7.3 Waveform extraction

- wf_left_ms (float); size in ms of the left sweep from the peak index. This number is negative.
- wf_right_ms (float): size in ms of the right sweep from the peak index. This number is positive.
- **mode (str): 'rand' or 'all' With 'all' all detected peaks are extracted. With 'all' only an randomized subset is taken.** Note that if you use tridesclous with the script/notebook method you can also choose by yourself which peak are choosen for waveform extraction. This can be usefull to avoid electrical/optical stimlation periods or force peak around stimulus periods.
- nb_max (int): for 'rand' mode this is the number of peak extracted. This number must be carrefully choosen. This highly depend on : the duration on which the catalogue constructor is done + the number of channel (and so the number of cells) + the density (firing rate) fo each cluster. Since this can't be known in advance, the user must explore cluster and extract again while changing this number given dense enough clusters. This have a strong imptact of the CPU and RAM. So do not choose to big number.

1.7.4 Waveform clean

- alien_value_threshold (float): units=one mad. above this threshold the waveforms is tag as "Alien" and not use for features and clustering

1.7.5 Noise snippet extraction

- nb_snippet (int): the number of noise snippet taken in the signal in between peaks.

1.7.6 Features extraction

Several methods possible. See *Important details*.

- **global_pca**:
  - n_components (int): number of components of the pca for all the channel.
- **peak_max** no parameters
- **pca_by_channel**:
  - n_components_by_channel (int): number of component for each channel.
- **neighborhood_pca**:
  - n_components_by_neighborhood (int): number of component by channel and its neighborhood
  - radius_um (float): radius around the channel in mircometers.

1.7.7 Cluster

Several methods possible. See *Important details*.

- **kmeans** : kmeans implemented in sklearn
  - n_clusters (int): number of cluster
- **onecluster** no clustering. All label set to 0.
- **gmm** gaussian mixture model implemented in sklearn
  - n_clusters (int): number of cluster
  - covariance_type (str): 'full', 'tied', 'diag', 'spherical'
  - n_init (int) The number of initializations to perform.
- **agglomerative** AgglomerativeClustering implemented in sklearn
  - n_clusters: number of cluster
- **dbscan** DBSCAN implemented in sklearn
  - eps (float): The maximum distance between two samples for them to be considered as in the same neighborhood.
- **hdbscan** HDBSCAN density base clustering without the problem of the **eps**
- **isosplit** ISOSPLIT5 develop for moutainsort (another sorter)
- **optics** OPTICS implemented in sklearn
  - min_samples (int): The number of samples in a neighborhood for a point to be considered as a core point.

- **sawchaincut** Home made automatic clustering, usefull for dense arrays. Autodetect well isolated cluster and put to trash ambiguous things.
- **pruningshears** Another home made automatic clustering. Internaly use hdbscan. Have better performance than **sawcahincut** but it is slower.

1.8 Real time

The Peeler have been design to do the processing chunk by chunk in mind. So the offline Peeler is also adapted to OnlinePeeler to be used in real time. OnlinePeeler is a pyacq Node.

pyacq is a system for distributed data acquisition and stream processing. It support some device use in electrophysiology (Blackrock, Multichannel system, Measurement computing, National Instrument, ...). Pyacq offer the possibility to dtsribute the computing on several machines. So it is particulary usefull in online spike sorting context when high channel count. The user will be able to distribute on several machines: the acquisition itself, the OnelinePeeler and some display.

pyacq and Tridesclous do not offer a strict real real time engine but an online engine which latency whihch can be controlled by the chunksize.

1.9 API for scripting

Scripting with tridesclous can avoid unnecessary click and play with the GUI in parameter dialogs if you need to process many files. Almost everything can be done 3 classes:

- `DataIO` for configuration of the dataset, format selection, PRB file assignement, ...
- `CatalogueConstructor` run all steps to construct the catalogue : signal processing, fetaure, clustering, ...
- `Peeler` run the template matching engine

Of course everything done by script can still be check and modify with the GUI (MainWindow, CatalogueWindow and PeelerWindow).

The best is to read examples in the git repo

1.9.1 Classes API

class tridesclous.dataio.**DataIO**(*dirname='test'*)

Class to acces the dataset (raw data, processed, catalogue, spikes) in read/write mode.

All operations on the dataset are done througth that class.

The dataio :

- work in a path. Almost everything is persistent.
- needed by CatalogueConstructor and Peeler
- have a datasource member that access raw data
- store/load processed signals
- store/load spikes
- store/load the catalogue
- deal with sevral channel groups given a PRB file

- deal with several segment of recording (aka several files for raw data)
- export the results (labeled spikes) to differents format

The underlying data storage is a simple tree on the file system. Everything is organised as simple as possible in sub folder (by channel group then segment index). In each folder:

- arrays.json describe the list of numpy array (name, dtype, shape)
- XXX.raw are the raw numpy arrays and load with a simple memmap.
- some array are struct arrays (aka array of struct)

The datasource system is based on neo.rawio so all format in neo.rawio are available in tridesclous. neo.rawio is able to read chunk of signals indexed on time axis and channel axis.

The raw dataset do not need to be inside the working directory but can be somewhere outside. The info.json describe the link to the *datasource* (raw data)

Many raw dataset are saved by the device with an underlying int16. DataIO save the processed signals as float32 by default. So if you have a 10Go raw dataset tridesclous will need at least 20 Go more for storage of the processed signals.

Usage:

```
# initialize a directory
dataio = DataIO(dirname='/path/to/a/working/dir')

# set a data source
filenames = ['file1.raw', 'file2.raw']
dataio.dataio.set_data_source(type='RawData', filenames=filenames,
                              sample_rate=10000, total_channel=16, dtype='int16')

# set a PRB file
dataio.set_probe_file('/path/to/a/file.prb')
# or dowload it
dataio.download_probe('kampff_128', origin='spyking-circus')

# check lenght and channel groups
print(dataio)
```

already_processed(*seg_num=0*, *chan_grp=0*, *length=None*)
Check if the segment is entirely processedis already computed until length

append_spikes(*seg_num=0*, *chan_grp=0*, *spikes=None*)
Append spikes.

channel_group_label(*chan_grp=0*)
Label of channel for a group.

download_probe(*probe_name*, *origin='kwikteam'*)
Download a prb file from github into the working dir.

The spiking-circus and kwikteam propose a list prb file. See:

- https://github.com/kwikteam/probes
- https://github.com/spyking-circus/spyking-circus/tree/master/probes

probe_name: str the name of file in github

origin: 'kwikteam' or 'spyking-circus' github project

export_spikes(*export_path=None*, *split_by_cluster=False*, *use_cell_label=True*, *formats=None*)
Export spikes to other format (csv, matlab, excel, . . .)

export_path: str or None export path. If None (default then inside working dir)

split_by_cluster: bool (default False) Each cluster is split to a diffrent file or not.

use_cell_label: bool (deafult True) if true cell_label is used if false cluster_label is used

formats: 'csv' or 'mat' or 'xlsx' The output format.

flush_processed_signals(*seg_num=0*, *chan_grp=0*, *processed_length=-1*)
Flush the underlying memmap for processed signals.

flush_spikes(*seg_num=0*, *chan_grp=0*)
Flush underlying memmap for spikes.

get_geometry(*chan_grp=0*)
Get the geometry for a given channel group in a numpy array way.

get_peak_values(*seg_num=0*, *chan_grp=0*, *sample_indexes=None*, *channel_indexes=None*)
Extract peak values

get_processed_length(*seg_num=0*, *chan_grp=0*)
Get the length in sample how already processed part of the segment.

get_segment_length(*seg_num*)
Segment length (in sample) for a given segment index

get_segment_shape(*seg_num*, *chan_grp=0*)
Segment shape for a given segment index and channel group.

get_signals_chunk(*seg_num=0*, *chan_grp=0*, *i_start=None*, *i_stop=None*, *signal_type='initial'*)
Get a chunk of signal for for a given segment index and channel group.

The signal can be the 'initial' (aka raw signal), the none filetered signals or the 'processed' signal.

seg_num: int segment index

chan_grp: int channel group key

i_start: int or None start index (included)

i_stop: int or None stop index (not included)

signal_type: str 'initial' or 'processed'

get_some_waveforms(*seg_num=0*, *chan_grp=0*, *sample_indexes=None*, *n_left=None*, *n_right=None*, *waveforms=None*, *channel_adjacency=None*, *channel_indexes=None*, *n_jobs=0*)
Exctract some waveforms given sample_indexes

if channel_adjacency not None and channel_indexes not None then it do sparse extaction

get_spikes(*seg_num=0*, *chan_grp=0*, *i_start=None*, *i_stop=None*)
Read spikes

iter_over_chunk(*seg_num=0*, *chan_grp=0*, *i_stop=None*, *chunksize=1024*, *pad_width=0*, *with_last_chunk=False*, ***kargs*)
Create an iterable on signals. ('initial' or 'processed')

for ind, sig_chunk in data.iter_over_chunk(seg_num=0, chan_grp=0, chunksize=1024, signal_type='processed'):
do_something_on_chunk(sig_chunk)

load_catalogue(*name='initial'*, *chan_grp=0*)
Load the catalogue dict.

nb_channel(*chan_grp=0*)
Number of channel for a channel group.

reset_processed_signals(*seg_num=0*, *chan_grp=0*, *dtype='float32'*)
Reset processed signals.

reset_spikes(*seg_num=0*, *chan_grp=0*, *dtype=None*)
Reset spikes.

save_catalogue(*catalogue*, *name='initial'*)
Save the catalogue made by *CatalogueConstructor* and needed by *Peeler* inside the working dir.

Note that you can construct several catalogue for the same dataset to compare then just change the name. Different folder name so.

set_channel_groups(*channel_groups*, *probe_filename='channels.prb'*)
Set manually the channel groups dictionary.

set_data_source(*type='RawData'*, ***kargs*)
Set the datasource. Must be done only once otherwise raise error.

type: str ('RawData', 'Blackrock', 'Neuralynx', ...) The name of the neo.rawio class used to open the dataset.

kargs: depends on the class used. They are transimted to neo class. So see neo doc for kargs

set_probe_file(*src_probe_filename*)
Set the probe file. The probe file is copied inside the working dir.

set_signals_chunk(*sigs_chunk*, *seg_num=0*, *chan_grp=0*, *i_start=None*, *i_stop=None*, *signal_type='processed'*)
Set a signal chunk (only for 'processed')

class tridesclous.catalogueconstructor.**CatalogueConstructor**(*dataio*, *chan_grp=None*, *name='catalogue_constructor'*)
The goal of CatalogueConstructor is to construct a catalogue of template (centroids) for the Peeler.

For so the CatalogueConstructor will:

- preprocess a duration of data
- detect peaks
- extract some waveform
- compute some feature from these waveforms
- find cluster
- compute some metrics ontheses clusters
- enable some manual trash/merge/split

At the end we can **make_catalogue_for_peeler**, this construct everything usefull for the Peeler : centroids (median for each cluster: the centroids, first and secnd derivative, median and mad of noise, ...

You need to have one CatalogueConstructor for each channel_group.

Since all operation are more or less slow each internal arrays is by default persistent on the disk (memory_mode='memmap'). With this when you instanciate a CatalogueConstructor you load all existing arrays already computed. For that tridesclous mainly use the numpy structured arrays (array of struct) with memmap. So, hacking internal arrays of the CatalogueConstructor should be easy outside python and tridesclous.

You can explore/save/reload internal state by borwsing this directory: *path_of_dataset/channel_group_XXX/catalogue_constructor*

Usage:

```
from tridesclous  import Dataio, CatalogueConstructor
dirname = '/path/to/dataset'
dataio = DataIO(dirname=dirname)
cc = CatalogueConstructor(dataio, chan_grp=0)

# preprocessing
cc.set_preprocessor_params(chunksize=1024,
    highpass_freq=300.,
    lowpass_freq=5000.,
    common_ref_removal=True,
    lostfront_chunksize=64,
    peak_sign='-',
    relative_threshold=5.5,
    peak_span_ms=0.3,
    )
cc.estimate_signals_noise(seg_num=0, duration=10.)
cc.run_signalprocessor(duration=60.)

# waveform/feature/cluster
cc.extract_some_waveforms(n_left=-25, n_right=40, mode='rand')
cc.clean_waveforms(alien_value_threshold=55.)
cc.project(method='global_pca', n_components=7)
cc.find_clusters(method='kmeans', n_clusters=7)

# manual stuff
cc.trash_small_cluster(n=5)
cc.order_clusters()

# do this before peeler
cc.make_catalogue_for_peeler()
```

For more example see examples.

Persitent atributes:

CatalogueConstructor have a mechanism to store/load some attributes in the dirname of dataio. This is usefull to continue previous woprk on a catalogue.

The attributes are almost all numpy arrays and stored using the numpy.memmap mechanism. So prefer local fast fast storage like ssd.

Here the list of theses attributes with shape and dtype. **N** is the total number of peak detected. **M** is the number of selected peak for waveform/feature/cluser. **C** is the number of clusters

- all_peaks (N,) dtype = [('index', 'int64'), ('cluster_label', 'int64'), ('channel', 'int64'), ('segment', 'int64')]
- signals_medians (nb_sample, nb_channel,) float32
- signals_mads (nb_sample, nb_channel,) float32
- clusters (c,) dtype= [('cluster_label', 'int64'), ('cell_label', 'int64'), ('extremum_channel', 'int64'), ('extremum_amplitude', 'float64'), ('waveform_rms', 'float64'), ('nb_peak', 'int64'), ('tag', 'U16'), ('annotations', 'U32'), ('color', 'uint32')]
- some_peaks_index (M) int64
- some_waveforms (M, width, nb_channel) float32
- some_features (M, nb_feature) float32

- channel_to_features (nb_chan, nb_component) bool
- some_noise_snippet (nb_noise, width, nb_channel) float32
- some_noise_index (nb_noise,) int64
- some_noise_features (nb_noise, nb_feature) float32
- centroids_median (C, width, nb_channel) float32
- centroids_mad (C, width, nb_channel) float32
- centroids_mean (C, width, nb_channel) float32
- centroids_std (C, width, nb_channel) float32
- spike_waveforms_similarity (M, M) float32
- cluster_similarity (C, C) float32
- cluster_ratio_similarity (C, C) float32

auto_merge_high_similarity(*threshold=0.95*)
Recursively merge all pairs with similarity hihger that a given threshold

clean_waveforms(*alien_value_threshold=100.0*, *recompute_all_centroid=True*)
Detect bad waveform (artefact, . . .) and tag them with allien label (-9)

compute_spike_waveforms_similarity(*method='cosine_similarity'*, *size_max=10000000.0*)
This compute the similarity spike by spike.

create_savepoint()
this create a copy of the entire catalogue_constructor subdir Usefull for the UI when the user wants to snapshot and try tricky merge/split.

estimate_signals_noise(*seg_num=0*, *duration=10.0*)
This estimate the median and mad on processed signals on a short duration. This will be necessary for normalisation in next steps.

Note that if the noise is stable even a short duratio is OK.

seg_num: int segment index

duration: float duration in seconds

extract_some_features(*method='global_pca'*, *selection=None*, ***params*)
Extract feature from waveforms.

extract_some_noise(*nb_snippet=300*)
Find some snipet of signal that are not overlap with peak waveforms.

Usefull to project this noise with the same tranform as real waveform and see the distinction between waveforma and noise in the subspace.

extract_some_waveforms(*n_left=None*, *n_right=None*, *wf_left_ms=None*, *wf_right_ms=None*, *index=None*, *mode='rand'*, *nb_max=10000*, *nb_max_by_channel=600*, *recompute_all_centroid=True*)
Extract waveform snippet for a subset of peaks (already detected).

Note that this operation is slow.

After this the attribute some_peaks_index will contain index in all_peaks that have waveforms.

n_left: int Left sweep in sample must be negative

n_right: int Right sweep in sample

wf_left_ms: Left sweep in ms must be negative

wf_right_ms: Right sweep in ms must be negative

index: None (by default) or numpy array of int If mode is None then the user can give a selection index of peak to extract waveforms.

mode: 'rand' (default) or 'rand_by_channel' or 'all' or None 'rand' select randomly some peak to extract waveform. 'rand_by_channel' work only in mode 'sparse' and random by channel using the

nb_max_by_channel args

If None then index must not be None.

nb_max: int When rand then is this the number of selected waveform.

find_clusters(*method='kmeans'*, *selection=None*, *order=True*, ***kargs*)
Find cluster for peaks that have a waveform and feature.

find_good_limits(*mad_threshold=1.1*, *channel_percent=0.3*, *extract=True*, *min_left=-5*, *max_right=5*)
Find goods limits for the waveform. Where the MAD is above noise level (=1.)

The technics constists in finding continuous samples above 10% of backgroud noise for at least 30% of channels

Parameters

mad_threshold: (default 1.1) threshold noise channel_percent: (default 0.3) percent of channel above this noise.

flush_info()
Flush info (mainly parameters) to json files.

make_catalogue_for_peeler(***kargs*)
Make and save catalogue in the working dir for the Peeler.

order_clusters(*by='waveforms_rms'*)
This reorder labels from highest rms to lower rms. The higher rms the smaller label. Negative labels are not reassigned.

project(*method='global_pca'*, *selection=None*, ***params*)
Extract feature from waveforms.

re_detect_peak(***kargs*)
Peak are detected while **run_signalprocessor**. But in some case for testing other threshold we can **re-detect peak** without signal processing.

method ['global' or 'geometrical'] Method for detection.

engine: 'numpy' or 'opencl' or 'numba' Engine for peak detection.

peak_sign: '-' or '+' Signa of peak.

relative_threshold: int default 7 Threshold for peak detection. The preprocessed signal have units expressed in MAD (robust STD). So 7 is MAD*7.

peak_span_ms: float default 0.3 Peak span to avoid double detection. In second.

reload_data()
Reload persitent arrays.

run_signalprocessor(*duration=60.0*, *detect_peak=True*)
this run (chunk by chunk), the signal preprocessing chain on all segments.

The duration can be clip for very long recording. For catalogue construction the user must have the intuition of how signal we must have to get enough spike to detect clusters. If the duration is too short clusters will not be visible. If too long the processing time will be unacceptable. This totally depend on the dataset (nb channel, spike rate ...)

This also detect peak to avoid to useless access to the storage.

duration: float duration in seconds for each segment

detect_peak: bool (default True) Also detect peak.

set_global_params(*chunksize=1024*, *memory_mode='memmap'*, *internal_dtype='float32'*, *mode='dense'*, *adjacency_radius_um=None*, *sparse_threshold=1.5*)

chunksize: int. default 1024 Length of each chunk for processing. For real time, the latency is between chunksize and 2*chunksize.

memory_mode: 'memmap' or 'ram' default 'memmap' By default all arrays are persistent with memmap but you can also have everything in ram.

internal_dtype: 'float32' (or 'float64') Internal dtype for signals/waveforms/features. Support of intteger signal and waveform is planned for one day and should boost the process!!

mode: str dense or sparse Choose the global mode dense or sparse.

adjacency_radius_um: float or None When mode='sparse' then this must not be None.

set_peak_detector_params(*method='global'*, *engine='numpy'*, *peak_sign='-'*, *relative_threshold=7*, *peak_span_ms=0.3*, *adjacency_radius_um=None*)

Set parameters for the peak_detector engine

method ['global' or 'geometrical'] Method for detection.

engine: 'numpy' or 'opencl' or 'numba' Engine for peak detection.

peak_sign: '-' or '+' Signa of peak.

relative_threshold: int default 7 Threshold for peak detection. The preprocessed signal have units expressed in MAD (robust STD). So 7 is MAD*7.

peak_span_ms: float default 0.3 Peak span to avoid double detection. In millisecond.

set_preprocessor_params(*engine='numpy'*, *highpass_freq=300.0*, *lowpass_freq=None*, *smooth_size=0*, *common_ref_removal=False*, *lostfront_chunksize=None*)

Set parameters for the preprocessor engine

engine='numpy' or 'opencl' If you have pyopencl installed and correct ICD installed you can try 'opencl' for high channel count some critial part of the processing is done on the GPU.

highpass_freq: float dfault 300 High pass cut frquency of the filter. Can be None if the raw dataset is already filtered.

lowpass_freq: float default is None Low pass frequency of the filter. None by default. For high samplign rate a low pass at 5~8kHz can be a good idea for smoothing a bit waveform and avoid high noise.

smooth_size: int default 0 This a simple smooth convolution kernel. More or less act like a low pass filter. Can be use instead lowpass_freq.

common_ref_removal: bool. False by dfault. The remove the median of all channel sample by sample.

lostfront_chunksize: int. default None size in sample of the margin at the front edge for each chunk to avoid border effect in backward filter. In you don't known put None then lostfront_chunksize will be int(sample_rate/highpass_freq)*3 which is quite robust (<5% error) compared to a true offline filtfilt.

split_cluster(*label*, *method='kmeans'*, ***kargs*)
This split one cluster by applying a new clustering method only on the subset.

tag_same_cell(*labels_to_group*)
In some situation in spike burst the amplitude change baldly. In theses cases we can have 2 distinct cluster but the user suspect that it is the same cell. In such cases the user must not merge the 2 clusters because the centroids will represent nothing and and the Peeler.will fail.

Instead we tag the 2 differents cluster as "same cell" so same cell_label.

This is a manual action.

class tridesclous.peeler.**Peeler**(*dataio*)
The peeler is core of spike sorting itself. It basically do a *template matching* on a signals.

This class nedd a *catalogue* constructed by `CatalogueConstructor`. Then the compting is applied chunk chunk on the raw signal itself.

So this class is the same for both offline/online computing.

At each chunk, the algo is basically this one:

1. apply the processing chain (filter, normamlize, ...)
2. Detect peaks
3. Try to classify peak and detect the *jitter*
4. With labeled peak create a prediction for the chunk
5. Substract the prediction from the processed signals.
6. Go back to **2** until there is no peak or only peaks that can't be labeled.
7. return labeld spikes from this or previous chunk and the processed signals (for display or recoding)

The main difficulty in the implemtation is to deal with edge because spikes waveforms can spread out in between 2 chunk.

Note that the global latency depend on this é paramters:

- lostfront_chunksize
- chunksize

1.10 Release notes

1.10.1 tridesclous release notes 1.4.1

2019-12-20

- Some hack on pruningshears to improve benchmark

1.10.2 tridesclous release notes 1.4.0

2019-12-12

- Do not recompute signal processor at peeler stage when catalogue have done a full processing
- handle version inside dataio
- bug fix for SF

- some hack on pruningshears

1.10.3 tridesclous release notes 1.3.3

2019-12-05

Another Bug fix for PyQt5/pyqtgraph.

1.10.4 tridesclous release notes 1.3.2

2019-12-04

Bug fix for PyQt5/pyqtgraph.

1.10.5 tridesclous release notes 1.3.1

2019-11-28

Mainly bugfix for spikeforest.

Some minor improvement on pruningshears

1.10.6 tridesclous release notes 1.3.0

2019-11-21

Major improvement:

- new peak detector taking in account geometry
- CatalogueConstructor have a sparse mode to improve performance on high channel count (>64)
- Peeler have several implementation * classic in the same idea as the previous one * geometrical (new) that better advantage of geometry and sparsity of template
- improvement GUI when many channel

1.10.7 tridesclous release notes 1.2.0

2019-06-11

Release mainly for parameters names change and make spikeforest wrapper more robust.

1.10.8 tridesclous release notes 1.2.1

2019-06-17

1.10.9 tridesclous release notes 1.2.2

2019-07-11

Debug GUI catalogue when split trash Debug empty catalogue when OpenCl Some try with isosplit5 (optional dep)

1.10.10 tridesclous release notes 1.1.0

201-05-28

Release only for making spiketoolkit unitest working. No other special reason.

1.10.11 tridesclous release notes 1.0.0

2018-09-04

Very first official release of tridesclous after 18 month of developpement.

Main core classes:

- DataIO
- CatalogueConstructor
- Peeler

Interactive UI with PyQt5 and pytgraph:

- MainWindow
- CatalogueWindow
- PeelerWindow

Online:

- pyacq support with: * OnlinePeeler (processing pyacq Node) * OnlineTraceViewer (viewer pyacq Node) * OnlineWaveformHistViewer (viewer pyacq Node) * OnlineWindow (main online window)

Bonus:

- some opencl processing
- dataset for testing
- interface to neo.rawio
- matplotlib plotting

Code care:

- circle-ci
- appveyor
- readthedoc

Python Module Index

t

Index