ClearMap Documentation

Release 0.9.2

Christoph Kirst

December 19, 2015

CONTENTS

ClearMap is a toolbox for the analysis and registration of volumetric data from cleared tissues.

ClearMap has been designed to analyse large 3D image stack datasets obtained with Light Sheet Microscopy of iDISCO+ cleared mouse brains samples immunolabeled for nuclear proteins. ClearMap performs image registration to the Allen Institute Brain Atlas, volumetric image processing and object detection and statistical analysis. In particular, the tools in *ClearMap* have been written with the mapping of Immediate Early Genes in the brain as the primary application.

However, the tools included with *ClearMap* should also be more broadly useful for data obtained with other types of microscopes, other types of markers, and other clearing techniques. Moreover, the registration and region segmentation capabilities of ClearMap are not depending on the Atlases and annotations we used in our study. Users are free to import their own reference files and annotation files, so the use of *ClearMap* can be easily expanded to other species, other organs or samples.

ClearMap is written in Python 2.7, and is designed to take advantage of parallel processing capabilities of modern workstations. We hope the open structure of the code will enable in the future many new modules to be added to ClearMap to broaden the range of applications to different types of biological objects or structures.

CHAPTER

ONE

USING CLEARMAP

1.1 Overview of ClearMap

ClearMap is a toolbox to analyze and register microscopy images of cleared tissue. It is targeted towards cleared brain tissue using the *iDISCO+ Clearing Method* but can be used with any volumetric imaging data. ClearMap contains a large number of functions dedicated to many aspects of 3D image manipulation and object detection, which could open a lot of possibilities for advanced users. For most users however, all relevant functions are explained in the tutorial in the next section, which contains a classic application case for ClearMap.

The ClearMap code package is structured into four main modules:

- *IO* for reading and writing images and data
- *Alignment* for resampling, reorientation and registration of images onto references
- *Image Processing* for correcting and quantifying the image data
- *Analysis* for the statistical analysis of the data

1.1.1 IO

ClearMap supports a wide range of image formats with a particular focus on volumetric data packaged as stacks or individual files:

Format	Description
TIF	tif images and stacks
RAW / MHD	raw image files with optional mhd header file
NRRD	nearly raw raster data files
IMS	Imaris image files
pattern	folder, file list or file pattern of a stack of 2d images

We recommend using when possible the mhd format, which is more broadly compatible than tif or nrrd.

Note: ClearMap can read the image data from a Bitplane's Imaris, but can't export image data as an Imaris file.

Images are represented internally as numpy arrays. ClearMap assumes images in arrays are arranged as [x,y], [x,y,z] or [x,y,z,c] where x,y,z correspond to the x,y,z coordinates as when viewed in an image viewer such as *[ImageJ]* and c to a possible color channel.

ClearMap also supports several data formats for storing data points, such as cell center coordinates or intensities:

Format	Description
CSV	comma separated values in text file, for exporting to other programs
NPY	numpy binary file, faster and more compact format for the point data
VTK	vtk point data file, for exporting to some programs
IMS	Imaris data file, for writing points onto an existing Imaris file

points files simply contain all point coordinates arranged as an array of [x,y,z] coordinates where each line is a detected cell center. *intensities* files are companion to point files (only for csv and npy formats), where each line contains informations about intensity and detected size for the corresponding center in the point file. Each line in the array of the intensities file has 4 rows organized as follows:

Row	Description
0	Max intensity of the cell center in the raw data
1	Max intensity of the cell center after the DoG filtering.
2	Max intensity of the cell center after the background subtraction
3	Cell size in voxels after the watershed detection

1.1.2 Alignment

The Alignment module provides tools to resample, reorient and register volumetric images in a fast parallel way.

Image registration is done by interfacing to the *[Elastix]* software package. This package allows to align cleared mouse brains onto the Allan brain atlas *[ABA]*.

1.1.3 Image Processing

ClearMap provides a number of image processing tools with a focus on the processing of large 3D volumetric images in parallel. For the detection of objects in 3D, ClearMap has a modular architecture. For the user, most of this is hidden and called by the `SpotDetection` function (see the example script).

The main processing modules include:

Module	Description
`BackgroundRemoval`	Background estimation and removal via morphological opening
`IlluminationCorrection`	Correction of vignetting and other illumination errors
`Filter`	Filtering of the image via large set of filter kernels
`GreyReconstruction`	Reconstruction of images
`SpotDetection`	Detection of local peaks
`CellDetection`	Detection of cell centers
`CellSizeDetection`	Detection of cell shapes via watershed
`IlastikClassification`	Classification of voxels via interface to *[Ilastik]*

The modular structure of this sub-packages allows for fast and flexible integration of additional modules for future developments.

1.1.4 Analysis

This part of ClearMap provides a toolbox for the statistical analysis and visualization of detected cells or structures and region specific analysis of annotated data.

For cleared mouse brains aligned to the *[ABA]* a wide range of statistical analysis tools with respect to the annotated brain regions in the atlas is supported. Two types of analysis are done:

- Voxel statistics, which are based on the heat-map generated from the detected cell centers. These are usually represented as image stacks of mean, standard deviation, p-values with False Discovery Rate options.

- Region statistics, which are based on the annotated regions from the reference annotation file. They are usually represented as spreadsheets containing the statistics for each region.

The Key modules are:

Module	Description
Statistics	Statistical tools for the analysis of detected cells
Voxelization	For voxel-based statistics: voxelization of cells for visualization and analysis
Label	For region-based statistics: tools to analyse data with the annotated reference files

The use of the modules is explained in the tutorial.

1.1.5 iDISCO+ Clearing Method

Robust quantification of 3D datasets requires images as uniform as possible for the signal properties, both on each plane, and also at all imaging depths. The iDISCO+ method is an evolution of the iDISCO whole-mount labeling technique to improve the diffusion and background of staining in large samples *[Renier2014]*, and the 3DISCO clearing technique *[Erturk2012]*. The iDISCO+ staining and clearing method is combined optimally with the very large field of view enabled by light sheet microscopy, in particular the ultramicroscope optical design, which enables low magnification imaging with high speed and relatively high resolution.

The datasets used to develop ClearMap are usually composed of two channels:

- The signal channel. Typically obtained in the far-red light spectrum, where the optical properties of the cleared tissue are at their best for signal-to-noise and transparency. It is recommended when possible to use nuclear reporters or proteins to facilitate the object detection.
- The autofluorescence channel, usually collected in the blue-green light spectrum. The background tissue fluorescence highlights the major structures of the tissue to facilitate the 3D image registration. Only the contrast between regions is important here, so it doesn't matter if the relative intensities between regions are not the same as on the reference scans.

See these videos for example of light sheet imaging of cleared tissues:

- Dopaminergic system in the embryonic mouse
- Cortical and hippocampal neurons in the adult mouse brain

More info can be found on the *[iDISCO]* webpage.

1.1.6 References

1.2 Installation

1.2.1 Requirements

ClearMap is written in Python 2.7. It should run on any Python environment, but it also relies on external softwares such as Elastix which may not run optimally on Windows or Apple systems. For typical use, we recommend a workstation running Ubuntu 14 or later with at least 4 CPU cores, 64Gb of RAM and SSD disks. 128Gb of RAM and 6 cores or above will have much increased performances. The processing time however will depend greatly on the parameters set, so your experience may be different. Also, large hard drives may be needed to host the raw data, although 1 to 4Tb of storage space should be enough for most users.

1.2.2 Installation

To install ClearMap, first create a folder to contain all the files for the program. Then, open a terminal window, change to the directory hosting ClearMap and download the code from GIT by running this command:

```
>>> git clone https://git.assembla.com/idisco.git
```

If you're starting from scratch, you also need to download individually the following softwares:

- To do the alignement, you should download Elastix (http://elastix.isi.uu.nl)
- If you wish to use the machine learning filters, download Ilastik (http://ilastik.org), version 0.5 (this is the version implemented in ClearMap). This is an optional download, only if you wish to use this more complete object detection framework for complex objects.

And then the following libraries (most of them available from the software center of Ubuntu, or via the `pip` command or `apt-get` command in a terminal):

- Dev tools for Python 2.7 (from the software center)
- Matplotlib (from the software center)
- Numpy (from the software center)
- Scipy (from the software center)
- Skimage (from the software center)
- Mahotas (from the developer's website)
- h5py (from the software center) (for Imaris files input/output only)
- openCV (from the software center)
- MayaVI (from the software center)
- libboost-all-dev (from the software center)
- PyOpenGL (from the software center)
- qimage2ndarray (from the software center)
- PyQt4 (from the software center)
- tifffile (from the software center)
- EVTK (from the developer website) (only necessary, for output to vtk files)
- libhdf5-dev (from the software center)
- Cython (from the software center)

If you're planning to use Ilastik, download Vigra (from the developer website).

We use Spyder to run the code.

1.2.3 Configuration

Open the file ClearMap/Settings.py to set the paths of installations for Ilastik and Elastix:

```
>>> IlastikPath = '/usr/local/ilastik-05-rc-final';
>>> ElastixPath = '/usr/local/elastix';
```

Note that Ilastik is optional. If you haven't installed it, you can set the path to `None`. You can also set the installation to run on multiple machines by setting a host specific path:

```
>>> if hostname == 'kagalaska.nld':  #Christoph's Laptop
>>>         IlastikPath = '/home/ckirst/programs/ilastik-05/';
>>>         ElastixPath = '/home/ckirst/programs/elastix/';
>>>     elif hostname == 'mtllab-Ubuntu': #Nico's Workstation
>>>         IlastikPath = '/usr/local/ilastik-05-rc-final';
>>>         ElastixPath = '/usr/local/elastix';
```

1.3 Tutorial

The goal of this tutorial is to explain the scripts we used to analyse samples. As an example, we will use a dataset from a Light Sheet imaged adult mouse brain stained for c-fos. The tutorial files also contain an autofluorescence file to enable the registration of the scan to the reference atlas. The tutorial files are found in the ClearMap/Scripts folder. They consist of :

- *The Parameter File* This sets the parameters individually for each sample
- *The Run File* This will run all the commands to process each sample individually
- *Analysis Tools* This scripts will run the analysis tools and group statistics for all samples in the batch

A project will usually contain 1 parameter file for each sample, 1 run file for the whole experiment and 1 analysis file for the whole experiment.

1.3.1 The Parameter File

The parameter is a Python script that will contain all the necessary informations to process each sample. An example script, *parameter_file_template.py* is provided in the ClearMap/Scripts folder. It contains the following sections:

Section	Description
Import modules	load from ClearMap the functions used here
Data parameters	points to the files used, their resolution and orientation
Cell detection	parameters for the cell detection, and module used
Heat map generation	to generate a voxelized map of the detected cells
Config Parameters	the parameters for memory and processors management
Run Parameters	you would usually not change these. They specify how the data will be processed

Import Modules

You would usually not change these. They are all the functions that will be used later either in the parameter file or in the execution file.

Data parameters

This is where you point to the files used, their resolution and orientation. It also defines which atlas and annotation files to use.

To set the directory where all files will be read and written for this sample:

```
>>> BaseDirectory = '/home/mtllab/exploration/sample1';
```

To set the image files used for the processing:

```
>>> cFosFile = os.path.join(BaseDirectory, 'cfos/0_8x-cfos-Table Z\d{4}.ome.tif');
>>> AutofluoFile = os.path.join(BaseDirectory, 'autofluo/0_8xs3-autofluo-Table Z\d{4}.ome.tif');
```

Note the use of the command `os.path.join` to link the set `BaseDirectory` with the folder where the files are. On the LaVision ultramicroscope system, the images files are generated not as stacks, but as numbered files in the ome.tif format. Each Z stack will end by `-Table Z0000.ome.tif`. The 0000 is the plane number. To indicate **ClearMap** to read the next planes, replace the 4 digits with the command `\d{4}`. On our system, files for each channel (here, c-fos and background fluorescence) are saved in a different stack, in a different folder.

To restrict the range for the object detection:

```
>>> cFosFileRange = {'x' : all, 'y' : (180, 2600), 'z' : all};
```

This range will only affect the region used for the cell detection. It will not be taken into account for the 3D image registration to the reference Atlas, nor for the voxelization or other analysis. This is useful to limit the amount of memory used. In our example, we use the full x range, the full z range, but restrict the y range. The camera on our system, an Andor sNEO CMOS, has a sensor size of 2160 x 2660. However, the lens used on for the acquisition, an Olympus 2X 0.5NA MVPLAPO, has a strong corner deformation, so we restrict the y range because no cells can be reliably detected outside of this range.

As a reminder, in the image files, the (0, 0, 0) coordinate correspond to the upper left corner of the first plane. To the opposite, the (2160, 2660, 2400) coordinate will be the bottom right corner of the last plane (here 2400, but can vary).

When optimizing the parameters for the object detection, you should dramatically restrict the range to speed up the detection. We recommend using 500 planes, 500 pixels on each side:

```
>>> cFosFileRange = {'x' : (500, 1000), 'y' : (500, 1000), 'z' : (500, 1000)};
```

But of course adapting the range to where the relevant objects are on your sample.

Next, to set the resolution of the original data (in µm / pixel):

```
>>> OriginalResolution = (4.0625, 4.0625, 3);
```

In this example, this is set for a zoom factor of 0.8X on the LaVision system with the 2X lens. This information can be found in the metadata of the tif file usually. If you don't know the pixel size, we recommend opening the stack with the plugin BioFormat on ImageJ, and go to « image » -> « show info » to read the metadata. On the LaVision file, this information is at the end of the list.

The orientation of the sample has to be set to match the orientation of the Atlas reference files. It is not mandatory to acquire the sample in the same orientation as the atlas. For instance, you can acquire the left side of the brain, and map it onto the right side of the atlas:

```
>>> FinalOrientation = (1,2,3);
```

The convention is as follow (examples given, any configuration is possible):

Value of the tuple	Description
(1, 2, 3)	The scan has the same orientation as the atlas reference
(-1, 2, 3)	The x axis is mirrored compared to the atlas
(1, -2, 3)	The y axis is mirrored compared to the atlas
(2, 1, 3)	Performs a rotation by exchanging the x and y axis
(3, 2, 1)	Performs a rotation by exchanging the z and x axis

For our samples, we use the following orientation to match our atlas files:

- The right side of the brain is facing the objective, lateral side up.
- The rostral side of the brain is up
- The dorsal side is facing left

- The ventral side is facing right

This means that in our scans, if we want to image the right hemisphere, we use (1, 2, 3) and if we want to image the left hemisphere, we use (-1, 2, 3).

To set the output for the voxelized heat map file:

```
>>> VoxelizationFile = os.path.join(BaseDirectory, 'points_voxelized.tif');
```

To set the resolution of the Atlas Files (in um/ pixel):

```
>>> AtlasResolution = (25, 25, 25);
```

To choose which atlas files you would like to use:

```
>>> PathReg        = '/home/mtllab/Documents/warping';
>>> AtlasFile      = os.path.join(PathReg, 'half_template_25_right.tif');
>>> AnnotationFile = os.path.join(PathReg, 'annotation_25_right.tif');
```

It is important to make sure that the Atlas used is in the correct orientation (see above), but also don't contain too much information outside of the field of view. While the registration program can deal with a bit of extra « bleed » outside of the sample, this should be kept to a minimum. We usually prepare different crops of the atlas file to match the usual field of views we acquire.

Cell detection

At this point, two detection methods exist: the `SpotDetection` and `Ilastik`:

- `SpotDetection` is designed for globular objects, such as neuron cell bodies or nuclei. This is the fastest method, and offers a greater degree of fine controls over the sensibility of the detection. However, it is not well suited for complex objects.
- `Ilastik` is a framework that relies on the user generating a classifier through the graphical interface of the Ilastik program, by painting over a few objects and over the background. The program will then learn to classify the pixels between objects or backgrouns based on the user indications. This is a very easy way to tune very complex filters to detect complex objects or textures. However, the classification is a black box, and very dependent of the user's classification.

In this tutorial, we will use the SpotDetection method. To choose which method to use for the cell detection:

```
>>> ImageProcessingMethod = "SpotDetection";
```

The parameters for the Spot Detection methods are then sorted in « dictionaries » by theme :

Dictionary name	Description
correctIlluminationParameter	If you have an intensity profile for your microscope, you can correct variations in illuminations here
removeBackgroundParameter	To set the background subtraction via morphological opening
filterDoGParameter	To set the parameters for the Difference of Gaussian filter
findExtendedMaximaParameter	If the object contains multiple peaks of intensity, this will collapse them into one peak
findIntensityParameter	Often, the center of the mass of an object is not the voxel of highest intensity. This is a correction for this
detectCellShapeParameter	This set the parameters for the cell shape « painting » via the watershed

Correcting the illumination:

Because of the Gaussian shape of the light sheet and of the objecting lens vignetting, the sample illumination is not uniform. While correcting the illumination can improve the uniformity of the cell detection, it is usually not really necessary if all samples where imaged the same way, as the same anatomical features will be positioned in the same region of the lens across samples. Nevertheless, to correct for variation in the illumination use:

```
>>> correctIlluminationParameter = {
>>>     "flatfield"  : None,
>>>     "background" : None,
>>>     "scaling"    : "Mean",
>>>     "save"       : None,
>>>     "verbose"    : True
>>> }
```

For this tutorial, we will not use the correction, so the `flatfield` parameter is set to `None`. Please note that you need to generate an intensity profile for your system if you wish to use this function.

Background Subtraction:

This is the most important pre-treatment step, usually always turned on. The background subtraction via morphological opening is not very sensitive to the size parameter used, as long as it is in the range of the size of the objects detected. The parameters for the background subtraction are as follow:

```
>>> removeBackgroundParameter = {
>>>     "size"    : (7,7),
>>>     "save"    : None,
>>>     "verbose" : True
>>> }
```

The parameter `size` is a tuple with (x,y) in pixels and correspond to an ellipsoid of this size. Of importance, you can check the result of the background subtraction by setting the `save` parameter to a filename. This will output a series of tif images you can open with ImageJ to check the results. For instance you could set `save` like this:

```
>>>     "save"    : os.path.join(BaseDirectory, 'background/background\d{4}.ome.tif');
```

You have to use the `\d{4}` notation to save the files as a series of images, otherwise only the first plane is saved!

Note: Only use the `save` function when you analyse a small subset of your data, otherwise the full stack will be written to the disk. Don't forget to turn off this parameter when you're done optimizing the filters.

Difference of Gaussians filter:

filterDoGParameter = { "size" : None, # (tuple or None) size for the DoG filter if None, do not correct for any background "sigma" : None, # (tuple or None) std of outer Guassian, if None autmatically determined from size "sigma2" : None, # (tuple or None) std of inner Guassian, if None autmatically determined from size "save" : None, # (str or None) file name to save result of this operation if None dont save to file "verbose" : True # (bool or int) print / plot information about this step

}

findExtendedMaximaParameter = { "hMax" : None, # (float or None) h parameter for the initial h-Max transform, if None, do not perform a h-max transform "size" : 5, # (tuple) size for the structure element for the local maxima filter "threshold" : 0, # (float or None) include only maxima larger than a threshold, if None keep all

localmaxima “save” : None, # (str or None) file name to save result of this operation if None dont save to file “verbose” : True # (bool or int) print / plot information about this step

}

findIntensityParameter = { “method” : ‘Max’, # (str, func, None) method to use to determine intensity (e.g. “Max” or “Mean”) if None take intensities at the given pixels “size” : (3,3,3) # (tuple) size of the box on which to perform the *method*

}

detectCellShapeParameter = { “threshold” : 700, # (float or None) threshold to determine mask, pixel below this are background if None no mask is generated “save” : None, # (str or None) file name to save result of this operation if None dont save to file “verbose” : True # (bool or int) print / plot information about this step if None take intensities at the given pixels

}

Parameters for cell detection using spot detection algorithm detectSpotsParameter = {

“correctIlluminationParameter” : correctIlluminationParameter, “removeBackgroundParameter” : removeBackgroundParameter, “filterDoGParameter” : filterDoGParameter, “findExtendedMaximaParameter” : findExtendedMaximaParameter, “findIntensityParameter” : findIntensityParameter, “detectCellShapeParameter” : detectCellShapeParameter

}

Heat map generation

To set the output for the voxelized heat map file:

```
>>> VoxelizationFile = os.path.join(BaseDirectory, 'points_voxelized.tif');
```

1.3.2 The Run File

1.3.3 Analysis Tools

1.4 ClearMap Image Analysis Tools

Here we introduce the main image processing steps for the detection of nuclear-located signal with examples.

The data is a small region isolated from an iDISCO+ cleared mouse brain immunostained against c-fos. This small stack is included in the ClearMap package in the Test/Data/ImageAnalysis/ folder:

```
>>> import os
>>> import ClearMap.IO as io
>>> import ClearMap.Settings as settings
>>> filename = os.path.join(settings.ClearMapPath, 'Test/Data/ImageAnalysis/cfos-substack.tif');
```

1.4.1 Visualizing 3D Images

Large images in 3d are best viewed in specialized software, such as Imaris for 3D rendering or ImageJ to parse the stacks. For the full size data, it is recommended to open the stacks in ImageJ using the « virtual stack » mode.

In ClearMap we provide some basic visualization tools to inspect the 3d data in the module `ClearMap.Visualization.Plot`.

To load them run

```
>>> import ClearMap.Visualization.Plot as plt
```

Tiled Plots

In our experience results of 3d image analysis is inspected most accurately by plotting each horizontal plane in the image in tiles that are coupled for zooming. Intermediate results from all the steps of the SpotDetection can also be written as image stacks and opened with ImageJ.

```
>>> data = io.readData(filename, z = (0,26));
>>> plt.plotTiling(data);
```

To only plot a particular subregion its possible to specify the x,y,z range.

```
>>> plt.plotTiling(data, x = (0,70), y = (0,50), z = (10,16));
```

Sometimes inverse colors may be better:

```
>>> plt.plotTiling(data, inverse = True,  z = (10,16));
```

1.4.2 Image Statistics

It is useful to gather some information about the image initially. For larger images that don't fit in memory in ClearMap certain statistics can be gathered in parallel via the module *ClearMap.ImageProcessing.ImageStatistics* module.

```
>>> import ClearMap.ImageProcessing.ImageStatistics as stat
>>> print stat.calculateStatistics(filename, method = 'mean')
2305.4042155294119
```

To get more information about the progress use the `verbose` option

```
>>> print stat.calculateStatistics(filename, method = 'mean', verbose = True)
ChunkSize: Estimated chunk size 51 in 1 chunks!
Number of SubStacks: 1
Process 0: processing substack 0/1
Process 0: file          = /home/ckirst/Science/Projects/BrainActivityMap/Analysis/ClearMap/Test/Data
Process 0: segmentation  = <function calculateStatisticsOnStack at 0x7fee9c25dd70>
Process 0: ranges: x,y,z = <built-in function all>,<built-in function all>,(0, 51)
Process 0: Reading data of size (250, 250, 51): elapsed time: 0:00:00
Process 0: Processing substack of size (250, 250, 51): elapsed time: 0:00:00
Total Time Image Statistics: elapsed time: 0:00:00
2305.4042155294119
```

Image statistics can be very helpful for modules, such as Ilastik, requiring a different bit depth than the original 16 or 12 bits files, as it helps to determine how to resample the images to a lower bit depth.

1.4.3 Background Removal

One of the first steps is often to remove background variations. The `ClearMap.Imageprocessing.BackgroundRemoval` module can be used. It performs the background subtraction by morphological opening. The parameter is set as (x,y) where x and y are the diameter of an ellipsoid of a size close to the typical object detected in pixels. The intensity of the signal is greatly reduced after the filtering, but the signal-to-noise ration is increased:

```
>>> import ClearMap.ImageProcessing.BackgroundRemoval as bgr
>>> dataBGR = bgr.removeBackground(data.asype('float'), size=(5,5), verbose = True);
>>> plt.plotTiling(dataBGR, inverse = True, z = (10,16));
```

Note that if the background feature size is choosen to small, this may result in removal of cells:

```
>>> dataBGR = bgr.removeBackground(data.astype('float'), size=(3,3), verbose = True);
>>> plt.plotTiling(dataBGR, inverse = True, z = (10,16));
```

1.4.4 Image Filter

A useful feature is to filter an image. Here the `ClearMap.Imageprocessing.Filter` package can be used.

To detect cells center, the difference of Gaussians filter is a powerful way to increase the contrast between the cells and the background. The size is set as (x,y,z), and usually x, y and z are about the typical size in pixels of the detected object. As after the background subtraction, the intensity of the signal is again reduced after the filtering, but the signal-to-noise ration is dramatically increased:

```
>>> from ClearMap.ImageProcessing.Filter.DoGFilter import filterDoG
>>> dataDoG = filterDoG(dataBGR, size=(8,8,4), verbose = True);
>>> plt.plotTiling(dataDoG, inverse = True, z = (10,16));
```

1.4.5 Maxima Detection

The `ClearMap.ImageProcessing.MaximaDetection` module contains a set of useful functions for the detection of local maxima. A labeled image can be visualized using the `ClearMap.Visualization.Plot.plotOverlayLabel()` routine.

```
>>> from ClearMap.ImageProcessing.MaximaDetection import findExtendedMaxima
>>> dataMax = findExtendedMaxima(dataDoG, hMax = None, verbose = True, threshold = 10);
>>> plt.plotOverlayLabel(  dataDoG / dataDoG.max(), dataMax.astype('int'), z = (10,16))
```

Its easier to see when zoomed in:

```
>>> plt.plotOverlayLabel(  dataDoG / dataDoG.max(), dataMax.astype('int'), z = (10,16), x = (50,100),
```

Note that for some cells, a maxima label in this subset might not be visible as maxima are detected in the entire image in 3D and the actual maxima might lie in layers not shown above or below the current planes.

Once the maxima are detected the cell coordinates can be determined:

```
>>> from ClearMap.ImageProcessing.MaximaDetection import findCenterOfMaxima
>>> cells = findCenterOfMaxima(data, dataMax);
>>> print cells.shape
(3670, 3)
```

We can also overlay the cell coordinates in an image:

```
>>> plt.plotOverlayPoints(data, cells, z = (10,16))
```

1.4.6 Cell Shape Detection

Finally once the cell centers are detected the `ClearMap.ImageProcessing.CellShapedetection` module can be used to detect the cell shape via a watershed.

```
>>> from ClearMap.ImageProcessing.CellSizeDetection import detectCellShape
>>> dataShape = detectCellShape(dataDoG, cells, threshold = 15);
>>> plt.plotOverlayLabel(dataDoG / dataDoG.max(), dataShape, z = (10,16))
```

Now we can perform some measurements:

```
>>> from ClearMap.ImageProcessing.CellSizeDetection import findCellSize, findCellIntensity
>>> cellSizes = findCellSize(dataShape, maxLabel = cells.shape[0]);
>>> cellIntensities = findCellIntensity(dataBGR, dataShape,  maxLabel = cells.shape[0]);
```

and plot those:

```
>>> import matplotlib.pyplot as mpl
>>> mpl.figure()
>>> mpl.plot(cellSizes, cellIntensities, '.')
>>> mpl.xlabel('cell size [voxel]')
>>> mpl.ylabel('cell intensity [au]')
```

CHAPTER

TWO

CLEARMAP FUNCTIONS

2.1 ClearMap package

ClearMap Registration, Image Analysis and Statistics Library.

ClearMap is a python toolbox for the analysis and registration of volumetric data from cleared tissues.

ClearMap is targeted towards large lightsheet volumetric imaging data of iDISCO+ cleared mouse brains samples, their registration to the Alan brain atlas, volumetric image processing and statistical analysis.

Author

Christoph Kirst, The Rockefeller University, New York City, 2015

License

GNU GENERAL PUBLIC LICENSE Version 3

2.1.1 Subpackages

ClearMap.IO package

This sub-package provides routines to read and write data

Two types of data files are discriminated:

- *Image data*
- *Point data*

Image data are files with data from the microscopy or results representing visualization of the analysis in e.g. volumetric form.

Point data are lists of e.g. cell coordinates or measured intensitites.

Image data

Images are represented internally as numpy arrays. CleaMap assumes images in arrays are aranged as [x,y], [x,y,z] or [x,y,z,c] where x,y,z correspond to the x,y,z coordinates as when viewed in an image viewer such as ImageJ. The c coordinate is a possible color channel.

Note: Many image libraries read images as [y,x,z] or [y,x] arrays!

The ClearMap toolbox supports a range of (volumetric) image formats:

Format	Descrition	Module
TIF	tif images and stacks	TIF
RAW / MHD	raw image files with optional mhd header file	RAW
NRRD	nearly raw raster data files	*NRRD*
IMS	imaris image file	Imaris
reg exp	folder, file list or file pattern of a stack of 2d images	*FileList*

The image format is inferred automatically from the file name extension.

For example to read image data use *readData()*:

```
>>> import os
>>> import ClearMap.IO as io
>>> import ClearMap.Settings as settings
>>> filename = os.path.join(settings.ClearMapPath,'Test/Data/Tif/test.tif');
>>> data = io.readData(filename);
>>> print data.shape
(20, 50, 10)
```

To write image data use *writeData()*:

```
>>> import os, numpy
>>> import ClearMap.IO as io
>>> import ClearMap.Settings as settings
>>> filename = os.path.join(settings.ClearMapPath,'Test/Data/Tif/test.tif');
>>> data = numpy.random.rand(20,50,10);
>>> data[5:15, 20:45, 2:9] = 0;
>>> data = 20 * data;
>>> data = data.astype('int32');
>>> res = io.writeData(filename, data);
>>> print io.dataSize(res);
(20, 50, 10)
```

Generally, the IO module is desinged to work with image sources which can be either files or already loaded numpy arrays. This is important to enable flexible parallel processing, without rewriting the data analysis routines.

For example:

```
>>> import numpy
>>> import ClearMap.IO as io
>>> data = numpy.random.rand(20,50,10);
>>> res = io.writeData(None, data);
>>> print res.shape;
(20, 50, 10)
```

Range parameter can be passed in order to only load sub sets of image data, usefull when the imags are very large. For example to load a sub-image

```
>>> import os, numpy
>>> import ClearMap.IO as io
>>> import ClearMap.Settings as settings
>>> filename = os.path.join(settings.ClearMapPath,'Test/Data/Tif/test.tif');
>>> res = io.readData(filename, data, x = (0,3), y = (4,6), z = (1,4));
>>> print res.shape;
(3, 2, 3)
```

Point data

ClearMap also supports several data formats for storing arrays of points, such as cell center coordinates or intensities.

Points are assumed to be an array of coordinates where the first array index is the point number and the second the spatial dimension, i.e. [i,d] The spatial dimension can be extended with additional dimensions for intensity ,easires or other properties.

Points can also be given as tuples (coordinate arrray, property array).

ClearMap supports the following files formats fro point like data:

Format	Descrition	Module
CSV	comma separated values in text file	*CSV*
NPY	numpy binary file	*NPY*
VTK	vtk point data file	*VTK*

The point file format is inferred automatically from the file name extension.

For example to read point data use *readPoints()*:

```
>>> import os
>>> import ClearMap.IO as io
>>> import ClearMap.Settings as settings
>>> filename = os.path.join(settings.ClearMapPath, 'Test/ImageProcessing/points.txt');
>>> points = io.readPoints(filename);
>>> print points.shape
(5, 3)
```

and to write it use *writePoints()*:

```
>>> import os, numpy
>>> import ClearMap.IO as io
>>> import ClearMap.Settings as settings
>>> filename = os.path.join(settings.ClearMapPath, 'Test/ImageProcessing/points.txt');
>>> points = numpy.random.rand(5,3);
>>> io.writePoints(filename, points);
```

Summary

- All routines accesing data or data properties accept file name strings or numpy arrays or None
- Numerical arrays represent data and point coordinates as [x,y,z] or [x,y]

ClearMap.IO.IO module

IO interface to read microscope and point data

Main module to distribute read and writing of individual data formats to the specialized sub-modules

See *ClearMap.IO* for details.

Author

Christoph Kirst, The Rockefeller University, New York City, 2015

pointFileExtensions = ['csv', 'txt', 'npy', 'vtk', 'ims']
list of extensions supported as a point data file

pointFileTypes = ['CSV', 'NPY', 'VTK', 'Imaris']
list of point data file types

pointFileExtensionToType = {'txt': 'CSV', 'vtk': 'VTK', 'csv': 'CSV', 'npy': 'NPY', 'ims': 'Imaris'}
map from point file extensions to point file types

dataFileExtensions = ['tif', 'tiff', 'mhd', 'raw', 'ims', 'nrrd']
list of extensions supported as a image data file

dataFileTypes = ['FileList', 'TIF', 'RAW', 'NRRD', 'Imaris']
list of image data file types

dataFileExtensionToType = {'tiff': 'TIF', 'mhd': 'RAW', 'nrrd': 'NRRD', 'raw': 'RAW', 'ims': 'Imaris', 'tif': 'TIF'}
map from image file extensions to image file types

fileExtension (*filename*)
Returns file extension if exists

Parameters **filename** (*str*) – file name

Returns *str* – file extension or None

isFile (*source*)
Checks if filename is a real file, returns false if it is directory or regular expression

Parameters **source** (*str*) – source file name

Returns *bool* – true if source is a real file

isFileExpression (*source*)
Checks if filename is a regular expression denoting a file list

Parameters **source** (*str*) – source file name

Returns *bool* – true if source is regular expression with a digit label

isPointFile (*source*)
Checks if a file is a valid point data file

Parameters **source** (*str*) – source file name

Returns *bool* – true if source is a point data file

isDataFile (*source*)
Checks if a file is a valid image data file

Parameters **source** (*str*) – source file name

Returns *bool* – true if source is an image data file

createDirectory (*filename*)
Creates the directory of the file if it does not exists

Parameters **filename** (*str*) – file name

Returns *str* – directory name

pointFileNameToType (*filename*)
Returns type of a point file

Parameters **filename** (*str*) – file name

Returns *str* – point data type in *pointFileTypes*

dataFileNameToType (*filename*)
Returns type of a image data file

Parameters **filename** (*str*) – file name

Returns *str* – image data type in *dataFileTypes*

dataFileNameToModule (*filename*)
Return the module that handles io for a data file

Parameters **filename** (*str*) – file name

Returns *object* – sub-module that handles a specific data type

pointFileNameToModule (*filename*)
Return the module that handles io for a point file

Parameters **filename** (*str*) – file name

Returns *object* – sub-module that handles a specific point file type

dataSize (*source*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*, ***args*)
Returns array size of the image data needed when read from file and reduced to specified ranges

Parameters

- **source** (*array or str*) – source data
- **x,y,z** (*tuple or all*) – range specifications, `all` is full range

Returns *tuple* – size of the image data after reading and range reduction

dataZSize (*source*, *z=<built-in function all>*, ***args*)
Returns size of the array in the third dimension, None if 2D data

Parameters

- **source** (*array or str*) – source data
- **z** (*tuple or all*) – z-range specification, `all` is full range

Returns *int* – size of the image data in z after reading and range reduction

toDataRange (*size*, *r=<built-in function all>*)
Converts range r to numeric range (min,max) given the full array size

Parameters

- **size** (*tuple*) – source data size
- **r** (*tuple or all*) – range specification, `all` is full range

Returns *tuple* – absolute range as pair of integers

See also:

toDataSize(), *dataSizeFromDataRange()*

toDataSize (*size*, *r=<built-in function all>*)
Converts full size to actual size given range r

Parameters

- **size** (*tuple*) – data size
- **r** (*tuple or all*) – range specification, `all` is full range

Returns *int* – data size

See also:

toDataRange(), *dataSizeFromDataRange()*

dataSizeFromDataRange (*dataSize*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*, ***args*)
Converts full data size to actual size given ranges for x,y,z

Parameters

- **dataSize** (*tuple*) – data size

- **x,z,y** (*tuple or all*) – range specifications, `all` is full range

Returns *tuple* – data size as tuple of integers

See also:

toDataRange(), *toDataSize()*

dataToRange(*data*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*, ***args*)
Reduces data to specified ranges

Parameters

- **data** (*array*) – full data array
- **x,z,y** (*tuple or all*) – range specifications, `all` is full range

Returns *array* – reduced data

See also:

dataSizeFromDataRange()

readData(*source*, ***args*)
Read data from one of the supported formats

Parameters

- **source** (*str, array or None*) – full data array, if numpy array simply reduce its range
- **x,z,y** (*tuple or all*) – range specifications, `all` is full range
- ****args** – further arguments specific to image data format reader

Returns *array* – data as numpy array

See also:

writeData()

writeData(*sink*, *data*, ***args*)
Write data to one of the supported formats

Parameters

- **sink** (*str, array or None*) – the destination for the data, if None the data is returned directly
- **data** (*array or None*) – data to be written
- ****args** – further arguments specific to image data format writer

Returns *array, str or None* – data or file name of the written data

See also:

readData()

copyFile(*source*, *sink*)
Copy a file from source to sink

Parameters

- **source** (*str*) – file name of source
- **sink** (*str*) – file name of sink

Returns *str* – name of the copied file

See also:

copyData(), *convertData()*

copyData (*source*, *sink*)
Copy a data file from source to sink, which can consist of multiple files

Parameters

- **source** (*str*) – file name of source
- **sink** (*str*) – file name of sink

Returns *str* – name of the copied file

See also:

copyFile(), *convertData()*

convertData (*source*, *sink*, ***args*)
Transforms data from source format to sink format

Parameters

- **source** (*str*) – file name of source
- **sink** (*str*) – file name of sink

Returns *str* – name of the copied file

Warning: Not optimized for large image data sets

See also:

copyFile(), *copyData()*

toMultiChannelData (**args*)
Concatenate single channel arrays to one multi channel array

Parameters ***args** (*arrays*) – arrays to be concatenated

Returns *array* – concatenated multi-channel array

pointsToCoordinates (*points*)
Converts a (coordiantes, properties) tuple to the coordinates only

Parameters **points** (*array or tuple*) – point data to be reduced to coordinates

Returns *array* – coordiante data

Notes

Todo: Move this to a class that handles points and their meta data

pointsToProperties (*points*)
Converts a (coordiante, properties) tuple to the properties only

Parameters **points** (*array or tuple*) – point data to be reduced to properties

Returns *array* – property data

Notes

Todo: Move this to a class that handles points and their meta data

pointsToCoordinatesAndProperties (*points*)
Converts points in various formats to a (coordinates, properties) tuple

Parameters **points** (*array or tuple*) – point data to be converted to (coordinates, properties) tuple

Returns *tuple* – (coordinates, properties) tuple

Notes

Todo: Move this to a class that handles points and their meta data

pointsToCoordinatesAndPropertiesFileNames (*filename*, *propertiesPostfix='_intensities'*, ***args*)
Generates a tuple of filenames to store coordinates and properties data separately

Parameters

- **filename** (*str*) – point data file name
- **propertiesPostfix** (*str*) – postfix on file name to indicate property data

Returns *tuple* – (file name, file name for properties)

Notes

Todo: Move this to a class that handles points and their meta data

pointShiftFromRange (*dataSize*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*, ***args*)
Calculate shift of points given a specific range restriction

Parameters

- **dataSize** (*str*) – data size of the full image
- **x,y,z** (*tuples or all*) – range specifications

Returns *tuple* – shift of points from original origin of data to origin of range reduced data

pointsToRange (*points*, *dataSize=<built-in function all>*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*, *shift=False*, ***args*)
Restrict points to a specific range

Parameters

- **points** (*array or str*) – point source
- **dataSize** (*str*) – data size of the full image
- **x,y,z** (*tuples or all*) – range specifications
- **shift** (*bool*) – shift points to relative coordinates in the reduced image

Returns *tuple* – points reduced in range and optionally shifted to the range reduced origin

readPoints (*source*, ***args*)
Read a list of points from csv or vtk

Parameters

- **source** (*str, array, tuple or None*) – the data source file
- ****args** – further arguments specific to point data format reader

Returns *array or tuple or None* – point data of source

See also:

writePoints()

writePoints (*sink*, *points*, ***args*)
Write a list of points to csv, vtk or ims files

Parameters

- **sink** (*str or None*) – the destination for the point data
- **points** (*array or tuple or None*) – the point data, optionally as (coordinates, properties) tuple
- ****args** – further arguments specific to point data format writer

Returns *str or array or tuple or None* – point data of source

See also:

readPoints()

writeTable (*filename*, *table*)
Writes a numpy array with column names to a csv file.

Parameters

- **filename** (*str*) – filename to save table to
- **table** (*annotated array*) – table to write to file

Returns *str* – file name

ClearMap.IO.CSV module

Interface to write csv files of cell coordinates / intensities

The module utilizes the csv file writer/reader from numpy.

Example

```
>>> import os, numpy
>>> import ClearMap.IO.CSV as csv
>>> import ClearMap.Settings as settings
>>> filename = os.path.join(settings.ClearMapPath, 'Test/ImageProcessing/points.txt');
>>> points = numpy.random.rand(5,3);
>>> csv.writePoints(filename, points);
>>> points2 = csv.readPoints(filename);
>>> print points2.shape
(5, 3)
```

writePoints (*filename*, *points*, ***args*)
Write point data to csv file

Parameters

- **filename** (*str*) – file name

- **points** (*array*) – point data

Returns *str* – file name

readPoints (*filename*, ***args*)
Read point data to csv file

Parameters

- **filename** (*str*) – file name
- ****args** – arguments for `pointsToRange()`

Returns *str* – file name

test ()
Test CSV module

ClearMap.IO.FileList module

Interface to read/write image stacks saved as a list of files

The filename is given as regular expression as described here.

It is assumd that there is a single digit like regular expression in the file name, i.e. `\d{4}` indicates a placeholder for an integer with four digits using traling 0s and `\d{}` would jus asume an integer with variable size.

For example: `/test\d{4}.tif` or `/test\d{}.tif`

Examples

```
>>> import os, numpy
>>> import ClearMap.Settings as settings
>>> import ClearMap.IO.FileList as fl
>>> filename = os.path.join(settings.ClearMapPath, 'Test/Data/FileList/test\d{4}.tif')
>>> data = numpy.random.rand(20,50,10);
>>> data = data.astype('int32');
>>> fl.writeData(filename, data);
>>> img = fl.readData(filename);
>>> print img.shape
(20, 50, 10)
```

readFileList (*filename*)
Returns list of files that match the regular expression

Parameters **filename** (*str*) – file name as regular expression

Returns *str, list* – path of files, file names that match the regular expression

splitFileExpression (*filename*)
Split the regular expression at the digit place holder

Parameters **filename** (*str*) – file name as regular expression

Returns *tuple* – file header, file extension, digit format

fileExperssionToFileName (*filename*, *z*)
Insert a number into the regular expression

Parameters

- **filename** (*str*) – file name as regular expression

- **z** (*int*) – z slice index

Returns *str* – file name

dataSize (*filename*, ***args*)
Returns size of data stored as a file list

Parameters

- **filename** (*str*) – file name as regular expression
- **x,y,z** (*tuple*) – data range specifications

Returns *tuple* – data size

dataZSize (*filename*, *z=<built-in function all>*, ***args*)
Returns size of data stored as a file list

Parameters

- **filename** (*str*) – file name as regular expression
- **z** (*tuple*) – z data range specification

Returns *int* – z data size

readDataFiles (*filename*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*, ***args*)
Read data from individual images assuming they are the z slices

Parameters

- **filename** (*str*) – file name as regular expression
- **x,y,z** (*tuple*) – data range specifications

Returns *array* – image data

readData (*filename*, ***args*)
Read image stack from single or multiple images

Parameters

- **filename** (*str*) – file name as regular expression
- **x,y,z** (*tuple*) – data range specifications

Returns *array* – image data

writeData (*filename*, *data*, *startIndex=0*)
Write image stack to single or multiple image files

Parameters

- **filename** (*str*) – file name as regular expression
- **data** (*array*) – image data
- **startIndex** (*int*) – index of first z-slice

Returns *str* – file name as regular expression

copyData (*source*, *sink*)
Copy a data file from source to sink when for entire list of files

Parameters

- **source** (*str*) – file name pattern of source
- **sink** (*str*) – file name pattern of sink

Returns *str* – file name patttern of the copy

test()
Test FileList module

ClearMap.IO.Imaris module

ClearMap.IO.NPY module

Interface to write binary files for point like data

The interface is based on the numpy library.

Example

```
>>> import os, numpy
>>> import ClearMap.Settings as settings
>>> import ClearMap.IO.NPY as npy
>>> filename = os.path.join(settings.ClearMapPath, 'Test/Data/NPY/points.npy');
>>> points = npy.readPoints(filename);
>>> print points.shape
(5, 3)
```

Author

Christoph Kirst, The Rockefeller University, New York City, 2015

writePoints(*filename*, *points*, ***args*)

readPoints(*filename*, ***args*)

test()
Test NPY module

ClearMap.IO.NRRD module

Interface to NRRD volumetric image data files.

The interface is based on nrrd.py, an all-python (and numpy) implementation for reading and writing nrrd files. See http://teem.sourceforge.net/nrrd/format.html for the specification.

Example

```
>>> import os, numpy
>>> import ClearMap.Settings as settings
>>> import ClearMap.IO.NRRD as nrrd
>>> filename = os.path.join(settings.ClearMapPath, 'Test/Data/Nrrd/test.nrrd');
>>> data = nrrd.readData(filename);
>>> print data.shape
(20, 50, 10)
```

Author

Copyright 2011 Maarten Everts and David Hammond.

Modified to integrate into ClearMap framework: - Christoph Kirst, The Rockefeller University, New York City, 2015

exception NrrdError
Bases: exceptions.Exception

Exceptions for Nrrd class.

parse_nrrdvector(*inp*)
Parse a vector from a nrrd header, return a list.

parse_optional_nrrdvector(*inp*)
Parse a vector from a nrrd header that can also be none.

readHeader(*filename*)
Parse the fields in the nrrd header

nrrdfile can be any object which supports the iterator protocol and returns a string each time its next() method is called — file objects and list objects are both suitable. If csvfile is a file object, it must be opened with the 'b' flag on platforms where that makes a difference (e.g. Windows)

```
>>> readHeader(("NRRD0005", "type: float", "dimension: 3"))
{'type': 'float', 'dimension': 3, 'keyvaluepairs': {}}
>>> readHeader(("NRRD0005", "my extra info:=my : colon-separated : values"))
{'keyvaluepairs': {'my extra info': 'my : colon-separated : values'}}
```

readData(*filename*, ***args*)
Read nrrd file image data

Parameters

- **filename** (*str*) – file name as regular expression
- **x,y,z** (*tuple*) – data range specifications

Returns *array* – image data

writeData(*filename*, *data*, *options={}*, *separateHeader=False*)
Write data to nrrd file

Parameters

- **filename** (*str*) – file name as regular expression
- **data** (*array*) – image data
- **options** (*dict*) – options dictionary
- **separateHeader** (*bool*) – write a separate header file

Returns *str* – nrrd output file name

To sample date use *options['spacings'] = [s1, s2, s3]* for 3d data with sampling deltas *s1*, *s2*, and *s3* in each dimension.

dataSize(*filename*, ***args*)
Read data size from nrrd image

Parameters

- **filename** (*str*) – file name as regular expression
- **x,y,z** (*tuple*) – data range specifications

Returns *tuple* – data size

dataZSize (*filename*, *z=<built-in function all>*, ***args*)
Read data z size from nrrd image

Parameters

- **filename** (*str*) – file name as regular expression
- **z** (*tuple*) – z data range specification

Returns *int* – z data size

copyData (*source*, *sink*)
Copy an nrrd file from source to sink

Parameters

- **source** (*str*) – file name pattern of source
- **sink** (*str*) – file name pattern of sink

Returns *str* – file name of the copy

Notes

Todo: dealt with nrdh header files!

test ()
Test NRRD IO module

ClearMap.IO.RAW module

ClearMap.IO.TIF module

ClearMap.IO.VTK module

Interface to write points to VTK files

Notes

- points are assumed to be in [x,y,z] coordinates as standard in ClearMap
- reading of points not supported at the moment!

Author

Christoph Kirst, The Rockefeller University, New York City, 2015

Modified from matlab code by Kannan Umadevi Venkataraju

writePoints (*filename*, *points*, *labelImage=None*)
Write point data to vtk file

Parameters

- **filename** (*str*) – file name
- **points** (*array*) – point data
- **labelImage** (*str, array or None*) – optional label image to determine point label

Returns *str* – file name

readPoints (*filename*, ***args*)
Read points form vtk file

Notes

•Not implmented yet !

ClearMap.Alignment package

This sub-package provides an interface to alignment tools in order to register cleared samples to atlases or reference samples.

Supported functionality:

- resampling and reorientation of large volumetric images in the `Resampling` module.
- registering volumetric data onto references via Elastix in the `Elastix` module.

Main routines for resampling are: `resampleData()` and `resamplePoints()`.

Main routines for elastix registration are: `alignData()`, `transformData()` and `transformPoints()`.

ClearMap.Alignment.Elastix module

ClearMap.Alignment.Resampling module

ClearMap.ImageProcessing package

This sub-package provides routines for volumetric image processing in parallel

This part of the *ClearMap* toolbox is desinged in a modular way to allow for fast and flexible extension and addition of specific image processing algorithms.

The toolbox part consists of two parts:

- *Volumetric Image Processing*
- *Parallel Image Processing*

Volumetric Image Processing

The image processing routines provided in the standard package are listed below

Module	Descrition
`BackgroundRemoval`	Background estimation and removal via morphological opening
`IlluminationCorrection`	Correction of vignetting and other illumination errors
`GreyReconstruction`	Reconstruction of images
`Filter`	Filtering of images via a large set of filter kernels
`MaximaDetection`	Detection of maxima and h-max transforms
`SpotDetection`	Detection of local peaks / spots / nuclei
`CellDetection`	Detection of cells
`CellSizeDetection`	Detection of cell shapes and volumes via e.g. watershed
`IlastikClassification`	Classification of voxels via interface to Ilastik

While some of these modules provide basic volumetric image processing functionality some routines combine those functions to provide predefined higher level cell detection, cell size and intensity measurements.

The higher level routines are optimized for iDISCO+ cleared mouse brain samples stained for cfos expression. Other data sets might require a redesign of these higher level functions.

Parallel Image Processing

For large volumetric image data sets from e.g. light sheet microscopy parallel processing is essential to speed up calculations.

In this toolbox the image processing is parallized via splitting a volumetric image stack into several sub-stacks, typically in z-direction. Because most of the image processig steps are non-local sub-stacks are created with overlaps and the results rejoined accordingly to minimize boundary effects.

Parallel processing is handled via the *StackProcessing* module.

External Packages

The *ImageProcessing* module makes use of external image processing packages including:

- Open Cv2
- Scipy
- Scikit-Image
- Ilastik

Routines form these packages were freely choosen to optimize for speed and memory consumption

ClearMap.ImageProcessing.StackProcessing module

Process a image stack in parallel or sequentially

In this toolbox image processing is parallized via splitting a volumetric image stack into several sub-stacks, typically in z-direction. As most of the image processig steps are non-local sub-stacks are created with overlaps and the results rejoined accordingly to minimize boundary effects.

Parallel processing is handled via this module.

Sub-Stacks The parallel processing module creates a dictionary with information on the sub-stack as follows:

Key	Description
`stackId`	id of the sub-stack
`nStacks`	total number of sub-stacks
`source`	source file/folder/pattern of the stack
`x`, `y`, `z`	the range of the sub-stack with in the full image
`zCenters`	tuple of the centers of the overlaps
`zCenterIndices`	tuple of the original indices of the centers of the overlaps
`zSubStackCenterIndices`	tuple of the indices of the sub-stack that correspond to the overlap centers

For exmaple the *writeSubStack()* routine makes uses of this information to write out only the sub-parts of the image that is will contribute to the final total image.

printSubStackInfo(*subStack*, *out=<open file '<stdout>', mode 'w'>*)
Print information about the sub-stack

Parameters

- **subStack** (*dict*) – the sub-stack info
- **out** (*object*) – the object to write the information to

writeSubStack (*filename*, *img*, *subStack=None*)
Write the non-redundant part of a sub-stack to disk

The routine is used to write out images when porcessed in parallel. It assumes that the filename is a patterned file name.

Parameters

- **filename** (*str or None*) – file name pattern as described in `FileList`, if None return as array
- **img** (*array*) – image data of sub-stack
- **subStack** (*dict or None*) – sub-stack information, if None write entire image see *Sub-Stacks*

Returns *str or array* – the file name pattern or image

joinPoints (*results*, *subStacks=None*, *shiftPoints=True*, ***args*)
Joins a list of points obtained from processing a stack in chunks

Parameters

- **results** (*list*) – list of point results from the individual sub-processes
- **subStacks** (*list or None*) – list of all sub-stack information, see *Sub-Stacks*
- **shiftPoints** (*bool*) – if True shift points to refer to origin of the image stack considered when range specification is given. If False, absolute position in entire image stack.

Returns *tuple* – joined points, joined intensities

calculateChunkSize (*size*, *processes=2*, *chunkSizeMax=100*, *chunkSizeMin=30*, *chunkOverlap=15*, *chunkOptimization=True*, *chunkOptimizationSize=<built-in function all>*, *verbose=True*)
Calculates the chunksize and other info for parallel processing

The sub stack information is described in *Sub-Stacks*

Parameters

- **processes** (*int*) – number of parallel processes
- **chunkSizeMax** (*int*) – maximal size of a sub-stack
- **chunkSizeMin** (*int*) – minial size of a sub-stack
- **chunkOverlap** (*int*) – minimal sub-stack overlap
- **chunkOptimization** (*bool*) – optimize chunck sizes to best fit number of processes
- **chunkOptimizationSize** (*bool or all*) – if True only decrease the chunk size when optimizing
- **verbose** (*bool*) – print information on sub-stack generation

Returns *tuple* – number of chunks, z-ranges of each chunk, z-centers in overlap regions

calculateSubStacks (*source*, *z=<built-in function all>*, *x=<built-in function all>*, *y=<built-in function all>*, ***args*)
Calculates the chunksize and other info for parallel processing and returns a list of sub-stack objects

The sub-stack information is described in *Sub-Stacks*

Parameters

- **source** (*str*) – image source
- **x,y,z** (*tuple or all*) – range specifications
- **processes** (*int*) – number of parallel processes
- **chunkSizeMax** (*int*) – maximal size of a sub-stack
- **chunkSizeMin** (*int*) – minial size of a sub-stack
- **chunkOverlap** (*int*) – minimal sub-stack overlap
- **chunkOptimization** (*bool*) – optimize chunck sizes to best fit number of processes
- **chunkOptimizationSize** (*bool or all*) – if True only decrease the chunk size when optimizing
- **verbose** (*bool*) – print information on sub-stack generation

Returns *list* – list of sub-stack objects

noProcessing(*img*, ***parameter*)

Perform no image processing at all and return original image

Used as the default functon in *parallelProcessStack()* and *sequentiallyProcessStack()*.

Parameters **img** (*array*) – imag

Returns *(array)* – the original image

parallelProcessStack(*source*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*, *sink=None*, *processes=2*, *chunkSizeMax=100*, *chunkSizeMin=30*, *chunkOverlap=15*, *chunkOptimization=True*, *chunkOptimizationSize=<built-in function all>*, *function=<function noProcessing>*, *join=<function joinPoints>*, *verbose=False*, ***parameter*)

Parallel process a image stack

Main routine that distributes image processing on paralllel processes.

Parameters

- **source** (*str*) – image source
- **x,y,z** (*tuple or all*) – range specifications
- **sink** (*str or None*) – destination for the result
- **processes** (*int*) – number of parallel processes
- **chunkSizeMax** (*int*) – maximal size of a sub-stack
- **chunkSizeMin** (*int*) – minial size of a sub-stack
- **chunkOverlap** (*int*) – minimal sub-stack overlap
- **chunkOptimization** (*bool*) – optimize chunck sizes to best fit number of processes
- **chunkOptimizationSize** (*bool or all*) – if True only decrease the chunk size when optimizing
- **function** (*function*) – the main image processing script
- **join** (*function*) – the fuction to join the results from the image processing script
- **verbose** (*bool*) – print information on sub-stack generation

Returns *str or array* – results of the image processing

sequentiallyProcessStack(*source*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*, *sink=None*, *chunkSizeMax=100*, *chunkSizeMin=30*, *chunkOverlap=15*, *function=<function noProcessing>*, *join=<function joinPoints>*, *verbose=False*, ***parameter*)

Sequential image processing on a stack

Main routine that sequentially processes a large image on sub-stacks.

Parameters

- **source** (*str*) – image source
- **x,y,z** (*tuple or all*) – range specifications
- **sink** (*str or None*) – destination for the result
- **processes** (*int*) – number of parallel processes
- **chunkSizeMax** (*int*) – maximal size of a sub-stack
- **chunkSizeMin** (*int*) – minial size of a sub-stack
- **chunkOverlap** (*int*) – minimal sub-stack overlap
- **chunkOptimization** (*bool*) – optimize chunck sizes to best fit number of processes
- **chunkOptimizationSize** (*bool or all*) – if True only decrease the chunk size when optimizing
- **function** (*function*) – the main image processing script
- **join** (*function*) – the fuction to join the results from the image processing script
- **verbose** (*bool*) – print information on sub-stack generation

Returns *str or array* – results of the image processing

ClearMap.ImageProcessing.CellDetection module

ClearMap.ImageProcessing.CellSizeDetection module

Subpackages

ClearMap.ImageProcessing.Filter package This sub-package provides various volumetric filter kernels and structure elments

A set of linear filters can be applied to the data using `LinearFilter`.

Because its utility for cell detection the difference of Gaussians filter is implemented directly in `DoGFilter`.

The fitler kernels defined in *`FilterKernel`* can be used in combination with the `Convolution` module.

Structured elements defined in `StructureElements` can be used in combination with various morphological operations, e.g. used in the :mod:~ClearMap.ImageProcessing.BackgroundRemoval‘ module.

ClearMap.ImageProcessing.Filter.LinearFilter module

ClearMap.ImageProcessing.Filter.DoGFilter module

ClearMap.ImageProcessing.Filter.Convolution module Convolve volumetric data with a 3d kernel, optimized for memory / float32 use

Based on scipy.signal routines.

Author

Original code from scipy.signal.

Modified by Chirstoph Kirst to optimize memory and sped and integration into ClearMap. The Rockefeller University, New York City, 2015

convolve (*x*, *k*, *mode='same'*)

Convolve array with kernel using float32 / complex64, optimized for memory consumption and speed

Parameters

- **x** (*array*) – data to be convolved
- **k** (*array*) – filter kernel

Returns *array* – convolution

ClearMap.ImageProcessing.Filter.FilterKernel module Implementation of various volumetric filter kernels

Filter Type Filter types defined by the `ftype` key include:

Type	Descrition
`mean`	uniform averaging filter
`gaussian`	Gaussian filter
`log`	Laplacian of Gaussian filter (LoG)
`dog`	Difference of Gaussians filter (DoG)
`sphere`	Sphere filter
`disk`	Disk filter

filterKernel (*ftype='Gaussian'*, *size=(5, 5)*, *sigma=None*, *radius=None*, *sigma2=None*)

Creates a filter kernel of a special type

Parameters

- **ftype** (*str*) – filter type, see *Filter Type*
- **size** (*array or tuple*) – size of the filter kernel
- **sigma** (*tuple or float*) – std for the first gaussian (if present)
- **radius** (*tuple or float*) – radius of the kernel (if applicable)
- **sigma2** (*tuple or float*) – std of a second gaussian (if present)

Returns *array* – structure element

filterKernel2D (*ftype='Gaussian'*, *size=(5, 5)*, *sigma=None*, *sigma2=None*, *radius=None*)

Creates a 2d filter kernel of a special type

Parameters

- **ftype** (*str*) – filter type, see *Filter Type*
- **size** (*array or tuple*) – size of the filter kernel
- **sigma** (*tuple or float*) – std for the first gaussian (if present)
- **radius** (*tuple or float*) – radius of the kernel (if applicable)

- **sigma2** (*tuple or float*) – std of a second gaussian (if present)

Returns *array* – structure element

filterKernel3D (*ftype='Gaussian'*, *size=(5, 5, 5)*, *sigma=None*, *sigma2=None*, *radius=None*)
Creates a 3d filter kernel of a special type

Parameters

- **ftype** (*str*) – filter type, see *Filter Type*
- **size** (*array or tuple*) – size of the filter kernel
- **sigma** (*tuple or float*) – std for the first gaussian (if present)
- **radius** (*tuple or float*) – radius of the kernel (if applicable)
- **sigma2** (*tuple or float*) – std of a second gaussian (if present)

Returns *array* – structure element

test ()
Test FilterKernel module

ClearMap.ImageProcessing.Filter.StructureElement module Routines to generate various structure elements

Structured elements defined by the `setype` key include:

Structure Element Types

Type	Descrition
`sphere`	Sphere structure
`disk`	Disk structure

Note: To be extended!

structureElement (*setype='Disk'*, *sesize=(3, 3)*)
Creates specific 2d and 3d structuring elements

Parameters

- **setype** (*str*) – structure element type, see *Structure Element Types*
- **sesize** (*array or tuple*) – size of the structure element

Returns *array* – structure element

structureElementOffsets (*sesize*)
Calculates offsets for a structural element given its size

Parameters **sesize** (*array or tuple*) – size of the structure element

Returns *array* – offsets to center taking care of even/odd ummber of elements

structureElement2D (*setype='Disk'*, *sesize=(3, 3)*)
Creates specific 2d structuring elements

Parameters

- **setype** (*str*) – structure element type, see *Structure Element Types*
- **sesize** (*array or tuple*) – size of the structure element

Returns *array* – structure element

structureElement3D (*setype='Disk'*, *sesize=(3, 3, 3)*)
Creates specific 3d structuring elements

Parameters

- **setype** (*str*) – structure element type, see *Structure Element Types*
- **sesize** (*array or tuple*) – size of the structure element

Returns *array* – structure element

ClearMap.ImageProcessing.IlluminationCorrection module

ClearMap.ImageProcessing.BackgroundRemoval module

ClearMap.ImageProcessing.GreyReconstruction module

ClearMap.ImageProcessing.SpotDetection module

ClearMap.ImageProcessing.MaximaDetection module

ClearMap.ImageProcessing.IlastikClassification module

ClearMap.ImageProcessing.ImageStatistics module

Functions to gather iamge statistics in large volumetric images

The main routines extract information from a large volumetric image, such as the maximum or mean.

calculateStatistics(*source*, *sink=None*, *calculateStatisticsParameter=None*, *method='Max'*, *remove=True*, *processMethod=<built-in function all>*, *verbose=False*, ***parameter*)

Calculate statisticsfrom image data

This is a main script to start extracting statistics of volumetric image data.

Parameters

- **source** (*str or array*) – Image source
- **sink** (*str or None*) – destination for the results
- **calculateStatisticsParameter** (*dict*) –

Name	Type	Descritption
method	(str or function)	function to extract statistic, must be trivially distributable if None, do not extract information
remove	(bool)	remove redundant overlap
verbose	(bool or int)	print / plot information about this step

- **method** (*str or function*)
- **processMethod** (*str or all*) – 'sequential' or 'parallel'. if all its choosen automatically
- **verbose** (*bool*) – print info
- ****parameter** (*dict*) – parameter for the image processing sub-routines

Returns list of statistics

calculateStatisticsOnStack(*img*, *calculateStatisticsParameter=None*, *method='Max'*, *remove=True*, *verbose=False*, *subStack=None*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Calculate a statistics from a large volumetric image

The statistics is assumed to be trivially distributable, i.e. max or mean.

Parameters

- **img** (*array*) – image data
- **calculateStatisticsParameter** (*dict*) –

Name	Type	Descritption
method	(str or function)	function to extract statistic, must be trivially distributable if None, do not extract information
remove	(bool)	remove redundant overlap
verbose	(bool or int)	print / plot information about this step

- **subStack** (*dict or None*) – sub-stack information
- **verbose** (*bool*) – print progress info
- **out** (*object*) – object to write progress info to

Returns *array or number* – extracted statistics

Note: One might need to choose zero overlap in the stacks to function properly!

---

joinStatistics(*results*, *calculateStatisticsParameter=None*, *method='Max'*, *subStacks=None*, ***parameter*)

Joins a list of calculated statistics

Parameters

- **results** (*list*) – list of statics results from the individual sub-processes
- **calculateStatisticsParameter** (*dict*) –

Name	Type	Descritption
method	(str or function)	function to extract statistic, must be trivially distributable if None, do not extract information

- **subStacks** (*list or None*) – list of all sub-stack information, see *Sub-Stacks*

Returns *list or object* – joined statistics

ClearMap.Analysis package

ClearMap analysis and statistics toolbox.

This part of ClearMap provides a toolbox for the statistical analysis and visualization of detected cells or structures and region specific analysis of annoated data.

For cleared mouse brains aligned to the Allen brain atlas a wide range of statistical analysis tools with respect to the anotated brain regions in the atlas is supported.

Key moduls are:

Module	Description
Voxelization	Voxelization of cells for visualization and analysis
Statistics	Statistical tools for the analysis of detected cells
Label	Tools to analysise data with espect to annotated refereneces

Subpackages

ClearMap.Analysis.Tools package Analysis and statistics tools not in standard python packages.

ClearMap.Analysis.Tools.Extrapolate module Method to extend interpolation objects to constantly / linearly extrapolate.

extrap1d(*x*, *y*, *interpolation='linear'*, *exterpolation='constant'*)
Interpolate on given values and extrapolate outside the given data

Parameters

- **x** (*numpy.array*) – x values of the data to interpolate
- **y** (*numpy.array*) – y values of the data to interpolate
- **interpolation** (*Optional[str]*) – interpolation method, see kind of scipy.interpolate.interp1d, default: "linear"
- **exterpolation** (*Optional[str]*) – interpolation method, either "linear" or "constant"

Returns *(function)* – inter- and extra-polation function

extrap1dFromInterp1d(*interpolator*, *exterpolation='constant'*)
Extend interpolation function to extrapolate outside the given data

Parameters

- **interpolator** (*function*) – interpolating function, see e.g. scipy.interpolate.interp1d
- **exterpolation** (*Optional[str]*) – interpolation method, either "linear" or "constant"

Returns *(function)* – inter- and extra-polation function

ClearMap.Analysis.Tools.MultipleComparisonCorrection module Correction methods for multiple comparison tests

correctPValues(*pvalues*, *method='BH'*)
Corrects p-values for multiple testing using various methods

Parameters

- **pvalues** (*array*) – list of p values to be corrected
- **method** (*Optional[str]*) – method to use: BH = FDR = Benjamini-Hochberg, B = FWER = Bonferoni

References

- Benjamini Hochberg, 1995
- Bonferoni correction
- R statistics package

Notes

•modified from http://statsmodels.sourceforge.net/ipdirective/generated/scikits.statsmodels.sandbox.stats.multicomp.multiple

estimateQValues(*pvalues*, *m=None*, *pi0=None*, *verbose=False*, *lowMemory=False*)
Estimates q-values from p-values

Parameters

- **pvalues** (*array*) – list of p-values
- **m** (*int or None*) – number of tests. If None, m = pvalues.size
- **pi0** (*float or None*) – estimate of m_0 / m which is the (true null / total tests) ratio, if None estimation via cubic spline.
- **verbose** (*bool*) – print info during execution
- **lowMemory** (*bool*) – if true use low memory version

Notes

•The q-value of a particular feature can be described as the expected proportion of false positives among all features as or more extreme than the observed one

•The estimated q-values are increasing in the same order as the p-values

References

•Storey and Tibshirani, 2003

•modified from https://github.com/nfusi/qvalue

ClearMap.Analysis.Tools.StatisticalTests module Some statistics tests not in standard python packages

testCramerVonMises2Sample(*x*, *y*)
Computes the Cramer von Mises two sample test.

This is a two-sided test for the null hypothesis that 2 independent samples are drawn from the same continuous distribution.

Parameters

- **x, y** (*sequence of 1-D ndarrays*) – two arrays of sample observations
- **assumed to be drawn from a continuous distribution, sample sizes**
- **can be different**

Returns *(float, float)* – T statistic, two-tailed p-value

References

•modified from https://github.com/scipy/scipy/pull/3659

ClearMap.Analysis.Label module

Label and annotation info from Allen Brain Atlas (v2)

Notes

- The annotation file is assumed to be in './Data/Annotation/annotation_25_right.tif' but can be set in the constant *DefaultLabeledImageFile*
- The mapping between labels and brain area information is found in the './Data/ARA2_annotation_info.csv' file. In the './Data/ARA2_annotation_info_collapse.csv' file a cross marks an area to which all sub-areas will be collapsed. The location of this file is set in *DefaultAnnotationFile*.
- For consistentcy cretain labels of the Allen brain atlas without annotation were assinged to their correct parent regions.
- A collapse collumn in the mapping file was added to allow for a region based collapse of statistics based on the inheritance structure of the annotated regions. These might need to be adjusted to the particular scientific question.

References

- Allen Brain Atlas

DefaultLabeledImageFile = '/Users/nicolasrenier/Documents/ClearMap/idisco/ClearMap/Test/Data/Annotation/annotati
str: default volumetric annotated image file

This file is by default the Allen brain annotated mouse atlas with 25um isotropic reslution.

DefaultAnnotationFile = '/Users/nicolasrenier/Documents/ClearMap/idisco/ClearMap/Data/ARA2_annotation_info_coll
str: default list of labels in the annotatd image and names of annotated regoins

This file is by default the lables for the Allen brain annotated mouse atlas with 25um isotropic reslution.

An extra column for collapse indicats how to automatically collapse data into the different bain regions if the `collapse` option is given.

class LabelRecord(*id*, *name*, *acronym*, *color*, *parent*, *collapse*)
Bases: `tuple`

Structure of a label for a annotated region

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

__getstate__()
Exclude the OrderedDict from pickling

__repr__()
Return a nicely formatted representation string

acronym
Alias for field number 2

collapse
Alias for field number 5

color
Alias for field number 3

id
Alias for field number 0

name
Alias for field number 1

parent
Alias for field number 4

class LabelInfo (*slf*, *annotationFile='/Users/nicolasrenier/Documents/ClearMap/idisco/ClearMap/Data/ARA2_annotation_info_co*
Bases: `object`

Class that holds infomration of the annotated regions

ids = None

names = None

acronyms = None

colors = None

parents = None

levels = None

collapse = None

collapseMap = None

initialize (*slf*, *annotationFile='/Users/nicolasrenier/Documents/ClearMap/idisco/ClearMap/Data/ARA2_annotation_info_*

name (*slf*, *iid*)

acronym (*slf*, *iid*)

color (*slf*, *iid*)

parent (*slf*, *iid*)

level (*slf*, *iid*)

toLabelAtLevel (*slf*, *iid*, *level*)

toLabelAtCollapseMap (*slf*, *iid*)

toLabelAtCollapse (*slf*, *iid*)

Label = <ClearMap.Analysis.Label.LabelInfo object>
Information on the annotated regions

initialize (*annotationFile='/Users/nicolasrenier/Documents/ClearMap/idisco/ClearMap/Data/ARA2_annotation_info_collapse.c*

labelAtLevel (*label*, *level*)

labelAtCollapse (*label*)

labelPoints (*points*, *labeledImage='/Users/nicolasrenier/Documents/ClearMap/idisco/ClearMap/Test/Data/Annotation/annotation*
level=None, *collapse=None*)

countPointsInRegions (*points*, *labeledImage='/Users/nicolasrenier/Documents/ClearMap/idisco/ClearMap/Test/Data/Annotati*
intensities=None, *intensityRow=0*, *level=None*, *allIds=False*, *sort=True*, *returnIds=True*, *returnCounts=False*, *collapse=None*)

labelToName (*label*)

labelToAcronym (*label*)

labelToColor (*label*)

writePAL (*filename*, *cols*)

writeLUT (*filename*, *cols*)

makeColorPalette (*filename=None*)
Creates a pal file for imaris based on label colors

makeColorAnnotations (*filename*, *labeledImage=None*)

test ()
Test Label module

ClearMap.Analysis.Statistics module

Create some statistics to test significant changes in voxelized and labeled data

TODO: cleanup / make generic

readDataGroup (*filenames*, *combine=True*, ***args*)
Turn a list of filenames for data into a numpy stack

readPointsGroup (*filenames*, ***args*)
Turn a list of filenames for points into a numpy stack

tTestVoxelization (*group1*, *group2*, *signed=False*, *removeNaN=True*, *pcutoff=None*)
t-Test on differences between the individual voxels in group1 and group2, group is a array of voxelizations

cutoffPValues (*pvals*, *pcutoff=0.05*)

colorPValues (*pvals, psign, positive=[1, 0], negative=[0, 1], pcutoff=None, positivetrend=[0, 0, 1, 0], negativetrend=[0, 0, 0, 1], pmax=None*)

mean (*group*, ***args*)

std (*group*, ***args*)

var (*group*, ***args*)

thresholdPoints (*points*, *intensities*, *threshold=0*, *row=0*)
Threshold points by intensities

weightsFromPrecentiles (*intensities, percentiles=[25, 50, 75, 100]*)

countPointsGroupInRegions (*pointGroup*, *labeledImage='/Users/nicolasrenier/Documents/ClearMap/idisco/ClearMap/Test/D*
intensityGroup=None, *intensityRow=0*, *returnIds=True*, *returnCounts=False*, *collapse=None*)
Generates a table of counts for the various point datasets in pointGroup

tTestPointsInRegions (*pointCounts1*, *pointCounts2*, *labeledImage='/Users/nicolasrenier/Documents/ClearMa*
signed=False, *removeNaN=True*, *pcutoff=None*, *equal_var=False*)
t-Test on differences in counts of points in labeled regions

testCompletedCumulatives (*data*, *method='AndersonDarling'*, *offset=None*, *plot=False*)
Test if data sets have the same number / intensity distribution by adding max intensity counts to the smaller sized data sets and performing a distribution comparison test

testCompletedInvertedCumulatives (*data*, *method='AndersonDarling'*, *offset=None*, *plot=False*)
Test if data sets have the same number / intensity distribution by adding zero intensity counts to the smaller sized data sets and performing a distribution comparison test on the reversed cumulative distribution

testCompletedCumulativesInSpheres(*points1*, *intensities1*, *points2*, *intensities2*, *dataSize='/Users/nicolasrenier/Documents/ClearMap/idisco/ClearMap/Test/Data/Annot*, *radius=100*, *method='AndresonDarling'*)
Performs completed cumulative distribution tests for each pixel using points in a ball centered at that cooridnates, returns 4 arrays p value, statistic value, number in each group

test()
Test the statistics array

ClearMap.Analysis.Voxelization module

ClearMap.Visualization package

This sub-package provides tools for the visualization of the alignment and analysis results

Author

Christoph Kirst, The Rockefeller University, New York City, 2015

ClearMap.Visualization.Plot module

ClearMap.Parameter module

ClearMap default parameter module.

This module defines default parameter used by various sub-packages.

See also:

Settings

detectCellParameter = {'findExtendedMaximaParameter': {'threshold': 0, 'save': None, 'verbose': False, 'hMax': 20, 'si
dict: Paramters for cell detection using the spot detection algorithm

See also:

IlastikParameter, StackProcessingParameter

IlastikParameter = {'rescale': None, 'backgroundSize': (15, 15), 'classifier': '/Test/Ilastik/classifier.h5'}
dict: Paramters for cell detection using Ilastik classification

- "classifier": ilastic classifier to use
- "rescale": rescale images before classification
- "backgroundSize": Background correctoin: None or (y,x) which is size of disk for gray scale opening

See also:

SpotDetectionParameter, StackProcessingParameter

processStackParameter = {'chunkOptimizationSize': <built-in function all>, 'processes': 2, 'chunkSizeMin': 30, 'chunkO

dict: Parameter for processing an image stack in parallel

- "processes": max number of parallel processes
- "chunkSizeMax" : maximal chunk size in z
- "chunkSizeMin" : minimal chunk size in z,
- "chunkOverlap" : overlap between two chunks,

- "chunkOptimization": optimize chunk size and number to number of processes
- "chunkOptimizationSize": increase chunk size for optimizaition (True, False or all = automatic)

See also:

`SpotDetectionParameter`, *`IlastikParameter`*

`AlignmentParameter` = {'fixedImageMask': None, 'alignmentDirectory': None, 'movingImage': '/Test/Data/Elastix/150524
dict: Parameter for Elastix alignment

- "alignmentDirectory" : directory to save the alignment result
- "movingImage": image to be aligned
- "fixedImage": reference image
- "affineParameterFile": elastix parameter files for affine alignment
- "bSplineParameterFile" : elastix parameter files for non-linear alignment

See also:

`Elastix`

`ResamplingParameter` = {'orientation': None, 'source': None, 'resolutionSink': (25, 25, 25), 'sink': None, 'resolutionSource
dict: Parameter for resampling data

- "source" : data source file
- "sink" : data output file

"resolutionSource": resolution of the raw data (in um / pixel) as (x,y,z)

"resolutionSink" : resolution of the reference / atlas image (in um/ pixel) as (x,y,z)

"orientation" [Orientation of the data set wrt reference as (x=1,y=2,z=3)] (-axis will invert the orientation, for other hemisphere use (-1, 2, 3), to exchnge x,y use (2,1,3) etc)

See also:

`Resampling`

`VoxelizationParameter` = {'method': 'Spherical', 'voxelizationSize': (1, 1, 1)}
dict: Parameter to calculate density voxelization

- "method": Method to voxelize: 'Spherical','Rectangular, 'Gaussian'
- "voxelizationSize": max size of the volume to be voxelized

See also:

`voxelization`

ClearMap.Settings module

Module to set *ClearMap's* internal parameter and paths to external programs.

Notes

Edit the *`setup()`* routine to point to the ilastik and elastix paths for specific hosts

See also:

- *`IlastikPath`*

- *ElastixPath*
- *Parameter*

IlastikPath = None
str: Absolute path to the Ilastik 0.5 installation

Notes

Ilastik Webpage

Ilastik 0.5 Download

ElastixPath = None
str: Absolue path to the elastix installation

Notes

Elastix Webpage

setup()
Setup ClearMap for specific hosts

Notes

Edit this routine to include special setttings for specific hosts

See also:

IlastikPath, *ElastixPath*

clearMapPath()
Returns root path to the ClearMap software

Returns *str* – root path to ClearMap

ClearMapPath = '/Users/nicolasrenier/Documents/ClearMap/idisco/ClearMap'
str: Absolute path to the ClearMap root folder

ClearMap.Utils package

This sub-package provides utility functions used throughout the package

Author

Christoph Kirst, The Rockefeller University, New York City, 2015

ClearMap.Utils.ParameterTools module

ParameterTools

Provides simple formatting tools to handle / print parameter dictionaries organized as key:value pairs.

Author

Christoph Kirst, The Rockefeller University, New York City, 2015

getParameter(*parameter*, *key*, *default=None*)
Gets a parameter from a dict, returns default value if not defined

Parameters

- **parameter** (*dict*) – parameter dictionary
- **key** (*object*) – key
- **default** (*object*) – deault return value if parameter not defined

Returns *object* – parameter value for key

writeParameter(*head=None*, *out=None*, ***args*)
Writes parameter settings in a formatted way

Parameters

- **head** (*str or None*) – prefix of each line
- **out** (*object or None*) – write to a specific output, if None return string
- ****args** – the parameter values as key=value arguments

Returns *str or None* – a formated string with parameter info

joinParameter(**args*)
Joins dictionaries in a consitent way

For multiple occurences of a key the value is defined by the first key : value pair.

Parameters ***args** – list of parameter dictonaries

Returns *dict* – the joined dictionary

ClearMap.Utils.ProcessWriter module

Provides simple formatting tools to print text with parallel process header

Author

Christoph Kirst, The Rockefeller University, New York City, 2015

class **ProcessWriter**(*process=0*)
Bases: `object`

Class to handle writing from parallel processes

process
int

the process number

writeString(*text*)
Generate string with process prefix

Parameters **text** (*str*) – the text input

Returns *str* – text with [process prefix

write(*text*)
Write string with process prefix to sys.stdout

Parameters **text** (*str*) – the text input

ClearMap.Utils.Timer module

Provides tools for timing

Author

Christoph Kirst, The Rockefeller University, New York City, 2015

class **Timer**(*verbose=False*)
Bases: object

Class to stop time and print results in formatted way

time
float

the time since the timer was started

start()
Start the timer

reset()
Reset the timer

elapsedTime(*head=None*, *asstring=True*)
Calculate elapsed time and return as formated string

Parameters

- **head** (*str or None*) – prefix to the string
- **asstring** (*bool*) – return as string or float

Returns *str or float* – elapsed time

printElapsedTime(*head=None*)
Print elapsed time as formated string

Parameters **head** (*str or None*) – prefix to the string

formatElapsedTime(*t*)
Format time to string

Parameters **t** (*float*) – time in seconds prefix

Returns *str* – time as hours:minutes:seconds

CHAPTER

THREE

INDICES

- genindex
- modindex
- search

CHAPTER

FOUR

AUTHOR AND LICENSE

4.1 ClearMap lead programming and design:

Christoph Kirst, *The Rockefeller University*

4.2 Scripts and specific applications:

Nicolas Renier and Christoph Kirst *The Rockefeller University*

4.3 Documentation:

Christoph Kirst and Nicolas Renier *The Rockefeller University*

4.4 Additional programming and consulting:

Kannan Umadevi Venkataraju *Cold Spring Harbor Laboratories*

4.5 License

GNU GENERAL PUBLIC LICENSE Version 3

See `LICENSE` or gnu.org for details.

[Erturk2012] Three-dimensional imaging of solvent-cleared organs using 3DISCO.iDISCO: A Simple, Rapid Method to Immunolabel Large Tissue Samples, A. Erturk, et al. Nat. Protocol 2012

[Renier2014] iDISCO: A Simple, Rapid Method to Immunolabel Large Tissue Samples for Volume Imaging, N. Renier, et al. Cell 2014

[Renier2015] ‘Mapping brain activity in the mouse at cellular resolution with volume imaging using immediate early genes, N. Renier, et al. in prep.

[iDISCO] **‘iDISCO webpage, http://idisco.info/ < http://idisco.info/>‘_**

[ABA] Allen Brain Atlas, http://www.brain-map.org/.

[Elastix] Elastix toolbox for rigid and nonrigid registration of images, http://elastix.isi.uu.nl

[Ilastik] Ilastik the interactive learning and segmentation toolkit, http://ilastik.org/

[ImageJ] ImageJ

C

Symbols

A

C

P

R

S

T

V

W