# ClearMap Documentation

***Release 0.9.2***

**Christoph Kirst**

December 18, 2015

*ClearMap* is a 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.

CHAPTER

ONE

# CONTENTS

## 1.1 Introduction

*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.

The package consists of 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 focus on volumetric data:

| Format | Descrition |
|---|---|
| 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 |

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]* and c to a possible color channel.

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

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

### 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 focus on processing large 3d volumetric images in parallel.

Main processing modules include:

| Module | Descrition |
|---|---|
| *BackgroundRemoval* | Background estimation and removal via morphological opening |
| *IlluminationCorrection* | Correction of vignetting and other illumination errors |
| *Filter* | Filtering of the imagee 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 felxible integration of additional modules.

### 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 annoated data.

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

Key moduls are

| Module | Descrition |
|---|---|
| *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 |

### 1.1.5 Examples

Examples can be found `here` and in the api documentation.

### 1.1.6 iDISCO+ Clearing Method

iDISCO stands for immunolabeling-enabled three-dimensional imaging of solvent-cleared organs. The iDISCO+ clearance method is decribed in *[Renier2014]* and *[Renier2015]*.

iDISCO is modeled on classical histology techniques, facilitating translation of section staining assays to intact tissues, as evidenced by compatibility with 28 antibodies to both endogenous antigens and transgenic reporters like GFP. iDISCO enables facile volume imaging of immunolabeled structures in complex tissues

**See these videos**

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

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

### 1.1.7 References

## 1.2 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

### 1.2.1 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 | Description | 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

Interface to Imaris Files

Module to read data and write points to Imaris files.

**Note:** To write points without errors make sure the original file has at least one spot object!

#### Example

```
>>> import os, numpy
>>> import ClearMap.IO.Imaris as ims
>>> from ClearMap.Settings import ClearMapPath
>>> filename = os.path.join(ClearMapPath,'Test/Data/Imaris/test for spots added spot.ims')
>>> ims.dataSize(filename);
(256, 320, 256)
```

**Todo:**

- Fix writing new spots to imaris file
- Get settings to directly render points as 'pixel' and not as spheres

Author

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

**openFile** (*filename*, *mode='a'*)
Open Imaris file as hdf5 object

**Parameters**

- **filename** (*str*) – file name
- **mode** (*str*) – argument to h5py.File

**Returns** *object* – h5py object

**closeFile** (*h5file*)
Close Imaris hdf5 file object

**Parameters** **h5file** (*object*) – h5py opject

**Returns** *bool* – success

**readDataSet** (*h5file*, *resolution=0*, *channel=0*, *timepoint=0*)
Open Imaris file and returns hdf5 image data

**Parameters**

- **h5file** (*object*) – h5py object
- **resolution** (*int*) – resolution level
- **channel** (*int*) – color channel
- **timepoint** (*int*) – time point

**Returns** *array* – image data

**dataSize** (*filename*, *resolution=0*, *channel=0*, *timepoint=0*, ***args*)
Read data size of the imaris image data set

**Parameters**

- **filename** (*str*) – imaris file name
- **resolution** (*int*) – resolution level
- **channel** (*int*) – color channel
- **timepoint** (*int*) – time point

**Returns** *tuple* – image data size

**dataZSize** (*filename*, ***args*)
Read z data size of the imaris image data set

**Parameters**

- **filename** (*str*) – imaris file name
- **resolution** (*int*) – resolution level
- **channel** (*int*) – color channel
- **timepoint** (*int*) – time point

**Returns** *int* – image z data size

**readData** (*filename*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*, *resolution=0*, *channel=0*, *timepoint=0*, ***args*)
Read data from imaris file

**Parameters**

- **filename** (*str*) – file name as regular expression
- **x,y,z** (*tuple*) – data range specifications
- **resolution** (*int*) – resolution level
- **channel** (*int*) – color channel
- **timepoint** (*int*) – time point

**Returns** *array* – image data

**getDataSize** (*h5file*)
Get the full data size in pixel from h5py imaris object

**Parameters** **h5file** (*object*) – h5py object

**Returns** *tuple* – image data size

**getDataExtent** (*h5file*)
Get the spatial extent of data from h5py imaris object

**Parameters** **h5file** (*object*) – h5py object

**Returns** *array* – spatial extend of image

**getScaleAndOffset** (*h5file*)
Determine scale and offset to transform pixel to spatial coordinates as used by imaris

**Parameters** **h5file** (*object*) – h5py object

**Returns** *tuple* – image scale (length / pixel) and offset (from origin)

**transformPointsToImaris** (*points*, *scale=(4.0625*, *4.0625*, *3)*, *offset=(0*, *0*, *0)*)
Transform pixel coordinates of cell centers to work in Imaris

**Parameters**

- **points** (*array*) – point coordinate array

- **scale** (*tuple*) – spatial scale of the image data
- **offset** (*tuple*) – spatial offset of the image data

**Returns** *array* – scaled points

**writePoints**(*filename*, *points*, *mode='o'*, *radius=0.5*, *scale=<built-in function all>*, *offset=None*)
Write points to Imaris file

**Parameters**

- **filename** (*str*) – imaris file name
- **points** (*array*) – point data
- **mode** (*str*) – 'o'= override, 'a'=add
- **radius** (*float*) – size of each point
- **scale** (*tuple or all*) – spatial scaling of points
- **offset** (*tuple or None*) – spatial offset of points

**Returns** *str* – file name of imaris file

**Note:** This routine is still experimental !

**writeData**(*filename*, ***args*)
Write image data to imaris file

**Note:** Not implemented yet !

**readPoints**(*filename*)
Read points from imaris file

**Note:** Not implemented yet !

**copyData**(*source*, *sink*)
Copy a imaris file from source to sink

**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 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

Simple Interface to read RAW/MHD files e.g. created by elastix

Todo: read subsets efficiently

#### Example

```
>>> import os, numpy
>>> from ClearMap.Settings import ClearMapPath
>>> import ClearMap.IO.RAW as raw
>>> filename = os.path.join(ClearMapPath, 'Test/Data/Raw/test.mhd')
>>> raw.dataSize(filename);
(20, 50, 10)
```

Author

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

**dataSize**(*filename*, ***args*)
Read data size from raw/mhd image

**Parameters**

- **filename** (*str*) – imaris file name
- **x,y,z** (*tuple or all*) – range specifications

**Returns** *int* – raw image data size

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

**Parameters**

- **filename** (*str*) – imaris file name
- **z** (*tuple or all*) – range specification

**Returns** *int* – raw image z data size

**readData**(*filename*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*)
Read data from raw/mhd image

**Parameters**

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

**Returns** *array* – image data

**writeHeader** (*filename*, *meta_dict*)
Write raw header mhd file

**Parameters**

- **filename** (*str*) – file name of header
- **meta_dict** (*dict*) – dictionary of meta data

**Returns** *str* – header file name

**writeRawData** (*filename*, *data*)
Write the data into a raw format file.

**Parameters**

- **filename** (*str*) – file name as regular expression
- **data** (*array*) – data to write to raw file

**Returns** *str* – file name of raw file

**writeData** (*filename*, *data*, ***args*)
Write data into to raw/mhd file pair

**Parameters**

- **filename** (*str*) – file name as regular expression
- **data** (*array*) – data to write to raw file

**Returns** *str* – file name of mhd file

**copyData** (*source*, *sink*)
Copy a raw/mhd file pair from source to sink

**Parameters**

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

**Returns** *str* – file name of the copy

**test** ()
Test RAW io module

### ClearMap.IO.TIF module

Interface to tif image files and stacks.

The interface makes use of the tifffile library.

### Example

```
>>> import os, numpy
>>> import ClearMap.IO.TIF as tif
>>> from ClearMap.Settings import ClearMapPath
>>> filename = os.path.join(ClearMapPath,'Test/Data/Tif/composite.tif')
>>> data = tif.readData(filename);
>>> print data.shape
(50, 100, 30, 4)
```

Author

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

**dataSize** (*filename*, ***args*)
Returns size of data in tif file

**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 z size of data in tif file

**Parameters**

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

**Returns** *int* – z data size

**readData** (*filename*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*, ***args*)
Read data from a single tif image or stack

**Parameters**

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

**Returns** *array* – image data

**writeData** (*filename*, *data*)
Write image data to tif file

**Parameters**

- **filename** (*str*) – file name
- **data** (*array*) – image data

**Returns** *str* – tif file name

**copyData** (*source*, *sink*)
Copy a data 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

**test** ()
Test 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

Interface to Elastix for alignment of volumetric data

The elastix documentation can be found here.

In essence, a transformation $T(x)$ is sought so that for a fixed image $F(x)$ and a moving image $M(x)$:

$$F(x) = M(T(x))$$

Once the map $T$ is estimated via elastix, transformix maps an image $I(x)$ from the moving image frame to the fixed image frame, i.e.:

$$I(x) \to I(T(x))$$

To register an image onto a reference image, the fixed image is typically choosed to be the image to be registered, while the moving image is the reference image. In this way an object identified in the data at position x is mapped via transformix as:

$$x \to T(x)$$

**Summary**

- elastix finds a transformation $T : \mathrm{fixedimage} \to \mathrm{movingimage}$
- the fixed image is image to be registered
- the moving image is typically the reference image
- the result folder may contain an image (mhd file) that is $T^{-1}(\mathrm{moving})$, i.e. has the size of the fixed image
- transformix applied to data gives $T^{-1}(\mathrm{data})$ !
- transformix applied to points gives $T(\mathrm{points})$ !
- point arrays are assumed to be in (x,y,z) coordinates consistent with (x,y,z) array represenation of images in ClearMap

Main routines are: *alignData()*, *transformData()* and *transformPoints()*.

**See also:**

Elastix documentation *Resampling*

**ElastixBinary = '/home/ckirst/programs/elastix/bin/elastix'**
str: the elastix executable

**Notes**

•setup in *initializeElastix()*

**ElastixLib = '/home/ckirst/programs/elastix/lib'**
str: path to the elastix library

**Notes**

•setup in *initializeElastix()*

**TransformixBinary = '/home/ckirst/programs/elastix/bin/transformix'**
str: the transformix executable

**Notes**

•setup in *initializeElastix()*

**Initialized = True**
bool: True if the elastixs binarys and paths are setup

#### Notes

•setup in *initializeElastix()*

**printSettings**()
Prints the current elastix configuration

**See also:**

*ElastixBinary*, *ElastixLib*, *TransformixBinary*, *Initialized*

**setElastixLibraryPath**(*path=None*)
Add elastix library path to the LD_LIBRARY_PATH variable in linux

**Parameters** **path** (*str or None*) – path to elastix root directory if None *ClearMap.Settings.ElastixPath* is used.

**initializeElastix**(*path=None*)
Initialize all paths and binaries of elastix

**Parameters**

- **path** (*str or None*) – path to elastix root directory, if None
- **:const:'ClearMap.Settings.ElastixPath' is used.**

**See also:**

*ElastixBinary*, *ElastixLib*, *TransformixBinary*, *Initialized*, *setElastixLibraryPath()*

**checkElastixInitialized**()
Checks if elastix is initialized

**Returns** *bool* – True if elastix paths are set.

**getTransformParameterFile**(*resultdir*)
Finds and returns the transformation parameter file generated by elastix

#### Notes

In case of multiple transformation parameter files the top level file is returned

**Parameters** **resultdir** (*str*) – path to directory of elastix results

**Returns** *str* – file name of the first transformation parameter file

**setPathTransformParameterFiles**(*resultdir*)
Replaces relative with abolsute path in the parameter files in the result directory

#### Notes

When elastix is not run in the directory of the transformation files the aboslute path needs to be given in each transformation file to point to the subsequent transformation files. This is done via this routine

**Parameters** **resultdir** (*str*) – path to directory of elastix results

**parseElastixOutputPoints**(*filename*, *indices=True*)
Parses the output points from the output file of transformix

**Parameters**

- **filename** (*str*) – file name of the transformix output file
- **indices** (*bool*) – if True return pixel indices otherwise float coordinates

**Returns** **points** (*array*) – the transformed coordinates

**getTransformFileSizeAndSpacing** (*transformfile*)
Parse the image size and spacing from a transformation parameter file

**Parameters** **transformfile** (*str*) – File name of the transformix parameter file.

**Returns** *(float, float)* – the image size and spacing

**getResultDataFile** (*resultdir*)
Returns the mhd result file in a result directory

**Parameters** **resultdir** (*str*) – Path to elastix result directory.

**Returns** *str* – The mhd file in the result directory.

**setTransformFileSizeAndSpacing** (*transformfile*, *size*, *spacing*)
Replaces size and scale in the transformation parameter file

**Parameters**

- **transformfile** (*str*) – transformation parameter file
- **size** (*tuple*) – the new image size
- **spacing** (*tuplr*) – the new image spacing

**rescaleSizeAndSpacing** (*size*, *spacing*, *scale*)
Rescales the size and spacing

**Parameters**

- **size** (*tuple*) – image size
- **spacing** (*tuple*) – image spacing
- **scale** (*tuple*) – the scale factor

**Returns** *(tuple, tuple)* – new size and spacing

**alignData** (*fixedImage*, *movingImage*, *affineParameterFile*, *bSplineParameterFile=None*, *resultDirectory=None*)
Align images using elastix, estimates a transformation $T$ : fixed image $\rightarrow$ moving image.

**Parameters**

- **fixedImage** (*str*) – image source of the fixed image (typically the reference image)
- **movingImage** (*str*) – image source of the moving image (typically the image to be registered)
- **affineParameterFile** (*str or None*) – elastix parameter file for the primary affine transformation
- **bSplineParameterFile** (*str or None*) – elastix parameter file for the secondary non-linear transformation
- **resultDirectory** (*str or None*) – elastic result directory

**Returns** *str* – path to elastix result directory

**transformData** (*source*, *sink=[]*, *transformParameterFile=None*, *transformDirectory=None*, *resultDirectory=None*)
Transform a raw data set to reference using the elastix alignment results

If the map determined by elastix is $T\text{fixed} \rightarrow \text{moving}$, transformix on data works as :math:'T^{-1}(mathrm{data})

**Parameters**

- **source** (*str or array*) – image source to be transformed
- **sink** (*str, [] or None*) – image sink to save transformed image to. if [] return the default name of the data file generated by transformix.
- **transformParameterFile** (*str or None*) – parameter file for the primary transformation, if None, the file is determined from the transformDirectory.
- **transformDirectory** (*str or None*) – result directory of elastix alignment, if None the transformParameterFile has to be given.
- **resultDirectory** (*str or None*) – the directorty for the transformix results

**Returns** *array or str* – array or file name of the transformed data

**deformationField**(*sink=[]*, *transformParameterFile=None*, *transformDirectory=None*, *resultDirectory=None*)

Create the deformation field T(x) - x

The map determined by elastix is $T\text{fixed} \rightarrow \text{moving}$

**Parameters**

- **sink** (*str, [] or None*) – image sink to save the transformation field; if [] return the default name of the data file generated by transformix.
- **transformParameterFile** (*str or None*) – parameter file for the primary transformation, if None, the file is determined from the transformDirectory.
- **transformDirectory** (*str or None*) – result directory of elastix alignment, if None the transformParameterFile has to be given.
- **resultDirectory** (*str or None*) – the directorty for the transformix results

**Returns** *array or str* – array or file name of the transformed data

**deformationDistance**(*deformationField*, *sink=None*, *scale=None*)

Compute the distance field from a deformation vector field

**Parameters**

- **deformationField** (*str or array*) – source of the deformation field determined by `deformationField()`
- **sink** (*str or None*) – image sink to save the deformation field to
- **scale** (*tuple or None*) – scale factor for each dimension, if None = (1,1,1)

**Returns** *array or str* – array or file name of the transformed data

**writePoints**(*filename*, *points*, *indices=True*)

Write points as elastix/transformix point file

**Parameters**

- **filename** (*str*) – file name of the elastix point file.
- **points** (*array or str*) – source of the points.
- **indices** (*bool*) – write as pixel indices or physical coordiantes

**Returns** *str* – file name of the elastix point file

**transformPoints**(*source*, *sink=None*, *transformParameterFile=None*, *transformDirectory=None*, *indices=True*, *resultDirectory=None*, *tmpFile=None*)

Transform coordinates math:*x* via elastix estimated transformation to $T(x)$

Note the transformation is from the fixed image coorindates to the moving image coordiantes.

**Parameters**

- **source** (*str*) – source of the points
- **sink** (*str or None*) – sink for transformed points
- **transformParameterFile** (*str or None*) – parameter file for the primary transformation, if None, the file is determined from the transformDirectory.
- **transformDirectory** (*str or None*) – result directory of elastix alignment, if None the transformParameterFile has to be given.
- **indices** (*bool*) – if True use points as pixel coordinates otherwise spatial coordinates.
- **resultDirectory** (*str or None*) – elastic result directory
- **tmpFile** (*str or None*) – file name for the elastix point file.

**Returns** *array or str* – array or file name of transformed points

**test**()

Test Elastix module

#### ClearMap.Alignment.Resampling module

The *Resampling* module provides methods to resample and reorient volumetric and point data.

Resampling the data is usually necessary as the first step to match the resolution and orientation of the reference object.

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

**Image Representation and Size**

**The module assumes that images in arrays are arranged as**

- [x,y] or
- [x,y,z]

where x,y,z correspond to the x,y,z coordinates as displayed in e.g. ImageJ. For example an image of size (512,512) stored in an array `img` will have:

```
>>> img.shape
(512,512)
```

Points are assumed to be given as x,y,z coordinates

Parameters such as *resolution* or *dataSize* are assumed to be given in (x,y) or (x,y,z) format, e.g.

```
>>> dataSize = (512,512)
```

**Orientation** The *orientation* parameter is a tuple of d numbers from 1 to d that specifies the permutation of the axes, a minus sign infront of a numbeer indicates inversion of that axes. For exmaple

```
>>> orientation=(2,-1)
```

indicates that x and y should be exchanged and the new y axes should be reversed.

Generally a re-orientation is composed of first a permutation of the axes and then inverting the indicated axes.

A *permutation* is an orientation without signs and with numbers from 0 to d-1.

### Examples

```
>>> import os
>>> import ClearMap.IO as io
>>> from ClearMap.Settings import ClearMapPath
>>> from ClearMap.Alignment.Resampling import resampleData
>>> filename = os.path.join(ClearMapPath,'Test/Data/OME/16-17-27_0_8X-s3-20HF_UltraII_C00_xyz-Table 2
>>> print io.dataSize(filename)
(2160, 2560, 21)
>>> data = resampleData(filename, sink = None, resolutionSource = (1,1,1), orientation = (1,2,3), res
>>> print data.shape
(216, 256, 10)
```

**fixOrientation**(*orientation*)

Convert orientation to standard format number sequence

**Parameters** **orientation** (*tuple or str*) – orientation specification

**Returns** *tuple* – orientation sequence

**See also:**

*Orientation*

**inverseOrientation**(*orientation*)

Returns the inverse permuation of the permutation orientation taking axis inversions into account.

**Parameters** **orientation** (*tuple or str*) – orientation specification

**Returns** *tuple* – orientation sequence

**See also:**

*Orientation*

**orientationToPermuation**(*orientation*)

Extracts the permuation from an orientation.

**Parameters** **orientation** (*tuple or str*) – orientation specification

**Returns** *tuple* – premutation sequence

**See also:**

*Orientation*

**orientResolution**(*resolution*, *orientation*)

Permutes a resolution tuple according to the given orientation.

**Parameters**

- **resolution** (*tuple*) – resolution specification
- **orientation** (*tuple or str*) – orientation specification

**Returns** *tuple* – oriented resolution sequence

**See also:**

*Orientation*

**orientResolutionInverse**(*resolution*, *orientation*)
Permutes a resolution tuple according to the inverse of a given orientation.

**Parameters**

- **resolution** (*tuple*) – resolution specification
- **orientation** (*tuple or str*) – orientation specification

**Returns** *tuple* – oriented resolution sequence

**See also:**

*Orientation*

**orientDataSize**(*dataSize*, *orientation*)
Permutes a data size tuple according to the given orientation.

**Parameters**

- **dataSize** (*tuple*) – resolution specification
- **orientation** (*tuple or str*) – orientation specification

**Returns** *tuple* – oriented dataSize sequence

**See also:**

*Orientation*

**orientDataSizeInverse**(*dataSize*, *orientation*)
Permutes a dataSize tuple according to the inverse of a given orientation.

**Parameters**

- **dataSize** (*tuple*) – dataSize specification
- **orientation** (*tuple or str*) – orientation specification

**Returns** *tuple* – oriented dataSize sequence

**See also:**

*Orientation*

**resampleDataSize**(*dataSizeSource*, *dataSizeSink=None*, *resolutionSource=None*, *resolutionSink=None*, *orientation=None*)
Calculate scaling factors and data sizes for resampling.

**Parameters**

- **dataSizeSource** (*tuple*) – data size of the original image
- **dataSizeSink** (*tuple or None*) – data size of the resmapled image
- **resolutionSource** (*tuple or None*) – resolution of the source image
- **resolutionSink** (*tuple or None*) – resolution of the sink image
- **orientation** (*tuple or str*) – re-orientation specification

**Returns** *tuple* – data size of the source tuple: data size of the sink tuple: resolution of source tuple: resolution of sink

**See also:**

*Orientation*

**fixInterpolation**(*interpolation*)
Converts interpolation given as string to cv2 interpolation object

**Parameters** **interpolation** (*str or object*) – interpolation string or cv2 object

**Returns** *object* – cv2 interpolation type

**resampleXY**(*source*, *dataSizeSink*, *sink=None*, *interpolation='linear'*, *out=<open file '<stdout>'*, *mode 'w'>*, *verbose=True*)
Resample a 2d image slice

This routine is used for resampling a large stack in parallel in xy or xz direction.

**Parameters**

- **source** (*str or array*) – 2d image source
- **dataSizeSink** (*tuple*) – size of the resmapled image
- **sink** (*str or None*) – location for the resmapled image
- **interpolation** (*str*) – interpolation method to use: 'linear' or None (nearest pixel)
- **out** (*stdout*) – where to write progress information
- **vebose** (*bool*) – write progress info if true

**Returns** *array or str* – resampled data or file name

**resampleData**(*source*, *sink=None*, *orientation=None*, *dataSizeSink=None*, *resolutionSource=(4.0625*, *4.0625*, *3)*, *resolutionSink=(25*, *25*, *25)*, *processingDirectory=None*, *processes=1*, *cleanup=True*, *verbose=True*, *interpolation='linear'*, ***args*)
Resample data of source in resolution and orientation

**Parameters**

- **source** (*str or array*) – image to be resampled
- **sink** (*str or None*) – destination of resampled image
- **orientation** (*tuple*) – orientation specified by permuation and change in sign of (1,2,3)
- **dataSizeSink** (*tuple or None*) – target size of the resampled image
- **resolutionSource** (*tuple*) – resolution of the source image (in length per pixel)
- **resolutionSink** (*tuple*) – resolution of the resampled image (in length per pixel)
- **processingDirectory** (*str or None*) – directory in which to perform resmapling in parallel, None a temporary directry will be created
- **processes** (*int*) – number of processes to use for parallel resampling
- **cleanup** (*bool*) – remove temporary files
- **verbose** (*bool*) – display progress information
- **interpolation** (*str*) – method to use for interpolating to the resmapled image

**Returns** *(array or str)* – data or file name of resampled image

#### Notes

- resolutions are assumed to be given for the axes of the intrinsic orientation of the data and reference as when viewed by matplotlib or ImageJ
- orientation: permuation of 1,2,3 with potential sign, indicating which axes map onto the reference axes, a negative sign indicates reversal of that particular axes
- only a minimal set of information to detremine the resampling parameter has to be given, e.g. dataSizeSource and dataSizeSink

**resampleDataInverse**(*sink*, *source=None*, *dataSizeSource=None*, *orientation=None*, *resolutionSource=(4.0625, 4.0625, 3)*, *resolutionSink=(25, 25, 25)*, *processingDirectory=None*, *processes=1*, *cleanup=True*, *verbose=True*, *interpolation='linear'*, ***args*)

Resample data inversely to `resampleData()` routine

**Parameters**

- **sink** (*str or None*) – image to be inversly resampled (=sink in `resampleData()`)
- **source** (*str or array*) – destination for inversly resmapled image (=source in `resampleData()`)
- **dataSizeSource** (*tuple or None*) – target size of the resampled image
- **orientation** (*tuple*) – orientation specified by permuation and change in sign of (1,2,3)
- **resolutionSource** (*tuple*) – resolution of the source image (in length per pixel)
- **resolutionSink** (*tuple*) – resolution of the resampled image (in length per pixel)
- **processingDirectory** (*str or None*) – directory in which to perform resmapling in parallel, None a temporary directry will be created
- **processes** (*int*) – number of processes to use for parallel resampling
- **cleanup** (*bool*) – remove temporary files
- **verbose** (*bool*) – display progress information
- **interpolation** (*str*) – method to use for interpolating to the resmapled image

**Returns** *(array or str)* – data or file name of resampled image

#### Notes

- resolutions are assumed to be given for the axes of the intrinsic orientation of the data and reference as when viewed by matplotlib or ImageJ
- orientation: permuation of 1,2,3 with potential sign, indicating which axes map onto the reference axes, a negative sign indicates reversal of that particular axes
- only a minimal set of information to detremine the resampling parameter has to be given, e.g. dataSizeSource and dataSizeSink

**resamplePoints**(*pointSource*, *pointSink=None*, *dataSizeSource=None*, *dataSizeSink=None*, *orientation=None*, *resolutionSource=(4.0625, 4.0625, 3)*, *resolutionSink=(25, 25, 25)*, ***args*)

Resample Points to map from original data to the coordinates of the resampled image

The resampling of points here corresponds to he resampling of an image in `resampleData()`

**Parameters**

- **pointSource** (*str or array*) – image to be resampled
- **pointSink** (*str or None*) – destination of resampled image
- **orientation** (*tuple*) – orientation specified by permuation and change in sign of (1,2,3)
- **dataSizeSource** (*str, tuple or None*) – size of the data source
- **dataSizeSink** (*str, tuple or None*) – target size of the resampled image
- **resolutionSource** (*tuple*) – resolution of the source image (in length per pixel)
- **resolutionSink** (*tuple*) – resolution of the resampled image (in length per pixel)

**Returns** *(array or str)* – data or file name of resampled points

#### Notes

•resolutions are assumed to be given for the axes of the intrinsic orientation of the data and reference as when viewed by matplotlib or ImageJ

•orientation: permuation of 1,2,3 with potential sign, indicating which axes map onto the reference axes, a negative sign indicates reversal of that particular axes

•only a minimal set of information to detremine the resampling parameter has to be given, e.g. dataSize-Source and dataSizeSink

**resamplePointsInverse**(*pointSource*, *pointSink=None*, *dataSizeSource=None*, *dataSizeSink=None*, *orientation=None*, *resolutionSource=(4.0625*, *4.0625*, *3)*, *resolutionSink=(25*, *25*, *25)*, ***args*)

Resample points from the coordinates of the resampled image to the original data

The resampling of points here corresponds to he resampling of an image in `resampleDataInverse()`

**Parameters**

- **pointSource** (*str or array*) – image to be resampled
- **pointSink** (*str or None*) – destination of resampled image
- **orientation** (*tuple*) – orientation specified by permuation and change in sign of (1,2,3)
- **dataSizeSource** (*str, tuple or None*) – size of the data source
- **dataSizeSink** (*str, tuple or None*) – target size of the resampled image
- **resolutionSource** (*tuple*) – resolution of the source image (in length per pixel)
- **resolutionSink** (*tuple*) – resolution of the resampled image (in length per pixel)

**Returns** *(array or str)* – data or file name of inversely resampled points

#### Notes

•resolutions are assumed to be given for the axes of the intrinsic orientation of the data and reference as when viewed by matplotlib or ImageJ

•orientation: permuation of 1,2,3 with potential sign, indicating which axes map onto the reference axes, a negative sign indicates reversal of that particular axes

•only a minimal set of information to detremine the resampling parameter has to be given, e.g. dataSize-Source and dataSizeSink

**sagittalToCoronalData**(*source*, *sink=None*)
Change from saggital to coronal orientation

**Parameters**

- **source** (*str or array*) – source data to be reoriented
- **sink** (*str or None*) – destination for reoriented image

**Returns** *str or array* – reoriented data

### 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 the *StackProcessing* module.

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 withing 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 SubStack

**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 SubStack
- **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 SubStack

**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 SubStack

**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

Cell Detection Module

This is the main routine to run the individual routines to detect cells om volumetric image data.

ClearMap supports two predefined image processing pipelines which will extend in the future:

#### Example

```
>>> import ClearMap.IO as io
>>> import ClearMap.Settings as settings
>>> from ClearMap.ImageProcessing.CellDetection import detectCells;
>>> fn = os.path.join(settings.ClearMapPath, 'Test/Data/Synthetic/test_iDISCO_\d{3}.tif');
>>> parameter = {"filterDoGParameter" : {"size": (5,5,5)}, "findExtendedMaximaParameter" : {"thresho
>>> img = io.readData(fn);
>>> img = img.astype('int16'); # converting data to smaller integer types can be more memory efficie
>>> res = detectCells(img, parameter);
>>> print res[0].shape
```

**See also:**

*ImageProcessing*

**detectCells** (*source*, *sink=None*, *method='SpotDetection'*, *processMethod=<built-in function all>*, *verbose=False*, ***parameter*)

Detect cells in data

This is a main script to start running the cell detection.

**Parameters**

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

| Method | Description |
|---|---|
| "SpotDetection" | uses predefined spot detection pipline |
| "Ilastik" | uses predefined pipline with cell classification via Ilastik |
| function | a user defined function |

- **processMethod** (*str or all*) – 'sequential' or 'parallel'. if all its choosen automatically
- **verbose** (*bool*) – print info

- ****parameter** (*dict*) – parameter for the image procesing sub-routines

Returns:

### ClearMap.ImageProcessing.CellSizeDetection module

Cell shape and size detection routines

The cell shape detection is based on a seeded and masked watershed.

**detectCellShape**(*img*, *peaks*, *detectCellShapeParameter=None*, *threshold=None*, *save=None*, *verbose=False*, *subStack=None*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Find cell shapes as labeled image

**Parameters**

- **img** (*array*) – image data
- **peaks** (*array*) – point data of cell centers / seeds
- **detectCellShape** (*dict*) –

| Name | Type | Descritption |
|---|---|---|
| *threshold* | (float or None) | threshold to determine mask, pixel below this are background if None no mask is generated |
| *save* | (tuple) | size of the box on which to perform the *method* |
| *verbose* | (bool or int) | print / plot information about this step |

- **verbose** (*bool*) – print progress info
- **out** (*object*) – object to write progress info to

**Returns** *array* – labeled image where each label indicates a cell

**findCellSize**(*imglabel*, *findCelSizeParameter=None*, *maxLabel=None*, *verbose=False*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Find cell size given cell shapes as labled image

**Parameters**

- **imglabel** (*array or str*) – labeled image, where each cell has its own label
- **findCelSizeParameter** (*dict*) –

| Name | Type | Descritption |
|---|---|---|
| *maxLabel* | (int or None) | maximal label to include, if None determine automatically |
| *verbose* | (bool or int) | print / plot information about this step |

- **verbose** (*bool*) – print progress info
- **out** (*object*) – object to write progress info to

**Returns** *array* – measured intensities

**findCellIntensity**(*img*, *imglabel*, *findCellIntensityParameter=None*, *maxLabel=None*, *method='Sum'*, *verbose=False*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Find integrated cell intensity given cell shapes as labled image

**Parameters**

- **img** (*array or str*) – image data
- **imglabel** (*array or str*) – labeled image, where each cell has its own label

- **findCellIntensityParameter** (*dict*) –

| Name | Type | Descritption |
|---|---|---|
| *maxLabel* | (int or None) | maximal label to include, if None determine automatically |
| *method* | (str) | method to use for measurment: 'Sum', 'Mean', 'Max', 'Min' |
| *verbose* | (bool or int) | print / plot information about this step |

- **verbose** (*bool*) – print progress info
- **out** (*object*) – object to write progress info to

**Returns** *array* – measured intensities

### 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** Linear filter module

**filterLinear**(*img*, *filterLinearParameter=None*, *ftype=None*, *size=None*, *sigma=None*, *sigma2=None*, *save=None*, *subStack=None*, *verbose=False*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Applies a linear filter to the image

**Parameters**

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

| Name | Type | Descritption |
|---|---|---|
| *ftype* | (str or None) | the type of the filter, see *Filter Type* if None do ot perform any fitlering |
| *size* | (tuple or None) | size for the filter if None, do not perform filtering |
| *sigma* | (tuple or None) | std of outer Guassian, if None autmatically determined from size |
| *sigma2* | (tuple or None) | std of inner Guassian, if None autmatically determined from size |
| *save* | (str or None) | file name to save result of this operation if None dont save to file |
| *verbose* | (bool or int) | print progress information |

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

- **out** (*object*) – object to write progress info to

**Returns** *array* – filtered image

**Note:** Converts image to float32 type if filter is active!

**ClearMap.ImageProcessing.Filter.DoGFilter module** DoG filter module

**filterDoG**(*img*, *filterDoGParameter=None*, *size=None*, *sigma=None*, *sigma2=None*, *save=None*, *verbose=None*, *subStack=None*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)
Difference of Gaussians (DoG) filter step

**Parameters**

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

| Name | Type | Descritption |
|---|---|---|
| *size* | (tuple or None) | size for the DoG filter if None, do not correct for any background |
| *sigma* | (tuple or None) | std of outer Guassian, if None autmatically determined from size |
| *sigma2* | (tuple or None) | std of inner Guassian, if None autmatically determined from size |
| *save* | (str or None) | file name to save result of this operation if None dont save to file |
| *verbose* | (bool or int) | print progress information |

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

**Returns** *array* – DoG filtered image

**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

Illumination correction toolbox.

The module provides a function to correct illumination/vignetting systematic variations in intensity.

The intensity image $I(x)$ given a flat field $F(x)$ and a background $B(x)$ the image is corrected to $C(x)$ as:

The module also has functionality to create flat field corections from measured intensity changes in a single direction, useful e.g. for lightsheet images, see e.g. `flatfieldLineFromRegression()`.

#### References

Fundamentals of Light Microscopy and Electronic Imaging, p. 421

**DefaultFlatFieldLineFile = '/home/ckirst/Science/Projects/BrainActivityMap/Analysis/ClearMap/Data/lightsheet_flatfi**
Default file of points along the illumination changing line for the flat field correction

**See also:**

*correctIllumination()*

**correctIllumination**(*img*, *correctIlluminationParameter=None*, *flatfield=None*, *background=None*, *scaling=None*, *save=None*, *verbose=False*, *subStack=None*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)
Correct illumination variations

The intensity image $I(x)$ given a flat field $F(x)$ and a background $B(x)$ the image is corrected to $C(x)$ as:

If the background is not given $B(x) = 0$.

The correction is done slice by slice assuming the data was collected with a light sheet microscope.

The image is finally optionally scaled.

**Parameters**

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

| Name | Type | Descritption |
|---|---|---|
| *flatfield* | (str, None or array) | flat field intensities, if None d onot correct image for illumination, if True the |
| *background* | (str, None or array) | background image as file name or array if None background is assumed to be zero |
| *scaling* | (str or None) | scale the corrected result by this factor if 'max'/'mean' scale to keep max/mean invariant |
| *save* | (str or None) | save the corrected image to file |
| *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* – illumination corrected image

### References

Fundamentals of Light Microscopy and Electronic Imaging, p 421

**See also:**

*DefaultFlatFieldLineFile*

**flatfieldFromLine**(*line*, *xsize*)
Creates a 2d flat field image from a 1d line of estimated intensities

**Parameters**

- **lines** (*array*) – array of intensities along y axis
- **xsize** (*int*) – size of image in x dimension

**Returns** *array* – full 2d flat field

**flatfieldLineFromRegression**(*data*, *sink=None*, *method='polynomial'*, *reverse=None*, *verbose=False*)

Create flat field line fit from a list of positions and intensities

The fit is either to be assumed to be a Gaussian:

or follows a order 6 radial polynomial

**Parameters**

- **data** (*array*) – intensity data as vector of intensities or (n,2) dim array of positions d=0 and intensities measurements d=1:-1
- **sink** (*str or None*) – destination to write the result of the fit
- **method** (*str*) – method to fit intensity data, 'Gaussian' or 'Polynomial'
- **reverse** (*bool*) – reverse the line fit after fitting
- **verbose** (*bool*) – print and plot information for the fit

**Returns** *array* – fitted intensities on points

### ClearMap.ImageProcessing.BackgroundRemoval module

Functions to remove background in images

The main routine subtracts a morphological opening from the original image for background removal.

**removeBackground**(*img*, *removeBackgroundParameter=None*, *size=None*, *save=None*, *verbose=False*, *subStack=None*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Remove background via subtracting a morphological opening from the original image

Background removal is done z-slice by z-slice.

**Parameters**

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

| Name | Type | Descritption |
|---|---|---|
| *size* | (tuple or None) | size for the structure element of the morphological opening if None, do not correct for any background |
| *save* | (str or None) | file name to save result of this operation if None dont save to file |
| *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* – background corrected image

### ClearMap.ImageProcessing.GreyReconstruction module

Grey reconstruction module

This morphological reconstruction routine was adapted from CellProfiler.

Author

Original author: Lee Kamentsky Copyright (c) 2003-2009 Massachusetts Institute of Technology Copyright (c) 2009-2011 Broad Institute

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

**reconstruct** (*seed*, *mask*, *method='dilation'*, *selem=None*, *offset=None*)
Performs a morphological reconstruction of an image.

Reconstruction uses a seed image, which specifies the values to dilate and a mask image that gives the maximum allowed dilated value at each pixel.

The algorithm is taken from [1]. Applications for greyscale reconstruction are discussed in [2] and [3].

**Parameters**

- **seed** (*array*) – seed image to be dilated or eroded.
- **mask** (*array*) – maximum (dilation) / minimum (erosion) allowed
- **method** (*str*) – {'dilation'|'erosion'}
- **selem** (*array*) – structuring element
- **offset** (*array or None*) – offset of the structuring element, None is centered

**Returns** *array* – result of morphological reconstruction.

---

**Note:** Operates on 2d images.

---

Reference:

**greyReconstruction** (*img*, *mask*, *greyReconstructionParameter=None*, *method=None*, *size=3*, *save=None*, *verbose=False*, *subStack=None*, *out=<open file '<stdout>'*, *mode 'w'>*, ***parameter*)
Calculates the grey reconstruction of the image

Reconstruction is done z-slice by z-slice.

**Parameters**

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

| Name | Type | Descritption |
|---|---|---|
| *method* | (tuple or None) | 'dilation' or 'erosion', if None return original image |
| *size* | (int or tuple) | size of structuring element |
| *save* | (str or None) | file name to save result of this operation if None dont save to file |
| *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

[1] Robinson, "Efficient morphological reconstruction: a downhill filter", Pattern Recognition Letters 25 (2004) 1759-1767.
[2] Vincent, L., "Morphological Grayscale Reconstruction in Image Analysis: Applications and Efficient Algorithms", IEEE Transactions on Image Processing (1993)
[3] Soille, P., "Morphological Image Analysis: Principles and Applications", Chapter 6, 2nd edition (2003), ISBN 3540429883.

**Returns** *array* – grey reconstructed image

### ClearMap.ImageProcessing.SpotDetection module

Functions to detect spots in images

The main routine `detectCells()` uses a difference of gaussian filter (see :mod:'~ClearMap.ImageProcessing.Filter) followed by a peak detection step.

#### Example

```
>>> import os
>>> import ClearMap.IO as io
>>> import ClearMap.Settings as settings
>>> import ClearMap.ImageProcessing.SpotDetection as sd
>>> fn = os.path.join(settings.ClearMapPath, 'Test/Data/Synthetic/test_iDISCO_\d{3}.tif');
>>> img = io.readData(fn);
>>> img = img.astype('int16'); # converting data to smaller integer types can be more memory efficie
>>> res = sd.detectSpots(img, dogSize = (5,5,5), flatfield = None, threshold = 5, cellShapeThreshold
>>> print 'Found %d cells !' % res[0].shape[0]
Illumination: flatfield          : None
Illumination: illuminationScaling: True
Illumination: background         : None
Background: backgroundSize: (15, 15)
Background: elapsed time: 0:00:00
DoG: dogSize: (5, 5, 5)
DoG: elapsed time: 0:00:00
Extended Max: threshold   : 5
Extended Max: localMaxSize: 5
Extended Max: hMax        : None
Extended Max: elapsed time: 0:00:00
Cell Centers: elapsed time: 0:00:00
Cell Shape: cellShapeThreshold: 1
Cell Shape:: elapsed time: 0:00:00
Cell Size:: elapsed time: 0:00:00
Cell Intensity: cellIntensityMethod: Max
Cell Intensity:: elapsed time: 0:00:00
Cell Intensity: cellIntensityMethod: Max
Cell Intensity:: elapsed time: 0:00:00
Cell Intensity: cellIntensityMethod: Max
Cell Intensity:: elapsed time: 0:00:00
Found 38 cells !
```

After execution this example inspect the result of the cell detection in the folder Test/Data/CellShape/**cellshape**_d{3}.tif.

**detectSpots**(*img*, *detectSpotsParameter=None*, *correctIlluminationParameter=None*, *removeBackgroundParameter=None*, *filterDoGParameter=None*, *findExtendedMaximaParameter=None*, *detectCellShapeParameter=None*, *verbose=False*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Detect Cells in 3d grayscale image using DoG filtering and maxima detection

**Effectively this function performs the following steps:**

- illumination correction via *correctIllumination()*
- background removal via *removeBackground()*

- difference of Gaussians (DoG) filter via `filterDoG()`
- maxima detection via *findExtendedMaxima()*
- cell shape detection via *detectCellShape()*
- cell intensity and size measurements via: *findCellIntensity()*, *findCellSize()*.

**Note:** Processing steps are done in place to save memory.

**Parameters**

- **img** (*array*) – image data
- **detectSpotParameter** – image processing parameter as described in the individual sub-routines
- **verbose** (*bool*) – print progress information
- **out** (*object*) – object to print progress information to

**Returns**

*tuple* – tuple of arrays (cell coordinates, raw intensity, fully filtered

intensty, illumination and background corrected intensity [, cell size])

**test**()
Test Spot Detection Module

#### ClearMap.ImageProcessing.MaximaDetection module

Collection of routines to detect maxima

Used for finding cells or intensity peaks.

**hMaxTransform**(*img*, *hMax*)
Calculates h-maximum transform of an image

**Parameters**

- **img** (*array*) – image
- **hMax** (*float or None*) – h parameter of h-max transform

**Returns** *array* – h-max transformed image if h is not None

**localMax**(*img*, *size=5*)
Calculates local maxima of an image

**Parameters**

- **img** (*array*) – image
- **size** (*float or None*) – size of volume to search for maxima

**Returns** *array* – mask that is True at local maxima

**extendedMax**(*img*, *hMax=0*)
Calculates extened h maxima of an image

Extended maxima are the local maxima of the h-max transform

**Parameters**

- **img** (*array*) – image
- **hMax** (*float or None*) – h parameter of h-max transform

**Returns** *array* – extended maxima of the image

**findExtendedMaxima**(*img*, *findExtendedMaximaParameter=None*, *hMax=None*, *size=5*, *threshold=None*, *save=None*, *verbose=None*, *subStack=None*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Find extended maxima in an image

Effectively this routine performs a h-max transfrom, followed by a local maxima search and thresholding of the maxima.

**Parameters**

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

| Name | Type | Descritption |
|---|---|---|
| *hMax* | (float or None) | h parameter for the initial h-Max transform if None, do not perform a h-max transform |
| *size* | (tuple) | size for the structure element for the local maxima filter |
| *threshold* | (float or None) | include only maxima larger than a threshold if None keep all localmaxima |
| *save* | (str or None) | file name to save result of this operation if None do not save result to file |
| *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* – binary image with True pixel at extended maxima

**See also:**

*hMaxTransform()*, *localMax()*

**findCenterOfMaxima**(*img*, *imgmax*, *findCenterOfMaximaParameter=None*, *save=None*, *verbose=False*, *subStack=None*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Find center of detected maxima weighted by intensity

**Parameters**

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

| Name | Type | Descritption |
|---|---|---|
| *save* | (str or None) | saves result of labeling the differnet maxima if None, do the lableling is not saved |
| *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* – coordinates of centers of maxima, shape is (n,d) where n is

number of maxima and d the dimension of the image

**findPixelCoordinates**(*imgmax*, *subStack=None*, *verbose=False*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Find coordinates of all pixel in an image with positive or True value

**Parameters**

- **img** (*array*) – image data
- **verbose** (*bool*) – print progress info
- **out** (*object*) – object to write progress info to

**Returns**

*array* – coordinates of centers of True pixels, shape is (n,d) where n is

number of maxima and d the dimension of the image

**findIntensity**(*img*, *centers*, *findIntensityParameter=None*, *method=None*, *size=(3, 3, 3)*, *verbose=False*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Find instensity value around centers in the image

**Parameters**

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

| Name | Type | Descritption |
|---|---|---|
| *method* | (str, func, None) | method to use to determine intensity (e.g. "Max" or "Mean") if None take intensities at the given pixels |
| *size* | (tuple) | size of the box on which to perform the *method* |
| *verbose* | (bool or int) | print / plot information about this step |

- **verbose** (*bool*) – print progress info
- **out** (*object*) – object to write progress info to

**Returns** *array* – measured intensities

### ClearMap.ImageProcessing.IlastikClassification module

Inteface to Illastik pixel classification

This module allows to integrate ilastik pixel classification into the *ClearMap* pipeline.

To train a classifier ilastik 0.5 should be used following these steps:

- genereate a classifier from illastik 0.5
- press 'Train and classify' button (eventhough the online traning is running) !
- save the classifier to a file
- use the classifiers file name in the ClearMap routine `classifyPixel()`
- try to avoid too many features in the classifier as classification gets very memory intensive otherwise

**Note:** Note that ilastik classification works in parallel, thus it is advised to process the data sequentially, see **:fun:'~ClearMap.Imageprocessing.StackProcessing.sequentiallyProcessStack'**

**Note:** Ilastik 0.5 works for images in uint8 format !

### References

- Ilastik
- Based on the ilastik interface from cell profiler

**initializeIlastik** (*path=None*)
Set system path for illastik installation

**Initialized = True**
bool: True if ilastik interface was sucessfully initialized .

**rescaleToIlastik** (*img*, *rescale=None*, *verbose=False*, *out=<open file '<stdout>'*, *mode 'w'>*, ***parameter*)
Rescale image to achieve uint8 format used by ilasstik

The function rescales the image and converts the image to uin8 to fit with the image representation used by ilastik.

**Parameters**

- **img** (*array*) – image data
- **rescale** (*float or None*) – rescaling factor

**Returns** *array* – uint8 version of the image

**rescaleFactorIlastik** (*source*, *processes=12*)
Determines rescale factor given a image file

**Parameters** **source** – source file / image

**Returns** *float* – rescale factor

**classifyPixel** (*img*, *classifyPixelParameter=None*, *subStack=None*, *verbose=False*, *out=<open file '<stdout>'*, *mode 'w'>*, ***parameter*)
Detect Cells Using a trained classifier in Ilastik

**Parameters**

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

| Name | Type | Descritption |
|---|---|---|
| *classifier* | (str or None) | saves result of labeling the differnet maxima if None dont correct image for illumination, if True the |
| *rescale* | (float,all, or None) | optional rescaling of the image to fit uint8 format used by ilastik, rescale |
| *save* | (str or None) | save the propabilities to belong to the classes to a file |
| *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* – probabilities for each pixel to belong to a class in the

classifier, shape is (img.shape, number of classes)

**classifyCells**(*img*, *classifyCellsParameter=None*, *classifier=None*, *rescale=None*, *save=None*, *verbose=False*, *subStack=None*, *out=<open file '<stdout>', mode 'w'>*, ***parameter*)

Detect Cells Using a trained classifier in Ilastik

The routine assumes that the first class is identifying the cells.

**Parameters** **img** (*array*) – image data classifyPixelParameter (dict):

| Name | Type | Descritption |
|---|---|---|
| *classifier* | (str or None) | saves result of labeling the differnet maxima if None dont correct image for illumination, if True the |
| *rescale* | (float or None) | optional rescaling of the image to fit uint8 format used by ilastik |
| *save* | (str or None) | save the detected cell pixel to a file |
| *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** *tuple* – centers of the cells, intensity measurments

**Note:** The routine could be poteNtially refined to make use of background detected by ilastik

#### ClearMap.ImageProcessing.ImageStatistics module

### 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 | Descrition |
|---|---|
| *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 = '/home/ckirst/Science/Projects/BrainActivityMap/Analysis/ClearMap/Test/Data/Annotation/**
str: default volumetric annotated image file

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

**DefaultAnnotationFile = '/home/ckirst/Science/Projects/BrainActivityMap/Analysis/ClearMap/Data/ARA2_annotation_i**
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='/home/ckirst/Science/Projects/BrainActivityMap/Analysis/ClearMap/Data/ARA2_annotation_*
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='/home/ckirst/Science/Projects/BrainActivityMap/Analysis/ClearMap/Data/ARA2_annotation*

**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='/home/ckirst/Science/Projects/BrainActivityMap/Analysis/ClearMap/Data/ARA2_annotation_info_col*

**labelAtLevel** (*label*, *level*)

**labelAtCollapse** (*label*)

**labelPoints** (*points*, *labeledImage='/home/ckirst/Science/Projects/BrainActivityMap/Analysis/ClearMap/Test/Data/Annotation/ann*
*level=None*, *collapse=None*)

**countPointsInRegions** (*points*, *labeledImage='/home/ckirst/Science/Projects/BrainActivityMap/Analysis/ClearMap/Test/Data/A*
*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='/home/ckirst/Science/Projects/BrainActivityMap/Analysis/ClearMap*
*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='/home/ckirst/Science/Projects/BrainActivityMap/Analysis/C*
*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='/home/ckirst/Science/Projects/BrainActivityMap/Analysis/ClearMap/Test/Data*
*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

Converts point data into voxel image data for visulaization and analysis

**voxelize**(*points*, *dataSize=None*, *sink=None*, *voxelizeParameter=None*, *method='Spherical'*, *size=(5, 5, 5)*, *weights=None*)
Converts a list of points into an volumetric image array

**Parameters**

- **points** (*array*) – point data array
- **dataSize** (*tuple*) – size of final image
- **sink** (*str, array or None*) – the location to write or return the resulting voxelization image, if None return array
- **voxelizeParameter** (*dict*) –

| Name | Type | Descritption |
|---|---|---|
| *method* | (str or None) | method for voxelization: 'Spherical', 'Rectangular' or 'Pixel' |
| *size* | (tuple) | size parameter for the voxelization |
| *weights* | (array or None) | weights for each point, None is uniform weights |

**Returns** *(array)* – volumetric data of smeared out points

**voxelizePixel**(*points*, *dataSize=None*, *weights=None*)
Mark pixels/voxels of each point in an image array

**Parameters**

- **points** (*array*) – point data array
- **dataSize** (*tuple or None*) – size of the final output data, if None size is determined by maximal point coordinates
- **weights** (*array or None*) – weights for each points, if None weights are all 1s.

**Returns** *(array)* – volumetric data with with points marked in voxels

**test**()
Test 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

Plotting routines for overlaying lables, tilings, and sectioning of 3d data sets

Supported functionality:

- plot volumetric data as a sequnece of tiles via `plotTiling()`
- overlay points on imags via `overlayPoints()` and `plotOverlayPoints()`
- overlay labeled images on gray scale images via `overlayLabel()` and `plotOverlayLabel()`

Author

Christoph Kirst, The Rockefeller University, 2015

**plotTiling**(*dataSource*, *tiling='automatic'*, *maxtiles=20*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*)

Plot 3d image as 2d tiles

**Parameters**

- **dataSouce** (*str or array*) – volumetric image data
- **tiling** (*str or tuple*) – tiling specification
- **maxtiles** – maximalnumber of tiles
- **x, y, z** (*all or tuple*) – sub-range specification

**Returns** *(object)* – figure handle

**overlayLabel**(*dataSource*, *labelSource*, *sink=None*, *alpha=False*, *labelColorMap='jet'*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*)

Overlay a gray scale image with colored labeled image

**Parameters**

- **dataSouce** (*str or array*) – volumetric image data
- **labelSource** (*str or array*) – labeled image to be overlayed on the image data
- **sink** (*str or None*) – destination for the overlayed image
- **alpha** (*float or False*) – transparency
- **labelColorMap** (*str or object*) – color map for the labels
- **x, y, z** (*all or tuple*) – sub-range specification

**Returns** *(array or str)* – figure handle

**See also:**

*overlayPoints()*

**plotOverlayLabel**(*dataSource*, *labelSource*, *alpha=False*, *labelColorMap='jet'*, *x=<built-in function all>*, *y=<built-in function all>*, *z=<built-in function all>*, *tiling='automatic'*, *maxtiles=20*)

Plot gray scale image overlayed with labeled image

**Parameters**

- **dataSouce** (*str or array*) – volumetric image data
- **labelSource** (*str or array*) – labeled image to be overlayed on the image data
- **alpha** (*float or False*) – transparency
- **labelColorMap** (*str or object*) – color map for the labels
- **x, y, z** (*all or tuple*) – sub-range specification
- **tiling** (*str or tuple*) – tiling specification
- **maxtiles** – maximalnumber of tiles

**Returns** *(object)* – figure handle

**See also:**

*overlayLabel()*

**overlayPoints**(*dataSource, pointSource, sink=None, pointColor=[1, 0, 0], x=<built-in function all>, y=<built-in function all>, z=<built-in function all>*)
Overlay points on 3D data and return as color image

**Parameters**

- **dataSouce** (*str or array*) – volumetric image data
- **pointSource** (*str or array*) – point data to be overlayed on the image data
- **pointColor** (*array*) – RGB color for the overlayed points
- **x, y, z** (*all or tuple*) – sub-range specification

**Returns** *(str or array)* – image overlayed with points

**See also:**

*overlayLabel()*

**plotOverlayPoints**(*dataSource, pointSource, pointColor=[1, 0, 0], x=<built-in function all>, y=<built-in function all>, z=<built-in function all>*)
Plot points overlayed on gray scale 3d image as tiles.

**Parameters**

- **dataSouce** (*str or array*) – volumetric image data
- **pointSource** (*str or array*) – point data to be overlayed on the image data
- **pointColor** (*array*) – RGB color for the overlayed points
- **x, y, z** (*all or tuple*) – sub-range specification

**Returns** *(object)* – figure handle

**See also:**

*plotTiling()*

**test**()
Test 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 = '/home/ckirst/programs/ilastik-05/'**
str: Absolute path to the Ilastik 0.5 installation

#### Notes

Ilastik Webpage

Ilastik 0.5 Download

**ElastixPath = '/home/ckirst/programs/elastix/'**
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 = '/home/ckirst/Science/Projects/BrainActivityMap/Analysis/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

TWO

# INDICES

- genindex
- modindex
- search

CHAPTER

THREE

# QUICKSTART

Install ClearMap by downloading it form github:

```
$ git clone git@git.assembla.com:idisco.git
```

You can access documentation in the interpreter with Python's built-in help function:

```
>>> import ClearMap as cm
>>> help(cm)
```

For more details see the Introduction, the `examples` and the :doc:‘api/ClearMap ‘ section.

CHAPTER

FOUR

# AUTHOR

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

CHAPTER

FIVE

# LICENSE

GNU GENERAL PUBLIC LICENSE Version 3

See `LICENSE` or gnu.org for details.

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

## D



## E



## F

## G



## H



## I



## J



## L

## M



## N



## O



## P



## R

## S



## T



## V

## W