

Package ‘LOMAR’

May 7, 2026

Type Package

Title Localization Microscopy Data Analysis

Version 0.5.2

Maintainer Jean-Karim Heriche <heriche@embl.de>

Description Read, register and compare point sets from single molecule localization microscopy.

URL <https://git.embl.org/heriche/lomar>

Depends R (>= 3.6.0)

Imports Rcpp, FNN, stats, data.table, parallel, doParallel, foreach,
proxy, reshape2, pracma, transport, RANN, ff, dbscan, EBImage,
tools, rhdf5, mclust, methods, abind, alphashape3d

LinkingTo BH (>= 1.78.0-0), Rcpp

Suggests testthat

License GPL-3

Encoding UTF-8

ByteCompile true

RoxygenNote 7.3.3

SystemRequirements gmp, fftw3

NeedsCompilation yes

Author Jean-Karim Heriche [cre, aut] (ORCID:
<<https://orcid.org/0000-0001-6867-9425>>)

Repository CRAN

Date/Publication 2026-02-05 12:10:19 UTC

Contents

| | |
|----------------------------------|---|
| apply_transformation | 3 |
| ary2ps | 3 |
| binning | 4 |
| circle_hough_transform | 5 |

| | |
|------------------------------------|----|
| coloc_index | 6 |
| costWd | 7 |
| cpd | 7 |
| crop_point_set | 9 |
| denoise | 10 |
| dist_to_boundary | 10 |
| dist_to_line | 11 |
| downsample | 11 |
| find_elbow | 12 |
| Gaussian_Wd | 12 |
| get_kernel_matrix | 13 |
| get_persistence_diagrams | 14 |
| get_shape | 15 |
| get_surface_area | 15 |
| GMM_Wd | 16 |
| gradientWd | 16 |
| group_events | 17 |
| icp | 18 |
| idx2rowcol | 19 |
| img2ps | 19 |
| jrmpe | 20 |
| local_densities | 22 |
| locprec2cov | 23 |
| locs2ps | 24 |
| locs_from_csv | 24 |
| multiple_registration | 25 |
| points2img | 26 |
| points_from_roi | 27 |
| point_sets_from_locs | 27 |
| point_sets_from_tiffs | 29 |
| ps2ary | 30 |
| pssk | 30 |
| q2dr | 31 |
| q2r | 31 |
| restore_coordinates | 32 |
| rotx | 32 |
| roty | 33 |
| rotz | 33 |
| scale_alpha_shape | 34 |
| shape_features_3d | 34 |
| sliced_Wd | 35 |
| standardize_coordinates | 35 |
| tr | 36 |
| wgmmreg | 36 |

apply_transformation *apply_transformation*

Description

Apply rotation and translation to a point set

Usage

```
apply_transformation(X, R, t, s)
```

Arguments

| | |
|---|--------------------------------|
| X | a point set as an N x D matrix |
| R | D x D rotation matrix |
| t | 1 x D translation vector |
| s | scaling factor |

Value

transformed point set as a N x D matrix

ary2ps *ary2ps*

Description

Convert a 4d array to a list of 3d point sets. The points are formed by extracting the coordinates of array values strictly above the given cut-off (default 0).

Usage

```
ary2ps(ary, bkg = 0)
```

Arguments

| | |
|-----|---|
| ary | a 4d array with last dimension indexing instances. |
| bkg | Extract points for array values strictly above this (default = 0) |

Value

a list of point sets.

binning

binning

Description

Binning in 1D, 2D or 3D.

Usage

```
binning(x, y, nbins, xrange = NULL)
```

Arguments

| | |
|--------|---|
| x | design matrix, dimension n x d with d in 1:3. |
| y | either a response vector of length n or NULL. |
| nbins | vector of length d containing number of bins for each dimension, may be set to NULL. |
| xrange | range for endpoints of bins for each dimension, either matrix of dimension 2 x d or NULL. xrange is increased if the cube defined does not contain all design points. |

Details

Copied from package aws which is no longer in CRAN. Original author: Joerg Polzehl (polzehl@wias-berlin.de) who adapted code of function binning in package sm.

Value

a list with elements:

- x matrix of coordinates of non-empty bin centers
- x.freq number of observations in nonempty bins
- midpoints.x1 bin centers in dimension 1
- midpoints.x2 bin centers in dimension 2
- midpoints.x3 bin centers in dimension 3
- breaks.x1 break points dimension 1
- breaks.x2 break points dimension 2
- breaks.x3 break points dimension 3
- table.freq number of observations per bin
- means means of y in non-empty bins (if y isn't NULL)
- devs standard deviations of y in non-empty bins (if y isn't NULL)

`circle_hough_transform`*Circle Hough transform*

Description

Extract coordinates of the centres of circles from a 2D image using the Hough transform

Usage

```
circle_hough_transform(  
  pixels,  
  rmin,  
  rmax,  
  threshold,  
  resolution = 360,  
  min.separation = rmin/4,  
  ncpu = 1  
)
```

Arguments

| | |
|-----------------------------|---|
| <code>pixels</code> | input data, either a matrix representing a 2D image or a data frame of signal coordinates with columns x, y. For images, background is expected to be 0 and signal to have positive values. |
| <code>rmin</code> | minimum search radius. |
| <code>rmax</code> | maximum search radius. |
| <code>threshold</code> | score threshold between 0 and 1. |
| <code>resolution</code> | number of steps in the circle transform (default: 360). This represents the maximum number of votes a point can get. |
| <code>min.separation</code> | distance between circle centres below which overlapping circles are considered the same and merged (default to 0.25*rmin) |
| <code>ncpu</code> | number of threads to use to speed up computation (default: 1) |

Value

a data frame with columns x, y, r and score

Examples

```
point.set <- data.frame(x = c(-9.8, -5.2, 12.5, 2.5, 4.5, 1.3, -0.2, 0.4, 9.3, -1.4, 0.5, -1.1, -7.7),  
  y = c(-4.2, 1.5, -0.5, 12, -3, -7.2, 10.9, 6.7, -1.3, 10, 6.7, -6.2, 2.9))  
circles <- circle_hough_transform(pixels = point.set, rmin = 3, rmax = 6, resolution = 100,  
  threshold = 0.1, ncpu = 1)
```

`coloc_index`*coloc_index*

Description

Compute a co-localization index between two sets of points. Adapted from: Willems and MacGillavry, A coordinate-based co-localization index to quantify and visualize spatial associations in single-molecule localization microscopy. *Sci Rep* 12, 4676 (2022). <https://doi.org/10.1038/s41598-022-08746-4>

Usage

```
coloc_index(  
  P1,  
  locprec1 = NULL,  
  locprecz1 = NULL,  
  P2,  
  locprec2 = NULL,  
  locprecz2 = NULL  
)
```

Arguments

| | |
|-----------|---|
| P1 | a point set as matrix or data frame with columns x,y,z. |
| locprec1 | (optional) localization precision in x,y for P1 |
| locprecz1 | (optional) localization precision along z for P1 |
| P2 | a point set as matrix or data frame with columns x,y,z. |
| locprec2 | (optional) localization precision in x,y for P2 |
| locprecz2 | (optional) localization precision along z for P2 |

Details

This can be seen as measuring the similarity between two spatial distributions. Co-clustering in dense structures can give values above 1.

Localization precision is optional but if used then all locprec parameters must be specified.

Value

a list with two elements:

- vector of co-localization indices for points in P1 relative to P2
- vector of co-localization indices for points in P2 relative to P1

| | |
|--------|---------------|
| costWd | <i>costWd</i> |
|--------|---------------|

Description

Objective function to minimize when using GMMs

Usage

```
costWd(Tr, X, Y, CX, CY, w1 = NULL, w2 = NULL, S = NULL)
```

Arguments

| | |
|----|---|
| Tr | Transformation vector as translation vector + rotation (angle in 2d, quaternion in 3d)) |
| X | matrix of means of first GMM (i.e. reference point set) |
| Y | matrix of means of second GMM (i.e. moving point set) |
| CX | array of covariance matrices of first GMM such that X[i,] has covariance matrix CX[:,i] |
| CY | array of covariance matrices of second GMM such that Y[i,] has covariance matrix CY[:,i] |
| w1 | (optional) vector of mixture weights of first GMM. |
| w2 | (optional) vector of mixture weights of second GMM. |
| S | (optional) array of pre-computed $\sqrt{\sqrt{CX[:,i]} \%*\% CY[:,j] \%*\% \sqrt{CX[:,i]}}$ |

Value

cost value

| | |
|-----|------------|
| cpd | <i>cpd</i> |
|-----|------------|

Description

Affine and rigid registration of two point sets using the coherent point drift algorithm. See: Myronenko A., Song X. (2010): "Point-Set Registration: Coherent Point Drift", IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 32, issue 12, pp. 2262-2275.

Usage

```

cpd(
  X,
  Y,
  w = 0,
  weights = NULL,
  scale = FALSE,
  maxIter = 100,
  subsample = NULL,
  tol = 1e-04,
  rotation.only = FALSE
)

```

Arguments

| | |
|----------------------------|--|
| <code>X</code> | reference point set, a $N \times D$ matrix |
| <code>Y</code> | point set to transform, a $M \times D$ matrix, |
| <code>w</code> | noise weight in the range $[0, 1)$ |
| <code>weights</code> | a $M \times N$ matrix of point correspondence weights |
| <code>scale</code> | logical (default: <code>FALSE</code>), whether to use scaling |
| <code>maxIter</code> | maximum number of iterations to perform (default: 100) |
| <code>subsample</code> | can be set either to a number in $[0,1]$ representing the fraction of points to randomly select or a list with vector elements <code>X</code> and <code>Y</code> containing indices of points to select in <code>X</code> and <code>Y</code> |
| <code>tol</code> | tolerance for determining convergence |
| <code>rotation.only</code> | logical (default: <code>FALSE</code>), if <code>TRUE</code> , don't perform translation |

Value

a list of

- `Y`: transformed point set,
- `R`: rotation matrix,
- `t`: translation vector,
- `s`: scaling factor,
- `P`: matrix of correspondence probabilities between the two point sets,
- `sigma`: final variance,
- `iter`: number of iterations performed,
- `converged`: boolean, whether the algorithm has converged.

Examples

```

data.file1 <- system.file("test_data", "parasaurolophusA.txt", package = "LOMAR",
  mustWork = TRUE)
PS1 <- read.csv(data.file1, sep = '\\t', header = FALSE)
data.file2 <- system.file("test_data", "parasaurolophusB.txt", package = "LOMAR",
  mustWork = TRUE)
PS2 <- read.csv(data.file2, sep = '\\t', header = FALSE)
transformation <- cpd(PS1, PS2, maxIter = 10, tol = 1e-3)
## Not run:
# Visualize registration outcome
library(rgl)
plot3d(PS1, col = "blue")
points3d(PS2, col = "green")
points3d(transformation[['Y']], col = "magenta")

## End(Not run)

```

| | |
|----------------|-----------------------|
| crop_point_set | <i>crop_point_set</i> |
|----------------|-----------------------|

Description

Retain points in the set that are within the given distance from the geometric median of the set. Using the geometric median is more robust than using the centre of mass (i.e. mean).

Usage

```
crop_point_set(point.set, size, center = NULL)
```

Arguments

| | |
|-----------|--|
| point.set | a point set as a matrix with columns x,y,z. |
| size | vector of distances from the target region centre along each axis. Points are discarded if they are outside the ellipsoid defined by size and centred on the given position. |
| center | (optional) coordinates of the centre of the target region. If not given, default to the geometric median of the point set. |

Value

point set as a matrix with columns x,y,z.

| | |
|---------|----------------|
| denoise | <i>denoise</i> |
|---------|----------------|

Description

Point density is estimated using a Gaussian mixture model and points in low density regions are considered as noise and removed.

Usage

```
denoise(points, k = 16, prob = 0.3)
```

Arguments

| | |
|--------|---|
| points | a data frame with columns x,y,z. |
| k | integer, number of mixture components for the GMM |
| prob | probability level in the range [0,1] to identify high density regions |

Value

a point set

| | |
|------------------|-------------------------|
| dist_to_boundary | <i>dist_to_boundary</i> |
|------------------|-------------------------|

Description

Given a point set and an alpha-shape, get the distance of each point to the closest boundary point of the alpha-shape. Points inside the shape get negative values.

Usage

```
dist_to_boundary(points, shape)
```

Arguments

| | |
|--------|---|
| points | a data frame with x,y,z columns |
| shape | an object of class ashape3d with a single alpha value |

Value

vector of distances (negative values indicate points inside the shape)

| | |
|--------------|---------------------|
| dist_to_line | <i>dist_to_line</i> |
|--------------|---------------------|

Description

Compute distance between a set of points and a line defined by two points

Usage

```
dist_to_line(pts, a = NULL, b = NULL)
```

Arguments

| | |
|-----|--|
| pts | a data frame or matrix with 3 columns of coordinates |
| a | vector of coordinates of a point on the line |
| b | a second point on the line |

Value

vector of distances

| | |
|------------|-------------------|
| downsample | <i>downsample</i> |
|------------|-------------------|

Description

Weighted downsampling of a point set. If point weights are not provided, they are computed to be proportional to the local density around each point.

Usage

```
downsample(point.set, n = NULL, k = NULL, weights = NULL)
```

Arguments

| | |
|-----------|---|
| point.set | a point set |
| n | integer, sample size. |
| k | integer, number of nearest neighbours to consider to estimate local density |
| weights | a vector of probability weights |

Value

a point set

| | |
|------------|-------------------|
| find_elbow | <i>find_elbow</i> |
|------------|-------------------|

Description

Find elbow in a 2D curve represented by a list of ordered values

Usage

```
find_elbow(values)
```

Arguments

| | |
|--------|--------------------------------------|
| values | vector of values in decreasing order |
|--------|--------------------------------------|

Details

This function finds the point with maximum distance from the line between the first and last points. Adapted from StackOverflow: <http://stackoverflow.com/questions/2018178/finding-the-best-trade-off-point-on-a-curve>

Value

index and value of the selected point

| | |
|-------------|--------------------|
| Gaussian_Wd | <i>Gaussian_Wd</i> |
|-------------|--------------------|

Description

Compute 2-Wasserstein distance between two Gaussian distributions

Usage

```
Gaussian_Wd(m1, m2, S1, S2, S = NULL)
```

Arguments

| | |
|----|---|
| m1 | mean of first distribution |
| m2 | mean of second distribution |
| S1 | variance of first distribution |
| S2 | variance of second distribution |
| S | (optional) matrix of pre-computed $\sqrt{\text{S1}}$ %*% S2 %*% $\sqrt{\text{S1}}$ |

Value

distance value

```
get_kernel_matrix      get_kernel_matrix
```

Description

Compute kernel/distance matrix between persistence diagrams.

Usage

```
get_kernel_matrix(
  Diag = NULL,
  method = c("sWd", "pssk"),
  dimensions = NULL,
  return.dist = FALSE,
  M = NULL,
  sigma = NULL,
  ncpu = 1,
  cluster.type = "PSOCK"
)
```

Arguments

| | |
|--------------|--|
| Diag | list of persistence diagrams as n x 3 matrices |
| method | which kernel or distance to compute. One of sWd (for sliced Wasserstein kernel) or pssk (for the persistence scale-space kernel) |
| dimensions | vector of the dimensions of the topological features to consider, if NULL (default) use all available dimensions |
| return.dist | logical (default: FALSE) for method sWd, whether to return the sliced Wasserstein distance matrix instead of the kernel. |
| M | number of slices for the sliced Wasserstein kernel |
| sigma | kernel bandwidth |
| ncpu | number of parallel threads to use for computation |
| cluster.type | type of multicore cluster to use, either PSOCK (default) or FORK |

Value

a matrix

Examples

```
PS <- list(data.frame(x = c(2.4,-6.9,4.6,-0.7,-3.3,-4.9,-3.5,-3.5,4.2,-7),
  y = c(5.7,1.9,4.8,3.4,-3,-2.1,7.2,1.8,6.1,-1.6),
  z = c(2.7,-0.1,-0.7,-0.6,0.4,-1.5,-0.6,-0.9,2.2,0.7)),
  data.frame(x = c(0,0,3.1,-5.6,-5,-7.4,-0.7,-7.7,-6.7,4,4.2,0.2,5.8,3.9,3.9),
  y = c(6.3,-6.1,-3.5,4.6,-4.1,0.3,8.8,-2.3,2.9,3.7,-1.4,-3.9,5.5,-1.2,-6.7),
  z = c(-1.5,1.7,-0.4,-1.4,1.8,1.7,-0.9,-1.8,-0.5,1.7,1.3,0.5,-1.4,1.6,-0.1)),
```

```

data.frame(x = c(-9.8,-5.2,12.5,2.5,4.5,1.3,-0.2,0.4,9.3,-1.4,0.5,-1.1,-7.7),
           y = c(-4.2,1.5,-0.5,12,-3,-7.2,10.9,6.7,-1.3,10,6.7,-6.2,2.9),
           z = c(3.4,-3.8,-1.4,1.8,3.5,2.5,2.6,-4.8,-3.8,3.9,4.1,-3.6,-4))
Dgs <- get_persistence_diagrams(point.sets = PS, maxdimension = 1, maxscale = 5, ncpu = 1)
K <- get_kernel_matrix(Diag = Dgs, method = 'sWd', dimensions = c(0,1), M = 10, sigma = 5)

```

```
get_persistence_diagrams
```

```
get_persistence_diagrams
```

Description

Compute persistence diagrams for a list of point sets. By default, compute persistent homology from the Vietoris-Rips filtration. If `use.dtm` is `TRUE`, compute instead the persistent homology of the sublevel set of the distance to measure evaluated over a grid.

Usage

```

get_persistence_diagrams(
  point.sets = NULL,
  maxdimension = NULL,
  maxscale = NULL,
  use.dtm = FALSE,
  m0 = NULL,
  grid.by = NULL,
  ncpu = 1,
  cluster.type = "PSOCK"
)

```

Arguments

| | |
|---------------------------|--|
| <code>point.sets</code> | list of point sets, each as a data frame with columns <code>x,y,z</code> |
| <code>maxdimension</code> | maximum dimension of the homological features to be computed |
| <code>maxscale</code> | limit of the Vietoris-Rips filtration |
| <code>use.dtm</code> | logical (default: <code>FALSE</code>), whether to use the distance to measure function |
| <code>m0</code> | parameter for the dtm function |
| <code>grid.by</code> | vector of space between points of the grid for the dtm function along each dimension |
| <code>ncpu</code> | number of parallel threads to use for computation |
| <code>cluster.type</code> | type of multicore cluster to use, either <code>PSOCK</code> (default) or <code>FORK</code> |

Value

a list of persistence diagrams as $n \times 3$ matrices. Each row is a topological feature and the columns are dimension, birth and death of the feature.

Examples

```
PS <- list(data.frame(x = c(2.4,-6.9,4.6,-0.7,-3.3,-4.9,-3.5,-3.5,4.2,-7),
  y = c(5.7,1.9,4.8,3.4,-3,-2.1,7.2,1.8,6.1,-1.6),
  z = c(2.7,-0.1,-0.7,-0.6,0.4,-1.5,-0.6,-0.9,2.2,0.7)),
  data.frame(x = c(0,0,3.1,-5.6,-5,-7.4,-0.7,-7.7,-6.7,4,4.2,0.2,5.8,3.9,3.9),
  y = c(6.3,-6.1,-3.5,4.6,-4.1,0.3,8.8,-2.3,2.9,3.7,-1.4,-3.9,5.5,-1.2,-6.7),
  z = c(-1.5,1.7,-0.4,-1.4,1.8,1.7,-0.9,-1.8,-0.5,1.7,1.3,0.5,-1.4,1.6,-0.1)))
Diags <- get_persistence_diagrams(point.sets = PS, maxdimension = 1, maxscale = 5, ncpu = 1)
```

get_shape

*get_shape***Description**

Get the the alpha-shape of a point set. If not given, the function automatically determines alpha using a downsampled point set. As a consequence, alpha and therefore the computed shape can vary slightly between runs.

Usage

```
get_shape(points, alpha = NULL)
```

Arguments

| | |
|--------|------------------------------------|
| points | a data frame with columns x, y, z. |
| alpha | (optional) positive number |

Value

an alpha-shape object of class `ashape3d`

get_surface_area

*get_surface_area***Description**

Compute the surface area of an alpha-shape by summing the surfaces of the boundary triangles

Usage

```
get_surface_area(as)
```

Arguments

| | |
|----|--|
| as | an alpha-shape object of class <code>ashape3d</code> |
|----|--|

Value

a numeric value

GMM_Wd

*GMM_Wd***Description**

Compute 2-Wasserstein distance between two Gaussian mixture models See: Delon J, Desolneux A. (2019) A Wasserstein-type distance in the space of Gaussian Mixture Models. hal-02178204v2

Usage

```
GMM_Wd(m1, m2, S1, S2, w1 = NULL, w2 = NULL, S = NULL)
```

Arguments

| | |
|----|---|
| m1 | matrix of means of first GMM |
| m2 | matrix of means of second GMM |
| S1 | array of covariance matrices of first GMM such that m1[i,] has covariance matrix S1[:,i] |
| S2 | array of covariance matrices of second GMM such that m2[i,] has covariance matrix S2[:,i] |
| w1 | (optional) vector of mixture weights of first GMM. |
| w2 | (optional) vector of mixture weights of second GMM. |
| S | (optional) array of pre-computed $\sqrt{\sqrt{S1[:,i]} \%*\% S2[:,j] \%*\% \sqrt{S1[:,i]}}$ |

Value

list of distance value d and optimal transport matrix ot

gradientWd

*gradientWd***Description**

Gradient of the objective function with respect to rotation and translation parameters

Usage

```
gradientWd(Tr, X, Y, CX, CY, w1 = NULL, w2 = NULL, S = NULL)
```

Arguments

| | |
|----|---|
| Tr | Transformation vector as translation vector + rotation (angle in 2d, quaternion in 3d)) |
| X | matrix of means of first GMM (i.e. reference point set) |
| Y | matrix of means of second GMM (i.e. moving point set) |
| CX | array of covariance matrices of first GMM such that X[i,] has covariance matrix C1[:,i] |
| CY | array of covariance matrices of second GMM such that Y[i,] has covariance matrix C2[:,i] |
| w1 | (optional) vector of mixture weights of first GMM. |
| w2 | (optional) vector of mixture weights of second GMM. |
| S | (optional) array of pre-computed $\sqrt{\sqrt{CX[:,i]} \%*\% CY[:,j] \%*\% \sqrt{CX[:,i]}}$ |

Value

gradient vector

| | |
|--------------|---------------------|
| group_events | <i>group_events</i> |
|--------------|---------------------|

Description

Localisation events are grouped by recursively clustering mutual nearest neighbours. Neighbours are determined using the Mahalanobis distance to account for anisotropy in the localisation precision. Since the Mahalanobis distance has approximately a chi-squared distribution, a distance threshold can be chosen from a chi-squared table where the number of degrees of freedom is the dimension and alpha can be seen as the probability of missing a localization event generated from the same fluorophore as the event under consideration.

Usage

```
group_events(points, locprec = NULL, locprecz = NULL, p = 0.1)
```

Arguments

| | |
|----------|---|
| points | a data frame with columns x,y,z. |
| locprec | localization precision in x,y |
| locprecz | localization precision along z, defaults to locprec |
| p | confidence level, see description. Defaults to 0.1 |

Value

a list with two elements:

- points: a point set as data frame with columns x,y,z
- membership: a vector of integers indicating the cluster to which each input point is allocated.

Examples

```

data.file1 <- system.file("test_data", "parasaurolophusA.txt", package = "LOMAR",
  mustWork = TRUE)
PS1 <- read.csv(data.file1, sep = '\\t', header = FALSE)
data.file2 <- system.file("test_data", "parasaurolophusB.txt", package = "LOMAR",
  mustWork = TRUE)
PS2 <- read.csv(data.file2, sep = '\\t', header = FALSE)
transformation <- icp(PS1, PS2, iterations = 10, tol = 1e-3)
## Not run:
# Visualize registration outcome
library(rgl)
plot3d(PS1, col = "blue")
points3d(PS2, col = "green")
points3d(transformation[['Y']], col = "magenta")

## End(Not run)

```

*idx2rowcol**idx2rowcol*

Description

Convert indices into a dist object to row, column coordinates of the corresponding distance matrix

Usage

```
idx2rowcol(idx, n)
```

Arguments

| | |
|------------------|-----------------------------------|
| <code>idx</code> | vector of indices |
| <code>n</code> | size of the n x n distance matrix |

Value

a matrix with two columns nr and nc

*img2ps**img2ps*

Description

Read an image into a point set. The points are formed by extracting the coordinates of voxel values strictly above the given cut-off (default 0).

Usage

```
img2ps(img = NULL, bkg = 0, crop.size = NULL)
```

Arguments

img either a 2d or 3d array or a path to a file containing a 2d or 3d image.
bkg Extract points for values strictly above this (default = 0).
crop.size vector (of length 2 or 3) containing the desired reduced size of the images along each dimension, e.g. `c(30,30,30)`.

Value

a point set as matrix with columns *x,y[,z]*

Examples

```
img.file <- system.file("test_data/img", "alien1_3d.tif", package = "LOMAR",
  mustWork = TRUE)
point_set <- img2ps(img = img.file, bkg = 0)
```

*jrm*pc

*jrm*pc

Description

Joint registration of multiple point sets See: G. D. Evangelidis, D. Kounades-Bastian, R. Horaud, and E. Z. Psarakis. A generative model for the joint registration of multiple point sets. In European Conference on Computer Vision, pages 109–122. Springer, 2014

Usage

```
jrmpc(
  V,
  C = NULL,
  K = NULL,
  g = NULL,
  initialPriors = NULL,
  updatePriors = TRUE,
  maxIter = 100,
  fixedVarIter = 0,
  tol = 0.01,
  initializeBy = NULL,
  model.selection = FALSE,
  model.selection.threshold = NULL,
  rotation.only = FALSE
)
```

Arguments

| | |
|--|---|
| <code>V</code> | list of point sets as $N \times D$ matrices |
| <code>C</code> | (optional) list of arrays of covariance matrices with $C[[j]][i]$ the covariance matrix associated with point i of set j . |
| <code>K</code> | (optional) number of components of the GMM, defaults to the average number of points in a set. |
| <code>g</code> | (optional) proportion of noisy points, defaults to $1/K$. If set, priors will be initialized uniformly. |
| <code>initialPriors</code> | (optional) vector of length K of prior probabilities. Defaults to uniform distribution using g . If set, will determine g so it is an error to specify g with <code>initialPriors</code> . |
| <code>updatePriors</code> | logical, whether to update priors at each iteration (default: TRUE). |
| <code>maxIter</code> | maximum number of iterations to perform (default: 100). |
| <code>fixedVarIter</code> | number of iterations before starting variance updates |
| <code>tol</code> | tolerance for determining convergence (default: $1e-2$). |
| <code>initializeBy</code> | (optional) how to initialize the GMM means. Defaults to distributing the means on the surface of the sphere enclosing all (centred) sets. Currently supported values are: <ul style="list-style-type: none"> • 'sampling': sample from the data, • a $K \times D$ matrix of points |
| <code>model.selection</code> | whether to perform model selection (default: FALSE). If set to TRUE, GMM components with no support in the data are deleted. |
| <code>model.selection.threshold</code> | value below which we consider a GMM component has no support, set to $1/K$ if not explicitly given |
| <code>rotation.only</code> | if set to TRUE, no translation is performed (default: FALSE) |

Value

a list of

- `Y`: list of transformed point sets as $N \times d$ matrices,
- `R`: list of $d \times d$ rotation matrices, one for each point set in `V`,
- `t`: list of translation vectors, one for each point set in `V`,
- `M`: centres of the GMM,
- `S`: variances of the GMM.
- `a`: list of posterior probabilities as $N \times K$ matrices
- `iter`: number of iterations
- `conv`: error value used to evaluate convergence relative to `tol`
- `z`: support scores of the GMM components

Examples

```

X <- read.csv(system.file("test_data", "parasaurolophusA.txt", package="LOMAR",
  mustWork = TRUE), sep = "\t")
Y <- read.csv(system.file("test_data", "parasaurolophusB.txt", package="LOMAR",
  mustWork = TRUE), sep = "\t")
Z <- read.csv(system.file("test_data", "parasaurolophusC.txt", package="LOMAR",
  mustWork = TRUE), sep = "\t")
PS <- list(X, Y, Z)
C <- list()
for(i in 1:3) {
  cv <- diag(0.1, ncol(PS[[i]])) + jitter(0.01, amount = 0.01)
  cv <- replicate(nrow(PS[[i]]), cv)
  C[[i]] <- cv
}
transformation <- jrmpc(PS, C = C, K = 100, maxIter = 20, tol = 0.01,
  model.selection = TRUE)
## Not run:
# Visualize registration outcome
library(rgl)
colours <- c("blue", "green", "magenta")
Yt <- transformation[['Y']]
plot3d(Yt[[1]], col = colours[1])
for(i in 2:length(Yt)) {
  points3d(Yt[[i]], col = colours[i])
}
# Visualize GMM centres highlighting those with high variance
GMM <- as.data.frame(cbind(transformation[['M']], transformation[['S']]))
colnames(GMM) <- c("x", "y", "z", "S")
colours <- rep("blue", nrow(GMM))
# Find high variance components
threshold <- quantile(transformation[['S']], 0.75)
high.var.idx <- which(transformation[['S']]>threshold)
colours[high.var.idx] <- "red"
plot3d(GMM[, c("x", "y", "z")], col = colours, type = 's', size = 2, box = FALSE, xlab = '',
  ylab = '', zlab = '', xlim = c(-0.15,0.15), ylim = c(-0.15, 0.15),
  zlim = c(-0.15, 0.15))

## End(Not run)

```

local_densities

local_densities

Description

Compute local point density at each point of a point set

Usage

```
local_densities(X, k = NULL)
```

Arguments

- x point set, a N x D matrix
- k (optional) number of nearest neighbors used (defaults to all points).

Details

Local density is computed as in Ning X, Li F, Tian G, Wang Y (2018) An efficient outlier removal method for scattered point cloud data. PLOS ONE 13(8):e0201280. <https://doi.org/10.1371/journal.pone.0201280>

Value

vector of density value for each point

| | |
|-------------|--------------------|
| locprec2cov | <i>locprec2cov</i> |
|-------------|--------------------|

Description

Converts localization precision columns to a list of arrays of covariance matrices

Usage

```
locprec2cov(point.sets, scale = FALSE)
```

Arguments

- point.sets a list of n point sets with locprec columns (locprecz column required for 3D data)
- scale logical, whether to scale the localization precision by the variance of the coordinates

Value

a list of 2x2xn or 3x3xn arrays.

| | |
|---------|----------------|
| locs2ps | <i>locs2ps</i> |
|---------|----------------|

Description

Cluster localizations into point sets using DBSCAN

Usage

```
locs2ps(
  points,
  eps,
  minPts,
  keep.locprec = TRUE,
  keep.channel = TRUE,
  cluster.2d = FALSE
)
```

Arguments

| | |
|--------------|---|
| points | a point set as a data frame of coordinates with columns x,y,z. |
| eps | DBSCAN parameter, size of the epsilon neighbourhood |
| minPts | DBSCAN parameter, number of minimum points in the eps region |
| keep.locprec | logical (default: TRUE), whether to preserve the localization precision columns |
| keep.channel | logical (default: TRUE), whether to preserve channel information column |
| cluster.2d | logical (default: FALSE), whether to cluster only using x,y (and ignore z) |

Value

a list of matrices with columns x,y,z and eventually locprec[z] and names set to the cluster indices.

| | |
|---------------|----------------------|
| locs_from_csv | <i>locs_from_csv</i> |
|---------------|----------------------|

Description

Reads and filters single molecule localization events from a csv file as typically output by the SMAP software. The main columns of interest are the coordinates (x, y, z), point set membership (site) and localization precision (locprec and locprecz).

Usage

```
locs_from_csv(
  file = NULL,
  roi = NULL,
  channels = NULL,
  frame.filter = NULL,
  llrel.filter = NULL,
  locprec.filter = 0,
  locprecz.filter = 0
)
```

Arguments

| | |
|-----------------|--|
| file | a csv file with columns x[nm], y[nm], z[nm] and optionally site[numbers], channel, locprec[nm] and locprecz[nm], other columns are ignored. |
| roi | region of interest, keep points within the specified volume. Must be a data frame with columns x,y,z and rows min and max defining a bounding box. |
| channels | vector of integers indicating which channel(s) of a multicolour experiment to get data from. |
| frame.filter | vector of min and max values, filter out points from frames outside the specified range. |
| llrel.filter | vector of min and max values, filter out points on log-likelihood (for fitted data). |
| locprec.filter | filter out points with locprec value greater than the specified number. Points with locprec == 0 are also removed. |
| locprecz.filter | filter out points with locprecz value greater than the specified number. Points with locprecz == 0 are also removed. |

Value

a data frame with columns x,y,z, optionally site, locprec and locprecz.

Examples

```
data.file <- system.file("test_data", "simulated_NUP107_data.csv", package = "LOMAR",
  mustWork = TRUE)
locs <- locs_from_csv(file = data.file, locprec.filter = 20)
```

multiple_registration *multiple_registration*

Description

Registration of multiple point sets using tree-guided progressive registration followed by iterative refinement.

Usage

```
multiple_registration(PS, registration, refine.iter = 20, ...)
```

Arguments

| | |
|--------------|---|
| PS | list of point sets |
| registration | pairwise registration method to use |
| refine.iter | Maximum number of refinement iterations (default: 20) |
| ... | additional arguments to the registration method |

Value

a list of

- Y: list of transformed point sets as $N \times d$ matrices

points2img

points2img

Description

Convert a data frame of point coordinates into an image. Expected photon count at each voxel is computed as in: F. Huang, S. L. Schwartz, J. M. Byars, and K. A. Lidke, “Simultaneous multiple-emitter fitting for single molecule super-resolution imaging,” *Biomed. Opt. Express* 2(5), 1377–1393 (2011).

Usage

```
points2img(points, voxel.size, method, channels = NULL, ncpu = 1)
```

Arguments

| | |
|------------|---|
| points | a point set as a data frame of coordinates with columns x,y,z. |
| voxel.size | a numeric vector of length 3 indicating the size of the voxel along x,y and z in the same unit as the coordinates (e.g. nm) |
| method | how to calculate voxel values. Available methods are: <ul style="list-style-type: none"> • 'histogram': value is the number of points (i.e. emitters) in the voxel • 'photon': value is the expected number of photons from the points in the voxel. Input data frame must have columns locprec, locprecz and phot[on]. |
| channels | vector of channels to consider, must be values present in the input data frame channel column |
| ncpu | number of threads to use to speed up computation (default: 1) |

Value

an array of dimensions x,y,z and channels if applicable

Examples

```
point.set <- data.frame(x = c(-9.8,-5.2,12.5,2.5,4.5,1.3,-0.2,0.4,9.3,-1.4,0.5,-1.1,-7.7),
  y = c(-4.2,1.5,-0.5,12,-3,-7.2,10.9,6.7,-1.3,10,6.7,-6.2,2.9),
  z = c(3.4,-3.8,-1.4,1.8,3.5,2.5,2.6,-4.8,-3.8,3.9,4.1,-3.6,-4))
img <- points2img(point.set, voxel.size = c(2,2,2), method = 'histogram')
```

points_from_roi *points_from_roi*

Description

Extract points within given bounding box. Points are translated so that (0,0,0) correspond to the bounding box corner defined by roi['min',c('x','y','z')]

Usage

```
points_from_roi(points, roi)
```

Arguments

points a point set as a data frame of coordinates with columns x,y,z.
roi a data frame with columns x,y,z and rows min and max defining a bounding box

Value

a data frame with same columns as input

point_sets_from_locs *point_sets_from_locs*

Description

Extracts list of point sets from a data frame of single molecule localization coordinates. By default, uses point set membership indicated in the site column.

Usage

```
point_sets_from_locs(
  locs = NULL,
  channels = NULL,
  min.cardinality = NULL,
  max.cardinality = NULL,
  crop.size = NULL,
  keep.locprec = TRUE,
  sample.size = NULL,
```

```

    ignore.site = FALSE,
    cluster.points = FALSE,
    eps = NULL,
    minPts = NULL
  )

```

Arguments

| | |
|------------------------------|--|
| <code>locs</code> | a data frame with columns <code>x[nm]</code> , <code>y[nm]</code> , <code>z[nm]</code> and optionally <code>site[numbers]</code> , <code>locprec[nm]</code> and <code>locprecz[nm]</code> , other columns are ignored. |
| <code>channels</code> | vector of integers indicating which channel(s) of a multicolour experiment to extract point sets from. |
| <code>min.cardinality</code> | filter out point sets with less than the specified number of points. |
| <code>max.cardinality</code> | filter out point sets with more than the specified number of points. |
| <code>crop.size</code> | remove points from a set if they are further away than the specified distance from the center of the set. |
| <code>keep.locprec</code> | logical (default:TRUE). Whether to keep locprec information for each point. |
| <code>sample.size</code> | returns this number of randomly selected point sets. Selects the point sets after applying eventual filtering. |
| <code>ignore.site</code> | logical (default: FALSE), set to TRUE if point set membership is not present or needed. |
| <code>cluster.points</code> | logical (default: FALSE), whether to cluster the points using DBSCAN (only if <code>ignore.site</code> is also TRUE). |
| <code>eps</code> | DBSCAN parameter, size of the epsilon neighbourhood |
| <code>minPts</code> | DBSCAN parameter, number of minimum points in the eps region |

Value

a list of matrices with columns `x,y,z`, optionally `locprec` and `name` set to the value of the `site` column (if applicable).

Examples

```

data.file <- system.file("test_data", "simulated_NUP107_data.csv", package = "LOMAR",
  mustWork = TRUE)
locs <- locs_from_csv(file = data.file, locprec.filter = 20)
point.sets <- point_sets_from_locs(locs, keep.locprec = TRUE, min.cardinality = 15)

```

point_sets_from_tiffs *point_sets_from_tiffs*

Description

Read in single molecule localization events from a series of 3D images in TIFF files where each image file represents a point set.

Usage

```
point_sets_from_tiffs(  
  image_dir = NULL,  
  pattern = NULL,  
  image.size = NULL,  
  sample.size = NULL,  
  sample.first = FALSE,  
  min.cardinality = NULL,  
  max.cardinality = NULL,  
  crop.size = NULL  
)
```

Arguments

| | |
|------------------------------|---|
| <code>image_dir</code> | path to a directory containing the TIFF files. |
| <code>pattern</code> | regular expression, select images whose file path matches the given pattern. |
| <code>image.size</code> | vector of length 3 containing the size of the images along each dimension, e.g. <code>c(40,40,40)</code> . |
| <code>sample.size</code> | if set, selects this number of images at random. A sample size larger than the available number of samples produces a warning and is ignored. |
| <code>sample.first</code> | if TRUE, samples are selected before applying any eventual filtering. This is more efficient as it avoids reading all data files. |
| <code>min.cardinality</code> | if set, filter out all point sets with less than the specified number of points. |
| <code>max.cardinality</code> | if set, filter out all point sets with more than the specified number of points. |
| <code>crop.size</code> | vector of length 3 containing the desired reduced size of the images along each dimension, e.g. <code>c(30,30,30)</code> . |

Value

a list with two elements:

- `point.sets`: a list of point sets as matrices with columns x,y,z and
- `file.names`: a vector of paths to the TIFF files from which the point sets were extracted.

Examples

```
data.dir <- system.file("test_data/img", package = "LOMAR", mustWork = TRUE)
point_sets <- point_sets_from_tiffs(image_dir = data.dir, pattern = "\\..tiff?$",
  image.size = c(64, 64, 4), min.cardinality = 10)
```

ps2ary *ps2ary*

Description

Convert a list of 3d point sets to a 4d array. Also works for 2d point sets to 3d array conversion.

Usage

```
ps2ary(point.sets, dims)
```

Arguments

`point.sets` a list of point sets.
`dims` vector of dimensions of the axes (x,y in 2d, x,y,z in 3d).

Value

a 3d or 4d array.

pssk *pssk*

Description

Compute the persistence scale-space kernel on persistence diagrams. Reference: Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pages 4741–4748, 2015.

Usage

```
pssk(Dg1 = NULL, Dg2 = NULL, sigma = NULL, dimensions = NULL)
```

Arguments

`Dg1` a persistence diagram as a $n1 \times 3$ matrix where each row is a topological feature and the columns are dimension, birth and death of the feature.
`Dg2` another persistence diagram as a $n2 \times 3$ matrix
`sigma` kernel bandwidth
`dimensions` vector of the dimensions of the topological features to consider, if NULL (default) use all available dimensions

Value

kernel value

Examples

```
D1 <- matrix(c(0,0,0,1,1,0,0,0,1.5, 3.5,2,2.5,3, 4, 6), ncol = 3, byrow = FALSE)
D2 <- matrix(c(0,0,1,1,0, 0, 1.2, 2, 1.4, 3.2,4.6,6.5), ncol = 3, byrow = FALSE)
K <- pssk(Dg1 = D1, Dg2 = D2, sigma = 1)
```

q2dr

Get derivative of 3D rotation matrix from quaternion

Description

Get derivative of 3D rotation matrix from quaternion

Usage

q2dr(q)

Arguments

q quaternion

Value

derivative of rotation matrix

q2r

Convert quaternion to rotation matrix
http://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation

Description

Convert quaternion to rotation matrix http://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation

Usage

q2r(q)

Arguments

q quaternion

Value

rotation matrix

restore_coordinates *restore_coordinates*

Description

Restore coordinates from mean 0 and standard deviation 1 to their original distribution

Usage

```
restore_coordinates(X, mu, sigma)
```

Arguments

| | |
|-------|--|
| X | standardized point set as N x D matrix |
| mu | 1 x D vector of means |
| sigma | standard deviation |

Value

N X D matrix of unstandardized coordinates

rotx *rotx*

Description

Create a rotation matrix representing a rotation of theta radians about the x-axis

Usage

```
rotx(theta)
```

Arguments

| | |
|-------|------------------|
| theta | angle in radians |
|-------|------------------|

Value

a 3x3 rotation matrix

roty

roty

Description

Create a rotation matrix representing a rotation of theta radians about the y-axis

Usage

roty(theta)

Arguments

theta angle in radians

Value

a 3x3 rotation matrix

rotz

rotz

Description

Create a rotation matrix representing a rotation of theta radians about the z-axis

Usage

rotz(theta)

Arguments

theta angle in radians

Value

a 3x3 rotation matrix

scale_alpha_shape *scale_alpha_shape*

Description

Uniformly scale an alpha-shape. Note that this computes the alpha-shape of the scaled point set associated with the input alpha-shape.

Usage

```
scale_alpha_shape(as, s)
```

Arguments

| | |
|----|---|
| as | an alpha-shape object of class ashape3d |
| s | scaling factor |

Value

an object of class ashape3d

shape_features_3d *shape_features_3d*

Description

Compute shape features of a 3D alpha-shape object

Usage

```
shape_features_3d(as)
```

Arguments

| | |
|----|---|
| as | an alpha-shape object of class ashape3d |
|----|---|

Details

Features are: - major.axis, minor.axis and least.axis: Lengths of the axes of the fitted ellipsoid - elongation: from 0 (line) to 1 (globular) - flatness: from 0 (flat) to 1 (spherical) - max.feret.diameter: Maximum Feret diameter - max.inscribed.radius: Radius of the largest inscribed sphere - sphericity: from 0 (not spherical) to 1 (perfect sphere) - concavity: fraction of the convex hull volume not in the object - volume - area: area of the surface of the alpha-shape - centre.x, centre.y and centre.z: coordinates of the geometric median of the points in the alpha-shape

Value

a named vector of numeric values or NULL if no non-singular vertices

sliced_Wd

sliced_Wd

Description

Compute sliced Wasserstein distance or kernel. Reference: Mathieu Carriere, Marco Cuturi, and Steve Oudot. Sliced Wasserstein kernel for persistence diagrams. In Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 664–673, 2017.

Usage

```
sliced_Wd(Dg1, Dg2, M = 10, sigma = 1, dimensions = NULL, return.dist = FALSE)
```

Arguments

| | |
|-------------|---|
| Dg1 | a persistence diagram as a $n_1 \times 3$ matrix where each row is a topological feature and the columns are dimension, birth and death of the feature. |
| Dg2 | another persistence diagram as a $n_2 \times 3$ matrix |
| M | number of slices (default: 10) |
| sigma | kernel bandwidth (default: 1) |
| dimensions | vector of the dimensions of the topological features to consider, if NULL (default) use all available dimensions |
| return.dist | logical (default: FALSE). Whether to return the kernel or distance value. |

Value

kernel or distance value

Examples

```
D1 <- matrix(c(0,0,0,1,1,0,0,0,1.5, 3.5,2,2.5,3, 4, 6), ncol = 3, byrow = FALSE)
D2 <- matrix(c(0,0,1,1,0, 0, 1.2, 2, 1.4, 3.2,4.6,6.5), ncol = 3, byrow = FALSE)
K <- sliced_Wd(Dg1 = D1, Dg2 = D2, M = 10, sigma = 1, return.dist = TRUE)
```

standardize_coordinates

standardize_coordinates

Description

Transform coordinates to have mean 0 and standard deviation 1

Usage

```
standardize_coordinates(X)
```

Arguments

x point set as N x D matrix

Value

a list of X: standardized matrix, mu: vector of means, sigma: standard deviation

| | |
|-----------|-----------|
| <i>tr</i> | <i>tr</i> |
|-----------|-----------|

Description

Compute the trace of a matrix

Usage

```
tr(x)
```

Arguments

x matrix

Value

trace of the matrix

| | |
|---------|----------------|
| wgmmreg | <i>wgmmreg</i> |
|---------|----------------|

Description

Rigid registration of two point sets by minimizing the Wasserstein distance between GMMs

Usage

```
wgmmreg(
  X,
  Y,
  CX,
  CY,
  wx = NULL,
  wy = NULL,
  maxIter = 200,
  subsample = NULL,
  tol = 1e-08
)
```

Arguments

| | |
|-----------|--|
| X | reference point set, a N x D matrix |
| Y | point set to transform, a M x D matrix, |
| CX | array of covariance matrices for each point in X |
| CY | array of covariance matrices for each point in Y |
| wx | (optional) vector of mixture weights for X. |
| wy | (optional) vector of mixture weights for Y. |
| maxIter | maximum number of iterations to perform (default: 200) |
| subsample | can be set either to a number in [0,1] representing the fraction of points to randomly select or a list with vector elements X and Y containing indices of points to select in X and Y |
| tol | tolerance for determining convergence (default: 1e-8) |

Value

a list of

- Y: transformed point set,
- R: rotation matrix,
- t: translation vector,
- c: final value of the cost function,
- converged: logical, whether the algorithm converged.

Examples

```
data.file1 <- system.file("test_data", "parasaurolophusA.txt", package = "LOMAR",
  mustWork = TRUE)
PS1 <- read.csv(data.file1, sep = '\t', header = FALSE)
data.file2 <- system.file("test_data", "parasaurolophusB.txt", package = "LOMAR",
  mustWork = TRUE)
C1 <- diag(0.1, ncol(PS1)) + jitter(0.01, amount = 0.01)
C1 <- replicate(nrow(PS1),C1)
```

```
PS2 <- read.csv(data.file2, sep = '\\t', header = FALSE)
C2 <- diag(0.1, ncol(PS2)) + jitter(0.01, amount = 0.01)
C2 <- replicate(nrow(PS2), C2)
transformation <- wgmmreg(PS1, PS2, C1, C2, subsample = 0.1, maxIter = 30, tol = 1e-4)
## Not run:
# Visualize registration outcome
library(rgl)
plot3d(PS1, col = "blue")
points3d(PS2, col = "green")
points3d(transformation[['Y']], col = "magenta")

## End(Not run)
```

Index

[apply_transformation](#), [3](#)
[ary2ps](#), [3](#)

[binning](#), [4](#)

[circle_hough_transform](#), [5](#)
[coloc_index](#), [6](#)
[costWd](#), [7](#)
[cpd](#), [7](#)
[crop_point_set](#), [9](#)

[denoise](#), [10](#)
[dist_to_boundary](#), [10](#)
[dist_to_line](#), [11](#)
[downsample](#), [11](#)

[find_elbow](#), [12](#)

[Gaussian_Wd](#), [12](#)
[get_kernel_matrix](#), [13](#)
[get_persistence_diagrams](#), [14](#)
[get_shape](#), [15](#)
[get_surface_area](#), [15](#)
[GMM_Wd](#), [16](#)
[gradientWd](#), [16](#)
[group_events](#), [17](#)

[icp](#), [18](#)
[idx2rowcol](#), [19](#)
[img2ps](#), [19](#)

[jrmpc](#), [20](#)

[local_densities](#), [22](#)
[locprec2cov](#), [23](#)
[locs2ps](#), [24](#)
[locs_from_csv](#), [24](#)

[multiple_registration](#), [25](#)

[point_sets_from_locs](#), [27](#)
[point_sets_from_tiffs](#), [29](#)
[points2img](#), [26](#)
[points_from_roi](#), [27](#)
[ps2ary](#), [30](#)
[pssk](#), [30](#)

[q2dr](#), [31](#)
[q2r](#), [31](#)

[restore_coordinates](#), [32](#)
[rotx](#), [32](#)
[roty](#), [33](#)
[rotz](#), [33](#)

[scale_alpha_shape](#), [34](#)
[shape_features_3d](#), [34](#)
[sliced_Wd](#), [35](#)
[standardize_coordinates](#), [35](#)

[tr](#), [36](#)

[wgmreg](#), [36](#)