Package ‘nucim’

June 10, 2021

Title Nucleome Imaging Toolbox

Version 1.0.11

Date 2021-06-09

Author Volker Schmid [aut, cre]

Maintainer Volker Schmid <`volker.schmid@lmu.de`>

biocViews

Depends R (>= 3.0.0), EBImage, bioimagetools (>= 1.1.4)

Imports fields, parallel, stringr

SystemRequirements tiff fftw libcurl openssl

Description Tools for 4D nucleome imaging.
Quantitative analysis of the 3D nuclear landscape recorded with super-resolved fluorescence microscopy.
See Volker J. Schmid, Marion Cremer, Thomas Cremer (2017) <doi:10.1016/j.ymeth.2017.03.013>.

License GPL-3

URL `https://bioimaginggroup.github.io/nucim/`

RoxygenNote 7.1.0

Suggests knitr, rmarkdown, R.rsp

VignetteBuilder knitr, R.rsp

BugReports `https://github.com/bioimaginggroup/nucim/issues`

NeedsCompilation no

Repository CRAN

Date/Publication 2021-06-10 04:40:05 UTC

R topics documented:

`barplot_with_interval` *Barplot with Intervals*

Description

Barplot with Intervals

Usage

```
barplot_with_interval(
  x,
  method = "minmax",
  qu = c(0, 1),
  ylim = NULL,
  horiz = FALSE,
  border = NA,
  ...
)
```

Arguments

`x`	matrix
`method`	method for intervals: "minmax" (default), "quantile" or "sd"
`qu`	vector of two quantiles for method="quantile
`ylim`	limits for y axis. Default:NULL is ylim=c(0,max(interval))
`horiz`	boolean: horizontal bars?
`border`	border parameter forwarded to barplot, default: NA is nor border
`...`	additional parameters forwarded to barplot

Value

plot

`barplot_with_interval_23`

Barplot with Intervals for two or three bars beside

Description

Barplot with Intervals for two or three bars beside

Usage

```
barplot_with_interval_23(x, method = "minmax", qu = c(0, 1), ylim = NULL, ...)
```

Arguments

`x`	array
`method`	method for intervals: "minmax" (default), "quantile" or "sd"
`qu`	vector of two quantiles for method="quantile
`ylim`	limits for y axis. Default:NULL is ylim=c(0,max(interval))
`...`	additional parameters forwarded to barplot

Value

plot

`class.neighbours` *Class neighbourhood distribution*

Description

Class neighbourhood distribution

Usage

```
class.neighbours(img, N, N.max = 7, cores = 1)
```

Arguments

`img`	Class image
`N`	which class
`N.max`	maximum class (default: 7)
`cores`	number of cores used in parallel (needs parallel package)

Value

vector of length N.max

`class.neighbours.folder`
class.neighbours.folder

Description

class.neighbours.folder

Usage

```
class.neighbours.folder(inputfolder, outputfolder, N = 7)
```

Arguments

`inputfolder`	Input folder
`outputfolder`	Output folder
`N`	Max class #'

Value

plots

`classify` *Classify DAPI*

Description

Classify DAPI

Usage

```
classify(blue, mask, N, beta = 0.1, z = 1/3, silent = TRUE)
```

Arguments

`blue`	DAPI channel (image)
`mask`	mask (image)
`N`	number of classes
`beta`	smoothing parameter used in potts model (default: 0.1)
`z`	scaling parameter: size of voxel in X-/Y-direction divided by the size of voxel in Z-direction (slice scaling parameter: size of voxel in X-/Y-direction divided by the size of voxel in Z-direction (slice thickness))
`silent`	boolean. Should algorithm be silent?

Value

image with classes

`classify.folder` *Classify DAPI*

Description

Classify DAPI

Usage

```
classify.folder(f, N, beta = 0.1, output = paste0("class", N), cores = 1)
```

Arguments

`f`	folder
`N`	number of classes
`beta`	beta parameter used in bioimagetools::segment()
`output`	output folder
`cores`	number of cores used in parallel (needs parallel package)

Value

results in "output" and "output"-n

`classify.single` *Classify DAPI*

Description

These functions are provided for compatibility with older version of the nucim package. They may eventually be completely removed.

Usage

```
classify.single(...)
```

Arguments

`...`	parameters for classify

Value

image with classes

`classify.table` *Count classes in classified image*

Description

Count classes in classified image

Usage

```
classify.table(class, N)
```

Arguments

`class`	classes image
`N`	number of classes

Value

table with number of voxels per class

`colors.in.classes` *Compute colors in classes distribution*

Description

Compute colors in classes distribution

Usage

```
colors.in.classes(
  classes,
  color1,
  color2 = NULL,
  mask = array(TRUE, dim(classes)),
  N = max(classes, na.rm = TRUE),
  type = "tresh",
  thresh1 = NULL,
  thresh2 = NULL,
  sd1 = 2,
  sd2 = 2,
  col1 = "green",
  col2 = "red",
  test = FALSE,
  plot = TRUE,
  beside = TRUE,
  ylim = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

`classes`	Image of classes
`color1`	Image of first color
`color2`	Image of second color
`mask`	Image mask
`N`	Maximum number of classes
`type`	Type of spot definition, see details
`thresh1`	Threshold for first color image
`thresh2`	Threshold for second color image
`sd1`	For automatic threshold, that is: mean(color1)+sd1*sd(color1)
`sd2`	For automatic threshold of color2
`col1`	Name of color 1
`col2`	Name of color 2

`test`	Compute tests: "Wilcoxon" for Wilcoxon rank-sum (Mann-Whitney U), chisq for Chi-squared test
`plot`	Plot barplots
`beside`	a logical value. If FALSE, the columns of height are portrayed as stacked bars, and if TRUE the columns are portrayed as juxtaposed bars.
`ylim`	limits for the y axis (plot)
`verbose`	verbose mode
`...`	additional plotting parameters

Details

Type of spot definitions: "thresh" or "t": Threshold based (threshold can be given by thresh1/2 or automatically derived) "voxel" or "v": Spots are given as binary voxel mask "intensity" or "i": Voxels are weighted with voxel intensity. Intensity is scaled to [0,1] after subtracting thresh1/2 (or automatic threshold)

Value

Table of classes with color 1 (and 2)

`colors.in.classes.folder`

Compute colors in classes distribution for folders

Description

Compute colors in classes distribution for folders

Usage

```
colors.in.classes.folder(
  path,
  color1,
  color2 = NULL,
  N = 7,
  type = "intensity",
  thresh1 = NULL,
  thresh2 = NULL,
  sd1 = 2,
  sd2 = 2,
  col1 = "green",
  col2 = "red",
  cores = 1,
  verbose = FALSE
)
```

Arguments

`path`	Path to root folder
`color1`	Image of first color
`color2`	Image of second color
`N`	Maximum number of classes
`type`	Type of spot definition, see details
`thresh1`	Threshold for first color image
`thresh2`	Threshold for second color image
`sd1`	For automatic threshold, that is: mean(color1)+sd1*sd(color1)
`sd2`	For automatic threshold of color2
`col1`	Name of color 1
`col2`	Name of color 2
`cores`	Number of cores used in parallel, cores=1 implies no parallelization
`verbose`	verbose mode

Value

Results are in folder colorsinclasses

---

compute.distance2border
Compute distance to border of classes

Description

Compute distance to border of classes

Usage

```
compute.distance2border(
  f,
  color,
  N,
  from.spots = FALSE,
  output = "dist2border",
  cores = 1
)
```

Arguments

`f`	folder of classes images
`color`	folder of color images ("spots-"color for spots images)
`N`	which class
`from.spots`	Logical.
`output`	output folder
`cores`	number of parallel cores which can be used

Value

images in output"-"color"-"N

`dapimask` *Mask DAPI in kernel*

Description

Mask DAPI in kernel

Usage

```
dapimask(
  img,
  size = NULL,
  voxelsize = NULL,
  thresh = "auto",
  silent = TRUE,
  cores = 1
)
```

Arguments

`img`	DAPI channel image (3d)
`size`	size of img in microns
`voxelsize`	size of voxel in microns
`thresh`	threshold for intensity. Can be "auto": function will try to find automatic threshold
`silent`	Keep silent?
`cores`	number of cores available for parallel computing

Value

mask image, array with same dimension as img.

`dapimask.file` *Automatic DAPI mask segmentation for files*

Description

Automatic DAPI mask segmentation for files

Usage

```
dapimask.file(
  file,
  folder = "blue",
  voxelsize = NULL,
  size = NULL,
  silent = FALSE,
  cores = 1
)
```

Arguments

`file`	file to read
`folder`	with
`voxelsize`	real size of voxel (in microns), if NULL (default), look in folder XYZmic
`size`	real size of image (in microns), if NULL (default), look in folder XYZmic
`silent`	Keep silent?
`cores`	Number of cores available for parallel computing

Value

nothing, DAPI mask image will be saved to dapimask/

`dapimask.folder` *Automatic DAPI mask segmentation for folder*

Description

Automatic DAPI mask segmentation for folder

Usage

```
dapimask.folder(
  path,
  folder = "blue",
  voxelsize = NULL,
  size = NULL,
  cores = 1
)
```

Arguments

`path`	path to folder with DAPI
`folder`	folder with DAPI images
`voxelsize`	real size of voxel (in microns), if NULL (default), look in folder XYZmic
`size`	real size of image (in microns), if NULL (default), look in folder XYZmic
`cores`	number of cores to use in parallel (need parallel package)

Value

nothing, results are in folder dapimask

`find.spots.file` *Detects spots for one file*

Description

Detects spots for one file

Usage

```
find.spots.file(
  file,
  dir,
  color,
  thresh = NULL,
  thresh.auto = FALSE,
  thresh.quantile = 0.9,
  filter = NULL,
  cores = 1
)
```

Arguments

`file`	file
`dir`	directory for results
`color`	which color, images have to be in folder with color name
`thresh`	threshold
`thresh.auto`	Logical. Find threshold automatically?
`thresh.quantile`	numeric. use simple
`filter`	2d-filter to use before spot detection
`cores`	number of cores to use in parallel (with parallel package only)

Value

spot images in spot-color/, number of spots as txt files in spot-color/

`find.spots.folder` *Detects spots*

Description

Detects spots

Usage

```
find.spots.folder(
  f,
  color,
  thresh = 1,
  thresh.auto = TRUE,
  filter = NULL,
  cores = 1
)
```

Arguments

`f`	path to folder
`color`	which color, images have to be in folder with color name
`thresh`	threshold
`thresh.auto`	Logical. Find threshold automatically?
`filter`	2d-filter to use before spot detection
`cores`	number of cores to use in parallel (with parallel package only)

Value

spot images in spot-color/, number of spots as txt files in spot-color/

`heatmap.color` *Heatmap colors for n classes*

Description

Heatmap colors for n classes

Usage

```
heatmap.color(n)
```

Arguments

`n`	number of colors.

Examples

```
barplot(8:1,col=heatmap.color(8))
```

`heatmap7` *Heatmap colors for 7 classes*

Description

Heatmap colors for 7 classes

Usage

```
heatmap7(...)
```

Arguments

`...`	parameters are ignored.

Examples

```
barplot(7:1,col=heatmap7())
```

`nearestClassDistances.folder` *Find all distances to next neighbour of all classes for folders*

Description

Find all distances to next neighbour of all classes for folders

Usage

```
nearestClassDistances.folder(
  path,
  N = 7,
  voxelsize = NULL,
  add = FALSE,
  cores = 1
)
```

Arguments

`path`	path to folder
`N`	number of classes, default: 7
`voxelsize`	real size of voxels (in microns), if NULL (default), look in folder XYZmic
`add`	if TRUE, only process images which have not been processed before (i.e. have been added to classN)
`cores`	number of cores to use in parallel (needs parallel package if cores>1)

Value

nothing, results are in folder distances in RData format

`plot_classify.folder` *Plot barplot for classified images in a folder*

Description

Plot barplot for classified images in a folder

Usage

```
plot_classify.folder(
  path,
  N = 7,
  cores = 1,
  col = grDevices::grey(0.7),
  method = "sd"
)
```

Arguments

`path`	path to folder
`N`	number of classes, default: 7
`cores`	number of cores to use in parallel (needs parallel package if cores>1)
`col`	color of bars, either one or a vector of hex RGB characters
`method`	method for error bars ("sd", "minmax", "quartile")

Value

plots

`plot_colors.in.classes.folder`

Plot for colors in classes distribution for folders

Description

Plot for colors in classes distribution for folders

Usage

```
plot_colors.in.classes.folder(path, col1 = "green", col2 = "red")
```

Arguments

`path`	path to folder
`col1`	color of channel 1
`col2`	color of channel 2

Value

plot

`plot_nearestClassDistances.folder`

Plots all distances to next neighbour of all classes for folders

---

Description

Plots all distances to next neighbour of all classes for folders

Usage

```
plot_nearestClassDistances.folder(
  path,
  N = 7,
  cores = 1,
  method = "quantile",
  qu = 0.01
)
```

Arguments

`path`	path to folder
`N`	number of classes, default: 7
`cores`	number of cores to use in parallel (needs parallel package if cores>1)
`method`	method for summarizing distances, either "min" or "quantile"
`qu`	quantile for method="quantile", default: 0.01

Value

plots

`splitchannel` *Split RGB channels*

Description

Split RGB channels

Usage

```
splitchannel(img, preprocess = TRUE)
```

Arguments

`img`	rgb image
`preprocess`	logical. Should preprocessing be applied?

Value

list with red, green, blue channels and size in microns.

`splitchannels` *Split RGB images into channels and pixel size information*

Description

These functions are provided for compatibility with older version of the nucim package. They may eventually be completely removed.

Usage

```
splitchannels(...)
```

Arguments

`...`	parameters for splitchannels.folder

Value

Nothing, folders red, green, blue and XYZmic include separate channels and pixel size information

---

`splitchannels.file` *Split channels into files and extracts size in microns*

Description

Split channels into files and extracts size in microns

Usage

```
splitchannels.file(file, channels, rgb.folder, normalize = FALSE)
```

Arguments

`file`	file name
`channels`	e.g. c("red","green","blue")
`rgb.folder`	folder with file
`normalize`	boolean. Should we try to do normalization?

Value

files in "./red/", "./green", "./blue" and "./XYZmic"

`splitchannels.folder` *Split RGB images into channels and pixel size information*

Description

Split RGB images into channels and pixel size information

Usage

```
splitchannels.folder(
  path,
  channels = c("red", "green", "blue"),
  rgb.folder = "rgb",
  normalize = FALSE,
  cores = 1
)
```

Arguments

`path`	Path to root folder
`channels`	Vector of channels in images
`rgb.folder`	Folder with RGB images
`normalize`	boolean. Should we try to do normalization
`cores`	Number of cores used in parallel, cores=1 implies no parallelization

Value

Nothing, folders red, green, blue and XYZmic include separate channels and pixel size information

Examples

```
splitchannels.folder("./")
```

`spots.combined` *Find spots using information from two channels*

Description

Find spots using information from two channels

Usage

```
spots.combined(
  red,
  green,
  mask,
  size = NULL,
  voxelsize = NULL,
  thresh.offset = 0.1,
  window = c(5, 5),
  min.sum.intensity = 2,
  max.distance = 0.5,
  use.brightest = FALSE,
  max.spots = NA,
  full.voxel = FALSE
)
```

Arguments

`red`	image
`green`	image
`mask`	image mask

`size`	size of img in microns
`voxelsize`	size of voxel in microns
`thresh.offset`	Thresh offset used in EBImage::thresh()
`window`	Half width and height of the moving rectangular window.
`min.sum.intensity`	spots smaller than min.sum.intensity are ignored
`max.distance`	use only spots with distance to other color spot smaller than max.distance
`use.brightest`	Logical; use only brightest in max.distance?
`max.spots`	maximum of spots (per channel), only when use brightest=TRUE
`full.voxel`	Logical; output contains full voxel instead of rgb intensities

Value

RGB image with spots will be written to output folder

`spots.combined.file` *Find spots using information from two channels*

Description

Find spots using information from two channels

Usage

```
spots.combined.file(
  file,
  size = NULL,
  voxelsize = NULL,
  folder = "./",
  thresh.offset = 0.1,
  min.sum.intensity = 2,
  max.distance = 0.5,
  use.brightest = FALSE,
  max.spots = 2,
  full.voxel = FALSE,
  output = "markers"
)
```

Arguments

`file`	File name
`size`	size of img in microns, if size and voxelsize are NULL, size is determined from folder XYZmic
`voxelsize`	size of voxel in microns

`folder`	Folder
`thresh.offset`	Thresh offset used in EBImage::thresh()
`min.sum.intensity`	spots smaller than min.sum.intensity are ignored
`max.distance`	use only spots with distance to other color spot smaller than max.distance
`use.brightest`	Logical; use only brightest in max.distance?
`max.spots`	maximum of spots (per channel), only when use brightest=TRUE
`full.voxel`	Logical; output contains full voxel instead of rgb intensities
`output`	output folder

Value

RGB image with spots will be written to output folder

`spots.combined.folder` *Find spots using information from two channels for folder*

Description

Find spots using information from two channels for folder

Usage

```
spots.combined.folder(
  path,
  size = NULL,
  voxelsize = NULL,
  thresh.offset = 0.1,
  min.sum.intensity = 2,
  max.distance = 0.5,
  use.brightest = FALSE,
  max.spots = 2,
  full.voxel = FALSE,
  output = "markers",
  cores = 1
)
```

Arguments

`path`	path to folder
`size`	size of img in microns, if size and voxelsize are NULL, size is determined from folder XYZmic
`voxelsize`	size of voxel in microns
`thresh.offset`	Thresh offset used in EBImage::thresh()

`min.sum.intensity`	spots smaller than min.sum.intensity are ignored
`max.distance`	use only spots with distance to other color spot smaller than max.distance
`use.brightest`	Logical; use only brightest in max.distance?
`max.spots`	maximum of spots (per channel), only when use brightest=TRUE
`full.voxel`	Logical; output contains full voxel instead of rgb intensities
`output`	output folder
`cores`	number of cores we can use of parallel computing (needs parallel package if cores>1)

Value

RGB image with spots will be written to output folder

`t_colors.in.classes.folder`

Test for colors in classes distribution for folders

Description

Test for colors in classes distribution for folders

Usage

```
t_colors.in.classes.folder(path, test = "Wilcoxon")
```

Arguments

`path`	path to folder
`test`	"Wilcoxon", "wilcox" or "U" for Wilcoxon rank-sum (Mann-Whitney U), chisq for Chi-squared test

Value

test results

Index