Package ‘biopixR’

March 25, 2024

Title Extracting Insights from Biological Images

Version 0.2.4

Description Combines the 'magick' and 'imager' packages to streamline image analysis, focusing on feature extraction and quantification from biological images, especially microparticles. By providing high throughput pipelines and clustering capabilities, 'biopixR' facilitates efficient insight generation for researchers (Schneider J. et al. (2019) <doi:10.21037/jlpm.2019.04.05>).

License LGPL (>= 3)

VignetteBuilder knitr

BuildVignettes true

Depends R (>= 4.2.0), data.table, imager, magick, tcltk, foreach

Suggests knitr, rmarkdown, doParallel, kohonen

Encoding UTF-8

RoxygenNote 7.2.3

LazyData true

LazyLoad yes

NeedsCompilation no

Language en-US

URL `https://github.com/Brauckhoff/biopixR`

BugReports `https://github.com/Brauckhoff/biopixR/issues`

Author Tim Brauckhoff [aut, cre] (<`https://orcid.org/0009-0002-0142-7017`>), Stefan Roediger [ctb] (<`https://orcid.org/0000-0002-1441-6512`>), Coline Kieffer [ctb]

Maintainer Tim Brauckhoff <`Tim.Brauckhoff@b-tu.de`>

Repository CRAN

Date/Publication 2024-03-25 19:10:09 UTC

R topics documented:

`adaptiveInterpolation` *Connects LineEnd with the nearest labeled region*

Description

Function scans an increasing radius around a line end and connects it with the nearest labeled region.

Usage

```
adaptiveInterpolation(
  end_points_df,
  diagonal_edges_df,
  clean_lab_df,
  lineends_cimg,
  radius = 5
)
```

Arguments

`end_points_df`	data frame with the coordinates of all line ends. can be obtained with `image_morphology`.
`diagonal_edges_df`	data frame with coordinates of diagonal line ends. can also be obtained by `image_morphology`.
`clean_lab_df`	data of type 'data.frame', containing the x, y and value information of every labeled region in an image. (only the edges should be labeled)
`lineends_cimg`	image with dimensions of the image with discontinuous edges. just for giving the dimensions of the output matrix.
`radius`	maximal radius that should be scanned for another cluster

Details

This function is intended to be part of the fillLineGaps function, which does the thresholding and line end detection preprocessing. The adaptiveInterpolation creates a matrix in the dimensions of the original image. At the beginning there are only background values (0) = black image. The function then searches for LineEnds and looks for a given radius around this line end for the nearest labeled region. The own cluster of the line end is of course not considered as nearest neighbor.If another cluster is found, the interpolatePixels function is used to connect the line end to the found cluster. This means that specified pixels of the matrix are transformed to a foreground value of (1). The diagonal line ends get a special treatment, because for the labeling function,7 the diagonal pixels are always treated as a separate cluster, which makes them difficult to reconnect. To deal with this problem, diagonal line ends ignore not only their cluster, but also the cluster of the direct neighbor. Thereafter, the same procedure as before is repeated, where pixel values are changed according to the interpolatePixel function.

Value

binary matrix that can be applied as an overlay, for example with `imager.combine` to fill the gaps between line ends.

Examples

```
# creating an artificial binary image
mat <- matrix(0, 8, 8)
mat[3, 1:2] <- 1
mat[4, 3] <- 1
mat[7:8, 3] <- 1
mat[5, 6:8] <- 1
mat_cimg <- as.cimg(mat)

# preprocessing / LineEnd detection / labeling  (done in fillLineGaps)
mat_cimg_m <- mirror(mat_cimg, axis = "x")
mat_magick <- cimg2magick(mat_cimg)
lineends <- image_morphology(mat_magick, "HitAndMiss", "LineEnds")
diagonalends <- image_morphology(mat_magick, "HitAndMiss", "LineEnds:2>")
lineends_cimg <- magick2cimg(lineends)
diagonalends_cimg <- magick2cimg(diagonalends)
end_points <- which(lineends_cimg == TRUE, arr.ind = TRUE)
end_points_df <- as.data.frame(end_points)
colnames(end_points_df) <- c("x", "y", "dim3", "dim4")
diagonal_edges <- which(diagonalends_cimg == TRUE, arr.ind = TRUE)
diagonal_edges_df <- as.data.frame(diagonal_edges)
colnames(diagonal_edges_df) <- c("x", "y", "dim3", "dim4")
lab <- label(mat_cimg_m)
df_lab <- as.data.frame(lab) |> subset(value > 0)
alt_x <- list()
alt_y <- list()
alt_value <- list()
for (g in 1:nrow(df_lab)) {
  if (mat_cimg_m[df_lab$x[g], df_lab$y[g], 1, 1] == 1) {
    alt_x[g] <- df_lab$x[g]
    alt_y[g] <- df_lab$y[g]
    alt_value[g] <- df_lab$value[g]
  }
}
clean_lab_df <- data.frame(
  x = unlist(alt_x),
  y = unlist(alt_y),
  value = unlist(alt_value)
)

# actual function
overlay <- adaptiveInterpolation(
  end_points_df,
  diagonal_edges_df,
  clean_lab_df,
  mat_cimg
)
parmax(list(mat_cimg_m, as.cimg(overlay$overlay))) |> plot()
```

---

beads *Image of microbeads*

Description

This fluorescence image, formatted as 'cimg' with dimensions of 117 x 138 pixels, shows microbeads. With a single color channel, the image provides an ideal example for in-depth analysis of microbead structures.

Usage

```
beads
```

Format

The image was imported using imager and is therefore of class: "cimg" "imager_array" "numeric"

Details

Dimensions: width - 117; height - 138; depth - 1; channel - 1

References

The image was provided by Coline Kieffer.

Examples

```
data(beads)
plot(beads)
```

`changePixelColor` *Change the color of pixels*

Description

Can be used to change the color of specified pixels in an image. The coordinates of the pixels are needed to colorize them.

Usage

```
changePixelColor(img, coords, color = "purple", visualize = FALSE)
```

Arguments

`img`	image (import by `load.image`)
`coords`	coordinates specifying which pixels to be colored (should be a X|Y Data frame (first column: X; second column: Y)).
`color`	color with which to replace specified pixels. can be either a an RGB triplet or one of the colors listed by `colors`.
`visualize`	if TRUE the resulting image gets plotted

Value

cimg with changed colors at desired positions and plot of the cimg

References

https://CRAN.R-project.org/package=countcolors

Examples

```
coordinates <- objectDetection(beads)
changePixelColor(beads, coordinates$coordinates)
```

`droplets` *Droplets containing microbeads*

Description

The image displays a water-oil emulsion with droplets observed through brightfield microscopy. It is formatted as 'cimg' and sized at 151 × 112 pixels. The droplets vary in size, and some contain microbeads, which adds complexity. Brightfield microscopy enhances the contrast between water and oil, revealing the droplet arrangement.

Usage

```
droplets
```

Format

The image was imported using imager and is therefore of class: "cimg" "imager_array" "numeric"

Details

Dimensions: width - 151; height - 112; depth - 1; channel - 1

References

The image was provided by Coline Kieffer.

Examples

```
data(droplets)
plot(droplets)
```

`droplet_beads` *Image of microbeads in luminescence channel*

Description

The image shows red fluorescence rhodamine microbeads measuring 151 x 112 pixels. The fluorescence channel was used to obtain the image, resulting in identical dimensions and positions of the beads as in the original image (droplets).

Usage

```
droplet_beads
```

Format

The image was imported using imager and is therefore of class: "cimg" "imager_array" "numeric"

Details

Dimensions: width - 151; height - 112; depth - 1; channel - 3

References

The image was provided by Coline Kieffer.

Examples

```
data(droplet_beads)
plot(droplet_beads)
```

`edgeDetection` *Canny edge detector*

Description

Adapted code from the imager `cannyEdges`) function without the usage of dplyr and purrr. If the threshold parameters are missing, they are determined automatically using a k-means heuristic. Use the alpha parameter to adjust the automatic thresholds up or down. The thresholds are returned as attributes. The edge detection is based on a smoothed image gradient with a degree of smoothing set by the sigma parameter.

Usage

```
edgeDetection(img, t1, t2, alpha = 1, sigma = 2)
```

Arguments

`img`	input image
`t1`	threshold for weak edges (if missing, both thresholds are determined automatically)
`t2`	threshold for strong edges
`alpha`	threshold adjustment factor (default 1)
`sigma`	smoothing

Value

Object of class 'cimg', displaying detected edges.

References

https://CRAN.R-project.org/package=imager

Examples

```
edgeDetection(beads) |> plot()
```

fillLineGaps *Reconnecting discontinuous lines*

Description

The function attempts to fill in edge discontinuities in order to enable normal labeling and edge detection.

Usage

```
fillLineGaps(
  droplet.img,
  bead.img = NULL,
  threshold = "13%",
  alpha = 0.75,
  sigma = 0.1,
  radius = 5,
  iterations = 2,
  visualize = TRUE
)
```

Arguments

`droplet.img`	image that contains discontinuous lines like edges or contours
`bead.img`	image that contains objects that should be removed before before applying the fill algorithm
`threshold`	"in %" (from `threshold`)
`alpha`	threshold adjustment factor for edge detection (from `cannyEdges`)
`sigma`	smoothing (from `cannyEdges`)
`radius`	maximal radius that should be scanned for another cluster
`iterations`	how many times the algorithm should find line ends and reconnect them to their closest neighbor
`visualize`	if TRUE (default) a plot is displayed highlighting the added pixels in the original image

Details

The function pre-processes the image to enable the application of adaptiveInterpolation. Pre-processing involves thresholding, optional object removal, LineEnd and diagonal LineEnd detection, and labeling. The threshold should be set to allow for some remaining "bridge" pixels between gaps to facilitate reconnection. For more details about reconnection, please consult adaptiveInterpolation. Post-processing involves thinning the lines. When removing objects from an image, their coordinates are collected using the objectDetection function. Next, the pixels of the detected objects are nullified in the original image, allowing the algorithm to proceed without the objects.

Value

image with continuous edges (closed gaps)

Examples

```
fillLineGaps(droplets)
```

`imgPipe` *Image analysis pipeline*

Description

This function serves as a pipeline that integrates tools for complete start-to-finish image analysis. It enables the handling of images from different channels, including the analysis of dual-color microbeads. This approach simplifies the workflow, providing a straightforward method to analyze complex image data.

Usage

```
imgPipe(
  img1 = img,
  color1 = "color1",
  img2 = NULL,
  color2 = "color2",
  img3 = NULL,
  color3 = "color3",
  alpha = 1,
  sigma = 2,
  sizeFilter = TRUE,
  upperlimit = "auto",
  lowerlimit = "auto",
  proximityFilter = TRUE,
  radius = "auto",
  parallel = FALSE
)
```

Arguments

`img1`	image (import by `load.image`)
`color1`	name of color in img1
`img2`	image (import by `load.image`)
`color2`	name of color in img2
`img3`	image (import by `load.image`)
`color3`	name of color in img3
`alpha`	threshold adjustment factor

`sigma`	smoothing
`sizeFilter`	applying sizeFilter function (default - TRUE)
`upperlimit`	highest accepted object size (only needed if sizeFilter = TRUE)
`lowerlimit`	smallest accepted object size (when 'auto' both limits are calculated by using the mean and the standard deviation)
`proximityFilter`	applying proximityFilter function (default - TRUE)
`radius`	distance from one center in which no other centers are allowed (in pixels)
`parallel`	if TRUE uses multiple cores (75 %) to process results

Value

list of 3 to 4 objects:

1. summary of all the microbeads in the image
2. detailed information about every single bead
3. (optional) result for every individual color
4. unfiltered coordinates of img1

Examples

```
result <- imgPipe(beads,
  alpha = 1, sigma = 2, upperlimit = 150,
  lowerlimit = 50
  )
plot(beads)
with(
  result$detailed,
  points(
    result$detailed$x,
    result$detailed$y,
    col = "darkgreen",
    pch = 19
    )
  )
```

`interactive_objectDetection`
Interactive object detection

Description

This function uses the objectDetection function to visualize the detected objects at varying threshold an smoothing parameters.

Usage

```
interactive_objectDetection(img, resolution = 0.1, return_param = FALSE)
```

Arguments

`img`	image (preferred import: `load.image`)
`resolution`	resolution of slider
`return_param`	used to define the final parameter values for alpha and sigma printed in the console (TRUE or FALSE).

Value

values of alpha and sigma

References

https://CRAN.R-project.org/package=magickGUI

Examples

```
if (interactive()) {
  interactive_objectDetection(beads)
}
```

`interpolatePixels` *Pixel Interpolation*

Description

Connects two points in a matrix, array, or an image.

Usage

```
interpolatePixels(row1, col1, row2, col2)
```

Arguments

`row1`	first row: together with col1 coordinate for the first point
`col1`	first column: together with row1 coordinate for the first point
`row2`	second row: together with col2 coordinate for the second point
`col2`	second column: together with row2 coordinate for the second point

Value

matrix containing the coordinates to connect the two input points

Examples

```
test <- matrix(0, 4, 4)
test[1, 1] <- 1
test[3, 4] <- 1
link <- interpolatePixels(1, 1, 3, 4)
test[link] <- 1
```

`objectDetection` *Object detection*

Description

This function identifies objects in an image using edge detection and labeling, gathering the coordinates and centers of the identified objects. The edges of detected objects are then highlighted for easy recognition.

Usage

```
objectDetection(img, alpha = 1, sigma = 2)
```

Arguments

`img`	image (import by `load.image`)
`alpha`	threshold adjustment factor
`sigma`	smoothing

Value

list of 4 objects:

1. data frame of labeled region with the central coordinates
2. all coordinates that are in labeled regions
3. size of labeled objects
4. image were object edges (purple) and detected centers (green) are colored

Examples

```
res_objectDetection <- objectDetection(beads, alpha = 1, sigma = 2)
res_objectDetection$marked_beads |> plot()
```

proximityFilter *Proximity-based exclusion*

Description

To detect objects within a defined range of one another, it is necessary to calculate their centers to determine proximity. Pairs that are too close will be discarded. (Input can be obtained by objectDetection function)

Usage

```
proximityFilter(centers, coordinates, radius = "auto")
```

Arguments

`centers`	center coordinates of objects (mx|my|value data frame)
`coordinates`	all coordinates of the objects (x|y|value data frame)
`radius`	distance from one center in which no other centers are allowed (in pixels)

Value

list of 3 objects:

1. center coordinates of remaining objects
2. all coordinates of remaining objects
3. size of remaining objects

Examples

```
res_objectDetection <- objectDetection(beads, alpha = 1, sigma = 2)
res_proximityFilter <- proximityFilter(
  res_objectDetection$centers,
  res_objectDetection$coordinates,
  radius = "auto"
  )
changePixelColor(
  beads,
  res_proximityFilter$coordinates,
  color = "darkgreen",
  visualize = TRUE
  )
```

`resultAnalytics` *Image Summary*

Description

Extracts all important information of the remaining microbeads. This function summarizes the data obtained by previous functions: objectDetection, proximityFilter and sizeFilter. Provides information like amount, intensity, size and density.

Usage

```
resultAnalytics(unfiltered, coordinates, size, img, parallel = FALSE)
```

Arguments

`unfiltered`	all coordinates from every object before applying filter functions
`coordinates`	all filtered coordinates of the objects (x|y|value data frame)
`size`	size of the objects
`img`	image (import by `load.image`)
`parallel`	if TRUE uses multiple cores (75 %) to process results

Value

list of 2 objects:

1. summary of all the microbeads in the image
2. detailed information about every single bead

Examples

```
res_objectDetection <- objectDetection(beads, alpha = 1, sigma = 2)
res_sizeFilter <- sizeFilter(
  res_objectDetection$centers,
  res_objectDetection$coordinates,
  lowerlimit = 50, upperlimit = 150
  )
res_proximityFilter <- proximityFilter(
  res_sizeFilter$centers,
  res_objectDetection$coordinates,
  radius = "auto"
  )
res_resultAnalytics <- resultAnalytics(
  unfiltered = res_objectDetection$coordinates,
  coordinates = res_proximityFilter$coordinates,
  size = res_proximityFilter$size,
  img = beads
  )
plot(beads)
```

```
with(
  res_objectDetection$centers,
  points(
    res_objectDetection$centers$mx,
    res_objectDetection$centers$my,
    col = "red",
    pch = 19
    )
  )
with(
  res_resultAnalytics$detailed,
  points(
    res_resultAnalytics$detailed$x,
    res_resultAnalytics$detailed$y,
    col = "darkgreen",
    pch = 19
    )
  )
```

sizeFilter *Size-based exclusion*

Description

Calculates the size of the objects in an image and discards objects based on a lower and an upper size limit. (Input can be obtained by objectDetection function)

Usage

```
sizeFilter(centers, coordinates, lowerlimit = "auto", upperlimit = "auto")
```

Arguments

`centers`	center coordinates of objects (needs to include 'value' representing the center number)
`coordinates`	all coordinates of the objects (x|y|value data frame)
`lowerlimit`	smallest accepted object size (when 'auto' both limits are calculated by using the mean and the standard deviation)
`upperlimit`	highest accepted object size

Value

list of 3 objects:

1. remaining centers after discarding according to size
2. remaining coordinates after discarding according to size
3. size of remaining objects

Examples

```
res_objectDetection <- objectDetection(beads, alpha = 1, sigma = 2)
res_sizeFilter <- sizeFilter(
  centers = res_objectDetection$centers,
  coordinates = res_objectDetection$coordinates,
  lowerlimit = 50, upperlimit = 150
  )
changePixelColor(
  beads,
  res_sizeFilter$coordinates,
  color = "darkgreen",
  visualize = TRUE
  )
```

Index