Package ‘biopixR’

November 11, 2024

Title Extracting Insights from Biological Images

Version 1.2.0

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), imager, magick

Imports data.table, cluster

Suggests tcltk, knitr, rmarkdown, doParallel, kohonen, imagerExtra, GPareto, foreach

Encoding UTF-8

RoxygenNote 7.3.2

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 <`brauctile@disroot.org`>

Repository CRAN

Date/Publication 2024-11-11 14:20:05 UTC

Contents

`adaptiveInterpolation` *Connects Line Ends with the nearest labeled region*

Description

The 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,
  img,
  radius = 5
)
```

Arguments

`end_points_df`	`data.frame` with the coordinates of all line ends (can be obtained by using `image_morphology`)
`diagonal_edges_df`	`data.frame` with coordinates of diagonal line ends (can also be obtained by using `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)
`img`	image providing the dimensions of the output matrix (import by `importImage`)
`radius`	maximal radius that should be scanned for another cluster

Details

This function is designed to be part of the `fillLineGaps` function, which performs the thresholding and line end detection preprocessing. The `adaptiveInterpolation` generates a matrix with dimensions matching those of the original image. Initially, the matrix contains only background values (0) corresponding to a black image. The function then searches for line ends and identifies the nearest labeled region within a given radius of the line end. It should be noted that the cluster of the line end in question is not considered a nearest neighbor. In the event that another cluster is identified, the `interpolatePixels` function is employed to connect the line end to the aforementioned cluster. This entails transforming the specified pixels of the matrix to a foreground value of (1). It is important to highlight that diagonal line ends receive a special treatment, as they are always treated as a separate cluster by the labeling function. This makes it challenging to reconnect them. To address this issue, diagonal line ends not only ignore their own cluster but also that of their direct neighbor. Thereafter, the same procedure is repeated, with pixel values being changed according to the `interpolatePixels` 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)
plot(mat_cimg)

# 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 seq_len(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)
```

`beads_large1` *Image of microbeads*

Description

This fluorescence image, formatted as 'cimg' with dimensions of 492 x 376 pixels, shows microbeads. With a single color channel, the image provides an ideal example for in-depth analysis of microbead structures. The image's larger size encompasses a greater number of microbeads, offering a broader range of experimental outcomes for examination.

Usage

```
beads_large1
```

Format

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

Details

Dimensions: width - 492; height - 376; depth - 1; channel - 1

References

The image was provided by Coline Kieffer.

Examples

```
data(beads_large1)
plot(beads_large1)
```

`beads_large2` *Image of microbeads*

Description

This fluorescence image, formatted as 'cimg' with dimensions of 1384 x 1032 pixels, shows microbeads. With a single color channel, the image provides an ideal example for in-depth analysis of microbead structures. The image's larger size encompasses a greater number of microbeads, offering a broader range of experimental outcomes for examination.

Usage

```
beads_large2
```

Format

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

Details

Dimensions: width - 1384; height - 1032; depth - 1; channel - 3

References

The image was provided by Coline Kieffer.

Examples

```
data(beads_large2)
plot(beads_large2)
```

`changePixelColor` *Change the color of pixels*

Description

The function allows the user to alter the color of a specified set of pixels within an image. In order to achieve this, the coordinates of the pixels in question must be provided.

Usage

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

Arguments

`img`	image (import by `importImage`)
`coordinates`	specifying which pixels to be colored (should be a xly data frame).
`color`	color to be applied to specified pixels: • color from the list of colors defined by `colors` • object of class factor
`visualize`	if TRUE the resulting image gets plotted

Value

Object of class 'cimg' with changed colors at desired positions.

References

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

Examples

```
coordinates <-
  objectDetection(beads,
                  method = 'edge',
                  alpha = 1,
                  sigma = 0)
changePixelColor(
  beads,
  coordinates$coordinates,
  color = factor(coordinates$coordinates$value),
  visualize = TRUE
)
```

`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`	image (import by `importImage`)
`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 (default 2)

Value

Object of class 'cimg', displaying detected edges.

References

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

Examples

```
edgeDetection(beads, alpha = 0.5, sigma = 0.5) |> 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(
  contours,
  objects = NULL,
  threshold = "13%",
  alpha = 1,
  sigma = 2,
  radius = 5,
  iterations = 2,
  visualize = TRUE
)
```

Arguments

`contours`	image that contains discontinuous lines like edges or contours
`objects`	image that contains objects that should be removed before applying the fill algorithm
`threshold`	"in %" (from `threshold`)
`alpha`	threshold adjustment factor for edge detection (from `edgeDetection`)
`sigma`	smoothing (from `edgeDetection`)
`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 in order to enable the implementation of the `adaptiveInterpolation` function. The pre-processing stage encompasses a number of operations, including thresholding, the optional removal of objects, the detection of line ends and diagonal line ends, and the labeling of pixels. The threshold should be set to allow for the retention of some "bridge" pixels between gaps, thus facilitating the subsequent process of reconnection. For further details regarding the process of reconnection, please refer to the documentation on `adaptiveInterpolation`. The subsequent post-processing stage entails the reduction of line thickness in the image. With regard to the possibility of object removal, the coordinates associated with these objects are collected using the `objectDetection` function. Subsequently, the pixels of the detected objects are set to null in the original image, thus allowing the algorithm to proceed without the objects.

Value

Image with continuous edges (closed gaps).

Examples

```
fillLineGaps(droplets)
```

`haralickCluster` *k-medoids clustering of images according to the Haralick features*

Description

This function performs k-medoids clustering on images using Haralick features, which describe texture. By evaluating contrast, correlation, entropy, and homogeneity, it groups images into clusters with similar textures. K-medoids is chosen for its outlier resilience, using actual images as cluster centers. This approach simplifies texture-based image analysis and classification.

Usage

```
haralickCluster(path)
```

Arguments

`path`	directory path to folder with images to be analyzed

Value

`data.frame` containing file names, md5sums and cluster number.

References

https://cran.r-project.org/package=radiomics

Examples

```
  path2dir <- system.file("images", package = 'biopixR')
  result <- haralickCluster(path2dir)
  print(result)
```

`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, for example the analysis of dual-color micro particles. 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",
  method = "edge",
  alpha = 1,
  sigma = 2,
  sizeFilter = FALSE,
  upperlimit = "auto",
  lowerlimit = "auto",
  proximityFilter = FALSE,
  radius = "auto"
)
```

Arguments

`img1`	image (import by `importImage`)
`color1`	name of color in img1
`img2`	image (import by `importImage`)
`color2`	name of color in img2
`img3`	image (import by `importImage`)
`color3`	name of color in img3
`method`	choose method for object detection ('edge' / 'threshold') (from `objectDetection`)
`alpha`	threshold adjustment factor (numeric / 'static' / 'interactive' / 'gaussian') (from `objectDetection`)
`sigma`	smoothing (numeric / 'static' / 'interactive' / 'gaussian') (from `objectDetection`)
`sizeFilter`	applying `sizeFilter` function (default - FALSE)
`upperlimit`	highest accepted object size (numeric / 'auto') (only needed if sizeFilter = TRUE)

`lowerlimit`	smallest accepted object size (numeric / 'auto') (only needed if sizeFilter = TRUE)
`proximityFilter`	applying `proximityFilter` function (default - FALSE)
`radius`	distance from one object in which no other centers are allowed (in pixels) (only needed if proximityFilter = TRUE)

Value

list of 2 to 3 objects:

- Summary of all the objects in the image.
- Detailed information about every single object.
- (optional) Result for every individual color.

See Also

`objectDetection()`, `sizeFilter()`, `proximityFilter()`, `resultAnalytics()`

Examples

```
result <- imgPipe(
  beads,
  alpha = 1,
  sigma = 2,
  sizeFilter = TRUE,
  upperlimit = 150,
  lowerlimit = 50
  )

# Highlight remaining microparticles
plot(beads)
with(
  result$detailed,
  points(
    result$detailed$x,
    result$detailed$y,
    col = "darkgreen",
    pch = 19
    )
  )
```

`importImage` *Import an Image File*

Description

This function is a wrapper to the `load.image` and `image_read` functions, and imports an image file and returns the image as a 'cimg' object. The following file formats are supported: TIFF, PNG, JPG/JPEG, and BMP. In the event that the image in question contains an alpha channel, that channel is omitted.

Usage

```
importImage(path2file)
```

Arguments

`path2file`	path to file

Value

An image of class 'cimg'.

Examples

```
path2img <- system.file("images/beads_large1.bmp", package = 'biopixR')
img <- importImage(path2img)
img |> plot()

path2img <- system.file("images/beads_large2.png", package = 'biopixR')
img <- importImage(path2img)
img |> plot()
```

`interactive_objectDetection`

Interactive object detection

Description

This function uses the `objectDetection` function to visualize the detected objects at varying input parameters.

Usage

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

Arguments

`img`	image (import by `importImage`)
`resolution`	resolution of slider
`return_param`	if TRUE the final parameter values for alpha and sigma are printed to the console (TRUE | FALSE)

Details

The function provides a graphical user interface (GUI) that allows users to interactively adjust the parameters for object detection:

- **Alpha:** Controls the threshold adjustment factor for edge detection.
- **Sigma:** Determines the amount of smoothing applied to the image.
- **Scale:** Adjusts the scale of the displayed image.

The GUI also includes a button to switch between two detection methods:

- **Edge Detection:** Utilizes the `edgeDetection` function. The alpha parameter acts as a threshold adjustment factor, and sigma controls the smoothing.
- **Threshold Detection:** Applies a thresholding method, utilizing `SPE` for background reduction and the `threshold` function. (No dependency on alpha or sigma!)

Value

Values of alpha, sigma and the applied method.

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`	row index for the first point
`col1`	column index for the first point
`row2`	row index for the second point
`col2`	column index for the second point

Value

Matrix containing the coordinates to connect the two input points.

Examples

```
# Simulate two points in a matrix
test <- matrix(0, 4, 4)
test[1, 1] <- 1
test[3, 4] <- 1
as.cimg(test) |> plot()

# Connect them with each other
link <- interpolatePixels(1, 1, 3, 4)
test[link] <- 1
as.cimg(test) |> plot()
```

`objectDetection` *Object detection*

Description

This function identifies objects in an image using either edge detection or thresholding methods. It gathers the coordinates and centers of the identified objects, highlighting the edges or overall coordinates for easy recognition.

Usage

```
objectDetection(img, method = "edge", alpha = 1, sigma = 2, vis = TRUE)
```

Arguments

`img`	image (import by `importImage`)
`method`	choose method for object detection ('edge' / 'threshold')
`alpha`	threshold adjustment factor (numeric / 'static' / 'interactive' / 'gaussian') (only needed for 'edge')
`sigma`	smoothing (numeric / 'static' / 'interactive' / 'gaussian') (only needed for 'edge')
`vis`	creates image were object edges/coordinates (purple) and detected centers (green) are highlighted (TRUE | FALSE)

Details

The `objectDetection` function provides several methods for calculating the alpha and sigma parameters, which are critical for edge detection:

1. **Input of a Numeric Value:**
    - Users can directly input numeric values for alpha and sigma, allowing for precise control over the edge detection parameters.

2. **Static Scanning:**
   - When both alpha and sigma are set to "static", the function systematically tests all possible combinations of these parameters within the range (alpha: 0.1 - 1.5, sigma: 0 - 2). This exhaustive search helps identify the optimal parameter values for the given image. (Note: takes a lot of time)
3. **Interactive Selection:**
   - Setting the alpha and sigma values to "interactive" initiates a Tcl/Tk graphical user interface (GUI). This interface allows users to adjust the parameters interactively, based on visual feedback. To achieve optimal results, the user must input the necessary adjustments to align the parameters with the specific requirements of the image. The user can also switch between the methods through the interface.
4. **Multi-Objective Optimization:**
   - For advanced parameter optimization, the function `easyGParetoptim` will be utilized for multi-objective optimization using Gaussian process models. This method leverages the 'GPareto' package to perform the optimization. It involves building Gaussian Process models for each objective and running the optimization to find the best parameter values.

Value

list of 3 objects:

- `data.frame` of labeled regions with the central coordinates (including size information).
- All coordinates that are in labeled regions.
- Image where object edges/coordinates (purple) and detected centers (green) are colored.

Examples

```
res_objectDetection <- objectDetection(beads,
                                       method = 'edge',
                                       alpha = 1,
                                       sigma = 0)
res_objectDetection$marked_objects |> plot()

res_objectDetection <- objectDetection(beads,
                                       method = 'threshold')
res_objectDetection$marked_objects |> plot()
```

`proximityFilter` *Proximity-based exclusion*

Description

In order to identify objects within a specified proximity, it is essential to calculate their respective centers, which serve to determine their proximity. Pairs that are in close proximity will be discarded. (Input can be obtained by `objectDetection` function)

Usage

```
proximityFilter(centers, coordinates, radius = "auto", elongation = 2)
```

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) (numeric / 'auto')
`elongation`	factor by which the radius should be multiplied to create the area of exclusion (default 2)

Details

The automated radius calculation in the `proximityFilter` function is based on the presumption of circular-shaped objects. The radius is calculated using the following formula:

$$\sqrt{\frac{A}{\pi}}$$

where A is the area of the detected objects. The function will exclude objects that are too close by extending the calculated radius by one radius length beyond the assumed circle, effectively doubling the radius to create an exclusion zone. Therefore the elongation factor is set to 2 by default, with one radius covering the object and an additional radius creating the area of exclusion.

Value

list of 2 objects:

- Center coordinates of remaining objects.
- All coordinates of remaining objects.

Examples

```
res_objectDetection <- objectDetection(beads,
                                       alpha = 1,
                                       sigma = 0)
res_proximityFilter <- proximityFilter(
  res_objectDetection$centers,
  res_objectDetection$coordinates,
  radius = "auto"
  )
changePixelColor(
  beads,
  res_proximityFilter$coordinates,
  color = "darkgreen",
  visualize = TRUE
  )
```

`resultAnalytics` *Result Calculation and Summary*

Description

This function summarizes the data obtained by previous functions: `objectDetection`, `proximityFilter` or `sizeFilter`. Extracts information like amount, intensity, size and density of the objects present in the image.

Usage

```
resultAnalytics(img, coordinates, unfiltered = NULL)
```

Arguments

`img`	image (import by `importImage`)
`coordinates`	all filtered coordinates of the objects (x|y|value data frame)
`unfiltered`	all coordinates from every object before applying filter functions

Details

The `resultAnalytics` function provides comprehensive summary of objects detected in an image:

1. **Summary**
    - Generates a summary of all detected objects, including the total number of objects, their mean size, size standard deviation, mean intensity, intensity standard deviation, estimated rejected objects, and coverage.
2. **Detailed Object Information**
    - Provides detailed information for each object, including size, mean intensity, intensity standard deviation, and coordinates.

Value

list of 2 objects:

- `summary`: A summary of all the objects in the image.
- `detailed`: Detailed information about every single object.

See Also

`objectDetection()`, `sizeFilter()`, `proximityFilter()`

Examples

```
res_objectDetection <- objectDetection(beads,
                                       alpha = 1,
                                       sigma = 0)
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(
  coordinates = res_proximityFilter$coordinates,
  unfiltered = res_objectDetection$coordinates,
  img = beads
  )
print(res_resultAnalytics$summary)
plot(beads)
with(
  res_objectDetection$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
    )
  )
```

`scanDir` *Scan Directory for Image Analysis*

Description

This function scans a specified directory, imports images, and performs various analyses including object detection, size filtering, and proximity filtering. Optionally, it can perform these tasks in parallel and log the process.

Usage

```
scanDir(
  path,
  parallel = FALSE,
  backend = "PSOCK",
  cores = "auto",
  method = "edge",
  alpha = 1,
  sigma = 2,
  sizeFilter = FALSE,
  upperlimit = "auto",
  lowerlimit = "auto",
  proximityFilter = FALSE,
  radius = "auto",
  Rlog = FALSE
)
```

Arguments

`path`	directory path to folder with images to be analyzed
`parallel`	processing multiple images at the same time (default - FALSE)
`backend`	'PSOCK' or 'FORK' (see `makeCluster`)
`cores`	number of cores for parallel processing (numeric / 'auto') ('auto' uses 75% of the available cores)
`method`	choose method for object detection ('edge' / 'threshold') (from `objectDetection`)
`alpha`	threshold adjustment factor (numeric / 'static' / 'interactive' / 'gaussian') (from `objectDetection`)
`sigma`	smoothing (numeric / 'static' / 'interactive' / 'gaussian') (from `objectDetection`)
`sizeFilter`	applying `sizeFilter` function (default - FALSE)
`upperlimit`	highest accepted object size (only needed if sizeFilter = TRUE)
`lowerlimit`	smallest accepted object size (numeric / 'auto')
`proximityFilter`	applying `proximityFilter` function (default - FALSE)
`radius`	distance from one center in which no other centers are allowed (in pixels) (only needed if proximityFilter = TRUE)
`Rlog`	creates a log markdown document, summarizing the results (default - FALSE)

Details

The function scans a specified directory for image files, imports them, and performs analysis using designated methods. The function is capable of parallel processing, utilizing multiple cores to accelerate computation. Additionally, it is able to log the results into an R Markdown file. Duplicate images are identified through the use of MD5 sums. In addition a variety of filtering options are available to refine the analysis. If logging is enabled, the results can be saved and rendered into a report. When `Rlog = TRUE`, an R Markdown file and a CSV file are generated in the current directory. More detailed information on individual results, can be accessed through saved RDS files.

Value

`data.frame` summarizing each analyzed image, including details such as the number of objects, average size and intensity, estimated rejections, and coverage.

See Also

`imgPipe()`, `objectDetection()`, `sizeFilter()`, `proximityFilter()`, `resultAnalytics()`

Examples

```
if (interactive()) {
  path2dir <- system.file("images", package = 'biopixR')
  results <- scanDir(path2dir, alpha = 'interactive', sigma = 'interactive')
  print(results)
  }
```

`shapeFeatures` *Extraction of Shape Features*

Description

This function analyzes the objects detected in an image and calculates distinct shape characteristics for each object, such as circularity, eccentricity, radius, and perimeter. The resulting shape attributes can then be grouped using a Self-Organizing Map (SOM) from the 'Kohonen' package.

Usage

```
shapeFeatures(
  img,
  alpha = 1,
  sigma = 2,
  xdim = 2,
  ydim = 1,
  SOM = FALSE,
  visualize = FALSE
)
```

Arguments

`img`	image (import by `load.image`)
`alpha`	threshold adjustment factor (numeric / 'static' / 'interactive' / 'gaussian') (from `objectDetection`)
`sigma`	smoothing (numeric / 'static' / 'interactive' / 'gaussian') (from `objectDetection`)

`xdim`	x-dimension for the SOM-grid (grid = hexagonal)
`ydim`	y-dimension for the SOM-grid (xdim * ydim = number of neurons)
`SOM`	if TRUE runs SOM algorithm on extracted shape features, grouping the detected objects
`visualize`	visualizes the groups computed by SOM

Value

`data.frame` containing detailed information about every single object.

See Also

`objectDetection()`, `resultAnalytics()`, `som`

Examples

```
shapeFeatures(
  beads,
  alpha = 1,
  sigma = 0,
  SOM = TRUE,
  visualize = TRUE
)
```

`sizeFilter` *Size-based exclusion*

Description

Takes 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 (value|mx|my|size data frame)
`coordinates`	all coordinates of the objects (x|y|value data frame)
`lowerlimit`	smallest accepted object size (numeric / 'auto' / 'interactive')
`upperlimit`	highest accepted object size (numeric / 'auto' / 'interactive')

Details

The `sizeFilter` function is designed to filter detected objects based on their size, either through automated detection or user-defined limits. The automated detection of size limits uses the 1.5*IQR method to identify and remove outliers. This approach is most effective when dealing with a large number of objects, (typically more than 50), and when the sizes of the objects are relatively uniform. For smaller samples or when the sizes of the objects vary significantly, the automated detection may not be as accurate, and manual limit setting is recommended.

Value

list of 2 objects:

- Remaining centers after discarding according to size.
- Remaining coordinates after discarding according to size.

Examples

```
res_objectDetection <- objectDetection(
  beads,
  method = 'edge',
  alpha = 1,
  sigma = 0
  )
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