Package 'pliman'

October 14, 2023

Title Tools for Plant Image Analysis

Version 2.1.0

Description Tools for single or batch image manipulation and analysis as described by Olivoto (2022) <doi:10.1111/2041-210X.13803> that can be used to quantify plant leaf area, assess disease severity, count objects, obtain shape measures, object landmarks, and compute Elliptical Fourier Analysis of the object outline, as described by Claude (2008) <doi:10.1007/978-0-387-77789-4>. Additionally, the package includes tools for analyzing grids, which enables high throughput field phenotyping using RGB imagery captured by unmanned aerial vehicles.

License GPL (>= 3)

URL https://github.com/TiagoOlivoto/pliman

BugReports https://github.com/TiagoOlivoto/pliman/issues

Depends R (>= 4.1)

Imports doParallel, foreach, Rcpp, sf, stars, terra

Suggests BiocManager, EBImage, knitr, leafem (>= 0.2.0), leaflet (>= 2.1.2), mapedit (>= 0.6.0), mapview (>= 2.11.0), rmarkdown, rstudioapi

LinkingTo Rcpp, RcppArmadillo

biocViews

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.2.3

NeedsCompilation yes

Author Tiago Olivoto [aut, cre] (<https://orcid.org/0000-0002-0241-9636>)

Maintainer Tiago Olivoto <tiagoolivoto@gmail.com>

Repository CRAN

Date/Publication 2023-10-14 17:40:02 UTC

R topics documented:

`analyze_objects` *Analyzes objects in an image*

Description

- `analyze_objects()` provides tools for counting and extracting object features (e.g., area, perimeter, radius, pixel intensity) in an image. See more at the **Details** section.
- `analyze_objects_iter()` provides an iterative section to measure object features using an object with a known area.
- `plot.anal_obj()` produces a histogram for the R, G, and B values when argument `object_index` is used in the function `analyze_objects()`.

Usage

```
analyze_objects(
  img,
  foreground = NULL,
  background = NULL,
  pick_palettes = FALSE,
  segment_objects = TRUE,
  viewer = get_pliman_viewer(),
  reference = FALSE,
  reference_area = NULL,
  back_fore_index = "R/(G/B)",
  fore_ref_index = "B-R",
  reference_larger = FALSE,
  reference_smaller = FALSE,
  pattern = NULL,
  parallel = FALSE,
  workers = NULL,
  watershed = TRUE,
  veins = FALSE,
  sigma_veins = 1,
  ab_angles = FALSE,
  ab_angles_percentiles = c(0.25, 0.75),
  width_at = FALSE,
  width_at_percentiles = c(0.05, 0.25, 0.5, 0.75, 0.95),
  haralick = FALSE,
  har_nbins = 32,
  har_scales = 1,
```

```
  har_band = 1,
  pcv = FALSE,
  pcv_niter = 100,
  resize = FALSE,
  trim = FALSE,
  fill_hull = FALSE,
  filter = FALSE,
  invert = FALSE,
  object_size = "medium",
  index = "NB",
  r = 1,
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  object_index = NULL,
  pixel_level_index = FALSE,
  return_mask = FALSE,
  efourier = FALSE,
  nharm = 10,
  threshold = "Otsu",
  k = 0.1,
  windowsize = NULL,
  tolerance = NULL,
  extension = NULL,
  lower_noise = 0.1,
  lower_size = NULL,
  upper_size = NULL,
  topn_lower = NULL,
  topn_upper = NULL,
  lower_eccent = NULL,
  upper_eccent = NULL,
  lower_circ = NULL,
  upper_circ = NULL,
  randomize = TRUE,
  nrows = 1000,
  plot = TRUE,
  show_original = TRUE,
  show_chull = FALSE,
  show_contour = TRUE,
  contour_col = "red",
  contour_size = 1,
  show_lw = FALSE,
  show_background = TRUE,
  show_segmentation = FALSE,
  col_foreground = NULL,
  col_background = NULL,
  marker = FALSE,
```

```
  marker_col = NULL,
  marker_size = NULL,
  save_image = FALSE,
  prefix = "proc_",
  dir_original = NULL,
  dir_processed = NULL,
  verbose = TRUE
)

## S3 method for class 'anal_obj'
plot(
  x,
  which = "measure",
  measure = "area",
  type = c("density", "histogram"),
  ...
)

analyze_objects_iter(pattern, known_area, verbose = TRUE, ...)
```

Arguments

`img`	The image to be analyzed.
`foreground, background`	A color palette for the foregrond and background, respectively (optional). If a chacarceter is used (eg., `foreground = "fore"`), the function will search in the current working directory a valid image named "fore".
`pick_palettes`	Logical argument indicating wheater the user needs to pick up the color palettes for foreground and background for the image. If `TRUE` `pick_palette()` will be called internally so that the user can sample color points representing foreground and background.
`segment_objects`	Segment objects in the image? Defaults to `TRUE`. In this case, objects are segmented using the index defined in the `index` argument, and each object is analyzed individually. If `segment_objects = FALSE` is used, the objects are not segmented and the entire image is analyzed. This is useful, for example, when analyzing an image without background, where an `object_index` could be computed for the entire image, like the index of a crop canopy.
`viewer`	The viewer option. This option controls the type of viewer to use for interactive plotting (eg., when `pick_palettes = TRUE`). If not provided, the value is retrieved using `get_pliman_viewer()`.
`reference`	Logical to indicate if a reference object is present in the image. This is useful to adjust measures when images are not obtained with standard resolution (e.g., field images). See more in the details section.
`reference_area`	The known area of the reference objects. The measures of all the objects in the image will be corrected using the same unit of the area informed here.

back_fore_index
: A character value to indicate the index to segment the foreground (objects and reference) from the background. Defaults to "R/(G/B)". This index is optimized to segment white backgrounds from green leaves and a blue reference object.

fore_ref_index
: A character value to indicate the index to segment objects and the reference object. It can be either an available index in pliman (see pliman_indexes() or an own index computed with the R, G, and B bands. Defaults to "B-R". This index is optimized to segment green leaves from a blue reference object after a white background has been removed.

reference_larger, reference_smaller
: Logical argument indicating when the larger/smaller object in the image must be used as the reference object. This only is valid when reference is set to TRUE and reference_area indicates the area of the reference object. IMPORTANT. When reference_smaller is used, objects with an area smaller than 1% of the mean of all the objects are ignored. This is used to remove possible noise in the image such as dust. So, be sure the reference object has an area that will be not removed by that cutpoint.

pattern
: A pattern of file name used to identify images to be imported. For example, if pattern = "im" all images in the current working directory that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be imported as a list. Providing any number as pattern (e.g., pattern = "1") will select images that are named as 1.-, 2.-, and so on. An error will be returned if the pattern matches any file that is not supported (e.g., img1.pdf).

parallel
: If TRUE processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time, especially when pattern is used is informed. When object_index is informed, multiple sections will be used to extract the RGB values for each object in the image. This may significantly speed up processing time when an image has lots of objects (say >1000).

workers
: A positive numeric scalar or a function specifying the number of parallel processes that can be active at the same time. By default, the number of sections is set up to 30% of available cores.

watershed
: If TRUE (default) performs watershed-based object detection. This will detect objects even when they are touching one other. If FALSE, all pixels for each connected set of foreground pixels are set to a unique object. This is faster but is not able to segment touching objects.

veins
: Logical argument indicating whether vein features are computed. This will call object_edge() and applies the Sobel-Feldman Operator to detect edges. The result is the proportion of edges in relation to the entire area of the object(s) in the image. Note that **THIS WILL BE AN OPERATION ON AN IMAGE LEVEL, NOT OBJECT!**.

sigma_veins
: Gaussian kernel standard deviation used in the gaussian blur in the edge detection algorithm

ab_angles
: Logical argument indicating whether apex and base angles should be computed. Defaults to FALSE. If TRUE, poly_apex_base_angle() are called and the base

and apex angles are computed considering the 25th and 75th percentiles of the object height. These percentiles can be changed with the argument `ab_angles_percentiles`.

`ab_angles_percentiles`
: The percentiles indicating the heights of the object for which the angle should be computed (from the apex and the bottom). Defaults to c(0.25, 0.75), which means considering the 25th and 75th percentiles of the object height.

`width_at`
: Logical. If `TRUE`, the widths of the object at a given set of quantiles of the height are computed.

`width_at_percentiles`
: A vector of heights along the vertical axis of the object at which the width will be computed. The default value is c(0.05, 0.25, 0.5, 0.75, 0.95), which means the function will return the width at the 5th, 25th, 50th, 75th, and 95th percentiles of the object's height.

`haralick`
: Logical value indicating whether Haralick features are computed. Defaults to `FALSE`.

`har_nbins`
: An integer indicating the number of bins using to compute the Haralick matrix. Defaults to 32. See Details

`har_scales`
: A integer vector indicating the number of scales to use to compute the Haralick features. See Details.

`har_band`
: The band to compute the Haralick features (1 = R, 2 = G, 3 = B). Defaults to 1.

`pcv`
: Computes the Perimeter Complexity Value? Defaults to `FALSE`.

`pcv_niter`
: An integer specifying the number of smoothing iterations for computing the Perimeter Complexity Value. Defaults to 100.

`resize`
: Resize the image before processing? Defaults to `FALSE`. Use a numeric value of range 0-100 (proportion of the size of the original image).

`trim`
: Number of pixels removed from edges in the analysis. The edges of images are often shaded, which can affect image analysis. The edges of images can be removed by specifying the number of pixels. Defaults to `FALSE` (no trimmed edges).

`fill_hull`
: Fill holes in the binary image? Defaults to `FALSE`. This is useful to fill holes in objects that have portions with a color similar to the background. IMPORTANT: Objects touching each other can be combined into one single object, which may underestimate the number of objects in an image.

`filter`
: Performs median filtering in the binary image? See more at `image_filter()`. Defaults to `FALSE`. Use a positive integer to define the size of the median filtering. Larger values are effective at removing noise, but adversely affect edges.

`invert`
: Inverts the binary image if desired. This is useful to process images with a black background. Defaults to `FALSE`. If `reference = TRUE` is use, `invert` can be declared as a logical vector of length 2 (eg., `invert = c(FALSE, TRUE)`. In this case, the segmentation of objects and reference from the foreground using `back_fore_index` is performed using the default (not inverted), and the segmentation of objects from the reference is performed by inverting the selection (selecting pixels higher than the threshold).

`object_size` The size of the object. Used to automatically set up `tolerance` and `extension` parameters. One of the following. `"small"` (e.g, wheat grains), `"medium"` (e.g, soybean grains), `"large"`(e.g, peanut grains), and `"elarge"` (e.g, soybean pods)‘.

`index` A character value specifying the target mode for conversion to binary image when `foreground` and `background` are not declared. Defaults to `"NB"` (normalized blue). See `image_index()` for more details. User can also calculate your own index using the bands names, e.g. `index = "R+B/G"`

`r, g, b, re, nir` The red, green, blue, red-edge, and near-infrared bands of the image, respectively. Defaults to 1, 2, 3, 4, and 5, respectively. If a multispectral image is provided (5 bands), check the order of bands, which are frequently presented in the 'BGR' format.

`object_index` Defaults to `FALSE`. If an index is informed, the average value for each object is returned. It can be the R, G, and B values or any operation involving them, e.g., `object_index = "R/B"`. In this case, it will return for each object in the image, the average value of the R/B ratio. Use `pliman_indexes_eq()` to see the equations of available indexes.

`pixel_level_index` Return the indexes computed in `object_index` in the pixel level? Defaults to `FALSE` to avoid returning large data.frames.

`return_mask` Returns the mask for the analyzed image? Defaults to `FALSE`.

`efourier` Logical argument indicating if Elliptical Fourier should be computed for each object. This will call `efourier()` internally. It `efourier = TRUE` is used, both standard and normalized Fourier coefficients are returned.

`nharm` An integer indicating the number of harmonics to use. Defaults to 10. For more details see `efourier()`.

`threshold` The theshold method to be used.

- By default (`threshold = "Otsu"`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold.
- If `threshold = "adaptive"`, adaptive thresholding (Shafait et al. 2008) is used, and will depend on the `k` and `windowsize` arguments.
- If any non-numeric value different than `"Otsu"` and `"adaptive"` is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index.

`k` a numeric in the range 0-1. when `k` is high, local threshold values tend to be lower. when `k` is low, local threshold value tend to be higher.

`windowsize` windowsize controls the number of local neighborhood in adaptive thresholding. By default it is set to `1/3 * minxy`, where `minxy` is the minimum dimension of the image (in pixels).

`tolerance` The minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbors, which is the highest.

`extension` Radius of the neighborhood in pixels for the detection of neighboring objects. Higher value smooths out small objects.

`lower_noise` To prevent noise from affecting the image analysis, objects with lesser than 10% of the mean area of all objects are removed (`lower_noise = 0.1`). Increasing this value will remove larger noises (such as dust points), but can remove desired objects too. To define an explicit lower or upper size, use the `lower_size` and `upper_size` arguments.

`lower_size, upper_size`
Lower and upper limits for size for the image analysis. Plant images often contain dirt and dust. Upper limit is set to `NULL`, i.e., no upper limit used. One can set a known area or use `lower_limit = 0` to select all objects (not advised). Objects that matches the size of a given range of sizes can be selected by setting up the two arguments. For example, if `lower_size = 120` and `upper_size = 140`, objects with size greater than or equal 120 and less than or equal 140 will be considered.

`topn_lower, topn_upper`
Select the top `n` objects based on its area. `topn_lower` selects the `n` elements with the smallest area whereas `topn_upper` selects the `n` objects with the largest area.

`lower_eccent, upper_eccent, lower_circ, upper_circ`
Lower and upper limit for object eccentricity/circularity for the image analysis. Users may use these arguments to remove objects such as square papers for scale (low eccentricity) or cut petioles (high eccentricity) from the images. Defaults to `NULL` (i.e., no lower and upper limits).

`randomize` Randomize the lines before training the model?

`nrows` The number of lines to be used in training step. Defaults to 2000.

`plot` Show image after processing?

`show_original` Show the count objects in the original image?

`show_chull` Show the convex hull around the objects? Defaults to `FALSE`.

`show_contour` Show a contour line around the objects? Defaults to `TRUE`.

`contour_col, contour_size`
The color and size for the contour line around objects. Defaults to `contour_col = "red"` and `contour_size = 1`.

`show_lw` If `TRUE`, plots the length and width lines on each object calling `plot_lw()`.

`show_background`
Show the background? Defaults to `TRUE`. A white background is shown by default when `show_original = FALSE`.

`show_segmentation`
Shows the object segmentation colored with random permutations. Defaults to `FALSE`.

`col_foreground, col_background`
Foreground and background color after image processing. Defaults to `NULL`, in which `"black"`, and `"white"` are used, respectively.

`marker, marker_col, marker_size`
The type, color and size of the object marker. Defaults to `NULL`, which plots the object id. Use `marker = "point"` to show a point in each object or `marker = FALSE` to omit object marker.

save_image — Save the image after processing? The image is saved in the current working directory named as proc_* where * is the image name given in img.

prefix — The prefix to be included in the processed images. Defaults to "proc_".

dir_original, dir_processed — The directory containing the original and processed images. Defaults to NULL. In this case, the function will search for the image img in the current working directory. After processing, when save_image = TRUE, the processed image will be also saved in such a directory. It can be either a full path, e.g., "C:/Desktop/imgs", or a subfolder within the current working directory, e.g., "/imgs".

verbose — If TRUE (default) a summary is shown in the console.

x — An object of class anal_obj.

which — Which to plot. Either 'measure' (object measures) or 'index' (object index). Defaults to "measure".

measure — The measure to plot. Defaults to "area".

type — The type of plot. Either "hist" or "density". Partial matches are recognized.

... — Depends on the function:

- For analyze_objects_iter(), further arguments passed on to analyze_objects().

known_area — The known area of the template object.

Details

A binary image is first generated to segment the foreground and background. The argument index is useful to choose a proper index to segment the image (see image_binary() for more details). It is also possible to provide color palettes for background and foreground (arguments background and foreground, respectively). When this is used, a general linear model (binomial family) fitted to the RGB values to segment fore- and background.

Then, the number of objects in the foreground is counted. By setting up arguments such as lower_size and upper_size, it is possible to set a threshold for lower and upper sizes of the objects, respectively. The argument object_size can be used to set up pre-defined values of tolerance and extension depending on the image resolution. This will influence the watershed-based object segmentation. Users can also tune up tolerance and extension explicitly for a better precision of watershed segmentation.

If watershed = FALSE is used, all pixels for each connected set of foreground pixels in img are set to a unique object. This is faster, especially for a large number of objects, but it is not able to segment touching objects.

There are some ways to correct the measures based on a reference object. If a reference object with a known area (reference_area) is used in the image and reference = TRUE is used, the measures of the objects will be corrected, considering the unit of measure informed in reference_area. There are two main ways to work with reference objects.

- The first, is to provide a reference object that has a contrasting color with both the background and object of interest. In this case, the arguments back_fore_index and fore_ref_index can be used to define an index to first segment the reference object and objects to be measured from the background, then the reference object from objects to be measured.

- The second one is to use a reference object that has a similar color to the objects to be measured, but has a contrasting size. For example, if we are counting small brown grains, we can use a brown reference template that has an area larger (says 3 times the area of the grains) and then uses `reference_larger = TRUE`. With this, the larger object in the image will be used as the reference object. This is particularly useful when images are captured with background light, such as the example 2. Some types: (i) It is suggested that the reference object is not too much larger than the objects of interest (mainly when the `watershed = TRUE`). In some cases, the reference object can be broken into several pieces due to the watershed algorithm. (ii) Since the reference object will increase the mean area of the object, the argument `lower_noise` can be increased. By default (`lower_noise = 0.1`) objects with lesser than 10% of the mean area of all objects are removed. Since the mean area will be increased, increasing `lower_noise` will remove dust and noises more reliably. The argument `reference_smaller` can be used in the same way

By using `pattern`, it is possible to process several images with common pattern names that are stored in the current working directory or in the subdirectory informed in `dir_original`. To speed up the computation time, one can set `parallel = TRUE`.

`analyze_objects_iter()` can be used to process several images using an object with a known area as a template. In this case, all the images in the current working directory that match the `pattern` will be processed. For each image, the function will compute the features for the objects and show the identification (id) of each object. The user only needs to inform which is the id of the known object. Then, given the `known_area`, all the measures will be adjusted. In the end, a data.frame with the adjusted measures will be returned. This is useful when the images are taken at different heights. In such cases, the image resolution cannot be conserved. Consequently, the measures cannot be adjusted using the argument `dpi` from `get_measures()`, since each image will have a different resolution. NOTE: This will only work in an interactive section.

- Additional measures: By default, some measures are not computed, mainly due to computational efficiency when the user only needs simple measures such as area, length, and width.
  - If `haralick = TRUE`, The function computes 13 Haralick texture features for each object based on a gray-level co-occurrence matrix (Haralick et al. 1979). Haralick features depend on the configuration of the parameters `har_nbins` and `har_scales`. `har_nbins` controls the number of bins used to compute the Haralick matrix. A smaller `har_nbins` can give more accurate estimates of the correlation because the number of events per bin is higher. While a higher value will give more sensitivity. `har_scales` controls the number of scales used to compute the Haralick features. Since Haralick features compute the correlation of intensities of neighboring pixels it is possible to identify textures with different scales, e.g., a texture that is repeated every two pixels or 10 pixels. By default, the Haralick features are computed with the R band. To chance this default, use the argument `har_band`. For example, `har_band = 2` will compute the features with the green band.
  - If `efourier = TRUE` is used, an Elliptical Fourier Analysis (Kuhl and Giardina, 1982) is computed for each object contour using `efourier()`.
  - If `veins = TRUE` (experimental), vein features are computed. This will call `object_edge()` and applies the Sobel-Feldman Operator to detect edges. The result is the proportion of edges in relation to the entire area of the object(s) in the image. Note that THIS WILL BE AN OPERATION ON AN IMAGE LEVEL, NOT an OBJECT LEVEL! So, If vein features need to be computed for leaves, it is strongly suggested to use one leaf per image.

- If ab_angles = TRUE the apex and base angles of each object are computed with poly_apex_base_angle(). By default, the function computes the angle from the first pixel of the apex of the object to the two pixels that slice the object at the 25th percentile of the object height (apex angle). The base angle is computed in the same way but from the first base pixel.
- If width_at = TRUE, the width at the 5th, 25th, 50th, 75th, and 95th percentiles of the object height are computed by default. These quantiles can be adjusted with the width_at_percentiles argument.

Value

analyze_objects() returns a list with the following objects:

- results A data frame with the following variables for each object in the image:
  - id: object identification.
  - x,y: x and y coordinates for the center of mass of the object.
  - area: area of the object (in pixels).
  - area_ch: the area of the convex hull around object (in pixels).
  - perimeter: perimeter (in pixels).
  - radius_min, radius_mean, and radius_max: The minimum, mean, and maximum radius (in pixels), respectively.
  - radius_sd: standard deviation of the mean radius (in pixels).
  - diam_min, diam_mean, and diam_max: The minimum, mean, and maximum diameter (in pixels), respectively.
  - major_axis, minor_axis: elliptical fit for major and minor axes (in pixels).
  - caliper: The longest distance between any two points on the margin of the object. See poly_caliper() for more details
  - length, width The length and width of objects (in pixels). These measures are obtained as the range of x and y coordinates after aligning each object with poly_align().
  - radius_ratio: radius ratio given by radius_max / radius_min.
  - theta: object angle (in radians).
  - eccentricity: elliptical eccentricity computed using the ratio of the eigen values (inertia axes of coordinates).
  - form_factor (Wu et al., 2007): the difference between a leaf and a circle. It is defined as 4*pi*A/P, where A is the area and P is the perimeter of the object.
  - narrow_factor (Wu et al., 2007): Narrow factor (caliper / length).
  - asp_ratio (Wu et al., 2007): Aspect ratio (length / width).
  - rectangularity (Wu et al., 2007): The similarity between a leaf and a rectangle (length * width/ area).
  - pd_ratio (Wu et al., 2007): Ratio of perimeter to diameter (perimeter / caliper)
  - plw_ratio (Wu et al., 2007): Perimeter ratio of length and width (perimeter / (length + width))
  - solidity: object solidity given by area / area_ch.
  - convexity: The convexity of the object computed using the ratio between the perimeter of the convex hull and the perimeter of the polygon.
  - elongation: The elongation of the object computed as 1 - width / length.

  - circularity: The object circularity given by perimeter ^ 2 / area.
  - circularity_haralick: The Haralick's circularity (CH), computed as CH = m/sd, where m and sd are the mean and standard deviations from each pixels of the perimeter to the centroid of the object.
  - circularity_norm: The normalized circularity (Cn), to be unity for a circle. This measure is computed as Cn = perimeter ^ 2 / 4*pi*area and is invariant under translation, rotation, scaling transformations, and dimensionless.
  - asm: The angular second-moment feature.
  - con: The contrast feature
  - cor: Correlation measures the linear dependency of gray levels of neighboring pixels.
  - var: The variance of gray levels pixels.
  - idm: The Inverse Difference Moment (IDM), i.e., the local homogeneity.
  - sav: The Sum Average.
  - sva: The Sum Variance.
  - sen: Sum Entropy.
  - dva: Difference Variance.
  - den: Difference Entropy
  - f12: Difference Variance.
  - f13: The angular second-moment feature.

- statistics: A data frame with the summary statistics for the area of the objects.
- count: If pattern is used, shows the number of objects in each image.
- obj_rgb: If object_index is used, returns the R, G, and B values for each pixel of each object.
- object_index: If object_index is used, returns the index computed for each object.
- Elliptical Fourier Analysis: If efourier = TRUE is used, the following objects are returned.
  - efourier: The Fourier coefficients. For more details see efourier().
  - efourier_norm: The normalized Fourier coefficients. For more details see efourier_norm().
  - efourier_error: The error between original data and reconstructed outline. For more details see efourier_error().
  - efourier_power: The spectrum of harmonic Fourier power. For more details see efourier_power().
- veins: If veins = TRUE is used, returns, for each image, the proportion of veins (in fact the object edges) related to the total object(s)' area.
- analyze_objects_iter() returns a data.frame containing the features described in the results object of analyze_objects().
- plot.anal_obj() returns a trellis object containing the distribution of the pixels, optionally for each object when facet = TRUE is used.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Gupta, S., Rosenthal, D. M., Stinchcombe, J. R., & Baucom, R. S. (2020). The remarkable morphological diversity of leaf shape in sweet potato (Ipomoea batatas): the influence of genetics, environment, and G×E. New Phytologist, 225(5), 2183–2195. doi:10.1111/NPH.16286

Haralick, R.M., K. Shanmugam, and I. Dinstein. 1973. Textural Features for Image Classification. IEEE Transactions on Systems, Man, and Cybernetics SMC-3(6): 610–621. doi:10.1109/TSMC.1973.4309314

Kuhl, F. P., and Giardina, C. R. (1982). Elliptic Fourier features of a closed contour. Computer Graphics and Image Processing 18, 236–258. doi: doi:10.1016/0146664X(82)90034X

Lee, Y., & Lim, W. (2017). Shoelace Formula: Connecting the Area of a Polygon and the Vector Cross Product. The Mathematics Teacher, 110(8), 631–636. doi:10.5951/mathteacher.110.8.0631

Montero, R. S., Bribiesca, E., Santiago, R., & Bribiesca, E. (2009). State of the Art of Compactness and Circularity Measures. International Mathematical Forum, 4(27), 1305–1335.

Chen, C.H., and P.S.P. Wang. 2005. Handbook of Pattern Recognition and Computer Vision. 3rd ed. World Scientific.

Wu, S. G., Bao, F. S., Xu, E. Y., Wang, Y.-X., Chang, Y.-F., and Xiang, Q.-L. (2007). A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network. in 2007 IEEE International Symposium on Signal Processing and Information Technology, 11–16. doi:10.1109/ISSPIT.2007.4458016

Examples

```
library(pliman)
img <- image_pliman("soybean_touch.jpg")
obj <- analyze_objects(img)
obj$statistics

########################### Example 1 ########################
# Enumerate the objects in the original image
# Return the top-5 grains with the largest area
top <-
 analyze_objects(img,
                 marker = "id",
                 topn_upper = 5)
top$results


#' ########################### Example 1 ########################
# Correct the measures based on the area of the largest object
# note that since the reference object

img <- image_pliman("flax_grains.jpg")
res <-
  analyze_objects(img,
                  index = "GRAY",
                  marker = "point",
```

```
                  show_contour = FALSE,
                  reference = TRUE,
                  reference_area = 6,
                  reference_larger = TRUE,
                  lower_noise = 0.3)


library(pliman)

img <- image_pliman("soy_green.jpg")
# Segment the foreground (grains) using the normalized blue index (NB, default)
# Shows the average value of the blue index in each object

rgb <-
   analyze_objects(img,
                   marker = "id",
                   object_index = "B",
                   pixel_level_index = TRUE)
# density of area
plot(rgb)

# histogram of perimeter
plot(rgb, measure = "perimeter", type = "histogram") # or 'hist'

# density of the blue (B) index
plot(rgb, which = "index")
```

`analyze_objects_shp` *Analyzes objects using shapefiles*

Description

Analyzes objects using shapefiles

Usage

```
analyze_objects_shp(
  img,
  nrow = 1,
  ncol = 1,
  buffer_x = 0,
  buffer_y = 0,
  prepare = FALSE,
  segment_objects = TRUE,
  viewer = get_pliman_viewer(),
  index = "R",
  r = 1,
```

```
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  shapefile = NULL,
  interactive = FALSE,
  plot = FALSE,
  parallel = FALSE,
  workers = NULL,
  watershed = TRUE,
  filter = FALSE,
  object_size = "medium",
  efourier = FALSE,
  object_index = NULL,
  veins = FALSE,
  width_at = FALSE,
  verbose = TRUE,
  invert = FALSE,
  ...
)
```

Arguments

`img`	An `Image` object
`nrow, ncol`	The number of rows and columns to generate the shapefile when `shapefile` is not declared. Defaults to 1.
`buffer_x, buffer_y`	Buffering factor for the width and height, respectively, of each individual shape's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the shape by 25% on each side.
`prepare`	Logical value indicating whether to prepare the image for analysis using `image_prepare()` function. Defaults to `FALSE`. Set to `TRUE` to interactively align and crop the image before processing.
`segment_objects`	Segment objects in the image? Defaults to `TRUE`. In this case, objects are segmented using the index defined in the `index` argument, and each object is analyzed individually. If `segment_objects = FALSE` is used, the objects are not segmented and the entire image is analyzed. This is useful, for example, when analyzing an image without background, where an `object_index` could be computed for the entire image, like the index of a crop canopy.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run

	`set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.
`index`	A character value specifying the target mode for conversion to binary image when `foreground` and `background` are not declared. Defaults to `"NB"` (normalized blue). See `image_index()` for more details. User can also calculate your own index using the bands names, e.g. `index = "R+B/G"`
`r, g, b, re, nir`	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. Defaults to 1, 2, 3, 4, and 5, respectively. If a multispectral image is provided (5 bands), check the order of bands, which are frequently presented in the 'BGR' format.
`shapefile`	(Optional) An object created with `image_shp()`. If `NULL` (default), both `nrow` and `ncol` must be declared.
`interactive`	If `FALSE` (default) the grid is created automatically based on the image dimension and number of nrow/columns. If `interactive = TRUE`, users must draw points at the diagonal of the desired bounding box that will contain the grid.
`plot`	Plots the processed images? Defaults to `FALSE`.
`parallel`	If `TRUE` processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time, especially when `pattern` is used is informed. When `object_index` is informed, multiple sections will be used to extract the RGB values for each object in the image. This may significantly speed up processing time when an image has lots of objects (say >1000).
`workers`	A positive numeric scalar or a function specifying the number of parallel processes that can be active at the same time. By default, the number of sections is set up to 30% of available cores.
`watershed`	If `TRUE` (default) performs watershed-based object detection. This will detect objects even when they are touching one other. If `FALSE`, all pixels for each connected set of foreground pixels are set to a unique object. This is faster but is not able to segment touching objects.
`filter`	Performs median filtering in the binary image? See more at `image_filter()`. Defaults to `FALSE`. Use a positive integer to define the size of the median filtering. Larger values are effective at removing noise, but adversely affect edges.
`object_size`	Argument to control control the watershed segmentation. See `analyze_objects()` for more details.
`efourier`	Logical argument indicating if Elliptical Fourier should be computed for each object. This will call `efourier()` internally. It `efourier = TRUE` is used, both standard and normalized Fourier coefficients are returned.
`object_index`	Defaults to `FALSE`. If an index is informed, the average value for each object is returned. It can be the R, G, and B values or any operation involving them, e.g., `object_index = "R/B"`. In this case, it will return for each object in the image, the average value of the R/B ratio. Use `pliman_indexes_eq()` to see the equations of available indexes.
`veins`	Logical argument indicating whether vein features are computed. This will call `object_edge()` and applies the Sobel-Feldman Operator to detect edges. The result is the proportion of edges in relation to the entire area of the object(s) in

	the image. Note that **THIS WILL BE AN OPERATION ON AN IMAGE LEVEL, NOT OBJECT!**.
`width_at`	Logical. If `TRUE`, the widths of the object at a given set of quantiles of the height are computed.
`verbose`	If `TRUE` (default) a summary is shown in the console.
`invert`	Inverts the binary image if desired. This is useful to process images with a black background. Defaults to `FALSE`. If `reference = TRUE` is use, `invert` can be declared as a logical vector of length 2 (eg., `invert = c(FALSE, TRUE)`. In this case, the segmentation of objects and reference from the foreground using `back_fore_index` is performed using the default (not inverted), and the segmentation of objects from the reference is performed by inverting the selection (selecting pixels higher than the threshold).
`...`	Aditional arguments passed on to analyze_objects.

Details

The `analyze_objects_shp` function performs object analysis on an image and generates shapefiles representing the analyzed objects. The function first prepares the image for analysis using the `image_prepare()` function if the `prepare` argument is set to `TRUE`. If a shapefile object is provided, the number of rows and columns for splitting the image is obtained from the shapefile. Otherwise, the image is split into multiple sub-images based on the specified number of rows and columns using the `object_split_shp()` function. The objects in each sub-image are analyzed using the `analyze_objects()` function, and the results are stored in a list. If parallel processing is enabled, the analysis is performed in parallel using multiple workers.

The output object provides access to various components of the analysis results, such as the analyzed object coordinates and properties. Additionally, the shapefiles representing the analyzed objects are included in the output object for further analysis or visualization.

Value

An object of class `anal_obj`. See more details in the `Value` section of `analyze_objects()`.

Examples

```
if(interactive()){
library(pliman)

# Computes the DGCI index for each flax leaf
flax <- image_pliman("flax_leaves.jpg", plot =TRUE)
res <-
   analyze_objects_shp(flax,
                       nrow = 3,
                       ncol = 5,
                       plot = FALSE,
                       object_index = "DGCI")
plot(flax)
plot(res$shapefiles)
plot_measures(res, measure = "DGCI")
}
```

`apply_fun_to_imgs` *Apply a function to images*

Description

Most of the functions in pliman can be applied to a list of images, but this can be not ideal to deal with lots of images, mainly if they have a high resolution. For curiosity, a 6000 x 4000 image use nearly 570 Megabytes of RAM. So, it would be impossible to deal with lots of images within R. `apply_fun_to_img()` applies a function to images stored in a given directory as follows:

- Create a vector of image names that contain a given pattern of name.
- Import each image of such a list.
- Apply a function to the imported image.
- Export the mutated image to the computer.

If parallel is set to `FALSE` (default), the images are processed sequentially, which means that one image needs to be imported, processed, and exported so that the other image can be processed. If parallel is set to `TRUE`, the images are processed asynchronously (in parallel) in separate R sessions (3) running in the background on the same machine. It may speed up the processing time when lots of images need to be processed.

Usage

```
apply_fun_to_imgs(
  pattern,
  fun,
  ...,
  dir_original = NULL,
  dir_processed = NULL,
  prefix = "",
  suffix = "",
  parallel = FALSE,
  workers = 3,
  verbose = TRUE
)
```

Arguments

`pattern`	A pattern to match the images' names.
`fun`	A function to apply to the images.
`...`	Arguments passed on to `fun`.
`dir_original, dir_processed`	The directory containing the original and processed images. Defaults to `NULL`, which means that the current working directory will be considered. **The processed image will overwrite the original image unless a prefix/suffix be used or a subfolder is informed in dir_processed argument**.

prefix, suffix	A prefix and/or suffix to be included in the name of processed images. Defaults to "".
parallel	If TRUE processes the images asynchronously (in parallel) in separate R sessions (3 by default) running in the background on the same machine. It may speed up the processing time, especially when pattern is used is informed.
workers	A positive numeric scalar or a function specifying the number of parallel processes that can be active at the same time. Defaults to 3.
verbose	Shows the progress in console? Defaults to TRUE.

Value

Nothing. The processed images are saved to the current working directory.

Examples

```
# apply_fun_to_imgs("pattern", image_resize, rel_size = 50)
```

---

as_image *Create an* Image *object*

Description

This function is a simple wrapper around EBImage::Image().

Usage

```
as_image(data, ...)
```

Arguments

data	A vector or array containing the pixel intensities of an image. If missing, the default 1x1 zero-filled array is used.
...	Additional arguments passed to EBImage::Image().

Value

An Image object.

Examples

```
img <-
as_image(rnorm(150 * 150 * 3),
         dim = c(150, 150, 3),
         colormode = 'Color')
plot(img)
```

`calibrate` *Calibrates distances of landmarks*

Description

Calibrating the actual size is possible if any interlandmark distance on the image is known. calibrate() can be used to determine the size of a known distance (cm) on the graph. I invite users to photograph the object together with a scale (e.g., ruler, micrometer...).

Usage

```
calibrate(img, viewer = get_pliman_viewer())
```

Arguments

`img`	An `Image` object
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.

Value

A numeric (`double`) scalar value indicating the scale (in pixels per unit of known distance).

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if(isTRUE(interactive())){
library(pliman)
#### compute scale (dots per unit of known distance) ####
# only works in an interactive section
# objects_300dpi.jpg has a known resolution of 300 dpi
img <- image_pliman("objects_300dpi.jpg")
# Larger square: 10 x 10 cm
# 1) Run the function calibrate()
# 2) Use the left mouse button to create a line in the larger square
# 3) Declare a known distance (10 cm)
# 4) See the computed scale (pixels per cm)
calibrate(img)
```

```
# scale ~118
# 118 * 2.54 ~300 DPI
}
```

`contours` *Contour outlines from five leaves*

Description

A list of contour outlines from five leaves. It may be used as example in some functions such as `efourier()`

Format

A list with five objects

- `leaf_1`
- `leaf_2`
- `leaf_3`
- `leaf_4`
- `leaf_5`

Each object is a `data.frame` with the coordinates for the outline perimeter

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Source

Personal data. The images were obtained in the Flavia data set downlodable at https://flavia.sourceforge.net/

`custom_palette` *Generate Custom Color Palette*

Description

This function generates a custom color palette using the specified colors and number of colors.

Usage

```
custom_palette(
  colors = c("#4B0055", "#00588B", "#009B95", "#53CC67", "yellow"),
  n = 100
)
```

Arguments

colors	A vector of colors to create the color palette. Default is c("steelblue", "salmon", "forestgreen").
n	The number of gradient colors in the color palette. Default is 100.

Value

A vector of colors representing the custom color palette.

Examples

```
# Generate a custom color palette with default colors and 10 colors
custom_palette()

# Generate a custom color palette with specified colors and 20 colors
custom_palette(colors = c("blue", "red"), n = 20)

# example code
library(pliman)
custom_palette(n = 5)
```

---

dist_transform *Distance map transform*

Description

Computes the distance map transform of a binary image. The distance map is a matrix which contains for each pixel the distance to its nearest background pixel.

Usage

```
dist_transform(binary)
```

Arguments

binary	A binary image

Value

An Image object or an array, with pixels containing the distances to the nearest background points

Examples

```
library(pliman)
img <- image_pliman("soybean_touch.jpg")
binary <- image_binary(img, "B")[[1]]
wts <- dist_transform(binary)
range(wts)
```

efourier *Elliptical Fourier Analysis*

Description

Computes Elliptical Fourier Analysis of closed outlines based on x and y-coordinates coordinates.

Usage

```
efourier(x, nharm = 10, align = FALSE, center = FALSE, smooth_iter = 0)
```

Arguments

x	A matrix, a data.frame a list of perimeter coordinates, often produced with object_contour() or a vector of landmarks produced with landmarks() or landmarks_regradi().
nharm	An integer indicating the number of harmonics to use. Defaults to 10.
align	Align the objects before computing Fourier analysis? Defaults to FALSE. If TRUE, the object is first aligned along the major caliper with poly_align().
center	Center the objects on the origin before computing Fourier analysis? Defaults to FALSE. If TRUE, the object is first centered on the origin with poly_center().
smooth_iter	The number of smoothing iterations to perform. This will smooth the perimeter of the objects using poly_smooth().

Details

Adapted from Claude (2008). pp. 222-223.

Value

A list of class efourier with:

- the harmonic coefficients (an, bn, cn and dn)
- the estimates of the coordinates of the centroid of the configuration (a0 and c0).
- The number of rows (points) of the perimeter outline (nr).
- The number of harmonics used (nharm).
- The original coordinates (coords).

If x is a list of perimeter coordinates, a list of efourier objects will be returned as an object of class iefourier_lst.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Kuhl, F. P., and Giardina, C. R. (1982). Elliptic Fourier features of a closed contour. Computer Graphics and Image Processing 18, 236–258. doi: doi:10.1016/0146664X(82)90034X

Examples

```
library(pliman)
leaf1 <- contours[[4]]
plot_polygon(leaf1)

#### default options
# 10 harmonics (default)
# without alignment

ef <- efourier(leaf1)
efourier_coefs(ef)

# object is aligned along the major caliper with `poly_align()`
# object is centered on the origin with `poly_center()`
# using a list of object coordinates
ef2 <- efourier(contours, align = TRUE, center = TRUE)
efourier_coefs(ef2)

# reconstruct the perimeter of the object
# Use only the first one for simplicity
plot_polygon(contours[[1]] |> poly_align() |> poly_center())
efourier_inv(ef2[[1]]) |> plot_contour(col = "red", lwd = 4)
```

`efourier_coefs` *Get Fourier coefficients*

Description

Extracts the Fourier coefficients from objects computed with `efourier()` and `efourier_norm()` returning a 'ready-to-analyze' data frame.

Usage

```
efourier_coefs(x)
```

Arguments

x	An object computed with `efourier()` or `efourier_norm()`.

Value

A `data.frame` object

Examples

```
library(pliman)

# a list of objects
efourier(contours) |> efourier_coefs()
```

```
# one object, normalized coefficients
efourier(contours[[4]]) |>
  efourier_norm() |>
  efourier_coefs()
```

`efourier_error` *Erros between the original and reconstructed outline*

Description

Computes the sum of squared distances between the original data and reconstructed outline. It allows examining reconstructed outlines with the addition of successive contributing harmonics indicated in the argument `nharm`.

Usage

```
efourier_error(
  x,
  nharm = NULL,
  type = c("error", "outline", "deviations"),
  plot = TRUE,
  ncol = NULL,
  nrow = NULL
)
```

Arguments

`x`	An object computed with `efourier()`.
`nharm`	An integer or vector of integers indicating the number of harmonics to use. If not specified the number of harmonics used in `x` is used.
`type`	The type of plot to produce. By default, a line plot with the sum of squared distances (y-axis) and the number of harmonics (x-axis) is produced. If `type = "outline"` is used, a plot with the original polygon and the constructed outline is produced. If `type = "deviations"` is used, a plot with the deviations from the original outline and reconstructed outline (y-axis) and points along the outline (x-axis) is produced.
`plot`	A logical to inform if a plot should be produced. Defaults to `TRUE`.
`ncol, nrow`	The number of rows or columns in the plot grid. Defaults to `NULL`, i.e., a square grid is produced.

Value

A list with the objects:

- `dev_points` A list with the deviations (distances) from original and predicted outline for each pixel of the outline.

- `data.frame` object with the minimum, maximum and average deviations (based on the outline points).

If `x` is an object of class `efourier_lst`, a list will be returned.

Examples

```
library(pliman)
ef <-
  contours[[1]] |>
  efourier(nharm = 30)

efourier_error(ef)

efourier_error(ef,
               nharm = 30,
               type = "outline")

efourier_error(ef,
               nharm = c(1, 4, 20),
               type = "deviations")
```

`efourier_inv` *Inverse Elliptical Fourier Analysis*

Description

Performs an inverse elliptical Fourier transformation to construct a shape, given a list with Fourier coefficients computed with `efourier()`.

Usage

```
efourier_inv(x, nharm = NULL, a0 = NULL, c0 = NULL, npoints = 500)
```

Arguments

`x`	An object of class `efourier` or `efourier_lst` computed with `efourier()`.
`nharm`	An integer indicating the number of harmonics to use. If not specified the number of harmonics used in `x` is used.
`a0, c0`	the estimates of the coordinates of the centroid of the configuration. If `NULL` (default), the generated coordinates will be centered on the position of the original shape given by `efourier()`.
`npoints`	The number of interpolated points on the constructed outline. Defaults to 500.

Details

Adapted from Claude (2008). pp. 223.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
library(pliman)
plot_polygon(contours, aspect_ratio = 1)
# without alignment
ef <- efourier(contours, nharm = 10, align = FALSE)
ief <- efourier_inv(ef)
plot_contour(ief, col = "red", lwd = 2)
```

---

efourier_norm *Normalized Fourier coefficients*

Description

The first harmonic defines an ellipse that best fits the outlines. One can use the parameters of the first harmonic to "normalize" the data so that they can be invariant to size, rotation, and starting position of the outline trace. This approach is referred to in the literature as the normalized elliptic Fourier. `efourier_norm()` calculates a new set of Fourier coefficients `An`, `Bn`, `Cn`, `Dn` that one can use for further multivariate analyses (Claude, 2008).

Usage

```
efourier_norm(x, start = FALSE)
```

Arguments

`x`	An object computed with `efourier()`.
`start`	Logical value telling whether the position of the starting point has to be preserved or not.

Details

Adapted from Claude (2008). pp. 226.

Value

A list with the following components:

- `A`, `B`, `C`, `D` for harmonic coefficients.
- `size` the magnitude of the semi-major axis of the first fitting ellipse.
- `theta` angle, in radians, between the starting and the semi-major axis of the first fitting ellipse.
- `psi` orientation of the first fitting ellipse
- `a0` and `c0`, harmonic coefficients.
- `lnef` the concatenation of coefficients.
- `nharm` the number of harmonics used.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
library(pliman)
leaf1 <- contours[[4]]
plot_polygon(leaf1)

# compute the Fourier coefficients
ef <- efourier(leaf1)
efourier_coefs(ef)

# Normalized Fourier coefficients

efn <- efourier_norm(ef)
efourier_coefs(efn)
```

`efourier_power` *Power in Fourier Analysis*

Description

Computes an spectrum of harmonic Fourier power. The power is proportional to the harmonic amplitude and can be considered as a measure of shape information. As the rank of harmonic increases, the power decreases and adds less and less information. We can evaluate the number of harmonics that we must select, so their cumulative power gathers 99% of the total cumulative power (Claude, 2008).

Usage

```
efourier_power(
  x,
  first = TRUE,
  thresh = c(0.8, 0.85, 0.9, 0.95, 0.99, 0.999),
  plot = TRUE,
  ncol = NULL,
  nrow = NULL
)
```

Arguments

`x`	An object of class `efourier`computed with `efourier()`.
`first`	Logical argument indicating whether to include the first harmonic for computing the power. See `Details`.
`thresh`	A numeric vector indicating the threshold power. The number of harmonics needed for such thresholds will then be computed.

plot	Logical argument indicating whether to produce a plot.
ncol, nrow	The number of rows or columns in the plot grid. Defaults to NULL, i.e., a square grid is produced.

Details

Most of the shape "information" is contained in the first harmonic. This is not surprising because this is the harmonic that best fits the outline, and the size of ellipses decreases as for explaining successive residual variation. However, one may think that the first ellipse does not contain relevant shape information, especially when differences one wants to investigate concern complex outlines. By using first = FALSE it is possible to remove the first harmonic for this computation. When working on a set of outlines, high-rank-harmonics can contain information that may allow groups to be distinguished (Claude, 2008).

Adapted from Claude (2008). pp. 229.

Value

A list with the objects:

- cum_power, a data.frame object with the accumulated power depending on the number of harmonics
-

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
library(pliman)
pw <- efourier(contours) |> efourier_power()
```

---

efourier_shape *Draw shapes based on Fourier coefficients*

Description

Calculates a 'Fourier elliptical shape' given Fourier coefficients

Usage

```
efourier_shape(
  an = NULL,
  bn = NULL,
  cn = NULL,
  dn = NULL,
```

```
  n = 1,
  nharm = NULL,
  npoints = 150,
  alpha = 4,
  plot = TRUE
)
```

Arguments

an	The a_n Fourier coefficients on which to calculate a shape.
bn	The b_n Fourier coefficients on which to calculate a shape.
cn	The c_n Fourier coefficients on which to calculate a shape.
dn	The d_n Fourier coefficients on which to calculate a shape.
n	The number of shapes to generate. Defaults to 1. If more than one shape is used, a list of coordinates is returned.
nharm	The number of harmonics to use. It must be less than or equal to the length of *_n coefficients.
npoints	The number of points to calculate.
alpha	The power coefficient associated with the (usually decreasing) amplitude of the Fourier coefficients.
plot	Logical indicating Whether to plot the shape. Defaults to ´TRUE‘

Details

efourier_shape can be used by specifying nharm and alpha. The coefficients are then sampled in an uniform distribution $(-\pi; \pi)$ and this amplitude is then divided by $harmonicrank^{a}lpha$. If alpha is lower than 1, consecutive coefficients will thus increase. See Claude (2008) pp.223 for the maths behind inverse ellipitical Fourier

Adapted from Claude (2008). pp. 223.

Value

A list with components:

- x vector of x-coordrdinates
- y vector of y-coordrdinates.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
library(pliman)
# approximation of the third leaf's perimeter
# 4 harmonics
image_pliman("potato_leaves.jpg", plot = TRUE)
```

```
efourier_shape(an = c(-7.34,  1.81,  -1.32, 0.50),
               bn = c(-113.88, 21.90, -0.31, -6.14),
               cn = c(-147.51, -20.89, 0.66, -14.06),
               dn = c(-0.48, 2.36, -4.36, 3.03))
```

`ellipse` *Confidence ellipse*

Description

Produces a confidence ellipse that is an iso-contour of the Gaussian distribution, allowing to visualize a 2D confidence interval.

Usage

```
ellipse(
  x,
  conf = 0.95,
  np = 100,
  plot = TRUE,
  fill = "green",
  alpha = 0.3,
  random_fill = TRUE
)
```

Arguments

`x`	A `matrix`, a `data.frame` or a `list` of perimeter coordinates, often produced with `object_contour()`.
`conf`	The confidence level. Defaults to `0.95`
`np`	Number of sampled points on the ellipse.
`plot`	Create a plot? Defaults to `TRUE`.
`fill`	The color to fill the ellipse. Defaults to `"green"`.
`alpha`	The alpha value to define the opacity of ellipse. Defaults to `0.3`
`random_fill`	Fill multiple ellipses with random colors? Defaults to `TRUE`.

Value

A matrix with coordinates of points sampled on the ellipse.

Note

Borrowed from Claude (2008), pp. 85

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
library(pliman)
ellipse(contours)
```

`get_pliman_viewer` *Get the value of the pliman_viewer option*

Description

Retrieves the current value of the pliman_viewer option used in the package.

Usage

```
get_pliman_viewer()
```

Value

The current value of the pliman_viewer option.

`ggplot_color` *ggplot2-like colors generation*

Description

Generate ggplot2

Usage

```
ggplot_color(n = 1)
```

Arguments

`n`	The number of colors. This works well for up to about eight colours, but after that it becomes hard to tell the different colours apart.

Examples

```
library(pliman)
ggplot_color(n = 3)
```

image_align *Aligns an* Image *object by hand*

Description

image_align() rotate an image given a line of desired aligment along the y axis that corresponds to the alignment of the objects (e.g., field plots). By default, the aligment will be to the vertical, which means that if the drawed line have an angle < 90º parallel to the x axis, the rotation angle wil be negative (anticlocwise rotation).

Usage

```
image_align(
  img,
  align = c("vertical", "horizontal"),
  viewer = get_pliman_viewer(),
  plot = TRUE
)
```

Arguments

img	An Image object
align	The desired alignment. Either "vertical" (default) or "horizontal".
viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
plot	Plots the aligned image? Defaults to TRUE.

Details

The image_align function aligns an image along the vertical or horizontal axis based on user-selected points. The alignment can be performed in either the base plotting system or using the mapview package for interactive visualization. If the viewer option is set to "base", the function prompts the user to select two points on the image to define the alignment line. If the viewer option is set to "mapview", the function opens an interactive map where the user can draw a polyline to define the alignment line. The alignment angle is calculated based on the selected points, and the image is rotated accordingly using the image_rotate function. The function returns the aligned image object.

Value

The img aligned

Examples

```
if(interactive()){
library(pliman)
flax <- image_pliman("flax_leaves.jpg", plot = TRUE)
aligned <- image_align(flax)
}
```

`image_augment` *Augment Images*

Description

This function takes an image and augments it by rotating it multiple times.

Usage

```
image_augment(
  img,
  pattern = NULL,
  times = 12,
  type = "export",
  dir_original = NULL,
  dir_processed = NULL,
  parallel = FALSE,
  verbose = TRUE
)
```

Arguments

`img`	An `Image` object.
`pattern`	A regular expression pattern to select multiple images from a directory.
`times`	The number of times to rotate the image.
`type`	The type of output: "export" to save images or "return" to return a list of augmented images.
`dir_original`	The directory where original images are located.
`dir_processed`	The directory where processed images will be saved.
`parallel`	Whether to perform image augmentation in parallel.
`verbose`	Whether to display progress messages.

Value

If type is "export," augmented images are saved. If type is "return," a list of augmented images is returned.

Examples

```
if(interactive()){
library(pliman)
img <- image_pliman("sev_leaf.jpg")
imgs <- image_augment(img, type = "return", times = 4)
image_combine(imgs)
}
```

`image_binary` *Creates a binary image*

Description

Reduce a color, color near-infrared, or grayscale images to a binary image using a given color channel (red, green blue) or even color indexes. The Otsu's thresholding method (Otsu, 1979) is used to automatically perform clustering-based image thresholding.

Usage

```
image_binary(
  img,
  index = "R",
  r = 1,
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  return_class = "ebimage",
  threshold = c("Otsu", "adaptive"),
  k = 0.1,
  windowsize = NULL,
  has_white_bg = FALSE,
  resize = FALSE,
  fill_hull = FALSE,
  filter = FALSE,
  invert = FALSE,
  plot = TRUE,
  nrow = NULL,
  ncol = NULL,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE
)
```

Arguments

`img`	An image object.
`index`	A character value (or a vector of characters) specifying the target mode for conversion to binary image. See the available indexes with `pliman_indexes()` and `image_index()` for more details.
`r, g, b, re, nir`	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. Defaults to 1, 2, 3, 4, and 5, respectively. If a multispectral image is provided (5 bands), check the order of bands, which are frequently presented in the 'BGR' format.
`return_class`	The class of object to be returned. If `"terra` returns a SpatRaster object with the number of layers equal to the number of indexes computed. If `"ebimage"` (default) returns a list of `Image` objects, where each element is one index computed.
`threshold`	The theshold method to be used. • By default (`threshold = "Otsu"`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold. • If `threshold = "adaptive"`, adaptive thresholding (Shafait et al. 2008) is used, and will depend on the `k` and `windowsize` arguments. • If any non-numeric value different than `"Otsu"` and `"adaptive"` is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index.
`k`	a numeric in the range 0-1. when `k` is high, local threshold values tend to be lower. when `k` is low, local threshold value tend to be higher.
`windowsize`	windowsize controls the number of local neighborhood in adaptive thresholding. By default it is set to `1/3 * minxy`, where `minxy` is the minimum dimension of the image (in pixels).
`has_white_bg`	Logical indicating whether a white background is present. If `TRUE`, pixels that have R, G, and B values equals to 1 will be considered as `NA`. This may be useful to compute an image index for objects that have, for example, a white background. In such cases, the background will not be considered for the threshold computation.
`resize`	Resize the image before processing? Defaults to `FALSE`. Use a numeric value as the percentage of desired resizing. For example, if `resize = 30`, the resized image will have 30% of the size of original image.
`fill_hull`	Fill holes in the objects? Defaults to `FALSE`.
`filter`	Performs median filtering in the binary image? (Defaults to `FALSE`). Provide a positive integer > 1 to indicate the size of the median filtering. Higher values are more efficient to remove noise in the background but can dramatically impact the perimeter of objects, mainly for irregular perimeters such as leaves with serrated edges.
`invert`	Inverts the binary image, if desired.
`plot`	Show image after processing?

nrow, ncol	The number of rows or columns in the plot grid. Defaults to NULL, i.e., a square grid is produced.
parallel	Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time when image is a list. The number of sections is set up to 70% of available cores.
workers	A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.
verbose	If TRUE (default) a summary is shown in the console.

Value

A list containing binary images. The length will depend on the number of indexes used.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

References

Otsu, N. 1979. Threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern SMC-9(1): 62–66. doi:10.1109/tsmc.1979.4310076

Shafait, F., D. Keysers, and T.M. Breuel. 2008. Efficient implementation of local adaptive thresholding techniques using integral images. Document Recognition and Retrieval XV. SPIE. p. 317–322 doi:10.1117/12.767755

Examples

```
library(pliman)
img <- image_pliman("soybean_touch.jpg")
image_binary(img, index = c("R, G"))
```

---

image_combine *Combines images to a grid*

Description

Combines several images to a grid

Usage

```
image_combine(
  ...,
  labels = NULL,
  nrow = NULL,
  ncol = NULL,
  col = "black",
  verbose = TRUE
)
```

Arguments

...	a comma-separated name of image objects or a list containing image objects.
labels	A character vector with the same length of the number of objects in ... to indicate the plot labels.
nrow, ncol	The number of rows or columns in the plot grid. Defaults to NULL, i.e., a square grid is produced.
col	The color for the plot labels. Defaults to col = "black".
verbose	Shows the name of objects declared in ... or a numeric sequence if a list with no names is provided. Set to FALSE to supress the text.

Value

A grid with the images in ...

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
library(pliman)
img1 <- image_pliman("sev_leaf.jpg")
img2 <- image_pliman("sev_leaf_nb.jpg")
image_combine(img1, img2)
```

---

image_create *Create an* Image *object of a given color*

Description

image_create() can be used to create an Image object with a desired color and size.

Usage

```
image_create(color, width = 200, heigth = 200, plot = FALSE)
```

Arguments

color	either a color name (as listed by grDevices::colors()), or a hexadecimal string of the form "#rrggbb".
width, heigth	The width and heigth of the image in pixel units.
plot	Plots the image after creating it? Defaults to FALSE.

Value

An object of class Image.

Examples

```
image_create("red")
image_create("#009E73", width = 300, heigth = 100)
```

`image_expand` *Expands an image*

Description

Expands an image towards the left, top, right, or bottom by sampling pixels from the image edge. Users can choose how many pixels (rows or columns) are sampled and how many pixels the expansion will have.

Usage

```
image_expand(
  img,
  left = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  edge = NULL,
  sample_left = 10,
  sample_top = 10,
  sample_right = 10,
  sample_bottom = 10,
  random = FALSE,
  filter = NULL,
  plot = TRUE
)
```

Arguments

`img`	An `Image` object.
`left, top, right, bottom`	The number of pixels to expand in the left, top, right, and bottom directions, respectively.
`edge`	The number of pixels to expand in all directions. This can be used to avoid calling all the above arguments
`sample_left, sample_top, sample_right, sample_bottom`	The number of pixels to sample from each side. Defaults to 20.
`random`	Randomly sampling of the edge's pixels? Defaults to `FALSE`.
`filter`	Apply a median filter in the sampled pixels? Defaults to `FALSE`.
`plot`	Plots the extended image? defaults to `FALSE`.

Value

An Image object

Examples

```
library(pliman)
img <- image_pliman("soybean_touch.jpg")
image_expand(img, left = 200)
image_expand(img, right = 150, bottom = 250, filter = 5)
```

`image_index` *Image indexes*

Description

`image_index()` Builds image indexes using Red, Green, Blue, Red-Edge, and NIR bands.

Generates a raster or density plot of the index values computed with `image_index()`.

Usage

```
image_index(
  img,
  index = NULL,
  r = 1,
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  return_class = c("ebimage", "terra"),
  resize = FALSE,
  has_white_bg = FALSE,
  plot = TRUE,
  nrow = NULL,
  ncol = NULL,
  max_pixels = 1e+05,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'image_index'
plot(x, type = c("raster", "density"), nrow = NULL, ncol = NULL, ...)
```

Arguments

img	An Image object. Multispectral mosaics can be converted to an Image object using mosaic_as_ebimage().
index	A character value (or a vector of characters) specifying the target mode for conversion to a binary image. Use pliman_indexes() or the details section to see the available indexes. Defaults to NULL (normalized Red, Green, and Blue). You can also use "RGB" for RGB only, "NRGB" for normalized RGB, "MULTISPECTRAL" for multispectral indices (provided NIR and RE bands are available) or "all" for all indexes. Users can also calculate their own index using the band names, e.g., index = "R+B/G".
r, g, b, re, nir	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. Defaults to 1, 2, 3, 4, and 5, respectively. If a multispectral image is provided (5 bands), check the order of bands, which are frequently presented in the 'BGR' format.
return_class	The class of object to be returned. If "terra returns a SpatRaster object with the number of layers equal to the number of indexes computed. If "ebimage" (default) returns a list of Image objects, where each element is one index computed.
resize	Resize the image before processing? Defaults to resize = FALSE. Use resize = 50, which resizes the image to 50% of the original size to speed up image processing.
has_white_bg	Logical indicating whether a white background is present. If TRUE, pixels that have R, G, and B values equals to 1 will be considered as NA. This may be useful to compute an image index for objects that have, for example, a white background. In such cases, the background will not be considered for the threshold computation.
plot	Show image after processing?
nrow, ncol	The number of rows or columns in the plot grid. Defaults to NULL, i.e., a square grid is produced.
max_pixels	integer > 0. Maximum number of cells to plot the index. If max_pixels < npixels(img), downsampling is performed before plotting the index. Using a large number of pixels may slow down the plotting time.
parallel	Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time when image is a list. The number of sections is set up to 70% of available cores.
workers	A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.
verbose	If TRUE (default) a summary is shown in the console.
...	Additional arguments passed to plot_index() for customization.
x	An object of class image_index.
type	The type of plot. Use type = "raster" (default) to produce a raster plot showing the intensity of the pixels for each image index or type = "density" to produce a density plot with the pixels' intensity.

Details

The following indexes are available in pliman.

- RGB color space
- `R` red
- `G` green
- `B` blue
- `NR` normalized red `R/(R+G+B)`.
- `NG` normalized green `G/(R+G+B)`
- `NB` normalized blue `B/(R+G+B)`
- `GB` green blue ratio `G/B`
- `RB` red blue ratio `R/B`
- `GR` green red ratio `G/R`
- `BI` brightness Index `sqrt((R^2+G^2+B^2)/3)`
- `BIM` brightness Index 2 `sqrt((R*2+G*2+B*2)/3)`
- `SCI` Soil Colour Index `(R-G)/(R+G)`
- `GLI` Green leaf index Vis Louhaichi et al. (2001) `(2*G-R-B)/(2*G+R+B)`
- `HI` Primary colours Hue Index (2*R-G-B)/(G-B)
- `NDGRI` Normalized green red difference index (Tucker, 1979) `(G-R)/(G+R)`
- `NDGBI` Normalized green blue difference index `(G-B)/(G+B)`
- `NDRBI` Normalized red blue difference index `(R-B)/(R+B)`
- `I` R+G+B
- `S` `((R+G+B)-3*B)/(R+G+B)`
- `L` R+G+B/3
- `VARI` A Visible Atmospherically Resistant Index `(G-R)/(G+R-B)`
- `HUE` Overall Hue Index `atan(2*(B-G-R)/30.5*(G-R))`
- `HUE2` atan(2*(R-G-R)/30.5*(G-B))
- `BGI` B/G
- `GRAY` `0.299*R + 0.587*G + 0.114*B`
- `GRAY2` `((R^2.2+(1.5*G)^2.2+(0.6*B)^2.2)/(1+1.5^2.2+0.6^2.2))^1/2.2`
- `GLAI` `(25*(G-R)/(G+R-B)+1.25)`
- `CI` Coloration Index `(R-B)/R`
- `SAT` Overhall Saturation Index `(max(R,G,B) - min(R,G,B)) / max(R,G,B)`
- `SHP` Shape Index `2*(R-G-B)/(G-B)`
- `RI` Redness Index `R**2/(B*G**3)`
- HSB color space
- `DGCI` Dark Green Color Index, based on HSB color space `60\*((G - B) / (max(R, G, B) - min(R, G, B)))`

- CIE-Lab color space
- L*: relative luminance (0.2126 * R + 0.7152 * G + 0.0722 * B)
- a*: 0.55*((R - (0.2126 * R + 0.7152 * G + 0.0722 * B)) / (1.0 - 0.2126))

When type = "raster" (default), the function calls plot_index() to create a raster plot for each index present in x. If type = "density", a for loop is used to create a density plot for each index. Both types of plots can be arranged in a grid controlled by the ncol and nrow arguments.

Value

A list containing Grayscale images. The length will depend on the number of indexes used.

A NULL object

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

References

Nobuyuki Otsu, "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber. 9 (1): 62-66. 1979. doi:10.1109/TSMC.1979.4310076

Examples

```
library(pliman)
img <- image_pliman("soybean_touch.jpg")
image_index(img, index = c("R, NR"))

# Example for S3 method plot()
library(pliman)
img <- image_pliman("sev_leaf.jpg")
# compute the index
ind <- image_index(img, index = c("R, G, B, NGRDI"), plot = FALSE)
plot(ind)

# density plot
plot(ind, type = "density")
```

---

image_prepare *Prepare an image*

Description

This function aligns and crops the image using either base or mapview visualization. This is useful to prepare the images to be analyzed with analyze_objects_shp()

Usage

```
image_prepare(img, viewer = get_pliman_viewer())
```

Arguments

img	An optional Image object
viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.

Value

The alighed/cropped image for further visualization or analysis.

Examples

```
# Example usage:
if(interactive()){
img <- image_pliman("mult_leaves.jpg")
image_prepare(img, viewer = "mapview")
}
```

---

image_segment *Image segmentation*

Description

- image_segment() reduces a color, color near-infrared, or grayscale images to a segmented image using a given color channel (red, green blue) or even color indexes (See image_index() for more details). The Otsu's thresholding method (Otsu, 1979) is used to automatically perform clustering-based image thresholding.
- image_segment_iter() Provides an iterative image segmentation, returning the proportions of segmented pixels.

Usage

```
image_segment(
  img,
  index = NULL,
  r = 1,
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  threshold = c("Otsu", "adaptive"),
  k = 0.1,
```

```
  windowsize = NULL,
  col_background = NULL,
  has_white_bg = FALSE,
  fill_hull = FALSE,
  filter = FALSE,
  invert = FALSE,
  plot = TRUE,
  nrow = NULL,
  ncol = NULL,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE
)

image_segment_iter(
  img,
  nseg = 2,
  index = NULL,
  invert = NULL,
  threshold = NULL,
  k = 0.1,
  windowsize = NULL,
  has_white_bg = FALSE,
  plot = TRUE,
  verbose = TRUE,
  nrow = NULL,
  ncol = NULL,
  parallel = FALSE,
  workers = NULL,
  ...
)
```

Arguments

`img`	An image object or a list of image objects.
`index`	• For `image_segment()`, a character value (or a vector of characters) specifying the target mode for conversion to binary image. See the available indexes with `pliman_indexes()`. See `image_index()` for more details. • For `image_segment_iter()` a character or a vector of characters with the same length of `nseg`. It can be either an available index (described above) or any operation involving the RGB values (e.g., `"B/R+G"`).
`r, g, b, re, nir`	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. Defaults to 1, 2, 3, 4, and 5, respectively. If a multispectral image is provided (5 bands), check the order of bands, which are frequently presented in the 'BGR' format.
`threshold`	The theshold method to be used. • By default (`threshold = "Otsu"`), a threshold value based on Otsu's method

is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold.
- If `threshold = "adaptive"`, adaptive thresholding (Shafait et al. 2008) is used, and will depend on the `k` and `windowsize` arguments.
- If any non-numeric value different than `"Otsu"` and `"adaptive"` is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index.

`k` a numeric in the range 0-1. when `k` is high, local threshold values tend to be lower. when `k` is low, local threshold value tend to be higher.

`windowsize` windowsize controls the number of local neighborhood in adaptive thresholding. By default it is set to `1/3 * minxy`, where `minxy` is the minimum dimension of the image (in pixels).

`col_background` The color of the segmented background. Defaults to `NULL` (white background).

`has_white_bg` Logical indicating whether a white background is present. If `TRUE`, pixels that have R, G, and B values equals to 1 will be considered as `NA`. This may be useful to compute an image index for objects that have, for example, a white background. In such cases, the background will not be considered for the threshold computation.

`fill_hull` Fill holes in the objects? Defaults to `FALSE`.

`filter` Performs median filtering in the binary image? See more at `image_filter()`. Defaults to `FALSE`. Use a positive integer to define the size of the median filtering. Larger values are effective at removing noise, but adversely affect edges.

`invert` Inverts the binary image, if desired. For `image_segmentation_iter()` use a vector with the same length of `nseg`.

`plot` Show image after processing?

`nrow, ncol` The number of rows or columns in the plot grid. Defaults to `NULL`, i.e., a square grid is produced.

`parallel` Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time when `image` is a list. The number of sections is set up to 70% of available cores.

`workers` A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.

`verbose` If `TRUE` (default) a summary is shown in the console.

`nseg` The number of iterative segmentation steps to be performed.

`...` Additional arguments passed on to `image_segment()`.

Value

- `image_segment()` returns list containing `n` objects where `n` is the number of indexes used. Each objects contains:
  - `image` an image with the RGB bands (layers) for the segmented object.
  - `mask` A mask with logical values of 0 and 1 for the segmented image.
- `image_segment_iter()` returns a list with (1) a data frame with the proportion of pixels in the segmented images and (2) the segmented images.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

References

Nobuyuki Otsu, "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber. 9 (1): 62-66. 1979. doi:10.1109/TSMC.1979.4310076

Examples

```
library(pliman)
img <- image_pliman("soybean_touch.jpg", plot = TRUE)
image_segment(img, index = c("R, G, B"))

# adaptive thresholding
```

---

image_segment_kmeans *Image segmentation using k-means clustering*

Description

Segments image objects using clustering by the k-means clustering algorithm

Usage

```
image_segment_kmeans(
  img,
  bands = 1:3,
  nclasses = 2,
  invert = FALSE,
  filter = FALSE,
  fill_hull = FALSE,
  plot = TRUE
)
```

Arguments

img	An Image object.
bands	A numeric integer/vector indicating the RGB band used in the segmentation. Defaults to 1:3, i.e., all the RGB bands are used.
nclasses	The number of desired classes after image segmentation.
invert	Invert the segmentation? Defaults to FALSE. If TRUE the binary matrix is inverted.
filter	Applies a median filtering in the binary matrix? Defaults to FALSE. Use a numeric integer to indicate the size of the median filter.
fill_hull	Fill holes in the objects? Defaults to FALSE.
plot	Plot the segmented image?

Value

A list with the following values:

- `image` The segmented image considering only two classes (foreground and background)
- `clusters` The class of each pixel. For example, if `ncluster = 3`, `clusters` will be a two-way matrix with values ranging from 1 to 3. `masks` A list with the binary matrices showing the segmentation.

References

Hartigan, J. A. and Wong, M. A. (1979). Algorithm AS 136: A K-means clustering algorithm. Applied Statistics, 28, 100–108. doi:10.2307/2346830

Examples

```
img <- image_pliman("la_leaves.jpg", plot = TRUE)
seg <- image_segment_kmeans(img)
seg <- image_segment_kmeans(img, fill_hull = TRUE, invert = TRUE, filter = 10)
```

---

image_segment_manual *Image segmentation by hand*

Description

This R code is a function that allows the user to manually segment an image based on the parameters provided. This only works in an interactive section.

Usage

```
image_segment_manual(
  img,
  shape = c("free", "circle", "rectangle"),
  type = c("select", "remove"),
  viewer = get_pliman_viewer(),
  resize = TRUE,
  edge = 5,
  plot = TRUE
)
```

Arguments

`img`	An `Image` object.
`shape`	The type of shape to use. Defaults to "free". Other possible values are "circle" and "rectangle". Partial matching is allowed.
`type`	The type of segmentation. By default (`type = "select"`) objects are selected. Use `type = "remove"` to remove the selected area from the image.

viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
resize	By default, the segmented object is resized to fill the original image size. Use resize = FALSE to keep the segmented object in the original scale.
edge	Number of pixels to add in the edge of the segmented object when resize = TRUE. Defaults to 5.
plot	Plot the segmented object? Defaults to TRUE.

Details

If the shape is "free", it allows the user to draw a perimeter to select/remove objects. If the shape is "circle", it allows the user to click on the center and edge of the circle to define the desired area. If the shape is "rectangle", it allows the user to select two points to define the area.

Value

A list with the segmented image and the mask used for segmentation.

Examples

```
if (interactive()) {
img <- image_pliman("la_leaves.jpg")
seg <- image_segment_manual(img)
plot(seg$mask)

}
```

---

image_segment_mask *Segment an* Image *object using a brush mask*

Description

It combines make_mask() and make_brush() to segment an Image object using a brush of desired size, shape, and position.

Usage

```
image_segment_mask(
  img,
  size,
  shape = "disc",
```

```
  rel_pos_x = 0.5,
  rel_pos_y = 0.5,
  type = c("binary", "shadow"),
  col_background = "white",
  plot = TRUE,
  ...
)
```

Arguments

`img`	A `Image` object
`size`	A numeric containing the size of the brush in pixels. This should be an odd number; even numbers are rounded to the next odd one.
`shape`	A character vector indicating the shape of the brush. Can be `"box"`, `"disc"`, `"diamond"`, `"Gaussian"` or `"line"` Defaults to `"disc"`.
`rel_pos_x, rel_pos_y`	A relative position to include the brush in the image. Defaults to 0.5. This means that the brush will be centered in the original image. Smaller values move the brush toward the left and top, respectively.
`type`	Defines the type of the mask. By default, a binary mask is applied. This results in white pixels in the original image that matches the 0s pixels in the brush. If `type = "shadow"` is used, a shadow mask is produced
`col_background`	Background color after image segmentation. Defaults to `"white"`.
`plot`	Plots the generated mask? Defaults to `TRUE`.
`...`	Further arguments passed on to `EBImage::makeBrush()`.

Value

A color `Image` object

Examples

```
img <- image_pliman("soybean_touch.jpg")
plot(img)
image_segment_mask(img, size = 601)
image_segment_mask(img,
                   size = 401,
                   shape = "diamond",
                   rel_pos_x = 0,
                   rel_pos_y = 0,
                   type = "shadow")
```

image_shp *Construct a shape file from an image*

Description

Creates a list of object coordinates given the desired number of nrow and columns. It starts by selecting 4 points at the corners of objects of interest in the plot space. Then, given `nrow` and `ncol`, a grid is drawn and the objects' coordinates are returned.

Usage

```
image_shp(
  img,
  nrow = 1,
  ncol = 1,
  buffer_x = 0,
  buffer_y = 0,
  interactive = FALSE,
  viewer = get_pliman_viewer(),
  col_line = "red",
  size_line = 2,
  col_text = "red",
  size_text = 1,
  plot = TRUE
)
```

Arguments

`img`	An object of class `Image`
`nrow`	The number of desired rows in the grid. Defaults to `1`.
`ncol`	The number of desired columns in the grid. Defaults to `1`.
`buffer_x, buffer_y`	Buffering factor for the width and height, respectively, of each individual shape's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the shape by 25% on each side.
`interactive`	If `FALSE` (default) the grid is created automatically based on the image dimension and number of rows/columns. If `interactive = TRUE`, users must draw points at the diagonal of the desired bounding box that will contain the grid.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are `"base"` and `"mapview"`. If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run

set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.

col_line, col_text
: The color of the line/text in the grid. Defaults to "red".

size_line, size_text
: The size of the line/text in the grid. Defaults to 2.5.

plot
: Plots the grid on the image? Defaults to TRUE.

Value

A list with row * col objects containing the plot coordinates.

Examples

```
library(pliman)
flax <- image_pliman("flax_leaves.jpg")
shape <- image_shp(flax, nrow = 3, ncol = 5)
```

---

image_square *Squares an image*

Description

Converts a rectangular image into a square image by expanding the rows/columns using image_expand().

Usage

```
image_square(img, plot = TRUE, ...)
```

Arguments

img
: An Image object.

plot
: Plots the extended image? defaults to FALSE.

...
: Further arguments passed on to image_expand().

Value

The modified Image object.

Examples

```
library(pliman)
img <- image_pliman("soybean_touch.jpg")
dim(img)
square <- image_square(img)
dim(square)
```

image_thinning_guo_hall *Perform Guo-Hall thinning on a binary image or list of binary images*

Description

This function performs the Guo-Hall thinning algorithm (Guo and Hall, 1989) on a binary image or a list of binary images.

Usage

```
image_thinning_guo_hall(
  img,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE,
  ...
)
```

Arguments

img	The binary image or a list of binary images to be thinned. It can be either a single binary image of class 'Image' or a list of binary images.
parallel	Logical, whether to perform thinning using multiple cores (parallel processing). If TRUE, the function will use multiple cores for processing if available. Default is FALSE.
workers	Integer, the number of workers (cores) to use for parallel processing. If NULL (default), it will use 40% of available cores.
verbose	Logical, whether to display progress messages during parallel processing. Default is TRUE.
plot	Logical, whether to plot the thinned images. Default is FALSE.
...	Additional arguments to be passed to image_binary() if img is not a binary image.

Value

If img is a single binary image, the function returns the thinned binary image. If img is a list of binary images, the function returns a list containing the thinned binary images.

References

Guo, Z., and R.W. Hall. 1989. Parallel thinning with two-subiteration algorithms. Commun. ACM 32(3): 359–373. doi:10.1145/62065.62074

Examples

```
library(pliman)
img <- image_pliman("potato_leaves.jpg", plot = TRUE)
image_thinning_guo_hall(img, index = "R", plot = TRUE)
```

`image_to_mat` *Convert an image to a data.frame*

Description

Given an object image, converts it into a data frame where each row corresponds to the intensity values of each pixel in the image.

Usage

```
image_to_mat(img, parallel = FALSE, workers = NULL, verbose = TRUE)
```

Arguments

`img`	An image object.
`parallel`	Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time when `image` is a list. The number of sections is set up to 70% of available cores.
`workers`	A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.
`verbose`	If `TRUE` (default) a summary is shown in the console.

Value

A list containing three matrices (R, G, and B), and a data frame containing four columns: the name of the image in `image` and the R, G, B values.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
library(pliman)
img <- image_pliman("sev_leaf.jpg")
dim(img)
mat <- image_to_mat(img)
dim(mat[[1]])
```

image_view *Create an interactive map view of an image*

Description

This function allows users to interactively edit and analyze an image using mapview and mapedit packages.

Usage

```
image_view(
  img,
  object = NULL,
  r = 1,
  g = 2,
  b = 3,
  alpha = 0.7,
  attribute = "area",
  title = "Edit the image",
  show = c("rgb", "index"),
  index = "B",
  max_pixels = 1e+06,
  downsample = NULL,
  color_regions = custom_palette(),
  quantiles = c(0, 1),
  domain = NULL,
  ...
)
```

Arguments

img	An Image object.
object	(Optional). An object computed with analyze_objects(). If an object is informed, an additional layer is added to the plot, showing the contour of the analyzed objects, with a color gradient defined by attribute.
r, g, b	The layer for the Red, Green and Blue band, respectively. Defaults to 1, 2, and 3.
alpha	The transparency level of the rectangles' color (between 0 and 1).
attribute	The name of the quantitative variable in the object_index to be used for coloring the rectangles.
title	The title of the map view. Use to provide short orientations to the user.
show	The display option for the map view. Options are "rgb" for RGB view and "index" for index view.
index	The index to use for the index view. Defaults to "B".

max_pixels	integer > 0. Maximum number of cells to use for the plot. If max_pixels < npixels(img), regular sampling is used before plotting.
downsample	integer; for each dimension the number of pixels/lines/bands etc that will be skipped; Defaults to NULL, which will find the best downsampling factor to approximate the max_pixels value.
color_regions	The color palette for displaying index values. Default is custom_palette().
quantiles	the upper and lower quantiles used for color stretching. If set to NULL, stretching is performed basing on 'domain' argument.
domain	the upper and lower values used for color stretching. This is used only if 'quantiles' is NULL. If both 'domain' and 'quantiles' are set to NULL, stretching is applied based on min-max values.
...	Additional arguments to be passed to downsample_fun.

Value

An sf object, the same object returned by mapedit::editMap().

Examples

```
if(interactive()){
# Example usage:
img <- image_pliman("sev_leaf.jpg")
image_view(img)
}
```

---

landmarks *Create image landmarks*

Description

An interactive section where the user will be able to click on the image to select landmarks manually is open. With each mouse click, a point is drawn and an upward counter is shown in the console. After n counts or after the user press Esc, the interactive process is interrupted and a data.frame with the x and y coordinates for the landmarks is returned.

Usage

```
landmarks(
  img,
  n = Inf,
  viewer = get_pliman_viewer(),
  scale = NULL,
  calibrate = FALSE
)
```

Arguments

img	An Image object.
n	The number of landmarks to produce. Defaults to Inf. In this case, landmarks are chosen up to the user press Esc.
viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
scale	A known scale of the coordinate values. If NULL (default) scale = 1 is used.
calibrate	A logical argument indicating whether a calibration step must be performed before picking up the landmarks. If so, calibrate() is called internally. Users must then select two points and indicate a known distance. A scale value will internally be computed and used in the correction of the coordinates (from pixels to the unit of the known distance).

Value

A data.frame with the x and y-coordinates from the landmarks.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if(isTRUE(interactive())){
library(pliman)
img <- image_pliman("potato_leaves.jpg")
x <- landmarks(img)
}
```

---

landmarks_add *Artificially inflates the number of landmarks*

Description

Interpolates supplementary landmarks that correspond to the mean coordinates of two adjacent landmarks.

Usage

```
landmarks_add(x, n = 3, smooth_iter = 0, plot = TRUE, nrow = NULL, ncol = NULL)
```

Arguments

x	A matrix, a data.frame a list of perimeter coordinates, often produced with object_contour(), landmarks(), or landmarks_regradi().
n	The number of iterations. Defaults to 3.
smooth_iter	The number of smoothing iterations to perform. This will smooth the perimeter of the interpolated landmarks using poly_smooth().
plot	Creates a plot? Defaults to TRUE.
ncol, nrow	The number of rows or columns in the plot grid when a list is used in x. Defaults to NULL, i.e., a square grid is produced.

Value

A Matrix of interpolated coordinates.

Examples

```
library(pliman)

# equally spaced landmarks
plot_polygon(contours[[4]])
ldm <- landmarks_regradi(contours[[4]], plot = FALSE)
points(ldm$coords, pch = 16)
segments(mean(ldm$coords[,1]),
         mean(ldm$coords[,2]),
         ldm$coords[,1],
         ldm$coords[,2])

ldm_add <- landmarks_add(ldm, plot = FALSE)
points(ldm_add, col = "red")
points(ldm$coords, pch = 16)

# smoothed version
ldm_add_smo <- landmarks_add(ldm, plot = FALSE, smooth_iter = 10)
lines(ldm_add_smo, col = "blue", lwd = 3)
```

---

landmarks_angle *Angles between landmarks*

Description

Computes the angle from two interlandmark vectors using the difference of their arguments using complex vectors (Claude, 2008).

Usage

```
landmarks_angle(x, unit = c("rad", "deg"))
```

Arguments

x	An object computed with landmarks().
unit	The unit of the angle. Defaults to radian (rad). Use unit = "deg" to return the angles in degrees.

Value

A matrix with the angles for each landmark combination.

Note

Borrowed from Claude (2008), pp. 50

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if(isTRUE(interactive())){
library(pliman)
img <- image_pliman("potato_leaves.jpg")
x <- landmarks(img)
landmarks_angle(x)
}
```

---

landmarks_dist *Distances between landmarks*

Description

Computes the distance between two landmarks as the square root of the sum of the squared differences between each coordinate (Claude, 2008).

Usage

```
landmarks_dist(x)
```

Arguments

x	An object computed with landmarks().

Value

A matrix with the distances for each landmark combination.

Note

Borrowed from Claude (2008), pp. 49

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if(isTRUE(interactive())){
library(pliman)
img <- image_pliman("potato_leaves.jpg")
x <- landmarks(img)
landmarks_dist(x)
}
```

`landmarks_regradi` *Pseudolandmarks with equally spaced angles*

Description

Select `n` landmarks that are spaced with a regular sequence of angles taken between the outline coordinates and the centroid.

Usage

```
landmarks_regradi(
  x,
  n = 50,
  close = TRUE,
  plot = TRUE,
  ncol = NULL,
  nrow = NULL
)
```

Arguments

`x`	A `matrix`, a `data.frame` a `list` of perimeter coordinates, often produced with `object_contour()`.
`n`	Number of points to be sampled. Defaults to 50.
`close`	Return a closed polygon? Defaults to `TRUE`.
`plot`	Create a plot? Defaults to `TRUE`.
`ncol, nrow`	The number of rows or columns in the plot grid when a `list` is used in `x`. Defaults to `NULL`, i.e., a square grid is produced.

Value

A list with the following objects:

- `pixindices`: Vector of radius indices.
- `radii`: Vector of sampled radii lengths.
- `Xc`: The centroid coordinate of `x` axis.
- `Yc`: The centroid coordinate of `y` axis.
- `coords`: Coordinates of sampled points arranged in a two-column matrix.

If `x` is a list, a list of objects described above is returned.

Note

Borrowed from Claude (2008), pp. 53

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
library(pliman)
plot_polygon(contours[[1]])
ldm <- landmarks_regradi(contours)
```

---

leading_zeros *Add leading zeros to a numeric sequence*

Description

Add `n` leading zeros to a numeric sequence. This is useful to create a character vector to rename files in a folder.

Usage

```
leading_zeros(x, n = 3)
```

Arguments

`x`	A numeric vector or a list of numeric vectors.
`n`	The number of leading zeros to add. Defaults to `3`.

Value

A character vector or a list of character vectors.

Examples

```
library(pliman)
leading_zeros(1:5)
leading_zeros(list(a = 1:3,
                   b = 1:5),
              n = 2)
```

`make_brush` *Makes a brush*

Description

Generates brushes of various sizes and shapes that can be used as structuring elements. See `EBImage::makeBrush()`.

Usage

```
make_brush(size, shape = "disc", ...)
```

Arguments

`size`	A numeric containing the size of the brush in pixels. This should be an odd number; even numbers are rounded to the next odd one.
`shape`	A character vector indicating the shape of the brush. Can be `"box"`, `"disc"`, `"diamond"`, `"Gaussian"` or `"line"` Defaults to `"disc"`.
`...`	Further arguments passed on to `EBImage::makeBrush()`.

Value

A 2D matrix of 0s and 1s containing the desired brush.

Examples

```
make_brush(size = 51) |> image()
make_brush(size = 51, shape = "diamond") |> image()
```

make_mask *Makes a mask in an image*

Description

Make a mask using an Image object and a brush.

Usage

```
make_mask(img, brush, rel_pos_x = 0.5, rel_pos_y = 0.5, plot = TRUE)
```

Arguments

img	A Image object
brush	An object created with make_brush()
rel_pos_x, rel_pos_y	A relative position to include the brush in the image. Defaults to 0.5. This means that the brush will be centered in the original image. Smaller values move the brush toward the left and top, respectively.
plot	Plots the generated mask? Defaults to TRUE.

Details

It applies a brush to an Image, selecting the Image pixels that match the brush values equal to 1. The position of the brush in the original image is controlled by the relative positions x (rel_pos_x) and y (rel_pos_y) arguments. The size of the brush must be smaller or equal to the smaller dimension of image.

Value

A binary image with 0s and 1s.

Examples

```
img <- image_pliman("soybean_touch.jpg")
make_mask(img, brush = make_brush(size = 201))
make_mask(img,
          brush = make_brush(size = 401, shape = "diamond"),
          rel_pos_x = 0.1,
          rel_pos_y = 0.8)
```

measure_disease *Performs plant disease measurements*

Description

- measure_disease() computes the percentage of symptomatic leaf area and (optionally) counts and compute shapes (area, perimeter, radius, etc.) of lesions in a sample or entire leaf using color palettes. See more at **Details**.
- measure_disease_iter() provides an iterative section for measure_disease(), where the user picks up samples in the image to create the needed color palettes.

Usage

```
measure_disease(
  img,
  img_healthy = NULL,
  img_symptoms = NULL,
  img_background = NULL,
  pattern = NULL,
  filter = 10,
  parallel = FALSE,
  workers = NULL,
  resize = FALSE,
  fill_hull = TRUE,
  index_lb = NULL,
  index_dh = "GLI",
  has_white_bg = FALSE,
  threshold = NULL,
  invert = FALSE,
  lower_size = NULL,
  upper_size = NULL,
  topn_lower = NULL,
  topn_upper = NULL,
  randomize = TRUE,
  nsample = 3000,
  watershed = FALSE,
  lesion_size = "medium",
  tolerance = NULL,
  extension = NULL,
  show_features = FALSE,
  show_segmentation = FALSE,
  plot = TRUE,
  show_original = TRUE,
  show_background = TRUE,
  show_contour = TRUE,
  contour_col = "white",
```

```
  contour_size = 1,
  col_leaf = NULL,
  col_lesions = NULL,
  col_background = NULL,
  marker = FALSE,
  marker_col = NULL,
  marker_size = NULL,
  save_image = FALSE,
  prefix = "proc_",
  name = NULL,
  dir_original = NULL,
  dir_processed = NULL,
  verbose = TRUE
)

measure_disease_iter(
  img,
  has_background = TRUE,
  r = 2,
  viewer = get_pliman_viewer(),
  show = "rgb",
  index = "NGRDI",
  ...
)
```

Arguments

`img`	The image to be analyzed.
`img_healthy`	A color palette of healthy tissues.
`img_symptoms`	A color palette of lesioned tissues.
`img_background`	A color palette of the background (if exists). These arguments can be either an `Image` object stored in the global environment or a character value. If a chacarceter is used (eg., `img_healthy = "leaf"`), the function will search in the current working directory a valid image that contains `"leaf"` in the name. Note that if two images matches this pattern, an error will occour.
`pattern`	A pattern of file name used to identify images to be processed. For example, if `pattern = "im"` all images that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be analyzed. Providing any number as pattern (e.g., `pattern = "1"`) will select images that are named as 1.-, 2.-, and so on.
`filter`	Performs median filtering in the binary image that segments the leaf from background? By default, a median filter of `size = 10` is applied. This is useful to reduce the noise and segment the leaf and background more accurately. See more at `image_filter()`. Set to `FALSE` to cancel median filtering.
`parallel`	Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time, especially when `pattern` is used is informed. The number of sections is set up to 30% of available cores.

workers — A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.

resize — Resize the image before processing? Defaults to FALSE. Use a numeric value of range 0-100 (proportion of the size of the original image).

fill_hull — Fill holes in the image? Defaults to TRUE. This is useful to fill holes in leaves, e.g., those caused by insect attack, ensuring the hole area will be accounted for the leaf, not background.

index_lb — The index used to segment the foreground (e.g., leaf) from the background. If not declared, the entire image area (pixels) will be considered in the computation of the severity.

index_dh — The index used to segment diseased from healthy tissues when img_healthy and img_symptoms are not declared. Defaults to "GLI". See image_index() for more details.

has_white_bg — Logical indicating whether a white background is present. If TRUE, pixels that have R, G, and B values equals to 1 will be considered as NA. This may be useful to compute an image index for objects that have, for example, a white background. In such cases, the background will not be considered for the threshold computation.

threshold — By default (threshold = NULL), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold. Inform any non-numeric value different than "Otsu" to iteratively choose the threshold based on a raster plot showing pixel intensity of the index. Must be a vector of length 2 to indicate the threshold for index_lb and index_dh, respectively.

invert — Inverts the binary image if desired. This is useful to process images with black background. Defaults to FALSE.

lower_size — Lower limit for size for the image analysis. Leaf images often contain dirt and dust. To prevent dust from affecting the image analysis, the lower limit of analyzed size is set to 0.1, i.e., objects with lesser than 10% of the mean of all objects are removed. One can set a known area or use lower_limit = 0 to select all objects (not advised).

upper_size — Upper limit for size for the image analysis. Defaults to NULL, i.e., no upper limit used.

topn_lower, topn_upper — Select the top n lesions based on its area. topn_lower selects the n lesions with the smallest area whereas topn_upper selects the n lesions with the largest area.

randomize — Randomize the lines before training the model? Defaults to TRUE.

nsample — The number of sample pixels to be used in training step. Defaults to 3000.

watershed — If TRUE (Default) implements the Watershed Algorithm to segment lesions connected by a fairly few pixels that could be considered as two distinct lesions. If FALSE, lesions that are connected by any pixel are considered unique lesions. For more details see EBImage::watershed().

lesion_size — The size of the lesion. Used to automatically tune tolerance and extension parameters. One of the following. "small" (2-5 mm in diameter, e.g, rust pustules), "medium" (0.5-1.0 cm in diameter, e.g, wheat leaf spot), "large" (1-2 cm in diameter, and "elarge" (2-3 cm in diameter, e.g, target spot of soybean).

`tolerance` The minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbors, which is the highest. Defaults to `NULL`, i.e., starting values are set up according to the argument `lesion_size`.

`extension` Radius of the neighborhood in pixels for the detection of neighboring objects. Defaults to 20. Higher value smooths out small objects.

`show_features` If `TRUE` returnS the lesion features such as number, area, perimeter, and radius. Defaults to `FALSE`.

`show_segmentation`
Shows the object segmentation colored with random permutations. Defaults to `TRUE`.

`plot` Show image after processing? Defaults to `TRUE`.

`show_original` Show the symptoms in the original image?

`show_background`
Show the background? Defaults to `TRUE`. A white background is shown by default when `show_original = FALSE`.

`show_contour` Show a contour line around the lesions? Defaults to `TRUE`.

`contour_col, contour_size`
The color and size for the contour line around objects. Defaults to `contour_col = "white"` and `contour_size = 1`.

`col_leaf` Leaf color after image processing. Defaults to `"green"`

`col_lesions` Symptoms color after image processing. Defaults to `"red"`.

`col_background` Background color after image processing. Defaults to `"NULL"`.

`marker, marker_col, marker_size`
The type, color and size of the object marker. Defaults to `NULL`, which shows nothing. Use `marker = "point"` to show a point in each lesion or `marker = "*"` where `"*"` is any variable name of the `shape` data frame returned by the function.

`save_image` Save the image after processing? The image is saved in the current working directory named as `proc_*` where `*` is the image name given in `img`.

`prefix` The prefix to be included in the processed images. Defaults to `"proc_"`.

`name` The name of the image to save. Use this to overwrite the name of the image in `img`.

`dir_original, dir_processed`
The directory containing the original and processed images. Defaults to `NULL`. In this case, the function will search for the image `img` in the current working directory. After processing, when `save_image = TRUE`, the processed image will be also saved in such a directory. It can be either a full path, e.g., `"C:/Desktop/imgs"`, or a subfolder within the current working directory, e.g., `"/imgs"`.

`verbose` If `TRUE` (default) a summary is shown in the console.

`has_background` A logical indicating if the image has a background to be segmented before processing.

r	The radius of neighborhood pixels. Defaults to 2. A square is drawn indicating the selected pixels.
viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
show	The show option for the mapview viewer, either "rgb" or "index".
index	The index to be shown when show = "rgb".
...	Further parameters passed on to measure_disease().

Details

In measure_disease(), a general linear model (binomial family) fitted to the RGB values is used to segment the lesions from the healthy leaf. If a pallet of background is provided, the function takes care of the details to isolate it before computing the number and area of lesions. By using pattern it is possible to process several images with common pattern names that are stored in the current working directory or in the subdirectory informed in dir_original.

If img_healthy and img_symptoms are not declared, RGB-based phenotyping of foliar disease severity is performed using the index informed in index_lb to first segment leaf from background and index_dh to segment diseased from healthy tissues.

measure_disease_iter() only run in an interactive section. In this function, users will be able to pick up samples of images to iteratively create the needed color palettes. This process calls pick_palette() internally. If has_background is TRUE (default) the color palette for the background is first created. The sample of colors is performed in each left-button mouse click and continues until the user press Esc. Then, a new sampling process is performed to sample the color of healthy tissues and then diseased tissues. The generated palettes are then passed on to measure_disease(). All the arguments of such function can be passed using the ... (three dots).

When show_features = TRUE, the function computes a total of 36 lesion features (23 shape features and 13 texture features). The Haralick texture features for each object based on a gray-level co-occurrence matrix (Haralick et al. 1979). See more details in analyze_objects().

Value

- measure_disease() returns a list with the following objects:
  - severity A data frame with the percentage of healthy and symptomatic areas.
  - shape,statistics If show_features = TRUE is used, returns the shape (area, perimeter, etc.) for each lesion and a summary statistic of the results.
- measure_disease_iter() returns a list with the following objects:
  - results A list with the objects returned by measure_disease().
  - leaf The color palettes for the healthy leaf.
  - disease The color palettes for the diseased leaf.
  - background The color palettes for the background.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
library(pliman)
img <- image_pliman("sev_leaf_nb.jpg")
healthy <- image_pliman("sev_healthy.jpg")
lesions <- image_pliman("sev_sympt.jpg")
image_combine(img, healthy, lesions, ncol = 3)

sev <-
 measure_disease(img = img,
                 img_healthy = healthy,
                 img_symptoms = lesions,
                 lesion_size = "large",
                 plot = TRUE)

# an interactive section
measure_disease_iter(img)
```

---

measure_disease_byl *Performs plant disease measurements by leaf*

Description

Computes the percentage of symptomatic leaf area using color palettes or RGB indexes by each leaf of an image. This allows, for example, processing replicates of the same treatment and obtaining the results for each replication with a single image. To do that, leaf samples are first splitten with `object_split()` and then, `measure_disease()` is applied to the list of leaves.

Usage

```
measure_disease_byl(
  img,
  index = "B",
  index_lb = "B",
  index_dh = "NGRDI",
  lower_size = NULL,
  watershed = TRUE,
  invert = FALSE,
  fill_hull = FALSE,
  filter = 3,
  threshold = "Otsu",
  extension = NULL,
```

```
  tolerance = NULL,
  object_size = "large",
  img_healthy = NULL,
  img_symptoms = NULL,
  plot = TRUE,
  save_image = FALSE,
  dir_original = NULL,
  dir_processed = NULL,
  pattern = NULL,
  parallel = FALSE,
  workers = NULL,
  show_features = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	The image to be analyzed.
`index`	A character value specifying the target mode for conversion to binary to segment the leaves from background. Defaults to "B" (blue). See `image_index()` for more details. Personalized indexes can be informed as, e.g., `index = "R*G/B`.
`index_lb`	The index used to segment the foreground (e.g., leaf) from the background. If not declared, the entire image area (pixels) will be considered in the computation of the severity.
`index_dh`	The index used to segment diseased from healthy tissues when `img_healthy` and `img_symptoms` are not declared. Defaults to `"GLI"`. See `image_index()` for more details.
`lower_size`	To prevent dust from affecting object segmentation, objects with lesser than `10%` of the mean of all objects are removed. . One can set a known area or use `lower_limit = 0` to select all objects (not advised).
`watershed`	If `TRUE` (default) performs watershed-based object detection. This will detect objects even when they are touching one other. If `FALSE`, all pixels for each connected set of foreground pixels are set to a unique object. This is faster but is not able to segment touching objects.
`invert`	Inverts the binary image if desired. This is useful to process images with a black background. Defaults to `FALSE`. If `reference = TRUE` is use, `invert` can be declared as a logical vector of length 2 (eg., `invert = c(FALSE, TRUE)`. In this case, the segmentation of objects and reference from the foreground using `back_fore_index` is performed using the default (not inverted), and the segmentation of objects from the reference is performed by inverting the selection (selecting pixels higher than the threshold).
`fill_hull`	Fill holes in the binary image? Defaults to `FALSE`. This is useful to fill holes in objects that have portions with a color similar to the background. IMPORTANT: Objects touching each other can be combined into one single object, which may underestimate the number of objects in an image.

`filter` Performs median filtering in the binary image? See more at `image_filter()`. Defaults to `FALSE`. Use a positive integer to define the size of the median filtering. Larger values are effective at removing noise, but adversely affect edges.

`threshold` The theshold method to be used.

- By default (`threshold = "Otsu"`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold.
- If `threshold = "adaptive"`, adaptive thresholding (Shafait et al. 2008) is used, and will depend on the `k` and `windowsize` arguments.
- If any non-numeric value different than `"Otsu"` and `"adaptive"` is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index.

`extension` Radius of the neighborhood in pixels for the detection of neighboring objects. Higher value smooths out small objects.

`tolerance` The minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbors, which is the highest.

`object_size` The size of the object. Used to automatically set up `tolerance` and `extension` parameters. One of the following. `"small"` (e.g, wheat grains), `"medium"` (e.g, soybean grains), `"large"`(e.g, peanut grains), and `"elarge"` (e.g, soybean pods)‘.

`img_healthy` A color palette of healthy tissues.

`img_symptoms` A color palette of lesioned tissues.

`plot` Show image after processing?

`save_image` Save the image after processing? The image is saved in the current working directory named as `proc_*` where `*` is the image name given in `img`.

`dir_original, dir_processed`
The directory containing the original and processed images. Defaults to `NULL`. In this case, the function will search for the image `img` in the current working directory. After processing, when `save_image = TRUE`, the processed image will be also saved in such a directory. It can be either a full path, e.g., `"C:/Desktop/imgs"`, or a subfolder within the current working directory, e.g., `"/imgs"`.

`pattern` A pattern of file name used to identify images to be processed. For example, if `pattern = "im"` all images that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be analyzed. Providing any number as pattern (e.g., `pattern = "1"`) will select images that are named as 1.-, 2.-, and so on.

`parallel` Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time, especially when `pattern` is used is informed. The number of sections is set up to 30% of available cores.

`workers` A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.

show_features	If TRUE returnS the lesion features such as number, area, perimeter, and radius. Defaults to FALSE.
verbose	If TRUE (default) a summary is shown in the console.
...	Additional arguments passed on to measure_disease().

Value

- A list with the following objects:
  - severity A data frame with the percentage of healthy and symptomatic areas for each leaf in the image(s).
  - shape,statistics If show_features = TRUE is used, returns the shape (area, perimeter, etc.) for each lesion and a summary statistic of the results.

Examples

```
library(pliman)
img <- image_pliman("mult_leaves.jpg", plot = TRUE)
sev <-
 measure_disease_byl(img = img,
                     index_lb = "B",
                     index_dh = "NGRDI",
                     workers = 2)
sev$severity
```

---

measure_disease_shp *Measure disease using shapefiles*

Description

This function calls measure_disease() in each image polygon of a shapefile object generated with image_shp() and bind the results into read-ready data frames.

Usage

```
measure_disease_shp(
  img,
  nrow = 1,
  ncol = 1,
  buffer_x = 0,
  buffer_y = 0,
  prepare = FALSE,
  viewer = "mapview",
  index_lb = "HUE2",
  index_dh = "NGRDI",
  pattern = NULL,
```

```
  threshold = NULL,
  invert = FALSE,
  dir_original = NULL,
  show_features = FALSE,
  interactive = FALSE,
  plot = TRUE,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

img	The image to be analyzed. Either an image of class Image or a character string containing the image name. In the last, the image will be searched in the root directory. Declare dir_original to inform a subfolder that contains the images to be processed.
nrow, ncol	The number of rows and columns to generate the shapefile. Defaults to 1.
buffer_x, buffer_y	Buffering factor for the width and height, respectively, of each individual shape's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the shape by 25% on each side.
prepare	Logical value indicating whether to prepare the image for analysis using image_prepare() function. This allows to align and crop the image before processing. Defaults to FALSE.
viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
index_lb	The index used to segment the foreground (e.g., leaf) from the background. If not declared, the entire image area (pixels) will be considered in the computation of the severity.
index_dh	The index used to segment diseased from healthy tissues when img_healthy and img_symptoms are not declared. Defaults to "GLI". See image_index() for more details.
pattern	A pattern of file name used to identify images to be processed. For example, if pattern = "im" all images that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be analyzed. Providing any number as pattern (e.g., pattern = "1") will select images that are named as 1.-, 2.-, and so on.
threshold	By default (threshold = NULL), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is

	informed, this value will be used as a threshold. Inform any non-numeric value different than "Otsu" to iteratively choose the threshold based on a raster plot showing pixel intensity of the index. Must be a vector of length 2 to indicate the threshold for `index_lb` and `index_dh`, respectively.
`invert`	Inverts the binary image if desired. This is useful to process images with black background. Defaults to `FALSE`.
`dir_original`	The directory containing the original and processed images. Defaults to `NULL`. In this case, the function will search for the image `img` in the current working directory.
`show_features`	If `TRUE` returnS the lesion features such as number, area, perimeter, and radius. Defaults to `FALSE`.
`interactive`	If `FALSE` (default) the grid is created automatically based on the image dimension and number of rows/columns. If `interactive = TRUE`, users must draw points at the diagonal of the desired bounding box that will contain the grid.
`plot`	Show image after processing? Defaults to `TRUE`.
`parallel`	Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time, especially when `pattern` is used is informed. The number of sections is set up to 30% of available cores.
`workers`	A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.
`verbose`	If `TRUE` (default) a summary is shown in the console.
`...`	Aditional arguments passed on to measure_disease.

Value

An object of class `plm_disease_byl`. See more details in the `Value` section of `measure_disease()`.

Examples

```
if(interactive()){
# severity for the three leaflets (from left to right)
img <- image_pliman("mult_leaves.jpg", plot = TRUE)
sev <-
 measure_disease_shp(img = img,
                     nrow = 1,
                     ncol = 3,
                     index_lb = "B",
                     index_dh = "NGRDI")
sev$severity
}
```

mosaic_crop *Crop a mosaic*

Description

Crop a SpatRaster object based on user-defined selection using an interactive map or plot.

Usage

```
mosaic_crop(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  show = c("rgb", "index"),
  index = "R",
  max_pixels = 5e+05,
  downsample = NULL,
  ...
)
```

Arguments

mosaic	A mosaic of class SpatRaster, generally imported with mosaic_input().
r	The layer for the Red band (default: 3).
g	The layer for the Green band (default: 2).
b	The layer for the Blue band (default: 1).
re	The layer for the Red-edge band (default: 4).
nir	The layer for the Near-infrared band(default: 5).
show	The display option for the map view. Options are "rgb" for RGB view and "index" for index view.
index	The index to use for the index view. Defaults to "B".
max_pixels	Maximum number of pixels to render in the map or plot (default: 500000).
downsample	Downsampling factor to reduce the number of pixels (default: NULL). In this case, if the number of pixels in the image (width x height) is greater than max_pixels a downsampling factor will be automatically chosen so that the number of plotted pixels approximates the max_pixels.
...	Additional arguments passed to mosaic_view().

Details

This function uses the mosaic_view function to display an interactive map or plot of the mosaic raster, allowing users to draw a rectangle to select the cropping area. The selected area is then cropped from the input mosaic and returned as a new SpatRaster object.

Value

A cropped version of `mosaic` based on the user-defined selection.

Examples

```
if(interactive()){
library(pliman)
# Load a raster showing the elevation of Luxembourg
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))

# Generate an interactive map using 'mapview' (works only in an interactive section)
cropped <- mosaic_crop(mosaic)
mosaic_view(cropped)
}
```

---

mosaic_index *Mosaic Index*

Description

Compute or extract an index layer from a multi-band mosaic raster.

Usage

```
mosaic_index(mosaic, index = "R", r = 3, g = 2, b = 1, re = 4, nir = 5)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`index`	The index to use for the index view. Defaults to "B".
`r`	The layer for the Red band (default: 3).
`g`	The layer for the Green band (default: 2).
`b`	The layer for the Blue band (default: 1).
`re`	The layer for the Red-edge band (default: 4).
`nir`	The layer for the Near-infrared band(default: 5).

Details

This function computes or extracts an index layer from the input mosaic raster based on the specified index name. If the index is not found in the package's predefined index list (see `image_index()` for more details), it attempts to compute the index using the specified band indices. The resulting index layer is returned as an `SpatRaster` object.

Value

An index layer extracted/computed from the mosaic raster.

Examples

```
library(pliman)
library(terra)
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
confusion <-
     matrix(rnorm(90*95, 100, 30),
            nrow = nrow(mosaic),
            ncol = ncol(mosaic))
confusion <- mosaic_input(confusion)
terra::ext(confusion) <- terra::ext(mosaic)
terra::crs(confusion) <- terra::crs(mosaic)
names(confusion) <- "confusion"
two_layers <- c(mosaic, confusion)
final <- mosaic_index(two_layers, "B+R", b = 1, r = 2)
mosaic_view(mosaic_input(final), viewer = "base")
```

---

mosaic_input *Create and Export mosaics*

Description

Create and Export mosaics

Usage

```
mosaic_input(mosaic, ...)

mosaic_export(mosaic, filename, overwrite = FALSE, ...)
```

Arguments

mosaic	• For mosaic_input(), a file path to the raster to imported, a matrix, array or a list of SpatRaster objects. • For mosaic_export(), an SpatRaster object.
...	Additional arguments passed to terra::rast() (mosaic_input()) or terra::writeRaster() (mosaic_output())
filename	character. The Output filename.
overwrite	logical. If TRUE, filename is overwritten.

Details

- mosaic_input() is a simply wrapper around terra::rast(). It creates a SpatRaster object from scratch, from a filename, or from another object.
- mosaic_export() is a simply wrapper around terra::writeRaster(). It write a SpatRaster object to a file.

Value

- `mosaic_input()` returns an `SpatRaster` object.
- `mosaic_export()` do not return an object.

Examples

```
library(pliman)

# create an SpatRaster object based on a matrix
x <- matrix(1:20, nrow = 4, ncol = 5)
rast <- mosaic_input(x)
mosaic_view(rast, viewer = "base", axes = TRUE)

# create a temporary filename for the example
f <- file.path(tempdir(), "test.tif")
mosaic_export(rast, f, overwrite=TRUE)
list.files(tempdir())
```

`mosaic_prepare` *Prepare a mosaic*

Description

Prepare an `SpatRaster` object to be analyzed in pliman. This includes cropping the original mosaic, aligning it, and cropping the aligned object. The resulting object is an object of class `Image` that can be further analyzed.

Usage

```
mosaic_prepare(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  crop_mosaic = FALSE,
  align = TRUE,
  crop_aligned = TRUE,
  rescale = TRUE,
  coef = 0,
  viewer = "mapview",
  max_pixels = 5e+05,
  show = "rgb",
  index = "R"
)
```

Arguments

mosaic	A mosaic of class SpatRaster, generally imported with mosaic_input().
r	The layer for the Red band (default: 3).
g	The layer for the Green band (default: 2).
b	The layer for the Blue band (default: 1).
re	The layer for the Red-edge band (default: 4).
nir	The layer for the Near-infrared band(default: 5).
crop_mosaic	Logical, whether to crop the mosaic interactively before aligning it (default: FALSE).
align	Logical, whether to align the mosaic interactively (default: TRUE).
crop_aligned	Logical, whether to crop the aligned mosaic interactively (default: TRUE).
rescale	Rescale the final values? If TRUE the final values are rescaled so that the maximum value is 1.
coef	An addition coefficient applied to the resulting object. This is useful to adjust the brightness of the final image. Defaults to 0.
viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
max_pixels	Maximum number of pixels to render in the map or plot (default: 500000).
show	The display option for the map view. Options are "rgb" for RGB view and "index" for index view.
index	The index to use for the index view. Defaults to "B".

Value

A prepared object of class Image.

Examples

```
if(interactive()){
library(pliman)
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
mosaic_prepare(mosaic)
}
```

mosaic_to_pliman *Mosaic to pliman*

Description

Convert an SpatRaster object to a Image object with optional scaling.

Usage

```
mosaic_to_pliman(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  rescale = TRUE,
  coef = 0
)
```

Arguments

mosaic	A mosaic of class SpatRaster, generally imported with mosaic_input().
r	The layer for the Red band (default: 3).
g	The layer for the Green band (default: 2).
b	The layer for the Blue band (default: 1).
re	The layer for the Red-edge band (default: 4).
nir	The layer for the Near-infrared band(default: 5).
rescale	Rescale the final values? If TRUE the final values are rescaled so that the maximum value is 1.
coef	An addition coefficient applied to the resulting object. This is useful to adjust the brightness of the final image. Defaults to 0.

Details

This function converts SpatRaster into an Image object, which can be used for image analysis in pliman. Note that if a large SpatRaster is loaded, the resulting object may increase considerably the memory usage.

Value

An Image object with the same number of layers as mosaic.

Examples

```
library(pliman)
# Convert a mosaic raster to an Image object
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
pliman_image <- mosaic_to_pliman(mosaic)
plot(pliman_image)
```

`mosaic_to_rgb` *Mosaic to RGB*

Description

Convert an `SpatRaster` to a three-band RGB image of class `Image`.

Usage

```
mosaic_to_rgb(mosaic, r = 3, g = 2, b = 1, coef = 0, plot = TRUE)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`r`	The layer for the Red band (default: 3).
`g`	The layer for the Green band (default: 2).
`b`	The layer for the Blue band (default: 1).
`coef`	An addition coefficient applied to the resulting object. This is useful to adjust the brightness of the final image. Defaults to 0.
`plot`	Logical, whether to display the resulting RGB image (default: TRUE).

Details

This function converts `SpatRaster` that contains the RGB bands into a three-band RGB image using pliman (EBImage). It allows you to specify the band indices for the red, green, and blue channels, as well as apply a scaling coefficient to the final image. By default, the resulting RGB image is displayed, but this behavior can be controlled using the `plot` parameter.

Value

A three-band RGB image represented as a pliman (EBImage) object.

Examples

```
library(pliman)
# Convert a mosaic raster to an RGB image and display it
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))

# Convert a mosaic raster to an RGB image without displaying it
rgb_image <- mosaic_to_rgb(c(mosaic * 2, mosaic - 0.3, mosaic * 0.8))
plot(rgb_image)
```

`mosaic_view` *Mosaic View*

Description

Mosaic View

Usage

```
mosaic_view(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  title = "",
  viewer = c("mapview", "base"),
  show = c("rgb", "index"),
  index = "B",
  max_pixels = 5e+05,
  downsample = NULL,
  alpha = 1,
  quantiles = c(0, 1),
  domain = NULL,
  color_regions = custom_palette(),
  axes = FALSE,
  ...
)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`r`	The layer for the Red band (default: 3).
`g`	The layer for the Green band (default: 2).
`b`	The layer for the Blue band (default: 1).

re	The layer for the Red-edge band (default: 4).
nir	The layer for the Near-infrared band(default: 5).
title	A title for the generated map or plot (default: "").
viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
show	The display option for the map view. Options are "rgb" for RGB view and "index" for index view.
index	The index to use for the index view. Defaults to "B".
max_pixels	Maximum number of pixels to render in the map or plot (default: 500000).
downsample	Downsampling factor to reduce the number of pixels (default: NULL). In this case, if the number of pixels in the image (width x height) is greater than max_pixels a downsampling factor will be automatically chosen so that the number of plotted pixels approximates the max_pixels.
alpha	opacity of the fill color of the raster layer(s).
quantiles	the upper and lower quantiles used for color stretching. If set to NULL, stretching is performed basing on 'domain' argument.
domain	the upper and lower values used for color stretching. This is used only if 'quantiles' is NULL. If both 'domain' and 'quantiles' are set to NULL, stretching is applied based on min-max values.
color_regions	The color palette for displaying index values. Default is custom_palette().
axes	logical. Draw axes? Defaults to FALSE.
...	Additional arguments passed on to terra::plot() when viewer = "base".

Details

The function can generate either an interactive map using the 'mapview' package or a static plot using the 'base' package, depending on the viewer and show parameters. If show = "index" is used, the function first computes an image index that can be either an RGB-based index or a multispectral index, if a multispectral mosaic is provided.

Value

An sf object, the same object returned by mapedit::editMap().

Examples

```
if(interactive()){
library(pliman)
# Load a raster showing the elevation of Luxembourg
```

```
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))

# Generate an interactive map using 'mapview'
mosaic_view(mosaic)

# Generate a static plot using 'base'
mosaic_view(mosaic, viewer = "base")
}
```

`object_edge` *Object edges*

Description

Applies the Sobel-Feldman Operator to detect edges. The operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions.

Usage

```
object_edge(img, sigma = 1, threshold = "Otsu", thinning = FALSE, plot = TRUE)
```

Arguments

`img`	An image or a list of images of class `Image`.
`sigma`	Gaussian kernel standard deviation used in the gaussian blur.
`threshold`	The theshold method to be used. If `threshold = "Otsu"` (default), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If any non-numeric value different than `"Otsu"` is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index. Alternatively, provide a numeric value to be used as the threshold value.
`thinning`	Logical value indicating whether a thinning procedure should be applied to the detected edges. See `image_skeleton()`
`plot`	Logical value indicating whether a plot should be created

Value

A binary version of `image`.

References

Sobel, I., and G. Feldman. 1973. A 3×3 isotropic gradient operator for image processing. Pattern Classification and Scene Analysis: 271–272.

Examples

```
library(pliman)
img <- image_pliman("sev_leaf_nb.jpg", plot = TRUE)
object_edge(img)
```

`object_export` *Export multiple objects from an image to multiple images*

Description

Givin an image with multiple objects, `object_export()` will split the objects into a list of objects using `object_split()` and then export them to multiple images into the current working directory (or a subfolder). Batch processing is performed by declaring a file name pattern that matches the images within the working directory.

Usage

```
object_export(
  img,
  pattern = NULL,
  dir_original = NULL,
  dir_processed = NULL,
  format = ".jpg",
  squarize = FALSE,
  augment = FALSE,
  times = 12,
  index = "NB",
  lower_size = NULL,
  watershed = FALSE,
  invert = FALSE,
  fill_hull = FALSE,
  filter = 2,
  threshold = "Otsu",
  extension = NULL,
  tolerance = NULL,
  object_size = "medium",
  edge = 20,
  remove_bg = FALSE,
  parallel = FALSE,
  verbose = TRUE
)
```

Arguments

`img`	The image to be analyzed.

pattern	A pattern of file name used to identify images to be processed. For example, if pattern = "im" all images in the current working directory that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be imported and processed. Providing any number as pattern (e.g., pattern = "1") will select images that are named as 1.-, 2.-, and so on. An error will be returned if the pattern matches any file that is not supported (e.g., img1.pdf).
dir_original	The directory containing the original images. Defaults to NULL. It can be either a full path, e.g., "C:/Desktop/imgs", or a subfolder within the current working directory, e.g., "/imgs".
dir_processed	Optional character string indicating a subfolder within the current working directory to save the image(s). If the folder doesn't exist, it will be created.
format	The format of image to be exported.
squarize	Squarizes the image before the exportation? If TRUE, image_square() will be called internally.
augment	A logical indicating if exported objects should be augmented using image_augment(). Defaults to FALSE.
times	The number of times to rotate the image.
index	A character value specifying the target mode for conversion to binary image when foreground and background are not declared. Defaults to "NB" (normalized blue). See image_index() for more details. User can also calculate your own index using the bands names, e.g. index = "R+B/G"
lower_size	Plant images often contain dirt and dust. To prevent dust from affecting the image analysis, objects with lesser than 10% of the mean of all objects are removed. Set lower_limit = 0 to keep all the objects.
watershed	If TRUE (default) performs watershed-based object detection. This will detect objects even when they are touching one other. If FALSE, all pixels for each connected set of foreground pixels are set to a unique object. This is faster but is not able to segment touching objects.
invert	Inverts the binary image if desired. This is useful to process images with a black background. Defaults to FALSE. If reference = TRUE is use, invert can be declared as a logical vector of length 2 (eg., invert = c(FALSE, TRUE). In this case, the segmentation of objects and reference from the foreground using back_fore_index is performed using the default (not inverted), and the segmentation of objects from the reference is performed by inverting the selection (selecting pixels higher than the threshold).
fill_hull	Fill holes in the binary image? Defaults to FALSE. This is useful to fill holes in objects that have portions with a color similar to the background. IMPORTANT: Objects touching each other can be combined into one single object, which may underestimate the number of objects in an image.
filter	Performs median filtering in the binary image? See more at image_filter(). Defaults to FALSE. Use a positive integer to define the size of the median filtering. Larger values are effective at removing noise, but adversely affect edges.
threshold	The theshold method to be used.

- By default (`threshold = "Otsu"`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold.
- If `threshold = "adaptive"`, adaptive thresholding (Shafait et al. 2008) is used, and will depend on the `k` and `windowsize` arguments.
- If any non-numeric value different than `"Otsu"` and `"adaptive"` is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index.

`extension` Radius of the neighborhood in pixels for the detection of neighboring objects. Higher value smooths out small objects.

`tolerance` The minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbors, which is the highest.

`object_size` The size of the object. Used to automatically set up `tolerance` and `extension` parameters. One of the following. `"small"` (e.g, wheat grains), `"medium"` (e.g, soybean grains), `"large"`(e.g, peanut grains), and `"elarge"` (e.g, soybean pods)‘.

`edge` The number of pixels to be added in the edge of the segmented object. Defaults to 5.

`remove_bg` If `TRUE`, the pixels that are not part of objects are converted to white.

`parallel` If `TRUE` processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time, especially when `pattern` is used is informed. When `object_index` is informed, multiple sections will be used to extract the RGB values for each object in the image. This may significantly speed up processing time when an image has lots of objects (say >1000).

`verbose` If `TRUE` (default) a summary is shown in the console.

Value

A `NULL` object.

Examples

```
if(interactive()){
library(pliman)
img <- image_pliman("potato_leaves.jpg")
object_export(img,
              remove_bg = TRUE)
}
```

`object_export_shp` *Export multiple objects from an image to multiple images*

Description

Givin an image with multiple objects, `object_export_shp()` will split the objects into a list of objects using `object_split_shp()` and then export them to multiple images into the current working directory (or a subfolder). Batch processing is performed by declaring a file name pattern that matches the images within the working directory.

Usage

```
object_export_shp(
  img,
  pattern = NULL,
  dir_original = NULL,
  dir_processed = NULL,
  format = ".jpg",
  subfolder = NULL,
  squarize = FALSE,
  nrow = 1,
  ncol = 1,
  buffer_x = 0,
  buffer_y = 0,
  interactive = FALSE,
  parallel = FALSE,
  verbose = TRUE,
  viewer = get_pliman_viewer()
)
```

Arguments

`img`	An object of class `Image`
`pattern`	A pattern of file name used to identify images to be processed. For example, if `pattern = "im"` all images in the current working directory that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be imported and processed. Providing any number as pattern (e.g., `pattern = "1"`) will select images that are named as 1.-, 2.-, and so on. An error will be returned if the pattern matches any file that is not supported (e.g., img1.pdf).
`dir_original`	The directory containing the original images. Defaults to `NULL`. It can be either a full path, e.g., `"C:/Desktop/imgs"`, or a subfolder within the current working directory, e.g., `"/imgs"`.
`dir_processed`	Optional character string indicating a subfolder within the current working directory to save the image(s). If the folder doesn't exist, it will be created.
`format`	The format of image to be exported.

`subfolder` Optional character string indicating a subfolder within the current working directory to save the image(s). If the folder doesn't exist, it will be created.

`squarize` Squarizes the image before the exportation? If `TRUE`, `image_square()` will be called internally.

`nrow` The number of desired rows in the grid. Defaults to `1`.

`ncol` The number of desired columns in the grid. Defaults to `1`.

`buffer_x, buffer_y` Buffering factor for the width and height, respectively, of each individual shape's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the shape by 25% on each side.

`interactive` If `FALSE` (default) the grid is created automatically based on the image dimension and number of rows/columns. If `interactive = TRUE`, users must draw points at the diagonal of the desired bounding box that will contain the grid.

`parallel` If `TRUE` processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time, especially when `pattern` is used is informed. When `object_index` is informed, multiple sections will be used to extract the RGB values for each object in the image. This may significantly speed up processing time when an image has lots of objects (say >1000).

`verbose` If `TRUE` (default) a summary is shown in the console.

`viewer` The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are `"base"` and `"mapview"`. If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.

Value

A `NULL` object.

Examples

```
if(interactive()){
library(pliman)
flax <- image_pliman("flax_leaves.jpg", plot = TRUE)
object_export_shp(flax)

}
```

`object_label` *Labels objects*

Description

All pixels for each connected set of foreground (non-zero) pixels in x are set to an unique increasing integer, starting from 1. Hence, max(x) gives the number of connected objects in x. This is a wrapper to EBImage::bwlabel or EBImage::watershed (if `watershed = TRUE`).

Usage

```
object_label(
  img,
  index = "B",
  invert = FALSE,
  fill_hull = FALSE,
  threshold = "Otsu",
  k = 0.1,
  windowsize = NULL,
  filter = FALSE,
  watershed = FALSE,
  tolerance = NULL,
  extension = NULL,
  object_size = "medium",
  plot = TRUE,
  ncol = NULL,
  nrow = NULL,
  verbose = TRUE
)
```

Arguments

`img`	An image object.
`index`	A character value (or a vector of characters) specifying the target mode for conversion to binary image. See the available indexes with `pliman_indexes()` and `image_index()` for more details.
`invert`	Inverts the binary image, if desired.
`fill_hull`	Fill holes in the objects? Defaults to `FALSE`.
`threshold`	The theshold method to be used. • By default (`threshold = "Otsu"`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold. • If `threshold = "adaptive"`, adaptive thresholding (Shafait et al. 2008) is used, and will depend on the `k` and `windowsize` arguments.

- If any non-numeric value different than "Otsu" and "adaptive" is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index.

k — a numeric in the range 0-1. when k is high, local threshold values tend to be lower. when k is low, local threshold value tend to be higher.

windowsize — windowsize controls the number of local neighborhood in adaptive thresholding. By default it is set to 1/3 * minxy, where minxy is the minimum dimension of the image (in pixels).

filter — Performs median filtering in the binary image? (Defaults to FALSE). Provide a positive integer > 1 to indicate the size of the median filtering. Higher values are more efficient to remove noise in the background but can dramatically impact the perimeter of objects, mainly for irregular perimeters such as leaves with serrated edges.

watershed — If TRUE (default) performs watershed-based object detection. This will detect objects even when they are touching one other. If FALSE, all pixels for each connected set of foreground pixels are set to a unique object. This is faster but is not able to segment touching objects.

tolerance — The minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbors, which is the highest.

extension — Radius of the neighborhood in pixels for the detection of neighboring objects. Higher value smooths out small objects.

object_size — The size of the object. Used to automatically set up tolerance and extension parameters. One of the following. "small" (e.g, wheat grains), "medium" (e.g, soybean grains), "large"(e.g, peanut grains), and "elarge" (e.g, soybean pods)‘.

plot — Show image after processing?

nrow, ncol — The number of rows or columns in the plot grid. Defaults to NULL, i.e., a square grid is produced.

verbose — If TRUE (default) a summary is shown in the console.

Value

A list with the same length of img containing the labeled objects.

Examples

```
img <- image_pliman("soybean_touch.jpg")
# segment the objects using the "B" (blue) band.
object_label(img, index = "B")
object_label(img, index = "B", watershed = TRUE)
```

`object_map` *Map Object Distances*

Description

Computes distances between objects in an `anal_obj` object and returns a list of distances, coefficient of variation (CV), and means.

Usage

```
object_map(object, by_column = "img", direction = c("horizontal", "vertical"))
```

Arguments

`object`	An `anal_obj` object computed with `analyze_objects_shp()`.
`by_column`	The column name in the object's results data frame to group objects by. Default is "img".
`direction`	The direction of mapping. Should be one of "horizontal" or "vertical". Default is "horizontal".

Value

A list with the following components:

`distances`	A list of distances between objects grouped by unique values in the specified column/row.
`cvs`	A vector of coefficient of variation (CV) values for each column/row.
`means`	A vector of mean distances for each column/row.

See Also

`analyze_objects_shp`

Examples

```
if(interactive()){
library(pliman)
flax <- image_pliman("flax_leaves.jpg", plot =TRUE)
res <-
   analyze_objects_shp(flax,
                       nrow = 3,
                       ncol = 1,
                       watershed = FALSE,
                       index = "R/(G/B)",
                       plot = FALSE)
plot(res$final_image_mask)
plot(res$shapefiles)
```

```
# distance from each leave within each row
result <- object_map(res)
result$distances
result$cvs
result$means
}
```

---

object_mark *Mark Object Points*

Description

Marks the coordinates of objects in an anal_obj object on a plot.

Usage

```
object_mark(object, col = "white")
```

Arguments

object	An anal_obj object computed with analyze_objects_shp() or analyze_objects_shp().
col	The color of the marked points. Default is "white".

See Also

analyze_objects_shp

Examples

```
library(pliman)
flax <- image_pliman("flax_leaves.jpg", plot =TRUE)
res <-
   analyze_objects(flax,
                       watershed = FALSE,
                       index = "R/(G/B)",
                       plot = FALSE)
object_mark(res)
```

`object_rgb` *Extract red, green and blue values from objects*

Description

Given an image and a matrix of labels that identify each object, the function extracts the red, green, and blue values from each object.

Usage

```
object_rgb(img, labels)
```

Arguments

`img`	An `Image` object
`labels`	A mask containing the labels for each object. This can be obtained with `EBImage::bwlabel()` or `EBImage::watershed()`

Value

A data.frame with `n` rows (number of pixels for all the objects) and the following columns:

- `id`: the object id;
- `R`: the value for the red band;
- `G`: the value for the blue band;
- `B`: the value for the green band;

Examples

```
library(pliman)
img <- image_pliman("soybean_touch.jpg")
# segment the objects using the "B" (blue) band (default)

labs <- object_label(img, watershed = TRUE)
rgb <- object_rgb(img, labs[[1]])
head(rgb)
```

`object_split` *Splits objects from an image into multiple images*

Description

Using threshold-based segmentation, objects are first isolated from background. Then, a new image is created for each single object. A list of images is returned.

Usage

```
object_split(
  img,
  index = "NB",
  lower_size = NULL,
  watershed = TRUE,
  invert = FALSE,
  fill_hull = FALSE,
  filter = 2,
  threshold = "Otsu",
  extension = NULL,
  tolerance = NULL,
  object_size = "medium",
  edge = 3,
  remove_bg = FALSE,
  plot = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	The image to be analyzed.
`index`	A character value specifying the target mode for conversion to binary image when `foreground` and `background` are not declared. Defaults to `"NB"` (normalized blue). See `image_index()` for more details. User can also calculate your own index using the bands names, e.g. `index = "R+B/G"`
`lower_size`	Plant images often contain dirt and dust. To prevent dust from affecting the image analysis, objects with lesser than 10% of the mean of all objects are removed. Set `lower_limit = 0` to keep all the objects.
`watershed`	If `TRUE` (default) performs watershed-based object detection. This will detect objects even when they are touching one other. If `FALSE`, all pixels for each connected set of foreground pixels are set to a unique object. This is faster but is not able to segment touching objects.
`invert`	Inverts the binary image if desired. This is useful to process images with a black background. Defaults to `FALSE`. If `reference = TRUE` is use, `invert` can be declared as a logical vector of length 2 (eg., `invert = c(FALSE, TRUE)`. In

this case, the segmentation of objects and reference from the foreground using back_fore_index is performed using the default (not inverted), and the segmentation of objects from the reference is performed by inverting the selection (selecting pixels higher than the threshold).

`fill_hull` Fill holes in the binary image? Defaults to FALSE. This is useful to fill holes in objects that have portions with a color similar to the background. IMPORTANT: Objects touching each other can be combined into one single object, which may underestimate the number of objects in an image.

`filter` Performs median filtering in the binary image? See more at image_filter(). Defaults to FALSE. Use a positive integer to define the size of the median filtering. Larger values are effective at removing noise, but adversely affect edges.

`threshold` The theshold method to be used.

- By default (threshold = "Otsu"), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold.
- If threshold = "adaptive", adaptive thresholding (Shafait et al. 2008) is used, and will depend on the k and windowsize arguments.
- If any non-numeric value different than "Otsu" and "adaptive" is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index.

`extension` Radius of the neighborhood in pixels for the detection of neighboring objects. Higher value smooths out small objects.

`tolerance` The minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbors, which is the highest.

`object_size` The size of the object. Used to automatically set up tolerance and extension parameters. One of the following. "small" (e.g, wheat grains), "medium" (e.g, soybean grains), "large"(e.g, peanut grains), and "elarge" (e.g, soybean pods)'.

`edge` The number of pixels to be added in the edge of the segmented object. Defaults to 5.

`remove_bg` If TRUE, the pixels that are not part of objects are converted to white.

`plot` Show image after processing?

`verbose` If TRUE (default) a summary is shown in the console.

`...` Additional arguments passed on to image_combine()

Value

A list of objects of class Image.

See Also

analyze_objects(), image_binary()

Examples

```
library(pliman)
img <- image_pliman("la_leaves.jpg", plot = TRUE)
imgs <- object_split(img) # set to NULL to use 50% of the cores
```

`object_split_shp` *Splits image objects based on a shape file*

Description

Here, `image_shp()` is used to create a shape file based on the desired number of rows and columns. Then, using the object coordinates, a list of `Image` objects is created.

Usage

```
object_split_shp(
  img,
  nrow = 1,
  ncol = 1,
  buffer_x = 0,
  buffer_y = 0,
  interactive = FALSE,
  viewer = get_pliman_viewer(),
  only_shp = FALSE,
  ...
)
```

Arguments

`img`	An object of class `Image`
`nrow`	The number of desired rows in the grid. Defaults to `1`.
`ncol`	The number of desired columns in the grid. Defaults to `1`.
`buffer_x, buffer_y`	Buffering factor for the width and height, respectively, of each individual shape's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the shape by 25% on each side.
`interactive`	If `FALSE` (default) the grid is created automatically based on the image dimension and number of rows/columns. If `interactive = TRUE`, users must draw points at the diagonal of the desired bounding box that will contain the grid.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are `"base"` and `"mapview"`. If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview

	package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
only_shp	If TRUE returns only the shapefiles with the coordinates for each image. If FALSE (default) returns the splitted image according to nrow and ncol arguments.
...	Other arguments passed on to image_shp()

Value

A list of Image objects

Examples

```
if(interactive()){
library(pliman)
flax <- image_pliman("flax_leaves.jpg", plot = TRUE)
objects <- object_split_shp(flax, nrow = 3, ncol = 5)
image_combine(objects$imgs)
}
```

---

object_to_color *Apply color to image objects*

Description

The function applies the color informed in the argument color to segmented objects in the image. The segmentation is performed using image indexes. Use image_index() to identify the better candidate index to segment objects.

Usage

```
object_to_color(img, index = "NB", color = "blue", plot = TRUE, ...)
```

Arguments

img	An image object.
index	A character value (or a vector of characters) specifying the target mode for conversion to binary image. See the available indexes with pliman_indexes() and image_index() for more details.
color	The color to apply in the image objects. Defaults to "blue".
plot	Plots the modified image? Defaults to TRUE.
...	Additional arguments passed on to image_binary().

Value

An object of class Image

Examples

```
library(pliman)
img <- image_pliman("la_leaves.jpg")
img2 <- object_to_color(img, index = "G-R")
image_combine(img, img2)
```

`otsu` *Calculate Otsu's threshold*

Description

Given a numeric vector with the pixel's intensities, returns the threshold value based on Otsu's method, which minimizes the combined intra-class variance

Usage

```
otsu(values)
```

Arguments

`values`	A numeric vector with the pixel values.

Value

A double (threshold value).

References

Otsu, N. 1979. Threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern SMC-9(1): 62–66. doi: doi:10.1109/tsmc.1979.4310076

Examples

```
img <- image_pliman("soybean_touch.jpg")
thresh <- otsu(img@.Data[,,3])
plot(img[,,3] < thresh)
```

palettes *Create image palettes*

Description

image_palette() creates image palettes by applying the k-means algorithm to the RGB values.

Usage

```
image_palette(img, npal = 5, proportional = TRUE, plot = TRUE)
```

Arguments

img	An image object.
npal	The number of color palettes.
proportional	Creates a joint palette with proportional size equal to the number of pixels in the image? Defaults to TRUE.
plot	Plot the generated palette? Defaults to TRUE.

Value

image_palette() returns a list with two elements:

- palette_list A list with npal color palettes of class Image.
- joint An object of class Image with the color palettes
- proportions The proportion of the entire image corresponding to each color in the palette
- rgbs The average RGB value for each palette

Examples

```
library(pliman)
img <- image_pliman("sev_leaf.jpg")
pal <- image_palette(img, npal = 4)

image_combine(pal$palette_list)
```

`pipe` *Forward-pipe operator*

Description

Pipe an object forward into a function or call expression.

Usage

```
lhs %>% rhs
```

Arguments

`lhs`	The result you are piping.
`rhs`	Where you are piping the result to.

Author(s)

Nathan Eastwood <nathan.eastwood@icloud.com> and Antoine Fabri <antoine.fabri@gmail.com>. The code was obtained from poorman package at https://github.com/nathaneastwood/poorman/blob/master/R/pipe.R

Examples

```
library(pliman)

# Basic use:
 iris %>% head()

# use to apply several functions to an image
img <- image_pliman("la_leaves.jpg")

img %>%
 image_resize(50) %>%        # resize to 50% of the original size
 object_isolate(id = 1) %>%  # isolate object 1
 image_filter() %>%          # apply a median filter
 plot()                      # plot
```

`pixel_index` *Get the pixel indices for a given row of a binary image*

Description

This function finds the first row in the bin matrix that has a value greater than 0 (TRUE). It then calculates the minimum, median, and maximum values for the pixels in that row and creates an array containing the row index, the minimum pixel index, the median pixel index, and the maximum pixel index.

Usage

```
pixel_index(bin, row = NULL, direction = "updown")
```

Arguments

`bin`	A logical matrix representing a binary image
`row`	An optional row index. If not provided, the function selects the first non-zero row.
`direction`	The direction for row selection when row is not provided. If set to `"updown"`, the function starts scanning from the top of the image towards the bottom. If set to `"downup"`, the function starts scanning from the bottom towards the top.

Value

A numeric vector containing the row index, the minimum pixel index, the median pixel index, and the maximum pixel index.

Examples

```
library(pliman)
leaf <- image_pliman("sev_leaf.jpg")
bin <- image_binary(leaf, "NB")[[1]]

# first row with leaf (17)
pixel_index(bin)

# index at the row 100
pixel_index(bin, row = 100)

plot(leaf)
points(x = 248, y = 17, pch = 16, col = "red", cex = 2)
points(x = 163, y = 100, pch = 16, col = "red", cex = 2)
points(x = 333, y = 100, pch = 16, col = "red", cex = 2)
```

pliman_images *Sample images*

Description

Sample images installed with the **pliman** package

Format

`*.jpg` format

- `flax_leaves.jpg` Flax leaves in a white background
- `flax_grains.jpg` Flax grains with background light.
- `la_back.jpg` A cyan palette representing the background of images la_pattern, la_leaves, and soybean_touch.
- `la_leaf.jpg` A sample of the leaves in `la_leaves`
- `la_leaves.jpg` Tree leaves with a sample of known area.
- `mult_leaves.jpg` Three soybean leaflets with soybean rust symptoms.
- `objects_300dpi.jpg` An image with 300 dpi resolution.
- `potato_leaves.jpg` Three potato leaves, which were gathered from Gupta et al. (2020).
- `sev_leaf.jpg` A soybean leaf with a blue background.
- `sev_leaf_nb.jpg` A soybean leaf without background.
- `sev_back.jpg` A blue palette representing the background of `sev_leaf`.
- `sev_healthy.jpg` Healthy area of `sev_leaf`.
- `sev_sympt.jpg` The symptomatic area `sev_leaf`.
- `shadow.jpg` A shaded leaf, useful to test adaptive thresholding
- `soy_green.jpg` Soybean grains with a white background.
- `soybean_grain.jpg` A sample palette of the grains in `soy_green`.
- `soybean_touch.jpg` Soybean grains with a cyan background touching one each other.
- `field_mosaic.jpg` An UVA image from a soybean field.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Source

Personal data, Gupta et al. (2020).

References

Gupta, S., Rosenthal, D. M., Stinchcombe, J. R., & Baucom, R. S. (2020). The remarkable morphological diversity of leaf shape in sweet potato (Ipomoea batatas): the influence of genetics, environment, and G×E. New Phytologist, 225(5), 2183–2195. doi:10.1111/NPH.16286

pliman_viewer *Global option for controlling the viewer in pliman package*

Description

Users can set the value of this option using `options("pliman_viewer", value)`. The default value is "base". Use "mapview" to allow image to be plotted/edited using the R packages mapview and mapedit

plot.image_shp *S3 method* `plot` *for* `image_shp` *objects*

Description

Draws the bounding boxes for each object computed with `image_shp()`.

Usage

```
## S3 method for class 'image_shp'
plot(
  x,
  img = NULL,
  col_line = "black",
  size_line = 2,
  col_text = "black",
  size_text = 0.75,
  ...
)
```

Arguments

`x`	An object computed with `image_shp()`.
`img`	The image that was used to compute the shapefile (optional)
`col_line, col_text`	The color of the line/text in the grid. Defaults to `"red"`.
`size_line, size_text`	The size of the line/text in the grid. Defaults to `2.5`.
`...`	Currently not used.

Value

A `NULL` object

Examples

```
library(pliman)
flax <- image_pliman("flax_leaves.jpg")
shape <- image_shp(flax, nrow = 3, ncol = 5)

# grid on the existing image
plot(flax)
plot(shape)
```

`plot_index` *Plot an image index*

Description

Plot an image index

Usage

```
plot_index(
  img = NULL,
  object = NULL,
  index = NULL,
  remove_bg = TRUE,
  viewer = get_pliman_viewer(),
  all_layers = TRUE,
  layer = 1,
  max_pixels = 5e+05,
  downsample = NULL,
  downsample_fun = NULL,
  color_regions = custom_palette(),
  ncol = NULL,
  nrow = NULL,
  aspect_ratio = NA
)
```

Arguments

`img`	An optional `Image` object or an object computed with `image_index()`. If `object` is provided, then the input image is obtained internally.
`object`	An object computed with `analyze_objects_shp()`. By using this object you can ignore `img`.
`index`	The index to plot. Defaults to the index computed from the `object` if provided. Otherwise, the `B` index is computed. See `image_index()` for more details.
`remove_bg`	Logical value indicating whether to remove the background when `object` is provided. Defaults to `TRUE`.

viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
all_layers	Render all layers when img is an object computed with image_index() and viewer = "mapview"?.
layer	The layer to plot when img is an object computed with image_index() and viewer = "mapview". Defaults to the first layer (first index computed).
max_pixels	integer > 0. Maximum number of cells to plot the index. If max_pixels < npixels(img), downsampling is performed before plotting the index. Using a large number of pixels may slow down the plotting time.
downsample	integer; for each dimension the number of pixels/lines/bands etc that will be skipped; Defaults to NULL, which will find the best downsampling factor to approximate the max_pixels value.
downsample_fun	function; if given, downsampling will apply downsample_fun`` to each of the the subtiles.
color_regions	The color palette for displaying index values. Default is custom_palette().
nrow, ncol	The number of rows or columns in the plot grid. Defaults to NULL, i.e., a square grid is produced.
aspect_ratio	Numeric, giving the aspect ratio y/x. Defaults to NA. See graphics::plot.window() for more details.

Value

None

Examples

```
if(interactive()){
# Example usage:
library(pliman)
img <- image_pliman("sev_leaf.jpg")
plot_index(img, index = "B")
}
```

plot_index_shp *Plot rectangles colored by a quantitative attribute and overlay on an RGB image*

Description

This function plots rectangles on top of an RGB image, where each rectangle is colored based on a quantitative variable. The quantitative variable is specified in the `attribute` argument and should be present in the `object_index` of the `object` computed using `analyze_objects_shp()`. The rectangles are colored using a color scale.

Usage

```
plot_index_shp(
  object,
  attribute = "coverage",
  color = c("red", "green"),
  viewer = c("mapview", "base"),
  max_pixels = 5e+05,
  downsample = NULL,
  downsample_fun = NULL,
  alpha = 0.5,
  legend.position = "bottom",
  na.color = "gray",
  classes = 6,
  round = 3,
  horiz = TRUE
)
```

Arguments

`object`	An object computed with `analyze_objects_shp()`.
`attribute`	The name of the quantitative variable in the `object_index` to be used for coloring the rectangles.
`color`	A vector of two colors to be used for the color scale.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are `"base"` and `"mapview"`. If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.
`max_pixels`	integer > 0. Maximum number of cells to plot the index. If `max_pixels < npixels(img)`, downsampling is performed before plotting the index. Using a large number of pixels may slow down the plotting time.

downsample	integer; for each dimension the number of pixels/lines/bands etc that will be skipped; Defaults to NULL, which will find the best downsampling factor to approximate the max_pixels value.
downsample_fun	function; if given, downsampling will apply downsample_fun`` to each of the the subtiles.
alpha	The transparency level of the rectangles' color (between 0 and 1).
legend.position	The position of the color legend, either "bottom" or "right".
na.color	The color to be used for rectangles with missing values in the quantitative variable.
classes	The number of classes in the color scale.
round	The number of decimal places to round the legend values.
horiz	Logical, whether the legend should be horizontal (TRUE) or vertical (FALSE).

Value

The function plots rectangles colored by the specified quantitative variable on top of the RGB image and shows the continuous color legend outside the plot.

Examples

```
if(interactive()){
library(pliman)

# Computes the DGCI index for each flax leaf
flax <- image_pliman("flax_leaves.jpg", plot =TRUE)
res <-
   analyze_objects_shp(flax,
                       buffer_x = 0.2,
                       buffer_y = 0.2,
                       nrow = 3,
                       ncol = 5,
                       plot = FALSE,
                       object_index = "DGCI")
plot(res$final_image)
plot_index_shp(res)
}
```

---

plot_lw *Plot length and width lines on objects*

Description

This function plots the length and width lines given an object computed with analyze_objects(). The function does not call plot.new, so it must be called after an image is plotted. This can be done either using, e.g., plot(img), or analyze_objects(..., plot = TRUE).

Usage

```
plot_lw(
  object,
  col_length = "red",
  col_width = "green",
  lwd_length = 2,
  lwd_width = 2
)
```

Arguments

object	An object computed with analyze_objects().
col_length	The color of the length line. Default is "red".
col_width	The color of the width line. Default is "green".
lwd_length	The line width of the length line. Default is 2.
lwd_width	The line width of the width line. Default is 2.

Details

This function takes an object computed with analyze_objects() and plots the length and width lines of each object onto an image. The length and width lines are calculated based on the position and orientation of the object, and are plotted using the specified colors and line widths.

Examples

```
img <- image_pliman("flax_leaves.jpg")
res <- analyze_objects(img, watershed = FALSE, show_contour = FALSE)
plot_lw(res)
```

---

poly_apex_base_angle *Calculate the apex and base angles of an object*

Description

This function calculates the apex and base angles of an object. It takes as input a matrix of coordinates and returns the apex angle, base angle, and the coordinates of the apex and base as a list. The angles are computed after the object is aligned in the vertical axis with poly_align().

Usage

```
poly_apex_base_angle(
  x,
  percentiles = c(0.25, 0.75),
  invert = FALSE,
  plot = TRUE
)
```

Arguments

x	A matrix of coordinates representing the contour of the object, often obtained with object_contour().
percentiles	A numeric vector of two percentiles between 0 and 1 indicating the height of the points from the top to the bottom. The function calculates the apex angle between the two percentiles and the base angle between the lowest point and the highest point.
invert	If TRUE, aligns the object along the horizontal axis.
plot	Plots the polygon with the points? Defaults to TRUE.

Value

A list containing the apex angle, base angle, apex coordinates, and base coordinates.

Examples

```
library(pliman)
# a matrix of coordinates
angls <- poly_apex_base_angle(contours[[2]])
angls

# or a list of coordinates
poly_apex_base_angle(contours)
```

---

poly_pcv *Compute Perimeter Complexity Value (PCV)*

Description

This function calculates the Perimeter Complexity Value (PCV) for a given set of coordinates representing a contour. The PCV measures the variation of distances between the original coordinates and the smoothed coordinates relative to the perimeter length of the original contour. See more in details section.

Usage

```
poly_pcv(x, niter = 100)
```

Arguments

x	A matrix or a list of matrices representing the coordinates of the polygon(s).
niter	An integer specifying the number of smoothing iterations. See poly_smooth() for more details.

Details

The PCV is computed using the following formula:

$$PCV = \frac{sum(dists) \times sd(dists)}{perim}$$

where $dists$ represents the distances between corresponding points in the original and smoothed coordinates, and $perim$ is the perimeter length of the smoothed contour.

The PCV is computed by first smoothing the input contour using a specified number of iterations. The smoothed contour is then used to compute the distances between corresponding points in the original and smoothed coordinates. These distances reflect the variations in the contour shape after smoothing. The sum of these distances represents the overall magnitude of the variations. Next, the sum of distances is multiplied by the standard deviation of the distances to capture the dispersion or spread of the variations. Finally, this value is divided by the perimeter length of the original contour to provide a relative measure of complexity. Therefore, the PCV provides a relative measure of complexity by considering both the magnitude and spread of the variations in the contour shape after smoothing.

Value

The PCV value(s) computed for the contour(s).

If `x` is a matrix, returns the complexity value of the polygon's perimeter. If `x` is a list of matrices, returns a numeric vector of complexity values for each polygon.

Examples

```
library(pliman)
set.seed(20)
shp <- efourier_shape(npoints = 1000)
poly_pcv(shp)

# increase the complexity of the outline
shp2 <- poly_jitter(shp, noise_x = 20, noise_y = 250, plot = TRUE)

smo <- poly_smooth(shp2, niter = 100, plot = FALSE)
plot_contour(smo, col = "red")
poly_pcv(shp2)
```

---

poly_width_at *Width at a given height*

Description

The function computes the polygonal convex hull of the points in `x` and then returns the number of points that lie below a specified set of heights along the vertical axis of the convex hull.

Usage

```
poly_width_at(
  x,
  at = c(0.05, 0.25, 0.5, 0.75, 0.95),
  unify = FALSE,
  plot = FALSE
)
```

Arguments

x	A vector containing two-dimensional data points (often produced with object_contour).
at	A vector of heights along the vertical axis of the convex hull at which to count the number of points below. The default value is `c(0.05, 0.25, 0.5, 0.75, 0.95)`, which means the function will return the number of points below the 5th, 25th, 50th, 75th, and 95th percentiles of the convex hull. If `at = "heights"` is used, the function returns the width for each point of the object length.
unify	A logical value indicating whether to use the unified convex hull calculation method. If unify = TRUE, coordinates in x will be first bound before computing the convex hull.
plot	A logical value that specifies whether the widths should be plotted.

Details

The convex hull computed from x is aligned along the major axis and then converted to a binary image. For each height in the at vector, the function computes the corresponding row number in the binary image (i.e., the row number that corresponds to the specified height along the vertical axis of the convex hull) and sums the values in that row to obtain the number of points that lie below the specified height. If the convex hull contains multiple polygons and unify = FALSE, the function loops over each polygon and returns a list of the number of points below the specified heights for each polygon. If the convex hull contains only one polygon or multiple polygons and unify = TRUE, the function returns a vector of the number of points below the specified heights for that single polygon.

Value

A vector with the widths of the convex hull at the specified heights or a list of vectors with the widths of each component.

Examples

```
cont <- contours[[2]]
plot_polygon(cont |> conv_hull() |> poly_align())
#  width below 5th, 25th, 50th, 75th, and 95th percentiles of the length
wd <- poly_width_at(cont)
wd

# width along the height
poly_width_at(cont, at = "height", plot = TRUE)
```

prepare_to_shp *Prepare images to analyze_objects_shp()*

Description

It is a simple wrapper around `image_align()` and `image_crop()`. In this case, only the option `viewer = "base"` is used. To use `viewer = "mapview"`, please, use such functions separately.

Usage

```
prepare_to_shp(img, align = "vertical")
```

Arguments

`img`	A `Image` object
`align`	The desired alignment. Either `"vertical"` (default) or `"horizontal"`.

Value

An aligned and cropped `Image` object.

Examples

```
if(interactive()){
img <- image_pliman("flax_leaves.jpg")
prepare_to_shp(img)
}
```

random_color *Random built-in color names*

Description

Randomly chooses single or multiple built-in color names which R knows about. See more at `grDevices::colors()`

Usage

```
random_color(n = 1, distinct = FALSE)
```

Arguments

`n`	The number of color names. Defaults to 1.
`distinct`	Logical indicating if the colors returned should all be distinct. Defaults to `FALSE`.

Value

A character vector of color names

Examples

```
library(pliman)
random_color(n = 3)
```

---

sad *Produces Santandard Area Diagrams*

Description

Given an object computed with `measure_disease()` or `measure_disease_byl()` a Standard Area Diagram (SAD) with `n` images are returned with the respective severity values.

Usage

```
sad(
  object,
  n,
  show_original = FALSE,
  show_contour = FALSE,
  nrow = NULL,
  ncol = NULL,
  ...
)
```

Arguments

`object`	An object computed with `measure_disease()` or `measure_disease_byl()`.
`n`	The number of leaves in the Standard Area Diagram.
`show_original`	Show original images? Defaults to `FALSE`, i.e., a mask is returned.
`show_contour`	Show original images? Defaults to `FALSE`, i.e., a mask is returned.
`nrow, ncol`	The number of rows and columns in the plot. See [image_combine())] [image_combine())]: R:image_combine())
`...`	Other arguments passed on to `measure_disease()`.

Details

The leaves with the smallest and highest severity will always be in the SAD. If `n = 1`, the leaf with the smallest severity will be returned. The others are sampled sequentially to achieve the `n` images after severity has been ordered in an ascending order. For example, if there are 30 leaves and `n` is set to 3, the leaves sampled will be the 1st, 15th, and 30th with the smallest severity values.

The SAD can be only computed if an image pattern name is used in argument `pattern` of `measure_disease()`. If the images are saved, the `n` images will be retrevied from `dir_processed` directory. Otherwise, the severity will be computed again to generate the images.

Value

A data frame with the severity values for the n sampled leaves. A plot with the standard area diagram can be saved by wrapping sad() with png().

References

Del Ponte EM, Pethybridge SJ, Bock CH, et al (2017) Standard area diagrams for aiding severity estimation: Scientometrics, pathosystems, and methodological trends in the last 25 years. Phytopathology 107:1161–1174. doi:10.1094/PHYTO02170069FI

Examples

```
## Not run:
library(pliman)
sev <-
measure_disease(pattern = "sev_leaf",
                img_healthy = "sev_healthy",
                img_symptoms = "sev_sympt",
                img_background  = "sev_back",
                plot = FALSE,
                save_image = TRUE,
                show_original = FALSE,
                dir_original = image_pliman(),
                dir_processed = tempdir())
sad(sev, n = 2)

## End(Not run)
```

---

separate_col *Turns a single character column into multiple columns.*

Description

Given either a regular expression or a vector of character positions, separate_col() turns a single character column into multiple columns.

Usage

```
separate_col(.data, col, into, sep = "[^[:alnum:]]+")
```

Arguments

.data	A data frame
col	Column name
into	Names of new variables to create as character vector
sep	The separator between columns. By default, a regular expression that matches any sequence of non-alphanumeric values.

Value

A mutated `.data`

Examples

```
library(pliman)
df <- data.frame(x = paste0("TRAT_", 1:5),
                 y = 1:5)
df
separate_col(df, x, into = c("TRAT", "REP"))
```

`set_pliman_viewer` *Set the value of the pliman_viewer option*

Description

Sets the value of the pliman_viewer option used in the package.

Usage

```
set_pliman_viewer(value)
```

Arguments

`value`	The value to be set for the pliman_viewer option.

`summary_index` *Summary an object index*

Description

If more than one index is available, the function performs a Principal Component Analysis and produces a plot showing the contribution of the indexes to the PC1 (see `pca()`). If an index is declared in `index` and a cut point in `cut_point`, the number and proportion of objects with mean value of `index` bellow and above `cut_point` are returned. Additionaly, the number and proportion of pixels bellow and above the cutpoint is shown for each object (id).

Usage

```
summary_index(
  object,
  index = NULL,
  cut_point = NULL,
  select_higher = FALSE,
  plot = TRUE,
  type = "var",
  ...
)
```

Arguments

object	An object computed with analyze_objects().
index	The index desired, e.g., "B". Note that these value must match the index(es) used in the argument object_index of analyze_objects().
cut_point	The cut point.
select_higher	If FALSE (default) selects the objects with index smaller than the cut_point. Use select_higher = TRUE to select the objects with index higher than cut_point.
plot	Shows the contribution plot when more than one index is available? Defaults to TRUE.
type	The type of plot to produce. Defaults to "var". See more at get_biplot().
...	Further arguments passed on to get_biplot().

Value

A list with the following elements:

- ids The identification of selected objects.
- between_id A data frame with the following columns
  - n The number of objects.
  - nsel The number of selected objects.
  - prop The proportion of objects selected.
  - mean_index_sel, and mean_index_nsel The mean value of index for the selected and non-selected objects, respectively.
- within_id A data frame with the following columns
  - id The object identification
  - n_less The number of pixels with values lesser than or equal to cut_point.
  - n_greater The number of pixels with values greater than cut_point.
  - less_ratio The proportion of pixels with values lesser than or equal to cut_point.
  - greater_ratio The proportion of pixels with values greater than cut_point.
- pca_res An object computed with pca()

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
library(pliman)
soy <- image_pliman("soy_green.jpg")
anal <- analyze_objects(soy, object_index = "G", pixel_level_index = TRUE)
plot_measures(anal, measure = "G")

summary_index(anal, index = "G", cut_point = 0.5)
```

utils_colorspace *Convert between colour spaces*

Description

rgb_to_srgb() Transforms colors from RGB space (red/green/blue) to Standard Red Green Blue (sRGB), using a gamma correction of 2.2.

- rgb_to_hsb() Transforms colors from RGB space (red/green/blue) to HSB space (hue/saturation/brightness).
- rgb_to_lab() Transforms colors from RGB space (red/green/blue) to CIE-LAB space

It is assumed that

Usage

```
rgb_to_hsb(object)

rgb_to_srgb(object)

rgb_to_lab(object)
```

Arguments

object	An Image object, an object computed with analyze_objects() with a valid object_index argument, or a data.frame/matrix. For the last, a three-column data (R, G, and B, respectively) is required.

Value

A data frame with the columns of the converted color space

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
if(interactive()){
library(pliman)
img <- image_pliman("sev_leaf.jpg")
rgb_to_lab(img)

# analyze the object and convert the pixels
anal <- analyze_objects(img, object_index = "B", pixel_level_index = TRUE)
rgb_to_lab(anal)
}
```

utils_dpi *Utilities for image resolution*

Description

Provides useful conversions between size (cm), number of pixels (px) and dots per inch (dpi).

- `dpi_to_cm()` converts a known dpi value to centimeters.
- `cm_to_dpi()` converts a known centimeter values to dpi.
- `pixels_to_cm()` converts the number of pixels to centimeters, given a known resolution (dpi).
- `cm_to_pixels()` converts a distance (cm) to number of pixels, given a known resolution (dpi).
- `distance()` Computes the distance between two points in an image based on the Pythagorean theorem.
- `dpi()` An interactive function to compute the image resolution given a known distance informed by the user. See more information in the **Details** section.
- `npixels()` returns the number of pixels of an image.

Usage

```
dpi_to_cm(dpi)

cm_to_dpi(cm)

pixels_to_cm(px, dpi)

cm_to_pixels(cm, dpi)

npixels(img)

dpi(img, viewer = get_pliman_viewer())

distance(img, viewer = get_pliman_viewer())
```

Arguments

`dpi`	The image resolution in dots per inch.
`cm`	The size in centimeters.
`px`	The number of pixels.
`img`	An image object.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package,

you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.

Details

dpi() only run in an interactive section. To compute the image resolution (dpi) the user must use the left button mouse to create a line of known distance. This can be done, for example, using a template with known distance in the image (e.g., la_leaves.jpg).

Value

- dpi_to_cm(), cm_to_dpi(), pixels_to_cm(), and cm_to_pixels() return a numeric value or a vector of numeric values if the input data is a vector.
- dpi() returns the computed dpi (dots per inch) given the known distance informed in the plot.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
library(pliman)
# Convert  dots per inch to centimeter
dpi_to_cm(c(1, 2, 3))

# Convert centimeters to dots per inch
cm_to_dpi(c(1, 2, 3))

# Convert centimeters to number of pixels with resolution of 96 dpi.
cm_to_pixels(c(1, 2, 3), 96)

# Convert number of pixels to cm with resolution of 96 dpi.
pixels_to_cm(c(1, 2, 3), 96)

if(isTRUE(interactive())){
#### compute the dpi (dots per inch) resolution ####
# only works in an interactive section
# objects_300dpi.jpg has a known resolution of 300 dpi
img <- image_pliman("objects_300dpi.jpg")
# Higher square: 10 x 10 cm
# 1) Run the function dpi()
# 2) Use the left mouse button to create a line in the higher square
# 3) Declare a known distance (10 cm)
# 4) See the computed dpi
dpi(img)


img2 <- image_pliman("la_leaves.jpg")
# square leaf sample (2 x 2 cm)
dpi(img2)
}
```

utils_file *Utilities for file manipulation*

Description

- `file_extension()` Get the extension of a file.
- `file_name()` Get the name of a file.
- `file_dir()` Get or directory of a file
- `manipulate_files()` Manipulate files in a directory with options to rename (insert prefix or suffix) and save the new files to the same or other provided directory.
- `pliman_indexes()` Get the indexes available in pliman.
- `pliman_indexes_eq()` Get the equation of the indexes available in pliman.

Usage

```
file_extension(file)

file_name(file)

file_dir(file)

manipulate_files(
  pattern,
  dir = NULL,
  prefix = NULL,
  name = NULL,
  suffix = NULL,
  extension = NULL,
  sep = "",
  save_to = NULL,
  overwrite = FALSE,
  remove_original = FALSE,
  verbose = TRUE
)

pliman_indexes()

pliman_indexes_eq()
```

Arguments

`file`	The file name.
`pattern`	A file name pattern.
`dir`	The working directory containing the files to be manipulated. Defaults to the current working directory.

prefix, suffix	A prefix or suffix to be added in the new file names. Defaults to NULL (no prefix or suffix).
name	The name of the new files. Defaults to NULL (original names). name can be either a single value or a character vector of the same length as the number of files manipulated. If one value is informed, a sequential vector of names will be created as "name_1", "name_2", and so on.
extension	The new extension of the file. If not declared (default), the original extensions will be used.
sep	An optional separator. Defaults to "".
save_to	The directory to save the new files. Defaults to the current working directory. If the file name of a file is not changed, nothing will occur. If save_to refers to a subfolder in the current working directory, the files will be saved to the given folder. In case of the folder doesn't exist, it will be created. By default, the files will not be overwritten. Set overwrite = TRUE to overwrite the files.
overwrite	Overwrite the files? Defaults to FALSE.
remove_original	Remove original files after manipulation? defaults to FALSE. If TRUE the files in pattern will be removed.
verbose	If FALSE, the code is run silently.

Value

- file_extension(), file_name(), and file_dir() return a character string.
- manipulate_files() No return value. If verbose == TRUE, a message is printed indicating which operation succeeded (or not) for each of the files attempted.

Examples

```
library(pliman)
# get file name, directory and extension
file <- "E:/my_folder/my_subfolder/image1.png"
file_dir(file)
file_name(file)
file_extension(file)

# manipulate files
dir <- tempdir()
list.files(dir)
file.create(paste0(dir, "/test.txt"))
list.files(dir)
manipulate_files("test",
                 dir = paste0(dir, "\\"),
                prefix = "chang_",
                save_to = paste0(dir, "\\"),
                overwrite = TRUE)
list.files(dir)
```

utils_image *Import and export images*

Description

Import images from files and URLs and write images to files, possibly with batch processing.

Usage

```
image_import(
  img,
  ...,
  which = 1,
  pattern = NULL,
  path = NULL,
  resize = FALSE,
  plot = FALSE,
  nrow = NULL,
  ncol = NULL
)

image_export(img, name, prefix = "", extension = NULL, subfolder = NULL, ...)

image_pliman(img, plot = FALSE)
```

Arguments

img	• For `image_import()`, a character vector of file names or URLs. • For `image_export()`, an Image object, an array or a list of images. • For `image_pliman()`, a charactere value specifying the image example. See `?pliman_images` for more details.
...	Alternative arguments passed to the corresponding functions from the `jpeg`, `png`, and `tiff` packages.
which	logical scalar or integer vector to indicate which image are imported if a TIFF files is informed. Defaults to 1 (the first image is returned).
pattern	A pattern of file name used to identify images to be imported. For example, if `pattern = "im"` all images in the current working directory that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be imported as a list. Providing any number as pattern (e.g., `pattern = "1"`) will select images that are named as 1.-, 2.-, and so on. An error will be returned if the pattern matches any file that is not supported (e.g., img1.pdf).
path	A character vector of full path names; the default corresponds to the working directory, `getwd()`. It will overwrite (if given) the path informed in `image` argument.
resize	Resize the image after importation? Defaults to `FALSE`. Use a numeric value of range 0-100 (proportion of the size of the original image).

plot	Plots the image after importing? Defaults to FALSE.
nrow, ncol	Passed on to image_combine(). The number of rows and columns to use in the composite image when plot = TRUE.
name	An string specifying the name of the image. It can be either a character with the image name (e.g., "img1") or name and extension (e.g., "img1.jpg"). If none file extension is provided, the image will be saved as a *.jpg file.
prefix	A prefix to include in the image name when exporting a list of images. Defaults to "", i.e., no prefix.
extension	When image is a list, extension can be used to define the extension of exported files. This will overwrite the file extensions given in image.
subfolder	Optional character string indicating a subfolder within the current working directory to save the image(s). If the folder doesn't exist, it will be created.

Value

- image_import() returns a new Image object.
- image_export() returns an invisible vector of file names.
- image_pliman() returns a new Image object with the example image required. If an empty call is used, the path to the tmp_images directory installed with the package is returned.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
library(pliman)
folder <- image_pliman()
full_path <- paste0(folder, "/sev_leaf.jpg")
(path <- file_dir(full_path))
(file <- basename(full_path))
image_import(img = full_path)
image_import(img = file, path = path)
```

---

utils_measures *Utilities for object measures*

Description

- get_measures() computes object measures (area, perimeter, radius) by using either a known resolution (dpi) or an object with known measurements.
- plot_measures() draws the object measures given in an object to the current plot. The object identification ("id") is drawn by default.

Usage

```
get_measures(
  object,
  measure = NULL,
  id = NULL,
  dpi = NULL,
  sep = "\\_|-",
  verbose = TRUE,
  digits = 5
)

plot_measures(
  object,
  measure = "id",
  id = NULL,
  hjust = NULL,
  vjust = NULL,
  digits = 2,
  size = 0.9,
  col = "white",
  ...
)
```

Arguments

object	An object computed with analyze_objects().
measure	For plot_measures(), a character string; for get_measures(), a two-sided formula, e.g., measure = area ~ 100 indicating the known value of object id. The right-hand side is the known value and the left-hand side can be one of the following. • area The known area of the object. • perimeter The known perimeter of the object. • radius_mean The known radius of the object. • radius_min The known minimum radius of the object. If the object is a square, then the radius_min of such object will be L/2 where L is the length of the square side. • radius_max The known maximum radius of the object. If the object is a square, then the radius_max of such object according to the Pythagorean theorem will be L x sqrt(2) / 2 where L is the length of the square side.
id	An object in the image to indicate a known value.
dpi	A known resolution of the image in DPI (dots per inch).
sep	Regular expression to manage file names. The function combines in the merge object the object measures (sum of area and mean of all the other measures) of all images that share the same filename prefix, defined as the part of the filename preceding the first hyphen (-) or underscore (_) (no hyphen or underscore is required). For example, the measures of images named L1-1.jpeg, L1-2.jpeg,

	and `L1-3.jpeg` would be combined into a single image information (L1). This feature allows the user to treat multiple images as belonging to a single sample, if desired. Defaults to `sep = "\\_|-"`.
`verbose`	If `FALSE`, runs the code silently.
`digits`	The number of significant figures. Defaults to `2`.
`hjust, vjust`	A numeric value to adjust the labels horizontally and vertically. Positive values will move labels to right (hjust) and top (vjust). Negative values will move the labels to left and bottom, respectively.
`size`	The size of the text. Defaults to `0.9`.
`col`	The color of the text. Defaults to `"white"`.
`...`	Further arguments passed on to `graphics::text()`.

Value

- For `get_measures()`, if `measure` is informed, the pixel values will be corrected by the value of the known object, given in the unit of the right-hand side of `meae`. If `dpi` is informed, then all the measures will be adjusted to the knosurwn `dpi`.
- If applied to an object of class `anal_obj`, returns a data frame with the object `id` and the (corrected) measures.
  - If applied to an object of class `anal_obj_ls`, returns a list of class `measures_ls`, with two objects: (i) `results`, a data frame containing the identification of each image (img) and object within each image (id); and (ii) `summary` a data frame containing the values for each image. If more than one object is detected in a given image, the number of objects (`n`), total area (`area_sum`), mean area (`area_mean`) and the standard deviation of the area (`area_sd`) will be computed. For the other measures (perimeter and radius), the mean values are presented.
- `plot_measures()` returns a `NULL` object, drawing the text according to the x and y coordinates of the objects in `object`.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
library(pliman)
img <- image_pliman("objects_300dpi.jpg")
plot(img)
# Image with four objects with a known resolution of 300 dpi
# Higher square: 10 x 10 cm
# Lower square: 5 x 5 cm
# Rectangle: 4 x 2 cm
# Circle: 3 cm in diameter

# Count the objects using the blue band to segment the image
```

```
results <-
   analyze_objects(img,
                index = "B",
                lower_noise = 0.1)
plot_measures(results, measure = "id")

# Get object measures by declaring the known resolution in dots per inch
(measures <- get_measures(results, dpi = 300))

# Calculated diagonal of the object 1
# 10 * sqrt(2) = 14.14

# Observed diagonal of the object 1
measures[1, "radius_max"] * 2


# Get object measures by declaring the known area of object 1
get_measures(results,
             id = 1,
             area ~ 100)
```

`utils_objects` *Utilities for working with image objects*

Description

- `object_id()` get the object identification in an image.
- `object_coord()` get the object coordinates and (optionally) draw a bounding rectangle around multiple objects in an image.
- `object_contour()` returns the coordinates (`x` and `y`) for the contours of each object in the image.
- `object_isolate()` isolates an object from an image.

Usage

```
object_coord(
  img,
  id = NULL,
  index = "NB",
  watershed = TRUE,
  invert = FALSE,
  filter = FALSE,
  fill_hull = FALSE,
  threshold = "Otsu",
  edge = 2,
```

```
  extension = NULL,
  tolerance = NULL,
  object_size = "medium",
  parallel = FALSE,
  workers = NULL,
  plot = TRUE
)

object_contour(
  img,
  pattern = NULL,
  dir_original = NULL,
  center = FALSE,
  index = "NB",
  invert = FALSE,
  filter = FALSE,
  fill_hull = FALSE,
  threshold = "Otsu",
  watershed = TRUE,
  extension = NULL,
  tolerance = NULL,
  object_size = "medium",
  parallel = FALSE,
  workers = NULL,
  plot = TRUE,
  verbose = TRUE
)

object_isolate(img, id = NULL, parallel = FALSE, workers = NULL, ...)

object_id(img, parallel = FALSE, workers = NULL, ...)
```

Arguments

`img`	An image of class `Image` or a list of `Image` objects.
`id`	• For `object_coord()`, a vector (or scalar) of object `id` to compute the bounding rectangle. Object ids can be obtained with `object_id()`. Set `id = "all"` to compute the coordinates for all objects in the image. If `id = NULL` (default) a bounding rectangle is drawn including all the objects. • For `object_isolate()`, a scalar that identifies the object to be extracted.
`index`	The index to produce a binary image used to compute bounding rectangle coordinates. See `image_binary()` for more details.
`watershed`	If `TRUE` (default) performs watershed-based object detection. This will detect objects even when they are touching one other. If `FALSE`, all pixels for each connected set of foreground pixels are set to a unique object. This is faster but is not able to segment touching objects.
`invert`	Inverts the binary image, if desired. Defaults to `FALSE`.

`filter` Performs median filtering in the binary image? See more at `image_filter()`. Defaults to `FALSE`. Use a positive integer to define the size of the median filtering. Larger values are effective at removing noise, but adversely affect edges.

`fill_hull` Fill holes in the objects? Defaults to `FALSE`.

`threshold` By default (`threshold = "Otsu"`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold. Inform any non-numeric value different than "Otsu" to iteratively chosen the threshold based on a raster plot showing pixel intensity of the index.

`edge` The number of pixels in the edge of the bounding rectangle. Defaults to `2`.

`extension, tolerance, object_size`
Controls the watershed segmentation of objects in the image. See `analyze_objects()` for more details.

`parallel` Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time when `image` is a list. The number of sections is set up to 50% of available cores.

`workers` A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.

`plot` Shows the image with bounding rectangles? Defaults to `TRUE`.

`pattern` A pattern of file name used to identify images to be imported. For example, if `pattern = "im"` all images in the current working directory that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be imported as a list. Providing any number as pattern (e.g., `pattern = "1"`) will select images that are named as 1.-, 2.-, and so on. An error will be returned if the pattern matches any file that is not supported (e.g., img1.pdf).

`dir_original` The directory containing the original images. Defaults to `NULL`, which means that the current working directory will be considered.

`center` If `TRUE` returns the object contours centered on the origin.

`verbose` If `TRUE` (default) a summary is shown in the console.

`...`
- For `object_isolate()`, further arguments passed on to `object_coord()`.
- For `object_id()`, further arguments passed on to `analyze_objects()`.

Value

- `object_id()` An image of class `"Image"` containing the object's identification.
- `object_coord()` A list with the coordinates for the bounding rectangles. If `id = "all"` or a numeric vector, a list with a vector of coordinates is returned.
- `object_isolate()` An image of class `"Image"` containing the isolated object.

Examples

```
library(pliman)
img <- image_pliman("la_leaves.jpg")
# Get the object's (leaves) identification
```

```
object_id(img)

# Get the coordinates and draw a bounding rectangle around leaves 1 and 3
object_coord(img, id = c(1, 3))

# Isolate leaf 3
isolated <- object_isolate(img, id = 3)
plot(isolated)
```

`utils_pca` *Utilities for Principal Component Axis analysis*

Description

- `pca()` Computes a Principal Component Analysis. It wrappers `stats::prcomp()`, but returns more results such as data, scores, contributions and quality of measurements for individuals and variables.
- `get_biplot()`: Produces a biplot for an object computed with `pca()`.
- `plot.pca()`: Produces several types of plots, depending on the `type` and `which` arguments.
  - `type = "var"` Produces a barplot with the contribution (`which = "contrib"`), qualitity of adjustment `which = "cos2"`, and a scatter plot with coordinates (`which = "coord"`) for the variables.
  - `type = "ind"` Produces a barplot with the contribution (`which = "contrib"`), qualitity of adjustment `which = "cos2"`, and a scatter plot with coordinates (`which = "coord"`) for the individuals.
  - `type = "biplot"` Produces a biplot.

Usage

```
pca(x, scale = TRUE)

get_biplot(
  x,
  axes = c(1, 2),
  show = c("both"),
  show_ind_id = TRUE,
  show_unit_circle = TRUE,
  expand = NULL
)

## S3 method for class 'pca'
plot(x, type = "var", which = "contrib", axis = 1, ...)
```

Arguments

x	• For pca(), a numeric or complex matrix (or data frame) which provides the data for the principal components analysis. • For plot.pca() and get_biplot(), an object computed with pca().
scale	A logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. Defaults to TRUE.
axes	The principal component axes to plot. Defaults to axes = c(1, 2), i.e., the first and second interaction principal component axis.
show	Which to show in the biplot. Defaults to "both" (both variables and individuals). One can also use "var", or "ind".
show_ind_id	Shows the labels for individuals? Defaults to TRUE.
show_unit_circle	Shows the unit variance circle? Defaults to TRUE.
expand	An expansion factor to apply when plotting the second set of points relative to the first. This can be used to tweak the scaling of the two sets to a physically comparable scale. Setting to TRUE will automatically compute the expansion factor. Alternatively, a numeric value can be informed.
type	One of "var" (to plot variables), "ind" (to plot individuals), or "biplot" to create a biplot.
which	Which measure to plot. Either which = "contribution" (default), which = "cos2" (quality of representation), or which = "coord" (coordinates)
axis	The axist to plot the contribution/cos2. Defaults to 1.
...	Further arguments passed on to get_biplot() when type = "biplot". Otherwise, When which = "coord", further arguments passed on to get_biplot(). When which = "contrib", or which = "cos2" further arguments passed on to graphics::barplot().

Value

- pca() returns a list including:
  - data: The raw data used to compute the PCA.
  - variances: Variances (eigenvalues), and proportion of explained variance for each component.
  - center,scale: the centering and scaling used.
  - ind,var A list with the following objects for individuals/variables, respectively.
  - coord: coordinates for the individuals/variables (loadings * the component standard deviations)
  - cos2: cos2 for the individuals/variables (coord^2)
  - contrib: The contribution (in percentage) of a variable to a given principal component: (cos2 * 100) / (total cos2 of the component)
- plot.pca() returns a list with the coordinates used.
- get_biplot() returns a NULL object

Examples

```
library(pliman)
pc <- pca(mtcars[1:10 ,1:6])
plot(pc)
plot(pc, type = "ind")
plot(pc, type = "var", which = "coord")
plot(pc, type = "ind", which = "coord")
plot(pc, type = "biplot")
```

`utils_pick` *Utilities for picking up points in an image*

Description

- `pick_count()` opens an interactive section where the user will be able to click in the image to count objects (points) manually. In each mouse click, a point is drawn and an upward counter is shown in the console. After `n` counts or after the user press Esc, the interactive process is terminated and the number of counts is returned.
- `pick_coord()` Picks coordinates from the image
- `pick_palette()` creates an image palette by picking up color point(s) from the image.
- `pick_rgb()` Picks up the RGB values from selected point(s) in the image.

Usage

```
pick_count(
  img,
  n = Inf,
  col = "red",
  viewer = get_pliman_viewer(),
  size = 0.8,
  plot = TRUE,
  verbose = TRUE
)

pick_coords(
  img,
  n = Inf,
  col = "red",
  viewer = get_pliman_viewer(),
  size = 0.8,
  verbose = TRUE
)

pick_rgb(
  img,
  n = Inf,
```

```
  col = "red",
  viewer = get_pliman_viewer(),
  size = 0.8,
  plot = TRUE,
  verbose = TRUE
)

pick_palette(
  img,
  n = Inf,
  r = 1,
  shape = "box",
  viewer = get_pliman_viewer(),
  show = "rgb",
  title = "Pick colors in the image",
  index = "B",
  random = TRUE,
  width = 100,
  height = 100,
  col = "red",
  size = 0.8,
  plot = TRUE,
  palette = TRUE,
  verbose = TRUE
)
```

Arguments

`img`	An `Image` object.
`n`	The number of points of the `pick_*` function. Defaults to `Inf`. This means that picking will run until the user press Esc.
`col, size`	The color and size for the marker point.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.
`plot`	Call a new `plot(img)` before processing? Defaults to `TRUE`.
`verbose`	If `TRUE` (default) shows a counter in the console.
`r`	The radius of neighborhood pixels. Defaults to `1`.
`shape`	A character vector indicating the shape of the brush around the selected pixel. It can be `"box"`, `"disc"`, `"diamond"`, `"Gaussian"` or `"line"`. Defaults to `"box"`. In this case, if `'r = 1'`, all the 8 surrounding pixels are sampled. Setting to

	"disc" and increasing the radius (r) will select surrounding pixels towards the format of a sphere around the selected pixel.
show	How to plot in mapview viewer, either 'rgb or 'index'.
title	The title of the map view when viewer is used.
index	The index to use for the index view. Defaults to 'B'.
random	Randomize the selected pixels? Defaults to TRUE.
width, height	The width and height of the generated palette. Defaults to 100 for both, i.e., a square image of 100 x 100.
palette	Plot the generated palette? Defaults to TRUE.

Value

- pick_count() returns data.frame with the x and y coordinates of the selected point(x).
- pick_rgb() returns a data.frame with the R, G, and B values of the selected point(s).
- pick_palette() returns an object of class Image.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
if(interactive()){
library(pliman)
img <- image_pliman("soybean_touch.jpg")

# start a counting process
pick_count(img)

# get rgb from point(s)
pick_rgb(img)

# create a palette from point(s)
pick_palette(img)
}
```

---

utils_polygon *Utilities for Polygons*

Description

Several useful functions for analyzing polygons. All of them are based on a set of coordinate points that describe the edge of the object(s). If a list of polygons is provided, it loops through the list and computes what is needed for each element of the list.

- Polygon measures

  - `conv_hull()` Computes the convex hull of a set of points.
  - `conv_hull_unified()` Computes the convex hull of a set of points. Compared to `conv_hull()`, `conv_hull_unified()` binds (unifies) the coordinates when x is a list of coordinates.
  - `poly_area()` Computes the area of a polygon given by the vertices in the vectors x and y using the Shoelace formula, as follows (Lee and Lim, 2017):

$$A = \frac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)\right|$$

    where `x` and `y` are the coordinates that form the corners of a polygon, and `n` is the number of coordinates.
  - `poly_angles()` Calculates the internal angles of the polygon using the law of cosines.
  - `poly_lw()` Returns the length and width of a polygon based on its alignment to the y-axis (with poly_align()). The length is defined as the range along the x-axis, and the width is defined as the range on the y-axis.
  - `poly_mass()` Computes the center of mass of a polygon given by the vertices in the vectors in `x`.
  - `poly_solidity()` Computes the solidity of a shape as the ratio of the shape area and the convex hull area.

- Perimeter measures
  - `poly_slide()` Slides the coordinates of a polygon given by the vertices in the vectors x and y so that the id-th point becomes the first one.
  - `poly_distpts()` Computes the Euclidean distance between every point of a polygon given by the vertices in the vectors x and y.
  - `poly_centdist()` Computes the Euclidean distance between every point on the perimeter and the centroid of the object.
  - `poly_perimeter()` Computes the perimeter of a polygon given by the vertices in the vectors x and y.
  - `poly_caliper()` Computes the caliper (also called the Feret's diameter) of a polygon given by the vertices in the vectors x and y.

- Circularity measures (Montero et al. 2009).
  - `poly_circularity()` computes the circularity (C), also called shape compactness or roundness measure, of an object. It is given by `C = P^2 / A`, where P is the perimeter and A is the area of the object.
  - `poly_circularity_norm()` computes the normalized circularity (Cn), which is unity for a circle. This measure is invariant under translation, rotation, scaling transformations, and is dimensionless. It is given by: `Cn = P^2 / 4*pi*A`.
  - `poly_circularity_haralick()` computes Haralick's circularity (CH). The method is based on computing all Euclidean distances from the object centroid to each boundary pixel. With this set of distances, the mean (`m`) and the standard deviation (`sd`) are computed. These statistical parameters are used to calculate the circularity, CH, of a shape as `CH = m/sd`.
  - `poly_convexity()` computes the convexity of a shape using the ratio between the perimeter of the convex hull and the perimeter of the polygon.
  - `poly_eccentricity()` computes the eccentricity of a shape using the ratio of the eigenvalues (inertia axes of coordinates).

  - `poly_elongation()` computes the elongation of a shape as `1 - width / length`.
- Utilities for polygons
  - `poly_check()` Checks a set of coordinate points and returns a matrix with x and y columns.
  - `poly_is_closed()` Returns a logical value indicating if a polygon is closed.
  - `poly_close()` and `poly_unclose()` close and unclose a polygon, respectively.
  - `poly_rotate()` Rotates the polygon coordinates by an angle (0-360 degrees) in the counterclockwise direction.
  - `poly_flip_x()`, `poly_flip_y()` flip shapes along the x-axis and y-axis, respectively.
  - `poly_align()` Aligns the coordinates along their longer axis using the var-cov matrix and eigen values.
  - `poly_center()` Centers the coordinates on the origin.
  - `poly_sample()` Samples n coordinates from existing points. Defaults to 50.
  - `poly_sample_prop()` Samples a proportion of coordinates from existing points. Defaults to 0.1.
  - `poly_spline()` Interpolates the polygon contour.
  - `poly_smooth()` Smooths the polygon contour using a simple moving average.
  - `poly_jitter()` Adds a small amount of noise to a set of point coordinates. See `base::jitter()` for more details.
- `poly_measures()` Is a wrapper around the `poly_*()` functions.

Usage

```
poly_check(x)

poly_is_closed(x)

poly_close(x)

poly_unclose(x)

poly_angles(x)

poly_limits(x)

conv_hull(x)

conv_hull_unified(x)

poly_area(x)

poly_slide(x, fp = 1)

poly_distpts(x)

poly_centdist(x)
```

```
poly_perimeter(x)

poly_rotate(x, angle, plot = TRUE)

poly_align(x, plot = TRUE)

poly_center(x, plot = TRUE)

poly_lw(x)

poly_eccentricity(x)

poly_convexity(x)

poly_caliper(x)

poly_elongation(x)

poly_solidity(x)

poly_flip_y(x)

poly_flip_x(x)

poly_sample(x, n = 50)

poly_sample_prop(x, prop = 0.1)

poly_jitter(x, noise_x = 1, noise_y = 1, plot = TRUE)

poly_circularity(x)

poly_circularity_norm(x)

poly_circularity_haralick(x)

poly_mass(x)

poly_spline(x, vertices = 100, k = 2)

poly_smooth(x, niter = 10, n = NULL, prop = NULL, plot = TRUE)

poly_measures(x)
```

Arguments

x	A 2-column matrix with the x and y coordinates. If x is a list of vector coordinates, the function will be applied to each element using base::lapply() or

	base::sapply().
fp	The ID of the point that will become the new first point. Defaults to 1.
angle	The angle (0-360) to rotate the object.
plot	Should the object be plotted? Defaults to TRUE.
n, prop	The number and proportion of coordinates to sample from the perimeter coordinates. In poly_smooth(), these arguments can be used to sample points from the object's perimeter before smoothing.
noise_x, noise_y	A numeric factor to define the noise added to the x and y axes, respectively. See base::jitter() for more details.
vertices	The number of spline vertices to create.
k	The number of points to wrap around the ends to obtain a smooth periodic spline.
niter	An integer indicating the number of smoothing iterations.

Value

- conv_hull() and poly_spline() returns a matrix with x and y coordinates for the convex hull/smooth line in clockwise order. If x is a list, a list of points is returned.
- poly_area() returns a double, or a numeric vector if x is a list of vector points.
- poly_mass() returns a data.frame containing the coordinates for the center of mass, as well as for the maximum and minimum distance from contour to the center of mass.
- poly_slides(), poly_distpts(), poly_spline(), poly_close(), poly_unclose(), poly_rotate(), poly_jitter(), poly_sample(), poly_sample_prop(), and poly_measures returns a data.frame.
- poly_perimeter(), poly_lw(), poly_eccentricity(), poly_convexity(), poly_caliper(), poly_elongation(), poly_circularity_norm(), poly_circularity_haralick() returns a double.

References

Lee, Y., & Lim, W. (2017). Shoelace Formula: Connecting the Area of a Polygon and the Vector Cross Product. The Mathematics Teacher, 110(8), 631–636. doi:10.5951/mathteacher.110.8.0631

Montero, R. S., Bribiesca, E., Santiago, R., & Bribiesca, E. (2009). State of the Art of Compactness and Circularity Measures. International Mathematical Forum, 4(27), 1305–1335.

Chen, C.H., and P.S.P. Wang. 2005. Handbook of Pattern Recognition and Computer Vision. 3rd ed. World Scientific.

Examples

```
library(pliman)
# A 2 x 2 square
df <- draw_square(side = 2)

# square area
poly_area(df)
```

```
# polygon perimeter
poly_perimeter(df)

# center of mass of the square
cm <- poly_mass(df)
plot_mass(cm)

# The convex hull will be the vertices of the square
(conv_square <- conv_hull(df) |> poly_close())
plot_contour(conv_square,
             col = "blue",
             lwd = 6)
poly_area(conv_square)


################### Example with a polygon ##################
x <- c(0, 1,   2, 3,  5, 2, -1, 0, 0)
y <- c(5, 6.5, 7, 3,  1, 1,  0, 2, 5)
df_poly <- cbind(x, y)

# area of the polygon
plot_polygon(df_poly, fill = "red")
poly_area(df_poly)

# perimeter of the polygon
poly_perimeter(df_poly)

# center of mass of polygon
cm <- poly_mass(df_poly)
plot_mass(cm,  col = "blue")

# vertices of the convex hull
(conv_poly <- conv_hull(df_poly))

# area of the convex hull
poly_area(conv_poly)

plot_polygon(conv_poly,
             fill = "red",
             alpha = 0.2,
             add = TRUE)


############ example of circularity measures ################
tri <- draw_circle(n = 200, plot = FALSE)
plot_polygon(tri, aspect_ratio = 1)
poly_circularity_norm(tri)

set.seed(1)
tri2 <-
  draw_circle(n = 200, plot = FALSE) |>
  poly_jitter(noise_x = 100, noise_y = 100, plot = FALSE)
```

```
plot_polygon(tri2, aspect_ratio = 1)
poly_circularity_norm(tri2)
```

`utils_polygon_plot` *Utilities for plotting polygons*

Description

- `plot_contour()` Plot contour lines.
- `plot_polygon()` Plots a polygon describing the objects.
- `plot_mass()` Plots the center of mass along with maximum and minimum radius.
- `plot_ellipse()` Plots an ellipse that fits the major and minor axis for each object.

Usage

```
plot_contour(x, id = NULL, col = "black", lwd = 1, ...)

plot_polygon(
  x,
  fill = "gray",
  random_fill = TRUE,
  points = FALSE,
  merge = TRUE,
  border = "black",
  alpha = 1,
  add = FALSE,
  nrow = NULL,
  ncol = NULL,
  aspect_ratio = 1,
  show_id = TRUE,
  xlim = NULL,
  ylim = NULL,
  ...
)

plot_mass(x, id = NULL, col = "black", cex = 1, lwd = 1)

plot_ellipse(object, id = NULL, col = "black", lwd = 1)
```

Arguments

`x`	A 2-column matrix with the `x` and `y` coordinates.
`id`	The object identification (numeric) to plot the contour/ellipse. By default (`id = NULL`), the contour is plotted to all objects.
`col, lwd, cex`	The color, width of the lines, and size of point, respectively.

`...`
- For `plot_contour()` and `plot_ellipse()` further arguments passed on to `graphics::lines()`.
- For `plot_mass()`, further arguments passed on to `graphics::points()`.
- For `plot_polygon()`, further arguments passed on to `graphics::polygon()`.

`fill, border, alpha` The color to fill the polygon, the color of the polygon's border, and the alpha transparency (1 opaque, 0 transparent).

`random_fill` Fill multiple objects with random colors? Defaults to `TRUE`.

`points` Plot the points? Defaults to `FALSE`.

`merge` Merge multiple objects into a single plot? Defaults to `TRUE`. If `FALSE`, a single call `plot()` will be used for each objects. Use `nrow` and `ncol` to control the number of rows and columns of the window.

`add` Add the current plot to a previous one? Defaults to `FALSE`.

`nrow, ncol` The number of rows and columns to use in the composite image. Defaults to `NULL`, i.e., a square grid is produced.

`aspect_ratio` The x/y aspect ratio. Defaults to `1`. This will set up the window so that one data unit in the y direction is equal to one data unit in the x direction. Set `aspect_ratio = NULL` to fit the object to the window size.

`show_id` Shows the object id? Defaults to `TRUE`.

`xlim, ylim` A numeric vector of length 2 (min; max) indicating the range of `x` and `y`-axes.

`object` An object computed with `analyze_objects()`.

Value

a `NULL` object.

Examples

```
plot_polygon(contours)
plot_contour(contours[[1]], id = 6, col = "red", lwd = 3)
```

`utils_rows_cols` *Utilities for handling with rows and columns*

Description

- `columns_to_rownames()`: Move a column of `.data` to its row names.
- `rownames_to_column()`: Move the row names of `.data` to a new column.
- `remove_rownames()`: Remove the row names of `.data`.
- `round_cols()` Rounds the values of all numeric variables to the specified number of decimal places (default 2).

Usage

```
column_to_rownames(.data, var = "rowname")

rownames_to_column(.data, var = "rowname")

remove_rownames(.data)

round_cols(.data, digits = 2)
```

Arguments

.data	A data frame
var	Name of column to use for rownames.
digits	The number of significant figures. Defaults to 2.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
library(pliman)
iris2 <- iris |> rownames_to_column()
head(iris2)
iris2$rowname <- paste0("r", iris2$rowname)
iris2 |> column_to_rownames("rowname") |> head()
```

---

utils_shapes *Utilities for drawing coordinates of known shapes*

Description

The functions computes the coordinates of common shapes such as squares triangles, rectangles and circles.

- `draw_circle()` Draws a perfect circle with a desired radius.
- `draw_square()` Draws a square with a desired side.
- `draw_rectangle()` Draws a rectangle given two desired sides.
- `draw_trian_equi()` Draws an equilateral triangle with a desired side.
- `draw_trian_rect()` Draws a triangle rectangle given two cathetus.
- `draw_n_tagon()` Draws polygons with `n` sides

Usage

```
draw_circle(radius = 1, n = 1000, plot = TRUE)

draw_square(side = 2, plot = TRUE)

draw_rectangle(side1 = 2, side2 = 3, plot = TRUE)

draw_trian_equi(side = 2, plot = TRUE)

draw_trian_rect(cat1 = 1, cat2 = 2, plot = TRUE)

draw_n_tagon(n, plot = TRUE)
```

Arguments

radius	The radius of the circle. Defaults to 1.
n	The number of sides in the n-tagon.
plot	Plots the result? Defaults to TRUE.
side	The side of the square/equilateral triangle. Defaults to 2.
side1, side2	The first and second sides of the rectangle. Defaults to 2 and 3, respectively.
cat1, cat2	The first and second cathetus of the right triangle. Defaults to 1, and 2, respectively.

Value

A data frame with the x and y coordinates

Examples

```
########## An example of a circle ##########
library(pliman)
radius <- 3
circ <- draw_circle(radius = radius)

# area
pi * radius ^ 2
poly_area(circ)

# perimeter
2 * pi * radius
poly_perimeter(circ)

############ An example of a square ############
side <- 2
(square <- draw_square(side = side))

# area
side ^ 2
poly_area(square)
```

```
# perimeter
side * 4
poly_perimeter(square)

############ An example of a rectangle ############
side1 <- 2
side2 <- 3
(rect <- draw_rectangle())

# area
poly_area(rect)

# perimeter
poly_perimeter(rect)
############  An example of an equilateral triangle #########
side <- 1 # defaults
(trig <- draw_trian_equi(side = side))

### area (b*h / 2)
# height of the triangle
(h <- (side * sqrt(3)) / 2)
side * h / 2

poly_area(trig)

### perimeter (side * 3)
poly_perimeter(trig)

############ An example of a rectangle triangle ##########
cat1 <- 2
cat2 <- 3
(df <- draw_trian_rect(cat1, cat2))
# area
(cat1 * cat2) / 2
poly_area(df)

# perimeter
cat1 + cat2 + sqrt(cat1^2 + cat2^2)
poly_perimeter(df)
############ An creating shapes with n sides ############
side <- 2
(square <- draw_square(side = side))

# area
side ^ 2
poly_area(square)

# perimeter
side * 4
poly_perimeter(square)
```

utils_stats — *These functions applies common statistics to a list of objects, returning a numeric vector.*

Description

These functions applies common statistics to a list of objects, returning a numeric vector.

Usage

```
mean_list(x, ...)

sd_list(x, ...)

max_list(x, ...)

min_list(x, ...)
```

Arguments

x	A data.frame or matrix with numeric values.
...	Further arguments passed on to the R base function (e.g, mean(), sd(), etc.)

Value

A numeric vector.

Examples

```
mean_list(list(a = 1:10, b = 2:20))
```

utils_transform — *Spatial transformations*

Description

Performs image rotation and reflection

- `image autocrop()` Crops automatically an image to the area of objects.
- `image_crop()` Crops an image to the desired area.
- `image_trim()` Remove pixels from the edges of an image (20 by default).
- `image_dimension()` Gives the dimension (width and height) of an image.
- `image_rotate()` Rotates the image clockwise by the given angle.
- `image_horizontal()` Converts (if needed) an image to a horizontal image.

- `image_vertical()` Converts (if needed) an image to a vertical image.
- `image_hreflect()` Performs horizontal reflection of the `image`.
- `image_vreflect()` Performs vertical reflection of the `image`.
- `image_resize()` Resize the `image`. See more at `EBImage::resize()`.
- `image_contrast()` Improve contrast locally by performing adaptive histogram equalization. See more at `EBImage::clahe()`.
- `image_dilate()` Performs image dilatation. See more at `EBImage::dilate()`.
- `image_erode()` Performs image erosion. See more at `EBImage::erode()`.
- `image_opening()` Performs an erosion followed by a dilation. See more at `EBImage::opening()`.
- `image_closing()` Performs a dilation followed by an erosion. See more at `EBImage::closing()`.
- `image_filter()` Performs median filtering in constant time. See more at `EBImage::medianFilter()`.
- `image_blur()` Performs blurring filter of images. See more at `EBImage::gblur()`.
- `image_skeleton()` Performs image skeletonization.

Usage

```
image_autocrop(
  img,
  index = "NB",
  edge = 5,
  filter = 3,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_crop(
  img,
  width = NULL,
  height = NULL,
  viewer = get_pliman_viewer(),
  show = "rgb",
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_dimension(img, parallel = FALSE, workers = NULL, verbose = TRUE)

image_rotate(
  img,
  angle,
  bg_col = "white",
```

```
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = TRUE
)

image_horizontal(
  img,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_vertical(
  img,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_hreflect(
  img,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_vreflect(
  img,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_resize(
  img,
  rel_size = 100,
  width,
  height,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)
```

```
image_trim(
  img,
  edge = NULL,
  top = NULL,
  bottom = NULL,
  left = NULL,
  right = NULL,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_dilate(
  img,
  kern = NULL,
  size = NULL,
  shape = "disc",
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_erode(
  img,
  kern = NULL,
  size = NULL,
  shape = "disc",
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_opening(
  img,
  kern = NULL,
  size = NULL,
  shape = "disc",
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_closing(
```

```
  img,
  kern = NULL,
  size = NULL,
  shape = "disc",
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_skeleton(
  img,
  kern = NULL,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE,
  ...
)

image_thinning(
  img,
  niter = 3,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE,
  ...
)

image_filter(
  img,
  size = 2,
  cache = 512,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_blur(
  img,
  sigma = 3,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)
```

```
image_contrast(
  img,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)
```

Arguments

img	An image or a list of images of class Image.
index	The index to segment the image. See image_index() for more details. Defaults to "NB" (normalized blue).
edge	• for image_autocrop() the number of pixels in the edge of the cropped image. If edge = 0 the image will be cropped to create a bounding rectangle (x and y coordinates) around the image objects. • for image_trim(), the number of pixels removed from the edges. By default, 20 pixels are removed from all the edges.
filter	Performs median filtering in the binary image. This is useful to remove noise (like dust) and improve the image autocropping method. See more at image_filter(). Set to FALSE to remove the median filtering.
parallel	Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time when image is a list. The number of sections is set up to 70% of available cores.
workers	A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.
verbose	If TRUE (default) a summary is shown in the console.
plot	If TRUE plots the modified image. Defaults to FALSE.
width, height	• For image_resize() the Width and height of the resized image. These arguments can be missing. In this case, the image is resized according to the relative size informed in rel_size. • For image_crop() a numeric vector indicating the pixel range (x and y, respectively) that will be maintained in the cropped image, e.g., width = 100:200
viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
show	How to plot in mapview viewer, either "rgb" or "index".
angle	The rotation angle in degrees.

bg_col — Color used to fill the background pixels, defaults to "white".

rel_size — The relative size of the resized image. Defaults to 100. For example, setting rel_size = 50 to an image of width 1280 x 720, the new image will have a size of 640 x 360.

top, bottom, left, right — The number of pixels removed from top, bottom, left, and right when using image_trim().

kern — An Image object or an array, containing the structuring element. Defaults to a brushe generated with EBImage::makeBrush().

size
- For image_filter() is the median filter radius (integer). Defaults to 3.
- For image_dilate() and image_erode() is an odd number containing the size of the brush in pixels. Even numbers are rounded to the next odd one. The default depends on the image resolution and is computed as the image resolution (megapixels) times 20.

shape — A character vector indicating the shape of the brush. Can be box, disc, diamond, Gaussian or line. Default is disc.

... — Additional arguments passed on to image_binary().

niter — The number of iterations to perform in the thinning procedure. Defaults to 3. Set to NULL to iterate until the binary image is no longer changing.

cache — The the L2 cache size of the system CPU in kB (integer). Defaults to 512.

sigma — A numeric denoting the standard deviation of the Gaussian filter used for blurring. Defaults to 3.

Value

- image_skeleton() returns a binary Image object.
- All other functions returns a modified version of image depending on the image_*() function used.
- If image is a list, a list of the same length will be returned.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
library(pliman)
img <- image_pliman("sev_leaf.jpg")
plot(img)
img <- image_resize(img, 50)
img1 <- image_rotate(img, 45)
img2 <- image_hreflect(img)
img3 <- image_vreflect(img)
img4 <- image_vertical(img)
image_combine(img1, img2, img3, img4)
```

utils_wd *Set and get the Working Directory quicky*

Description

- `get_wd_here()` gets the working directory to the path of the current script.
- `set_wd_here()` sets the working directory to the path of the current script.
- `open_wd_here()` Open the File Explorer at the directory path of the current script.
- `open_wd()` Open the File Explorer at the current working directory.

Usage

```
set_wd_here(path = NULL)

get_wd_here(path = NULL)

open_wd_here(path = get_wd_here())

open_wd(path = getwd())
```

Arguments

path	Path components below the project root. Defaults to NULL. This means that the directory will be set to the path of the file. If the path doesn't exist, the user will be asked if he wants to create such a folder.

Value

- `get_wd_here()` returns a full-path directory name.
- `get_wd_here()` returns a message showing the current working directory.
- `open_wd_here()` Opens the File Explorer of the path returned by `get_wd_here()`.

Examples

```
## Not run:
get_wd_here()
set_wd_here()
open_wd_here()

## End(Not run)
```

watershed2 *Alternative watershed algorithm*

Description

This is a basic watershed algorithm that can be used as a faster alternative to EBImage::watershed(). I strongly suggest using this only with round objects, since it doesn't consider both 'extension' and 'tolerance' arguments of EBImage::watershed().

Usage

```
watershed2(binary, dist_thresh = 0.75, plot = TRUE)
```

Arguments

binary	A binary image
dist_thresh	The distance threshold to create the
plot	If TRUE (default) plots the labeled objects

Value

The labelled version of binary.

Examples

```
library(pliman)
img <- image_pliman("soybean_touch.jpg")
binary <- image_binary(img, "B")[[1]]
wts <- watershed2(binary)
range(wts)
```

Index